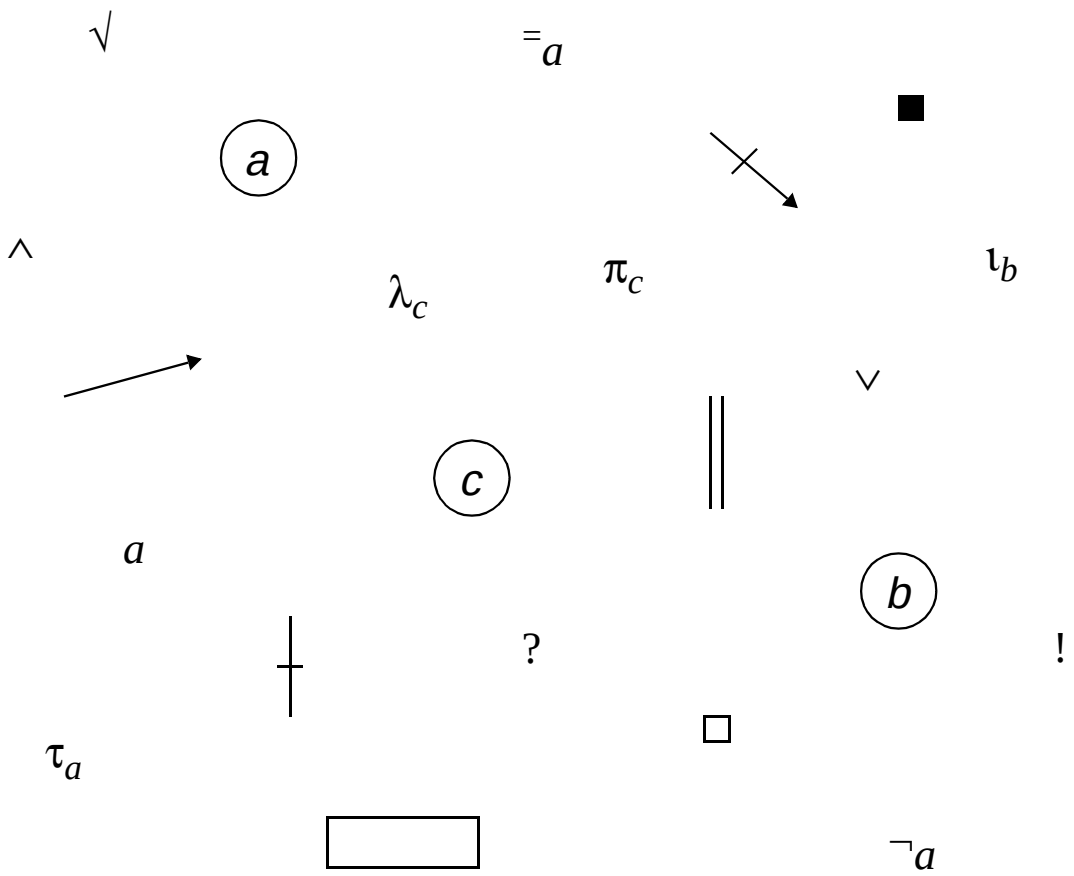
Dick Quartel

Action relations

Basic design concepts for behaviour modelling and refinement



ACTION RELATIONS

BASIC DESIGN CONCEPTS FOR BEHAVIOUR MODELLING AND REFINEMENT

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. F.A. van Vught,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op donderdag 19 februari 1998 te 15.00 uur.

door

Dirk Anton Cornelis Quartel

geboren op 29 augustus 1967
te Hellevoetsluis

Dit proefschrift is goedgekeurd door:

promotor	prof. dr. ir. C.A. Vissers
assistent-promotor	dr. Luís Ferreira Pires

Preface

In this thesis we develop basic conceptual means for the design of distributed information systems to facilitate designers in building system designs, in analysing and structuring relevant system properties, and in correctly refining abstract system designs into more detailed ones that brings these designs closer to their actual implementation.

Distributed information systems perform information processing functions by means of the cooperation of multiple geographically distributed systems, interconnected via communication networks. The conceptual means developed in this thesis are originally derived from the design needs of these systems, however, they are equally applicable to the modelling and redesign of business processes.

The main challenge in distributed systems design is to cope with their large complexity, which is expected to only increase in the future. Evidence for this expectation are the following observations:

- society becomes vitally dependent of distributed systems, requiring systems to become robust, i.e., resistant to irregular internal and external behaviour. Examples of systems that have become part of our daily life are banking systems, information retrieval systems, communication systems, traffic-guiding systems and logistics systems;
- technological progress allows the development of advanced systems, supporting demands for improved functionality, reliability and performance. Examples of such systems are car navigation systems, tele-conferencing systems, tele-surgery systems and service-oriented business organisations;
- the availability of advanced systems accelerates the demand for and development of more advanced systems. Business organisations may be redesigned before a previous redesign has been fully implemented, e.g., due to the availability of improved production systems, and advanced telematic systems or information systems.

An important concern to this thesis is to keep control of the design process of complex distributed systems. Our approach to deal with this is based on the step-wise development of distributed systems at subsequent abstraction levels. This approach requires the conceptual means to make proper system abstractions, usually denoted as models or designs. Distributed systems are simply too complex to be dealt with in full detail in a single step. Proper system abstractions though can provide insight in the system characteristics and their relationships, by allowing one to concentrate on characteristics that are considered only relevant at specific abstraction levels. Nonetheless, in order to obtain system implementations, all system details have to be considered throughout the development process. This requires conceptual means that support the structuring of system characteristics at a certain abstraction level and refinement of (parts of) these characteristics into more detailed ones at a

lower abstraction level. Furthermore, these conceptual means should enable one to determine whether the system details added at lower abstraction levels conform to the system characteristics defined at higher abstraction levels.

We distinguish the following conceptual means for the design of distributed systems:

- *design concepts*, which are abstractions of common essential characteristics of distributed systems;
- *structuring techniques*, which are used to compose a design defining the required characteristics of a distributed system from (compositions of) basic design concepts;
- *refinement operations*, which are used to transform a design into a more detailed design, while adding characteristics.

Design concepts constitute the building blocks of designs. The set of defined design concepts including their composition rules constitute a design model. This thesis is concerned with the definition, structuring and manipulation of the most elementary building blocks, which are called basic design concepts. Basic design concepts determine the expressive power of a design model, since any design concept in a consistent design model should be either a basic design concept or a composition of basic design concepts.

Many research activities dealing with the systematic support for the modelling and design of distributed systems are carried out with varying degrees of success. This support comprises, amongst others, design methods, conceptual frameworks, formal description techniques and automated tools. The combination of concepts, languages, tools, methods, skills and preferences that determine the productivity of a design process is called a design culture in [73]. In order to remain competitive, industrial companies should continuously be alert of, develop and improve their design culture. The underlying design concepts of many design cultures, unfortunately, are not always clear or precisely defined, severely hampering the effectiveness of such a design culture.

We believe the success of a design culture is largely determined by the choice, correct understanding and precise definition of its basic design concepts, since they determine a designer's capability to conceive, structure and refine the essential characteristics of distributed systems. Therefore, we consider the development of formal description techniques subordinate to the development of (basic) design models. The purpose of formal description techniques is to enable the precise and unambiguous documentation, communication and analysis of system designs. Furthermore, they should represent (basic) design concepts in a direct and intuitive way, allowing designers to conceive and model designs straightforwardly and comprehensibly without being forced to use specification tricks.

This thesis is based on the work of [16] which represents the status of the design methodology of our research group, i.e., the Architecture group of the Computer Science department at the University of Twente, in 1994. The main achievements of this work comprise the definition of:

- a set of basic design concepts, including the action concept and the concept of causality relation;

- two basic behaviour structuring techniques: causality-oriented and constraint-oriented structuring;
- basic design operations for behaviour refinement.

These results constitute an expressive set of basic conceptual means for the design of distributed systems. Certain aspects of these conceptual means, however, need further elaboration: basic design concepts are only defined informally, quantitative aspects of causality relations are partially considered (in particular probability), repetitive behaviours are not modelled in a satisfactory way, and behaviour refinement is restricted to specific behaviour patterns.

Therefore, this thesis aims at:

- the further elaboration of the causality relation concept, including the quantitative properties of time and probability;
- the formal definition of the identified basic design concepts;
- the development of techniques for modelling repetitive behaviours;
- the complete definition of the behaviour refinement design operation, supporting the refinement of arbitrary behaviours.

The approach followed in this thesis is characterized by:

- the use of an abstraction hierarchy to structure the qualitative and quantitative characteristics of action relations. This enables a step-wise and modular approach towards the elaboration of the causality relation concept, which is used as the basic building block to model action relations;
- the use of a so called execution model to define a compositional and formal semantics for behaviours defined in terms of causality relations. In this model, a behaviour is defined by explicitly enumerating all possible executions of this behaviour.

This thesis is structured as follows:

- Chapter 1 - *Introduction*: gives a general problem description in terms of the background, motivation, application domain, objectives and structure of this thesis.
- Chapter 2 - *Aspects of the design methodology*: discusses some aspects of our methodology that are relevant for a proper understanding of this thesis. This discussion comprises our basic design model, our systems design approach, and our approach towards the representation and formal definition of design concepts.
- Chapter 3 - *Execution model*: presents the formal model we use to define the formal semantics of behaviours defined in terms of causality relations. This model is also used to give insight in the number, complexity and variety of relations that can be defined between actions.
- Chapter 4 - *Action relations*: presents an initial definition of the causality relation concept, which supports the modelling of the basic characteristics of action relations, including temporal ordering and the uncertainty of action occurrences. In order to model uncertainty, the uncertainty attribute is introduced. This chapter only considers

behaviours of two actions to perform an elaborate analysis of the various types of temporal ordering relations that can be defined between two actions.

- Chapter 5 - *Monolithic behaviours*: gives a complete definition of the *and*- and the *or*-operator on causality conditions introduced in Chapter 4, to support the modelling of temporal ordering relations between multiple actions. This chapter also defines rules to combine causality relations of multiple actions into consistent behaviour models.
- Chapter 6 - *Information, time and location attributes*: extends the causality relation concept with the information, time and location attribute. This allows one to relate information, time and location values established in different action occurrences.
- Chapter 7 - *Probability attributes*: extends the causality relation concept with the integral probability attribute and the stochastic probability attribute. These attributes refine the uncertainty attribute, allowing one to quantify the uncertainty of action occurrences.
- Chapter 8 - *Behaviour refinement*: defines two basic design operations for behaviour refinement: causality refinement and action refinement. This chapter defines rules to determine whether a concrete behaviour obtained by refining an abstract behaviour conforms to this abstract behaviour.
- Chapter 9 - *Causality-oriented behaviour composition*: introduces the causality-oriented structuring technique and extends this technique to enable the modelling of (infinitely) repetitive behaviours.
- Chapter 10 - *Case study: OSI Connection-oriented Transport Service*: applies our basic design model to the modelling of the OSI Connection-oriented Transport Service, including the modelling of quality of service characteristics.
- Chapter 11 - *Conclusions*: summarizes the main results of our research and indicates some directions for further work.

Acknowledgements

A number of people have contributed to this thesis. I like to thank some of them below.

First of all, I am indebted to my supervisor, professor Chris Vissers, who created the opportunity to perform this research. The discussions he led were an important source of inspiration. I learned to appreciate his never-ending quest for better understanding, precisely defining and consistently applying the architectural principles that underlie systems design. In this way, he also laid the foundation for this thesis. Furthermore, the convincing way in which he stressed the importance of these principles over and over again, compelled me to understand the importance of my work.

The help of Luís Ferreira Pires was indispensable in the detailed elaboration of this thesis. Many times I rushed into his room to discuss yet another ‘serious’ technical problem I had encountered. His ability to quickly understand and structure most of these problems, made it a lot easier for me to tackle them properly. I also thank him for his patience and confidence in periods of stress.

My gratitude goes to the other members of the Architecture group: Mark de Weger, Marten van Sinderen, Ciro de Barros Barbosa and, recently, Cléver Guareis de Farias, for their support and companionship. I enjoyed the many interesting discussions with my (room)mate Mark. Marten set up the case study on behaviour refinement.

Further I like to thank the members of the Tele-Informatics and Open Systems (TIOS') group, especially Aiko Pras, Wilma Hiddink and Lex Heerink, for creating a pleasant working environment. From the Telematics Research Centre I want to acknowledge Henry Franken, Wil Janssen, Henk Jonkers and Wouter Teeuw for their contribution to the part of this research that has been carried out in the Testbed project.

Finally, I want to thank my wife Petra for her support and encouragement, and Rutger, Hebe and Jonathan for condoning my ‘absence’ and for remembering my name.

Dick Quartel
January 15, Hengelo

Table of Contents

Chapter 1

Introduction

This chapter introduces the background of this thesis, defines its objectives, discusses the approach we follow to achieve these objectives, and motivates the relevance of this work. This thesis develops some elements of a design methodology for distributed systems. This chapter argues that the development of a design methodology should be based on properly chosen and precisely defined basic design concepts. The careful choice, elaboration and precise definition of a limited set of basic design concepts is an important aim of this thesis.

The structure of this chapter is as follows. Section 1.1 gives the rationale for our work. Section 1.2 discusses the role of our results in design methodologies for distributed systems. Section 1.3 presents the objectives of this thesis. And Section 1.4 discusses our approach.

1.1 Rationale

In general, a distributed system is too complex to be designed in full detail in a single design step. Instead, multiple designs of the system have to be made at distinct, but related abstraction levels. A design defines the characteristics of the system that are considered essential at a specific abstraction level, and the relationships between these characteristics. A design abstracts therefore from other characteristics, which are considered irrelevant at this abstraction level. As an example, we consider the architecture of buildings, in particular the architecture of an office-block. At a high abstraction level we may consider the number of floors, offices, conference-rooms and canteens that can be built as the essential characteristics of an office-block, whereas at a lower abstraction level we may be concerned with the location and interconnection of offices, conference rooms and canteens.

In the conception of the essential characteristics of a system, designers use mental images of these characteristics, which we call *design concepts*. A design is a composition of these concepts, allowing designers to conceive the whole system. For example, in the conception of an office-block we use the concepts of floor, office, conference-room and canteen. These concepts are mental images (abstractions) of real floors, offices, conference-rooms and canteens, respectively.

The specific composition of concepts in a design defines the *structure* of a design. This structure generally depends on the specific design objectives. For example, an office-block may be structured such that each floor has its own conference-room and canteen, or, alternatively, such that all conference-rooms and canteens are concentrated on a single floor.

An accurate design of the real system can be obtained by subsequently adding more detailed characteristics to the design, while preserving the previously defined characteristics. This process is called step-wise *refinement*, which renders multiple designs at subsequent abstraction levels. For example, at some abstraction level, the design of a floor may be defined as a structure of multiple offices of various sizes and shapes, which are interconnected by a corridor. At a lower abstraction level, the design of some office may be refined by defining the location of light switches and wall-outlets for electricity, telephone and network. This more detailed design should comply to the size and shape of the office and, for instance, to the location of its door(s), which are defined at a higher abstraction level.

Based on the above, we distinguish the following conceptual means to facilitate the elaboration of designs of a system at different abstraction levels: basic design concepts, structuring techniques and refinement operations.

1.1.1 Basic design concepts

Basic design concepts model common, elementary and essential characteristics of distributed systems, abstracting from characteristics that are irrelevant to the fundamental purpose of the system. In this way, they constitute the basic (elementary) building blocks for the construction of system designs. For example, the concepts of door, wall and window are basic concepts for an architect of buildings, since they are part of any real building. Instead, an architect abstracts from concepts like furniture, wallpaper or carpet, since they are irrelevant to the most fundamental purpose of the building.

Basic design concepts can be composed into a *composite design concept*, which represents some frequently encountered composition of system characteristics, e.g., a common system part or a common set of system requirements (or properties). The use of composite design concepts should speed up the design process. For example, an architect may use the concept of room to design an apartment consisting of an entrance interconnecting a living room, a kitchen, two bedrooms and a bathroom. The concept of room is composed, amongst others, of the basic concepts of wall, window and door.

The collection of basic design concepts, including rules for their composition, is called a *basic design model*. A basic design model extended with composite design concepts is called a *design model*. The expressive power of a design model is determined by its basic design concepts, since any design concept in a consistent design model should be either a basic design concept or a composition of basic design concepts. Therefore, basic design concepts should be selected carefully ([76, 77, 57, 40]).

An important quality criterion for the selection of basic design concepts is *generality*. Two types of generality are distinguished:

- *generality within the application domain*, which means that basic design concepts should be applicable to all systems in the application domain. The use of a generic (and limited) set of basic design concepts adds to the comprehensibility and consistent use of a (basic) design model;
- *generality within the design process*, which means that basic design concepts should

be applicable throughout a large part of the design process. This minimizes the number of necessary design models, including the associated mapping and translation functions that have to be used along the design trajectory. A design model that supports this type of generality is called a *broad-spectrum* design model in [17].

Other quality criteria for the selection of basic design concepts can be derived from general design quality criteria as described, e.g., in [76, 66, 3, 64]. For example, *orthogonality* and *completeness* define that basic concepts should model independent and complementary characteristics, respectively, such that the design model is capable of modelling all relevant system characteristics. Another important quality criterion is *parsimony*, which advocates that a minimal set of basic concepts has to be defined. In our case, parsimony is supported by the quality criteria of orthogonality and both types of generality discussed above. Despite these general quality criteria, however, the selection of basic design concepts is an engineering practice, which is also influenced by specific design objectives.

Example: the action concept

The action concept is identified in [16, 57, 67, 66] as a basic design concept to model the essential characteristics of activities. This concept is used in this work as a basic design concept, since the behaviour of any distributed system consists of multiple related activities. Furthermore, the action concept can be used at multiple abstraction levels, since an activity that is modelled by an action at a certain abstraction level can be modelled as multiple related sub-activities at a lower abstraction level.

When considering the timing characteristics of activities, the following alternative definitions of an action that models an activity can be considered:

1. an action models the execution period of the activity, which is characterized by the time moment at which the activity starts and the time moment at which the activity terminates, making its result available. This corresponds, e.g., to the definition of a structured event in [55] and to the definition of an operation execution in [42]; or
2. an action (only) models the time moment at which the activity terminates, making its result available. This corresponds to the definition in [16, 57, 67, 66].

Figure 1.1 depicts an example behaviour *B*, which defines the sequential composition of two actions *r* and *e*. These actions model the receipt of a data unit and the calculation of an error detection function on this data unit, respectively. We assume that the ordering relation between actions *r* and *e* is defined as follows: the modelled timing characteristics of action *r* must all precede the modelled timing characteristics of action *e*. This implies that (i) in case of the first definition, the execution of *e* can only start after the execution of *r* has finished, which corresponds, e.g., to the definition of the ordering relation in [55] and to the definition of the strong precedence relation in [42], and (ii) in case of the second definition, the execution of *e* can only finish after the execution of *r* has finished, which corresponds to the definition of the enabling relation in [16, 57, 67, 66].

Figure 1.1 also depicts two alternative refinements (implementations) *B1* and *B2* of behaviour *B*. Action *r* is refined into the sequential composition of four actions r_1, r_2, r_3 and r_4,

which model the receipt of four successive data segments that together constitute the receipt of the data unit modelled by action *r*. Correspondingly, action *e* is refined into the sequential composition of four actions e_1, e_2, e_3 and e_4 that model the decomposition of the error detection function into the calculation of this function on the four data segments received in r_1, r_2, r_3 and r_4, respectively. Refinements *B1* and *B2* model the semi-concurrent and sequential execution of the refinements of actions *r* and *e*, respectively.

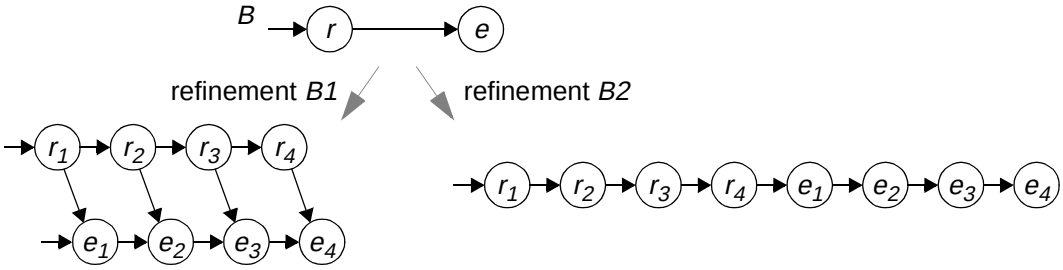


Figure 1.1: Generality of the action concept

The first definition of the action concept only allows refinement *B2*, since action *r* must terminate before action *e* can start. The second definition allows both refinements *B1* and *B2*, since it only prescribes that action *r* must be finished before action *e* can finish. We conclude that the second definition is more general than the first definition (under the assumption of the time constraints imposed by the ordering relation), based on the following observations:

- the first definition unnecessarily restricts the implementation of behaviour *B*, since it prescribes the (relative) time at which the activity represented by action *e* must start. We consider the starting time irrelevant to the fundamental purpose of this activity;
- in general, whenever the start of some activity is considered relevant, it can be modelled using the second definition, by defining it as an action.

Design and specification

We define a *design* as a composition of (basic) design concepts. We consider a design of a system as a *prescription* for implementation, as opposed to a description of the system characteristics that can be observed. We assume therefore that all system characteristics prescribed in a design should be implemented. We apply a prescriptive interpretation of design in our work, since we want to prescribe that certain behaviour characteristics, which can in general not be observed, have to be implemented. An example of such a characteristic is the causal relationship between two actions.

A design constitutes a conception of a system, which only exists in the designer's mind. We define a *specification* as the symbolic representation of a design. Specifications are needed to allow designers to document, communicate and reason about designs during the design process.

1.1.2 Structuring techniques

The specific composition of basic design concepts in a design defines the *structure* of this design at the finest level of granularity. In general, we are interested in the structure of a design at the coarsest level of granularity, i.e., the main structure, where we consider the design as a composition of multiple parts (sub-designs). Structuring techniques aid a designer in composing a design from multiple sub-designs. These techniques may structure different aspects of a system. For example, a sub-design may represent a system part, a system requirement or a part of its state-space.

The importance of structuring system designs is well motivated in the literature ([61, 65, 57, 80, 54]). The purpose of structuring is generally twofold: (i) to improve the comprehensibility of the design, or (ii) to facilitate the construction of the system by prescribing its implementation structure. Examples of derived purposes are reusability of components, maintainability, open-endedness and concurrent implementation of (sub-)designs.

For example, various specification styles to structure specifications ([75]) have been developed for the formal specification language LOTOS ([71, 4]). These styles can be used to improve the comprehensibility of specifications. A particular style, the resource-oriented style, can be used to prescribe the implementation structure of a system, provided that we interpret a process in the specification structure as the representation of a system part.

This thesis considers two basic structuring techniques from [16, 57]. For an elaborate treatment of structuring techniques based on the basic design model developed in this thesis we refer to [80, 82, 81].

1.1.3 Refinement operations

A distributed system can be developed using a top-down design approach based on stepwise refinement. *Refinement* is defined as the addition of design details to a design through the replacement of abstract system characteristics by more concrete system characteristics, bringing the design closer to the real system. In a systematic design methodology, multiple designs of a system are made at subsequent abstraction levels during the design process.

The terms *abstract design* and *concrete design* are often used in combination to denote two designs defined at subsequent abstraction levels, where the concrete design is a refinement of the abstract design. The activity of transforming an abstract design into a concrete design is called a *design step* or *refinement step*.

The design activities performed in a design step are determined by *design objectives*. The design objectives of a design step define which characteristics of the abstract design have to be refined and which system requirements have to be fulfilled by the concrete design. In order to perform a design step, *design operations* or *refinement operations* have to be developed. Refinement operations provide the technical means to achieve generic design objectives, by defining the manipulations of (basic) design concepts needed to perform a design step and by defining rules to determine the correctness of these manipulations. Figure 1.1 illustrates two alternative correct refinement steps.

The assessment of the correctness of a design step consists of: (i) validating whether the resulting concrete design incorporates the added system requirements and (ii) assessing whether the concrete design conforms to the abstract design. The validation of system requirements can be performed by analysing certain properties of the concrete design. The conformance between an abstract design and a concrete design can be checked by assessing whether these designs are related according to a certain conformance relation.

Figure 1.2 shows the relationships between some of the concepts introduced above. Conformance relations *C* and *C'* determine the conformance between abstract design *n* and concrete design *n+1* and between abstract design *n+1* and concrete design *n+2*, respectively. Requirements *R1* and *R2* represent the system requirements that are incorporated in design *n+1* and design *n+2*, respectively.

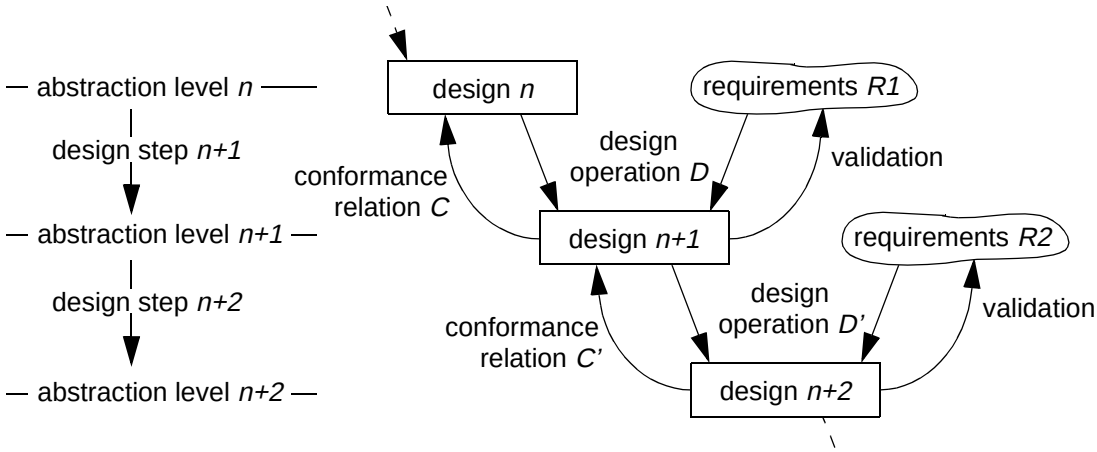


Figure 1.2: Design steps in the step-wise refinement approach

1.2 Design methodology

A *design methodology* is defined as a collection of design methods which are based on design concepts and supported by design notations and tools. The success of a design methodology is determined by the support offered to designers for producing correct designs in an effective way.

Figure 1.3 depicts the main elements that play a role in the development of a design methodology for distributed systems. We argue that any effective design methodology should be based on a carefully chosen and precisely defined (basic) design model. Below we motivate the central role of a design model by briefly discussing how a design model influences the other elements depicted in Figure 1.3.

A *design notation* is necessary to produce specifications. Therefore, a design notation should enable the intuitive and concise representation of design concepts and their possible compositions. Metaphorically, a design notation can be considered as the window through which the underlying design concepts are made visible (tangible) to designers.

A *formal model* can be used to define precisely and unambiguously the meaning of a design model and its corresponding design notation in terms of mathematical concepts. The map-

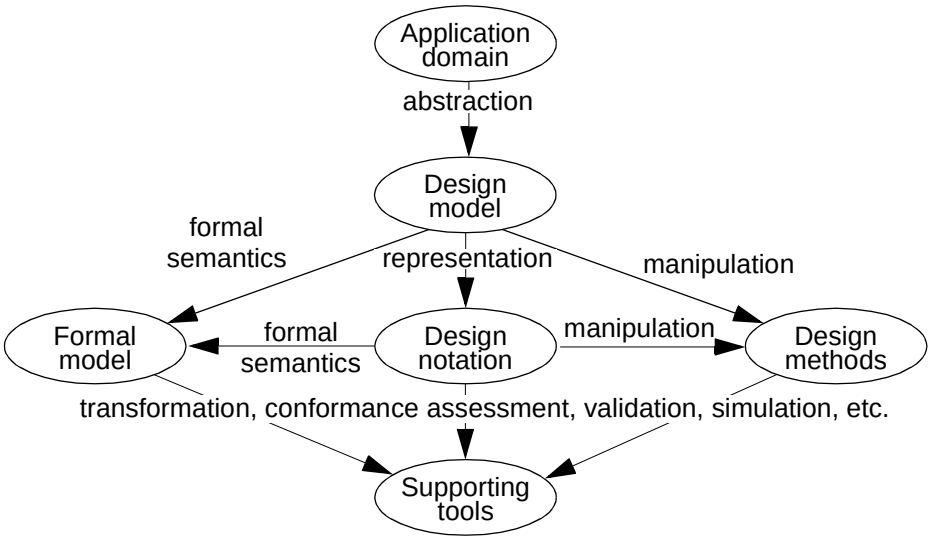


Figure 1.3: Actors in the development of a design methodology

ping of the design model and its design notation onto a formal model is called the *formal semantics* of the design model and its design notation, respectively. In this way, the formal model allows the use of mathematical models to compare, analyse and manipulate designs. Section 2.3 elaborates on the relationships between a design model, a design notation and a formal model.

Design methods provide guidelines to a designer to perform design steps that are necessary in the design process. These guidelines comprise the definition of generic abstraction levels, which are called *design milestones*, in terms of well-defined and generic design objectives. These guidelines can also include the type of system requirements considered at the design milestones, and the definition of structuring techniques and generic design operations to reach these milestones.

Supporting tools enable the (partially) automated analysis of specifications, such as verification, validation, simulation and performance analysis. Furthermore, design operations may be also (partially) automated by tools for the (semi-)automatic transformation of abstract designs into more concrete designs. A formal model provides the mathematical basis for the correct implementation of these (software) tools. The development of tool support is not considered in this thesis.

The *application domain* is represented in Figure 1.3 as the environment in which a design methodology is applied. The definition of (basic) design concepts should be based on the identification of (elementary,) common and generic characteristics of the objects in the application domain. In our case, these objects are existing distributed information systems and distributed information systems that have to be implemented.

Indirectly, our application domain also comprises objects that can be considered as distributed systems and therefore can be modelled using the design model presented in this thesis. For example, our design model is also applicable to the area of business process redesign, since business processes can be modelled as distributed systems. In particular, the design

methodology for business process redesign developed in the Testbed project [19, 20, 21] is based on the basic design model presented in this thesis.

Example

An example of a comprehensive design methodology for distributed systems is described in [6]. This design methodology uses the FDT LOTOS, and has been elaborated in the ESPRIT II Lotosphere project. Some design methods developed in the Lotosphere project are supported by an integrated tool environment (Lite). This design methodology is based on the basic design concepts of process, interaction and temporal ordering. This methodology also specializes basic design concepts and introduces composite ones in order to specify services and protocols. Examples of specializations of basic design concepts are service primitives, service access points and protocol data units. The generic structures for specifying services and protocols are examples of composite design concepts.

1.3 Objectives

The aim of this thesis is to contribute to the development of basic conceptual means for distributed systems design. We build on previous work reported in [16, 57, 67, 66], which defines, amongst others, the following expressive set of basic conceptual means:

- a basic design model, which consists of the following basic design concepts:
  - *functional entity*: a logical or physical part of a system;
  - *action*: a unit of activity performed by a functional entity;
  - *interaction*: a common action performed by two or more functional entities;
  - *causality relation*: defines the condition for the occurrence of an (inter)action, which is called the *causality condition*, in terms of how the (inter)action depends on the occurrences or non-occurrences of other (inter)actions;
  - *action point* and *interaction point*: logical or physical location at which actions or interactions occur, respectively.

  These concepts are defined in two separate, but related domains: the *behaviour domain*, which comprises the action, interaction and causality relation concepts, and the *entity domain*, which comprises the action point, interaction point and functional entity concepts;
- two basic structuring techniques for behaviour (de)composition:
  - *causality-oriented behaviour structuring*, which allows the structuring of a complex behaviour in terms of less complex sub-behaviours and their (causal) relationships. This technique is based on the decomposition of a causality relation by means of a syntactical construct, which allows an action and the condition for its occurrence to be defined in distinct sub-behaviours;
  - *constraint-oriented behaviour structuring*, which allows the structuring of a behaviour in terms of a conjunction of conditions and constraints on actions defined in separate sub-behaviours. This technique is based on the decomposition of an action

into an interaction, such that multiple interaction contributions can be distributed over distinct sub-behaviours;

- two basic design operations for the refinement of an abstract behaviour (containing abstract actions) into a concrete behaviour (containing concrete actions):
  - *causality refinement*, in which causality relations between abstract actions are replaced by causality relations involving their corresponding concrete actions and some inserted actions;
  - *action refinement*: in which an abstract action is replaced by an activity involving multiple concrete actions and their causality relations.

The work on the conceptual means above is not complete yet. The following aspects need to be elaborated in more detail:

- *semantics*: the meaning of the basic design concepts is defined in architectural terms, i.e., comprehensible to an architect (or designer) of distributed systems, and using natural language. A formal semantics has to be defined in order to add precision and to allow the use of mathematical models to compare, analyse and manipulate designs;
- *action attributes*: the information, time, location and probability attributes of actions are defined to model the result, time moment, location and conditional probability of action occurrences. The modelling of constraints on the values that can be established by (a combination of) the information, time and location attributes, possibly involving value references between causally related actions, needs more attention. The modelling of the conditional probability of action occurrences is only considered for the sequential composition of actions so far;
- *repetitive behaviours*: the causality-oriented structuring technique allows in principle the modelling of (infinitely) repetitive behaviours. Currently, repetitive behaviours can only be modelled by means of copying the definition of the repeated sub-behaviour in the behaviour specification, which becomes infeasible in case of (infinitely) many repetitions;
- *extensions to refinement rules*: the scope of the action refinement and causality refinement operations is limited to behaviours defining conjunctions of action relations (i.e., relations between actions). The refinement of behaviours containing disjunctions of action relations is not yet fully supported.

The contribution of this thesis consists of the elaboration of the above aspects for the action concept and the causality relation concept. The results obtained with the action concept also apply to the interaction concept, since an interaction is considered a refinement of an action. Therefore, the interaction concept is not considered separately in this work. This thesis does not elaborate the basic design concepts of the entity domain.

Summarizing, the objectives of this thesis are:

1. the complete definition of the action concept and the causality relation concept, including the information, time, location and probability attributes, in architectural terms and in formal (mathematical) terms;

2. the complete definition of the causality refinement and action refinement operations, supporting the refinement of any behaviour involving conjunctions and disjunctions of action relations;
3. the extension of the causality-oriented structuring technique with the means to support the definition of (infinitely) repetitive behaviours in an effective way.

1.4 Approach

The structure of this thesis reflects the successive elaboration of the objectives identified above. The approach applied here is explained below.

Objective 1: Architectural and formal definition of the action concept and the causality relation concept

Chapter 2 gives an overview of our basic design model, which introduces, motivates and reconsiders (if necessary) the basic design concepts identified in [16, 57, 67, 66]. Subsequently, this chapter concentrates on the action and causality relation concepts in order to

- give a precise architectural and formal definition of the action concept; and
- define and structure the relevant characteristics of action relations in terms of an abstraction hierarchy.

This abstraction hierarchy allows a modular approach towards the development of the causality relation concept, which constitutes the main part of this thesis. A module corresponds to a definition of the causality relation concept that allows the modelling of certain characteristics of action relations identified in the abstraction hierarchy. Modules are defined at subsequent hierarchical levels, in which causality relations preserve the characteristics defined at preceding levels.

Chapters 4 and 5 develop the basic module, in which causality relations model temporal ordering relations between actions, without considering the information, time and location attributes, but including the uncertainty attribute. The uncertainty attribute is an abstraction of the probability attribute of [16], which models whether an action must or may occur when its causality condition is satisfied. The scope of Chapter 4 is limited to behaviours consisting of only two actions, in order to restrain the large variety of temporal relations that can be defined using the causality relation concept, making our analysis intelligible. Chapter 5 considers behaviours consisting of multiple actions and gives a full definition of the conjunction and disjunction operators on causality conditions.

Chapter 6 develops three orthogonal modules, which support the independent design of information, time and location attribute constraints, including the design of references between attribute values of causally related actions. Chapter 6 also develops a mixed module, which integrates the previous modules to allow the design of constraints on combinations (mixtures) of information, time and location attributes, including references between values of distinct attribute types. All four modules are extensions of the basic module, such

that the causality relation concept obtained in these modules is a refinement of the definition used in the basic module.

Chapter 7 develops two extensions of the basic module that model two types of probability attributes: the integral probability attribute and the stochastic probability attribute. The integral probability attribute models the (conditional) probability that an action occurs once its causality condition is satisfied, in terms of a set of real numbers in the range from 0 to 1. This attribute can be defined independently of the other action attributes, under certain conditions. The stochastic probability attribute models the (conditional) probability of the occurrence of an action as a function of the time moments at which it is allowed to occur. This attribute can be considered as a refinement of the use of the time attribute in combination with the integral probability attribute.

The definitions of the causality relation concept developed in our hierarchical structure are accompanied by their corresponding formal definitions throughout this thesis. These formal definitions are presented in separate sections headed “Formal definition”. The formal semantics of causality relations is defined in terms of the constraints they impose on the execution of the involved actions. This corresponds to the definition of the possible behaviour executions allowed by these causality relations. Chapter 3 presents the so called execution model which is used to define the formal semantics of causality relations.

Objective 2: Definition of action refinement and causality refinement operations

Chapter 8 presents an integrated set of methods for supporting action refinement and causality refinement. Refinement rules are defined indirectly, by defining abstraction rules for assessing the conformance between an abstract behaviour and the corresponding concrete behaviour in these refinements.

Chapter 8 illustrates our set of methods by means of a case study consisting of the design of a system that supports a client-server interaction. At the highest abstraction level we assume that direct interactions between the client application and the server application are possible. At a lower abstraction level we implement these interactions using a federation of remote traders, which communicate via a common communication infrastructure.

Objective 3: Repetitive behaviours

Chapter 9 extends the causality-oriented structuring technique defined in [16] to allow the modelling of (infinitely) repetitive behaviours. For this purpose, we introduce the notions of behaviour type and behaviour instantiation. Through behaviour instantiation one can (dynamically) create multiple instances of a single behaviour type definition.

Case study: OSI Connection-oriented Transport Service

Chapter 10 applies our basic design model to the modelling of the behaviour of the OSI Connection-oriented Transport Service. This case study also includes the modelling of timing and probability characteristics imposed by the QoS parameters of the transport service.

Figure 1.4 depicts the structure of the remainder of this thesis, defining the main dependencies between chapters.

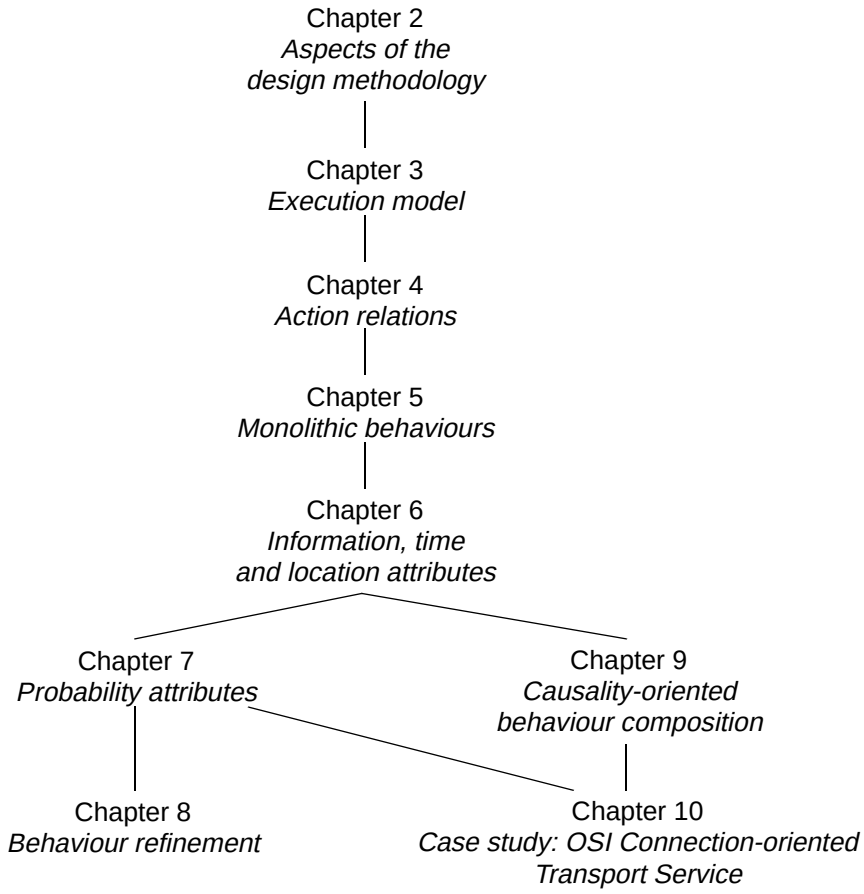


Figure 1.4: Structure of this thesis

References to related work are given in some of the chapters, in separate sections. Two indexes are added at the end of this thesis. The first index is an alphabetical list of terms and the second index is a list of symbols and functions that are introduced and referred to in this thesis.

Chapter 2

Aspects of the design methodology

This chapter reviews some aspects of our design methodology that are relevant for a proper understanding of this thesis. This review comprises aspects of our design model, design approach, design notation and formal model. Most attention is paid to the design model.

This chapter precisely defines the action concept and explains its use to design the relevant characteristics of activities. Furthermore, the notions of related and independent actions are discussed. The relevant characteristics of relations between actions are identified and structured in terms of an abstraction hierarchy. This abstraction hierarchy determines our approach towards the development of the concept of causality relation in the next chapters.

The structure of this chapter is as follows. Section 2.1 introduces and motivates the basic design concepts we consider necessary to design distributed systems. Section 2.2 discusses our approach towards systems design, including some basic design operations. Section 2.3 addresses the representation and formal definition of basic design concepts. Section 2.4 precisely defines the action concept and the related interaction concept. Section 2.5 defines the notions of related actions and independent actions, and identifies and structures the relevant characteristics of relations between actions. And Section 2.6 presents the conclusions.

2.1 Basic design concepts

This section introduces and motivates the basic design concepts we consider necessary to design distributed systems. These concepts are structured into two separate, but related conceptual domains: the entity domain and the behaviour domain.

2.1.1 Entity domain

Three basic design concepts are identified in the entity domain: entities, interaction points and action points.

Entities

The entity concept is identified in many design models as an important concept for the modelling of systems. Alternative terms that are used to denote entities are: ‘objects’, ‘modules’ or ‘resources’.

We define an entity as a carrier of certain characteristics. For example, a sculpture is an entity that is made of certain materials, in a certain shape and with a certain colour. The materials, shape and colour are some of the characteristics of a sculpture. Another example of an entity is a communication network. Some of the relevant characteristics of this entity are the type of data transfer (e.g., connection-oriented or connection-less), and the supported quality of service.

The entity concept is necessary when we have to consider the existence of something apart from its characteristics ([79]). The entity concept allows one to model that something exists or has to be created, without the necessity to model its characteristics. For example, with the entity concept we are able to identify different sculptures without describing the different sets of characteristics of each sculpture. With the entity concept we are able to consider the materials, shape or colour of a specific sculpture. Furthermore, we are able to distinguish between two sculptures carrying identical characteristics by assigning these sculptures different names.

In our application domain, entities are logical or physical (parts of) distributed systems. The entity concept allows one to model that (parts of) distributed systems exist or have to be created, without being forced to consider or define their characteristics explicitly at the same time. In addition, we want to distinguish the different distributed systems (or system parts) we identify. Therefore, we consider each distinguished distributed system (part), or entity, to be unique and we assume that we can unambiguously refer to them by using entity identifiers (entity names).

This thesis focuses on the design of the behaviour of distributed systems. In this respect, an entity (or distributed system) can be defined as a carrier of behaviour characteristics. Section 2.1.2 discusses basic design concepts to design the behaviour of distributed systems.

Interaction points

An entity makes (some of) its characteristics accessible for the entity's environment through its interaction points. *Interaction points* model the logical or physical mechanisms through which an entity can interact with the environment. Therefore, an entity can be said to be delimited by its interaction points from the perspective of its environment. As a consequence, an entity without interaction points is meaningless for its environment.

A distributed system interacts with other systems in its environment, which are called its *users*, by performing common activities, which are called *interactions*, via its interaction points. The contribution of the system in these interactions and the relationships it establishes between these contributions, defines the function of the system. This function represents the purpose of the system and forms the initial system specification.

For example, the entire surface of a sculpture may be considered as a single interaction point. Possible interactions at this interaction point are the touching of the surface or the reflection of light by the surface. The purpose of the sculpture may be to appeal to the observer's imagination in a certain way, e.g., by means of its shape or colour. Another example of interaction points are the network access points of a communication network.

Possible interactions at network access points are the establishment or release of a connection, and the sending or receiving of data. The purpose of a communication network is the transfer of data from the sending user to the receiving user.

Two or more entities can only interact when they have one or more interaction points in common. For example, touching the surface of a sculpture is a common activity of the observer and the sculpture. For example, in order to exchange messages via a communication network, network users must be attached to one or more network access points. Since network access points are interaction points between the communication network and its users, network access points are also interaction points of the network users.

Figure 2.1(i) depicts an entity structure of three entities, representing a communication network and two users. Each user is attached to the communication network via a single network access point (interaction point), which is represented by the intersection (overlap) of their entities. The intersection symbolizes that an interaction point models a common mechanism of two (or more) entities through which they can interact.

Figure 2.1(ii) depicts an alternative representation, in which entities are graphically represented by non-overlapping polygons with cut-off corners. Interaction points are represented by ovals, which overlap small parts of the entities that share these interaction points. Names may be represented within entities or interaction points in order to identify them uniquely. Alternatively, names may be represented in text boxes, which are linked to the corresponding entities or interaction points. The representation of Figure 2.1(ii) is used in the sequel.

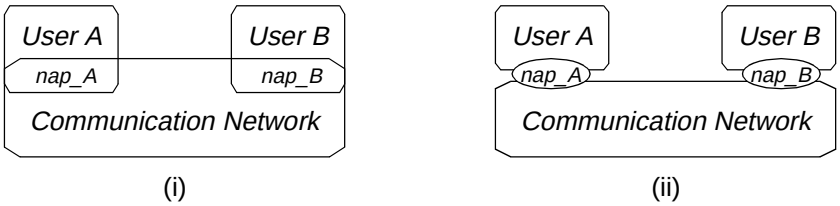


Figure 2.1: Example of an entity structure

Entity (de)composition

In order to add internal structure, an entity may be decomposed into two or more sub-entities. These sub-entities should be interconnected via internal interaction points, in order to be able to cooperate. Furthermore, the interaction points of the original entity should be maintained.

Figure 2.2(i) depicts a decomposition of the Communication Network entity of Figure 2.1, into five sub-entities, which are interconnected via four internal interaction points. Sub-entities *ES_A* and *ES_B* represent end sub-systems which are connected to two distinct local area networks represented by sub-entities *LAN_1* and *LAN_2*. Sub-entity *IS_C* represents an intermediate sub-system, which interconnects both local area networks.

The sub-entities of Figure 2.2(i) are defined at a lower abstraction level than the Communication Network entity. This is reflected by representing the Communication Network entity

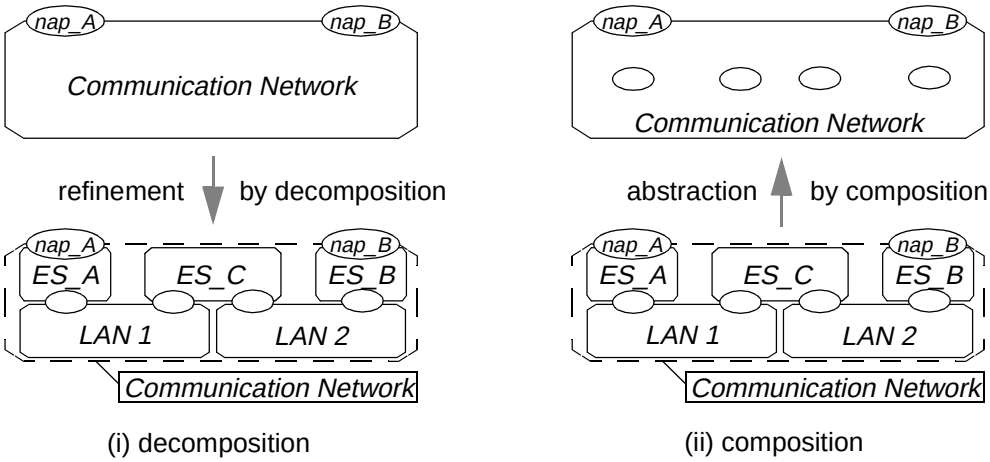


Figure 2.2: Example of entity (de)composition

as a dashed polygon in the decomposition. This representation convention enables one to relate entity structures that are defined at different abstraction levels.

The inverse of decomposition is composition. Whereas decomposition allows one to refine an entity by defining its internal structure, composition allows one to abstract from (parts of) this internal structure. For example, when an entity is composed of two or more sub-entities, one is often interested in the entity as a whole, and considers its internal structure as irrelevant.

The above does not necessarily imply that one has to abstract from all sub-entities or internal interaction points, when an entity is composed. In particular, one may choose to abstract from the sub-entities, but retain the internal interaction points. When considering the whole as a single entity, the internal interaction points are called *action points*. An action point allows one to model that some activity is performed through some internal mechanism of an entity, without specifying which sub-entities are involved in this activity.

Figure 2.2(ii) depicts a *Communication Network* entity with four action points, which is obtained from the refined entity in Figure 2.2(i) by abstracting from all five sub-entities. Action points are graphically represented by ovals which are completely contained within a single entity.

2.1.2 Behaviour domain

Behaviours model the functions of entities in terms of what these entities do. Although other characteristics of entities can be thought of, such as colour, smell or size, behaviour characteristics are the essential things we want to consider of distributed systems.

Three basic design concepts are identified to model the behaviours of distributed systems: actions, interactions and causality relations.

Activities and actions

We consider the execution of activities as the essential tasks we want distributed systems to perform. Examples of activities are transferring a data unit, performing a tele-conference meeting, filling up an application form for a life-insurance, inserting a credit card in a cash dispenser, assembling a car, or collecting income taxes of five million households. Alternative terms that are used to denote activities are ‘processes’ and ‘behaviours’.

We consider the following characteristics as being essential in the design of activities that are to be performed by distributed systems:

- an activity yields a certain *result*. Examples of results are: (i) a (produced) good, e.g., an (assembled) car, and a (baked) bread, (ii) (produced) information, e.g., the news in a paper, and an (arrived) message, or (iii) some (delivered) service, e.g., a nursed patient;
- an activity elapses in time, but is finished at a certain *moment*. After this moment the result of an activity is available to other activities. For example, an e-mail message is delivered after the last byte of information has been received, only after this moment the complete message is available;
- the result of an activity is available at a certain (logical or physical) *location*. For example, a car is assembled in a factory, news is collected in a newsroom and printed in a printing-office, and nursing takes place in, e.g., a hospital.

We introduce the *action concept* to model the above defined essential characteristics of result, moment and location of an activity, while abstracting from all other characteristics of the activity. By definition an action is performed by a single entity.

The relevant characteristics of activities are modelled by means of action attributes. The result of an activity, the moment when an activity is finished, and the location at which the result of an activity is available are modelled by the *information attribute*, *time attribute*, and *location attribute*, respectively. These attributes represent the characteristics of the occurrence of an activity that are of interest to other activities. Therefore, the combination of these attributes is also called the *result* of an action. Dependent on the characteristics we want to model, none, one or more action attributes may be defined.

In addition, we want to distinguish the different activities we identify. Therefore, we consider each distinguished activity to be unique and we assume that we can unambiguously refer to actions by using action identifiers (action names).

Figure 2.3 depicts an action *send*, which represents the activity of sending an e-mail message. Actions are graphically represented by circles, and action names are represented in the corresponding circles, or within a text-box that is associated with this circle. Action attributes are always represented within such a text-box. The values of the information, time, and location attributes are represented by the symbols ι, τ, and λ, respectively. In this example, action *send* prescribes the sending of the message “Hello Mark” from e-mail address dick@cs at 15.00 hours on the 15th of May in 1997.

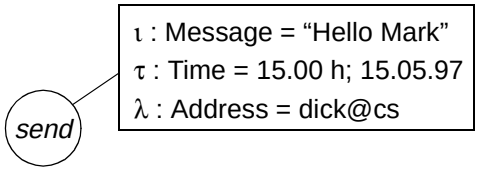


Figure 2.3: Example of an action

The definition of action *send* in Figure 2.3 prescribes one specific value for the information, time and location attributes. In general, an action definition may prescribe that an action attribute establishes a value from a range of one or more attribute values. This range may be considered as a constraint on the possible values that can be established by the attribute. For example, the definition of the time attribute of action *send* may be replaced by

τ : Time | 14.00 h; 15.05.97 < τ < 16.00 h; 15.05.97 ,

which defines that the message is sent at some moment between 14.00 and 16.00 hours on the 15th of May in 1997. The symbol “|” is used to separate the declaration of an attribute from the equation defining a constraint on the possible values of this attribute.

In this thesis, we use ad-hoc notations to represent action attributes. The interpretation of these notations should be clear and intuitive. In the above example, the data type of an action attribute is indicated; e.g., the location attribute is of type Address. The indication of data types is optional, and is only meant to make the example more appealing. Furthermore, action attributes may be omitted when no constraints are defined on their possible values; e.g., in Figure 2.3 the definition of the time attribute can be omitted when action *send* is allowed to occur at an arbitrary moment.

Interactions

In general, activities may be performed by a single or multiple entities. Activities performed by a single entity are modelled by actions. The *interaction concept* is introduced to model activities that are performed by two or more entities. Interactions have the same attributes as actions, except that each entity involved in an interaction may define its own constraints on the possible values that can be established in these attributes. Therefore, an interaction can only occur if all involved entities are willing to perform this interaction, and the constraints of all entities can be satisfied. The involvement or participation of a single entity in an interaction is called an *interaction contribution*.

Figure 2.4 depicts an interaction, which represents the sending of an e-mail message as the common activity of two entities: the e-mail system and the e-mail user (sender). Interactions are represented as two or more connected circle segments, representing the interaction contributions of the entities that are involved. A name of an interaction contribution consists of two parts separated by a dot symbol: an *interaction name*, which denotes the interaction and is the same for all involved contributions, and (optionally) an *entity name*, which denotes the entity that contributes to the interaction. Interaction names are underlined in order to distinguish them from action names.

In Figure 2.4, the left interaction contribution represents the contribution of the e-mail sender, which prescribes that the message “Hello Mark” should be sent from e-mail address

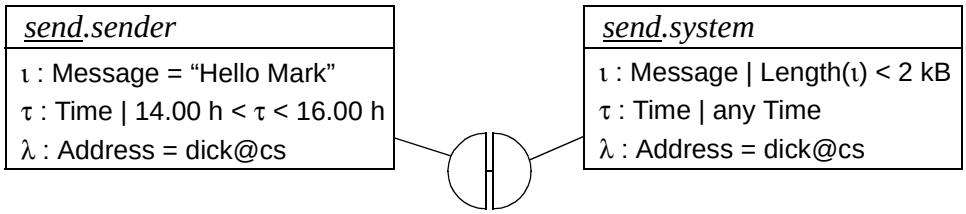


Figure 2.4: Example of an interaction

dick@cs somewhere between 14.00 and 16.00 hours (the date has been omitted for brevity). The right interaction contribution represents the contribution of the e-mail system, which prescribes that it is willing to send any message with a maximal length of 2 kB at any moment from the specific e-mail address dick@cs. The constraint any Time explicitly represents that any time moment is allowed. Alternatively, the time attribute definition of *send.system* could have been omitted.

Precise definitions of the action concept and the interaction concept are presented in Section 2.4. In the sequel, we use the term action to denote actions and interactions, unless the context explicitly indicates otherwise.

Relations

In addition to activities, we consider the execution of relationships between activities as the essential tasks we want distributed systems to perform. In general, many different relations may be defined between activities. Examples of some complex relations are:

- an acknowledgement is sent after a timer has expired and no messages have been received before the timer expired;
- a batch process is executed at the end of every day in order to process all applications of life insurances that were issued that day. This implies that the batch process is related to a variable number of life insurance applications;
- a tele-conference is established when at least 75 percent of all invited persons indicate within *T* time units after the invitation that they are willing to participate. This implies that a negative confirmation can be sent as soon as 25 percent of the invited persons have responded with a negative answer, or when the maximal response time after the invitation has expired and insufficient positive answers have been received.

Since activities are modelled in terms of actions, relations between activities are modelled in terms of relations between actions. Figure 2.5 depicts an example of a simple relation between two actions *send* and *receive*, which represent the activities of sending and receiving an e-mail message, respectively. The arrow is interpreted as the relation that action *receive* may occur after action *send* has occurred. Furthermore, the information, time and location attributes of both actions are related, since the attributes of action *receive* refer to the attribute values that are established in action *send*. The attribute value of a particular action is denoted as: attribute $\text{symbol}_{\text{action name}}$.

The location attributes of both actions are defined as (e-mail) addresses. The information attributes of *send* and *receive* are defined as messages, which consist of an ASCII string and a destination or source address, respectively. Furthermore, the following constraints are defined on the information, time and location attributes of action *receive*:

- the address of *receive* is equal to the destination address in the message of *send*;
- the address of *receive* must be in the domain @cs;
- the message of *receive* is composed of the ASCII string in the message of *send* and the (source) address of *send*;
- the length of the message of *receive* can be maximal 2 kB;
- the time attribute of *receive* defines that the message should arrive at the destination within ΔT time units after it has been sent.

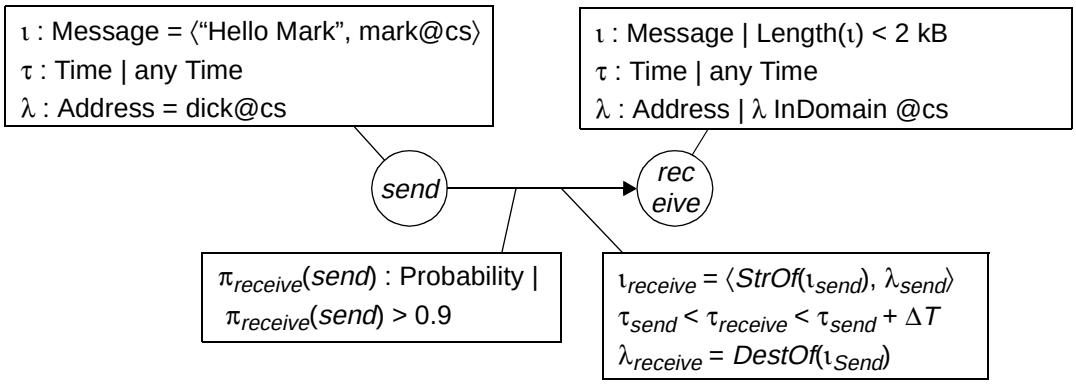


Figure 2.5: Example of two related actions

The above example introduces an additional type of constraint on the action attributes of action *receive*, viz., constraints that are imposed by the relation between actions *send* and *receive* in terms of references between the attribute values of both actions; e.g.: $\iota_{receive} = \langle StrOf(\iota_{send}), \lambda_{send}\rangle$. These constraints are distinguished from constraints that are imposed by action *receive* itself, e.g.: Length(ι) < 2 kB.

This distinction is reflected in Figure 2.5 by associating separate text-boxes with action *receive* and with the relation between actions *send* and *receive*. In the sequel, we allow the representation of constraints imposed by action relations in the text-box that is associated with an action. However, the conceptual distinction between both types of constraints should be kept in mind.

In addition, the example in Figure 2.5 introduces the *probability attribute* as an attribute of a relation between actions. In the example, the relation between actions *send* and *receive* defines a condition for the occurrence of *receive*, viz., that *send* must have occurred before *receive* is allowed to occur. The probability attribute $\pi_{receive}(send)$ defines the conditional probability that *receive* occurs after *send* has occurred. In this case the probability is defined to be larger than 0.9, which means that from every 100 executions of this behaviour in which a message is sent, on the average at least in 90 of these executions the message is delivered. Consequently, the probability attribute models the unreliability of (the implementation of) the relation between actions *send* and *receive*. Similar to the above, we allow

(constraints on) the conditional probability of action occurrences to be represented in the text-boxes that are associated with these actions.

We introduce the concept of *causality relation* to design relations between actions. One of the main objectives of this thesis is to develop a better understanding and formalization of this concept. For this purpose, Section 2.5 identifies the relevant characteristics of relations, and structures them into an abstraction hierarchy in order to allow a step-wise development of the causality relation concept.

Activity (de)composition

An action is the most abstract model of an activity, which defines *what* result is established by this activity, when and where this result is available, and with what probability. In order to define *how* this activity establishes its result, we have to model the activity in more detail.

A more detailed model of an activity is obtained by decomposing it into multiple sub-activities and their relationships. The relevant characteristics of these sub-activities can be modelled again by distinct actions at a lower abstraction level. The action *concept* thus is applied at various abstraction levels and granularity.

Figure 2.6 depicts the modelling of the activity of sending a message, at two different abstraction levels. At the most abstract level, a single action *send* models what result is established by the entire activity *Send*. At a more detailed level, activity *Send* models how this result is achieved by decomposing the activity into four related sub-activities, which are modelled by four distinct actions. Intuitively, these models are considered consistent if the result of action *send_confirm*, which is the final action of activity *Send*, corresponds to (conforms to) the result of action *send*.

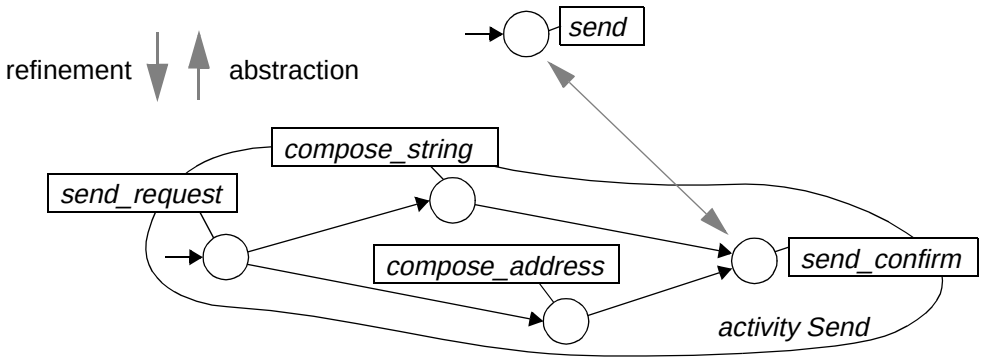


Figure 2.6: Example of activity (de)composition

The purpose of activity decomposition is to model an activity in more detail. For example, the decomposition of Figure 2.6 allows one to model the time it takes to send a message in terms of the difference between the time moments of actions *send_request* and *send_confirm*, which model the initial command to start the composition of a message and the final command to submit the message to the network when it's ready.

The inverse of activity decomposition is activity composition, i.e., the composition of an activity from two or more actions representing sub-activities. Activity composition allows

one to abstract from the internal structure of an activity, and to focus on what is established by this activity as a whole. By applying activity composition, an activity is modelled by a single abstract action, such that the result of this action conforms to the result that is established by the entire activity.

For example, when composing activity *Send* from actions *send_request*, *compose_string*, *compose_address* and *send_confirm*, we may not be interested in the time it takes to compose a message, or the possibility to compose the ASCII string and destination address of a message independently. Action *send* abstracts from these details that are irrelevant for our modelling purposes.

The terms action and activity are used to denote (parts of) behaviours at subsequent abstraction levels. Actions are abstractions of activities, while activities are compositions of actions defined at a lower abstraction level. In Figure 2.6, actions *send_request*, *compose_string*, *compose_address* and *send_confirm* are defined at a lower abstraction level than action *send*.

2.1.3 Relation between the entity domain and the behaviour domain

The entity domain and the behaviour domain are related to each other by an assignment relation. Behaviours have to be executed by a carrier for which we introduced the notion of entity, therefore each behaviour should be assigned to an entity which performs this behaviour, and each entity should be assigned to a behaviour which defines the function of this entity. For brevity, we denote the assignment relation as the assignment of behaviours to entities.

The following consistency rules must be obeyed, when assigning behaviours to entities:

1. actions of a behaviour happen at action points of the entity to which the behaviour is assigned;
2. interactions of a behaviour happen at interaction points which are shared by the entities to which the interaction contributions are assigned. Interactions between entities can only occur at the interaction points they share;
3. related actions and related interaction contributions should be assigned to the same entity.

The third rule implies that behaviours consist of one or more related actions or interaction contributions. Furthermore, behaviours are delimited by interaction contributions in a similar way as entities are delimited by interaction points. Behaviours can only interact with other behaviours via these interaction contributions.

Figure 2.7 depicts the assignment of three behaviours to three entities. Behaviours are denoted by rounded rectangles. Behaviour B_2 represents the acceptance of a message from the sending user, the sending of this message via a PDU by some internal protocol entity, the receipt of this PDU by some internal remote protocol entity, and the delivery of this message to the receiving user. Behaviours B_1 and B_3 represent the submission of a message

to the network by the sending user, and the acceptance of a message from the network by the receiving user, respectively.

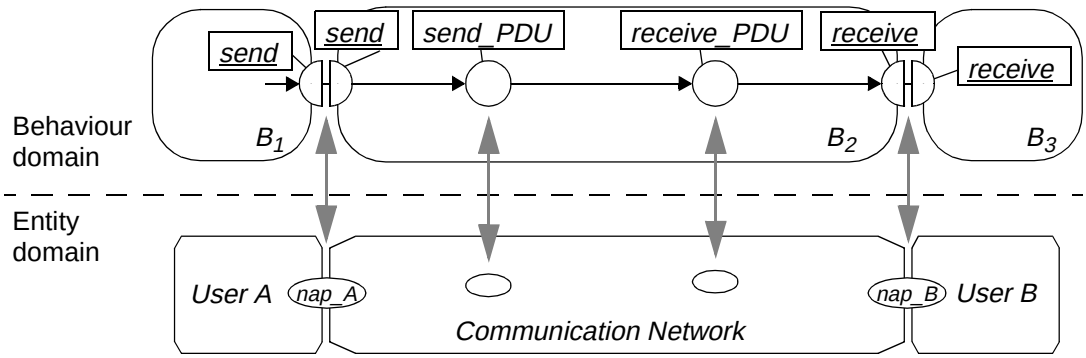


Figure 2.7: Example of the assignment of behaviours to entity

The assignment of actions and interactions to action points and interaction points is performed via the location attribute, respectively. Action points and interaction points model the mechanisms through which actions and interactions are performed, respectively. In our design model, we only consider the location of these mechanisms, such that action points and interaction points are represented by their locations. For example, the location attribute of interaction *send* is defined as: $\lambda_{send} = nap\_A$.

Partitioned actions

For reasons of comprehensibility, one may want to structure a behaviour as a composition of multiple interacting sub-behaviours, without assigning these sub-behaviours to distinct entities. For example, these sub-behaviours may represent constraints or requirements for which it is undefined whether they are implemented by the same or distinct entities.

In this case, an action that is distributed over multiple sub-behaviours is called a *partitioned action*. Similar to an interaction, a partitioned action is a common activity of multiple sub-behaviours. However, in contrast to an interaction, the sub-behaviours that share a partitioned action are not assigned to distinct entities.

In this thesis, we concentrate on the behaviour domain. Since the differences between interactions and partitioned actions can be found in their assignment to entities, the distinction between partitioned actions and interactions is irrelevant in this work. Therefore, we use the term interaction in the sequel to denote both interactions and partitioned actions.

2.2 Design methodology

This section presents our approach to distributed systems design, generic design milestones in the design process, and basic design operations that are necessary to reach these milestones.

2.2.1 System perspectives

Our approach to systems design is based on a careful consideration of the system concept. A generic definition of a system can be found in Webster's dictionary:

> *A system is a regularly interacting or interdependent group of items forming a unified whole*.

This definition reflects two different perspectives of a system: an *external perspective* and an *internal perspective*. The external system perspective only defines *what* function is performed by a system (unified whole), and considers only the possible interaction of this system with its environment. This implies that a system is defined as an entity with one or more interaction points, and its behaviour is defined by one or more interaction contributions and their relations.

The internal perspective defines *how* the system function is performed by an internal structure (group of items). We distinguish two internal perspectives: an *integrated perspective* and a *distributed perspective*. The integrated perspective defines the internal system functions, while abstracting from the internal system parts that perform these functions. This implies that a system is defined as an entity with one or more interaction points and one or more action points, and its behaviour is defined by one or more interaction contributions, which are related through one or more actions.

The distributed perspective defines the internal system parts that perform the internal system functions identified in the integrated perspective. This implies that a system is defined as a composition of two or more sub-entities which are interconnected by (internal) interaction points, and the system behaviour is defined by two or more interacting sub-behaviours.

Figure 2.8 depicts the external, integrated and distributed system perspectives in terms of the entity domain.

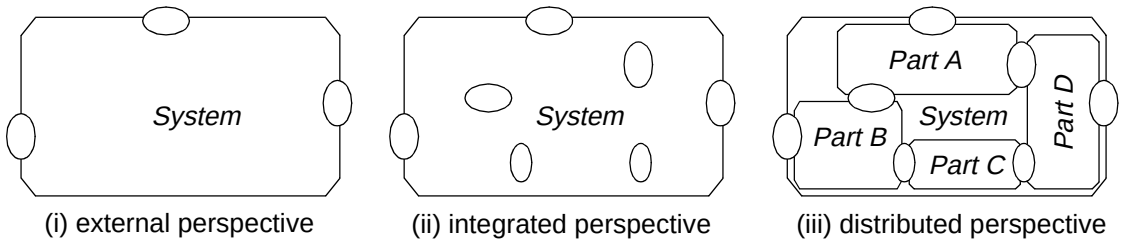


Figure 2.8: System perspectives

Repeated application of the above system perspectives provides a basis for a top-down design approach. Initially, one defines the system functions from the external perspective. Subsequently, one defines how these system functions are provided in terms of internal sub-functions from the integrated perspective. Finally, one defines the system parts that perform these internal sub-functions from the distributed perspective. This process can be applied recursively to the identified system parts, until a direct mapping onto available implementation components becomes possible.

2.2.2 Design milestones

Based on the above system perspectives, some generic design milestones relevant for distributed systems design are presented, by identifying their objectives and their relative position in a design process ([57]). For convenience these design milestones are represented primarily in terms of the entity domain.

Identification of system and environment

Objective: identification of the distributed system and the application environment, in terms of the application entities that use the system and the way these entities cooperate. This design milestone is used to determine the activities of the application environment that should be supported by the distributed system, and the degree of support to be provided.

The requirements on application support to be provided by the distributed system, determine a boundary between the system and its environment. Figure 2.9(i) depicts this design milestone. The entities are represented by dashed lines in order to indicate that they are *not* precisely defined yet.

Service definition

Objective: definition of the shared boundary between the system and its environment. This design milestone defines the common behaviour of the system and its environment, which is called the *service*, and abstracts from the many different ways in which the responsibilities and constraints for providing the service may be distributed between the system and the environment. A service is defined in terms of (common) actions and their relations.

Because the individual contributions of the system and the environment to the service are not defined, both entities are not distinguished at this abstraction level. The service is therefore assigned to a single entity, which is called the *interaction system between the system and its environment*. This interaction system only comprises that part of the environment that is relevant for the definition of the service, and abstracts from the rest of the environment. Consequently, the interaction system does not have any interaction points. Figure 2.9(ii) depicts this design milestone.

Definition of service provider and service users

Objective: definition of the behaviour of the system, which is also called the *service provider*, as it is observed by its environment. At this abstraction level responsibilities and constraints for performing the service are assigned to the service provider and to its environment, by defining the individual interaction contributions of the service provider and the environment. In this way the observable behaviour of the service provider is defined, as well as part of the observable behaviour of the environment. The environment consists of the service users or application entities.

This design milestone is useful to delimit the functionality of the service provider. The internal structure of the service provider is not considered at this abstraction level. Interac-

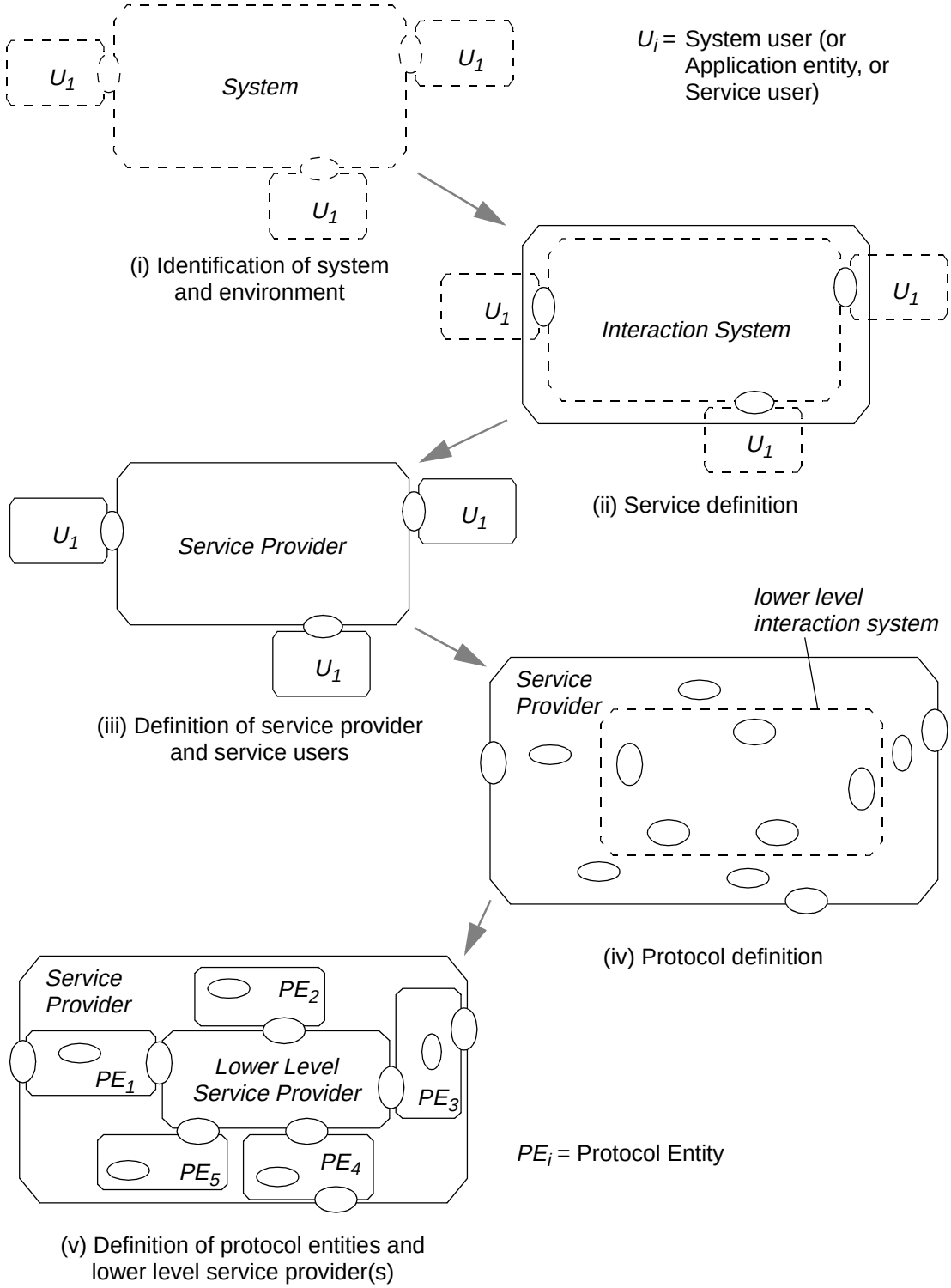


Figure 2.9: Some generic design milestones

tions and interaction points between service users and service providers are also called service primitives and service access points, respectively. Figure 2.9(iii) depicts this design milestone.

Protocol definition

Objective: definition of how the observable behaviour of the service provider is offered, while abstracting from possible decompositions of the service provider. Therefore, the internal behaviour of the service provider is defined in terms actions and their relations, which is called the *protocol*.

The definition of the internal structure of the service provider, in terms of the logical distribution of actions and associated action points, should anticipate on the design objectives of the next design milestone. This implies that the designer should already have some decomposition of the service provider in mind. Figure 2.9(iv) depicts this design milestone.

Definition of protocol entities and lower level service provider(s)

Objective: definition of the internal structure of the service provider in terms of a composition of distributed protocol entities which are interconnected by one or more lower level service providers. At this abstraction level, responsibilities and constraints for performing the protocol are assigned to protocol entities and lower level service provider(s) by defining their interaction contributions.

The common behaviour of the protocol entities and the lower level service provider(s) is defined by the protocol. This implies that a protocol definition provides the functional requirements for the definition of the lower level services and the definition of how they are used to provide the observable behaviour of the service provider.

Similar to the Service in Figure 2.9 (ii), lower level services can be assigned to distinct entities representing the (lower level) interaction systems between protocol entities and lower level service providers. Since these interaction systems only comprise that part of the protocol entities that are relevant for the definition of the lower level services, they do not have any interaction points.

Figure 2.9(v) depicts this design milestone. The dashed entity in Figure 2.9(iv) represents the interaction system between all protocol entities PE_i and the *Lower Level Service Provider.*

Interface refinement

An *interface* is defined as the common behaviour of the service provider and one (or more) service user(s) at a single interaction point. This implies that an interface definition is conceptually equal to a service definition of a service provider that shares only a single interaction point with its environment.

The presentation of the milestones of Figure 2.9(iii) and (v) may suggest that the responsibilities for performing the actions at the interfaces between the service provider and service users, or between the lower level service provider and protocol entities, respectively, have to be assigned to the involved entities. However, in some cases it is better to defer the

assignment of (part of) the responsibilities to later design steps; for example if technology imposes a specific assignment, e.g., the use of available interfaces.

Figure 2.10 gives an example of this, where the local constraints on the occurrences of the interactions between the service users and service provider are assigned to separate entities called Local Service Interface (LSI) entities. The assignment of local service constraints to a separate entity is particularly useful in the standardization of open distributed systems, where the distribution of these constraints over the service users and the service provider can be left to the implementer.

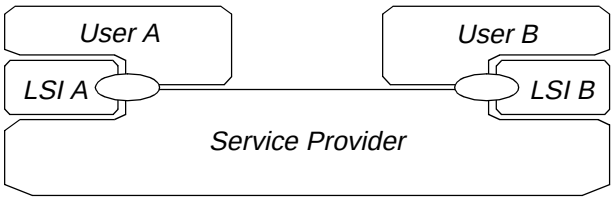


Figure 2.10: Example of interface refinement

2.2.3 Design operations

In order to bridge the gaps between the design milestones that are identified in Section 2.2.2, the following basic design operations are identified: entity refinement, action point refinement and interaction point refinement in the entity domain, and action refinement, action relation refinement and interaction refinement in the behaviour domain.

Entity refinement

The objective of entity refinement is to define the internal structure of an entity. Two types of entity refinement are distinguished:

1. the replacement of an abstract entity by a composition of concrete entities interconnected by interaction points, such that the abstract entity comprises this composition of concrete entities; and
2. the insertion of action points within an entity.

Figure 2.11 illustrates both types of entity refinement. The dashed entity represents the original abstract entity.

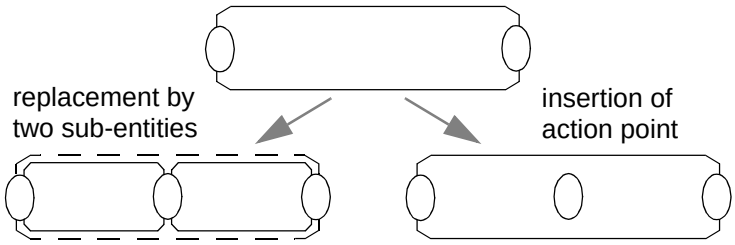


Figure 2.11: Examples of entity refinement

Action point refinement

Action point refinement allows one to define an action point in more detail. Two types of action point refinement are distinguished:

1. the replacement of an abstract action point by a collection of concrete action points, such that the abstract action point comprises the concrete action points; and
2. the replacement of an action point by an interaction point. This type of action point refinement must be performed in combination with the first type of entity refinement discussed above.

Figure 2.12 illustrates both types of action point refinement. Dashed ovals are used to indicate the abstract action point and its refinements.

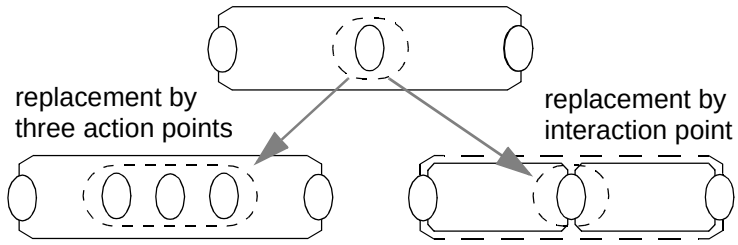


Figure 2.12: Examples of action point refinement

Interaction point refinement

Interaction point refinement allows one to define an interaction point in more detail. Two types of interaction point refinement are distinguished:

1. the replacement of an abstract interaction point by an entity with two or more interaction points, such that the abstract interaction point comprises this entity; and
2. the replacement of an abstract interaction point by a collection of concrete interaction points, such that the abstract interaction point comprises the concrete interaction points.

Figure 2.13 illustrates both types of refinement. Dashed ovals are used to indicate the abstract interaction point and its refinements.

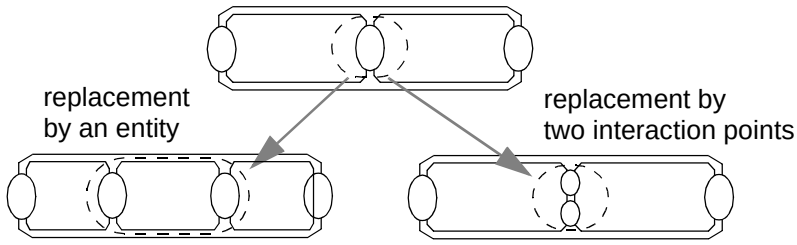


Figure 2.13: Examples of interaction point refinement

Action refinement

The objective of action refinement is to define the internal structure of the activity that is modelled by an action. Two types of action refinement are distinguished:

1. the replacement of an abstract action by an activity consisting of two or more concrete actions and their relations, such that the concrete characteristics that are modelled by this activity conform to the abstract characteristics that are modelled by the abstract action; or
2. the replacement of an abstract action by a more concrete interaction, such that the characteristics that are modelled by the abstract action conform to the characteristics that are modelled by the concrete interaction.

Figure 2.14 illustrates both types of action refinement. Figure 2.6 represents a concrete instance of the left type of refinement. Figures 2.3 and 2.4 represent a concrete instance of the right type of refinement.

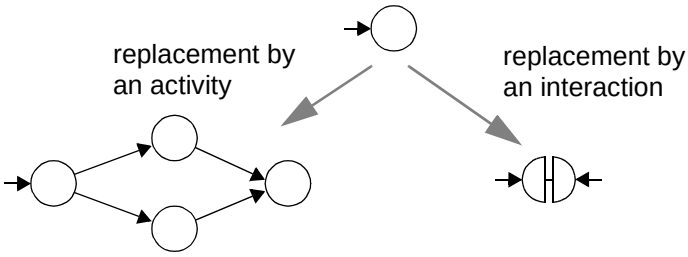


Figure 2.14: Examples of action refinement

Action relation refinement

Action relation refinement allows one to define relations between actions in more detail. This design operation consists of replacing an abstract relation between abstract actions by two or more concrete relations between these abstract actions and one or more additional concrete actions, such that the concrete characteristics that are modelled by these concrete relations and these additional concrete actions conform to the abstract characteristics that are modelled by the abstract relation.

Figure 2.15 illustrates the refinement of an action relation. Dashed ovals are used to indicate the abstract relation and its refinement, including the actions that are involved in this refinement. A concrete instance of this example is the insertion of an action *forward* in the relation between actions *send* and *receive* in Figure 2.5, which models the forwarding of the message in an intermediate system. Action *receive* is involved in this refinement, since in the refined behaviour it refers via inserted action *forward* to the attribute values of action *send*.

Interaction refinement

The objective of interaction refinement is to define the interaction contributions in more detail. Interaction refinement consists of replacing an abstract interaction by a concrete

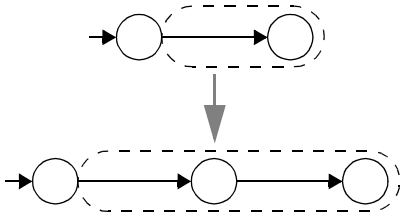


Figure 2.15: Example of action relation refinement

activity consisting of two or more concrete interactions and their relations, such that the concrete characteristics that are modelled by this activity conform to the abstract characteristics that are modelled by the abstract interaction contributions.

Figure 2.16 illustrates interaction refinement. A concrete instance of this example is the refinement of the receipt of a data unit in the receipt of two data segments at a parallel interface, which together form the original data unit.

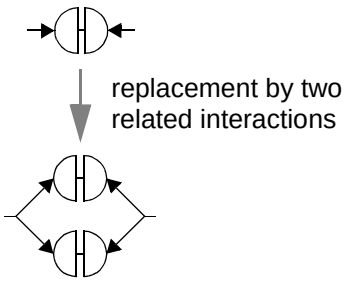


Figure 2.16: Example of interaction refinement

Combinations of design operations

More complex design operations can be defined, by applying the design operations presented above in combination. For example, interface refinement may be considered as a design operation in which interactions and the associated interaction points between two or more entities are refined using (inter)action refinement and interaction point refinement ([5, 63]). Action refinement and action relation refinement in combination allow the refinement of arbitrary behaviours consisting of actions and their relations, which we call *behaviour refinement*. Behaviour refinement is elaborated in Chapter 8.

2.3 Design notation and formal semantics

This section addresses our approach towards the representation and formal definition of design concepts. The distinction between a concept and its representation is discussed. The term formal semantics and its use is explained. And the choice of the design notation used in this thesis is motivated.

2.3.1 Specification of designs

Designs are conceptual models which are conceived and manipulated in a designer's mind. Due to our limited capabilities for capturing complex designs, we are forced to represent designs in order to allow documentation, communication and reasoning about their properties. Furthermore, designs have to be represented in computer readable and interpretable form in order to enable the (partially) automated analysis or manipulation of designs.

The symbolic representation of a design is called a *specification*. A specification is composed of notational elements (symbols) which represent the (basic) design concepts from the design model. The set of all notational elements and the rules defining their possible arrangements are called a *design notation* (or design syntax).

In this thesis we use a graphical and a textual design notation, which consist of (mainly) graphical and textual symbols, respectively. Both design notations should support the representation of the complete design model in a consistent way, such that graphical specifications can be transformed into their textual equivalents, and vice versa. Graphical symbols are often convenient to represent the structure of a design, while textual symbols are often used to represent a design in full detail.

Figure 2.17 depicts the distinction between a design and its specification, and the corresponding distinction between a design model and its design notation ([16, 73]).

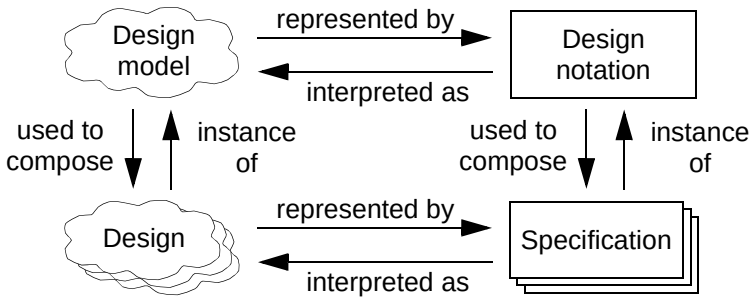


Figure 2.17: Design and specification

A design notation must guarantee the unique representation of (basic) design concepts and their combination rules. This is a necessary requirement to allow the unambiguous interpretation of a specification by different designers. The interpretation of the elements of a design notation, and their possible compositions, in terms of the concepts of the design model, and their possible compositions, is called the *architectural semantics* of a design notation.

The combination of a design model, a design notation, and the architectural semantics of this design notation in terms of the design model, is called here a *design language*. Alternatively, the term specification language is used. We prefer the term design language, since this term better reflects that a design language is used for the specification as well as the manipulation and analysis of designs at different abstraction levels.

2.3.2 Formal semantics

The *formal semantics* of a design notation defines the interpretation of the elements of this notation, and their possible compositions, in terms of the concepts of some elementary mathematical model, and their possible compositions. Such a mathematical model and its concepts are often called a *formal model* and formal concepts, respectively.

The representation of a design model in terms of a design notation in combination with the formal semantics of this design notation defines indirectly how the characteristics that are modelled by the design concepts and their combination rules are mapped onto the properties of the formal concepts and their combination rules. This mapping defines the interpretation of design concepts, and their possible compositions, in terms of formal concepts, and their possible compositions, which is called the *formal semantics of the design model*.

The representation of a formal model is called a *formal notation*. This representation in combination with the formal semantics of a design notation defines indirectly a mapping of the elements of the design notation, and their possible compositions, onto the elements of the formal notation, and their possible compositions. This mapping is called the *formal representation of a design notation*. Similarly, a mapping of the design concepts in the design model, and their possible compositions, onto the elements of the formal notation, and their possible compositions, is defined, which is called the *formal representation of a design model*.

A design language with a corresponding formal semantics (and formal notation) is called a *formal design language*. Figure 2.18 depicts the elements of a formal design language, including the mappings that can be defined between these elements. A mapping from element *E1* onto element *E2* is denoted by the domain of *E2*. A distinction is made between an architectural (design) domain and a formal domain, and between a representational domain and a semantical (conceptual) domain. For example, the mapping of the formal notation onto the design model is denoted as the *architectural semantics* of the formal notation, and the mapping of the formal notation onto the design notation is denoted as the *architectural representation* of the formal notation.

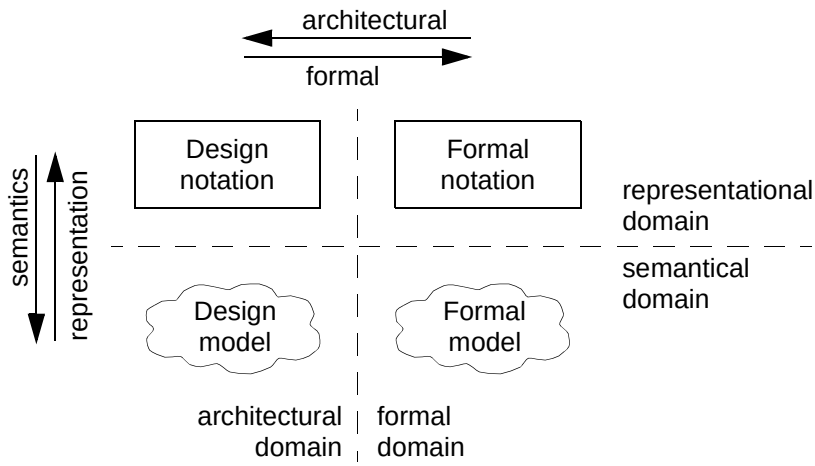


Figure 2.18: Elements of a formal design language

Motivation

The motivations for defining a formal semantics of a design language are:

- define the interpretation of specifications with mathematical precision, such that the system characteristics that have to be implemented can be determined unambiguously from a specification;
- allow one to compare different specifications in terms of mathematical structures, such that these specifications can be distinguished or considered equivalent w.r.t. (certain aspects of) the system characteristics that are defined;
- provide a basis for making automated tools facilitate the analysis and manipulation of designs. In general, the manipulation of formal concepts can be implemented more systematically than the manipulation of design concepts.

Execution model

Inherent to the distinction we make between the behaviour domain and the entity domain, a system specification consists of:

- a behaviour specification defining the behaviour of the system;
- an entity specification defining the entity structure of the system;
- a specification of the relation between the behaviour specification and the entity specification, defining the assignment of behaviours to entities.

In this thesis, the formal semantics of behaviour specifications is defined in terms of (behaviour) executions. The formal model for behaviour specifications is called *execution model*, and is based on the formal concept of *execution*. An execution defines what happens in a particular run of some system in terms of, e.g., the actions that occur, the causal relationships between these action occurrences and the attribute values that are established in these actions. The entire system behaviour is defined by enumerating all possible executions of this system.

The formal semantics of entity specifications is not defined in this thesis, nor the formal semantics of the specification of the assignment relation between a behaviour specification and an entity specification, since we think that the architectural definitions of these are already precise enough. An illustration of how the formal semantics of entity specifications can be defined is found in [25].

2.3.3 Design language

In this thesis we use a design notation that fits our needs to represent (basic) design concepts and their combination rules in a satisfactory way to discuss their development and definition. The choice of this notation is partially based on the notation of the design language that is being developed in the Testbed project [19].

Two qualitative characteristics for structuring a design language are considered:

- *completeness*: the ability to represent any design concept and combination rule of our design model;
- *conciseness*: the ease and efficiency with which these design concepts and combination rules can be represented.

Based on these characteristics, we distinguish between a basic design language and extended design languages.

Basic design language

A minimal requirement for our design language is that it supports the representation of all basic design concepts and their combination rules. This allows one to represent any design concept in our design model, since a design concept is either a basic design concept or a composition of basic design concepts. Such a composition is called a *composite design concept*. Basic design concepts are represented directly by their corresponding notational elements (symbols), while composite design concepts are represented indirectly in terms of compositions of notational elements.

Notational elements that are defined to represent basic design concepts and their combination rules are called *basic notational elements*. A design language which defines notational elements for its basic design concepts and combination rules, and does not define additional notational elements for the direct representation of composite design concepts, is called a *basic design language*.

A basic design language defines the minimal notation that is needed to achieve completeness. Consequently, the architectural semantics of a specification that is produced using a basic design language is defined exclusively in terms of basic design concepts and their combination rules.

Extended design language

In general, a basic design language lacks conciseness. Although it allows the representation of any design concept in the design model, the representation of composite design concepts may become voluminous and therefore incomprehensible and not intuitive.

In order to improve the conciseness of a (basic) design language, *extended notational elements* are introduced, to allow the direct representation of composite design concepts. This is depicted in Figure 2.19. A design language should only be extended to facilitate the representation of frequently used design concepts, limiting in this way the number of (extended) notational elements.

An extended notational element is actually a shorthand notation for a particular composition of basic notational elements that is used to represent the corresponding design concept in terms of the basic design language. This implies that an extended notational element must be defined such that it can always be transformed into a composition of basic notational elements. This guarantees consistent extensions of the basic design language.

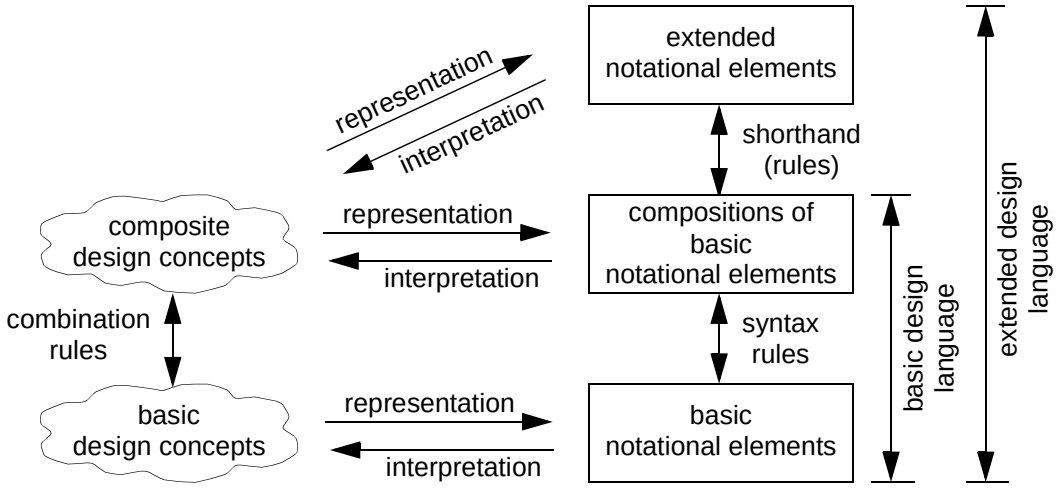


Figure 2.19: Basic and extended design language

In addition, the definition of a mapping between extended notational elements and basic notational elements facilitates the definition of the formal semantics of a design notation. In case each specification can be transformed into a composition of basic notational elements, it suffices to define the formal semantics of the basic design notation.

2.4 The action concept

This section elaborates on the action and interaction concepts, which have been introduced and motivated in Section 2.1.2. A precise definition of the action concept is presented. The interaction concept is defined insofar it differs from the action concept.

2.4.1 Basic definition

An action is defined as

> *a unit of activity that is performed by an entity, which takes place for the purpose of establishing a certain result.*

The action concept models the relevant characteristics of some activity in the real world. An action is the most abstract model of an activity which we want to consider and represent as a unit of behaviour at a certain abstraction level, i.e., the action cannot be split at this abstraction level without violating its consideration as a unit. This property is called *atomicity* and is inherent to the choice of an action as a basic design concept.

The atomicity property imposes that an action should be implemented reliably, such that:

- either an action occurs, which means that the performing entity can refer to the action occurrence and the result that has been established;
- or an action does not occur at all, which means that the performing entity can not refer to the action occurrence nor to any (partial) result that could have been established.

The reliability requirement is based on the fundamental assumption that a design should be considered as a prescription for implementation. A designer should be able to assess that some activity that is defined at an abstract level, can be made to happen in the implementation. At an abstract level a designer neither wants to be concerned with the many different ways in which an implementation provides some prescribed behaviour, nor with the many different ways in which an implementation may fail to provide this behaviour. Consequently, a designer may only define some activity by means of a single action if this activity can be implemented reliably. If the reliable implementation of an action cannot be guaranteed, the corresponding activity should be modelled by a composition of multiple actions, making this unreliability explicit ([74]).

In addition, we consider each instance of activity to be unique, and we assume that we can unambiguously refer to an action by using an action name. This implies that any action only occurs once, or not at all, in the execution of some system behaviour.

Impossible actions

We assume it is only meaningful to define (i.e., to prescribe) an action when this action occurs in at least one behaviour execution. The definition of impossible actions, i.e., actions that do not occur in any behaviour execution unnecessarily increases the complexity of a design and results in inefficient implementations. Therefore, a specification should not contain impossible actions.

The above does not imply that we do not have to consider impossible actions when developing a design model. Since designing is a creative process, when a design model is used by designers incorrect design decisions can be made, which may lead to the definition of impossible actions. The design model should assist a designer in detecting impossible actions when the behaviour definition is analysed, and enable the designer to remove impossible actions from the behaviour definition. Consequently, impossible actions have to be considered for analytical purposes while their definition has to be avoided for technical reasons.

Activity modelling

In general, different designers may identify different activities in the design of the same system behaviour, leading to different designs. Since the quality of these designs may differ significantly, design methods are developed to assist a designer in the process of identifying activities. We consider the following differences:

- the abstraction level at which an activity is modelled; and
- the use of action attributes to model the result of an activity.

An activity may be modelled by a single action, at the highest abstraction level, or by compositions of actions, at lower abstraction levels. The second option implies that the activity is decomposed into sub-activities, which are modelled by distinct actions. Consequently, different decompositions render different models of the activity.

In addition, an activity may establish distinct results in distinct behaviour executions. This may be modelled in the following alternative ways:

1. each instance of the activity that establishes a distinct result is considered a distinct activity, and is modelled by a distinct action (without using action attributes);
2. all instances of the activity are modelled by the same action, and each distinct result is modelled by an alternative attribute value of this action;
3. a combination of alternatives 1 and 2.

For example, action *receive* in Figure 2.5 models the activity of receiving a single message, which may contain different ASCII strings, depending on the string that is established in the preceding *send* action. According to the first alternative, this activity may be modelled by a set of actions, such that only one of these actions may occur, and each distinct action models the receipt of one specific message.

Designers are free to use any of the above alternatives for modelling (the results of) activities. However, in almost all cases, the second alternative is more concise and renders models that are better to understand when compared with the first alternative.

2.4.2 Action attributes

In general, a specification may prescribe many alternative executions of a behaviour, which differ, amongst others, in the actions that are executed, the relationships between the executed actions, and the attribute values that are established in the executed actions. The actual behaviour execution may be determined by interactions with the behaviour's environment or internal non-determinism, e.g., due to choice relations or unreliability.

Corresponding to the above, we distinguish between an action specification and an action occurrence. An *action occurrence* prescribes a specific execution of an action, which is characterized by the established result and the moment and location at which this result is available. We define an action occurrence as the combination of the action name and the established information, time and location values.

An *action specification* prescribes the possible occurrences of some action in any behaviour execution. In general, an action may establish different attribute values, depending on the specific behaviour execution. Furthermore, the possible attribute values that can be established may be limited. For example, action *receive* in Figure 2.5 may establish different information values depending on the message that is sent, while the possible messages that can be accepted is limited to messages with a certain maximal length.

In addition, an action may impose constraints on the possible combinations of values of different action attributes. For example, the definition of action *receive* could be extended with a constraint defining that in sub-domain `sub1@cs` messages with a maximal length of 2 kB, and in sub-domain `sub2@cs` messages with a maximal length of 4 kB can be established.

Based on the above, we define an action specification as the combination of the action name and the following attribute value domains:

- *information value domain*, which defines the possible *information values* that can be established, i.e., the possible results that can be established in the action;
- *time value domain*, which defines the possible *time values* that can be established, i.e., the possible moments at which the action can occur;
- *location value domain*, which defines the possible *location values* that can be established, i.e., the distinct locations at which the action result can be made available;
- *mixed value domain*, which defines the possible combinations of information, time and location values that can be established.

In principle, it suffices to define only the mixed value domain, since it comprises the information, time and location value domains. However, for reasons of clarity we also define the latter domains explicitly.

Time and location values

We assume the result of an activity and the location at which this result is made available, can be modelled by a single information value and a single location value, respectively. Such a value can, however, be a structured value that models a composition of sub-results or sub-locations, respectively. For example, an e-mail message may be represented as a structured information value, which consists of a destination field, subject field and contents field. Typical examples of structured location values are addresses. For example, the location "Enschede" comprises all sub-locations representing the locations of streets in Enschede.

2.4.3 Formal definition

This section defines the notions of action specification and action occurrence in terms of mathematical structures. For this purpose the following domains are defined:

- $\boldsymbol{A}$ denotes the domain of action names;
- $\boldsymbol{I}$ denotes the domain of information values;
- $\boldsymbol{T} \subseteq \boldsymbol{R}$ denotes the domain of time moments. We assume that a total ordering relation "<" is defined on all time moments in $\boldsymbol{T}$;
- $\boldsymbol{L}$ denotes the domain of locations;

We assume that the identity relation "=" is defined on the elements of these domains.

Definition 2.1 An *action specification* is defined as a five tuple $\langle a, \mathrm{I}, \mathrm{T}, \Lambda, \varsigma \rangle$, where:

- $a \in \boldsymbol{A}$, is the action name which uniquely identifies the action;
- $\mathrm{I} \subseteq \boldsymbol{I}$, is the information value domain;
- $\mathrm{T} \subseteq \boldsymbol{T}$, is the time value domain;
- $\Lambda \subseteq \boldsymbol{L}$, is the location value domain;
- $\varsigma \subseteq \mathrm{I} \times \mathrm{T} \times \Lambda$, is the mixed value domain. ■

The information, time, location and mixed value domains of action a are denoted as I_a, T_a, Λ_a and ς_a, or alternatively as $\mathrm{I}(a)$, $\mathrm{T}(a)$, $\Lambda(a)$, and $\varsigma(a)$ when required for text editor reasons, respectively.

The definition of an action attribute is optional. In case an action attribute is left undefined, the corresponding value domain should be removed (or alternatively, a specific symbol should be introduced to explicitly denote this).

Definition 2.2 An *action occurrence* is defined as a four tuple $\langle a, \iota, \tau, \lambda \rangle$, where:

- $a \in \boldsymbol{A}$, is the action name which uniquely identifies the action;
- $\iota \in \mathrm{I}_a$, is the information value that is established;
- $\tau \in \mathrm{T}_a$, is the time value that is established;
- $\lambda \in \Lambda_a$, is the location value that is established. ■

The information, time and location values of an action occurrence a are denoted as ι_a, τ_a and λ_a, or alternatively as $\iota(a)$, $\tau(a)$ and $\lambda(a)$, respectively. In case an action does not occur in a behaviour execution, the attribute values of this action occurrence are undefined.

The semantics of an action specification is defined in terms of the possible action occurrences it allows.

Definition 2.3 Action specification $\langle a, \mathrm{I}, \mathrm{T}, \Lambda, \varsigma \rangle$ defines the following set of possible action occurrences: $\{ \langle a, \iota, \tau, \lambda \rangle \mid \langle \iota, \tau, \lambda \rangle \in \varsigma \}$. ■

2.4.4 The interaction concept

An interaction is an action which is performed by multiple entities in cooperation. We define an interaction as follows:

> *a unit of common activity shared by multiple entities, in which cooperation between these entities takes place for the purpose of establishing a certain common result to which each entity can refer.*

The essential difference between the action concept and the interaction concept is the distribution of the responsibility for performing an (inter)action over multiple entities, while an action is performed by a single entity.

Reliability (revisited)

An interaction models the relevant characteristics of an activity performed through the cooperation of multiple entities. Similar to the action concept, the interaction concept is atomic and should therefore be implemented reliably, such that:

- either an interaction happens, which means that all involved entities can refer to the interaction occurrence and to the result that has been established;
- or an interaction does not happen, which means that none of the potentially involved

entities can refer to the interaction occurrence or to any result that could have been established.

Attribute value establishment

The same action attributes as defined in Sections 2.4.2 and 2.4.3 are used to model the result of an interaction. The attribute values established in an interaction occurrence are shared by all entities involved in the interaction.

Each involved entity may impose its own constraints on the attribute values that can be established in an interaction. The constraints of an entity on the possible attribute values that can be established is called an *interaction contribution*. Consequently, an interaction contribution defines under what conditions an entity is willing to contribute to the occurrence of this interaction.

An attribute value can be established if the constraints imposed by all entities on the corresponding action attribute can be satisfied. Therefore, an interaction models the result of a negotiation between the constraints of all involved entities. An interaction cannot happen if the superposition of the constraints on each of its attributes can not be satisfied by any value.

Considering the negotiation constraints between two entities in a single interaction, three basic forms of value establishment are distinguished ([74]):

- *value checking*, which represents that one entity requires a specific value x to be established, while the other entity requires a specific value y to be established. In order to allow the interaction to happen the following condition must hold: $x = y$;
- *value passing*, which represents that one entity requires a specific value x to be established, while the other entity allows any value from a set of values Y to be established. In order to allow the interaction to happen the following condition must hold: $x \in Y$;
- *value generation*, which represents that one entity allows any value from a set of values X to be established, while the other entity allows any value from a set of values Y to be established. In order to allow the interaction to happen the following condition must hold: $X \cap Y \neq \emptyset$.

Various combinations of these basic negotiation forms are possible. Furthermore, they can be extended in a straightforward way to multiple attributes and interactions involving more than two entities. In case the superposition of all interaction constraints implies that multiple alternative attribute values can be established, a non-deterministic choice is made between these values.

Actions as integrated interactions

An action may model an *integrated interaction*, which then abstracts from the individual interaction contributions of the involved entities. Actions that are not established by multiple entities are not abstractions of interactions, since such actions are not distributed over multiple entities at lower abstraction levels.

Figure 2.20 depicts an example of an integrated action *a* that is implemented as an interaction $\underline{a}$ consisting of three interaction contributions. In this example, action *a* and the interaction contributions of $\underline{a}$ may refer to some time value τ_0, which is assumed to be established somewhere else.

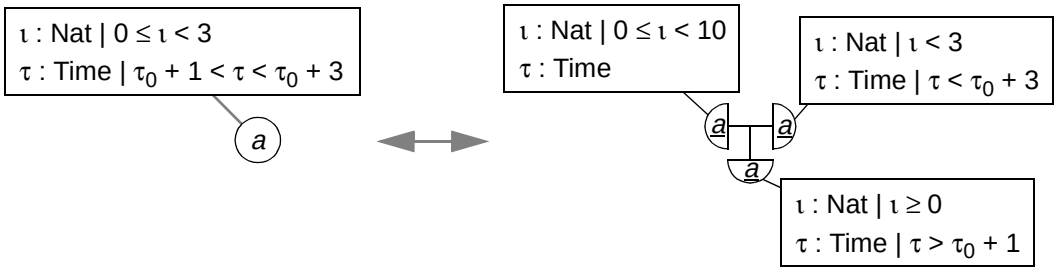


Figure 2.20: Example of an integrated interaction

Since interactions are considered as refinements of actions, the definition of the action concept and its combination rules also applies to the interaction concept and its combination rules. Therefore, for brevity and simplicity reasons, the term action is used to denote actions as well as interactions in the sequel. The term interaction is only used when we consider behaviour characteristics that are specific to interactions.

2.4.5 Formal definition

This section defines the notions of interaction contribution and interaction specification in terms of mathematical structures. For this purpose the following domains are defined:

- $\underline{\boldsymbol{A}}$ denotes the domain of interaction names;
- ***Ent*** denotes the domain of entity names;

We assume that the identity relation "=" is defined on the elements of these domains.

Definition 2.4 An *interaction contribution* is defined as a five tuple $\langle\langle \underline{a}, ent\rangle, \mathrm{I}, \mathrm{T}, \Lambda, \varsigma\rangle$, where:

- $\langle \underline{a}, ent\rangle \in \underline{\boldsymbol{A}} \times \boldsymbol{En}$, is the interaction contribution name which uniquely identifies the contribution of entity *ent* in interaction $\underline{a}$;
- $\mathrm{I} \subseteq \boldsymbol{I}$, is the information value domain;
- $\mathrm{T} \subseteq \boldsymbol{T}$, is the time value domain;
- $\Lambda \subseteq \boldsymbol{L}$, is the location value domain;
- $\varsigma \subseteq \mathrm{I} \times \mathrm{T} \times \Lambda$, is the mixed value domain. ■

The information, time, location and mixed value domains of interaction contribution $\langle \underline{a}, ent\rangle$ are denoted as $\mathrm{I}_{\langle \underline{a}, ent\rangle}$, $\mathrm{T}_{\langle \underline{a}, ent\rangle}$, $\Lambda_{\langle \underline{a}, ent\rangle}$ and $\varsigma_{\langle \underline{a}, ent\rangle}$, or alternatively as $\mathrm{I}(\langle \underline{a}, ent\rangle)$, $\mathrm{T}(\langle \underline{a}, ent\rangle)$, $\Lambda(\langle \underline{a}, ent\rangle)$ and $\varsigma(\langle \underline{a}, ent\rangle)$, respectively.

Definition 2.5 An *interaction specification* is defined as a tuple $\langle \underline{a}, IC\rangle$, where:

- $\underline{a} \in \underline{\boldsymbol{A}}$, is the interaction name which uniquely identifies the interaction;

- $IC \subset \underline{\boldsymbol{A}} \times \boldsymbol{En}$, is the set of interaction contributions, with $|IC| \geq 2$. ■

The set of interaction contributions of interaction $\underline{a}$ is denoted as $IC_{\underline{a}}$ or $IC(\underline{a})$.

The semantics of an interaction specification is defined in terms of the possible interaction occurrences it allows. An interaction occurrence is defined as an action occurrence, since both model the same characteristics of the result of an activity.

Definition 2.6 Interaction specification $\langle \underline{a}, IC \rangle$ defines the following set of possible (inter)action occurrences: $\{ \langle \underline{a}, \iota, \tau, \lambda \rangle \mid \langle \iota, \tau, \lambda \rangle \in \varsigma ,\ \varsigma = \cap \{ \varsigma_{\langle \underline{a}, ent \rangle} \mid \langle \underline{a}, ent \rangle \in IC \} \}$. ■

2.5 Action relations

This section provides the basis for the development of the concept of causality relation in the remainder of this thesis. Some basic assumptions are presented and the notions of related actions and independent actions are defined. The relevant characteristics of action relations are identified and structured in terms of an abstraction hierarchy. Based on this abstraction hierarchy, our design model is decomposed into design modules.

2.5.1 Basic definitions

We assume that two actions are either related or independent in a behaviour execution.

In general, the occurrence of an action depends on the satisfaction of some condition. When considering the conditions for the occurrence of an action *a* in a behaviour execution, we distinguish between two alternative cases:

1. the occurrence of action *a* depends on the occurrences or non-occurrences of one or more specific other actions; or
2. the occurrence of action *a* does not depend on the occurrences or non-occurrences of specific other actions.

In the first case the condition for the occurrence of action *a* has to be satisfied by the occurrences or non-occurrences of the specific other actions. This allows one to control the occurrences of individual actions of a system behaviour by making them dependent of the occurrences or non-occurrences of other actions of this system behaviour.

In the second case the condition for the occurrence of action *a* is neither satisfied nor dissatisfied by the occurrences or non-occurrences of other actions. This implies that the condition for the occurrence of action *a* is either satisfied or dissatisfied by definition. The latter alternative is, however, abandoned, since we do not want to design impossible actions. Therefore, we assume that action *a* in this case is allowed to occur from the beginning of the behaviour execution. Actions that are allowed to occur from the beginning of a behaviour execution are called *initial actions*.

Summarizing, the condition for the occurrence of an action is either satisfied by definition, i.e., from the beginning of a behaviour execution, or is satisfied by the occurrences or non-

occurrences of other actions in an execution. Therefore, a behaviour definition is composed from initial actions which initiate the behaviour and from actions that depend on other actions of the behaviour. Based on this modelling assumption, the notions of related actions and independent actions are defined below.

Related and independent actions

Two actions are *related* in an execution when the occurrence of one action depends on the occurrence or non-occurrence of the other action. This is defined precisely as follows:

> two actions *a* and *b* are related in an execution if and only if (i) the occurrence of action *a* depends on the occurrence or non-occurrence of action *b*, or (ii) the occurrence of action *b* depends on the occurrence or non-occurrence of action *a*, or both (i) and (ii) apply.

Two actions are *independent* in an execution when they are not related. Independent actions are characterized by the absence of a relation between them. Therefore, we do not consider independence as a (type of) relation.

In the sequel, we will use expressions like "action *a* depends on action *b*" or "*a* depends on *b*" in order to denote more briefly that "the occurrence of action *a* depends on the occurrence or non-occurrence of action *b*".

Action relations

In general, two actions may be related differently in different behaviour executions. An *action relation* defines how two actions are related in some (or all) behaviour executions. For example, consider a disabling relation between actions *a* and *b* in which the occurrence of *a* disables the occurrence of *b*. This action relation defines that action *a* can be related in two different ways to action *b* in different behaviour executions: (i) either *a* depends on the non-occurrence of *b*, such that *a* is allowed to occur when *b* has not occurred yet, or (ii) *a* depends on the occurrence of *b* such that *a* is allowed to occur after *b* has occurred.

Statically and dynamically related actions

In general, two actions may be related in some (or all) behaviour executions or may be independent in some (or all) behaviour executions. Based on this property, a distinction is made between statically related, statically independent and dynamically related actions:

> two actions of a behaviour are *statically related* when both actions are related in all possible executions of this behaviour;
>
> two actions of a behaviour are *statically independent* when both actions are independent in all possible executions of this behaviour;
>
> two actions of a behaviour are *dynamically related* (or dynamically independent) when both actions are neither statically related nor statically independent, i.e., both actions are related in at least one execution and are independent in at least one other execution of this behaviour.

The notions of statically related and dynamically related actions are illustrated by means of an example. Figure 2.21 depicts two behaviours *B1* and *B2*, consisting of three actions *a*, *b* and *c* in which actions *a* and *b* are initial actions. Behaviour *B1* defines that action *c* depends on the occurrences of *a* and *b*, i.e., *c* is only allowed to occur after both *a* and *b* have occurred. Behaviour *B2* defines that action *c* either depends on the occurrence of *a* and is independent of the occurrence of *b*, or depends on the occurrence of *b* and is independent of the occurrence of *a*. The symbols ■ and □ represent the *and*- and *or*-operator, respectively, which are defined in Chapters 4 and 5.

Figure 2.21: Example of statically and dynamically related actions

In behaviour *B1*, actions *a* and *c* and actions *b* and *c* are statically related actions, since they are related in any behaviour execution. In behaviour *B2*, actions *a* and *c* and actions *b* and *c* are dynamically related actions, since a behaviour execution is possible in which they are related and a behaviour execution is possible in which they are independent.

2.5.2 Relation characteristics

The notion of related actions defines the most elementary (abstract) characteristic of relations between activities. In order to model meaningful and realistic relations, additional characteristics have to be considered. We consider the following characteristics as essential in the modelling of relations between activities:

- *temporal ordering of action occurrences*. When two actions are related, their time moments are normally related by some ordering relation. Examples of common temporal ordering relations are sequential composition, arbitrary interleaving, synchronization and disabling;
- *references between action attributes*. The attribute values of an action may depend on the attribute values of other actions. For example, the result of action *receive* depends on the result of the previous *send* action in Figure 2.5, and the outcome of inspecting the balance of one's bank account depends on the amounts of previous payments and withdrawals;
- *unreliability and non-determinism of action occurrences*. Action occurrences may be uncertain due to probabilistic features, unreliable implementations, or non-deterministic action relations. Examples are data loss, as modelled by the non-occurrence of action *receive* in Figure 2.5, and the non-deterministic choice between two action occurrences which are related by an exclusion relation (choice).

Probability attribute

We introduce the probability attribute to model the probability, unreliability and non-determinism of action occurrences. The probability attribute defines, for each condition that

allows the occurrence of an action, the (conditional) probability that this action occurs when this condition is satisfied. For example, $\pi_{receive}(send)$ defines in Figure 2.5 the probability that action *receive* occurs when action *send* has occurred.

In principle, the probability of action occurrences may be considered as a characteristic of activities, in particular when activities depend on the satisfaction of a single (necessary) condition. However, we assume that an action may depend on multiple alternative conditions, such that the satisfaction of one alternative condition allows the occurrence of this action. Furthermore, we assume that the probability that an action occurs when one of its alternative conditions is satisfied may be different for each distinct alternative condition. Therefore, we model the probability of action occurrences as an attribute of the relations between actions and their conditions. Since we assume these conditions are satisfied by the occurrence or non-occurrence of other actions, we may also consider the probability attribute as an attribute of action relations.

Three types of probability attributes are distinguished:

- *the uncertainty attribute*, which defines the uncertainty that an action occurs. Two uncertainty values are distinguished:
  - the *must* uncertainty value, which defines that an action must occur when a certain condition is satisfied; and
  - the *may* uncertainty value, which defines that an action may or may not occur when a certain condition is satisfied;
- *the integral probability attribute*, which defines the integral probability that an action occurs when a certain condition is satisfied. This integral probability is defined by a real number in the range (0..1];
- *the stochastic probability attribute*, which defines a distribution of the probability that an action occurs over the time period when this action is enabled by a certain condition. This distribution is defined by a probability distribution function over time.

The above types of probability attributes constitute subsequent abstraction levels at which the probability of action occurrences can be defined. The uncertainty attribute is an abstraction of the integral probability attribute, since the must value is equal to the integral probability value 1 and the may value represents all integral probability values smaller than 1. The integral probability value is an abstraction of the stochastic probability attribute, since an integral probability value abstracts from the distribution of this value over time.

For example, consider the sequential composition of two actions *send* and *receive* representing the sending and subsequent delivery of a message. The following probability attributes model the unreliability of the delivery of a message at three related abstraction levels:

uncertainty attribute: $\upsilon_{receive}(send) = may$;
integral probability attribute: $\pi_{receive}(send) = 0.9$;
stochastic probability attribute: $\pi_{receive}(send, \tau) = 0.18 \times (\tau - \tau_{send})$,
with: $\tau_{send} < \tau \leq \tau_{send} + 5$.

In this example, the uncertainty attribute models the possible loss of a message, i.e., a behaviour execution exists in which the message is delivered and a behaviour execution exists in which the message is lost. The integral probability attribute defines that the message is lost with a probability of 0.1, i.e., from every 100 messages that are sent on the average 90 are delivered. The stochastic probability attribute models that the message is delivered in time interval $(\tau_{send}..\tau]$, i.e., $\tau_{send} < \tau_{receive} \leq \tau$, with probability $\pi_{receive}(send, \tau)$, where we assume the timing constraint that the message must be delivered within 5 time units after action *send* has occurred, such that $\tau_{send} < \tau \leq \tau_{send} + 5$. Figure 2.22 depicts the uniform probability distribution function $\pi_{receive}(send, \tau)$.

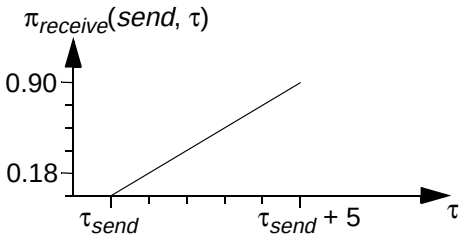


Figure 2.22: Uniform distribution function

We assume that the integral probability attribute may define multiple alternative integral probability values for each condition of an action, in order to allow one to model that the probability of an action occurrence lies within a certain range of acceptable probability values. For example, in the above example one may define that the probability of delivery is larger than 90 percent, i.e.: $\pi_{receive}(send) > 0.9$. A similar property holds for the stochastic probability attribute. For example, a probability distribution function may be parameterized, such as a poisson or exponential distribution function, allowing one to define multiple acceptable parameter values. Instead, we assume that the uncertainty attribute may only define a single uncertainty value for each condition of an action.

Differences between action attributes and probability attributes

Action attribute values (i.e., information, time and location attribute values) model characteristics of activities and probability attribute values model characteristics of relations between activities. The following essential difference between action attributes and probability attributes is related to this distinction:

- the information, time and location attributes model characteristics of individual behaviour executions; whereas
- the probability attributes model characteristics of sets of behaviour executions.

When an action occurs, a single information, time and location value are established. These action values model relevant characteristics of an individual action occurrence in an individual behaviour execution. Occurrences of this action in different behaviour executions may establish different information, time or location values.

Suppose a behaviour is executed a sufficiently large (infinite) number of times. The probability attribute defines for a particular condition of an action, the ratio between the number of times this condition is satisfied and this action occurs, and the total number of times this

condition is satisfied (and this action occurs or does not occur). Consequently, the probability attribute models a characteristic of the set of executions in which the condition is satisfied and in which the action occurs or does not occur.

In contrast to the action attributes, the probability attribute does not model a behaviour characteristic that can be prescribed for an individual behaviour execution. When considering individual behaviour executions one can only prescribe that an action occurs or does not occur.

2.5.3 Abstraction hierarchy

This section structures the characteristics of action relations in terms of an *abstraction hierarchy*. This abstraction hierarchy defines an ordering among the relevant characteristics of action relations, in which more general and abstract characteristics are placed higher in the hierarchy, and less general, more detailed characteristics are placed lower in the hierarchy. Each hierarchical level defines a limited set of characteristics, such that characteristics at lower hierarchical levels are refinements of characteristics at higher levels. For example, the characteristic of temporal ordering is defined at a lower hierarchical level than the notion of related actions as defined in Section 2.5.1, since action occurrences can only be ordered when they are related. Consequently, a temporal ordering relation is considered a refinement of the notion of related actions.

An abstraction hierarchy allows one to model an action relation at subsequent abstraction levels, since it defines the ordering in which one may add and remove characteristics. The refinement of a model of an action relation is performed by adding characteristics that are defined at lower levels in the abstraction hierarchy. The reverse process, i.e., abstracting from some characteristic of an action relation, is performed by removing this characteristic from the model. The removal of a characteristic at a certain level of the abstraction hierarchy implies the removal of characteristics that are defined as refinements of this characteristics at lower levels in the abstraction hierarchy.

Furthermore, each characteristic *chr* of an action relation defined at some hierarchical level is refined (as far as possible) into an orthogonal and complete set of (sub-)characteristics chr_1, .., chr_n at a subsequent level. For example, when considering the temporal ordering of two related actions *a* and *b*, this relation can be refined into the following three orthogonal (sub-)relations: *a* occurs before *b*, *a* and *b* occur at the same time, or *a* occurs after *b*.

The aim of an abstraction hierarchy is to facilitate a step-wise development of the concept of causality relation in the next chapters. This design concept should provide the conceptual means to model action relations. In the first development step the conceptual means are developed to model the most abstract characteristics of action relations. Subsequent steps refine these conceptual means to model more detailed characteristics of action relations. Furthermore, the abstraction hierarchy shows that the conceptual means for some characteristics can be developed independently.

Figure 2.23 depicts the top-level structure of the abstraction hierarchy. The hierarchy starts with actions and their elementary characteristics such as atomicity discussed in Section 2.4,

which is included because it represents a basic modelling assumption that is relevant to the modelling of independent actions and related actions. The subsequent hierarchical level depicts the distinction between the notions of independent and related actions as discussed in Section 2.5.1.

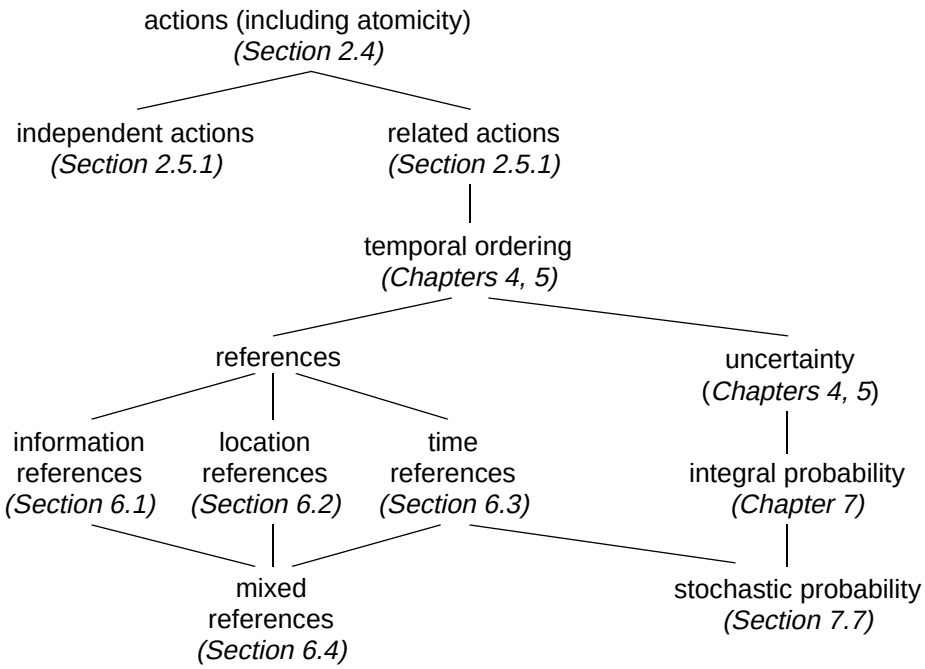


Figure 2.23: Abstraction hierarchy

The elaboration of the temporal ordering characteristic determines the possible conditions for the occurrences of actions. Therefore, probability attributes are modelled as refinements of temporal ordering relations. We (by definition) request that actions can not depend on future (non-)occurrences of actions, which implies that actions may not refer to action attributes of future action occurrences. Therefore, reference relations are also modelled as refinements of temporal ordering relations.

Information, location and time references can be modelled independently of each other, since they represent orthogonal characteristics of relations. Reference relations can be modelled independently of the uncertainty and integral probability attributes under certain conditions, i.e., they represent orthogonal characteristics of relations for certain behaviours (this subtlety is not represented in Figure 2.23). The stochastic probability attributes integrates characteristics of the integral probability attribute and characteristics of time reference relations. Mixed reference relations model references involving different types of action attributes.

Figure 2.23 indicates for each identified characteristic of action relations the chapters or sections in which this characteristic is elaborated.

2.5.4 Design modules

The abstraction hierarchy of Figure 2.23 allows one to develop a hierarchy of design modules. A *design module* is (part of) a design model which defines the necessary concepts and combination rules to model a particular (sub-)set of behaviour characteristics. Figure 2.24 depicts our module hierarchy.

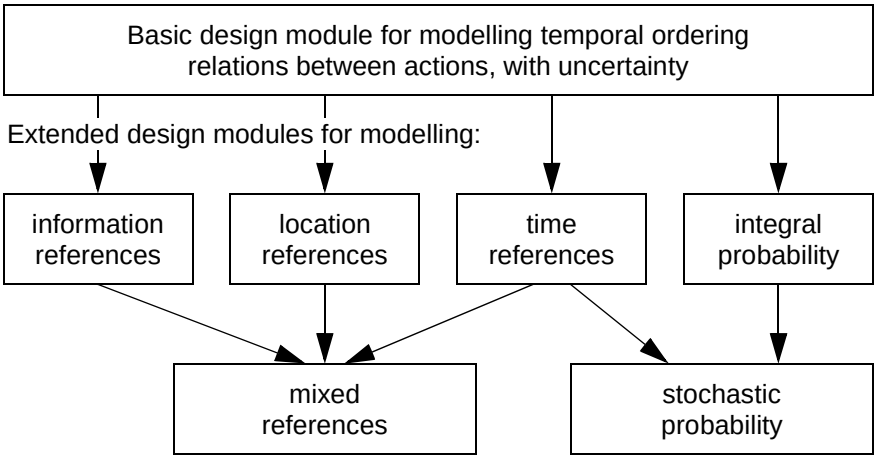


Figure 2.24: Hierarchy of design modules

The module hierarchy starts with a basic design module which allows the modelling of temporal ordering relations between actions without action attributes, but including the uncertainty attribute. We were forced to include the uncertainty attribute because designers should at least define whether an action must or may occur when one of its conditions is satisfied. This module provides support for the modelling of the characteristics that are defined at the three highest levels of the abstraction hierarchy in Figure 2.23, except for the “references” node.

The remainder of the hierarchy represents modular extensions of the basic design module with quantitative behaviour characteristics. Each subsequent module in the module hierarchy corresponds to subsequent nodes in the abstraction hierarchy, adding support for the modelling of the corresponding quantitative characteristics. The relationships between these modules is determined by the relationships between the corresponding characteristics in the abstraction hierarchy. Two types of relationships are distinguished:

- a module is an extension (refinement) of another module in case it provides support for the modelling of additional behaviour characteristics which are defined at (a) lower level(s) of the abstraction hierarchy. For example, the time references module is an extension of the basic module which provides additional support for the modelling of time references;
- two modules are independent (orthogonal) in case they provide support for the modelling of behaviour characteristics in different branches of the abstraction hierarchy. For example, the information and time references modules are independent because information references and time references are orthogonal characteristics which can be modelled independently.

The extension of design modules according to the abstraction hierarchy renders series of "backwards compatible" modules. Behaviour models defined in terms of the concepts of a particular design module have the same architectural semantics in design modules defined as extensions of this module. For example, a specification which only defines the temporal ordering of action occurrences has the same semantics in all design modules.

The union of independent design modules renders a design module which supports the modelling of the union of the characteristics of the individual modules. The abstraction hierarchy enforces that common characteristics are modelled identically across different modules while orthogonal characteristics can be modelled independently. The latter implies that the union of models of orthogonal characteristics is possible. For example, the time and integral probability modules can be combined in order to specify references between time attributes independently of the integral probability of action occurrences (under certain conditions). The definition of the probability of action occurrences in terms of distribution functions, however, requires the integration of time and probability. Therefore, the stochastic probability module is defined as an extension of the time and integral probability modules. Analogously, the mixed references module which supports the modelling of references between information, time and location attributes, is defined as an extension of the information, time and location modules.

2.6 Conclusions

This chapter introduces a limited set of basic design concepts that are necessary to design distributed systems. These concepts are structured into two related conceptual domains: the entity domain and the behaviour domain. The concepts identified in the entity domain are: entity, interaction point and action point. The concepts identified in the behaviour domain are: action, interaction and causality relation. These basic design concepts, and their composition rules, constitute our basic design model.

We explain the distinction between a design and its specification. A design is a conceptual model that is conceived and manipulated in a designer's mind. A specification is the symbolic representation of a design. A design notation must guarantee the unique representation of design concepts and their composition rules. The combination of a design model, a design notation and the architectural semantics of this design notation in terms of the design model, constitutes a design language.

A design language with a formal semantics is called a formal design language. A formal semantics adds precision, enables the use of mathematics to analyse and manipulate behaviours and provides a basis for the development of tool support. In this thesis, we develop a formal basic design language based on the (inter)action and causality relation concept.

In this chapter, we elaborate the action and interaction concepts in detail and we give a formal definition of these concepts. An action models the essential characteristics of a unit of activity that is performed by a single entity. We consider the following characteristics of an activity as essential: the result that is established by the activity, the moment at which the activity is finished and makes its total result available, and the location at which this result is made available. These characteristics are modelled by the information, time and location

attributes of an action, respectively. An interaction is considered a refinement of an action, which models how an activity is performed through the cooperation of multiple entities.

The causality relation concept is the basic building block for the modelling of action relations. This concept is developed in Chapters 4 to 7 of this thesis. In this chapter, we identify the essential characteristics of action relations and structure these characteristics in an abstraction hierarchy, to enable a systematic and modular development of the causality relation concept. These characteristics include the temporal ordering of action occurrences, references between the information, time and location attributes of ordered actions, and the unreliability or non-determinism of action occurrences. The latter characteristic is modelled using probability attributes.

Chapter 3

Execution model

This chapter introduces a so called ‘execution model’, which is used to define a behaviour by explicitly enumerating all possible executions of this behaviour. Because of its elementary, unambiguous and explicit character, the execution model is suitable for defining the formal semantics of action relations, the subject of this thesis. This forms the main reason for introducing the execution model.

The execution model, though intuitively appealing, lacks conciseness since the number of possible executions of a behaviour generally becomes large, and therefore unsurveyable, even for simple behaviours. Yet the execution model is suitable to provide insight in the number, complexity and variety of relations that can be defined between actions.

The structure of this chapter is as follows. Section 3.1 introduces and motivates the concept of execution and explains our approach towards the elaboration of this concept. Section 3.2 discusses the modelling of some elementary behaviour characteristics in terms of the execution model. Section 3.3 discusses the modelling of temporal ordering relations between action occurrences. Section 3.4 explains how the uncertainty attribute is modelled. Section 3.5 presents an overview of all possible temporal relations between two actions. Section 3.6 discusses two techniques for the constraint-oriented composition of behaviours with multiple actions. Section 3.7 discusses the modelling of action attribute values that are established in action occurrences, and the modelling of reference relations between attribute values of different action occurrences. Section 3.8 explains the modelling of the integral and stochastic probability attributes. Section 3.9 gives some formal definitions. Section 3.10 discusses related work. And Section 3.11 presents the conclusions.

3.1 Introduction

The primary purpose of the execution model is to define a formal semantics for causality relations. Causality relations are introduced in Chapter 4 to model action relations. The execution model is based on precise mathematical structures and operations on these structures. This allows one to define the formal interpretation of behaviours modelled by causality relations through a mapping of these causality relations onto their mathematical representation in terms of executions.

The execution model is also intended to provide a basis for the development of automated tool support for analysing and manipulating behaviour specifications, and to identify and structure the many different relations that can be defined between actions.

Execution concept

The execution model defines a set of concepts and combination rules that are necessary and sufficient to model behaviours in terms of their possible executions. The execution concept is central in this model. An *execution* represents the outcome of a possible run of a system performing a specified behaviour. This outcome comprises the actions that have occurred, the results that have been established in these actions, and how action occurrences are related in the particular execution.

Depending on its environment or on internal non-determinism, a specified behaviour may be executed in many different ways. Therefore, in the execution model, a behaviour is formally and completely defined by the set of *all* possible (but different) executions allowed by this behaviour. The term *execution-based behaviour definition* is used to explicitly denote a behaviour defined in terms of the execution model. The adjective 'execution-based' is omitted when it is clear from the context.

Figure 3.1 depicts a behaviour *B*, which is assumed to be specified in terms of causality relations, and its formal semantics in terms of execution-based behaviour *E*, which defines all possible executions of *B*. An execution is graphically represented by a box.

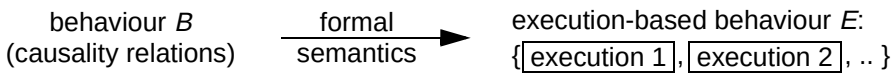


Figure 3.1: Formal semantics of behaviour *B*

Motivation

The main reasons for choosing this execution model to define the semantics of causality relations are the following:

- *intuition and understandability*: the execution concept is easy to understand and is close to the intuition of a designer, who is used to think in terms of the results that have to be established by some system behaviour during its execution;
- *modelling power*: the execution model allows one to formally represent all relevant characteristics of action relations, such as causality, independence, temporal relations, probability and (real) time constraints;
- *ease of formalization*: the behaviour characteristics modelled by the execution concept can be defined relatively straightforward in terms of the properties of common mathematical structures, such as sets, relations and first order logic;
- *compositional semantics*: the execution model allows one to formalize individual characteristics of action relations modelled by causality relations in a compositional way;
- *basis for tool support*: the execution model provides a basis for the development of

software tools in order to support the (semi-)automated analysis and manipulation of behaviour specifications.

Development of the execution concept

The abstraction hierarchy of Section 2.5.3 is used in this chapter to perform a step-wise development of the execution concept, since it provides a structure for understanding action relations. The execution concept should support the modelling of the characteristics of action relations that are identified in this abstraction hierarchy. By using this abstraction hierarchy we obtain definitions of the execution concept at multiple, but compatible, levels of detail.

3.2 Elementary behaviour characteristics

This section discusses the modelling of the elementary behaviour characteristics identified at the two most abstract levels of our abstraction hierarchy. These characteristics are modelled using the execution concept.

3.2.1 Atomicity of actions

The first behaviour characteristic of our abstraction hierarchy consists of the atomicity property of actions, which is discussed in Section 2.4.1. The atomicity property of an action imposes that an action either occurs or does not occur in an execution. This implies that the execution concept should allow one to define which actions occur in a particular execution of a behaviour and which actions do not occur in this execution.

Since it is possible that a specific action occurs or does not occur in an execution, the set of possible executions of a behaviour is divided into two disjoint subsets: a subset that contains all executions in which this action occurs and another subset that contains all executions in which this action does not occur. An action that does not occur in any execution is an impossible action. In case an action is made impossible by mistake, its conditions should be reconsidered or it should be discarded.

Figure 3.2 depicts the decomposition of the set of executions defined by an execution-based behaviour *E* into the subset consisting of all possible executions in which action *a* occurs, and the subset consisting of all possible executions in which *a* does not occur.

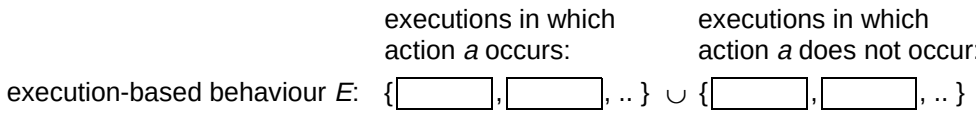


Figure 3.2: Decomposition of execution-based behaviour *E*

3.2.2 Independent and related actions

The second behaviour characteristic of the abstraction hierarchy defines that two actions *a* and *b* are either related or independent in an execution, which is discussed in Section 2.5.1.

The execution concept should allow one to model the effect of actions *a* and *b* being related or independent on the outcome of a possible run of a system that performs the behaviour containing actions *a* and *b*.

In case action *a* does not occur in an execution, we can not relate the outcome of the execution of *a* to the outcome of the execution of action *b*, since the execution of action *a* has not been realized. For example, we can not define the temporal relation between the non-occurrence of *a* and the occurrence of *b*, nor can the occurrence of *b* refer to the non-occurrence of *a*, since no result is established by *a*.

In case actions *a* and *b* occur in an execution, we should be able to explicitly express whether the outcome of the execution of *a* and the outcome of the execution of *b* are related or independent. For example, we want to be able to prescribe that the occurrences of *a* and *b* are temporally related (see Section 3.3), and want to be able to prescribe that the occurrence of *b* refers to the result established in the occurrence of *a* (see Section 3.7).

3.2.3 Formal definition

A behaviour *B* is defined in terms of actions and their causality relations in our design model. Behaviour *B* ranges over ***B*,** the domain of these behaviour definitions. Optionally, a behaviour definition *B* may be extended with a parameter which denotes the set of actions involved in this behaviour, e.g., $B(\{a, b, c\})$. This action set is called the *action domain* of a behaviour, and is denoted by the symbol *Ac*. The action domain of a particular behaviour *B* is denoted as Ac_B, or alternatively as $Ac(B)$.

The execution model is used to define the formal semantics of a behaviour *B* in terms of an execution-based behaviour *E* that defines all possible executions of *B*. Similar to behaviour *B*, the corresponding execution-based behaviour *E* may be extended with an action domain as a parameter, e.g., $E(\{a, b, c\})$. The action domain of *E* is denoted as Ac_E or $Ac(E)$.

The execution concept is formally defined below.

Definition 3.1 An execution $\chi \in E(Ac)$ is defined as a triple $\langle A, \overline{A}, Rel\rangle$, where

- $A \subseteq Ac$ is a set of action occurrences;
- $\overline{A} = Ac - A$ is a set of action non-occurrences;
- $Rel \subseteq A \times A$ is a relation between action occurrences. ■

We assume that each action defined in a behaviour $B(Ac)$ is also found in every execution of this behaviour, either as an occurrence or as a non-occurrence. Given that χ is an execution of $B(Ac)$, action set *A* represents the actions that occur and action set $\overline{A}$ represents the actions that do not occur in this execution, such that $A \cup \overline{A} = Ac$ and $A \cap \overline{A} = \varnothing$. Optionally, action set *Ac* may be defined as a parameter of an execution, e.g., $\chi(\{a, b, c\})$, which is called the action domain of an execution. The set of action occurrences *A*, the set of action non-occurrences $\overline{A}$ and the action domain *Ac* of χ are denoted as A_χ, $\overline{A}_\chi$, and Ac_χ, or alternatively as $A(\chi)$, $\overline{A}(\chi)$, and $Ac(\chi)$, respectively. In general, χ ranges over ***X***, the domain of executions.

Relation *Rel* represents that two action occurrences are related to each other in execution χ. Two action occurrences *a* and *b* are independent in χ when they are not related by relation *Rel*, i.e., $\langle a, b\rangle \notin Rel$. For example, in case $\chi = \langle\{a, b, c\}, \{d\}, \{\langle b, c\rangle\}\rangle$, action occurrences *b* and *c* are related, and action occurrence *a* is independent of action occurrences *b* and *c*. The relation *Rel* of χ is denoted as Rel_χ or $Rel(\chi)$.

Execution-based behaviours

An execution-based behaviour *E* represents a non-empty set of executions of some behaviour *B*. *E* ranges over $\boldsymbol{E}$, the domain of execution-based behaviour definitions, with $\boldsymbol{E} = \wp(\boldsymbol{X})\text{-}\{\varnothing\}$. Given a behaviour $B(Ac)$, we can determine the corresponding execution-based behaviour $E(Ac)$, such that the action domain of each execution in $E(Ac)$ is equal to *Ac*.

Definition 3.2 An execution-based behaviour $E(Ac) \in \boldsymbol{E}$ is defined as a non-empty set of executions having the same action domain *Ac*, such that $\forall\chi \in E \mid Ac_\chi = Ac$. ■

3.3 Temporal ordering relations

This section refines relation *Rel* in order to support the modelling of temporal ordering relations between the occurrences of related actions. A temporal ordering relation between two actions *a* and *b* defines the possible orderings of the time moments τ_a and τ_b at which these actions occur, respectively.

3.3.1 Modelling of temporal ordering

The following elementary temporal orderings of two related action occurrences *a* and *b* are distinguished:

- $a < b$, which defines that action *a* occurs before action *b* occurs, i.e., $\tau_a < \tau_b$;
- $a = b$, which defines that actions *a* and *b* occur at the same time, i.e., $\tau_a = \tau_b$;
- $a > b$, which defines that action *a* occurs after action *b* has occurred, i.e., $\tau_a > \tau_b$.

For any two action occurrences *a* and *b* in an execution, the above elementary temporal orderings are disjunctive, i.e., only one of them applies, and are complete, i.e., no other ordering is possible.

Therefore, two disjunctive temporal ordering relations are identified in order to define the temporal ordering of the action occurrences in a single execution:

- the ordering relation <, which defines that two action occurrences are ordered in time, i.e., $a < b \Leftrightarrow \tau_a < \tau_b$; and
- the synchronization relation =, which defines that two action occurrences happen at the same time, i.e., $a = b \Leftrightarrow \tau_a = \tau_b$.

Properties of < and =

The ordering relation < satisfies the following properties:

- *irreflexive*: no action occurrence can be ordered with itself, since it does not model a meaningful relation;
- *anti-symmetric*: two ordered action occurrences cannot be ordered as well in the opposite way;
- *transitive*: two action occurrences are ordered in case both occurrences are ordered via a common third action occurrence, i.e.: $a < b$ and $b < c$ implies $a < c$.

The synchronization relation = satisfies the following properties:

- *irreflexive*: the synchronization of an action occurrence with itself does not model a meaningful relation, and therefore it is not considered;
- *symmetric*: the synchronization of two action occurrences defines the same time conditions for both action occurrences;
- *transitive*: two action occurrences are synchronized in case both occurrences synchronize with a common third action occurrence, i.e.: $a = b$ and $b = c$ implies $a = c$.

The composition of the ordering relation < and the synchronization relation = is equal to the ordering relation <, and is commutative, such that: (i) $a < b$ and $b = c$ implies $a < c$ and (ii) $a = b$ and $b < c$ implies $a < c$. Consequently, the following properties hold: $< \bullet = \subset <$ and $= \bullet < \subset <$, where operator $\bullet$ represents the composition of relations.

Based on the above, we conclude that the synchronization of two action occurrences a and b implies that

- a inherits every relation of b with a third action occurrence c, such that:
  if $a = b$ and $b\ R\ c$ implies $a\ R\ c$, with R is < or R is =; and
- b inherits every relation of a with a third action occurrence c, such that:
  if $a = b$ and $a\ R\ c$ implies $b\ R\ c$, with R is < or R is =.

3.3.2 Formal definition

The refinement of relation *Rel* by means of temporal ordering relations < and = leads to the following redefinition of χ.

Definition 3.3 An execution $\chi \in E$ is defined as a four-tuple $\langle A, \overline{A}, <, = \rangle$, where

- A and $\overline{A}$ are defined as in Definition 3.1;
- $< \subseteq A \times A$ is the ordering relation between action occurrences;
- $= \subseteq A \times A$ is the synchronization relation between action occurrences;

such that relations < and = are disjoint, i.e., $< \cap =$ is empty. ■

The ordering and synchronization relations define a refinement of relation *Rel* introduced in Definition 3.1, such that *Rel* is $< \cup =$. Furthermore, these refinements are disjunctive (orthogonal), since two action occurrences are either related by $<$ or related by $=$, but not by both. Two action occurrences a and b are independent when they are not related by $<$ nor by $=$, i.e.: a and b being independent implies $\langle a, b\rangle \notin\ < \cup =$.

The ordering relation $<$ and the synchronization relation $=$ of some execution χ are briefly denoted as $<_\chi$ or $<(\chi)$, and $=_\chi$ or $=(\chi)$, respectively.

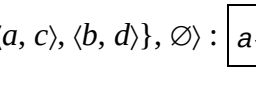
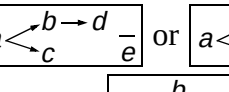
3.3.3 Graphical notation

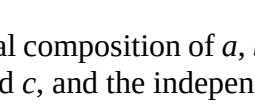
A graphical notation is introduced to represent executions in a more convenient and comprehensible way than the mathematical notation used so far. This notation consists of the following syntax rules:

1. an execution is represented by a rectangle-shaped box;
2. an action occurrence is represented by an action identifier within the box that represents the corresponding execution;
3. an action non-occurrence is either explicitly represented by an overlined action identifier within the box that represents the corresponding execution, or is implicitly represented by omitting this action identifier. In the latter case, the action domain of this execution should be clear from the context;
4. the ordering of two action occurrences $a < b$ is represented by an arc pointing from action occurrence a to action occurrence b. Transitive arcs are optional;
5. the synchronization of two action occurrences $a = b$ is represented by a double-line between both action occurrences. Transitive double-lines are optional;
6. the independence of two action occurrences is implicitly represented by the absence of a symbol linking both action occurrences.

In addition, the convention is adopted to arrange ordered action occurrences such that arrows point as much as possible from left to right.

The graphical representations of two example executions are given below:

$\chi_1 = \langle\{a, b, c, d\}, \{e\}, \{\langle a, b\rangle, \langle a, c\rangle, \langle b, d\rangle\}, \varnothing\rangle :$  or  ;

$\chi_2 = \langle\{a, b, c, d\}, \varnothing, \{\langle a, b\rangle, \langle a, c\rangle, \langle b, d\rangle, \langle c, d\rangle\}, \{\langle b, c\rangle\}\rangle :$  .

Execution χ_1 defines the sequential composition of a, b and d, the sequential composition of a and c, the independence of b and c, and the independence of c and d. The first representation of χ_1 explicitly denotes the non-occurrence of e. Execution χ_2 defines the sequential composition of a, b and d, the sequential composition of a, c and d, and the synchronization of b and c.

3.3.4 Alternative temporal ordering relations

More complex temporal ordering relations can be modelled by defining alternative executions with different temporal ordering relations. For example, the relation "*a* not before *b*" can be defined by two alternative executions: one execution which defines the ordering $a = b$ and another execution which defines the ordering $b < a$.

Figure 3.3 represents an execution-based behaviour, which models the sequential composition of action occurrence *a* and the interleaved occurrences of three actions *b*, *c* and *d*. Six executions are distinguished to represent the alternative orderings of *b*, *c* and *d*.

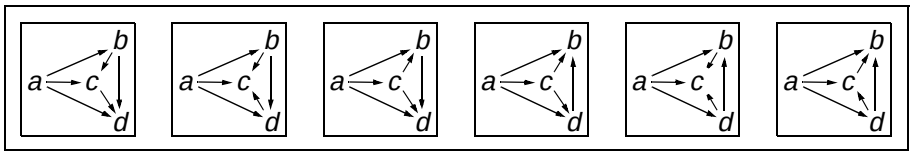


Figure 3.3: Alternative temporal orderings

The executions in Figure 3.3 are surrounded by a box representing the execution-based behaviour as a whole. The usage of this box is optional, and is only used in the sequel when confusion may arise w.r.t. which execution belongs to which behaviour, in case multiple behaviours are being represented in a single picture.

3.4 Uncertainty of actions

This section explains the modelling of the uncertainty characteristic (attribute) of actions in terms of executions.

In general, an action is allowed to occur when one or more conditions are satisfied. These conditions are modelled implicitly in the execution model by enumerating all possible executions that may result from the satisfaction or dissatisfaction of these conditions. The concept of causality relation is introduced in Chapter 4 to model conditions for the occurrence of actions explicitly. In this chapter, we abstract from the specific (types of) conditions that may be specified, and consider only the existence of such conditions.

The uncertainty attribute of an action defines for each condition that allows the occurrence of this action, whether this action *must* or *may* occur when this condition is satisfied. The uncertainty characteristic of actions is modelled as follows in the execution model:

- in case an action *must* occur when a certain condition is satisfied, this action occurs in all executions in which this condition is satisfied, i.e., no execution exists in which this action does not occur once this condition is satisfied;
- in case an action *may* occur when a certain condition is satisfied, the same executions are possible as when the action must occur and, in addition, for each of these executions an execution is possible in which this action does not occur (including the removal of all other action occurrences which depend on the occurrence of this action) provided that no other condition is satisfied due to which this action must occur.

For example, consider a behaviour of three actions *a*, *b* and *c*, in which *a* and *b* are initial actions, and *c* either depends on *a* and is independent of *b*, such that *c* is allowed to occur after *a* has occurred, or *c* depends on *b* and is independent of *a*, such that *c* is allowed to occur after *b* has occurred. Figure 3.4 (i) depicts the two possible executions, assuming that *a* and *b* must occur, and *c* must occur when one, or both, of its conditions are satisfied. Figure 3.4(ii) depicts all possible executions when assuming that *a* and *b* may occur, and *c* may occur when one, or both of its conditions are satisfied. In addition, Figure 3.4(ii) depicts two executions in grey, which become impossible when we define that *c* must occur when *a* has occurred and may occur when *b* has occurred.

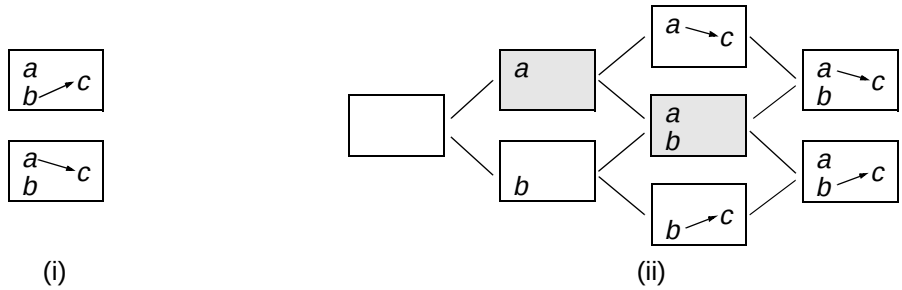


Figure 3.4: Modelling of uncertainty

3.5 Behaviours with two actions

Based on the behaviour characteristics considered so far, any execution-based behaviour with two actions *a* and *b* contains one or more of the following executions: [a b], [a→b], [b→a], [a=b], [a], [b], [ ]. This renders 2^7-1 = 127 different behaviours.

These behaviours can be structured into the following three categories:

- behaviours in which actions *a* and *b* are statically independent;
- behaviours in which actions *a* and *b* are statically related;
- behaviours in which actions *a* and *b* are dynamically related (or independent).

This section gives an overview of the behaviours in these categories and structures them in terms of so called behaviour families.

3.5.1 Statically independent actions

The execution-based behaviours with two statically independent actions *a* and *b* consist of:

- execution [a b], which is always possible due to the assumption that actions *a* and *b* can occur in at least one execution and the characteristic that actions *a* and *b* are independent. In other words, the occurrence of *a* can not prevent the occurrence of *b*, nor vice versa; and
- optionally, execution [a], execution [b], or executions [a], [b] and [ ], which model the uncertainty of the occurrence of *a*, the uncertainty of the occurrence of *b*, and the uncertainty of the occurrences of *a* and *b*, respectively.

The above renders 4 different behaviours of two statically independent actions *a* and *b*. Table 3.5 depicts these behaviours in terms of an *execution table*. Each row of this table represents a behaviour by indicating the executions that are possible using the symbol '1' and the executions that are impossible using the symbol '-'.

Table 3.5: Behaviours of two statically independent actions

independence of *a* and *b*	$\boxed{a \quad b}$	$\boxed{a \rightarrow b}$	$\boxed{b \rightarrow a}$	$\boxed{a = b}$	$\boxed{a \quad\quad}$	$\boxed{\quad\quad b}$	$\boxed{\quad\quad}$
must *a* and must *b*	1	-	-	-	-	-	-
must *a* and may *b*	1	-	-	-	1	-	-
may *a* and must *b*	1	-	-	-	-	1	-
may *a* and may *b*	1	-	-	-	1	1	1

3.5.2 Statically related actions

The execution-based behaviours with two statically related actions consist of one or more of the executions $\boxed{a \rightarrow b}$, $\boxed{b \rightarrow a}$, $\boxed{a = b}$, $\boxed{a \quad\quad}$, $\boxed{b \quad\quad}$, $\boxed{\quad\quad}$. The exclusion of execution $\boxed{a \quad b}$ implies that actions *a* and *b* are related in all possible executions of these behaviours.

The above renders 2^6-1 = 63 different behaviours of statically related actions. These behaviours can be structured into *behaviour families*, each family consisting of two or more behaviours which share the same behaviour characteristics. These common characteristics are represented in terms of the execution model by executions that are possible for all family members, and executions that are impossible for all family members.

In general, executions that are possible for every member of a behaviour family are called *mandatory executions*. Executions that are impossible for every family member are called *forbidden executions*. Executions that are possible for some (but not all) family members are called *optional executions*. The union of the mandatory and forbidden executions of some behaviour family are also called *characteristic executions*, since they characterize this family.

Table 3.6 depicts a possible structuring of all 63 behaviours of statically related actions in terms of behaviour families. Each row of this execution table represents a behaviour family by indicating its mandatory executions using the symbol '1', its forbidden executions using the symbol '-', and its optional executions using the symbol 'x'. Furthermore, characteristic executions are depicted in **bold** font.

The identification of the behaviour families in Table 3.6 is based on the identification of architecturally meaningful relations, such as enabling, disabling, synchronization and combinations thereof. The mandatory executions of a behaviour family represent the family member with minimal uncertainty, while the optional executions in other family members represent the addition of uncertainty w.r.t. the occurrence of *a* or *b*.

For example, behaviour family "interleaving of *a* and *b*" consists of eight family members:
{$\boxed{a \rightarrow b}$, $\boxed{b \rightarrow a}$}, {$\boxed{a \rightarrow b}$, $\boxed{b \rightarrow a}$, $\boxed{\quad\quad}$},

Table 3.6: Overview of behaviour families of two statically related actions

behaviour family	$\boxed{a \quad b}$	$\boxed{a \rightarrow b}$	$\boxed{b \rightarrow a}$	$\boxed{a = b}$	$\boxed{a \quad\quad}$	$\boxed{\quad\quad b}$	$\boxed{\quad\quad}$
a enables *b*	-	**1**	-	-	x	-	x
b enables *a*	-	-	**1**	-	-	x	x
synchronization of *a* and *b*	-	-	-	**1**	-	-	x
choice between *a* and *b*	-	-	-	-	**1**	**1**	x
a disables *b*	-	-	**1**	-	**1**	x	x
b disables *a*	-	**1**	-	-	x	**1**	x
interleaving of *a* and *b*	-	**1**	**1**	-	x	x	x
sync-choice of *a* and *b*	-	-	-	**1**	**1**	**1**	x
a sync-excludes *b*	-	-	-	**1**	**1**	-	x
b sync-excludes *a*	-	-	-	**1**	-	**1**	x
a sync-enables *b*	-	**1**	-	**1**	x	-	x
b sync-enables *a*	-	-	**1**	**1**	-	x	x
a sync-disables *b*	-	-	**1**	**1**	**1**	x	x
b sync-disables *a*	-	**1**	-	**1**	x	**1**	x
temporal freedom of *a* and *b*	-	**1**	**1**	**1**	x	x	x
a excludes *b*	-	-	-	-	**1**	-	x
b excludes *a*	-	-	-	-	-	**1**	x
deadlock	-	-	-	-	-	-	**1**

$\{\boxed{a \rightarrow b}, \boxed{b \rightarrow a}, \boxed{a \quad\quad}\}$, $\{\boxed{a \rightarrow b}, \boxed{b \rightarrow a}, \boxed{a \quad\quad}, \boxed{\quad\quad}\}$,
$\{\boxed{a \rightarrow b}, \boxed{b \rightarrow a}, \boxed{\quad\quad b}\}$, $\{\boxed{a \rightarrow b}, \boxed{b \rightarrow a}, \boxed{\quad\quad b}, \boxed{\quad\quad}\}$,
$\{\boxed{a \rightarrow b}, \boxed{b \rightarrow a}, \boxed{a \quad\quad}, \boxed{\quad\quad b}\}$, and $\{\boxed{a \rightarrow b}, \boxed{b \rightarrow a}, \boxed{a \quad\quad}, \boxed{\quad\quad b}, \boxed{\quad\quad}\}$.

This family is characterized by the mandatory executions $\boxed{a \rightarrow b}$ and $\boxed{b \rightarrow a}$, and the forbidden executions $\boxed{a \quad b}$ and $\boxed{a = b}$. Executions $\boxed{a \quad\quad}$, $\boxed{\quad\quad b}$, $\boxed{\quad\quad}$ are optional executions. The mandatory executions model the interleaving of the occurrences of *a* and *b*. The optional executions model the uncertainty of the occurrences of *a* and *b*.

Behaviour family "*a* disables *b*" shows how disabling relations are modelled in the execution model. The mandatory executions $\boxed{b \rightarrow a}$ and $\boxed{a \quad\quad}$ model the disabling of the occurrence of *b* by the occurrence of *a*. The occurrence of *b* is uncertain due to its possible disabling by *a*. The optional executions $\boxed{\quad\quad b}$ and $\boxed{\quad\quad}$ model the uncertainty of the occurrence of *a* and additional uncertainty of the occurrence of *b*, respectively.

The behaviour families in the last three rows define one or more impossible actions, which violates our assumption that an action should occur in at least one execution. In general, the

definition of impossible actions should be considered as a design error or implies that these actions are irrelevant (redundant) and can be removed from the behaviour definition.

Figure 3.7 depicts a hierarchy among the behaviour families that are identified in Table 3.6. This hierarchy is based on the characteristic executions these behaviour families have in common. In order to group some families, ‘pseudo’ families are introduced which are depicted in *oblique* font. For readability, the symbol ‘0’ is used instead of the symbol ‘-’. The hierarchy is built by replacing ‘x’ symbols by ‘0’ or ‘1’ symbols, in subsequent steps. The solid lines represent these replacements. For brevity, the impossibility of execution $\boxed{a \quad b}$ is not indicated, i.e., the six symbols denote executions $\boxed{a \rightarrow b}$, $\boxed{b \rightarrow a}$, $\boxed{a = b}$, $\boxed{a \quad\quad}$, $\boxed{b \quad\quad}$ and $\boxed{\quad\quad}$, respectively.

3.5.3 Dynamically related actions

In case of dynamically related actions, there exists at least one execution in which actions *a* and *b* are related and one execution in which *a* and *b* are independent. The execution sets of these behaviours consist of:

- execution $\boxed{a \quad b}$, which is always possible; and
- one or more of the executions $\boxed{a \rightarrow b}$, $\boxed{b \rightarrow a}$, $\boxed{a = b}$, $\boxed{a \quad\quad}$, $\boxed{b \quad\quad}$, $\boxed{\quad\quad}$.

This renders 2^6-1 = 63 different behaviours of two dynamically related actions. These behaviours correspond to the behaviours of Table 3.6, extended with execution $\boxed{a \quad b}$. For example, behaviour {$\boxed{a \quad b}$, $\boxed{a \rightarrow b}$} represents that actions *a* and *b* occur independently, or actions *a* and *b* are related, such that *a* occurs before *b*.

The 63 behaviours mentioned above also comprise the latter three behaviours of Table 3.5. Therefore, an alternative interpretation of these behaviours is that execution $\boxed{a \quad\quad}$, execution $\boxed{b \quad\quad}$, or executions $\boxed{a \quad\quad}$, $\boxed{b \quad\quad}$ and $\boxed{\quad\quad}$ may result from three distinct relations between actions *a* and *b* in which the occurrence of *b*, the occurrence of *a*, or the occurrence of *a* or *b* is forbidden, respectively. For example, the fourth behaviour of Table 3.5 may be considered as a behaviour in which actions *a* and *b* are independent in some executions, and are related by a choice relation in the other executions.

The above implies that the execution model is not able to distinguish behaviours in which actions *a* and *b* are independent in every execution, from behaviours in which *a* and *b* are independent in some executions and are related by one of the three relations mentioned above in the other executions. The reason for this is that executions $\boxed{a \quad\quad}$, $\boxed{b \quad\quad}$ and $\boxed{\quad\quad}$ do not represent whether actions *a* and *b* are related or independent. For example, execution $\boxed{a \quad\quad}$ may result from the independence of actions *a* and *b* with the occurrence of *b* being uncertain, or from a relation between *a* and *b* in which the occurrence of *b* is uncertain or in which the occurrence of *b* is disabled by the occurrence of *a*. Consequently, there exists no bijection from the domain of behaviours *B* to the domain of execution-based behaviours *E*.

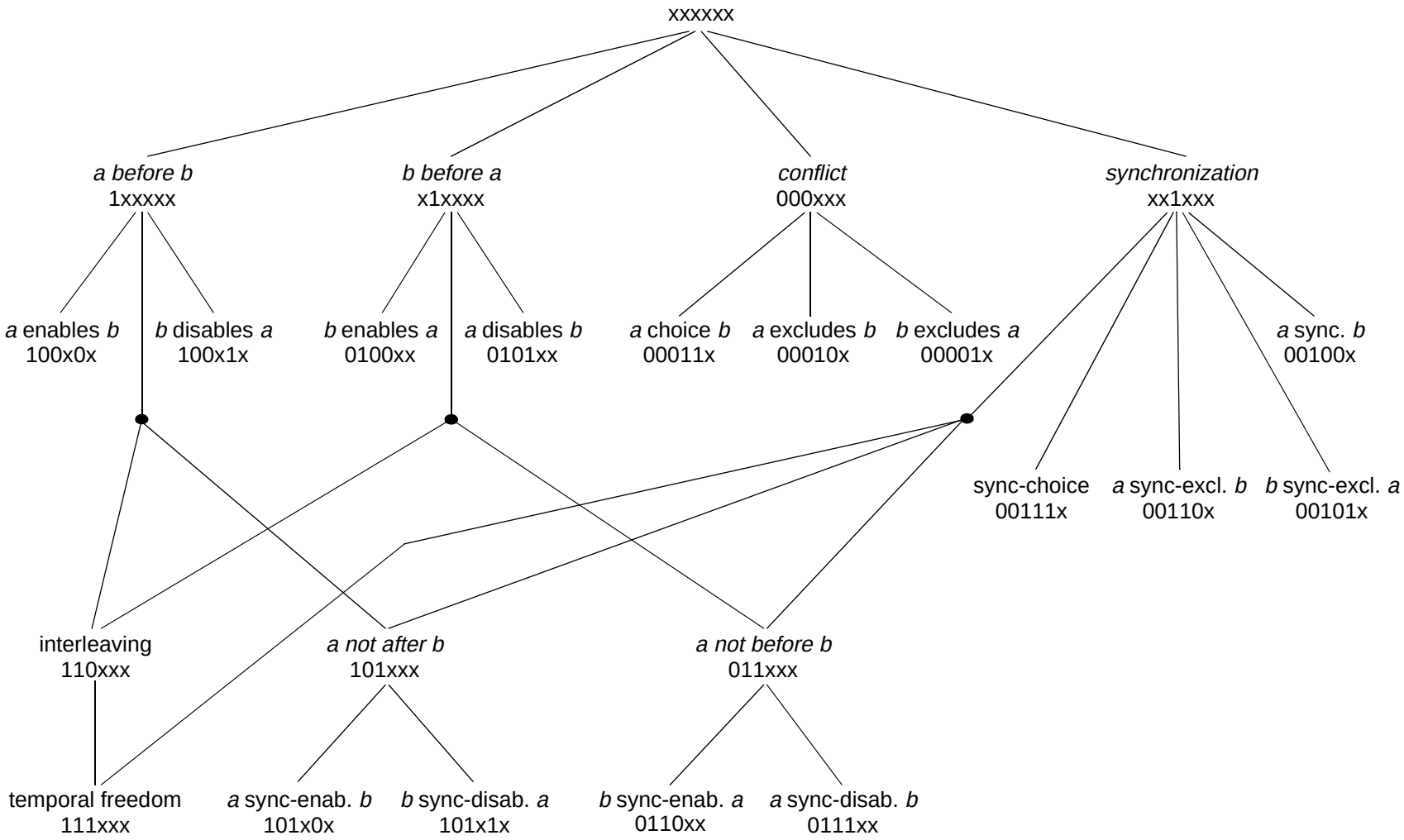


Figure 3.7: Family tree of temporal relations between two actions

3.6 Behaviours with multiple actions

In general, the enumeration of all possible executions of a behaviour $B(Ac)$ becomes lengthy and unsurveyable for multiple actions Ac. This section presents two techniques to compose an execution-based behaviour $E(Ac)$ from multiple smaller execution-based behaviours involving a subset of Ac, which enable one to structure $E(Ac)$ and improve its comprehensibility. Both techniques are denoted as *constraint-oriented* composition techniques, since they consider executions as defining constraints on the execution of the involved actions. In the next chapters, these techniques are used to define the formal semantics of a behaviour $B(Ac)$ in a compositional way.

3.6.1 Executions modelling constraints

An execution can be considered as defining constraints on the execution of the involved actions. For example, the execution

- $\boxed{a \quad b}$, defines that actions a and b must occur, and the occurrences of a and b must be independent;
- $\boxed{a \rightarrow b}$, defines that actions a and b must occur, and the occurrences of a and b must be related such that a occurs before b;
- $\boxed{a = b}$, defines that actions a and b must occur, and the occurrences of a and b must be related such that a and b occur at the same time;
- $\boxed{a \quad \bar{b}}$, defines that action a must occur and action b must not occur;
- $\boxed{\bar{a} \quad \bar{b}}$, defines that actions a and b must not occur.

Both independence and the temporal ordering relations < and = define constraints on the execution of the involved action occurrences. Independence prescribes the absence of a relation between action occurrences, while the temporal ordering relations prescribe a certain ordering of action occurrences.

Alternative constraints

According to the above, an execution-based behaviour can be considered as defining alternative constraints on the execution of the involved actions. For example, behaviour $E(\{a, b\}) = \{\boxed{a \quad \bar{b}}, \boxed{\bar{a} \quad b}\}$ defines that either action a must occur and b must not occur, or action b must occur and action a must not occur.

3.6.2 Free behaviours

Based on the interpretation of an execution-based behaviour $E(Ac)$ as a set of alternative constraints on the execution of the actions in Ac, a behaviour can be defined that does not impose any constraint on the execution of the actions in Ac, in the sense that it allows maximal freedom in the execution of these actions. Behaviours that satisfy this property are behaviours which allow all possible executions of the involved actions. These behaviours are called *free behaviours*, and are denoted as $E\text{-}Free(Ac)$.

Behaviour *E-Free*(*Ac*) consists of all possible executions χ, such that:

- the union of the set of action occurrences of χ and the set of action non-occurrences of χ is equal to action set *Ac*, i.e., $Ac = A_\chi \cup \overline{A}_\chi$;
- the ordering and synchronization relations of χ are non-overlapping subsets of $A \times A$, i.e., $<_\chi \subseteq A \times A$, $=_\chi \subseteq A \times A$, and $<_\chi \cap =_\chi = \emptyset$.

For example, behaviours *E-Free*({*a*}) and *E-Free*({*a*, *b*}) are defined as follows:

- *E-Free*({*a*}) = {$\boxed{a}$, $\boxed{\overline{a}}$};
- *E-Free*({*a*, *b*}) = {$\boxed{a\ \ b}$, $\boxed{a \rightarrow b}$, $\boxed{b \rightarrow a}$, $\boxed{a = b}$, $\boxed{a\ \ \overline{b}}$, $\boxed{\overline{a}\ \ b}$, $\boxed{\overline{a}\ \ \overline{b}}$}.

The above execution sets do not impose any constraints on the execution of action *a*, and the execution of actions *a* and *b*, from the perspective of the entire behaviour, respectively. In other words, the alternative constraints represented by behaviours *E-Free*(*Ac*) allow maximal freedom in the execution of the actions in *Ac*.

Formal definition

The behaviour *E-Free*(*Ac*) for an action domain *Ac* is formally defined below.

Definition 3.4 Behaviour *E-Free*(*Ac*) is defined as:

$$E\text{-}Free(Ac) = \{\langle A, \overline{A}, <, =\rangle \mid A \cup \overline{A} = Ac,\ A \cap \overline{A} = \emptyset,\ < \subseteq A \times A,\ = \subseteq A \times A,\ <_\chi \cap =_\chi = \emptyset\}. \blacksquare$$

3.6.3 Constraint-oriented composition (1)

This section presents the first constraint-oriented composition technique, which defines an execution-based behaviour *E*(*Ac*) as the restriction of free behaviour *E-Free*(*Ac*) by multiple smaller behaviours $E_i(Ac_i)$, with $Ac_i \subset Ac$, $1 \le i \le n$ and $n \ge 2$, each of which defines constraints on the execution of a subset of the actions in *Ac*. The restriction operation enforces that *E*(*Ac*) represents the conjunction of the constraints represented by all $E_i(Ac_i)$. The restriction operation is defined below.

Restriction operator

The restriction of a behaviour *E*(*Ac*) by a behaviour *E*'(*Ac*'), denoted as *E*(*Ac*) \ *E*'(*Ac*'), with $Ac' \subseteq Ac$, renders a behaviour which consists of the executions from *E*(*Ac*) that satisfy the constraints represented by an execution from *E*'(*Ac*'). An execution χ from *E*(*Ac*) satisfies the constraints represented by an execution χ' from *E*'(*Ac*') if all action occurrences and action non-occurrences in χ' are contained in χ, and all action occurrences of χ' are independent or related in the same way in χ and χ'. In this case, χ' is called a *sub-execution* of χ.

Consequently, *E*(*Ac*) \ *E*'(*Ac*') is defined as the subset of *E*(*Ac*) that contains all executions $\chi \in E(Ac)$ for which an execution $\chi' \in E'(Ac')$ that is a sub-execution of χ exists.

For example, the restriction of behaviour *E*({*a*, *b*, *c*}) = {$\boxed{a \leftarrow b\ \ c}$, $\boxed{a\ \ b \rightarrow c}$, $\boxed{a \leftarrow b\ \ \overline{c}}$, $\boxed{\overline{a}\ \ b \rightarrow c}$, $\boxed{\overline{a}\ \ b\ \ \overline{c}}$, $\boxed{\overline{a}\ \ \overline{b}\ \ \overline{c}}$} by behaviour *E*'({*a*, *b*}) = {$\boxed{a\ \ b}$,

$\boxed{a \quad \bar{b}}$, $\boxed{\bar{a} \quad b}$} results in behaviour $E(\{a, b, c\}) \setminus E'(\{a, b\}) = \{\boxed{a \quad b \rightarrow c}, \boxed{\bar{a} \quad b \rightarrow c}, \boxed{\bar{a} \quad b \quad \bar{c}}\}$.

The restriction of a behaviour $E(Ac)$ by multiple smaller behaviours $E_i(Ac_i)$, with $Ac_i \subset Ac$, $1 \le i \le n$ and $n \ge 2$, is obtained by the repeated application of the restriction operator. The restriction operator is associative, i.e., the order in which behaviour $E(Ac)$ is restricted by these behaviours is irrelevant, such that: $(E(Ac) \setminus E_1(Ac_1)) \setminus E_2(Ac_2) = (E(Ac) \setminus E_2(Ac_2)) \setminus E_1(Ac_1)$. Therefore, the restriction operator can be generalized towards the restriction of a behaviour $E(Ac)$ by a set of behaviours, which is represented by the symbol \\, e.g., $E(Ac) \setminus\setminus \{E_i(Ac_i) \mid i \}$.

For example, the restriction of behaviour $E(\{a, b, c\}) = \{\boxed{a \leftarrow b \quad c}, \boxed{a \quad b \rightarrow c}, \boxed{a \leftarrow b \quad c}, \boxed{\bar{a} \quad b \rightarrow c}, \boxed{\bar{a} \quad b \quad \bar{c}}, \boxed{\bar{a} \quad b \quad \bar{c}}\}$ by behaviours $E_1(\{a, b\}) = \{\boxed{a \quad b}, \boxed{a \quad \bar{b}}, \boxed{\bar{a} \quad b}\}$ and $E_2(\{b, c\}) = \{\boxed{b \rightarrow c}\}$ results in behaviour $E(\{a, b, c\}) \setminus\setminus \{E_1(\{a, b\}), E_2(\{b, c\})\} = \{\boxed{a \quad b \rightarrow c}, \boxed{\bar{a} \quad b \rightarrow c}\}$.

Restriction of free behaviour

Based on the restriction operator, the conjunction of the constraints as represented by multiple behaviours $E_i(Ac_i)$, with $1 \le i \le n$ and $n \ge 2$, is defined as the restriction of free behaviour *E-Free*(Ac), with $Ac = \cup_i Ac_i$, by these behaviours. The resulting restricted behaviour $E(Ac)$ = *E-Free*(Ac) $\setminus\setminus \{E_i(Ac_i) \mid i \}$ consists of all executions χ in *E-Free*(Ac) for which an execution χ_i can be found in each behaviour $E_i(Ac_i)$, such that χ_i is a sub-execution of χ. In other words, each execution χ in $E(Ac)$ represents the conjunction of the constraints represented by an execution χ_i from each behaviour $E_i(Ac_i)$. Therefore, $E(Ac)$ represents the conjunction of the constraints represented by all $E_i(Ac_i)$.

For example, the conjunction of behaviours $E_1(\{a, b\}) = \{\boxed{b \rightarrow a}\}$ and $E_2(\{b, c\}) = \{\boxed{b \rightarrow c}\}$ is equal to *E-Free*($\{a, b, c\}$) $\setminus\setminus \{E_1(\{a, b\}), E_2(\{b, c\})\}$ =

$\{ \boxed{b \begin{smallmatrix} \nearrow a \\ \searrow c \end{smallmatrix}}, \boxed{b \begin{smallmatrix} \nearrow a \\ \downarrow \\ \searrow c \end{smallmatrix}}, \boxed{b \begin{smallmatrix} \nearrow a \\ \uparrow \\ \searrow c \end{smallmatrix}}, \boxed{b \begin{smallmatrix} \nearrow a \\ \Downarrow \\ \searrow c \end{smallmatrix}} \}$.

Constraint-oriented composition of behaviours

In general, a behaviour $E(Ac)$ can be composed in a constraint-oriented way as the conjunction of multiple smaller behaviours $E_i(Ac_i)$, with $1 \le i \le n$ and $n \ge 2$, using the restriction operator as presented above. The following *completeness rules* must be obeyed to obtain a complete constraint-oriented definition of $E(Ac)$:

1. each action of $E(Ac)$ should be defined by at least one of the behaviours $E_i(Ac_i)$, such that $Ac = \cup_i Ac_i$;
2. the independence or relation between each pair of actions in Ac should be defined by one of the behaviours $E_i(Ac_i)$ or by the composition of two (or more) of the behaviours $E_i(Ac_i)$.

The need for the second rule is illustrated by the conjunction of $E_1(\{a, b\}) = \{\boxed{b \rightarrow a}\}$ and $E_2(\{b, c\}) = \{\boxed{b \rightarrow c}\}$. The independence or relation between actions b and c is not defined

in the behaviours E_1 and E_2, nor by the composition of E_1 and E_2, which implies that both actions can occur independently, or related as $b < c$, $c < b$ and $b = c$, in alternative behaviour executions. Even in case this is desired, these alternatives should be defined explicitly by E_1 or E_2, or by an additional behaviour E_3.

The following variant of the previous example obeys the second rule. The conjunction of behaviours $E_1(\{a, b\}) = \{\boxed{a \rightarrow b}\}$ and $E_2(\{b, c\}) = \{\boxed{b \rightarrow c}\}$ is equal to $E\text{-}Free(\{a, b, c\}) \setminus \{E_1(\{a, b\}), E_2(\{b, c\})\} = \{\boxed{a \rightarrow b \rightarrow c}\}$. In this case the relation between b and c is defined implicitly by the composition of E_1 and E_2, due to the transitivity of relation $<$.

Formal definition

The notion of sub-execution is formally defined by means of relation $\sqsubseteq$, such that $\chi_1 \sqsubseteq \chi_2$ renders *true* if χ_1 is a sub-execution of χ_2, and renders *false* otherwise.

Definition 3.5 Relation $\sqsubseteq : \boldsymbol{X} \times \boldsymbol{X}$ is defined as

$$\langle A_1, \overline{A}_1, <_1, =_1\rangle \sqsubseteq \langle A_2, \overline{A}_2, <_2, =_2\rangle \Leftrightarrow \quad A_1 \subseteq A_2 \ \wedge \ \overline{A}_1 \subseteq \overline{A}_2 \ \wedge \ <_1 = <_2 \cap (A_1 \times A_1) \ \wedge \ =_1 = =_2 \cap (A_1 \times A_1) \ . \quad \blacksquare$$

The restriction operator \ is formally defined below.

Definition 3.6 The restriction operator $\setminus : \wp(\boldsymbol{X}) \times \wp(\boldsymbol{X}) \rightarrow \wp(\boldsymbol{X})$ is defined as

$$E \setminus E' = \{ \chi \mid \chi \in E, \exists \chi' \in E' \mid \chi' \sqsubseteq \chi \} \ . \quad \blacksquare$$

The associativity property of the restriction operator is proven below.

Property 3.7 $(E_1 \setminus E_2) \setminus E_3 = (E_1 \setminus E_2) \cap (E_1 \setminus E_3)$.

Proof:

$$\begin{aligned}(E_1 \setminus E_2) \setminus E_3 &= \{ \chi_1 \mid \chi_1 \in E_1, \exists \chi_2 \in E_2, \chi_2 \sqsubseteq \chi_1 \} \setminus E_3 \\ &= \{ \chi_1 \mid \chi_1 \in E_1, \exists \chi_2 \in E_2, \chi_3 \in E_3 \mid \chi_2 \sqsubseteq \chi_1, \chi_3 \sqsubseteq \chi_1 \} \\ &= \{ \chi_1 \mid \chi_1 \in E_1, \exists \chi_2 \in E_2, \chi_2 \sqsubseteq \chi_1 \} \cap \{ \chi_1 \mid \chi_1 \in E_1, \exists \chi_3 \in E_3, \chi_3 \sqsubseteq \chi_1 \} \\ &= (E_1 \setminus E_2) \cap (E_1 \setminus E_3) \ . \quad \blacksquare\end{aligned}$$

Property 3.8 $(E_1 \setminus E_2) \setminus E_3 = (E_1 \setminus E_3) \setminus E_2$.

Proof: Follows from Property 3.7 and the commutativity of the $\cap$. ■

The generalized restriction operator \\ is formally defined using Property 3.7.

Definition 3.9 The restriction operator $\setminus\setminus : \wp(\boldsymbol{X}) \times \wp(\wp(\boldsymbol{X})) \rightarrow \wp(\boldsymbol{X})$ is defined as

$$E \setminus\setminus \{E_i \mid i \} = \cap\{E \setminus E_i \mid i \} \ . \quad \blacksquare$$

3.6.4 Partial executions

A drawback of the technique for defining behaviours in a constraint-oriented way presented in Section 3.6.3 is that the number of executions of a free behaviour grows exponentially

with the number of actions. Free behaviours of more than three actions are already too large to be manipulated within a reasonable amount of time (or space).

The reason for the use of free behaviours is that the conjunction of multiple constraints (executions) generally can not be defined directly in case these constraints involve different actions. For example, the conjunction of constraints $\boxed{b \rightarrow a}$ and $\boxed{b \rightarrow c}$ is not equal to constraint $\boxed{a \leftarrow b \rightarrow c}$, since this constraint defines b and c to be independent whereas the original constraints neither impose the independent nor the related occurrences of b and c. We are forced to represent the absence of a constraint between b and c by considering all possible constraints between b and c, i.e., by considering all possible executions in which b and c are independent or related by < or =. These executions can be obtained by the restriction of the free behaviour of actions a, b and c, i.e., $E\text{-}Free(\{a, b, c\}) \setminus\setminus \{\{\boxed{b \rightarrow a}\}, \{\boxed{b \rightarrow c}\}\}$, as is illustrated by one of the examples in Section 3.6.3.

The limitation above is caused by the implicit definition of independence in executions. Two action occurrences are defined to be independent when they are neither related by < nor by =. Due to this choice we are not able to define that the independence or relation between two actions is ‘yet undefined’. The possibility the leave the independence or relation between two actions undefined in an execution, would however facilitate the manipulation of executions, and make the impractical use of free behaviours unnecessary in the constraint-oriented definition of execution-based behaviours (see Section 3.6.5).

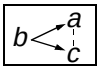
In order to simplify the definition of execution-based behaviours in terms of constraints, we introduce the *partial execution* concept. A partial execution is a (regular) execution extended with an independence relation | between action occurrences, i.e., $| \subseteq \boldsymbol{A} \times \boldsymbol{A}$. This relation is used to explicitly define the independence of two action occurrences. The independence relation is irreflexive and symmetric, but is not transitive. The adjective ‘regular’ is used to explicitly denote executions as defined in Definition 3.3.

A partial execution is a regular execution when each pair of action occurrences in the partial execution is either related by <, = or |. However, we allow the independence or relation between two action occurrences to be undefined in a partial execution, which motivated its name. In that case, a partial execution is not a regular execution. Consequently, the domain of partial executions comprises the domain of regular executions $\boldsymbol{X}$, but not vice versa. In the sequel, the term execution is used to denote regular executions and partial executions, when no confusion is possible.

Graphical notation

Partial executions are represented in the same way as executions, except for action occurrences which are neither related by <, =, nor |. These action occurrences are connected by dotted lines.

For example, the conjunction of the constraints $\boxed{b \rightarrow a}$ and $\boxed{b \rightarrow c}$ is represented by the partial execution:



.

Formal definition

The notion of partial execution is formally defined below.

Definition 3.10 A partial execution $e\chi$ is defined as a five-tuple $\langle A, \overline{A}, <, =, | \rangle$, where

- $\langle A, \overline{A}, <, =\rangle$ is as defined in Definition 3.3;
- $| \subseteq A \times A$ is the independence relation between action occurrences;

such that relations $<$, $=$ and $|$ are disjoint: $(< \cup =) \cap | = \emptyset$.

The independence relation $|$ is irreflexive, symmetric and not transitive. ■

Partial execution $e\chi$ ranges over $\boldsymbol{XX}$, the domain of partial executions. Domain $\boldsymbol{XX}$ comprises domain $\boldsymbol{X}$, i.e., $\boldsymbol{X} \subset \boldsymbol{XX}$. Therefore, the standard assumed identity relation on regular executions and partial executions is extended in order to allow the comparison between both types of executions.

Definition 3.11 The identity relation $= \subseteq \boldsymbol{XX} \times \boldsymbol{XX}$ is extended with the following equation:

$$\langle A_1, \overline{A}_1, <_1, =_1\rangle = \langle A_2, \overline{A}_2, <_2, =_2, |_2\rangle \Leftrightarrow \quad A_1 = A_2 \ \wedge \ \overline{A}_1 = \overline{A}_2 \ \wedge \ <_1 = <_2 \ \wedge \ =_1 = =_2 \ \wedge \ <_2 \cup =_2 \cup |_2 = A_2 \times A_2 \ . \quad ■$$

In addition, the notion of sub-execution is extended to partial executions.

Definition 3.12 Relation $\sqsubseteq : \boldsymbol{XX} \times \boldsymbol{XX}$ is defined as

$$\langle A_1, \overline{A}_1, <_1, =_1, |_1\rangle \sqsubseteq \langle A_2, \overline{A}_2, <_2, =_2, |_2\rangle \Leftrightarrow A_1 \subseteq A_2 \ \wedge \ \overline{A}_1 \subseteq \overline{A}_2 \ \wedge \ <_1 = <_2 \cap (A_1 \times A_1) \ \wedge \ =_1 = =_2 \cap (A_1 \times A_1) \ \wedge \ |_1 = |_2 \cap (A_1 \times A_1) \ . \quad ■$$

Partial execution-based behaviours

Execution sets which contain one or more partial executions represent incompletely defined behaviours, which are called *partial execution-based behaviours*. Partial execution-based behaviours are used as intermediate results of manipulations on execution sets in the definition of the semantics of causality relations in the next chapters. The final results of these manipulations should be completely defined behaviours, which are represented by execution sets consisting exclusively of regular executions.

Partial execution-based behaviour EE ranges over $\boldsymbol{EE}$, the domain of partial execution-based behaviours. Analogously to executions, the domain of regular behaviours $\boldsymbol{E}$ is contained in the domain of partial behaviours $\boldsymbol{EE}$, i.e., $\boldsymbol{E} \subset \boldsymbol{EE}$. This implies that a partial behaviour is a regular behaviour, when this behaviour consists exclusively of regular executions. The term (execution-based) behaviour is used to denote regular behaviours and partial behaviours, when no confusion is possible.

3.6.5 Constraint-oriented composition (2)

This section presents the second constraint-oriented composition technique, which defines an execution-based behaviour $E(Ac)$ as the cross-conjunction of multiple smaller behaviours $E_i(Ac_i)$, with $Ac_i \subset Ac$, $1 \leq i \leq n$ and $n \geq 2$, each of which defines constraints on the execution of a subset of the actions in Ac. The cross-conjunction operation enforces that $E(Ac)$ represents the conjunction of the constraints represented by all $E_i(Ac_i)$, without using free behaviours. This technique is based on the property that the conjunction of constraints represented by partial executions can be defined directly. The conjunction and cross-conjunction operations are introduced and defined below.

Conjunction of partial executions

The conjunction of two partial executions corresponds to the conjunction of the constraints that are represented by both executions. The partial execution $e\chi$ that results from the conjunction of two partial executions $e\chi_1$ and $e\chi_2$ satisfies both the constraints represented by $e\chi_1$ and the constraints represented by $e\chi_2$. Therefore, partial execution $e\chi$ is defined by the following rules:

1. actions that must occur in $e\chi_1$ or $e\chi_2$, must occur in $e\chi$;
2. actions that must not occur in $e\chi_1$ or $e\chi_2$, must not occur in $e\chi$;
3. action occurrences that are independent in $e\chi_1$ or $e\chi_2$, must be independent in $e\chi$;
4. action occurrences that are ordered or synchronized in $e\chi_1$ or $e\chi_2$, must be related in the same way in $e\chi$.

The conjunction of $e\chi_1$ and $e\chi_2$ is impossible, however, in case both executions have common actions, such that:

- an action must occur in $e\chi_1$ and must not occur in $e\chi_2$, or vice-versa;
- action occurrences are independent in $e\chi_1$ and are ordered or synchronized in $e\chi_2$, or vice-versa;
- action occurrences are ordered or synchronized in $e\chi_1$ and are differently related in $e\chi_2$, or vice-versa;
- action occurrences are ordered or synchronized by the composition of the ordering and synchronization relations of $e\chi_1$ and $e\chi_2$, but are independent or differently related in $e\chi_1$ or $e\chi_2$.

In case the conjunction of two partial executions is impossible, these executions are called *incompatible*. The conjunction of two incompatible executions is undefined.

The conjunction of multiple partial executions corresponds to the conjunction of the constraints represented by all these executions. The conjunction operation is idempotent, commutative and associative. Furthermore, the conjunction of multiple partial executions is undefined if and only if at least two of these executions are incompatible.

For example, the conjunction of executions $\boxed{a \rightarrow b}$, $\boxed{b \rightarrow c}$ and $\boxed{a \rightarrow c}$ is equal to execution $\boxed{a \rightarrow b \rightarrow c}$. The same execution results when execution $\boxed{a \rightarrow c}$ is removed, since the transitivity of the ordering relation < implies that executions $\boxed{a \rightarrow b}$ and $\boxed{b \rightarrow c}$ define an ordering between the occurrences of *a* and *c*. Alternatively, the same execution results when execution $\boxed{a \rightarrow c}$ is replaced by execution $\boxed{a \quad\quad}$ with action domain $\{a\}$, since this execution does not consider, and therefore does not constrain, the execution of action *c*.

An example of two incompatible executions are $\boxed{a \rightarrow b}$ and $\boxed{a \quad \overline{b}}$. The constraint that action *b* must occur according to the former execution and the constraint that action *b* must not occur according to the latter execution, are incompatible. Another example of two incompatible executions are $\boxed{a \rightarrow b}$ and $\boxed{a \quad b}$, since *a* and *b* are related in the former and independent in the latter execution.

For example, the conjunction of executions $\boxed{a \rightarrow b}$ and $\boxed{a \quad c}$ is equal to partial execution $\boxed{a \rightarrow b \cdots c}$. The latter partial execution defines that the independence or relation between *b* and *c* is yet undefined. The conjunction of $\boxed{a \rightarrow b \cdots c}$ with execution $\boxed{b \leftarrow c}$ is equal to execution $\boxed{a \rightarrow b \leftarrow c}$. However, executions $\boxed{a \rightarrow b \cdots c}$ and $\boxed{b \rightarrow c}$ are incompatible since the transitive closure of the ordering $a < b$ in $\boxed{a \rightarrow b \cdots c}$ and the ordering $b < c$ in $\boxed{b \rightarrow c}$ renders $a < c$, which is inconsistent with the independence of *a* and *c* in the former execution.

Cross-conjunction of partial behaviours

The cross-conjunction of two partial behaviours EE_1 and EE_2 is defined as partial behaviour *EE* which consists of all possible executions $e\chi$, such that $e\chi$ is the conjunction of two compatible executions $e\chi_1$ and $e\chi_2$, with $e\chi_1 \in EE_1$ and $e\chi_2 \in EE_2$. Consequently, behaviour *EE* consists of all possible alternative constraints which satisfy both an alternative (sub-)constraint from EE_1 and an alternative (sub-)constraint from EE_2. The cross-conjunction operation is idempotent, commutative and associative.

For example, the cross-conjunction of behaviours $E_1(\{a\}) = \{\boxed{a \quad\quad}, \boxed{\overline{a} \quad\quad}\}$, $E_2(\{a, b\}) = \{\boxed{a \rightarrow b}, \boxed{a \quad \overline{b}}, \boxed{\overline{a} \quad \overline{b}}\}$, and $E_3(\{b, c\}) = \{\boxed{b \rightarrow c}, \boxed{\overline{b} \quad \overline{c}}\}$, is equal to behaviour $E(\{a, b, c\}) = \{\boxed{a \rightarrow b \rightarrow c}, \boxed{a \quad \overline{b} \quad \overline{c}}, \boxed{\overline{a} \quad \overline{b} \quad \overline{c}}\}$. Figure 3.8 illustrates a possible calculation of this cross-conjunction. First, the cross-conjunction of E_1 and E_2 is calculated, which is called E_{12}. Behaviour E_{12} is obtained by calculating for every possible pair of executions $\langle \chi_1, \chi_2 \rangle$, with $\chi_1 \in E_1$ and $\chi_2 \in E_2$, the conjunction of χ_1 and χ_2 if they are compatible, and by discarding the pair $\langle \chi_1, \chi_2 \rangle$ if they are incompatible. Possible and impossible conjunctions of executions are represented by solid and dashed connecting lines, respectively. Subsequently, the cross-conjunction of behaviours E_{12} and E_3 is calculated in a similar way.

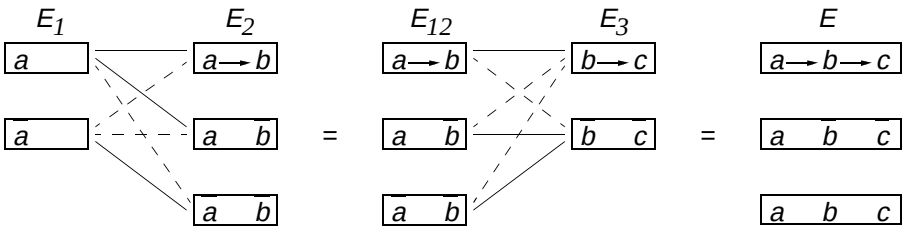


Figure 3.8: Cross-conjunction of three behaviours

Constraint-oriented composition using partial executions

The cross-conjunction operation can be used to compose an execution-based behaviour $E(Ac)$ from two or more smaller behaviours $E_i(Ac_i)$, where each behaviour $E_i(Ac_i)$ defines some constraints on a subset Ac_i of the actions in Ac. This allows one to (de)compose execution-based behaviours from (into) smaller behaviours representing a subset of the constraints of the (de)composed behaviour. Consequently, the cross-conjunction and conjunction operations may be considered as (de)composition operators.

For example, consider the behaviour of Figure 3.8 represented by behaviour $E(\{a, b, c\})$. This behaviour is structured into three smaller behaviours $E_1(\{a\})$, $E_2(\{a, b\})$ and $E_3(\{b, c\})$, which define the following constraints:

- E_1 defines that action a is an initial action which may or may not occur;
- E_2 defines that action b may occur after action a has occurred; and
- E_3 defines that action c must occur after action b has occurred.

The following *completeness rules*, already presented in Section 3.6.3, must be obeyed to obtain a complete constraint-oriented definition of $E(Ac)$:

1. each action of $E(Ac)$ should be defined by at least one of the behaviours $E_i(Ac_i)$, such that $Ac = \cup_i Ac_i$;
2. the independence or relation between each pair of actions in Ac should be defined by one of the behaviours $E_i(Ac_i)$ or by the composition of two (or more) of the behaviours $E_i(Ac_i)$.

The second rule is needed to avoid that the result of the cross-conjunction contains partial executions that are not regular executions. For example, consider the constraint-oriented composition of behaviour $E(\{a, b, c\})$ in Figure 3.9. The cross-conjunction of behaviours $E_1(\{a, b\})$ and $E_2(\{a, c\})$ renders a partial execution in which the independence or relation between occurrences b and c is undefined. The subsequent cross-conjunction with behaviour $E_3(\{b, c\})$ resolves this.

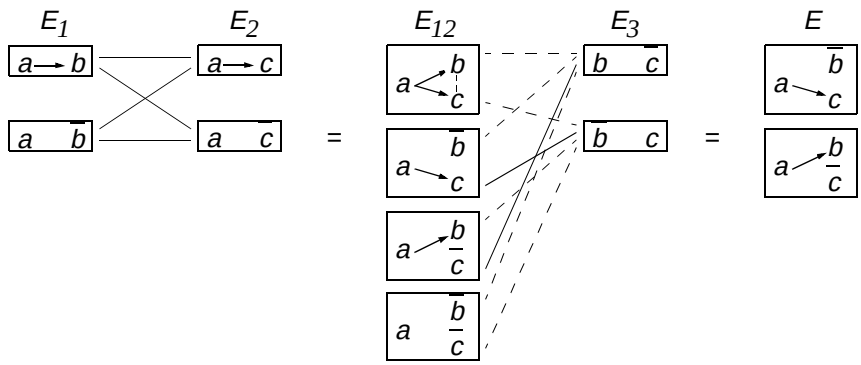


Figure 3.9: Cross-conjunction of three behaviours (2)

Formal definitions

The notions of compatible executions, conjunctions of partial executions and cross-conjunctions of partial behaviours are formally defined below. The conjunction and cross-conjunction operations are formally represented by the product and cross-product operators $\times$ and $\otimes$, respectively. The boolean function $Compat : \boldsymbol{XX} \times \boldsymbol{XX} \rightarrow \{true, false\}$ is introduced to represent whether a pair of executions is compatible, or not, i.e., $Compat(e\chi_1, e\chi_2)$ results in the value *true* if $e\chi_1$ and $e\chi_2$ are compatible, or in the value *false* if they are not.

Definition 3.13 The product operator $\times : \boldsymbol{XX} \times \boldsymbol{XX} \rightarrow \boldsymbol{XX}$ defines the conjunction of two partial executions $e\chi_1 = \langle A_1, \overline{A}_1, <_1, =_1, |_1\rangle$ and $e\chi_2 = \langle A_2, \overline{A}_2, <_2, =_2, |_2\rangle$, such that:

$$
\begin{aligned}
e\chi_1 \times e\chi_2 \;=\; & \langle A_{1\times2}, \overline{A}_{1\times2}, <_{1\times2}, =_{1\times2}, |_{1\times2}\rangle && \text{if } Compat(e\chi_1, e\chi_2) \\
& \text{with:} \quad A_{1\times2} = A_1 \cup A_2 , \\
& \qquad \overline{A}_{1\times2} = \overline{A}_1 \cup \overline{A}_2 , \\
& \qquad <_{1\times2} = (<_1 \cup <_2 \cup (<_1 \cup <_2) \bullet =_{1\times2} \cup =_{1\times2} \bullet (<_1 \cup <_2) \cup \\
& \qquad\qquad =_{1\times2} \bullet (<_1 \cup <_2) \bullet =_{1\times2})^+ , \\
& \qquad =_{1\times2} = (=_1 \cup =_2)^+ , \\
& \qquad |_{1\times2} = |_1 \cup |_2 ; \\
= \; & \text{undefined} && \text{otherwise ;}
\end{aligned}
$$

where

- $<^+$ represents the transitive closure of $<$, such that: $<^+ = \cup\{ <^i \mid i \geq 1\}$;
- $=^+$ represents the transitive closure of $=$, such that: $=^+ = \cup\{ =^i \mid i \geq 1\}$. ■

Definition 3.14 Two partial executions $\langle A_1, \overline{A}_1, <_1, =_1, |_1\rangle$ and $\langle A_2, \overline{A}_2, <_2, =_2, |_2\rangle$ are *compatible* if and only if the following compatibility requirements are fulfilled:

1. $A_{1\times2} \cap \overline{A}_{1\times2} = \emptyset$, which represents that the sets of occurrences and non-occurrences should be disjoint in the conjunction of compatible executions;
2. $<_{1\times2}^{-1} \cap <_{1\times2} = \emptyset$, which represents that two action occurrences a and b can not be related as both $a < b$ and $b < a$ in the conjunction of compatible executions;
3. $<_{1\times2} \cap =_{1\times2} = \emptyset$, which represents that two action occurrences a and b can not be related as both $a < b$ and $a = b$ in the conjunction of compatible executions;
4. $(<_{1\times2} \cup =_{1\times2}) \cap |_{1\times2} = \emptyset$, which represents that two action occurrences a and b can not be both independent and related as $a < b$ or $a = b$ in the conjunction of compatible executions;

where

- $A_{1\times2}, \overline{A}_{1\times2}, <_{1\times2}, =_{1\times2}$ and $|_{1\times2}$ are as defined in Definition 3.13; and
- $<^{-1}$ represents the inverse of $<$, such that: $<^{-1} = \{\langle b, a\rangle \mid \langle a, b\rangle \in <\}$. ■

Definition 3.15 The cross-product operator $\otimes : \wp(\boldsymbol{XX}) \times \wp(\boldsymbol{XX}) \rightarrow \wp(\boldsymbol{XX})$ defines the cross-conjunction of two sets of partial executions, such that:

$$EE_1 \otimes EE_2 = \{e\chi_1 \times e\chi_2 \mid e\chi_1 \in EE_1, e\chi_2 \in EE_2, Compat(e\chi_1, e\chi_2)\} . \quad ■$$

Some useful properties of function *Compat* and the product and cross-product operators $\times$ and $\otimes$ are presented below.

Property 3.16 Given two executions $e\chi_1 = \langle A_1, \overline{A}_1, <_1, =_1, |_1\rangle$ and $e\chi_2 = \langle A_2, \overline{A}_2, <_2, =_2, |_2\rangle$ with the same action domain, the following holds:

$Compat(e\chi_1, e\chi_2) \Leftrightarrow e\chi_1 = e\chi_2$, if $A_1 \cup \overline{A}_1 = A_2 \cup \overline{A}_2$.

Proof: This follows immediately from Definitions 3.13 and 3.14. ■

Property 3.17 The product operator $\times : \boldsymbol{XX} \times \boldsymbol{XX} \rightarrow \boldsymbol{XX}$ is idempotent, commutative and associative.

Proof: The idempotence, commutativity and associativity properties are inherited from the set operator $\cup$, and the commutativity of the *Compat* operator.

The proof of the idempotence and commutativity properties are trivial. In order to proof the associativity property $e\chi_1 \times (e\chi_2 \times e\chi_3) = (e\chi_1 \times e\chi_2) \times e\chi_3$ we need to show that:

$Compat(e\chi_1, (e\chi_2 \times e\chi_3)) \wedge Compat(e\chi_2, e\chi_3)$
$\Leftrightarrow Compat((e\chi_1 \times e\chi_2), e\chi_3) \wedge Compat(e\chi_1, e\chi_2)$,

because the *Compat* operator does not obey the associativity property (in contrast to the set operator $\cup$).

Proof of $\Rightarrow$:

Based on Definition 3.14, this implication must be proven for each compatibility requirement:

(i) $Compat(e\chi_1, (e\chi_2 \times e\chi_3)) \wedge Compat(e\chi_2, e\chi_3)$
$\Rightarrow A_{1\times(2\times3)} \cap \overline{A}_{1\times(2\times3)} = \varnothing$
$\Leftrightarrow (A_1 \cup A_{2\times3}) \cap (\overline{A}_1 \cup \overline{A}_{2\times3}) = (A_1 \cup A_2 \cup A_3) \cap (\overline{A}_1 \cup \overline{A}_2 \cup \overline{A}_2) = \varnothing$
$\Leftrightarrow A_{(1\times2)\times3} \cap \overline{A}_{(1\times2)\times3} = \varnothing$;

(ii) $Compat(e\chi_1, (e\chi_2 \times e\chi_3)) \wedge Compat(e\chi_2, e\chi_3)$
$\Rightarrow <_{1\times(2\times3)}^{-1} \cap <_{1\times(2\times3)} = \varnothing \ \wedge \ <_{2\times3}^{-1} \cap <_{2\times3} = \varnothing$
$\Leftrightarrow <_{1\times(2\times3)}^{-1} \cap <_{1\times(2\times3)} = \varnothing$, using $<_{2\times3} \subseteq <_{1\times(2\times3)}$
$\Leftrightarrow <_{(1\times2)\times3}^{-1} \cap <_{(1\times2)\times3} = \varnothing$, using $<_{(1\times2)\times3} = <_{1\times(2\times3)}$
$\Leftrightarrow <_{(1\times2)\times3}^{-1} \cap <_{(1\times2)\times3} = \varnothing \ \wedge \ <_{1\times2}^{-1} \cap <_{1\times2} = \varnothing$, using $<_{1\times2} \subseteq <_{(1\times2)\times3}$;

(iii) $Compat(e\chi_1, (e\chi_2 \times e\chi_3)) \wedge Compat(e\chi_2, e\chi_3)$
$\Rightarrow <_{1\times(2\times3)} \cap =_{1\times(2\times3)} = \varnothing \ \wedge \ <_{2\times3} \cap =_{2\times3} = \varnothing$
$\Leftrightarrow <_{1\times(2\times3)} \cap =_{1\times(2\times3)} = \varnothing$, using $<_{2\times3} \subseteq <_{1\times(2\times3)}$ and $=_{2\times3} \subseteq =_{1\times(2\times3)}$
$\Leftrightarrow <_{(1\times2)\times3} \cap =_{(1\times2)\times3} = \varnothing$, using $<_{(1\times2)\times3} = <_{1\times(2\times3)}$ and $=_{(1\times2)\times3} = =_{1\times(2\times3)}$
$\Leftrightarrow <_{(1\times2)\times3} \cap =_{(1\times2)\times3} = \varnothing \ \wedge \ <_{1\times2} \cap =_{1\times2} = \varnothing$,
using $<_{1\times2} \subseteq <_{(1\times2)\times3}$ and $=_{1\times2} \subseteq =_{(1\times2)\times3}$;

(iv) $Compat(e\chi_1, (e\chi_2 \times e\chi_3)) \wedge Compat(e\chi_2, e\chi_3)$
$\Rightarrow (<_{1\times(2\times3)} \cup =_{1\times(2\times3)}) \cap |_{1\times(2\times3)} = \varnothing \ \wedge \ (<_{2\times3} \cup =_{2\times3}) \cap |_{2\times3} = \varnothing$
$\Leftrightarrow (<_{1\times(2\times3)} \cup =_{1\times(2\times3)}) \cap |_{1\times(2\times3)} = \varnothing$
$\Leftrightarrow (<_{(1\times2)\times3} \cup =_{(1\times2)\times3}) \cap |_{(1\times2)\times3} = \varnothing$,
using $<_{(1\times2)\times3} = <_{1\times(2\times3)}$, $=_{(1\times2)\times3} = =_{1\times(2\times3)}$ and $|_{(1\times2)\times3} = |_{1\times(2\times3)}$;
$\Leftrightarrow (<_{(1\times2)\times3} \cup =_{(1\times2)\times3}) \cap |_{(1\times2)\times3} = \varnothing \ \wedge \ (<_{1\times2} \cup =_{1\times2}) \cap |_{1\times2} = \varnothing$;

such that *(i)*, *(ii)*, *(iii)* and *(iv)* $\Rightarrow Compat(\chi_1, \chi_2) \wedge Compat((\chi_1 \times \chi_2), \chi_3)$.
The proof of the properties $<_{(1\times2)\times3} = <_{1\times(2\times3)}$ and $=_{(1\times2)\times3} = =_{1\times(2\times3)}$ are added to the end of this chapter in Section 3.9.

Proof of $\Leftarrow$: analogously. ■

Property 3.18 The cross-product operator $\otimes : \wp(\boldsymbol{XX}) \times \wp(\boldsymbol{XX}) \rightarrow \wp(\boldsymbol{XX})$ is idempotent, commutative and associative.

Proof: The idempotence, commutativity and associativity properties are inherited from the product operator $\times$. ■

Property 3.19 The cross-conjunction of two behaviours with the same action domain is equal to the intersection of both behaviours, using the intersection operator on sets, such that:

$E_1(Ac) \otimes E_2(Ac) = E_1(Ac) \cap E_2(Ac)$.

Proof: This property follows directly from Property 3.16. ■

Property 3.20 Free-behaviour *E-Free*(Ac) is the identity element of any behaviour $E(Ac)$ with the same action domain under the cross-conjunction operation, such that:

E-Free$(Ac) \otimes E(Ac) = E(Ac)$.

Proof: *E-Free*$(Ac) \otimes E(Ac) =$ *E-Free*$(Ac) \cap E(Ac) = E(Ac)$, using the property that $E(Ac) \subseteq$ *E-Free*(Ac) and Property 3.19. ■

3.6.6 Correspondence between restriction and cross-conjunction

An execution-based behaviour $E(Ac)$ can be defined in a constraint-oriented way as the composition of multiple smaller behaviours $E_i(Ac_i)$ using the techniques presented in Sections 3.6.3 and 3.6.5. The correspondence between both techniques is defined by the following equation:

E-Free$(Ac) \setminus\setminus \{E_i(Ac_i) \mid i\} = \otimes\{E_i(Ac_i) \mid i\}$, if the first completeness rule is satisfied.

Formal definition

Property 3.22 formally defines and proves the above equation. This property uses two other properties which are presented first.

Property 3.21 The following relation between the notion of sub-execution and compatibility holds:

$\chi_1(Ac_1) \sqsubseteq \chi_2(Ac_2) \Leftrightarrow Compat(\chi_1(Ac_1), \chi_2(Ac_2))$, if $Ac_1 \subseteq Ac_2$.

Proof:

$\Rightarrow$: this implication follows immediately from Definitions 3.14 and 3.25.

$\Leftarrow$: the proof of this implication consists of the following two steps:

(i) $Compat(\chi_1(Ac_1), \chi_2(Ac_2))$
$\Rightarrow A_{1\times2} \cap \overline{A}_{1\times2} = (A_1 \cup A_2) \cap (\overline{A}_1 \cup \overline{A}_2) = \varnothing$

$\Rightarrow A_1 \subseteq A_2$ and $\overline{A}_1 \subseteq \overline{A}_2$, using the assumption $Ac_1 \subseteq Ac_2$;

(ii) $Compat(\chi_1(Ac_1), \chi_2(Ac_2))$
$\Rightarrow <_{1\times2}^{-1} \cap <_{1\times2} = \emptyset \wedge <_{1\times2} \cap =_{1\times2} = \emptyset \wedge (<_{1\times2} \cup =_{1\times2}) \cap |_{1\times2} = \emptyset$
$\Rightarrow <_1 = <_2 \cap (A_1 \times A_1)$ and $=_1 = =_2 \cap (A_1 \times A_1)$,
using $A_1 \times A_1 \subseteq A_2 \times A_2$, $<_1 \cup =_1 \cup |_1 = A_1 \times A_1$ and $<_2 \cup =_2 \cup |_2 = A_2 \times A_2$;

the combination of *(i)* and *(ii)* implies $\chi_1(Ac_1) \sqsubseteq \chi_2(Ac_2)$. ■

Property 3.22 The following relation between the restriction operator \ and the cross-conjunction operator ⊗ holds:

$E_2(Ac_2) \setminus E_1(Ac_1) = E_2(Ac_2) \otimes E_1(Ac_1)$, if $Ac_1 \subseteq Ac_2$.

Proof:

$E_2(Ac_2) \setminus E_1(Ac_1)$
$= \{\chi_2 \mid \chi_2 \in E_2(Ac_2), \exists\chi_1 \in E_1(Ac_1) \mid \chi_1 \sqsubseteq \chi_2\}$, using Definition 3.6
$= \{\chi_2 \times \chi_1 \mid \chi_2 \in E_2(Ac_2), \chi_1 \in E_1(Ac_1), \chi_1 \sqsubseteq \chi_2\}$, using $\chi_1 \sqsubseteq \chi_2 \Rightarrow \chi_2 = \chi_2 \times \chi_1$
$= \{\chi_2 \times \chi_1 \mid \chi_2 \in E_2(Ac_2), \chi_1 \in E_1(Ac_1), Compat(\chi_1, \chi_2)\}$, using Property 3.21
$= E_2(Ac_2) \otimes E_1(Ac_1)$. ■

Property 3.23 Assuming $Ac = \cup_i Ac_i$ the following equation holds:

$E\text{-}Free(Ac) \setminus\setminus \{E_i(Ac_i) \mid i\} = \otimes\{E_i(Ac_i) \mid i\}$.

Proof:

$E\text{-}Free(Ac) \setminus\setminus \{E_i(Ac_i) \mid i\}$
$= \cap\{E\text{-}Free(Ac) \setminus E_i(Ac_i) \mid i\}$, using Definition 3.9
$= \cap\{E\text{-}Free(Ac) \otimes E_i(Ac_i) \mid i\}$, using Property 3.22 and $Ac_i \subseteq Ac$
$= \otimes\{E\text{-}Free(Ac) \otimes E_i(Ac_i) \mid i\}$, using Property 3.19
$= \otimes\{E_i(Ac_i) \mid i\}$, using Properties 3.18 and 3.20. ■

3.6.7 Lack of conciseness

Despite the composition operators × and ⊗, the execution model lacks conciseness to model and structure behaviours with multiple actions in a clear and comprehensible way. This is illustrated by considering all possible execution-based behaviours consisting of three actions *a*, *b* and *c*.

The set of all possible executions of three actions $E\text{-}Free(\{a, b, c\})$ is obtained by the cross-conjunction of the following three free behaviours of two actions:

- $E\text{-}Free_1(\{a, b\}) = \{[a\ \ b], [a \rightarrow b], [b \rightarrow a], [a = b], [a\ \ \ ], [b\ \ \ ], [\ \ \ ]\}$;
- $E\text{-}Free_2(\{b, c\}) = \{[b\ \ c], [b \rightarrow c], [c \rightarrow b], [b = c], [b\ \ \ ], [c\ \ \ ], [\ \ \ ]\}$;
- $E\text{-}Free_3(\{a, c\}) = \{[a\ \ c], [a \rightarrow c], [c \rightarrow a], [a = c], [a\ \ \ ], [c\ \ \ ], [\ \ \ ]\}$.

The above cross-conjunction operation renders 45 different executions of three actions, i.e., $|E\text{-}Free(\{a, b, c\})| = 45$. This implies that the number of different behaviours of three actions that can in principle be defined is equal to 2^{45}-1.

The execution model does not provide the concepts to capture the common characteristics of behaviours, which define similar relations between actions *a*, *b* and *c*. The availability of these concepts would facilitate the structuring of all 2^{45}-1 behaviours into behaviour families. Furthermore, they would facilitate the construction of behaviours by representing the desired characteristics of the relations between actions *a*, *b* and *c* directly, instead of representing them indirectly by enumerating all possible executions.

3.7 References to action attribute values

This section presents the refinements of the execution concept that are necessary to model attribute values, i.e., the information, time and location values, that are established in action occurrences, and to model reference relations between attribute values of different action occurrences.

3.7.1 Executions with result values

In an execution, an action occurrence *a* may establish an information value ι_a, a time value τ_a, and a location value λ_a. Therefore, the execution concept is extended with an *information function* ι, a *time function* τ and a *location function* λ, which associates with each action occurrence *a* in an execution the information value $\iota(a)$, or ι_a, the time value $\tau(a)$, or τ_a, and the location value $\lambda(a)$, or λ_a, that is established by *a*. Figure 3.10 illustrates this extension using two alternative graphical representations. For example, action occurrence *a* establishes character ‘i’ as information value, which is made available at time moment 1 at location ‘C1’.

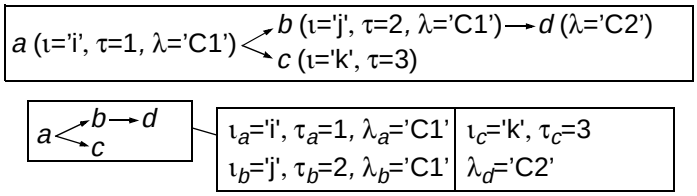


Figure 3.10: Executions with attribute values

The definition of action attribute values is optional. In case the characteristic represented by some action attribute is irrelevant for the purpose of the model, the corresponding attribute value may be omitted.

Modelling of the time attribute

The time function τ represents relative time moments which are measured w.r.t. some reference point in time. We assume that this time reference point is common to all executions of an execution-based behaviour, and represents the time moment at which the execution of this behaviour is initiated.

For example, time function τ in Figure 3.10 defines that actions *a*, *b* and *c* occur 1, 2 and 3 time units after some presumed time reference point, respectively. This allows one to conclude, e.g., that *c* occurs 2 time units after *a*. The granularity of a time unit may be chosen

differently depending on the particular design problem. For example, the size of a time unit may range from a nano-second to represent activities that take place at the rate of a CPU-clock, to calendar months (or years) to represent the delay between the ordering and delivering of a PC in our department at a rather ‘accurate’ level of detail.

In case a behaviour can be partitioned into two or more independent sub-behaviours, one may argue that distinct time reference points should be associated with these sub-behaviours. This can be achieved, however, by defining these sub-behaviours as separate behaviours. Furthermore, we want to have the possibility to model a common time reference point for independent sub-behaviours by defining them within a single behaviour definition.

Since we do not model the time moments of action occurrences relative to their immediate predecessors in an execution, nor relative to synchronized action occurrences, the following consistency rules are imposed on the time function τ:

- if $a < b$ implies $\tau(a) < \tau(b)$;
- if $a = b$ implies $\tau(a) = \tau(b)$.

3.7.2 Modelling of attribute value references

In general, the establishment of a result value in an action occurrence *a* may depend on attribute values that are established in other action occurrences. In that case we say that action occurrence *a refers* to the attribute values of other action occurrences, or that a *reference relation* is defined between an action attribute of action *a* and action attributes of other actions. Reference relations between information, time and location attributes are called information, time and location reference relations, respectively.

We define that action occurrence *a* may refer to attribute values of another action occurrence *b*, when *a* depends on *b* and *a* occurs after *b*, i.e., $b < a$. The motivation for this definition is that attribute value references from one action occurrence to another implies a dependency (relation) between these action occurrences. Furthermore, at first, we consider referring in its common meaning as referring to something that has been established in the past.

Figure 3.11 illustrates the modelling of the information reference relation $\iota_b = \iota_a + 1$ between two ordered action occurrences *a* and *b*, assuming that ι_a and ι_b are natural numbers. This reference relation is modelled by an infinite number of executions, which is indicated by the dashed line.

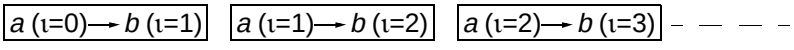


Figure 3.11: Modelling of reference relation $\iota_b = \iota_a + 1$

Another type of reference relation considered in this thesis is the mutual reference between attribute values of two synchronized action occurrences. Figure 3.12 depicts the executions of a behaviour of three actions *a*, *b* and *c*, where *c* represents the initiation of two parallel processes *a* and *b*, which must finish simultaneously. In addition, the mutual reference relation $\lambda_a \neq \lambda_b$, with $\lambda_a, \lambda_b \in \{P1, P2, P3\}$, is defined between *a* and *b*, defining that *a* and *b*

must be dispatched at one of three different processors, such that *a* and *b* are not executed at the same processor.

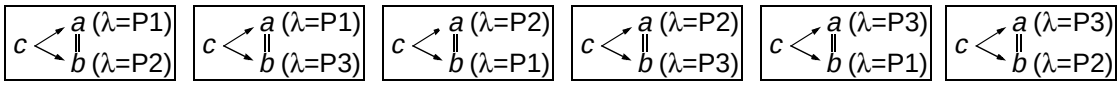


Figure 3.12: Modelling of mutual reference relation $\lambda_a \neq \lambda_b$

In principle, any relation between attribute values of action occurrences can be modelled in terms of the execution model, including relations we want to forbid. For example, it is even possible to define a relation between the attribute values of two independent actions. We will only consider the modelling of (mutual) reference relations between action occurrences related by < or =, in the sequel.

Implicit time references

The ordering and synchronization relations < and =, define implicit time value reference relations between related action occurrences. For example, in case of Figure 3.11 the ordering relation $a < b$ implies that $\tau_a < \tau_b$, and in case of Figure 3.12 the synchronization relation $a = b$ implies that $\tau_a = \tau_b$.

These implicit time references must be consistent with the definition of (explicit) references between the time values of action occurrences. This requirement is reflected in the following consistency rules:

1. in case two action occurrences *a* and *b* are related as $a < b$, the definition of an (explicit) reference relation between their time values should be consistent with the implicit time reference relation $\tau_a < \tau_b$;
2. in case two action occurrences *a* and *b* are related as $a = b$, the definition of an (explicit) reference relation between their time values should be consistent with the implicit time reference relation $\tau_a = \tau_b$.

For example, the definition of explicit time reference relation $\tau_a + 1 < \tau_b < \tau_a + 3$ is consistent with the implicit time reference relation $\tau_a < \tau_b$ in Figure 3.11, but is inconsistent with the implicit time reference relation $\tau_a = \tau_b$ in Figure 3.12.

3.7.3 Compositions of attribute value references

The information, time and location functions represent additional constraints on the execution of the involved actions, namely, the attribute values that must be established in the action occurrences of this execution. These constraints are called *attribute value constraints*. Accordingly, constraints represented by the information, time and location functions are called information, time and location value constraints, respectively.

For example, execution $\boxed{a\ (\iota=\text{"hoi"}, \tau=1, \lambda=\text{"here"}) \longrightarrow b\ (\tau=2)}$ represents the following attribute value constraints in addition to the constraints represented by execution $\boxed{a \longrightarrow b}$ as discussed in Section 3.6:

- action *a* must occur 1 time unit after the initiation of the execution, must establish information value “hoi”, and must take place at location “here”;
- action *b* must occur 2 time units after the initiation of the execution, and no constraints are imposed on the information value established by *b* or the location where *b* takes place.

Extension of conjunction and cross-conjunction operations

The conjunction operation is refined in order to define the conjunction of attribute value constraints that are represented by distinct executions. In addition to the rules given in Section 3.6.5, the execution $e\chi$ that results from the conjunction of executions $e\chi_1$ and $e\chi_2$ is defined by the following rule:

5. actions that establish a certain attribute value in $e\chi_1$ or $e\chi_2$, must establish the same value in $e\chi$;

As a consequence, the following additional source of incompatibility has to be considered. The conjunction of executions $e\chi_1$ and $e\chi_2$ is impossible in case both executions have common actions, such that:

- these actions establish different information values, time values or location values in $e\chi_1$ and $e\chi_2$.

We assume that the absence of the definition of an action attribute value implies that the corresponding action attribute is irrelevant, and therefore no constraint is imposed on the action attribute values that may be established. Consequently, we assume that the conjunction of an execution which defines no constraint on a particular action attribute with an execution which defines a specific value for this action attribute renders an execution which defines the same attribute value as the latter execution.

Figure 3.13 depicts the conjunction of two compatible executions. This example also illustrates the conjunction of an attribute value constraint in some execution with the absence of a constraint on the corresponding action attribute in another execution. For example, the conjunction of the upper left execution which defines the information value ι_b = ‘j’, with the lower left execution which allows any information value to be established in *b*, renders an execution which defines the information value ι_b = ‘j’. This property allows one to distribute constraints on attribute values over multiple executions.

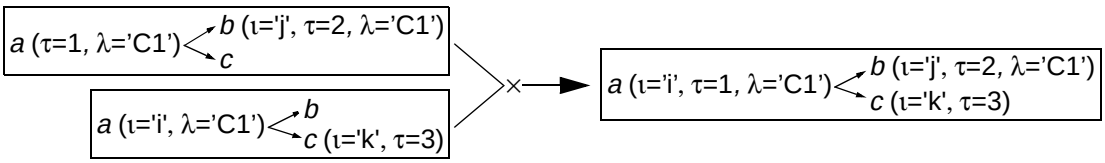


Figure 3.13: Conjunction of executions with attribute values

Figure 3.14 depicts two incompatible executions, since both executions establish a different time value in action occurrence *a*, and also a different information value in action occurrence *c*, each of which is sufficient for incompatibility.

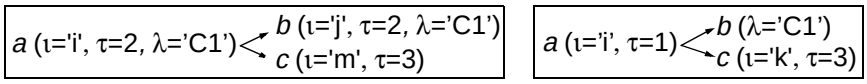


Figure 3.14: Incompatible executions with attribute values

Orthogonal composition of reference relations

Based on the refined conjunction operation, the cross-conjunction operation can be used to compose an execution-based behaviour from three sub-behaviours E_ι, E_τ and E_λ, such that:

- E_ι defines all information value constraints (and no other attribute value constraints);
- E_τ defines all time value constraints (and no other attribute value constraints);
- E_λ defines all location value constraints (and no other attribute value constraints).

This implies that information, time and location value constraints can be defined as orthogonal constraints.

Figure 3.15 depicts the cross-conjunction of two execution-based behaviours E_ι and E_τ, such that: E_ι defines the information reference relation $\iota_b = \iota_a + 1$ and E_τ defines the time reference relation $\tau_b = \tau_a + 1$. The resulting behaviour E allows all possible executions which satisfy the reference relations $\iota_b = \iota_a + 1$ and $\tau_b = \tau_a + 1$, i.e., E defines all possible combinations of information and time value constraints that may be imposed on action occurrences a and b by these reference relations.

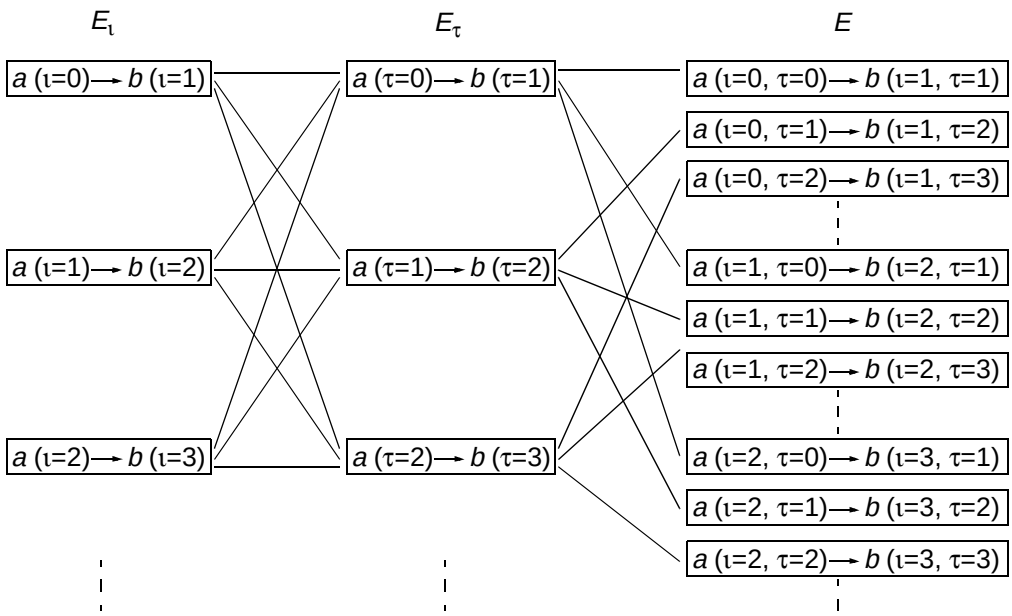


Figure 3.15: Cross-conjunction of orthogonal result value constraints

The above property of the cross-conjunction operation allows one to define action attributes and reference relations between action attributes independently of each other. Furthermore, this property guarantees that the execution model supports the independent (orthogonal) development and union of the information, time and location reference modules identified in Section 2.5.4.

Reference relations between different types of action attribute values

In the foregoing, we have considered reference relations between attribute values of the same type, i.e., between information, time or location values. In addition, the execution model allows one to define reference relations between distinct types of attribute values. In this respect, time values and location values may be considered as information values with specific interpretations.

Figure 3.16 depicts an example of a reference relation between information and time values. The represented behaviour models the sequential composition of action occurrences *a* and *b*, such that the information value of *b* is equal to the time interval between *a* and *b*, which is represented by reference relation $\iota_b = \tau_b - \tau_a$.

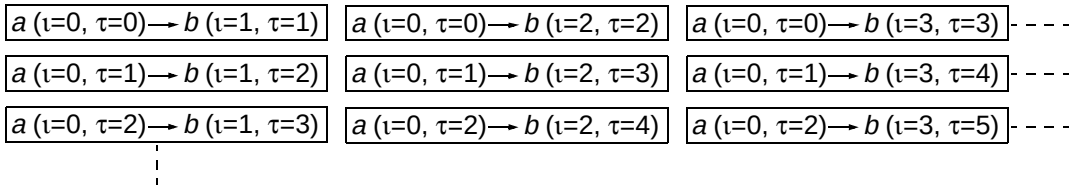


Figure 3.16: Reference relation $\iota_b = \tau_b - \tau_a$

3.7.4 Formal definition

This section formally defines the refinements of the execution concept, the conjunction operation, and its compatibility requirements as discussed in the preceding sections.

The formal definition of an execution extended with an information, time and location function is given below.

Definition 3.24 An execution $\chi \in E$ is defined as a seven-tuple $\langle A, \overline{A}, <, =, \iota, \tau, \lambda \rangle$, where

- A, $\overline{A}$, $<$ and $=$ are as defined in Definition 3.3;
- $\iota : A \rightarrow \boldsymbol{I}$ is a (partial) information function, which defines the information value that is established by (a subset of) the action occurrences in χ;
- $\tau : A \rightarrow \boldsymbol{T}$ is a (partial) time function, which defines the time value that is established by (a subset of) the action occurrences in χ;
- $\lambda : A \rightarrow \boldsymbol{L}$ is a (partial) location function, which defines the location value that is established by (a subset of) the action occurrences in χ.

In addition the following consistency rules w.r.t. the time function τ must be obeyed:

$$\forall \langle a, b \rangle \in <_\chi \mid \tau(a) < \tau(b) \text{ and } \forall \langle a, b \rangle \in =_\chi \mid \tau(a) = \tau(b) \ . \qquad \blacksquare$$

Some instances of the information, time and location function may be undefined, since the definition of attribute values is optional. The instances $\iota(a)$, $\tau(a)$ and $\lambda(a)$ are also denoted as ι_a, τ_a and λ_a, respectively.

In case the information, time or location values of action occurrences in an execution are irrelevant, the corresponding information, time or location function can be omitted from

Definition 3.24. This renders executions with different mathematical structures. Accordingly, domain $\boldsymbol{X}$ consists of the following disjoint sub-domains:

$$\boldsymbol{X} = \boldsymbol{X'} \cup \boldsymbol{X}_{\iota} \cup \boldsymbol{X}_{\tau} \cup \boldsymbol{X}_{\lambda} \cup \boldsymbol{X}_{\iota\tau} \cup \boldsymbol{X}_{\iota\lambda} \cup \boldsymbol{X}_{\tau\lambda} \cup \boldsymbol{X}_{\iota\tau\lambda} ,$$

which represent the sub-domains of executions with the following mathematical structures, respectively: $\langle A, \overline{A}, <, =\rangle$, $\langle A, \overline{A}, <, =, \iota\rangle$, $\langle A, \overline{A}, <, =, \tau\rangle$, $\langle A, \overline{A}, <, =, \lambda\rangle$, $\langle A, \overline{A}, <, =, \iota, \tau\rangle$, $\langle A, \overline{A}, <, =, \iota, \lambda\rangle$, $\langle A, \overline{A}, <, =, \tau, \lambda\rangle$, $\langle A, \overline{A}, <, =, \iota, \tau, \lambda\rangle$. In the sequel, sub-domain $\boldsymbol{X'}$ is also denoted as $\boldsymbol{X}$, i.e., without the prime, when it is clear from the context that information, time and location values of action occurrences are not considered.

The refinement of the partial execution concept with the information, time and location function, can be performed analogously to the refinement of the (regular) execution concept above (by adding the independence relation "|"). The corresponding refinement of the identity relation on partial executions (see Definition 3.11) is straightforward.

The refinements of the conjunction operation and its compatibility requirements are formally represented by re-definitions of the boolean function *Compat* and the product operator $\times$ for sub-domain $\boldsymbol{XX}_{\iota\tau\lambda}$, respectively. The corresponding refinements for the other sub-domains of $\boldsymbol{XX}$ can be derived by omitting the information, time or location function.

Definition 3.25 Two partial executions $\langle A_1, \overline{A}_1, <_1, =_1, |_1, \iota_1, \tau_1, \lambda_1\rangle$ and $\langle A_2, \overline{A}_2, <_2, =_2, |_2, \iota_2, \tau_2, \lambda_2\rangle$ are *compatible* if and only if the following compatibility requirements are fulfilled:

- $\langle A_1, \overline{A}_1, <_1, =_1, |_1\rangle$ and $\langle A_2, \overline{A}_2, <_2, =_2, |_2\rangle$ are compatible according to Definition 3.14;
- $\forall a \in (A_1 \cap A_2) \mid \iota_1(a)$ and $\iota_2(a)$ are defined implies $\iota_1(a) = \iota_2(a)$;
- $\forall a \in (A_1 \cap A_2) \mid \tau_1(a)$ and $\tau_2(a)$ are defined implies $\tau_1(a) = \tau_2(a)$;
- $\forall a \in (A_1 \cap A_2) \mid \lambda_1(a)$ and $\lambda_2(a)$ are defined implies $\lambda_1(a) = \lambda_2(a)$. ■

Definition 3.26 The product operator $\times : \boldsymbol{XX}_{\iota\tau\lambda} \times \boldsymbol{XX}_{\iota\tau\lambda} \rightarrow \boldsymbol{XX}_{\iota\tau\lambda}$ defines the *product* of two executions $e\chi_1 = \langle A_1, \overline{A}_1, <_1, =_1, |_1, \iota_1, \tau_1, \lambda_1\rangle$ and $e\chi_2 = \langle A_2, \overline{A}_2, <_2, =_2, |_2, \iota_2, \tau_2, \lambda_2\rangle$, such that:

$$\begin{aligned} e\chi_1 \times e\chi_2 \quad &= \langle A_{1\times2}, \overline{A}_{1\times2}, <_{1\times2}, =_{1\times2}, |_{1\times2}, \iota_{1\times2}, \tau_{1\times2}, \lambda_{1\times2}\rangle \qquad \text{if } Compat(e\chi_1, e\chi_2) \\ &\qquad \text{with:} \quad A_{1\times2}, \overline{A}_{1\times2}, <_{1\times2}, =_{1\times2}, |_{1\times2} \text{ as defined in Definition 3.13,} \\ &\qquad\qquad \iota_{1\times2} = \iota_1 \cup \iota_2,\ \tau_{1\times2} = \tau_1 \cup \tau_2,\ \lambda_{1\times2} = \lambda_1 \cup \lambda_2; \\ &= \text{undefined} \qquad\qquad \text{otherwise.} \end{aligned}$$

■

Property 3.27 The product operator $\times : \boldsymbol{XX}_{\iota\tau\lambda} \times \boldsymbol{XX}_{\iota\tau\lambda} \rightarrow \boldsymbol{XX}_{\iota\tau\lambda}$ is idempotent, commutative and associative.

Proof: This property is inherited from the union operator $\cup$ and Property 3.17. ■

3.8 Probability of action occurrences

This section presents a refinement of the notion of execution-based behaviour that is necessary to model the probability of executions, and explains how the integral and stochastic probability attribute can be modelled using this refinement.

3.8.1 Probability of executions

In the preceding sections we have (implicitly) assumed that executions happen with a probability larger than zero. In other words, when a behaviour is executed an infinite number of times, we assume that each execution that has been defined in the corresponding execution-based behaviour definition happens at least once. In case an execution happens with a probability equal to zero, i.e., it never happens, we assume that this execution should be removed from the behaviour definition.

In order to model the integral and stochastic probability of action occurrences, we extend an execution-based behaviour E with a probability function π_E to define the probability of subsets of executions in E, i.e., the probability of individual executions as well as particular groupings of executions in E, such that: $\pi_E : \wp(E) \rightarrow \wp(\boldsymbol{P})$.

When assuming that a behaviour is executed a sufficiently large (infinite) number of times, we define the probability of an execution as the ratio between the number of times this execution happens when the corresponding behaviour is executed and the total number of times this behaviour is executed. Correspondingly, the probability of a set of executions is defined as the ratio between the number of times that one of these executions happen when the corresponding behaviour is executed and the total number of times this behaviour is executed.

In general, we want to be able to define that the actual probability of a set of executions of some system behaviour lies within a certain range of acceptable probability values. Therefore, the probability of a set of executions is defined by a set of one or more possible probability values, which should be a subset of the probability domain $\boldsymbol{P}$.

The following consistency rules on the definition of the probability function π_E must be obeyed:

1. the probability of a subset of executions $E' \subseteq E$ is equal to the sum of the individual executions in E, i.e., $\pi_E(E') = \Sigma \{\pi_E(\{\chi\}) \mid \chi \in E'\}$;
2. the sum of the probability of all individual executions in E is equal to probability value 1, i.e., $\pi_E(E) = 1$.

Figure 3.17 depicts an execution-based behaviour of five actions *a*, *b*, *c*, *d* and *e*. The probability function π is defined in a separate text-box, which is attached to the behaviour. Furthermore, the executions are tagged with a name χ_i in order to be able to refer to them. Alternatively, the probability of an execution may be defined in a text-box that is attached to the execution itself. For simplicity, we consider here only singletons as probability values. We allow singletons to be represented as a single probability value instead of a set consisting of one probability value.

Based on the definition of the probability function π in Figure 3.17, the following behaviour description can be derived. Action *a* occurs with a probability of 0.9 (= 1 - $\pi(\{\chi_1\})$). After *a* has occurred a choice is made between the occurrence of action *b* with a probability of 0.63 (= $\pi(\{\chi_4\})/\pi(\{\chi_2, \chi_3, \chi_4, \chi_5\} = \pi(\{\chi_4\})/0.9$), the occurrence of action *c* with a probability of 0.32 (= $\pi(\{\chi_3, \chi_5\})/0.9$), or the non-occurrences of *b* and *c* with a probability of 0.05 (= $\pi(\{\chi_2\})/0.9$). In case *b* occurs, action *d* must occur with a probability of 1. And in case *c*

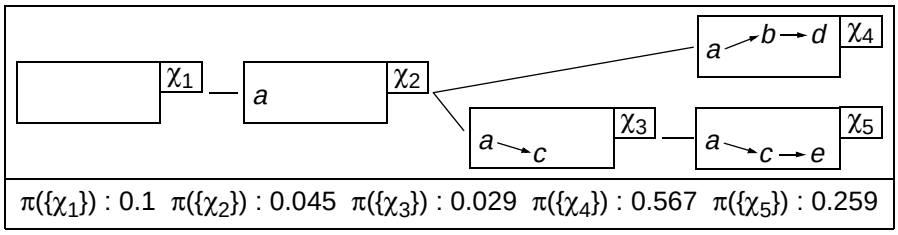


Figure 3.17: Probability function π

occurs, action *e* must occur with a probability of 0.9 (= $\pi(\{\chi_5\})/\pi(\{\chi_3, \chi_5\})$). The solid lines in Figure 3.17 represent a prefix relationship between the involved executions, which is defined in Section 3.8.4.

3.8.2 Modelling of the integral probability attribute

The integral probability attribute of an action *a* defines for each condition γ that allows the occurrence of *a*, the integral probability that *a* occurs when this condition is satisfied. The (conditional) probability that *a* occurs when γ is satisfied is denoted as $\pi_a(\gamma)$.

Consider a behaviour *E* in which action *a* is defined. The integral probability $\pi_a(\gamma)$ is equal to the ratio between the probability of execution set $E\backslash a\wedge\gamma$ and the probability of execution set $E\backslash\gamma$, where

- $E\backslash a\wedge\gamma$ represents the restriction of *E* to the (sub)set of executions in which condition γ is satisfied and action *a* occurs;
- $E\backslash\gamma$ represents the restriction of *E* to the (sub)set of executions in which condition γ is satisfied (and action *a* either occurs or does not occur).

Consequently, the integral probability attribute is modelled by defining probability function π_E such that:

$$\pi_E(E\backslash a\wedge\gamma) \,/\, \pi_E(E\backslash\gamma) = \pi_a(\gamma),$$

for all actions *a* in *E* and all conditions γ that allow the occurrence of *a*.

Figure 3.18 illustrates the modelling of the integral probability attributes of two actions *a* and *b*, which consist of the probability attribute values $\pi_a(\gamma_a) = 0.9$ and $\pi_b(\gamma_b) \in (0.6..1]$, respectively, where condition γ_a represents that *a* is allowed to occur from the beginning of an execution, and condition γ_b represents that *b* is allowed to occur after *a* has occurred. The probability values of $\pi(\{\chi_1\})$, $\pi(\{\chi_2\})$ and $\pi(\{\chi_3\})$ are determined using the definition above and equation $\pi(\{\chi_1, \chi_2, \chi_3\}) = 1$.

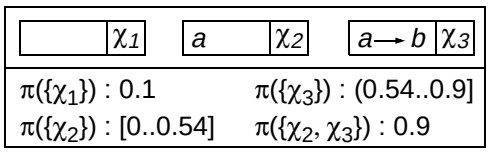


Figure 3.18: Modelling of integral probability attribute

Refinement of the uncertainty attribute

The way in which the integral probability attribute is modelled in the execution model is a refinement of the way in which the uncertainty attribute is modelled. This can be understood by describing the correspondence between both attributes.

Considering a behaviour E in which an action a is allowed to occur when condition γ is satisfied, the uncertainty attribute value $\upsilon_a(\gamma)$ and the integral probability attribute value $\pi_a(\gamma)$ are related as follows:

- $\upsilon_a(\gamma)$ = *must* corresponds to $\pi_a(\gamma) = 1$: action a must occur in every execution in which condition γ is satisfied, which corresponds to equation $\pi_E(E\backslash a \wedge \gamma) / \pi_E(E\backslash\gamma) = 1$;
- $\upsilon_a(\gamma)$ = *may* corresponds to $\pi_a(\gamma) \in (0..1)$: action a may occur in an execution in which condition γ is satisfied, which corresponds to equation $0 < \pi_E(E\backslash a \wedge \gamma) / \pi_E(E\backslash\gamma) < 1$.

When the above correspondence is obeyed, the uncertainty attribute and the integral probability attribute allow the same behaviour E. In case of the uncertainty attribute, all executions in E are assumed to occur with a probability larger than zero, such that the sum of the probability of all executions in E is equal to 1. The probability attribute allows one to refine the probability of these executions by constraining the possible probability values of (sets of) executions using the probability function π.

Example

Figure 3.19 illustrates a refinement of the behaviour in Figure 3.4(ii), which is based on the following definition of the integral probability attribute:

- $\pi_a(\gamma_a) = 1$ and $\pi_b(\gamma_b) = 0.8$, where γ_a and γ_b represent the condition that a and b are allowed to occur from the beginning of an execution, respectively;
- $\pi_c(\gamma_c) = 0.8$, where γ_c represents the condition that c either depends on a and is independent of b such that c is allowed to occur after a has occurred, or c depends on b and is independent of a such that c is allowed to occur after b has occurred.

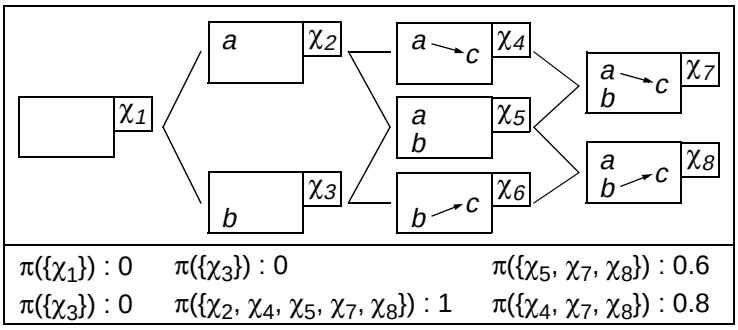


Figure 3.19: Modelling of integral probability attribute (2)

The probability function π is derived using the following definitions, with $E = \{\chi_1, .., \chi_8\}$:

- $\pi_a(\gamma_a) = \pi(E\backslash a \wedge \gamma_a) / \pi(E\backslash\gamma_a) = \pi(\{\chi_2, \chi_4, \chi_5, \chi_7, \chi_8\}) / \pi(E) = 1$. This equation implies that: $\pi(\{\chi_2, \chi_4, \chi_5, \chi_7, \chi_8\}) = 1$ and $\pi(\{\chi_1\}) = \pi(\{\chi_3\}) = \pi(\{\chi_6\}) = 0$;

- $\pi_b(\gamma_b) = \pi(E\backslash b \wedge \gamma_b) / \pi(E\backslash \gamma_b) = \pi(\{\chi_3, \chi_5, \chi_6, \chi_7, \chi_8\}) / \pi(E) = 0.6$. Based on this and the above equations, the following holds: $\pi(\{\chi_5, \chi_7, \chi_8\}) = 0.6$;
- $\pi_c(\gamma_c) = \pi(E\backslash c \wedge \gamma_c) / \pi(E\backslash \gamma_c) = \pi(\{\chi_4, \chi_6, \chi_7, \chi_8\}) / \pi(\{\chi_2, \chi_3, \chi_4, \chi_5, \chi_6, \chi_7, \chi_8\}) = 0.8$. Based on this and the above equations, the following holds: $\pi(\{\chi_4, \chi_7, \chi_8\}) = 0.8$.

Furthermore, the equations $\pi(\{\chi_2, \chi_4\}) = 0.4$ and $\pi(\{\chi_2, \chi_5\}) = 0.2$ can be derived by substituting $\pi(\{\chi_5, \chi_7, \chi_8\}) = 0.6$ and $\pi(\{\chi_4, \chi_7, \chi_8\}) = 0.8$ in $\pi(\{\chi_2, \chi_4, \chi_5, \chi_7, \chi_8\}) = 1$, respectively.

We conclude that the resulting behaviour is not a proper refinement of the behaviour in Figure 3.4(ii), since executions χ_1, χ_3 and χ_5 have become impossible. A proper refinement would have been obtained by requiring that $\pi_a(\gamma_a) < 1$.

Orthogonality of the integral probability attribute

The interpretation of the integral probability attribute (and the uncertainty attribute) in terms of the execution model is defined independently of the interpretation of the information, time and location attributes. Consequently, the probability function π and the information, time and location functions ι, τ and λ define orthogonal constraints on the executions of a behaviour. A distinction is, however, that the probability function defines constraints on sets of executions, whereas the information, time and location functions define constraints on individual executions.

The above property allows one to define the integral probability attribute (and the uncertainty attribute) independently of the information, time and location attributes. Furthermore, this property guarantees that the execution model supports the independent (orthogonal) development and union of the integral probability module and the information, time and location references modules identified in Section 2.5.4.

3.8.3 Modelling of the stochastic probability attribute

The stochastic probability attribute of an action *a* defines, for each condition γ that allows the occurrence of this action, a probability distribution function. This probability distribution function is denoted as $\pi_a(\gamma, \tau)$, and defines the probability that *a* occurs within time interval $(\tau_\gamma..\tau]$, i.e., $\tau_\gamma < \tau_a \leq \tau$, when assuming that γ enables *a* from moment τ_γ, and γ is satisfied when *a* occurs. Consequently, the stochastic probability attribute is a refinement of the integral probability attribute, in which the integral probability is distributed over the time period in which an action is enabled by a certain condition. The correspondence between the integral probability $\pi_a(\gamma)$ and the stochastic probability $\pi_a(\gamma, \tau)$ is therefore defined by the equation $\pi_a(\gamma) = \lim_{\tau\to\infty} \pi_a(\gamma, \tau)$.

Based on the above, the stochastic probability attribute is modelled in terms of the execution model by relating the probability function π and the time function τ. The time function is used to model the time intervals between the occurrence of an action *a* and the moment the condition of *a* enables the occurrence of *a* in terms of distinct executions. The probability function is used to define the probability of these executions as follows.

Consider a behaviour E in which action a is defined. The stochastic probability $\pi_a(\gamma, \tau)$ is equal to the ratio between the probability of execution set $E\backslash a \wedge \gamma \wedge \tau$ and the probability of execution set $E\backslash\gamma$, where

- $E\backslash a \wedge \gamma \wedge \tau$ represents the restriction of E to the (sub)set of executions in which action a occurs within τ time units after condition γ enables the occurrence of a;
- $E\backslash\gamma$ represents the restriction of E to the (sub)set of executions in which condition γ enables the occurrence of a (and action a either occurs or does not occur).

Consequently, the stochastic probability attribute is modelled by defining probability function π_E such that:

$$\pi_E(E\backslash a \wedge \gamma \wedge \tau) \,/\, \pi_E(E\backslash\gamma) = \pi_a(\gamma, \tau) \,,$$

for all actions a in E and all conditions γ that allow the occurrence of a.

The modelling of the stochastic probability attribute as explained above is illustrated by means of two examples. For convenience, we only consider the discrete time case. Therefore, time domain $\boldsymbol{T}$ is restricted to discrete time domain $\boldsymbol{Z} \subset \boldsymbol{T}$, and we assume a sampling period of 1, such that: $\boldsymbol{Z} = \{.., -1, 0, 1, 2, ..\}$ represents a two-sided infinite discrete time axis.

Furthermore, we use probability *density* functions instead of probability distribution functions in the examples below. We believe this is more intuitive, since in the discrete time case a probability density function models the probability that an action occurs *at* a certain time moment. Figure 3.20 depicts four probability density functions, which are used in the examples below. For example, density function $\pi_b(\gamma_b, \tau)$ defines that action b occurs 1 time unit after γ_b enables b with probability 0.07, 2 time units after γ_b enables b with probability 0.14, etc.

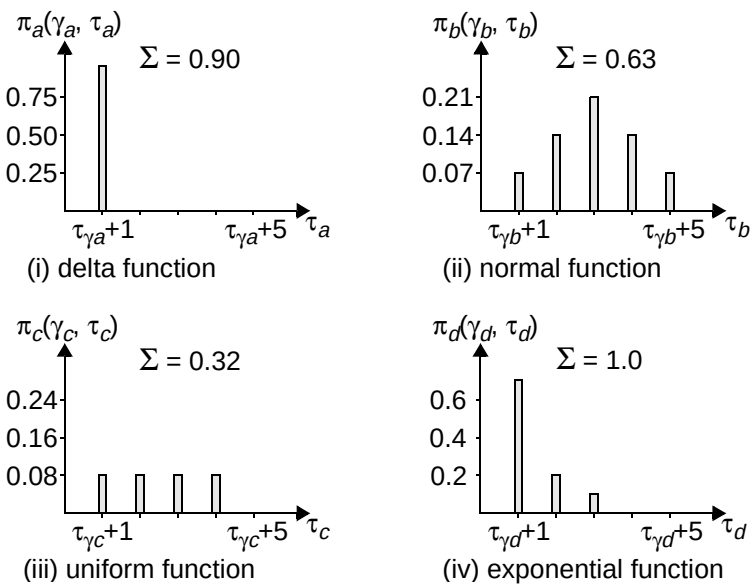


Figure 3.20: Probability density functions

Example 1

Figure 3.21 illustrates the modelling of the stochastic probability attributes of two actions *a* and *b*, which consist of the probability density functions $\pi_a(\gamma_a, \tau_a)$ and $\pi_b(\gamma_b, \tau_b)$ as defined in Figure 3.20, respectively, where condition γ_a represents that *a* is allowed to occur from the beginning of an execution, and condition γ_b represents that *b* is allowed to occur after *a* has occurred.

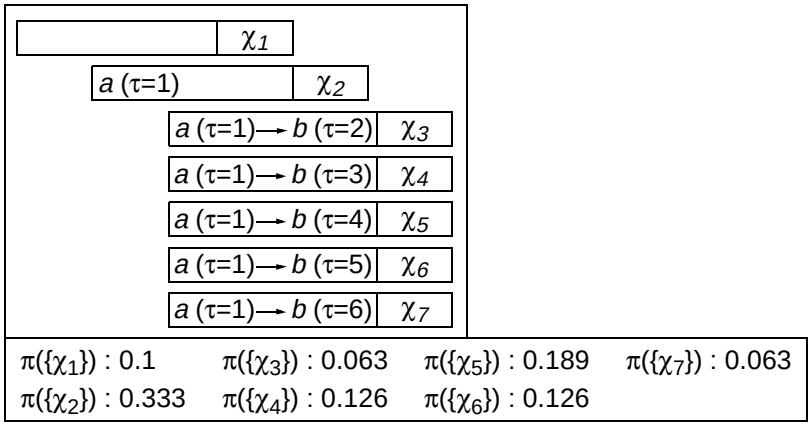


Figure 3.21: Modelling of stochastic probability attribute

The probability values $\pi(\{\chi_1\})$ and $\pi(\{\chi_3\})$ are derived using the following definitions, with $E = \{\chi_1, .., \chi_7\}$:

- $\pi_a(\gamma_a, 1) = \pi(E \backslash a \wedge \gamma_a \wedge 1) / \pi(E \backslash \gamma_a) = \pi(\{\chi_2, .., \chi_7\}) / \pi(E) = 0.9$.
  This equation implies that $\pi(\{\chi_1\}) = 0.1$ and $\pi(\{\chi_2, .., \chi_7\}) = 0.9$;
- $\pi_b(\gamma_b, 1) = \pi(E \backslash b \wedge \gamma_b \wedge 1) / \pi(E \backslash \gamma_b) = \pi(\{\chi_3\}) / \pi(\{\chi_2, .., \chi_7\}) = 0.07$.
  This and the above equations imply that $\pi(\{\chi_3\}) = 0.063$.

Example 2

Figure 3.22 depicts a refinement of the behaviour in Figure 3.17. The stochastic probability attributes of actions *a*, *b*, *c*, *d* and *e* consist of the probability density functions $\pi_a(\gamma_a, \tau_a)$, $\pi_b(\gamma_b, \tau_b)$, $\pi_c(\gamma_c, \tau_c)$, $\pi_d(\gamma_d, \tau_d)$ and $\pi_e(\gamma_e, \tau_e) = \pi_a(\gamma_a, \tau_a)$ as defined in Figure 3.20, respectively, where:

- condition γ_a represents that *a* is allowed to occur from the beginning of an execution;
- condition γ_b represents that *b* is allowed to occur after *a* has occurred and *c* does not occur;
- condition γ_c represents that *c* is allowed to occur after *a* has occurred and *b* does not occur;
- condition γ_d represents that *d* is allowed to occur after *b* has occurred; and
- condition γ_e represents that *e* is allowed to occur after *c* has occurred.

Executions $\chi_{5.x.y}$, $\chi_{3.x}$ and $\chi_{6.x.y}$ are refinements of executions χ_5, χ_3 and χ_6 in Figure 3.17, which model the time intervals between action occurrences, respectively. The probability

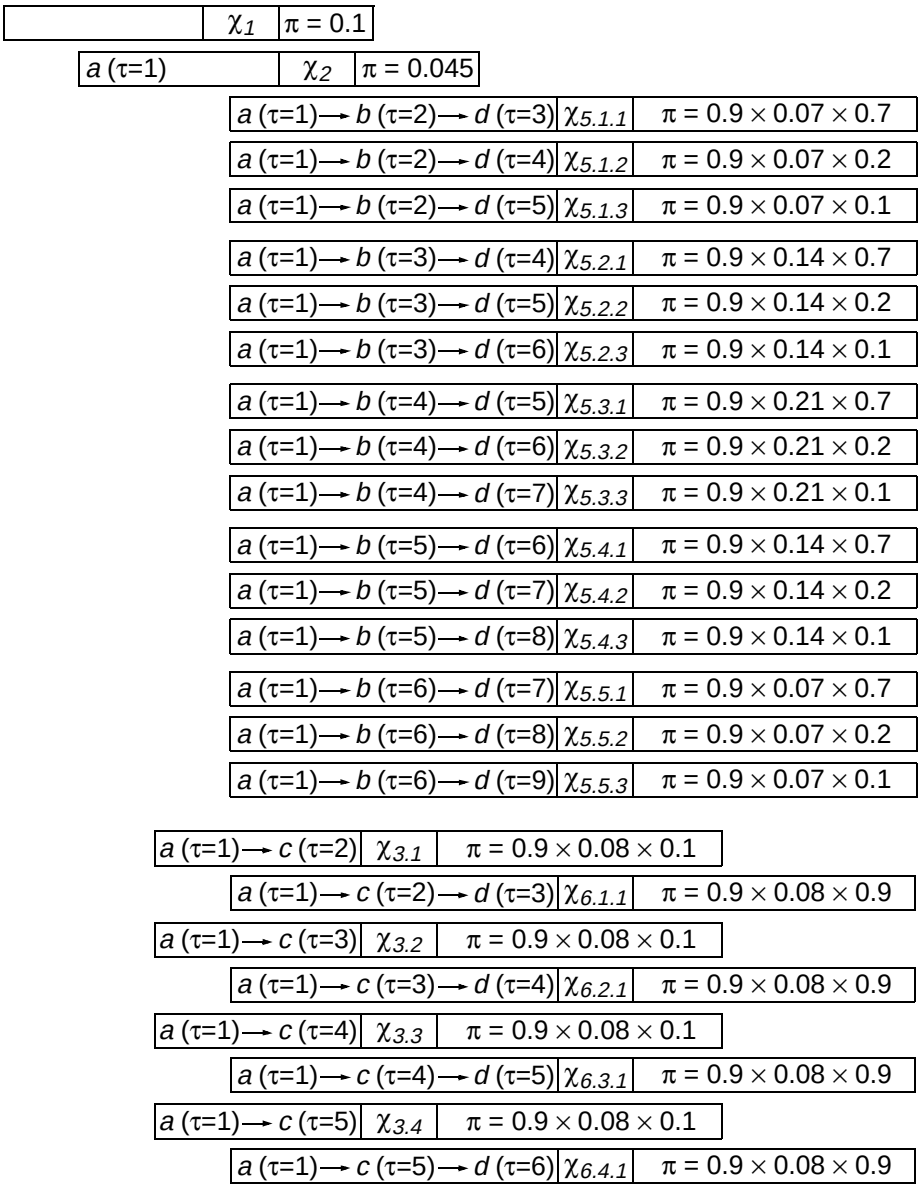


Figure 3.22: Modelling of stochastic probability attribute (2)

values of these executions are represented in separate boxes that are attached to the corresponding executions. The expressions used to represent these probability values reflect how these values can be derived from the behaviour in Figure 3.17 and the probability density functions in Figure 3.20. This derivation is left as an exercise to the reader.

Modelling of different types of ‘stochastic’ probability attributes

Similar to the stochastic probability attribute, the execution model supports the modelling of other types of ‘stochastic’ probability attributes in which the probability function π is related to the information function ι, the location function λ, or a particular combination of the information, time and location functions. For example, variations of the ‘stochastic’ probability attribute may be defined in which the probability of the occurrence of an action

is distributed over the set of information values that can be established by this action, or is distributed over the set of locations at which this action makes its result available.

3.8.4 Formal definition

The refinement of the notion of execution-based behaviour is formally defined below.

Definition 3.28 An execution-based behaviour *EP* is defined as a tuple $\langle E, \pi \rangle$, where

- E is a set of executions as defined in Definition 3.2;
- $\pi : \wp(E) \rightarrow \wp(\boldsymbol{P})$ is a probability function, which defines the possible probability values of a set of executions.

The following consistency requirements must be obeyed by probability function π:

- $\pi(E') = \Sigma \{\pi(\{\chi\}) \mid \chi \in E'\}$, with $E' \subseteq E$;
- $\pi(E) = 1$. ■

Execution-based behaviour *EP* ranges over ***EP***, the domain of execution-based behaviours refined with a probability function.

The prefix relation between executions is represented by the symbol $\angle_\chi$. This relation is defined below.

Definition 3.29 The prefix relation $\angle_\chi : \boldsymbol{X} \times \boldsymbol{X}$ is defined as:

$$\langle A_1, \overline{A}_1, <_1, =_1 \rangle \angle_\chi \langle A_2, \overline{A}_2, <_2, =_2 \rangle \Leftrightarrow \begin{array}{l} A_1 \cup \overline{A}_1 = A_2 \cup \overline{A}_2 \ \wedge \\ A_1 \subset A_2 \ \wedge \\ <_1 = <_2 \cap (A_1 \times A_1) \wedge \ =_1 = =_2 \cap (A_1 \times A_1). \end{array}$$ ■

3.9 Formal definition

This section presents the proofs of some properties of relation < and = that are used in the proof of Property 3.17.

Property 3.30 $=_{(1\times2)\times3} = =_{1\times(2\times3)}$.

Proof:

$$\begin{aligned} =_{(1\times2)\times3} &= (=_{1\times2} \cup =_3)^+ \\ &= ((=_1 \cup =_2)^+ \cup =_3)^+ \\ &= (=_1 \cup =_2 \cup =_3)^+ \\ &= (=_1 \cup (=_2 \cup =_3)^+)^+ \\ &= =_{1\times(2\times3)} . \end{aligned}$$ ■

Property 3.31 $<_{(1\times2)\times3} = <_{1\times(2\times3)}$.

Proof:

$$<_{(1\times2)\times3} = (<_{1\times2} \cup <_3 \cup (<_{1\times2} \cup <_3) \bullet =_{(1\times2)\times3} \cup =_{(1\times2)\times3} \bullet (<_{1\times2} \cup <_3) \cup$$

$$
\begin{aligned}
& =_{(1\times2)\times3} \bullet (<_{1\times2} \cup <_3) \bullet =_{(1\times2)\times3})^+ \\
= \; & (<_1 \cup <_2 \cup (<_1 \cup <_2) \bullet =_{1\times2} \cup =_{1\times2} \bullet (<_1 \cup <_2) \cup =_{1\times2} \bullet (<_1 \cup <_2) \bullet =_{1\times2} \cup \\
& <_3 \cup \\
& (<_1 \cup <_2 \cup (<_1 \cup <_2) \bullet =_{1\times2} \cup =_{1\times2} \bullet (<_1 \cup <_2) \cup \\
& \quad =_{1\times2} \bullet (<_1 \cup <_2) \bullet =_{1\times2}) \bullet =_{(1\times2)\times3} \cup \\
& <_3 \bullet =_{(1\times2)\times3} \cup \\
& =_{(1\times2)\times3} \bullet (<_1 \cup <_2 \cup (<_1 \cup <_2) \bullet =_{1\times2} \cup =_{1\times2} \bullet (<_1 \cup <_2) \cup \\
& \quad =_{1\times2} \bullet (<_1 \cup <_2) \bullet =_{1\times2}) \cup \\
& =_{(1\times2)\times3} \bullet <_3 \cup \\
& =_{(1\times2)\times3} \bullet (<_1 \cup <_2 \cup (<_1 \cup <_2) \bullet =_{1\times2} \cup =_{1\times2} \bullet (<_1 \cup <_2) \cup \\
& \quad =_{1\times2} \bullet (<_1 \cup <_2) \bullet =_{1\times2}) \bullet =_{(1\times2)\times3} \cup \\
& =_{(1\times2)\times3} \bullet <_3 \bullet =_{(1\times2)\times3})^+ \\
= \; & (<_1 \cup <_2 \cup (<_1 \cup <_2) \bullet =_{1\times2} \cup =_{1\times2} \bullet (<_1 \cup <_2) \cup =_{1\times2} \bullet (<_1 \cup <_2) \bullet =_{1\times2} \cup \\
& <_3 \cup \\
& (<_1 \cup <_2 \cup =_{1\times2} \bullet (<_1 \cup <_2)) \bullet =_{(1\times2)\times3} \cup \\
& <_3 \bullet =_{(1\times2)\times3} \cup \\
& =_{(1\times2)\times3} \bullet (<_1 \cup <_2 \cup (<_1 \cup <_2) \bullet =_{1\times2}) \cup \\
& =_{(1\times2)\times3} \bullet <_3 \cup \\
& =_{(1\times2)\times3} \bullet (<_1 \cup <_2) \bullet =_{(1\times2)\times3} \cup \\
& =_{(1\times2)\times3} \bullet <_3 \bullet =_{(1\times2)\times3})^+ \\
= \; & (<_1 \cup <_2 \cup <_3 \cup \\
& (<_1 \cup <_2) \bullet =_{(1\times2)\times3} \cup <_3 \bullet =_{(1\times2)\times3} \cup \\
& =_{(1\times2)\times3} \bullet (<_1 \cup <_2) \cup =_{(1\times2)\times3} \bullet <_3 \cup \\
& =_{(1\times2)\times3} \bullet (<_1 \cup <_2) \bullet =_{(1\times2)\times3} \cup =_{(1\times2)\times3} \bullet <_3 \bullet =_{(1\times2)\times3})^+ \\
= \; & (<_1 \cup <_2 \cup <_3 \cup \\
& (<_1 \cup <_2 \cup <_3) \bullet =_{(1\times2)\times3} \cup =_{(1\times2)\times3} \bullet (<_1 \cup <_2 \cup <_3) \cup \\
& =_{(1\times2)\times3} \bullet (<_1 \cup <_2 \cup <_3) \bullet =_{(1\times2)\times3})^+ \\
= \; & (<_1 \cup <_2 \cup <_3 \cup \\
& (<_1 \cup <_2 \cup <_3) \bullet =_{1\times(2\times3)} \cup =_{1\times(2\times3)} \bullet (<_1 \cup <_2 \cup <_3) \cup \\
& =_{1\times(2\times3)} \bullet (<_1 \cup <_2 \cup <_3) \bullet =_{1\times(2\times3)})^+ \\
= \; & <_{1\times(2\times3)} \qquad \blacksquare
\end{aligned}
$$

3.10 Related work

The mathematical structure that underlies the execution concept corresponds to a partially ordered set (poset) of action occurrences extended with a synchronization relation. Posets are commonly used to model the behaviours of distributed systems in which actions may be independent (real parallelism). Such systems are often denoted as *concurrent systems*.

In [37, 43], Katoen and Langerak use (families of) labelled partially ordered sets (lposets) to define the semantics of event structures. An *lposet* is defined as a triple $\langle E, \leq, l \rangle$, with E defining a set of events, $\leq \subseteq E \times E$ defining a partial order on E, and $l : E \rightarrow \boldsymbol{A}$ is an action labelling function, where $\boldsymbol{A}$ denotes a set of actions. An event e represents the occurrence of action $l(e)$. Distinct events may represent distinct occurrences of the same action. A family of lposets is a non-empty set of (finite) lposets that is downwards closed with respect to the prefix ordering on lposets.

In [55], Pratt uses a partially ordered multiset (pomset) as the basic concept to define a rich language for modelling concurrent system behaviours. This language comprises many algebraic and logical operators on pomsets. A *pomset* is an isomorphism class of lposets. In this thesis, an action may occur only once, since we assume that an action is an abstraction of a single instance of an activity. This implies that an action in this thesis corresponds to an event in languages based on lposets or pomsets.

In [22], Gaifman uses the basic concept of computation to model concurrent behaviours. A *computation* is defined as a four-tuple $\langle V, <^c, <^t, \mu\rangle$, where V is a set of events, μ is a labelling function (similar to l above), and partial orders $<^c$ and $<^t$ represent a *causal* precedence relation and a *temporal* precedence relation, such that $<^c \subseteq <^t$, respectively. Two events e_1 and e_2 are causally related iff $e_1 <^c e_2$ or $e_2 <^c e_1$, and are independent (concurrent) otherwise. Two events e_1 and e_2 occur simultaneously iff neither $e_1 <^t e_2$ nor $e_2 <^t e_1$.

The temporal precedence relation $<^t$ allows one to define the temporal ordering of independent action occurrences. The temporal ordering of related actions is implied by the causal precedence relation, i.e., $e_1 <^c e_2$ implies $e_1 <^t e_2$. In our opinion, one is not completely free in defining the temporal precedence relation on independent actions. This is illustrated by the behaviour $\{\langle\{a, b\}, \emptyset, \{\langle a, b\rangle\}, \mathrm{I}\rangle\}$, where I represents the identity relation. This behaviour consists of a single computation in which actions a and b are independent, and the occurrence of a always precedes the occurrence of b. However, in order to guarantee precedence relation $a <^t b$ in any behaviour run one must relate actions a and b somehow, which implies that $a <^c b$, unless the behaviour's environment has been made responsible for the ordering between a and b.

Gaifman argues in [22] that concurrent actions can be performed in either order, or simultaneously. This leads to the conclusion that it is sufficient to define the causal precedence relation only, and remove the temporal precedence relation. We agree with this conclusion, since from a prescriptive perspective we are not interested in the temporal ordering of independent actions. We consider independent actions as actions which have no relation, and therefore we do not want to bring them into relation by considering their possible orderings.

One should be careful, however, in defining that a behaviour consisting of the occurrence of two independent actions a and b is equivalent to a behaviour in which actions a and b can occur in either order, or simultaneously. This property may hold in case time is not explicitly modelled. However, in case time constraints can be modelled, the above is not necessarily true. For example, consider the behaviour depicted in Figure 3.23, using our design model. Despite that actions a and b are independent, their relation with action c enforces that action a always occurs before action b.

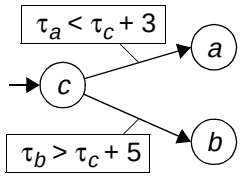


Figure 3.23: Independent actions with time constraints

We conclude that the temporal precedence relation $<^t$ is not useful from a prescriptive perspective. Instead, it may be used to represent the ordering of independent actions from an observational perspective. In that case, one should however question whether it is always possible to observe the difference between related and independent actions.

3.11 Conclusions

In this chapter we introduce the execution model. In this model, a behaviour is defined by enumerating all possible executions of this behaviour. The term execution-based behaviour definition is used to denote a behaviour defined in terms of the execution model.

An execution represents the outcome of a possible run of a system that performs a specified behaviour. This outcome comprises the actions that have occurred, the information, time and location values that have been established in these actions, and how action occurrences are related in the particular execution. An execution also gets one or more probability values, which represent the probability that this execution is the outcome of a system run. In this respect, a behaviour is considered an experiment and an execution is considered a possible outcome of this experiment. The sum of the probability of all possible executions of a behaviour is equal to 1.

We present two techniques to compose an execution-based behaviour from multiple smaller execution-based behaviours. Both techniques are denoted as constraint-oriented composition techniques, since they consider executions as defining constraints on the execution of the involved actions. One of these techniques is used to define the formal semantics of causality relations in Chapters 4 to 7. This technique is based on the notion of partial execution. In contrast to a regular execution, a partial execution can leave undefined whether two actions are related or independent. This property allows one to define an execution-based behaviour E as the cross-conjunction of multiple smaller behaviours E_i, each of which defines constraints on the execution of a subset of the actions in E. The cross-conjunction operation guarantees that E represents the conjunction of the constraints represented by all behaviours E_i.

The mathematical structure of the execution concept is based on partially ordered sets (posets) extended with a synchronization relation. Posets are commonly used in the literature for the modelling of concurrent system behaviours. We have used the abstraction hierarchy of Section 2.5.3 as a framework to develop the execution concept, in order to ensure that this concept supports the modelling of all relevant characteristics of action relations we have identified, and thus is sufficiently expressive to define the formal semantics of causality relations.

Chapter 4

Action relations

This chapter introduces the concept of causality relation as a basic building block to model action relations. We focus on the modelling of the basic characteristics of action relations that have been identified in our abstraction hierarchy in Section 2.5.3, including temporal ordering and the uncertainty attribute.

The causality relation concept allows an elaborate and systematic analysis of the various types of temporal ordering relations that can be defined between actions. The aim of this analysis is to identify a limited set of basic and generic concepts to model these relations. In addition, this analysis is used to identify rules to combine the identified concepts into consistent and more complex behaviour models. In this chapter, we only consider behaviours consisting of two actions.

The structure of this chapter is as follows. Section 4.1 introduces the causality relation concept and explains our approach towards the elaboration of this concept. Section 4.2 presents the technique we use to define the formal semantics of causality relations. Section 4.3 defines a limited set of basic causality relations. Section 4.4 introduces two composition operators to define composite causality relations. Section 4.5 presents rules to combine the causality relations of two actions into consistent behaviour models. Section 4.6 structures all possible behaviour models of two actions into a hierarchy of behaviour families. Section 4.7 introduces the notion of alternative behaviour as a means to structure behaviour models into simpler behaviour models. And Section 4.8 presents the conclusions.

4.1 Introduction

The causality relation concept is introduced as the basic building block for the modelling of relations between actions. We explain our approach towards the development of this concept.

4.1.1 The causality relation concept

We model relations between actions as a composition of causality relations. A *causality relation* defines for an individual action, which is called the *result action*, the condition for the occurrence of this action. This condition consists of:

- a *causality condition*, which defines how the occurrence of the result action depends on the occurrences or non-occurrences of other actions;

- *action attribute constraints*, which define how the occurrence of the result action and, possibly, the information, time and location attribute values of the result action, depend on the information, time and location attribute values established by actions in the causality condition; and
- a *probability attribute*, which defines the probability of the occurrence of the result action when the causality condition and action attribute constraints are satisfied.

Satisfaction of the causality condition and the action attribute constraints is a necessary condition for the occurrence of the result action. A discussion on action attribute constraints is deferred to Chapter 6.

A causality relation generally defines only a part of the relation between two or more actions. For example, to model the relation between two actions a and b by causality relations, this relation has to be decomposed into the dependency of a on b and the dependency of b on a, which are defined by the causality relations of a and b, respectively.

Causality conditions

A causality condition defines how the occurrence of an action depends on the occurrences or non-occurrences of other actions. This condition is defined in terms of actions that must occur or must not occur to allow the occurrence of the result action. For example, the causality condition of action a may define that action b must have occurred before a can occur.

The causality condition of an action consists of one or more alternative conditions, such that the satisfaction of any alternative condition is sufficient to allow the occurrence of this action. Therefore, the causality condition of an action is satisfied if and only if at least one of its alternative conditions is satisfied.

We assume that an alternative condition is minimal in the sense that this condition is not again composed of multiple alternative (sub-)conditions: an alternative condition is either an elementary condition or is a conjunction of elementary conditions, such that the alternative condition is satisfied if and only if all elementary conditions are satisfied.

The following types of causality conditions are then distinguished:

- *conjunctive* causality conditions, which consist of the conjunction of two or more elementary causality conditions; and
- *disjunctive* causality conditions, which consist of the disjunction of two or more conjunctive causality conditions.

The causality condition of a result action is either an elementary, a conjunctive or a disjunctive causality condition. This implies that causality conditions are defined in disjunctive normal form. The conjunctive elements of a causality condition are called *alternative* causality conditions.

For example, the causality condition of action a defines that action b must occur or actions c and d must occur to allow the occurrence of a, which is denoted as $b \vee (c \wedge d) \rightarrow a$. Causality

condition $b \vee (c \wedge d)$ consists of two alternative causality conditions: the alternative condition that b must occur and the alternative condition that c and d must occur. Consequently, the occurrence of b or the occurrences of c and d, or both, allow the occurrence of a. Alternative condition $c \wedge d$ consists of a conjunction of two elementary causality conditions: the elementary condition that c must occur and the elementary condition that d must occur. These elementary conditions are not alternative conditions, since both conditions must be satisfied in order to allow the occurrence of a.

Uncertainty attribute

We define that the satisfaction of an alternative causality condition of an action *allows* the occurrence of this action, but does not enforce its occurrence. Whether an action actually occurs or not when an alternative causality condition is satisfied is defined by the probability attribute of this action. From the three types of probability attributes introduced in Section 2.5.2, this chapter only considers the uncertainty attribute to denote the possibility that an action does or does not occur.

The *uncertainty attribute* defines for each alternative causality condition the (relative) uncertainty that an action occurs when this condition is satisfied. One of two possible uncertainty values can be associated with an alternative causality condition:

- the *must* value, defines that the result action must occur when the associated alternative condition is satisfied; or
- the *may* value, defines that the result action may or may not occur when the associated alternative condition is satisfied.

In the above example, one may define that action a may occur when action b occurs, but must occur when actions c and d occur. This implies that no execution is possible in which c and d occur and a does not occur. However, in case c or d (or both) do not occur while b occurs, two executions are possible: one in which a occurs and one in which a does not occur.

Since we define that the satisfaction of an alternative causality condition only allows the occurrence of the result action, but does not demand its occurrence, the may uncertainty value does not impose an additional constraint on the occurrence of the result action. This choice is also denoted as the *may-interpretation of causality conditions*. Instead, the must uncertainty value defines an additional constraint, because the result action has to occur in an execution when the associated alternative condition is satisfied.

The uncertainty attribute of an action models the uncertainty that the activity represented by this action happens, despite that its causality condition is satisfied. It allows one to abstract from the (precise) conditions that cause the non-occurrence, or disable the occurrence of an action. Examples of its use are the modelling of unreliability of implementations and the modelling of non-determinism.

Textual design notation

The following two textual notations are used to represent the causality relation of an action:

$\gamma 1_{\mu 1} \vee ... \vee \gamma n_{\mu n} \rightarrow a$
or
$\gamma 1 \vee ... \vee \gamma n \rightarrow a\ [\upsilon(\gamma 1) = \mu 1, .., \upsilon(\gamma n) = \mu n]$

where

- *a* is an action name which identifies result action *a*;
- $\gamma 1, .., \gamma n$ are alternative causality conditions of action *a*;
- $\mu 1, .., \mu n$ are the uncertainty values associated with $\gamma 1, .., \gamma n$, respectively; and
- υ is the uncertainty attribute of the relation between action *a* and its alternative causality conditions;
- $\vee$ is the *or*-operator, which represents the disjunction of alternative causality conditions.

The first notation represents uncertainty values as subscripts of their corresponding alternative causality conditions. The second notation represents the association of uncertainty values with alternative causality conditions between square brackets after the result action. The symbols '!' and '?' may be used as shorthands to represent the *must* and *may* uncertainty values, respectively.

The symbol $\rightarrow$ symbolizes the "cause-effect" relationship between the satisfaction of the causality condition (the cause) and the occurrence of action *a* (the effect). The adjective 'causality' in the term causality condition also refers to this cause-effect relationship. In other words, the causality condition of action *a* defines how this action causally depends on other actions, by defining how the occurrences or non-occurrences of these actions may cause the occurrence of *a*.

Behaviour definitions

A behaviour consists of one or more actions and their relations. Since we have chosen to model action relations in terms of causality relations, a behaviour is defined by a set of causality relations, one per action of the behaviour.

Causality conditions explicitly define how result actions depend on other actions of the behaviour. This implies that the independence of actions is defined implicitly by the absence of (the explicit definition of) a dependency between these actions. Actions are assumed to be independent when they are not related by their causality conditions (i.e., when a relation is absent).

Since there are situations in which we want to denote explicitly the actions that do not depend on other actions, a special causality condition is introduced: the start condition. The *start condition* is always satisfied from a certain time moment determined by the behaviour's environment. The term start condition refers to its typical use as the causality condi-

tion of initial actions, which are allowed to occur from the beginning (start) of a behaviour. The start condition is represented by the symbol √.

The textual notation of a behaviour definition consists of, successively, a behaviour identifier, the symbol '=', and the set of causality relations delimited by the set symbols '{' and '}'. For example, behaviour $B = \{\surd_{?} \rightarrow a, \surd_{?} \rightarrow b\}$ defines two initial actions *a* and *b*, which may (not) occur in an execution. Alternatively, this behaviour is represented as : $B = \{\surd \rightarrow a\ [\upsilon(\surd) = ?], \surd \rightarrow b\ [\upsilon(\surd) = ?]\}$. Since no dependency is defined between both actions, they may occur independently of each other. This renders the following four possible executions of behaviour *B*: $\boxed{a\quad b}$, $\boxed{a\quad \overline{b}}$, $\boxed{\overline{a}\quad b}$, $\boxed{\overline{a}\quad \overline{b}}$.

Figure 4.1 depicts behaviour *B* in terms of our graphical design notation. The start condition of an initial action is graphically represented by a solid arrow that points to the initial action and is not connected to any other action. Uncertainty values are represented in boxes, which are linked to the corresponding alternative causality condition.

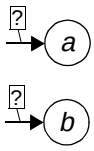


Figure 4.1: Independence of actions *a* and *b*

4.1.2 Development of the causality relation concept

The remainder of this chapter presents a step-wise development of the causality relation concept. The purpose of presenting the development steps, and not just the final result, is to identify, explain and motivate the modelling choices that underlie the definition of this concept. The following development steps are distinguished:

1. *Identification of basic causality conditions*. Section 4.3 identifies and defines a complete and minimal set of basic causality conditions that are necessary to model temporal ordering relations between actions. *Basic causality conditions* are the most elementary causality conditions that are distinguished to model relations.

   Furthermore, all possible combinations (associations) of uncertainty values with basic causality conditions are discussed. A causality relation which defines a basic causality condition and its associated uncertainty value is called a *basic causality relation*.

2. *Introduction of composition operators*. Section 4.4 presents two composition operators to build more complex causality conditions from basic causality conditions. The *and*-operator is introduced to define conjunctions of causality conditions and the *or*-operator is introduced to define alternative causality conditions. Compositions of (basic) causality conditions are called *composite causality conditions*.

   Furthermore, the association of uncertainty values with alternative causality conditions is discussed. A causality relation which defines a composite causality condition and its associated uncertainty value(s) is called a *composite causality relation*.

3. *Identification of combination rules*. Section 4.5 presents rules to combine the causality relations of two actions into consistent behaviour models. These rules are needed because the arbitrary combination of causality relations may render inconsistent behaviour models.

4.1.3 Pre-formal design notation

We define the formal semantics of causality relations by a function which performs a mapping of expressions in our design notation, representing (parts of) causality relations, onto expressions in the notation of the execution model, representing the executions allowed by these causality relations. This function is composed of two sub-functions:

- function $[\![\ ]\!]_{\zeta}$, which performs a mapping of expressions in our design notation onto expressions in a pre-formal design notation. This notation is an alternative to represent (parts of) causality relations;
- function $[\![\ ]\!]_{\chi}$, which performs a mapping of expressions in the pre-formal design notation onto expressions in the notation of the execution model.

Figure 4.2 illustrates this, where symbols $L_{causality}$, $L'_{causality}$ and $L_{execution}$ represent the notation of our design model, the pre-formal notation of our design model, and the notation of the execution model, respectively.

$$L_{causality} \xrightarrow[\zeta]{[\![\ ]\!]} L'_{causality} \xrightarrow[\chi]{[\![\ ]\!]} L_{execution}$$

Figure 4.2: Definition of formal semantics

$L'_{causality}$ is introduced as an intermediate notation between $L_{causality}$ and $L_{execution}$, in order to facilitate the definition of the formal semantics of causality relations. $L'_{causality}$ represents (parts of) causality relations in terms of simple mathematical structures, which are easier to manipulate than expressions in $L_{causality}$. Therefore, $L'_{causality}$ is only used in the sections containing formal definitions to define the formal semantics of causality relations.

$L_{causality}$ is more concise and intuitive. Therefore, this notation is used in all other sections of this thesis to explain and illustrate the development of our design concepts. Function $[\![\ ]\!]_{\zeta}$ is only defined informally in this chapter, because this function is straightforward for behaviours of two actions. Part of notation $L'_{causality}$ is introduced below, and is elaborated in the sections of this thesis containing formal definitions.

Definition 4.1 The *causality relation* of an action a is defined as a triple $\langle a, \Gamma, \upsilon \rangle$, where

- $a \in \boldsymbol{A}$, is the result action;
- $\Gamma \in \boldsymbol{CC}$, is the causality condition of a; and
- $\upsilon : \{a\} \times \Gamma \rightarrow \boldsymbol{U}$, is the uncertainty attribute of a. ■

Domain $\boldsymbol{C}$ denotes the universe of elementary and conjunctive causality conditions, and is ranged over by γ. Domain $\boldsymbol{CC} = \wp(\boldsymbol{C}) - \{\varnothing\}$ denotes the universe of (disjunctive) causality conditions, and is ranged over by Γ. The empty set is excluded from this domain, since the absence of a condition (dependency) is represented by the start condition.[1] The conjunctive

elements of a causality condition Γ are called alternative causality conditions.

The *or*-operator $\vee$ in $L_{causality}$ is (implicitly) represented by the union operator $\cup$ in $L'_{causality}$. For example, the expression $\Gamma = \gamma 1 \vee ... \vee \gamma n$ in $L_{causality}$ corresponds to the expression $\Gamma = \{\gamma 1\} \cup .. \cup \{\gamma n\} = \{\gamma 1, .., \gamma n\}$ in $L'_{causality}$. For reasons of convenience and conciseness, the operands of the *or*-operator do not have to be represented as sets of alternative causality conditions in $L_{causality}$.

The uncertainty attribute of action a is defined in terms of a function which associates an uncertainty value with each alternative causality condition γ in Γ. The domain of uncertainty values is denoted as $\boldsymbol{U}$, with $\boldsymbol{U} = \{must, may\}$. The association of an uncertainty value with an alternative causality condition of some result action, is called an *uncertainty association*. The uncertainty attribute of an action consists of a set of uncertainty associations, one per alternative causality condition, and is ranged over by φ.

Symbol $\boldsymbol{CR}$ denotes the domain of causality relations and is ranged over by ρ. The causality condition, uncertainty attribute and causality relation of an action a are denoted as Γ_a, υ_a, and ρ_a, or alternatively as $\Gamma(a)$, $\upsilon(a)$, and $\rho(a)$, respectively. Function $Ac : \boldsymbol{CR} \rightarrow \boldsymbol{A}$ denotes the result action of a causality relation, such that: $Ac(\langle a, \Gamma, \upsilon\rangle) = a$.

The term *causality-based* behaviour definition is introduced as the counterpart of the term execution-based behaviour definition, in order to denote behaviours that are defined in terms of causality relations. The adjective ‘causality-based’ is omitted when this is clear from the context.

Definition 4.2 A *causality-based behaviour definition* B consists of a set of causality relations, with one causality relation being defined for each action of B, and $B \subset \boldsymbol{CR}$. ■

Symbol $\boldsymbol{B}$ denotes the domain of causality-based behaviour definitions, and is ranged over by B. Function $Ac : \boldsymbol{B} \rightarrow \wp(\boldsymbol{A})$ denotes the actions of a behaviour, such that: $Ac(B) = \{Ac(\rho) \mid \rho \in B\}$. Optionally, this set can be defined as a parameter of B, e.g., $B(\{a, b, c\})$.

4.2 Execution semantics

This section explains the technique we use to define the formal semantics of causality relations in terms of the execution model of Chapter 3. This semantics is called *execution semantics*.

4.2.1 Execution semantics of causality relations

A causality relation imposes a set of alternative constraints on the execution of the involved actions. Each alternative constraint allows a distinct execution. Therefore, the semantics of a causality relation ρ can be represented by an execution set $E(Ac)$, where Ac is restricted to

1. An alternative modelling choice is to define $\Gamma = \{\surd\}$ and $\Gamma = \varnothing$ as being identical. This choice would disallow, however, the definition of other alternative causality conditions in addition to the start condition.

the actions that are involved in ρ, and $E(Ac)$ consists of the possible executions of actions Ac that are allowed by ρ.

The execution set $E(Ac)$ only constrains the execution of the actions in Ac, but does not constrain other actions of the behaviour. Constraints on the execution of other actions are defined by other causality relations of the behaviour.

For example, consider the causality relation of action a which defines that a may occur after another action b has occurred. This causality relation allows the following executions: $E(\{a, b\}) = \{ \boxed{b \rightarrow a}, \boxed{b \quad \bar{a}}, \boxed{\bar{b} \quad \bar{a}} \}$. In other words, this causality relation constrains the execution of actions a and b, which is possibly part of a larger behaviour involving other actions. The only executions allowed are those in which either b occurs before a, or b occurs and a does not occur, or neither a nor b occurs.

The possible executions of a behaviour B are determined by the sets of alternative constraints that are imposed by the causality relations of all actions in B. Each execution of B must be allowed by at least one alternative constraint of each individual causality relation in B. Therefore, the execution set of a behaviour B is equal to the cross-conjunction of the sets of executions that are allowed by the individual causality relations in B. The cross-conjunction operation is explained in Section 3.6.

For example, consider a behaviour B of three actions a, b and c, in which a is an initial action which may occur, action b may occur after a has occurred, and c must occur after b has occurred. The sets of alternative constraints that are imposed by the causality relations of a, b and c are represented by execution sets: $E_1(\{a\}) = \{\boxed{a}, \boxed{\bar{a}}\}$, $E_2(\{a, b\}) = \{\boxed{a \rightarrow b}, \boxed{a \quad \bar{b}}, \boxed{\bar{a} \quad \bar{b}}\}$, and $E_3(\{b, c\}) = \{\boxed{b \rightarrow c}, \boxed{b \quad \bar{c}}\}$, respectively.[2] The cross-conjunction of these sets gives the following execution set of behaviour B: $E_B(\{a, b, c\}) = \{\boxed{a \rightarrow b \rightarrow c}, \boxed{a \quad b \quad \bar{c}}, \boxed{\bar{a} \quad \bar{b} \quad \bar{c}}\}$. Figure 3.8 in Section 3.6 illustrates the calculation of the cross-conjunction operation.

4.2.2 Decomposition

The alternative constraints of a causality relation ρ can be decomposed into alternative constraints that are imposed by its causality condition Γ and the alternative constraints that are imposed by its uncertainty attribute υ. Therefore, the cross-conjunction of E_Γ and E_υ is equal to E_ρ, where E_Γ, E_υ and E_ρ represent the execution sets allowed by ρ, Γ and υ.

For the purpose of orthogonality, the execution sets E_Γ and E_υ should only represent those constraints that are inherent to causality condition Γ and those constraints that are inherent to uncertainty attribute υ, respectively. Execution set E_υ only represents constraints concerning the occurrences or non-occurrences of actions, but does not represent constraints

2. Execution sets E_1 and E_2 do not represent the constraint that action a is independent of actions b and c and the constraint that b is independent of c, respectively. Since these constraints do not change the outcome of the cross-conjunction operation they are omitted for simplification. A complete elaboration of this (type of) example is given in the next chapter.

concerning the temporal ordering of action occurrences. The latter constraints are completely represented by execution set E_Γ.

Due to the may-interpretation of causality conditions, the uncertainty attribute imposes no (additional) constraints on the execution of the involved actions in case it associates the *may* uncertainty value with all alternative causality conditions. This implies that the set of alternative constraints imposed by the uncertainty attribute can be represented by execution set *E-Free*(*Ac*), where *Ac* represents the set of actions involved in the corresponding causality relation.

A disadvantage of the use of execution set *E-Free*(*Ac*) is that this set explicitly defines all possible independences and temporal relations between the action occurrences in *Ac*, whereas the uncertainty attribute does not model constraints concerning the (absence of) temporal relations between action occurrences. Therefore, execution set *EE-Free*(*Ac*) is introduced, which leaves the independence or relation between action occurrences undefined, such that:

$$EE\text{-}Free(Ac) = \{\langle A_\chi, \overline{A}_\chi, <_\chi, =_\chi, |_\chi \rangle \mid A_\chi \cup \overline{A}_\chi = Ac, <_\chi = \varnothing, =_\chi = \varnothing, |_\chi = \varnothing \}.$$

Similar to *E-Free*(*Ac*), execution set *EE-Free*(*Ac*) does not constrain the independence or temporal relation between action occurrences, but in contrast to *E-Free*(*Ac*), *EE-Free*(*Ac*) represents this by leaving the independence or relation between action occurrences undefined. Consequently, *EE-Free*(*Ac*) only represents the occurrences or non-occurrences of actions explicitly. Since this better reflects the concern of the uncertainty attribute, we prefer the use of *EE-Free*(*Ac*) to represent the absence of constraints on the occurrences or non-occurrences of actions.

For example, consider the causality relation of action *b* in the latter example of the previous section. The set of alternative constraints imposed by the causality condition of *b* is represented by execution set: $E_{\Gamma(b)}(\{a, b\}) = \{\boxed{a \rightarrow b}, \boxed{a \quad b}, \boxed{\overline{a} \quad \overline{b}}\}$. The uncertainty attribute of *b* does not impose any additional constraint. This corresponds to the following set of executions: $E_{\upsilon(b)}(\{a, b\}) = EE\text{-}Free(\{a, b\}) = \{\boxed{a \cdots b}, \boxed{a \quad b}, \boxed{\overline{a} \quad b}, \boxed{\overline{a} \quad \overline{b}}\}$. The cross-conjunction of sets $E_{\Gamma(b)}(\{a, b\})$ and $E_{\vartheta(b)}(\{a, b\})$ is equal to $E_2(\{a, b\})$.

In case the *must* uncertainty value is associated with one or more alternative causality conditions, the uncertainty attribute imposes as additional constraint that some executions in execution set E_Γ are not allowed. Consequently, execution set E_υ is a subset of *EE-Free*(*Ac*), which defines that all executions of E_Γ are allowed.

For example, consider the causality relation of action *c* in the latter example of the previous section. The set of alternative constraints imposed by the causality condition of *c* is represented by execution set $E_{\Gamma(c)}(\{b, c\}) = \{\boxed{b \rightarrow c}, \boxed{b \quad c}, \boxed{\overline{b} \quad \overline{c}}\}$. The uncertainty attribute of *c* imposes as additional constraint that action *c* must occur when *b* has occurred. This implies that execution $\boxed{b \quad \overline{c}}$ is not allowed. Since the uncertainty attribute of *c* does not impose any other constraint, the allowed execution set is equal to: $E_{\upsilon(c)}(\{b, c\}) = EE\text{-}Free(\{b, c\}) - \{\boxed{b \quad \overline{c}}\} = \{\boxed{b \cdots c}, \boxed{b \quad c}, \boxed{\overline{b} \quad \overline{c}}\}$. The cross-conjunction of execution sets $E_{\Gamma(c)}(\{b, c\})$ and $E_{\upsilon(c)}(\{b, c\})$ is equal to $E_3(\{b, c\})$.

Decomposition of causality conditions and uncertainty attribute constraints

In addition to the above, the alternative constraints imposed by the causality condition of an action can be decomposed into alternative constraints that are imposed by each of its alternative causality conditions. Furthermore, the alternative constraints imposed by the uncertainty attribute of this action can be decomposed into alternative constraints imposed by the individual associations of uncertainty values with alternative causality conditions. This property allows us to define the execution semantics of causality relations in a compositional way. Therefore, the execution semantics of alternative causality conditions and the execution semantics of associations of uncertainty values with alternative causality conditions are defined independently, in the sequel.

4.2.3 Formal definition

The symbols $[\![$ and $]\!]$ denote the function that defines the execution semantics of (part of) a causality relation, or a set of causality relations. For example, $[\![B]\!]$, $[\![\langle a, \Gamma, \upsilon\rangle]\!]$, $[\![\langle a, \Gamma\rangle]\!]$, $[\![\langle a, \gamma\rangle]\!]$ and $[\![\upsilon]\!]$ denote the execution semantics of behaviour B, the causality relation of action a, the causality condition of a, an alternative causality condition of a, and of the uncertainty attribute of a, respectively. In the latter case, action a does not have to be denoted explicitly, since the result action is part of the definition of υ.

The following two characteristics of causality relations are formally defined below:

- the execution semantics of a causality relation is equal to the cross-conjunction of the execution semantics of its causality condition and the execution semantics of its uncertainty attribute;
- the execution semantics of a causality-based behaviour definition is equal to the cross-conjunction of the execution semantics of its individual causality relations.

Definition 4.3 The execution semantics of causality relation $\langle a, \Gamma, \upsilon\rangle$ is defined as:

$$[\![\langle a, \Gamma, \upsilon\rangle]\!] = [\![\langle a, \Gamma\rangle]\!] \otimes [\![\upsilon]\!] \ . \qquad \blacksquare$$

Definition 4.4 The execution semantics of a causality-based behaviour B is defined as:

$$[\![B]\!] = \otimes\{[\![\rho]\!] \mid \rho \in B\} \ . \qquad \blacksquare$$

4.2.4 Completeness and correctness

The set of possible executions $[\![B]\!]$ of a behaviour B obtained using Definition 4.4, must be complete, i.e., the set contains all possible executions, and must be correct, i.e., the set does not contain impossible executions. These requirements are guaranteed by the characteristics of the product and cross-product operators. A formal proof is given in Property 4.5 below.

Furthermore, $[\![B]\!]$ should only contain regular executions. This implies that the sets of partial executions $[\![\rho]\!]$, for all $\rho \in B$, must satisfy the completeness rules identified in Section 3.6.5. This requirement is guaranteed by the execution semantics of the *or*-operator, and is

explained in the next chapter. For behaviours of two actions this requirement needs no further attention.

Formal definition

The boolean function *CompatSet* is introduced as a generalization of function *Compat* towards a set of executions (see Section 3.6.5), such that:

$$\begin{aligned} CompatSet(E) &= true && \text{if } \forall \chi_i, \chi_j \in E \mid Compat(\chi_i, \chi_j), \text{ with } E = \{\chi_1,.., \chi_n\}, n \geq 1; \\ &= false && \text{otherwise.} \end{aligned}$$

Property 4.5 $[\![B]\!]$ is complete and correct.

Proof: The proof of this assertion is by contradiction:

- assume that execution set $[\![B]\!]$ is incomplete. In this case, there exists a possible execution χ of behaviour *B*, such that:
  $\chi \notin [\![B]\!]$ and $\forall \rho \in B\ \exists \chi_\rho \in [\![\rho]\!] \mid Compat(\chi, \chi_\rho) = true$.

  The cross-product operation includes all cross-products of compatible executions, such that:
  $$\begin{aligned} [\![B]\!] &= \otimes\{[\![\rho]\!] \mid \rho \in B\} \\ &= \{\times\{\chi_1,.., \chi_n\} \mid \chi_1 \in [\![\rho_1]\!], .., \chi_n \in [\![\rho_n]\!] \wedge CompatSet(\{\chi_1,.., \chi_n\})\}\ . \end{aligned}$$

  Consequently, $\chi \in [\![B]\!]$ is true, which contradicts with assumption $\chi \notin [\![B]\!]$;
- assume that execution set $[\![B]\!]$ is incorrect. In this case, there exists an execution χ which is not allowed by behaviour *B*, such that:
  $\chi \in [\![B]\!]$ and $\exists \rho \in B\ \forall \chi_\rho \in [\![\rho]\!] \mid Compat(\chi, \chi_\rho) = false$.

  This is in conflict with the definition of the cross-product operator (see above), which excludes products of execution sets which contain two or more incompatible executions.

  Consequently, $\chi \notin [\![B]\!]$ is true, which contradicts with assumption $\chi \in [\![B]\!]$. ■

4.3 Basic causality relations

This section defines a limited set of basic causality conditions necessary to model temporal relations between actions. Furthermore, the interpretation of the possible uncertainty associations of these causality conditions is defined. In the remainder of this chapter, we consider a behaviour consisting of two actions *a* and *b*.

4.3.1 No temporal ordering

At the second level of the abstraction hierarchy in Section 2.5.3, temporal ordering is not considered yet. At this level we only consider the possibility that the occurrence of an action *a* may depend on the occurrence or non-occurrence of another action *b*, while abstracting from the relation between the time attributes of *a* and *b*.

Based on the above, the following two elementary causality conditions of action *a* are identified:

- $\gamma_a = b$, defines that the occurrence of action *b* is a condition for the occurrence of action *a*. Action *a* is allowed to occur in an execution if action *b* occurs in this execution;
- $\gamma_a = \overline{b}$, defines that the non-occurrence of action *b* is a condition for the occurrence of action *a*. Action *a* is allowed to occur in an execution if action *b* does not occur in this execution.

Causality conditions *b* and $\overline{b}$ are *exclusive*, since either condition *b* is satisfied, i.e., action *b* occurs in an execution, or condition $\overline{b}$ is satisfied, i.e., action *b* does not occur in an execution. This implies that the conjunction of causality conditions *b* and $\overline{b}$ renders a causality condition which can never be satisfied. Therefore, the conjunction of causality conditions *b* and $\overline{b}$ is not allowed as a condition for an action.

Causality conditions *b* and $\overline{b}$ are *complete*, since in case action *a* depends on action *b* in an execution, either action *a* depends on the occurrence of action *b* or action *a* depends on the non-occurrence of action *b*. This implies that the causality condition $\Gamma_a = b \vee \overline{b}$, which consists of alternative causality conditions *b* and $\overline{b}$, is always satisfied. Nonetheless, this causality condition is not equivalent with start condition $\surd$. Causality condition $\Gamma_a = b \vee \overline{b}$ defines a relation between *a* and *b* in which *a* depends on *b*, whereas the start condition $\surd$ defines that action *a* is independent of any other actions, particularly of action *b*.

Since temporal ordering is an elementary behaviour characteristic, such that the modelling of behaviours without temporal ordering is not meaningful within our application domain, we add temporal ordering to our model before elaborating on the various types of relations that can be defined between two actions *a* and *b*.

4.3.2 Temporal ordering

The causality relation concept should allow one to model the temporal ordering of the time moments τ_a and τ_b when actions *a* and *b* occur, respectively. Three orthogonal temporal orderings of the occurrences of two related actions *a* and *b* are distinguished: $\tau_a < \tau_b$, $\tau_a = \tau_b$ or $\tau_a > \tau_b$. In order to model these possible orderings in terms of causality relations, the causality condition $\gamma_a = b$ is refined into the following elementary causality conditions:

- $\gamma_a = {}^{<}b$, defines that the occurrence of action *b* is a condition for the occurrence of action *a*, such that occurrence of *b* must precede the occurrence of *a*. Action *a* is allowed to occur in an execution when *b* occurs in this execution before *a*.
  Assuming that *a* and *b* occur, the following must hold: $\tau_b < \tau_a$;
- $\gamma_a = {}^{=}b$, defines that the occurrence of action *b* is a condition for the occurrence of action *a*, such that the occurrence of *b* must happen simultaneously with the occurrence of *a*. Action *a* is allowed to occur in an execution when *b* occurs in this execution at the same time as *a*.
  Assuming that *a* and *b* occur, the following must hold: $\tau_b = \tau_a$;
- $\gamma_a = {}^{>}b$, defines that the occurrence of action *b* is a condition for the occurrence of action *a*, such that the occurrence of *b* must follow the occurrence of *a*. Action *a* is

allowed to occur in an execution when *b* will occur in this execution after *a*. Assuming that *a* and *b* occur, the following must hold: $\tau_b > \tau_a$.

Causality condition *b* defines that action *a* is allowed to occur if action *b* occurs, without defining any constraint regarding the temporal ordering of τ_b and τ_a, since the temporal ordering characteristic is not considered at that level of the abstraction hierarchy. The addition of the temporal ordering characteristic refines condition *b* into three exclusive causality conditions, which define that action *a* is allowed to occur when action *b* occurs before, simultaneously with, or after action *a*, respectively.

Figure 4.3 illustrates this refinement. Assuming that action *b* occurs at τ_b, Figure 4.3 relates the time moment τ_b to the time period in which action *a* is allowed to occur. Whether action *a* must or may occur within this time period is determined by the uncertainty attribute of *a*.

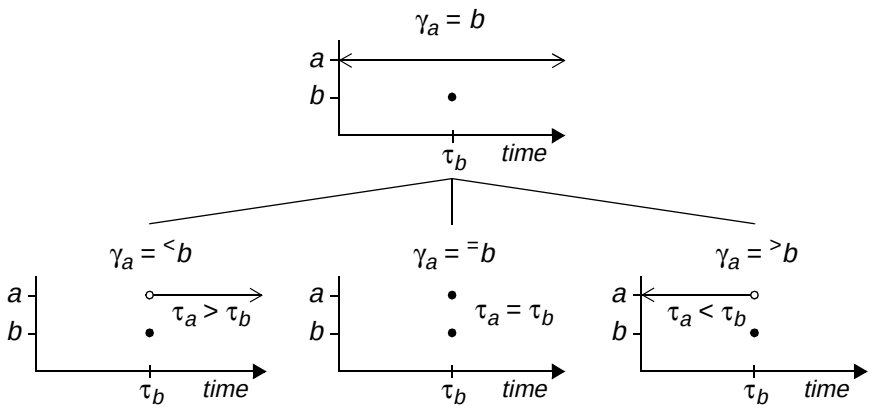


Figure 4.3 Refinement of causality condition *b*

The addition of the temporal ordering characteristic does not allow a refinement of causality condition $\overline{b}$, since we do not consider the ordering of the occurrences and non-occurrences of actions. For example, an intuitive decomposition of condition $\overline{b}$ into the more elementary conditions ${}^{<}\overline{b}$, ${}^{=}\overline{b}$ and ${}^{>}\overline{b}$ does *not* render a proper refinement of the characteristics that are modelled by condition $\overline{b}$. In contrast to condition $\overline{b}$, which defines that actions *a* and *b* do not occur in the same execution, conditions ${}^{<}\overline{b}$, ${}^{=}\overline{b}$ and ${}^{>}\overline{b}$ allow *a* and *b* to occur in the same execution, such that $\tau_b \geq \tau_a$, $\tau_b \neq \tau_a$ and $\tau_b \leq \tau_a$, respectively. Furthermore, the disjunction of these elementary conditions allows any ordering between actions *a* and *b*.

The addition of the temporal ordering characteristic neither affects the start condition √, since this condition does not define a relation between the occurrences of two actions.

4.3.3 Referring to the past, the present and the future

Causality conditions ${}^{>}b$ and $\overline{b}$ define that the occurrence of action *a* (partly) depends on "its future". In case of condition ${}^{>}b$, the occurrence of *a* depends on the future occurrence of *b* such that *b* must occur *after a* has occurred. In case of condition $\overline{b}$, the occurrence of *a* depends on the non-occurrence of *b*, such that action *b* is neither allowed to occur before, simultaneously nor *after* the occurrence of *a*.

We choose to disallow that the occurrence of an action depends on ‘future’ occurrences or ‘future’ non-occurrences of other actions (see ‘Motivation’ below). Therefore, the following restriction is imposed on the definition of causality conditions:

> the causality condition of an action *a* should neither define that another action *b* must occur after the occurrence of action *a*, nor define that another action *b* must not occur after the occurrence of action *a*.

Figure 4.4 illustrates this so called “past and present restriction”. Action *a* is assumed to occur at τ_a. The restriction that the causality condition of action *a* should not refer to occurrences or non-occurrences of actions in the future, is represented by the grey area.

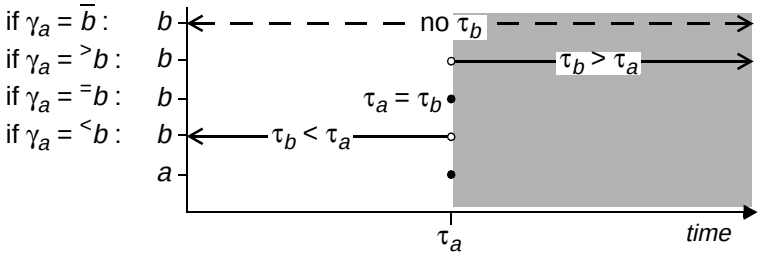


Figure 4.4: “Past and present restriction”

The “past and present” restriction prohibits causality condition ${}^{>}b$ and constrains causality condition $\overline{b}$ to condition ${}^{\leq}\overline{b}$, where

> $\gamma_a = {}^{\leq}\overline{b}$, defines that the non-occurrence of action *b* is a condition for the occurrence of action *a*, until *a* has occurred. Action *a* is allowed to occur when *b* does not occur before *a* and *b* does not occur simultaneously with *a*.
> Assuming that *b* occurs, the following must hold: either *a* is disabled by *b*, or *a* occurs such that $\tau_a < \tau_b$.

Motivation

The motivations for imposing the “past and present restriction” are:

- *minimal number of elementary causality conditions*: causality conditions ${}^{>}b$ and $\overline{b}$ that are forbidden and constrained by the above restriction can also be modelled in terms of the elementary causality condition ${}^{<}a$ and the elementary causality conditions ${}^{\leq}\overline{b}$ and ${}^{\leq}\overline{a}$, respectively:
  - the temporal ordering between *a* and *b* modelled by causality condition $\gamma_a = {}^{>}b$ can also be modelled in terms of causality condition $\gamma_b = {}^{<}a$, which implies that the direction of the dependency between *a* and *b* is reversed;
  - the exclusion of the occurrence of *b* after the occurrence of *a* modelled by causality condition $\gamma_a = \overline{b}$, can also be modelled in terms of the combination of causality conditions $\gamma_a = {}^{\leq}\overline{b}$ and $\gamma_b = {}^{\leq}\overline{a}$.
- *designer’s intuition*: when composing a behaviour one is used to think in terms of cause-effect relationships, such that the cause precedes the effect. In general, one does not think of an event (the effect) being dependent of some future event (the cause). A specific case is when cause and effect collapse in time, i.e., the moment of time at

which the cause happens cannot be distinguished from the moment of time at which the effect happens. Since this represents a realistic design concern that should be covered by our design model, occurrence condition $^{=}b$ is not excluded by the "past and present" restriction;

- *implementation complexity*: an implementation can only decide to execute an action based upon events that have happened before this decision is made. In case of occurrence condition $^{>}b$ the decision to execute *a* would have to be based upon some "agreement" to execute *b* after *a* has been executed. This agreement, if at all possible, involves the knowledge that the causality condition of *b* will be satisfied in the future. The satisfaction of this causality condition may, however, depend on other actions which should, possibly, occur after *a* or *b* have occurred, etc. Consequently, in order to establish an "agreement about future action occurrences" a complex network of dependencies may have to be analysed. This increases the complexity of an implementation significantly. This could have been avoided by modifying (the direction of) the dependency between *a* and *b* as has been discussed under the first sub-bullet of the first bullet above. A similar reasoning applies to causality condition $\overline{b}$.

4.3.4 Basic causality conditions

From the analysis carried out in Sections 4.3.2 and 4.3.3 we can conclude that three elementary causality conditions suffice to model how an action *a* may depend on another action *b*: $^{<}b$, $^{=}b$ and $^{\leq}\overline{b}$. Since these causality conditions are used as the most elementary building blocks to construct more complex causality conditions, they are called *basic* causality conditions. For reasons of conciseness and convenience, conditions $^{<}b$ and $^{\leq}\overline{b}$ are represented by the symbols *b* and $\neg b$ in the sequel, respectively.

The definitions of the basic causality conditions *b*, $\neg b$ and $^{=}b$ of action *a* in the preceding sections are summarized below. The constraints these conditions impose on the execution of actions *a* and *b* are also defined.

- $\gamma_a = b$, defines that action *a* is allowed to occur when action *b* occurs before action *a*. This condition is called an *enabling condition* and action *b* is called an *enabling action*, since the occurrence of *b* enables the occurrence of *a*.

  Enabling condition $\gamma_a = b$ constrains the execution of action *a*, but does not constrain the execution of action *b*. This basic causality condition allows (only) the following executions of *a* and *b*:

  - $\boxed{b \rightarrow a}$ and $\boxed{b \quad a}$, *a* may occur after *b* has occurred;
  - $\boxed{\overline{b} \quad a}$, the non-occurrence of *b* implies that *a* can not occur.

- $\gamma_a = \neg b$, defines that action *a* is allowed to occur when action *b* does not occur before nor simultaneously with action *a*. This condition is called a *disabling condition* and action *b* is called a *disabling action*, since the occurrence of *b* disables the occurrence of *a* when *a* has not occurred already before *b*.

  It is important to note that disabling condition $\gamma_a = \neg b$ constrains the execution of *both* *a* and *b*. The condition that *a* is not allowed to occur simultaneously with *b* implies the reciprocal condition that *b* is not allowed to occur simultaneously with *a*. The conse-

quences of this reciprocal aspect of the disabling condition are elaborated in Section 4.5.

Basic causality condition $\gamma_a = \neg b$ allows (only) the following executions of *a* and *b*:

- $\boxed{a \rightarrow b}$, if *a* and *b* occur then *a* must occur before *b*;
- $\boxed{\bar{a} \quad b}$, the occurrence of *b* disables the occurrence of *a*;
- $\boxed{a \quad \bar{b}}$ and $\boxed{\bar{a} \quad \bar{b}}$, *a* may occur when *b* does not occur.

- $\gamma_a = {}^{=}b$, defines that action *a* is allowed to occur when *b* occurs simultaneously with *a*. This condition is called a *synchronization condition* and actions *a* and *b* are called *synchronized actions*, since the occurrences of *a* and *b* have to be synchronized.

  The synchronization condition constrains the execution of both *a* and *b* in the same way. The condition that *a* is allowed to occur when *b* occurs simultaneously with *a*, implies the reciprocal condition that *b* is allowed to occur when *a* occurs simultaneously with *b*. The consequences of this reciprocal aspect of the synchronization condition are elaborated in Section 4.5.

  Basic causality condition $\gamma_a = {}^{=}b$ allows (only) the following executions of *a* and *b*:

  - $\boxed{a = b}$, if *a* occurs it occurs synchronously with *b*;
  - $\boxed{\bar{a} \quad b}$, *b* can occur due to another condition than condition ${}^{=}a$;
  - $\boxed{\bar{a} \quad \bar{b}}$, if *b* does not occur, *a* can not occur.

In case a basic causality condition is an alternative causality condition of action *a*, the uncertainty attribute of *a* must define one of two alternative uncertainty associations, i.e.: $\upsilon_a(\gamma_a) = \mathit{must}$ or $\upsilon_a(\gamma_a) = \mathit{may}$, with $\gamma_a \in \{b, \neg b, {}^{=}b\}$. These uncertainty associations are defined below in terms of the constraints they impose on the execution of *a* and *b*, in addition to the constraints of the corresponding basic causality conditions.

- $\upsilon(a, b)$, defines the uncertainty that action *a* occurs when action *b* occurs before action *a*, such that:

  - $\upsilon(a, b) = \mathit{must}$, defines that *a* must occur when *b* occurs before *a*.

    This uncertainty association constrains the execution of *a*, since it does not allow an execution in which *b* occurs and *a* does not occur, i.e., execution $\boxed{b \quad \bar{a}}$ is disallowed;

  - $\upsilon(a, b) = \mathit{may}$, defines that *a* may (not) occur when *b* occurs before *a*.

    This uncertainty attribute does not constrain the execution of *a* and *b*.

- $\upsilon(a, \neg b)$, defines the uncertainty that action *a* occurs when action *b* does not occur before nor simultaneously with action *a*.

  This uncertainty is equal to the uncertainty that *a* occurs (i) when *b* occurs after *a*, or (ii) when *b* does not occur at all. The satisfaction of sub-condition (i) implies the occurrence of *a* however. Therefore, the uncertainty defined by $\upsilon(a, \neg b)$ is equal to the uncertainty that action *a* occurs when action *b* does not occur at all, such that:

  - $\upsilon(a, \neg b) = \mathit{must}$, defines that *a* must occur when *b* does not occur.

This uncertainty association constrains the execution of a (and b), since it does not allow an execution in which a and b do not occur, i.e., execution $\boxed{\bar{a} \quad \bar{b}}$ is disallowed;

- $\upsilon(a, \neg b)$ = *may*, defines that a may (not) occur when b does not occur.

  This uncertainty attribute does not constrain the execution of a and b.

- $\upsilon(a, {}^{=}b)$, defines the uncertainty that action a occurs when action b occurs simultaneously with action a.

  The satisfaction of the condition that b occurs simultaneously with a implies the occurrence of a. Therefore, $\upsilon(a, {}^{=}b)$ is equal to the uncertainty that a synchronizes with b, such that:

  - $\upsilon(a, {}^{=}b)$ = *must*, defines that a and b must synchronize.

    This uncertainty association constrains the execution of a and b, since it does not allow an execution in which a and b do not occur, i.e., execution $\boxed{\bar{a} \quad \bar{b}}$ is disallowed;

  - $\upsilon(a, {}^{=}b)$ = *may*, defines that a and b may (not) synchronize.

    This uncertainty attribute does not constrain the execution of a and b.

The identification of three basic causality conditions b, $\neg b$ and ${}^{=}b$ and the identification of two uncertainty values *must* and *may*, renders six basic causality relations of action a:

- the *must enabling causality relation*: $\langle a, b, \langle a, b, must\rangle\rangle$ or $b_! \rightarrow a$;
- the *may enabling causality relation*: $\langle a, b, \langle a, b, may\rangle\rangle$ or $b_? \rightarrow a$;
- the *must disabling causality relation*: $\langle a, \neg b, \langle a, \neg b, must\rangle\rangle$ or $\neg b_! \rightarrow a$;
- the *may disabling causality relation*: $\langle a, \neg b, \langle a, \neg b, may\rangle\rangle$ or $\neg b_? \rightarrow a$;
- the *must synchronization causality relation*: $\langle a, {}^{=}b, \langle a, {}^{=}b, must\rangle\rangle$ or ${}^{=}b_! \rightarrow a$;
- the *may synchronization causality relation*: $\langle a, {}^{=}b, \langle a, {}^{=}b, may\rangle\rangle$ or ${}^{=}b_? \rightarrow a$;

4.3.5 Formal definition

The formal definition of the execution semantics of the basic causality conditions $\gamma_a = b$, $\gamma_a = \neg b$ and $\gamma_a = {}^{=}b$, can be derived directly from the informal definitions in the preceding section.

Definition 4.6 The execution semantics of the basic causality conditions b, $\neg b$ and ${}^{=}b$ of an action a is defined as:

- $[\![\langle a, b\rangle]\!] = \{ \boxed{b \rightarrow a}, \boxed{b \quad \bar{a}}, \boxed{\bar{b} \quad \bar{a}} \}$;
- $[\![\langle a, \neg b\rangle]\!] = \{ \boxed{a \rightarrow b}, \boxed{\bar{a} \quad \bar{b}}, \boxed{a \quad \bar{b}}, \boxed{\bar{b} \quad \bar{a}} \}$;
- $[\![\langle a, {}^{=}b\rangle]\!] = \{ \boxed{a = b}, \boxed{\bar{a} \quad \bar{b}}, \boxed{\bar{a} \quad \bar{b}} \}$. ■

A *may* uncertainty association does not impose (additional) constraints on the execution of a and b. Therefore, the execution semantics of a *may* uncertainty association is equal to execution set $EE\text{-}Free(\{a, b\})$, which allows any execution of actions a and b.

A *must* uncertainty association disallows one execution of $EE\text{-}Free(\{a, b\})$. Therefore, the execution semantics of a *must* uncertainty association is equal to execution set $EE\text{-}Free(\{a, b\}) - E_{dis}(\{a, b\})$, where $E_{dis}(\{a, b\})$ represents a set containing the disallowed execution. The complement operator *Comp* is introduced below in order to allow the execution semantics of a *must* uncertainty association to be defined in terms of execution set $E_{dis}(\{a, b\})$. This proves convenient when defining the execution semantics of compositions of uncertainty associations later on.

Definition 4.7 The complement of execution set $E(Ac)$, with $E(Ac) \subseteq EE\text{-}Free(Ac)$, is defined by the execution operator $Comp : \boldsymbol{EE} \rightarrow \boldsymbol{EE}$, such that:

$$Comp(E(Ac)) = EE\text{-}Free(Ac) - E(Ac) \; . \qquad \blacksquare$$

Definition 4.8 The execution semantics of the uncertainty associations $\langle a, \gamma_a, must\rangle$ and $\langle a, \gamma_a, may\rangle$, with $\gamma_a \in \{b, \neg b, {}^{=}b\}$, is defined as:

- $[\![\langle a, b, must\rangle]\!] \;=\; Comp(\{\, \boxed{b \quad a} \,\})$;
- $[\![\langle a, b, may\rangle]\!] \;=\; EE\text{-}Free(\{a, b\})$;
- $[\![\langle a, \neg b, must\rangle]\!] \;=\; Comp(\{\, \boxed{\bar{a} \quad \bar{b}} \,\})$;
- $[\![\langle a, \neg b, may\rangle]\!] \;=\; EE\text{-}Free(\{a, b\})$;
- $[\![\langle a, {}^{=}b, must\rangle]\!] \;=\; Comp(\{\, \boxed{\bar{a} \quad \bar{b}} \,\})$;
- $[\![\langle a, {}^{=}b, may\rangle]\!] \;=\; EE\text{-}Free(\{a, b\})$. ■

4.3.6 The start condition

The start condition $\gamma_a = \surd$ is an elementary condition which models that result action a is allowed to occur from the beginning of the behaviour. This implies that $\gamma_a = \surd$ allows the executions $\boxed{a \quad}$ and $\boxed{\bar{a} \quad}$, when the start condition is an alternative causality condition of a.

This definition of the start condition is independent of the context (behaviour) in which action a is defined. When the start condition is an alternative causality condition of action a in a behaviour B with actions Ac_B, the alternative causality condition $\gamma_a = \surd$ defines also that action a is independent of all other actions in B. In case of a behaviour of two actions a and b, this implies that $\gamma_a = \surd$ allows the following executions:

- $\boxed{a \quad b}$, $\boxed{a \quad \bar{b}}$, $\boxed{\bar{a} \quad b}$, $\boxed{\bar{a} \quad \bar{b}}$; and
- $\boxed{a \rightarrow b}$, since action b may depend on the occurrence of action a (see also Section 4.5).

The execution set $\{\boxed{a \quad b}, \boxed{a \quad \bar{b}}, \boxed{\bar{a} \quad b}, \boxed{\bar{a} \quad \bar{b}}, \boxed{a \rightarrow b}\}$ may be considered as an additional constraint on the execution of action a, such that the execution semantics of $\gamma_a = \surd$ in the context of a behaviour of two actions a and b is defined by the cross-conjunction of this

execution set and execution set {[a], [$\bar{a}$]}. A general definition of this additional constraint for behaviours of multiple actions is provided in the next chapter.

Formal definition

Two definitions of the execution semantics of the start condition of an action *a* and its possible uncertainty associations are given below: (i) one which is independent of the context of action *a*, and (ii) one for the specific case of a behaviour of two actions *a* and *b*.

Definition 4.9 The execution semantics of the start condition √ of an action *a*, and its possible uncertainty associations ⟨*a*, √, *must*⟩ and ⟨*a*, √, *may*⟩ is defined as:

$[\![\langle a, \surd\rangle]\!] = \{[a], [\bar{a}]\}$;

$[\![\langle a, \surd, must\rangle]\!] = \{[a]\}$;

$[\![\langle a, \surd, may\rangle]\!] = \{[a], [\bar{a}]\}$. ■

Definition 4.10 Assuming a behaviour consisting of two actions *a* and *b*, the execution semantics of the start condition √ of action *a*, and its possible uncertainty associations ⟨*a*, √, *must*⟩ and ⟨*a*, √, *may*⟩ is defined as:

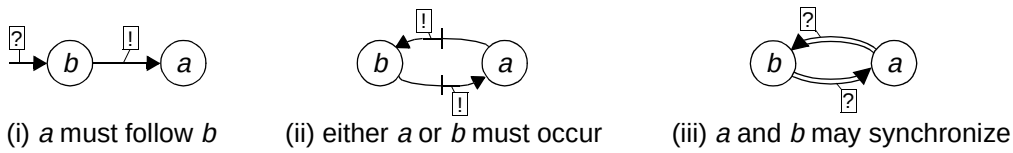
$[\![\langle a, \surd\rangle]\!] = \{[a\ b], [a \rightarrow b], [a\ \bar{b}], [\bar{a}\ b], [\bar{a}\ \bar{b}]\}$;

$[\![\langle a, \surd, must\rangle]\!] = Comp(\{[\bar{a}\ b], [\bar{a}\ \bar{b}]\})$;

$[\![\langle a, \surd, may\rangle]\!] = EE\text{-}Free(\{a, b\})$. ■

4.3.7 Examples: some simple action relations

Figure 4.5 depicts some simple action relations between actions *a* and *b* that can be defined using the basic causality conditions discussed above. In addition, Figure 4.5 introduces our graphical notation for causality relations.



(i) *a* must follow *b* (ii) either *a* or *b* must occur (iii) *a* and *b* may synchronize

Figure 4.5: Examples of some simple action relations

Each of the action relations is explained below:

- *a* must follow *b*: defines the sequential ordering of *b* and *a*, such that *b* may occur and *a* must occur after *b*. The corresponding textual notation is $\{b_! \rightarrow a, \surd_? \rightarrow b\}$. Enabling condition *b* of action *a* is graphically represented by a solid arrow from *b* to *a*. Uncertainty values are represented in boxes, which are linked to the corresponding causality conditions;
- either *a* or *b* must occur: defines a choice between *a* and *b*, such that one of both actions must occur. This choice is modelled as a mutual disabling. The corresponding textual notation is $\{\neg b_! \rightarrow a, \neg a_! \rightarrow b\}$. Disabling condition $\neg b$ of action *a* is graphically represented by a solid arrow from *b* to *a*, with a vertical bar in between. This vertical bar

symbolizes the possibility that the occurrence of b blocks the occurrence of a;

- a and b may synchronize: defines the possible synchronization of a and b, such that both actions occur at the same time, or both actions do not occur at all. The corresponding textual notation is $\{{}^{=}b_{?} \rightarrow a, {}^{=}a_{?} \rightarrow b\}$. Synchronization condition ${}^{=}b$ of action a is graphically represented by a solid double lined arrow from b to a.

The possible executions of the above behaviours can be obtained using the execution semantics presented so far. For example, the possible executions of behaviour $\{\neg b_{!} \rightarrow a, \neg a_{!} \rightarrow b\}$ is obtained as follows:

$$
\begin{aligned}
[\![\{\neg b_{!} \rightarrow a, \neg a_{!} \rightarrow b\}]\!] &= [\![\neg b_{!} \rightarrow a]\!] \otimes [\![\neg a_{!} \rightarrow b]\!] \\
&= \{ \boxed{a \quad \overline{b}}, \boxed{\overline{a} \quad b} \}
\end{aligned}
$$

with:

$$
\begin{aligned}
[\![\neg b_{!} \rightarrow a]\!] &= [\![\langle a, \neg b\rangle]\!] \otimes [\![\langle a, \neg b, \mathit{must}\rangle]\!] \\
&= \{ \boxed{a \rightarrow b}, \boxed{\overline{a} \quad b}, \boxed{a \quad \overline{b}}, \boxed{\overline{b} \quad \overline{a}} \} \\
&\quad \otimes \{ \boxed{a \cdots b}, \boxed{\overline{a} \quad b}, \boxed{a \quad \overline{b}} \} \\
&= \{ \boxed{a \rightarrow b}, \boxed{\overline{a} \quad b}, \boxed{a \quad \overline{b}} \} ; \\
[\![\neg a_{!} \rightarrow b]\!] &= [\![\langle b, \neg a\rangle]\!] \otimes [\![\langle b, \neg a, \mathit{must}\rangle]\!] \\
&= \{ \boxed{b \rightarrow a}, \boxed{a \quad \overline{b}}, \boxed{\overline{a} \quad b} \} .
\end{aligned}
$$

For convenience, the execution semantics of an expression $e \in L_{causality}$ may be represented as $[\![\, e \,]\!]$ instead of $[\![\, [\![\, e \,]\!]_{\ldots} \,]\!]$.

Example: deadlock

Figure 4.6 depicts a behaviour with a cyclic enabling relation between actions a and b. This behaviour only allows the empty execution, i.e.: $[\![\{b_{?} \rightarrow a, a_{?} \rightarrow b\}]\!] = \{ \boxed{\overline{a} \quad \overline{b}} \}$, which represents a deadlock between actions a and b.

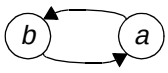


Figure 4.6: Deadlock

Shorthand notation

We introduce as a shorthand notation the possibility to omit the explicit definition of uncertainty attribute values. In case the definition of an uncertainty attribute value is omitted, we assume that the *may* uncertainty value is defined by default. For example, $b \rightarrow a$ is a shorthand notation for $b_{?} \rightarrow a$.

4.4 Composite causality relations

The basic causality conditions of Definition 4.6 and the start condition of Definition 4.10 are defined as the most elementary building blocks from which more complex causality conditions can be built. Two composition operators are defined to compose causality conditions from other causality conditions: the *and*-operator which defines a conjunction of conditions, and the *or*-operator which defines a disjunction of conditions. The *and*-operator and

the *or*-operator are denoted by the symbols $\wedge$ and $\vee$, respectively. Causality conditions composed from other causality conditions are called *composite causality conditions*.

4.4.1 The *and*-operator

The conjunction of two causality conditions defines that both causality conditions must be satisfied in order to allow the occurrence of the result action. The *and*-operator enables one to restrict the condition under which an action is allowed to occur, by defining that multiple conditions must be satisfied.

The basic causality conditions b, $\neg b$, and $\overline{\overline{b}}$ are exclusive, i.e., the satisfaction of one of them implies the dissatisfaction of the others. Therefore, the conjunction of two (or more) of these basic causality conditions renders a composite causality condition that can never be satisfied. In general, conjunctions of distinct basic causality conditions that involve the same action are always dissatisfied.

The conjunction of one of the basic causality conditions $\gamma_a \in \{b, \neg b, \overline{\overline{b}}\}$ with start condition $\surd$ is equal to γ_a, i.e., $\gamma_a \wedge \surd = \gamma_a$. The start condition is the identity element of the conjunction operation.

4.4.2 The *or*-operator

The disjunction of two conjunctive causality conditions defines two alternative causality conditions, such that only one of both causality conditions has to be satisfied in order to allow the occurrence of the result action. The *or*-operator enables one to relax the condition under which an action is allowed to occur, by defining alternative conditions that allow the occurrence of this action.

For example, causality relation $b \vee \neg b \rightarrow a$ defines two alternative conditions that allow the occurrence of action a, i.e.: a is allowed to occur after b has occurred, or a is allowed to occur when b does not occur before nor simultaneously with a. The disjunction operation relaxes the condition for the occurrence of a w.r.t. the individual conditions b and $\neg b$, since a is allowed to occur before the (possible) occurrence of b and is allowed to occur after the occurrence of b. The only restriction left is that a and b are not allowed to occur simultaneously, because this restriction is defined by both conditions b and $\neg b$.

Compositions of the basic causality conditions *b*, ¬*b* and $\overline{\overline{b}}$

The following composite causality conditions of action a are distinguished, which consist of the disjunction of two or more of the basic causality conditions b, $\neg b$ and $\overline{\overline{b}}$:

- $\Gamma_a = b \vee \neg b$, defines that a is allowed to occur after b has occurred, or is allowed to occur when b does not occur before nor simultaneously with the occurrence of a.
  In case a and b occur, the following must hold: $\tau_a < \tau_b$ or $\tau_b < \tau_a$.

  This condition allows the following executions of a and b:

  - $\boxed{b \rightarrow a}$, $\boxed{a \rightarrow b}$, $\boxed{b \quad \overline{a}}$, $\boxed{a \quad \overline{b}}$, $\boxed{\overline{a} \quad \overline{b}}$;

- $\Gamma_a = b \vee {}^{=}b$, defines that *a* is allowed to occur after *b* has occurred, or is allowed to occur simultaneously with the occurrence of *b*.
  In case *a* and *b* occur, the following must hold: $\tau_b < \tau_a$ or $\tau_a = \tau_b$.

  This condition allows the following executions of *a* and *b*:

  - $\boxed{b \rightarrow a}$, $\boxed{a = b}$, $\boxed{b \quad a}$, $\boxed{a \quad b}$;
- $\Gamma_a = \neg b \vee {}^{=}b$, defines that *a* is allowed to occur when *b* does not occur before nor simultaneously with the occurrence of *a*, or is allowed to occur simultaneously with the occurrence of *b*.
  In case *a* and *b* occur, the following must hold: $\tau_a < \tau_b$ or $\tau_a = \tau_b$.

  This condition allows the following executions of *a* and *b*:

  - $\boxed{a \rightarrow b}$, $\boxed{a = b}$, $\boxed{b \quad a}$, $\boxed{a \quad b}$, $\boxed{a \quad b}$;
- $\Gamma_a = b \vee \neg b \vee {}^{=}b$, defines that *a* is allowed to occur after *b* has occurred, or is allowed to occur when *b* does not occur before nor simultaneously with the occurrence of *a*, or is allowed to occur simultaneously with the occurrence of *b*.
  In case *a* and *b* occur, the following must hold: $\tau_a < \tau_b$ or $\tau_a = \tau_b$ or $\tau_b < \tau_a$.

  This condition allows the following executions of *a* and *b*:

  - $\boxed{b \rightarrow a}$, $\boxed{a \rightarrow b}$, $\boxed{a = b}$, $\boxed{b \quad a}$, $\boxed{a \quad b}$, $\boxed{a \quad b}$.

The *or*-operator relaxes the constraints on the execution of actions *a* and *b*. A disjunction of multiple basic causality conditions allows all executions allowed by the individual basic conditions and disallows the executions disallowed by all individual basic conditions. This implies that the execution semantics of a disjunction of multiple basic causality conditions is defined by the union of the execution semantics of the individual basic causality conditions.

Composite causality conditions involving the start condition

The start condition may be composed with one of the basic causality conditions *b*, $\neg b$ and ${}^{=}b$ or with one of the above composite causality conditions, using the *or*-operator. The resulting composite causality conditions define that action *a* is independent of action *b* in some executions, while action *a* depends on action *b* in other executions. Consequently, the disjunction of the start condition with other causality conditions enables one to model dynamically related actions. Behaviours of dynamically related actions are discussed in Section 4.6.2.

A composite causality condition Γ_a which consists of the disjunction of the start condition $\surd$ with a causality condition $\Gamma_a' \subseteq \{b, \neg b, {}^{=}b\}$, such that $\Gamma_a = \surd \vee \Gamma_a'$, allows all executions allowed by $\surd$ or Γ_a', and disallows the executions disallowed by $\surd$ and Γ_a'. Therefore, the execution semantics of Γ_a is defined by the union of the executions semantics of $\surd$ and the execution semantics of Γ_a'.

Uncertainty attribute

The uncertainty attribute of action a must associate an uncertainty value with each alternative causality condition in Γ_a. This implies that the uncertainty attribute υ_a may consist of multiple uncertainty associations, one per alternative causality condition.

In principle, the uncertainty attribute of action a may define an arbitrary association of uncertainty values with the alternative causality conditions in Γ_a. For example, in case of condition $\Gamma_a = b \vee \neg b \vee {}^{=}b$, the uncertainty attribute may define one of $2^3 = 8$ different combinations of uncertainty associations, e.g.: $\upsilon_a(a, b) = \mathit{must}$, $\upsilon_a(a, \neg b) = \mathit{may}$, and $\upsilon_a(a, {}^{=}b) = \mathit{may}$, which is also represented as: $\upsilon_a = \{\langle a, b, \mathit{must}\rangle, \langle a, \neg b, \mathit{may}\rangle, \langle a, {}^{=}b, \mathit{may}\rangle\}$.

It is important to note that only must uncertainty associations constrain the execution of a and b. Each must uncertainty association disallows a particular execution, such that the union of these disallowed executions constitutes the set of executions that are disallowed by the uncertainty attribute. The execution semantics of the uncertainty attribute is equal to the complement of the latter set.

4.4.3 Formal definition

This section restricts the definition of the execution semantics of disjunctive causality conditions to all possible disjunctions of basic causality conditions involving a single action and the start condition $\surd$. The definition of the execution semantics of conjunctions of causality conditions is not considered here. A general definition of the execution semantics of conjunctions and disjunctions of causality conditions is presented in Chapter 5.

Definition 4.11 The execution semantics of the disjunctive causality condition Γ_a of action a is defined as:

$$[\![\langle a, \Gamma_a\rangle]\!] = \cup\{ [\![\langle a, \gamma\rangle]\!] \mid \gamma \in \Gamma_a \}, \quad \text{with } \Gamma_a \in \wp(\Gamma)\text{-}\{\varnothing\} \text{ and } \Gamma = \{\surd, b, \neg b, {}^{=}b\}. \qquad \blacksquare$$

Definition 4.12 The execution semantics of the uncertainty attribute υ_a of action a is defined as:

$$[\![\upsilon_a]\!] = \mathit{Comp}(\cup\{\mathit{Comp}([\![\varphi]\!]) \mid \varphi \in \upsilon_a \}), \quad \text{with } \upsilon_a \subseteq \{a\} \times \Gamma \times \boldsymbol{U} \text{ and } \Gamma = \{\surd, b, \neg b, {}^{=}b\}. \qquad \blacksquare$$

The following property presents an alternative way to obtain $[\![\upsilon_a]\!]$.

Property 4.13 The execution semantics of the uncertainty attribute υ_a of action a is equal to the intersection of the execution semantics of the constituent uncertainty associations:

$$[\![\upsilon_a]\!] = \cap\{[\![\varphi]\!] \mid \varphi \in \upsilon_a\}, \quad \text{with } \upsilon_a \subseteq \{a\} \times \Gamma \times \boldsymbol{U} \text{ and } \Gamma = \{\surd, b, \neg b, {}^{=}b\}.$$

Proof: The proof is based on Definitions 4.7 and 4.12.

$$\begin{aligned}
[\![\upsilon_a]\!] &= \mathit{Comp}(\cup\{\mathit{Comp}([\![\varphi]\!]) \mid \varphi \in \upsilon_a \}) \\
&= \mathit{EE\text{-}Free}(\{a, b\}) - (\cup\{ \mathit{EE\text{-}Free}(\{a, b\}) - [\![\varphi]\!] \mid \varphi \in \upsilon_a\}) \\
&= \mathit{EE\text{-}Free}(\{a, b\}) - (\mathit{EE\text{-}Free}(\{a, b\}) - (\cap\{ [\![\varphi]\!] \mid \varphi \in \upsilon_a\})) \\
&= \cap\{ [\![\varphi]\!] \mid \varphi \in \upsilon_a\}
\end{aligned} \qquad \blacksquare$$

The order in which the alternative causality conditions of an action are specified is irrelevant. Parentheses may be used to structure causality conditions containing multiple alternative causality conditions.

Property 4.14 The *or*-operator is idempotent, commutative and associative.

Proof: These properties are inherited from the union operator on sets. ■

4.4.4 Examples: more complex action relations

Figure 4.7 depicts some action relations that can be defined using the composite causality conditions discussed above:

- (atomic) interleaving of a and b: defines that actions a and b can occur in any arbitrary order, but are not allowed to occur simultaneously. Furthermore, the interleaving relation is defined as an atomic relation, in the sense that either a and b occur both in an execution or none of both actions occur. The corresponding textual notation is $\{b_! \vee \neg b_? \rightarrow a,\ a_! \vee \neg a_? \rightarrow b\}$. The symbols □ and ■ are used to graphically represent the disjunction and conjunction of two or more causality conditions, respectively;
- a not before b: defines that action b occurs before action a or actions a and b occur simultaneously. Since the *may* uncertainty value is assumed by default, a and b may not occur even when one of their alternative conditions is satisfied. The corresponding textual notation is $\{b \vee {}^{=}b \rightarrow a,\ \neg a \vee {}^{=}a \rightarrow b\}$.

(i) (atomic) interleaving of a and b (ii) a not before b

Figure 4.7: Examples of some more complex action relations

The possible executions of the above behaviours can be obtained using the execution semantics presented so far. For example, the set of possible executions of behaviour $\{b_! \vee \neg b_? \rightarrow a,\ a_! \vee \neg a_? \rightarrow b\}$ is obtained as follows:

$$\begin{aligned}
[\![\{b_! \vee \neg b_? \rightarrow a,\ a_! \vee \neg a_? \rightarrow b\}]\!] &= [\![b_! \vee \neg b_? \rightarrow a]\!] \otimes [\![a_! \vee \neg a_? \rightarrow b]\!] \\
&= \{\ \boxed{b \rightarrow a}\ ,\ \boxed{a \rightarrow b}\ ,\ \boxed{\overline{b}\quad \overline{a}}\ \}
\end{aligned}$$

with:

$$\begin{aligned}
[\![b_! \vee \neg b_? \rightarrow a]\!] &= [\![\{\langle a, \{b, \neg b\}\rangle\}]\!] \otimes [\![\{\langle a, b, \mathit{must}\rangle, \langle a, \neg b, \mathit{may}\rangle\}]\!] \\
&= \{\ \boxed{b \rightarrow a}\ ,\ \boxed{a \rightarrow b}\ ,\ \boxed{a\quad \overline{b}}\ ,\ \boxed{\overline{b}\quad \overline{a}}\ \}
\end{aligned}$$

with:

$$\begin{aligned}
[\![\{\langle a, \{b, \neg b\}\rangle\}]\!] &= [\![\langle a, b\rangle]\!] \cup [\![\langle a, \neg b\rangle]\!] \\
&= \{\ \boxed{b \rightarrow a}\ ,\ \boxed{a \rightarrow b}\ ,\ \boxed{\overline{a}\quad b}\ ,\ \boxed{a\quad \overline{b}}\ ,\ \boxed{\overline{b}\quad \overline{a}}\ \}
\end{aligned}$$

$$\begin{aligned}
[\![\{\langle a, b, \mathit{must}\rangle, \langle a, \neg b, \mathit{may}\rangle\}]\!] &= [\![\langle a, b, \mathit{must}\rangle]\!] \cap [\![\langle a, \neg b, \mathit{may}\rangle]\!] \\
&= \{\ \boxed{a \cdots b}\ ,\ \boxed{a\quad b}\ ,\ \boxed{\overline{b}\quad \overline{a}}\ \}
\end{aligned}$$

$$[\![a_! \vee \neg a_? \rightarrow b]\!] = \{\ \boxed{b \rightarrow a}\ ,\ \boxed{a \rightarrow b}\ ,\ \boxed{\overline{a}\quad b}\ ,\ \boxed{\overline{b}\quad \overline{a}}\ \}\ \#\ \text{analogously}\ \#.$$

4.5 Combinations of causality relations

A behaviour consisting of two actions *a* and *b* is defined by the combination of the causality relations of these actions. Some combinations of causality relations render inconsistent behaviour models. The causality relations of *a* and *b* may define inconsistent causality conditions or inconsistent uncertainty associations. In general, inconsistencies are characterized by the definition of alternative causality conditions or uncertainty associations that can never be satisfied.

For example, the combination of causality relations $\{b \rightarrow a, a \rightarrow b\}$ results in an inconsistent behaviour. Due to the cyclic dependency between *a* and *b*, enabling conditions *a* and *b* can never be satisfied. Another example is behaviour $\{{}^{=}b_{!} \rightarrow a, {}^{=}a_{?} \rightarrow b\}$, which defines two inconsistent uncertainty associations. Uncertainty association $\upsilon_a({}^{=}b)$ = *must* defines that *a* and *b* must synchronize, while $\upsilon_b({}^{=}a)$ = *may* defines that *a* and *b* may synchronize.

The above examples show that one can not combine causality relations of actions *a* and *b* in an arbitrary way. A causality relation of action *a* imposes certain restrictions on the possible causality relations of action *b* with which this causality relation can be combined, and vice-versa. These (mutual) restrictions are reflected in a number of combination rules, which define for each of the elementary causality conditions of action *a*, i.e., $\gamma_a = \surd$, $\gamma_a = b$, $\gamma_a = {}^{=}b$, and $\gamma_a = {}^{\neg}b$, with which elementary causality conditions of action *b* it can be combined. Using these rules, the consistent combinations of causality conditions Γ_a and Γ_b of actions *a* and *b* can be determined and inconsistencies can be eliminated from a specification.

A consistent combination of causality conditions Γ_a and Γ_b represents a *behaviour family*, which consists of all possible behaviour definitions $\{\Gamma_a \rightarrow a\ [\upsilon_a], \Gamma_b \rightarrow b\ [\upsilon_b]\}$, which are called *family members*, such that υ_a and υ_b define allowed combinations of uncertainty associations. Therefore, associated with the combination rules mentioned above, rules are defined to determine the consistent combinations of uncertainty associations.

The notation $\{\Gamma_a \gtrdot a, \Gamma_b \gtrdot b\}$ is introduced to represent a behaviour family, which is characterized by the combination of causality conditions Γ_a and Γ_b. The representation $\{\Gamma_a \rightarrow a, \Gamma_b \rightarrow b\}$ can not be used for this purpose, due to the may interpretation of causality conditions.

4.5.1 Combination rules for $\gamma_a = \surd$

The elementary causality condition $\gamma_a = \surd$ defines that action *a* is allowed to occur independently of action *b*. The definition of this causality condition (implicitly) makes one of the following assumptions about the causality relation of action *b*:

1. actions *a* and *b* are defined to occur independently. In this case the causality relation of *b* defines that *b* is allowed to occur independently of *a*, i.e., $\surd \in \Gamma_b$;
2. the occurrence of *b* depends upon the occurrence of *a*. In this case the causality relation of *b* defines that *b* is allowed to occur after *a* has occurred, i.e., $a \in \Gamma_b$.

When assuming $\Gamma_a = \{\surd\}$, the following consistent behaviour families are distinguished:

- $\{\surd \gtrdot a, \surd \gtrdot b\}$, which represents a family of behaviours which have in common the independence of actions *a* and *b*. This family is called "independence of *a* and *b*";
- $\{\surd \gtrdot a, a \gtrdot b\}$, which represents a family of *enabling relations*, in which the occurrence of *a* (the enabling action) enables the occurrence of *b* (the enabled action). This family is called "*a* enables *b*".

The individual members of these two families are distinguished by the values of uncertainty association $\upsilon_a(\surd)$ and of uncertainty associations $\upsilon_b(\surd)$ and $\upsilon_b(a)$, respectively. These uncertainty associations can be defined independently, since action *a* is independent of action *b* in both families. This implies both families have four members.

The above is summarized in the following combination rule.

Combination Rule 1:

The elementary causality condition $\gamma_a = \surd$ can be combined with:

(i) $\gamma_b = \surd$, which models the independence of *a* and *b*; or

(ii) $\gamma_b = a$, which models the enabling of *b* by *a*.

Consequently, if $\surd \in \Gamma_a$ then $\surd \in \Gamma_b$ or $a \in \Gamma_b$, or both.

The uncertainty association $\upsilon_a(\surd)$ can be defined independently of the uncertainty associations $\upsilon_b(\surd)$ and $\upsilon_b(a)$.

4.5.2 Combination rules for $\gamma_a = b$

The elementary causality condition $\gamma_a = b$ defines that action *a* is allowed to occur when action *b* has occurred before *a* in an execution. The definition of this causality condition assumes that *a* and *b* are allowed to occur in the same execution, and that *b* is allowed to occur before *a* occurs. This implies that the causality relation of *b* should either

1. define that *b* is allowed to occur independently of *a*, i.e., $\surd \in \Gamma_b$;
2. define that *b* is allowed to occur when *a* does not occur before nor simultaneously with *a*, i.e., $\neg a \in \Gamma_b$.

Behaviour family $\{b \gtrdot a, \surd \gtrdot b\}$ is equivalent to the one described by Combination Rule 1 *(ii)*, with alternated roles of *a* and *b*.

The combination $\{b \gtrdot a, \neg a \gtrdot b\}$ is considered inconsistent, since making the occurrence of action *b* explicitly depend on the non-occurrence of action *a* is redundant. The causality condition $\Gamma_b = \{\neg a\}$ should only be specified if it is possible that action *a* occurs before action *b*. In our case, this possibility is excluded by the specification of causality condition $\Gamma_a = \{b\}$. Therefore, causality condition $\Gamma_b = \{\neg a\}$ should be replaced by $\Gamma_b = \{\surd\}$. A complementary explanation is given when the combination rules for the basic causality relations represented by $\neg b \gtrdot a$ are discussed. These rules show that the combination of elementary conditions $\gamma_a = b$ and $\gamma_b = \neg a$ is only allowed when the causality relation(s) of *a* (and *b*) define one or more additional alternative causality conditions.

The above is summarized in the following combination rule.

Combination Rule 2:

The combination of causality conditions $\Gamma_a = \{b\}$ and $\Gamma_b = \{\neg a\}$ is not allowed.

4.5.3 Combination rules for $\gamma_a = {}^{=}b$

The elementary causality condition $\gamma_a = {}^{=}b$ defines that action *a* is allowed to occur when action *b* occurs simultaneously with action *a*. Action *a* can only synchronize with action *b* if action *b* is allowed to synchronize with action *a*. This implies that the causality relation of action *b* should contain the synchronization condition ${}^{=}a$, i.e., ${}^{=}a \in \Gamma_b$.

The synchronization condition is a reciprocal condition for *a* and *b*. Therefore, a synchronization relation between two actions is defined by the combination of reciprocal synchronization conditions in the causality relations of both actions. Behaviour family $\{{}^{=}b \gtrdot a, {}^{=}a \gtrdot b\}$ represents a family of *synchronization relations*, which is called "synchronization of *a* and *b*".

The uncertainty associations $\upsilon_a({}^{=}b)$ and $\upsilon_b({}^{=}a)$ are related. Because synchronization requires that actions *a* and *b* occur at the same time, the uncertainty that both actions synchronize is the same for both actions, i.e., $\upsilon_a({}^{=}b) = \upsilon_b({}^{=}a)$. Therefore, the family "synchronization of *a* and *b*" consists of two distinct synchronization relations.

The above is summarized in the following combination rule.

Combination Rule 3:

The elementary causality condition $\gamma_a = {}^{=}b$ can be combined only with the elementary causality condition $\gamma_b = {}^{=}a$, which models the synchronization of *a* and *b*.

Consequently, if ${}^{=}b \in \Gamma_a$ then ${}^{=}a \in \Gamma_b$.

The values of the uncertainty associations $\upsilon_a({}^{=}b)$ and $\upsilon_b({}^{=}a)$ must be the same.

Implementation aspects

In order to implement a synchronization relation between actions *a* and *b*, the activities *a'* and *b'* that implement actions *a* and *b*, should agree on the time moment at which they will occur, respectively. This agreement can be implemented as the outcome of some negotiation between the activities *a'* and *b'*. In general, such a negotiation is needed since the simultaneous occurrence of actions *a* and *b* cannot be enforced by one of both actions. Dependent of the specific design problem, the implementation of this negotiation may, for example, vary from a simple centralized mechanism in which a common sub-activity *c'* of activities *a'* and *b'* dictates the moment of occurrence, to a more complex distributed mechanism in which multiple information units have to exchanged between activities *a'* and *b'* in order to implement a multi-round negotiation. After an agreement has been reached, the implementation should schedule the execution of activities *a'* and *b'* such that it is guaranteed that they terminate simultaneously at the agreed time moment.

4.5.4 Combination rules for $\gamma_a = \neg b$

The elementary causality condition $\gamma_a = \neg b$ defines that action *a* is allowed to occur when action *b* has not occurred before nor occurs simultaneously with action *a* in an execution. This causality condition makes the following two assumptions about the causality relation of action *b*:

1. it should allow the non-occurrence of *b*, or should allow that *b* occurs after *a* occurs. This implies that the following restrictions are imposed on the causality relation of action *b*, respectively:

   (i) the disabling condition $\neg a$ is an alternative causality condition of *b*, i.e., $\neg a \in \Gamma_b$;

   (ii) the enabling condition *a* is an alternative causality condition of *b*, i.e., $a \in \Gamma_b$; or

   (iii) a combination of (i) and (ii).

   In case this assumption does not hold, action *a* can never occur and should be removed from the behaviour definition;

2. it should allow that either *b* disables *a*, or that *b* occurs simultaneously with *a*. This implies that the following restrictions are imposed on the causality relation of action *b*, respectively:

   (i) the disabling condition $\neg a$ is an alternative causality condition of *b*, i.e., $\neg a \in \Gamma_b$;

   (ii) the synchronization condition $^{=}a$ is an alternative condition of *b*, i.e., $^{=}a \in \Gamma_b$; or

   (iii) a combination of (i) and (ii).

   In case this assumption does not hold, the causality condition of action *b* consists of the start condition √ or the enabling condition *a*. The combination of these causality conditions with causality condition $\Gamma_a = \{\neg b\}$ is not allowed by Combination Rules 1 and 2 (with alternated roles of *a* and *b*).

The architectural motivation behind both assumptions is that disabling condition $\neg b$ contains the (sub-)condition that actions *a* and *b* are not allowed to occur at the same time. This 'non-synchronization' condition is a reciprocal condition for *a* and *b*. Therefore, in order to allow that action *a* occurs due to the satisfaction of condition $\neg b$, the causality relation of action *b* should either

1. define that action *b* is not allowed to occur simultaneously with action *a*. This implies that the causality condition of action *b* must contain the disabling condition $\neg a$, i.e., $\neg a \in \Gamma_b$. In addition, the causality condition of action *b* may, optionally, define the enabling condition *a* as an alternative condition, i.e., $a \in \Gamma_b$.

   The causality condition of action *b* may not consist of the enabling condition *a* only, i.e., $\Gamma_b \neq \{a\}$, since in that case the occurrence of action *a* would not depend on the non-occurrence of *b*, as has been discussed before;

2. define that action b is only allowed to occur simultaneously with action a, i.e., $\Gamma_b = \{^{=}a\}$. In that case, the causality condition of action a should define the synchronization condition $^{=}b$ as an alternative condition, i.e., $\Gamma_a = \{\neg b, {}^{=}b\}$ (see Combination Rule 3). This implies that either actions a and b synchronize when conditions $^{=}a$ and $^{=}b$ are satisfied, or only action b is allowed to occur when condition $\neg a$ is satisfied;
3. define the disjunction of the two options above. In this case, the causality condition of action a is equal to $\Gamma_a = \{\neg b, {}^{=}b\}$ or $\Gamma_a = \{b, \neg b, {}^{=}b\}$ and the causality condition of action b is equal to $\Gamma_b = \{\neg a, {}^{=}a\}$ or $\Gamma_b = \{a, \neg a, {}^{=}a\}$. This implies that actions a and b must decide between their simultaneous occurrences or their non-simultaneous occurrences. In the latter case actions a and b should also decide which action is allowed to occur (first).

Based on the above restrictions, the following behaviour families are distinguished, which are structured into four categories representing the basic causality conditions involved.

Combinations of disabling conditions:

- family "choice between a and b": $\{\neg b \gtrdot a, \neg a \gtrdot b\}$. This family consists of choice relations which model the choice between the occurrence of a and the occurrence of b;

Combinations of disabling and enabling conditions:

- family "b disables a": $\{\neg b \gtrdot a, \neg a \vee a \gtrdot b\}$. This family consists of disabling relations which model the disabling of the occurrence of action a by the occurrence of action b. The occurrence of b disables the occurrence of a, similar to a choice relation. Instead, the occurrence of b is only disabled at the time moment when a occurs, since alternative enabling condition a allows b to occur after a has occurred;
- family "a disables b": $\{\neg b \vee b \gtrdot a, \neg a \gtrdot b\}$. This family is equivalent to family "b disables a" with alternated roles of a and b;
- family "interleaving of a and b": $\{\neg b \vee b \gtrdot a, \neg a \vee a \gtrdot b\}$. This family consists of interleaving relations which model the interleaved occurrences of a and b. Disabling conditions $\neg b$ and $\neg a$ enforce that either a or b must or may occur first. Subsequently, enabling conditions b and a allow a to occur after b has occurred or b to occur after a has occurred, respectively;

Combinations of disabling and synchronization conditions:

- family "sync-choice of a and b": $\{\neg b \vee {}^{=}b \gtrdot a, \neg a \vee {}^{=}a \gtrdot b\}$. This family models the possible synchronization of a and b, or the possible exclusive occurrences of a and b;
- family "a sync-excludes b": $\{\neg b \vee {}^{=}b \gtrdot a, {}^{=}a \gtrdot b\}$. This family models the possible synchronization of a and b, or the possible exclusive occurrence of a;
- family "b sync-excludes a": $\{{}^{=}b \gtrdot a, \neg a \vee {}^{=}a \gtrdot b\}$. This family is equivalent to family "a sync-excludes b" with alternated roles of a and b.

Combinations of disabling, enabling and synchronization conditions:

- family "a sync-enables b": $\{\neg b \vee {}^{=}b \gtrdot a, a \vee {}^{=}a \gtrdot b\}$. This family models the possible sequential ordering of a and b, or the possible synchronization of a and b. It extends

family "*a* sync-excludes *b*" with the alternative enabling condition *a*, which allows the occurrence of *b* after *a* has occurred;

- family "*b* sync-enables *a*": $\{b \vee {}^{=}b \gtrdot a, \neg a \vee {}^{=}a \gtrdot b\}$. This family is equivalent to family "*a* sync-enables *b*" with alternated roles of actions *a* and *b*;
- family "*a* sync-disables *b*": $\{b \vee \neg b \vee {}^{=}b \gtrdot a, \neg a \vee {}^{=}a \gtrdot b\}$. This family models the possible disabling of *b* by *a*, or the possible sequential ordering of *b* and *a*, or the possible synchronization of *a* and *b*. It extends family "sync-choice of *a* and *b*" with alternative enabling condition *b*, which allows action *a* to occur after action *b* has occurred;
- family "*b* sync-disables *a*": $\{\neg b \vee {}^{=}b \gtrdot a, a \vee \neg a \vee {}^{=}a \gtrdot b\}$. This family is equivalent to family "*a* sync-disables *b*" with alternated roles of actions *a* and *b*;
- family "temporal freedom of *a* and *b*": $\{b \vee \neg b \vee {}^{=}b \gtrdot a, a \vee \neg a \vee {}^{=}a \gtrdot b\}$. This family models the possible interleaved occurrences of *a* and *b*, or the possible synchronization of actions *a* and *b*. It extends family "sync-choice of *a* and *b*" with alternative enabling conditions *b* and *a*, which allow action *a* to occur after action *b* has occurred and action *b* to occur after action *a* has occurred, respectively.

The uncertainty associations $\upsilon_a(\neg b)$ and $\upsilon_b(\neg a)$ are related. The definition of distinct uncertainty values for $\upsilon_a(\neg b)$ and $\upsilon_b(\neg a)$ is considered inconsistent, since one allows the non-occurrences of *a* and *b* while the other requires that at least one action occurs. The possibility that *a* and *b* may not occur in a mutual disabling (choice) relation is a reciprocal condition for *a* and *b*. Therefore, the values of the uncertainty associations $\upsilon_a(\neg b)$ and $\upsilon_b(\neg a)$ should be the same.

Furthermore, the definition of distinct values for $\upsilon_a({}^{=}b)$ and $\upsilon_a(\neg b)$ is considered inconsistent, since in that case one uncertainty association allows the non-occurrences of *a* and *b*, while the other forbids this possibility. The same applies to $\upsilon_b({}^{=}a)$ and $\upsilon_b(\neg a)$. Consequently, the values of the uncertainty associations $\upsilon_a({}^{=}b)$, $\upsilon_b({}^{=}a)$, $\upsilon_a(\neg b)$ and $\upsilon_b(\neg a)$ should all be the same. Uncertainty associations $\upsilon_a(b)$ and $\upsilon_b(a)$ can be defined independently of uncertainty associations $\upsilon_a({}^{=}b)$, $\upsilon_b({}^{=}a)$, $\upsilon_a(\neg b)$ and $\upsilon_b(\neg a)$, since enabling conditions do not share any reciprocal sub-conditions with disabling or synchronization conditions.

The above is summarized in the following combination rule.

Combination Rule 4:

The elementary causality condition $\gamma_a = \neg b$ can be combined with:

(i) $\gamma_b = \neg a$, which models a choice between *a* and *b*;

(ii) a synchronization relation between *a* and *b* defined in disjunction with $\gamma_a = \neg b$;

(iii) a combination of *(i)* and *(ii)*;

(iv) $\gamma_b = a$, which models the enabling of *b* by *a*, if this combination is defined in disjunction with (i), (ii) or (iii).

Consequently, if $\neg b \in \Gamma_a$ then $\neg a \in \Gamma_b$ or ${}^{=}a \in \Gamma_b$, or both.

The definition of the uncertainty associations of actions *a* and *b* should obey the following rules:

- the values of the uncertainty associations $\upsilon_a(\bar{\ }b)$, $\upsilon_b(\bar{\ }a)$, $\upsilon_a(\neg b)$ and $\upsilon_b(\neg a)$ must be the same;
- uncertainty associations $\upsilon_a(b)$ and $\upsilon_b(a)$ can be defined independently of each other and independently of other uncertainty associations.

Implementation aspects

In order to implement a choice relation between actions *a* and *b*, the activities *a'* and *b'* that implement actions *a* and *b*, should agree on which one of both actions is allowed to occur, respectively. This agreement can be implemented as the outcome of some negotiation between the activities *a'* and *b'*. In general, such a negotiation is needed since the choice between the occurrence of actions *a* and *b* cannot be enforced by one of both actions. Dependent of the specific design problem, the implementation of this negotiation may, for example, vary from a simple mechanism in which the activity that takes the initiative 'wins', to a more complex mechanism which determines the choice using probabilistic parameters.

Once it has been decided which activity is allowed to occur, say *a'*, the implementation should disable (block) the occurrence of the other activity *b'*. This guarantees that only one activity can occur. However, in case the causality relation of action *b* contains the enabling condition *a* as an alternative condition, the implementation should enable (de-block) the occurrence of activity *b'* again immediately after activity *a'* has occurred.

In case the choice relation is defined in combination with a synchronization relation, the negotiation mentioned above becomes even more complex. In this case, activities *a'* and *b'* should first agree on a choice between two options: *(i)* the simultaneous or *(ii)* the non-simultaneous occurrence of actions *a* and *b*. In case option *(i)* is chosen, the implementation should schedule the execution of both activities such that it is guaranteed that they terminate simultaneously at the agreed time moment. In case option *(ii)* is chosen, the implementation should further decide upon which one of both actions is allowed to occur (first) as it has been discussed above.

4.6 Behaviour families

This section presents an overview of all behaviour families consisting of two actions *a* and *b* that can be defined when obeying the combination rules of the previous section. This overview is structured into two behaviour categories:

1. *a static behaviour* category, consisting of behaviour families in which *a* and *b* are either statically independent or statically related;
2. *a dynamic behaviour* category, consisting of behaviour families in which *a* and *b* are dynamically related.

Furthermore, the correspondence between these causality-based behaviour families and the execution-based behaviour families identified in Section 3.5 is indicated. In the remainder of this section we omit the adjective 'causality-based' if possible.

4.6.1 Static behaviour families

Static behaviour families are identified in Section 4.5. This section structures these behaviour families according to the notions of independent and related actions, and the derived notions of one-sided related and two-sided related actions.

Independent and related actions

Based on the modelling assumption that two actions are either related or independent in an execution, the static behaviour families of Section 4.5 are divided into two parts: the single family "independence of *a* and *b*" and the remaining families. Figure 4.8 depicts this division.

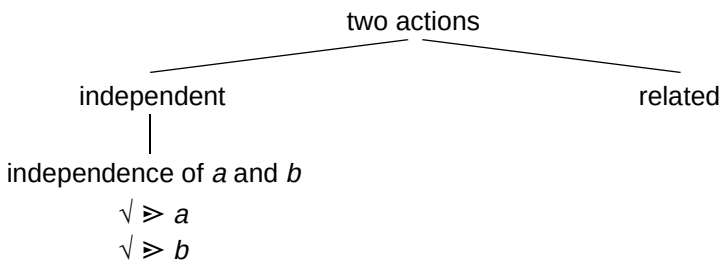


Figure 4.8: Structure of static behaviour families

The execution semantics of behaviour family "independence of *a* and *b*" and the remaining behaviour families is defined by the corresponding execution-based behaviour families in Tables 3.5 and 3.6, respectively.

Whether two actions *a* and *b* are independent or related is defined explicitly by the execution model, only in case both actions occur in the same execution. Execution $\boxed{a \quad b}$ represents the independent occurrences of *a* and *b*, while executions $\boxed{a \rightarrow b}$, $\boxed{b \rightarrow a}$ and $\boxed{a = b}$ represent the related occurrences of *a* and *b*. Instead, executions $\boxed{a \quad \bar{b}}$, $\boxed{\bar{a} \quad b}$ and $\boxed{\bar{a} \quad \bar{b}}$ may result from the execution of a behaviour in which *a* and *b* are independent or related.

As a consequence, the behaviour characteristics of 'independence' and 'action relation' are not mapped onto individual executions, but on combinations of executions. Figures 4.9(i) and (ii) illustrate this in terms of the execution table of two statically independent actions and the execution table of two statically related actions, respectively. Two statically independent actions are characterized by the mandatory execution $\boxed{a \quad b}$ and the impossible executions $\boxed{a \rightarrow b}$, $\boxed{b \rightarrow a}$ and $\boxed{a = b}$, while two statically related actions are characterized by the impossible execution $\boxed{a \quad b}$. Contrary to Chapter 3, impossible executions are indicated by a big cross.

One-sided and two-sided relations

Based on the definition of two related actions in Section 2.5.1, two types of relations can be distinguished:

1. *one-sided relations*, in which action *a* depends on action *b* and action *b* is independent of action *a*, or vice-versa; and

$a\ \ b$	$a \rightarrow b$	$b \rightarrow a$	$a = b$	$a\ \ \bar{b}$	$\bar{a}\ \ b$	$\bar{a}\ \ \bar{b}$
1						
1						
1						
1						

(i) two statically independent actions

$a\ \ b$	$a \rightarrow b$	$b \rightarrow a$	$a = b$	$a\ \ \bar{b}$	$\bar{a}\ \ b$	$\bar{a}\ \ \bar{b}$

(ii) two statically related actions

Figure 4.9: Execution tables of two statically independent and two statically related actions

2. *two-sided relations*, in which action a depends on action b and action b depends on action a.

For example, relation $\{\surd \rightarrow a, a \rightarrow b\}$ is a one-sided relation, and relation $\{\neg b \rightarrow a, \neg a \rightarrow b\}$ is a two-sided relation.

Figure 4.10 depicts the decomposition of the behaviour families of two related actions a and b into behaviour families of one-sided relations and behaviour families of two-sided relations. Two behaviour families of one-sided relations are identified, i.e., "a enables b" and "b enables a", which comprise eight distinct relations.

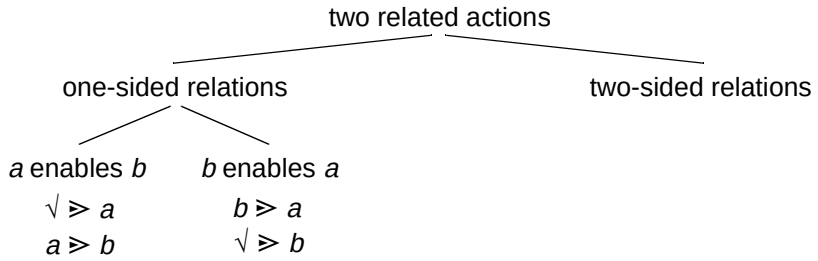


Figure 4.10: Structure of static behaviour families (2)

Figure 4.11 depicts the execution table of two statically related actions in more detail. This table is divided into two complementary parts: a part representing the eight possible one-sided relations between a and b, and a part representing all possible two-sided relations between a and b.

In principle, any combination of the executions $\boxed{a \rightarrow b}$, $\boxed{b \rightarrow a}$, $\boxed{a = b}$, $\boxed{a\ \ \bar{b}}$, $\boxed{\bar{a}\ \ b}$ and $\boxed{\bar{a}\ \ \bar{b}}$ can be defined by means of a two-sided relation between a and b, including the ones represented by one-sided relations. For example, combination $\boxed{a \rightarrow b}$, $\boxed{a\ \ \bar{b}}$ and $\boxed{\bar{a}\ \ \bar{b}}$ can be defined by means of the two-sided relation $\{\neg b \rightarrow a, a \rightarrow b\}$. We have decided, however, that the definition of a two-sided relation is inconsistent (redundant) in case the corresponding set of possible executions can be defined in terms of a one-sided relation. As a consequence, one-sided and two-sided relations are represented by complementary parts of the execution table.

Figure 4.12 depicts an overview of the two-sided behaviour families identified in Section 4.5. The structuring of these behaviour families into multiple levels is explained in Section 4.7.3. For convenience, family names are abbreviated.

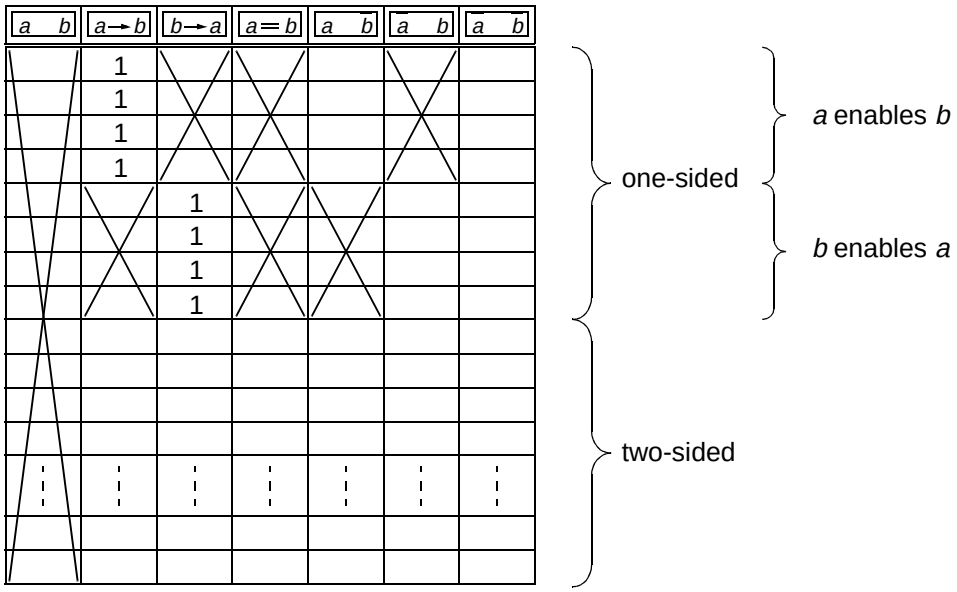


Figure 4.11: Execution table of two statically related actions

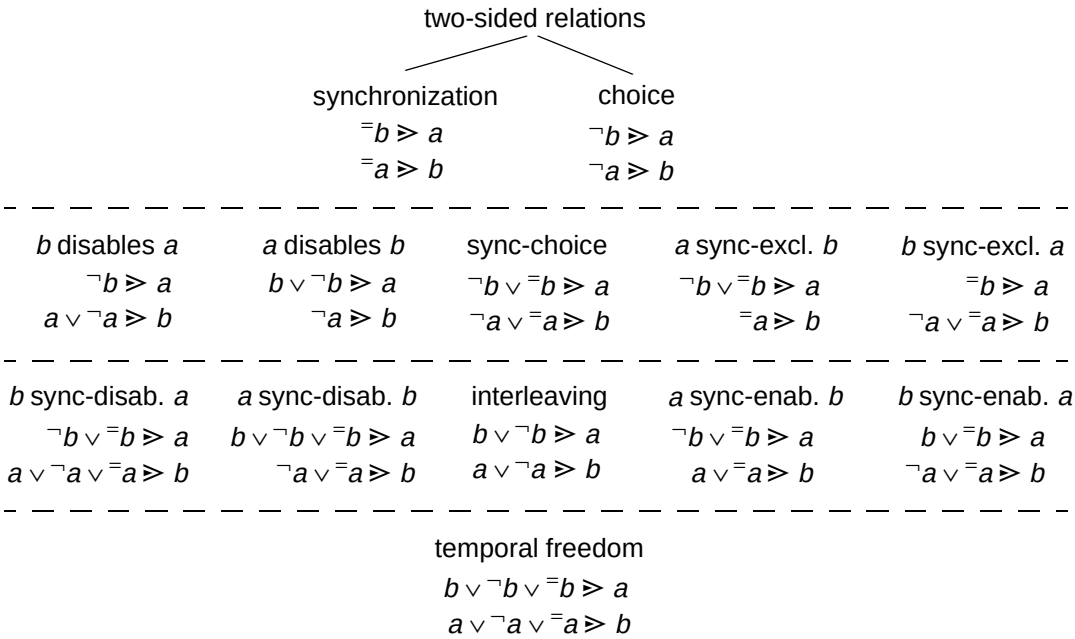


Figure 4.12: Structure of static behaviour families (3)

In addition, the execution model identifies in Table 3.6 the behaviour families: "*a* excludes *b*", "*b* excludes *a*" and "deadlock". These behaviour families did not result from our causality-based line of reasoning, since they violate the principle 'what does not happen should not be defined' and therefore are not useful from a prescriptive (design) perspective. Instead, it is useful to consider them for analytical purposes and in order to allow the systematic structuring of complex action relations into disjunctions of elementary action relations in Section 4.7. In terms of causality relations, the three families are defined as:

- family "*a* excludes *b*": $\{{}^{\neg}b \gtrdot a, \dagger \gtrdot b\}$. This family models a relation between action *a* and *b* in which the occurrence of action *b* is always excluded, i.e., "the (mere) exist-

ence of a excludes the occurrence of b";

- family "b excludes a": $\{\dagger \gtrdot a, \neg a \gtrdot b\}$. This family is equivalent to family "a excludes b" with alternated roles of actions a and b.
- family "deadlock (of a and b)": $\{\dagger \gtrdot a, \dagger \gtrdot b\}$. This family models a relation in which both the occurrences of a and b are impossible.

The dagger symbol $\dagger$ represents an impossible condition which is always dissatisfied. The "dagger" is a zero element of the *or*-operator, i.e., $\dagger \vee \gamma = \gamma$. Furthermore, the uncertainty association of an impossible condition is left undefined, since this condition does never allow the occurrence of the result action.

Relations of the families "a excludes b" and "b excludes a" are also called *asymmetric exclusion* relations. An asymmetric exclusion relation between actions a and b should be defined in combination (disjunction) with the complementary asymmetric exclusion relation between a and b, or the synchronization relation between a and b.

'One-and-a-half' sided relations

An additional category of behaviour families is obtained by the disjunction of behaviours from an one-sided behaviour family with behaviours from a two-sided behaviour family. For simplicity, these behaviour families are called '*one-and-a-half*' sided behaviour families.

The disjunction of two behaviours B_1 and B_2 is represented as $B_1 \square B_2$, where $\square$ is called the *or*-operator on behaviours. The or-operator $\square$ is defined as follows:

$$B_1 \square B_2 = \{\Gamma 1_a \vee \Gamma 2_a \rightarrow a\ [\upsilon 1_a, \upsilon 2_a], \Gamma 1_b \vee \Gamma 2_b \rightarrow b\ [\upsilon 1_b, \upsilon 2_b]\},$$
if no conflicting uncertainty associations are defined by $\upsilon 1_a$, $\upsilon 2_a$, $\upsilon 1_b$ and $\upsilon 2_b$;

$$= \text{undefined, if otherwise,}$$

with: $B_1 = \{\Gamma 1_a \rightarrow a\ [\upsilon 1_a], \Gamma 1_b \rightarrow b\ [\upsilon 1_b]\}$ and $B_2 = \{\Gamma 2_a \rightarrow a\ [\upsilon 2_a], \Gamma 2_b \rightarrow b\ [\upsilon 2_b]\}$.

For example, the disjunction of enabling relation $\{\surd \rightarrow a, a \rightarrow b\}$ and disabling relation $\{\neg b \rightarrow a, a \vee \neg a \rightarrow b\}$ is equal to relation $\{\surd \vee \neg b \rightarrow a, a \vee \neg a \rightarrow b\}$. In case of disabling relation $\{\neg b \rightarrow a, a_! \vee \neg a \rightarrow b\}$ the disjunction would be undefined, since uncertainty association $\upsilon_b(a) = \mathit{must}$ of this relation is in conflict with uncertainty association $\upsilon_b(a) = \mathit{may}$ of the enabling relation.

The *or*-operator on behaviours is a generalization of the *or*-operator on causality conditions. The *or*-operator $\square$ is idempotent, commutative and associative.

In general, a 'one-and-a-half' sided behaviour family can be represented by one of the following alternative forms, with $\{\Gamma_a \gtrdot a, \Gamma_b \gtrdot a\}$ being a two-sided behaviour family:

- $\{\surd \vee \Gamma_a \gtrdot a, a \vee \Gamma_b \gtrdot b\}$;
- $\{b \vee \Gamma_a \gtrdot a, \surd \vee \Gamma_b \gtrdot b\}$.

The execution semantics of these relations is identical to the execution semantics of two-sided behaviour families of the following forms, respectively:

- $\{\neg b \vee \Gamma_a \gtrdot a, a \vee \Gamma_b \gtrdot b\}$;
- $\{b \vee \Gamma_a \gtrdot a, \neg a \vee \Gamma_b \gtrdot b\}$.

This implies that ‘one-and-a-half’ sided relations cannot be distinguished from two-sided relations in the execution model. For example, relation $\{\surd \vee \neg b \rightarrow a, a \vee \neg a \rightarrow b\}$ and relation $\{\neg b \rightarrow a, a \vee \neg a \rightarrow b\}$ have the same execution semantics: $\{\boxed{a \rightarrow b}, \boxed{a \quad \bar{b}}, \boxed{\bar{a} \quad b}, \boxed{\bar{a} \quad \bar{b}}\}$.

4.6.2 Dynamic behaviour families

Dynamic behaviour families consist of behaviours in which actions *a* and *b* are dynamically related. These behaviours define executions in which *a* and *b* are independent and executions in which *a* and *b* are related. The following categories of dynamic behaviour families are distinguished:

1. behaviour families consisting of the disjunctions of behaviours from the behaviour family “independence of *a* and *b*” with behaviours from a one-sided or two-sided behaviour family;
2. behaviour families consisting of the disjunctions of behaviours from both one-sided behaviour families.

The former category consists of behaviour families of the form $\{\surd \vee \Gamma_a \gtrdot a, \surd \vee \Gamma_b \gtrdot b\}$, with $\{\Gamma_a \gtrdot a, \Gamma_b \gtrdot b\}$ being either a one-sided or a two-sided behaviour family. The second category of behaviour families consists of a single behaviour family $\{\surd \vee b \gtrdot a, \surd \vee a \gtrdot b\}$.

The characteristic of ‘dynamically related’ actions can be derived from an execution-based behaviour by the presence of execution $\boxed{a \quad b}$ and one or more of the executions $\boxed{a \rightarrow b}$, $\boxed{b \rightarrow a}$ and $\boxed{a = b}$. The combination of these executions define that the occurrences of *a* and *b* are either independent or related. Exceptions are the latter three execution-based behaviours of Table 3.5 in Section 3.5.1, which represent the execution semantics of three static independent behaviours of family $\{\surd \gtrdot a, \surd \gtrdot b\}$ and represent the execution semantics of the dynamic asymmetric exclusion relations $\{\surd_! \vee \neg b_! \rightarrow a, \surd_? \vee \dagger \rightarrow b\}$ and $\{\surd_? \vee \dagger \rightarrow a, \surd_! \vee \neg a_! \rightarrow b\}$ and of the dynamic choice relation $\{\surd_? \vee \neg b_? \rightarrow a, \surd_? \vee \neg a_? \rightarrow b\}$, respectively.

In general, the execution semantics of causality relations does not allow one to discern between different dynamic action relations. For example, the execution semantics of the dynamic action relations $\{\surd \vee b \vee \neg b \rightarrow a, \surd \vee a \vee \neg a \rightarrow b\}$, $\{\surd \vee b \vee \neg b \rightarrow a, \surd \vee a \rightarrow b\}$, $\{\surd \vee b \rightarrow a, \surd \vee a \vee \neg a \rightarrow b\}$, and $\{\surd \vee b \rightarrow a, \surd \vee a \rightarrow b\}$ are mapped onto the same execution set $\{\boxed{a \quad b}, \boxed{a \rightarrow b}, \boxed{b \rightarrow a}, \boxed{a \quad \bar{b}}, \boxed{\bar{a} \quad b}, \boxed{\bar{a} \quad \bar{b}}\}$.

Figure 4.13 depicts an overview of the types of behaviours consisting of two actions that have been considered so far.

4.6.3 Formal definition

The *or*-operator ❑ on behaviours of two actions *a* and *b* is formally defined below.

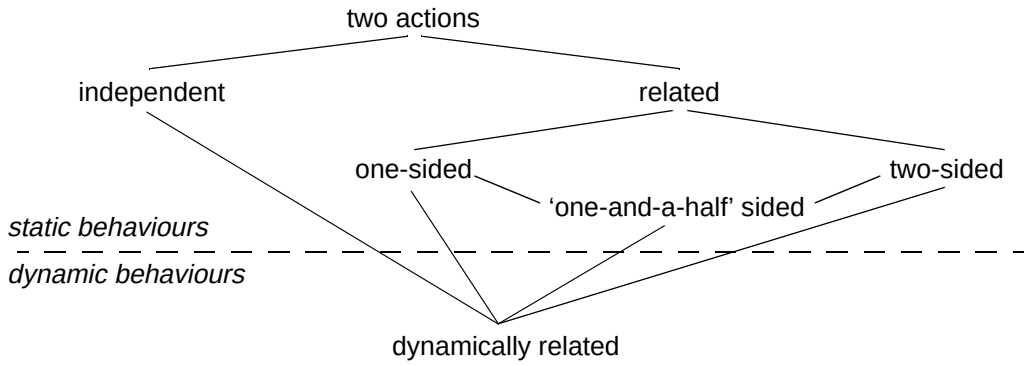


Figure 4.13: Structure of static and dynamic behaviour families

Definition 4.15 The disjunction of two behaviours $B_1 = \{\langle a, \Gamma 1_a, \upsilon 1_a\rangle, \langle b, \Gamma 1_b, \upsilon 1_b\rangle\}$ and $B_2 = \{\langle a, \Gamma 2_a, \upsilon 2_a\rangle, \langle b, \Gamma 2_b, \upsilon 2_b\rangle\}$ is defined as:

$$B_1 \,\square\, B_2 = \{\langle a, \Gamma 1_a \cup \Gamma 2_a, \upsilon 1_a \cup \upsilon 2_a\rangle, \langle b, \Gamma 1_b \cup \Gamma 2_b, \upsilon 1_b \cup \upsilon 2_b\rangle\},$$
$$\text{if } \forall \gamma \in \Gamma 1_a \cap \Gamma 2_a \mid \upsilon 1_a(\gamma) = \upsilon 2_a(\gamma) \text{ and } \forall \gamma \in \Gamma 1_b \cap \Gamma 2_b \mid \upsilon 1_b(\gamma) = \upsilon 2_b(\gamma)\ ;$$
$$= \text{undefined, if otherwise.}$$ ■

Property 4.16 The *or*-operator $\square : \boldsymbol{B} \times \boldsymbol{B} \rightarrow \boldsymbol{B}$ is idempotent, commutative and associative.

Proof: These properties are inherited from the union operator $\cup$ on sets. ■

Two definitions of the execution semantics of the impossible condition $\dagger$ of an action a are given below: (i) one which is independent of the context of action a, and (ii) one for the specific case of a behaviour of two actions a and b.

Definition 4.17 The executions semantics of the impossible condition $\dagger$ of an action a is defined as: $[\![\langle a, \surd\rangle]\!] = \{\boxed{\bar{a}\quad}\}$. ■

Definition 4.18 Assuming a behaviour with two actions a and b, the execution semantics of the impossible condition $\dagger$ of action a is defined as: $[\![\langle a, \surd\rangle]\!] = \{\boxed{\bar{a}\quad b}, \boxed{\bar{a}\quad \bar{b}}\}$. ■

4.7 Alternative behaviours

This section defines the notion of alternative behaviour as an additional building block to (de)compose behaviour models. The motivation for introducing alternative behaviours in addition to the more elementary building block of causality relation is to provide more insight in the structure of (complex) behaviour models. Furthermore, the notion of alternative behaviour proofs convenient to define consistency rules for behaviour models with multiple actions in Chapter 5, and to develop the behaviour refinement design operation in Chapter 8.

4.7.1 Definition

An *alternative behaviour BA* consisting of two actions a and b, is defined as an elementary behaviour which consists of the combination of two elementary causality relations, such

that the causality conditions of a and b consist of a single alternative causality condition, and the corresponding uncertainty association has the value *may*:

$BA = \{\gamma_a \rightarrow a\ [\upsilon_a(a) = may],\ \gamma_b \rightarrow b\ [\upsilon_b(b) = may]\}$,

with $\gamma_a \in \{\surd, b, \neg b, {}^{=}b\}$ and $\gamma_b \in \{\surd, a, \neg a, {}^{=}a\}$.

An alternative behaviour defines a possible combination of elementary causality conditions due to which one or both actions may occur in an execution. The possible combinations are determined by the combination rules of Section 4.5.

Separate consideration of causality conditions and uncertainty attributes

In this thesis, alternative behaviours are used to structure the causality conditions of actions. Due to the may-interpretation of causality conditions, we consider alternative behaviours with may uncertainty associations only. Must uncertainty associations are considered separately as additional constraints on compositions of alternative behaviours.

The above implies that a behaviour definition $B = \{\Gamma_a \rightarrow a\ [\upsilon_a],\ \Gamma_b \rightarrow b\ [\upsilon_b]\}$ is divided into two orthogonal parts:

- a *causality condition part* $B_\Gamma = \{\Gamma_a \rightarrow a, \Gamma_b \rightarrow b\}$, which defines the causality conditions of the actions in B and assumes the values of all uncertainty associations equal to the value *may*;
- an *uncertainty attribute part* $B_\upsilon = \{\upsilon_a, \upsilon_b\}$, which defines the set of uncertainty attributes of the actions in B. Alternatively, the uncertainty attributes may be restricted to the set of *must* uncertainty associations, since *may* uncertainty associations define no (additional) constraints on the execution of the involved actions.

The execution semantics of behaviour definition B is equal to the cross-conjunction of the execution semantics of its causality condition part B_Γ and the execution semantics of its uncertainty attribute part B_υ. A proof of this property is given in Section 4.7.5.

4.7.2 Identification

The following alternative behaviours of two actions a and b are identified:

- the independent behaviour:

  $\{\surd \rightarrow a, \surd \rightarrow b\}$, or: (a) (b) ;
- the one-sided enabling relations "a enables b" and "b enables a":

  $\{\surd \rightarrow a, a \rightarrow b\}$, or: (a)⟶(b) ((a)-1→(b)) ;

  $\{b \rightarrow a, \surd \rightarrow b\}$, or: (a)⟵(b) ((a)←1-(b)) ;
- the two-sided synchronization relation "synchronization of a and b":

  $\{{}^{=}b \rightarrow a, {}^{=}a \rightarrow b\}$, or: (a)═══(b) ;
- the two-sided choice or (symmetric) exclusion relation "choice between a and b":

$\{\neg b \to a, \neg a \to b\}$, or: $a \vdash\!\dashv b$;

- the two-sided enabling relations "*a* enables *b*" and "*b* enables *a*":

  $\{\neg b \to a, a \to b\}$, or: $a \longrightarrow b$ ($a \xrightarrow{2} b$) ;

  $\{b \to a, \neg a \to b\}$, or: $a \longleftarrow b$ ($a \xleftarrow{2} b$) ;

- the two sided asymmetric exclusion relations "*a* excludes *b*" and "*b* excludes *a*":

  $\{\neg b \to a, \dagger \to b\}$, or: $a \leftarrow\!\vdash \not{b}$;

  $\{\dagger \to a, \neg a \to b\}$, or: $\not{a} \vdash\!\rightarrow b$;

A graphical notation is used in addition to the textual notation, because the graphical notation is easier to conceive:

- the graphical representation $a \vdash\!\dashv b$ is a shorthand of $a \leftarrow\!\vdash\!\dashv\!\rightarrow b$;
- the graphical representation $a = b$ is a shorthand of $a \leftarrow\!\rightarrow b$;
- the graphical representations $a \xrightarrow{1} b$ and $a \xrightarrow{2} b$ explicitly denote a one-sided and a two-sided enabling relation;
- an impossible action *a* is graphically represented as: $\not{a}$.

Strictly, the choice relation $a \vdash\!\dashv b$ is not an elementary behaviour, but can be composed as the disjunction of the asymmetric exclusion relations $a \leftarrow\!\vdash \not{b}$ and $\not{a} \vdash\!\rightarrow b$. Despite that, we consider this behaviour as an alternative behaviour since it is a useful and intuitive building block in the composition of more complex behaviours. The two-sided enabling relations $a \xrightarrow{2} b$ and $a \xleftarrow{2} b$, and the asymmetric exclusion relations $a \leftarrow\!\vdash \not{b}$ and $\not{a} \vdash\!\rightarrow b$ should always be defined in disjunction with other alternative behaviours, according to the combination rules of Section 4.5.

The alternative behaviours identified above constitutes the domain of alternative behaviours ***BA*(*a*, *b*)**, and is ranged over by *BA*. The alternative behaviours defining two related actions *a* and *b* are also called *alternative action relations*.

4.7.3 Compositions of alternative behaviours

In general, the causality condition part of a behaviour definition *B* consisting of two actions *a* and *b* can be decomposed into a disjunction of one or more alternative behaviours, such that

$B_\Gamma = \square\, Alt(B_\Gamma)$,

where function *Alt* denotes the set of alternative behaviours in which B_Γ can be decomposed, such that:

$Alt(B_\Gamma) = \{BA = \{\gamma_a \to a, \gamma_b \to b\} \mid \gamma_a \in \Gamma_a, \gamma_b \in \Gamma_b, BA \in \mathbf{BA(a, b)}\}$.

Alternative behaviours $Alt(B_\Gamma)$ represent all possible combinations of alternative causality conditions of behaviour B_Γ due to which one or both actions can occur. A behaviour definition B_Γ which can not be decomposed into a disjunction of alternative behaviours is considered an inconsistent behaviour, since in that case an alternative causality condition is

defined for some action which can not be combined with an alternative causality condition of another action, and therefore should be removed from the behaviour definition.

An alternative behaviour in $Alt(B_\Gamma)$ directly represents the *maximal execution* allowed by this alternative behaviour, when also the choice relation is decomposed into the more elementary asymmetric exclusion relations and when assuming that all (possible) actions in an alternative behaviour *must* occur. Figure 4.14 illustrates this for the interleaving relation $B_\Gamma = \{b \vee \neg b \rightarrow a, a \vee \neg a \rightarrow b\}$. This relation can be decomposed into the alternative action relations $\textcircled{a}\!\longrightarrow\!\textcircled{b}$, $\textcircled{a}\!\longleftarrow\!\textcircled{b}$, $\textcircled{a}\!\leftarrow\!\!+\!\!-\!\cancel{\textcircled{b}}$ and $\cancel{\textcircled{a}}\!-\!\!+\!\!\rightarrow\!\textcircled{b}$, which represent the maximal executions $\boxed{a \rightarrow b}$, $\boxed{b \rightarrow a}$, $\boxed{a \quad \bar{b}}$ and $\boxed{\bar{a} \quad b}$, respectively.

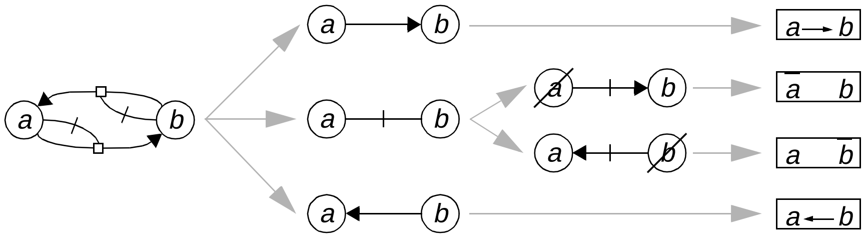


Figure 4.14: Decomposition of interleaving

These maximal executions and all their possible prefixes constitute the executions allowed by B_Γ. The prefixes of an execution χ are the executions derived from χ in which one or more action occurrences become non-occurrences. For example, executions $\boxed{a \quad \bar{b}}$ and $\boxed{\bar{a} \quad \bar{b}}$ are the prefixes of execution $\boxed{a \rightarrow b}$. Consequently, the possible executions of the interleaving relation of Figure 4.14 consist of the represented maximal executions and execution $\boxed{\bar{a} \quad \bar{b}}$. The prefix relation is defined in Section 3.8.4.

Execution semantics

The above property implies that the execution semantics of the causality condition part of a behaviour definition is equal to the union of the execution semantics of the alternative behaviours in which this behaviour can be decomposed. This property is formally defined and proven in Section 4.7.5.

For example, the execution semantics of the causality condition part of disabling relations $\{b \vee \neg b \gtrdot a, \neg a \gtrdot b\}$ can be determined as follows:

$$\begin{aligned} [\![\{b \vee \neg b \rightarrow a, \neg a \rightarrow b\}]\!] &= [\![\{b \rightarrow a, \neg a \rightarrow b\}]\!] \cup [\![\{\neg b \rightarrow a, \neg a \rightarrow b\}]\!] \\ &= \{\boxed{b \rightarrow a}, \boxed{\bar{a} \quad b}, \boxed{\bar{a} \quad \bar{b}}\} \cup \{\boxed{a \quad \bar{b}}, \boxed{\bar{a} \quad b}, \boxed{\bar{a} \quad \bar{b}}\} \\ &= \{\boxed{b \rightarrow a}, \boxed{\bar{a} \quad b}, \boxed{a \quad \bar{b}}, \boxed{\bar{a} \quad \bar{b}}\}. \end{aligned}$$

Whereas the execution semantics of a behaviour is uniquely determined by its constituent alternative behaviours, the inverse it not true. In general, it is impossible to derive uniquely the constituent alternative behaviours from the execution semantics of this behaviour. As explained in Section 4.6, the execution semantics of different causality-based behaviours may be mapped onto the same execution set.

Review of identified behaviour families

The causality condition part of the behaviour families discussed in Section 4.6 can be decomposed into:

- the independent alternative behaviour, in case of the static behaviour family "independence of *a* and *b*"; or
- a disjunction of one or more alternative action relations, in case of all other static behaviour families; or
- a disjunction of the independent alternative behaviour and one or more alternative action relations, in case of the dynamic behaviour families.

The hierarchy of Figure 4.12 represents the number of alternative action relations from which the causality condition parts of two-sided relations are composed. Each hierarchical level n represents the two-sided relations composed of n alternative action relations. For example, the causality condition part of the interleaving relations $\{b \vee \neg b \gg a, a \vee \neg a \gg b\}$ can be decomposed into the alternative action relations $\{\neg b \rightarrow a, a \rightarrow b\}$, $\{b \rightarrow a, \neg a \rightarrow b\}$ and $\{\neg b \rightarrow a, \neg a \rightarrow b\}$.

4.7.4 Graphical shorthand notations

Figure 4.15 shows the graphical shorthand notations of some frequently used two-sided relations.

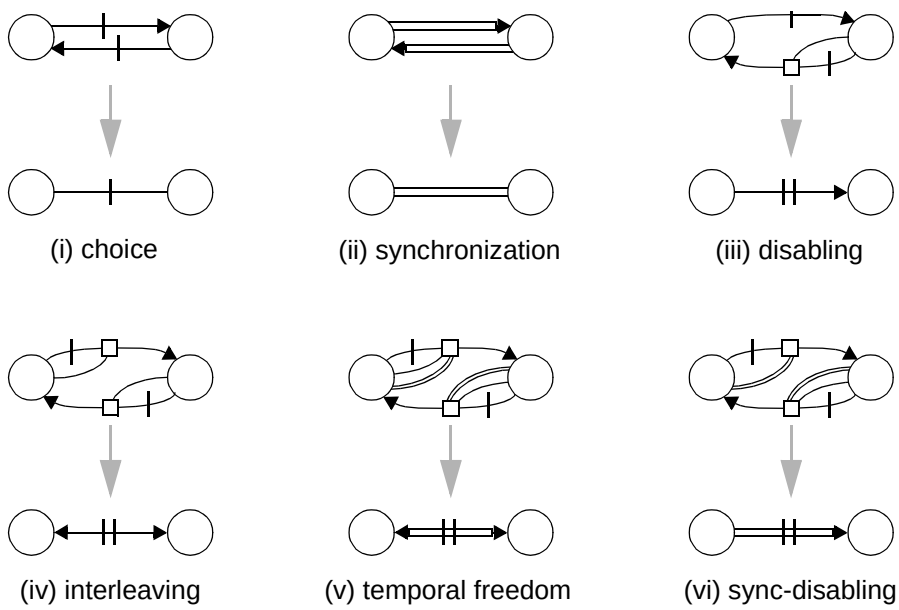


Figure 4.15: Graphical shorthand notations

4.7.5 Formal definition

The decomposition of a behaviour definition into a causality condition part and an uncertainty attribute part is formally defined below.

Property 4.19 A behaviour definition $B = \{\langle a, \Gamma_a, \upsilon_a\rangle, \langle b, \Gamma_b, \upsilon_b\rangle\}$ can be decomposed into

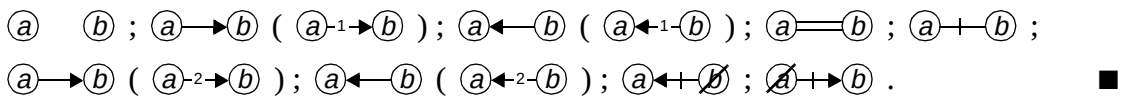
- a causality condition part $B_\Gamma = \{\langle a, \Gamma_a, \upsilon_a'\rangle, \langle b, \Gamma_b, \upsilon_b'\rangle\}$, with:
  $\forall\gamma_a \in \Gamma_a \mid \upsilon_a'(\gamma_a) = may$ and $\forall\gamma_b \in \Gamma_b \mid \upsilon_b'(\gamma_b) = may$;
- an uncertainty attribute part $B_\upsilon = \{\upsilon_a, \upsilon_b\}$,

such that: $[\![B]\!] = [\![B_\Gamma]\!] \otimes [\![B_\upsilon]\!]$, with $[\![B_\upsilon]\!] = \otimes\{[\![\upsilon]\!] \mid \upsilon \in B_\upsilon\}$.

Proof: This property follows immediately from Definitions 4.3 and 4.4, and the may-interpretation of causality conditions. ■

The domain of alternative behaviours consisting of two actions is summarized below.

Definition 4.20 Domain ***BA(a, b)*** consists of the following alternative behaviours for actions a and $b \in \boldsymbol{A}$, with $a \neq b$:



The following property formally defines that the execution semantics of the causality condition part of a behaviour definition is equal to the union of the execution semantics of the alternative behaviours in which this behaviour can be decomposed. This property is based on some properties of inconsistent combinations of elementary causality conditions, which are defined first.

Property 4.21 The following inconsistent combinations of elementary causality conditions of two actions a and b are identified:

1. $C_1 = \{\langle a, \gamma_a\rangle, \langle b, \gamma_b\rangle \mid \langle\gamma_a, \gamma_b\rangle \in \{\langle\surd, {}^{=}a\rangle, \langle{}^{=}b, \surd\rangle, \langle b, a\rangle, \langle b, {}^{=}a\rangle, \langle{}^{=}b, a\rangle, \langle\neg b, {}^{=}a\rangle, \langle{}^{=}b, \neg a\rangle\}$;
2. $C_2 = \{\langle a, \gamma_a\rangle, \langle b, \gamma_b\rangle \mid \langle\gamma_a, \gamma_b\rangle \in \{\langle\surd, \neg a\rangle, \langle\neg b, \surd\rangle\}$.

The execution semantics of these inconsistent combinations is defined as:

(i) $\forall\{\langle a, \gamma_a\rangle, \langle b, \gamma_b\rangle\} \in C_1 \mid [\![\langle a, \gamma_a\rangle]\!] \otimes [\![\langle b, \gamma_b\rangle]\!] = \{\boxed{\bar{a} \quad \bar{b}}\}$;

(ii) $\forall\{\langle a, \gamma_a\rangle, \langle b, \gamma_b\rangle\} \in C_2 \mid [\![\langle a, \gamma_a\rangle]\!] \otimes [\![\langle b, \gamma_b\rangle]\!] = \{\boxed{\bar{a} \quad b}, \boxed{a \quad \bar{b}}, \boxed{\bar{a} \quad \bar{b}}\}$.

Proof: Follows immediately from the combination rules and the execution semantics of the elementary causality conditions. ■

Property 4.22 Assuming the causality condition part B_Γ of a behaviour B obeys the combination rules of Sections 4.5, the following property holds:

$[\![B_\Gamma]\!] = \cup\{[\![BA]\!] \mid BA \in Alt(B_\Gamma)\}$.

Proof: The proof of this property is based on Definitions 4.3 and 4.4, Property 4.21, the may-interpretation of causality conditions, and the distributivity of $\otimes$ over $\cup$.

$$
\begin{aligned}
[\![B_\Gamma]\!] &= [\![\langle a, \Gamma_a\rangle]\!] \otimes [\![\langle b, \Gamma_b\rangle]\!] \otimes [\![\upsilon_a']\!] \otimes [\![\upsilon_b']\!] \\
&= \cup\{[\![\langle a, \gamma_a\rangle]\!] \mid \gamma_a \in \Gamma_a\} \otimes \cup\{[\![\langle b, \gamma_b\rangle]\!] \mid \gamma_b \in \Gamma_b\} \\
&= \cup\{[\![\langle a, \gamma_a\rangle]\!] \otimes [\![\langle b, \gamma_b\rangle]\!] \mid \gamma_a \in \Gamma_a, \gamma_b \in \Gamma_b\} \\
&= \cup\{[\![\langle a, \gamma_a\rangle]\!] \otimes [\![\langle b, \gamma_b\rangle]\!] \mid \gamma_a \in \Gamma_a, \gamma_b \in \Gamma_b, \gamma_a \text{ and } \gamma_b \text{ are consistent}\} \\
&= \cup\{[\![BA]\!] \mid BA \in Alt(B_\Gamma)\}
\end{aligned}
$$

$Alt(B_\Gamma)$ represents all consistent combinations of elementary causality conditions. Inconsistent combinations only add execution $\boxed{\bar{a} \quad \bar{b}}$, or executions $\boxed{\bar{a} \quad b}$, $\boxed{a \quad \bar{b}}$, $\boxed{\bar{a} \quad \bar{b}}$ in case of combinations $\langle\surd, \neg a\rangle$ and $\langle\neg b, \surd\rangle$ (see Property 4.21). However, execution $\boxed{\bar{a} \quad \bar{b}}$ is already allowed by all consistent combinations. Furthermore, in case combination $\langle\surd, \neg a\rangle$ is part of the causality conditions of actions a and b, one of the consistent combinations $\langle\surd, \surd\rangle$ or $\langle\surd, a\rangle$ and one of the consistent combinations $\langle\neg b, \neg a\rangle$ or $\langle\dagger, \neg a\rangle$ must be defined, which implies that executions $\boxed{\bar{a} \quad b}$, $\boxed{a \quad \bar{b}}$, $\boxed{\bar{a} \quad \bar{b}}$ are already allowed by these consistent combinations. A similar reasoning applies to the inconsistent combination $\langle\neg b, \surd\rangle$. Consequently, inconsistent combinations do not contribute to the execution semantics of B_Γ. ■

4.8 Conclusions

We introduce the causality relation concept as the basic building block for the modelling of action relations. In this chapter, a causality relation defines the causality condition and uncertainty attribute of an action. A behaviour is defined by a set of causality relations, one per action of the behaviour.

The causality condition of an action defines how the occurrence of this action depends on the occurrences or non-occurrences of other actions. We have identified four elementary causality conditions: the start condition, the enabling condition, the disabling condition and the synchronization condition. These conditions can be composed into more complex conditions using the *and*- and *or*-operator. Since we have only considered behaviours of two actions in this chapter, the full elaboration of these operators is deferred to Chapter 5.

The uncertainty attribute of an action defines for each alternative condition of this action the uncertainty that the action occurs when this condition is satisfied. Two uncertainty values are distinguished: the *must* value defines that the action must occur, and the *may* value defines that the action may or may not occur when the associated condition is satisfied.

In this chapter, we perform an elaborate and systematic analysis of the various types of relations that can be defined between two actions using the causality relation concept. Since some combinations of causality relations render inconsistent behaviour models, we have defined rules for the combination of the causality conditions and uncertainty attributes of two actions. These rules allow one to determine consistent combinations of causality relations and eliminate inconsistent ones.

The consistent combinations are structured in terms of a hierarchy of behaviour families. This structure distinguishes between independent, statically related and dynamically related actions. Statically related actions are further divided into one-sided and two-sided related actions. The obtained structure of independent and static behaviour families is equal to the structure of corresponding execution-based behaviour families identified in Chapter 3.

Chapter 5

Monolithic behaviours

This chapter extends the definition of the *and*- and *or*-operators introduced in Chapter 4 to include the composition of causality conditions involving multiple actions. This extension allows the modelling of behaviours with more than two actions.

The scope of this chapter is restricted to the modelling of finite monolithic behaviours. The modelling of (infinitely) repetitive behaviours is discussed in Chapter 9. Furthermore, only the temporal ordering and uncertainty characteristics of behaviours are considered.

The structure of this chapter is as follows. Section 5.1 discusses the modelling of conjunctions of causality conditions using the *and*-operator. Section 5.2 discusses the modelling of disjunctions of causality conditions using the *or*-operator. Section 5.3 presents combination rules for modelling consistent causality conditions. Section 5.4 presents combination rules for modelling consistent uncertainty associations. Section 5.5 discusses related work. And Section 5.6 presents the conclusions.

5.1 Conjunctions of causality conditions

This section defines the *and*-operator, which enables the representation of conjunctions of causality conditions, and presents rules for using this operator.

5.1.1 The *and*-operator

The following elementary causality conditions have been considered so far: the start condition $\surd$, the enabling condition b, the disabling condition $\neg b$ and the synchronization condition $^{=}b$.

Often one needs to represent that multiple elementary conditions involving different actions must all be satisfied for a certain action to happen. Conjunctive causality conditions define the conjunction of two or more enabling, disabling or synchronization conditions involving different actions. The *and*-operator $\wedge : \boldsymbol{C} \times \boldsymbol{C} \rightarrow \boldsymbol{C}$ allows the representation of the conjunction of two elementary or conjunctive causality conditions. The conjunction of multiple elementary or conjunctive conditions can be represented by the repeated application of the *and*-operator. The domain of elementary and conjunctive causality conditions $\boldsymbol{C}$ is completely determined by the elementary causality conditions, the *and*-operator and some composition rules explained further on.

The conjunction of two elementary or conjunctive causality conditions $\gamma 1$ and $\gamma 2$ of result action *a* is defined as follows:

$\gamma_a = \gamma 1 \wedge \gamma 2$, defines that $\gamma 1$ and $\gamma 2$ are both conditions for the occurrence of result action *a*, such that *a* is allowed to occur only when both $\gamma 1$ and $\gamma 2$ are satisfied in an execution.

The *and*-operator is used to model that the occurrence of the result action depends on the occurrences or non-occurrences of multiple other actions in an execution. The interpretation of the *and*-operator is illustrated by the following examples:

- $\gamma_a = b \wedge c$, defines that both actions *b* and *c* must have occurred before action *a* is allowed to occur;
- $\gamma_a = b \wedge \neg c$, defines that action *a* is allowed to occur after action *b* has occurred and when action *c* has not occurred before *a* nor occurs simultaneously with *a*;
- $\gamma_a = b \wedge \neg c \wedge {}^{=}d$ defines that action *a* is allowed to occur simultaneously with *d* after action *b* has occurred and when action *c* has not occurred before *a* nor occurs simultaneously with *a*.

The *and*-operator is idempotent, commutative and associative. Parentheses can be used to structure conjunctions of causality conditions.

Composition rules

We prescribe that each alternative causality condition of some result action should be written as an elementary or conjunctive causality condition. The following rules must be obeyed in order to guarantee the consistent definition (composition) of alternative causality conditions:

- the result action itself can not be used in the definition of alternative causality conditions, since we assume an action may not depend on its own occurrence or non-occurrence. Thus, conditions $\gamma_a = \neg a$ and $\gamma_a = a \wedge \neg c$, for example, are not allowed;
- the conjunction of conditions involving different elementary causality conditions of the same action is not allowed. For example, conditions $\gamma_a = b \wedge \neg b$ and $\gamma_a = b \wedge (\neg c \wedge {}^{=}b)$ can never be satisfied and therefore are not allowed.

Uncertainty attribute

A single uncertainty value is associated with an alternative causality condition. For example, the uncertainty association $\upsilon_a(b \wedge \neg c \wedge {}^{=}d) = \mathit{must}$ defines that result action *a* must occur in executions where condition $b \wedge \neg c \wedge {}^{=}d$ is satisfied. The association of distinct uncertainty values with the constituent (elementary) causality conditions of a composite alternative causality condition makes no sense, and is therefore not allowed, since the result action is only allowed to occur when all constituent elementary causality conditions are satisfied.

5.1.2 Formal definition

The execution semantics of conjunctive causality conditions and their uncertainty associations is defined below. Conjunctive causality conditions are represented identically in the design notation $L_{causality}$ and in the pre-formal design notation $L'_{causality}$ (see Section 4.1.3).

Definition 5.1 The execution semantics of conjunctive causality conditions is defined by the following rules:

$$[\![\langle a, \gamma 1 \wedge \gamma 2\rangle]\!] = [\![\langle a, \gamma 1\rangle]\!] \otimes [\![\langle a, \gamma 2\rangle]\!] \text{ , with } \gamma 1, \gamma 2 \in \boldsymbol{C} \text{ ;}$$

$$[\![\langle a, \gamma 1 \wedge \gamma 2, \mu\rangle]\!] = Comp(\, Comp([\![\langle a, \gamma 1, \mu\rangle]\!]\,) \otimes Comp([\![\langle a, \gamma 2, \mu\rangle]\!]\,)\,), \text{ with } \mu \in \boldsymbol{U}. \qquad \blacksquare$$

The set of partial executions allowed by condition $\gamma_a = \gamma 1 \wedge \gamma 2$ is equal to the cross-conjunction of the sets of partial executions allowed by $\gamma_a = \gamma 1$ and $\gamma_a = \gamma 2$. The cross-conjunction operation guarantees that $[\![\langle a, \gamma 1 \wedge \gamma 2\rangle]\!]$ represents the set of all possible partial executions that satisfy the conjunction of the constraints represented by an execution from $[\![\langle a, \gamma 1\rangle]\!]$ and an execution from $[\![\langle a, \gamma 2\rangle]\!]$.

For example, the execution semantics of $\gamma_a = b \wedge \neg c$ is obtained as follows:

$$\begin{aligned} [\![\langle a, b \wedge \neg c\rangle]\!] &= [\![\langle a, b\rangle]\!] \otimes [\![\langle a, \neg c\rangle]\!] \\ &= \{\boxed{b \rightarrow a}, \boxed{b \quad \bar{a}}, \boxed{\bar{b} \quad \bar{a}}\} \otimes \{\boxed{a \rightarrow c}, \boxed{\bar{a} \quad c}, \boxed{a \quad \bar{c}}, \boxed{\bar{a} \quad \bar{c}}\} \end{aligned}$$

$$= \{ \boxed{\begin{matrix} & a & \\ b & & c \end{matrix}}, \boxed{\begin{matrix} & a & \\ b & & \bar{c} \end{matrix}}, \boxed{\begin{matrix} & \bar{a} & \\ b & \text{- - -} & c \end{matrix}}, \boxed{\begin{matrix} & \bar{a} & \\ b & & \bar{c} \end{matrix}}, \boxed{\begin{matrix} & \bar{a} & \\ \bar{b} & & c \end{matrix}}, \boxed{\begin{matrix} & \bar{a} & \\ \bar{b} & & \bar{c} \end{matrix}} \}.$$

A dotted line between two action occurrences b and c in the graphical notation of a partial execution represents that this execution leaves undefined whether b and c are independent or related (see Section 3.6.4). The partial execution therefore defines that b and c either occur independently or are related in one of the following ways: $b < c$, $c < b$, or $b = c$.

The set of partial executions disallowed by uncertainty association $\langle a, \gamma 1 \wedge \gamma 2, \mu\rangle$ is equal to the cross-conjunction of the partial execution sets disallowed by uncertainty associations $\langle a, \gamma 1, \mu\rangle$ and $\langle a, \gamma 2, \mu\rangle$. The cross-conjunction operation guarantees that $Comp([\![\langle a, \gamma 1 \wedge \gamma 2, \mu\rangle]\!]\,)$ represents the set of all possible partial executions disallowed by a partial execution from $Comp([\![\langle a, \gamma 1, \mu\rangle]\!]\,)$ and by a partial execution from $Comp([\![\langle a, \gamma 2, \mu\rangle]\!]\,)$. The *Comp* operator is needed to derive the sets of disallowed partial executions from $[\![\langle a, \gamma 1, \mu\rangle]\!]$ and $[\![\langle a, \gamma 2, \mu\rangle]\!]$, and subsequently to derive the set of allowed partial executions from the cross-conjunction of both sets of disallowed executions.

For example, the execution semantics of $\upsilon_a(b \wedge \neg c) = must$ is obtained as follows:

$$\begin{aligned} [\![\langle a, b \wedge \neg c, must\rangle]\!] &= Comp(Comp([\![\langle a, b, must\rangle]\!]\,) \otimes Comp([\![\langle a, \neg c, must\rangle]\!]\,)) \\ &= Comp(\{\boxed{b \quad \bar{a}}\} \otimes \{\boxed{\bar{a} \quad \bar{c}}\}) \\ &= Comp(\{\boxed{b \quad \bar{a} \quad \bar{c}}\}) \\ &= EE\text{-}Free(\{a, b, c\}) - \{\boxed{b \quad \bar{a} \quad \bar{c}}\}) \,. \end{aligned}$$

In case $\mu = may$, the execution semantics of $\langle a, \gamma 1, \mu\rangle$, $\langle a, \gamma 2, \mu\rangle$ and $\langle a, \gamma 1 \wedge \gamma 2, \mu\rangle$ is equal to $EE\text{-}Free(Ac(\gamma 1))$, $EE\text{-}Free(Ac(\gamma 2))$ and $EE\text{-}Free(Ac(\gamma 1 \wedge \gamma 2))$, respectively, where function

$Ac : \boldsymbol{C} \rightarrow \wp(\boldsymbol{A})$ renders the set of actions in the definition of an alternative causality condition. This is consistent with Definition 5.1, since:

$$\begin{aligned}
[\![\langle a, \gamma 1 \wedge \gamma 2, \mu\rangle]\!] &= Comp(\, Comp([\![\langle a, \gamma 1, \mu\rangle]\!]\,) \otimes Comp([\![\langle a, \gamma 2, \mu\rangle]\!]\,)\,), \text{ with } \mu = may \\
&= Comp(\, Comp(EE\text{-}Free(Ac(\gamma 1))) \otimes Comp(EE\text{-}Free(Ac(\gamma 2)))\,) \\
&= Comp(\, \varnothing \otimes \varnothing\,) = Comp(\, \varnothing\,) \\
&= EE\text{-}Free(Ac(\gamma 1 \wedge \gamma 2)).
\end{aligned}$$

Property 5.2 The *and*-operator $\wedge : \boldsymbol{C} \times \boldsymbol{C} \rightarrow \boldsymbol{C}$ is idempotent, commutative and associative.

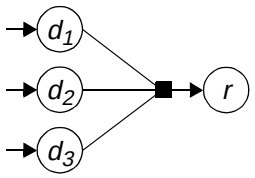
Proof: These properties are inherited from the cross-conjunction operator $\otimes$. ■

5.1.3 Examples

The use of the *and*-operator is illustrated by means of some example behaviours.

Example: reassembly of a data unit

Figure 5.1 depicts an example of the conjunction of enabling conditions. The conjunction operator is graphically represented by the symbol ■. Action *r* models the reassembly of three data units. Actions d_1, d_2 and d_3 model the arrival of these data units. The conjunction models that the reassembly activity can only be finished after all three data units have been received.

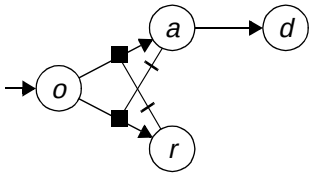


$$\begin{aligned}
B = \{\, d_1 \wedge d_2 \wedge d_3 &\rightarrow r, \\
\surd &\rightarrow d_1, \\
\surd &\rightarrow d_2, \\
\surd &\rightarrow d_3 \,\}
\end{aligned}$$

Figure 5.1: Conjunction of enabling conditions

Example: product ordering

Figure 5.2 depicts an example of the conjunction of enabling and disabling conditions. Action *o* models the ordering of a product. After action *o* has occurred, a choice is made between the occurrence of action *a* or the occurrence of action *r*, which model the acceptance and the refusal of the order, respectively. The occurrence of *a* excludes the occurrence of *r*, and vice versa. After action *a* has occurred, action *d*, which models the delivery of the ordered product, is allowed to occur.

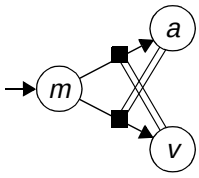


$$\begin{aligned}
B = \{ \quad \surd &\rightarrow o, \\
o \wedge \neg r &\rightarrow a, \\
o \wedge \neg a &\rightarrow r, \\
a &\rightarrow d \,\}
\end{aligned}$$

Figure 5.2: Conjunction of enabling and disabling conditions

Example: lip-synchronization

Figure 5.3 depicts an example of the conjunction of enabling and synchronization conditions. Action *m* models the arrival of a multi-media data unit, containing a combined audio video sample. After the data unit has been received, the audio and video sample must be played simultaneously, which is modelled by the synchronization of actions *a* and *v*, respectively.



$$B = \{ \quad \surd \rightarrow m,\; m \wedge {}^{=}v \rightarrow a,\; m \wedge {}^{=}a \rightarrow v \}$$

Figure 5.3: Conjunction of enabling and synchronization conditions

5.2 Disjunctions of causality conditions

This section defines the *or*-operator, which enables the representation of disjunctions of alternative causality conditions, and presents rules for using this operator.

5.2.1 The *or*-operator

The causality condition of an action is defined as a single alternative causality condition or as a disjunction of multiple alternative causality conditions. Alternative causality conditions are defined using the *and*-operator as discussed in Section 5.1.

Often one needs to represent that multiple alternative causality conditions, of which at least one must be satisfied, may cause a certain action to happen. Disjunctive causality conditions define the disjunction of two or more alternative causality conditions. The *or*-operator $\vee : \boldsymbol{CC} \times \boldsymbol{CC} \rightarrow \boldsymbol{CC}$ allows the representation of the disjunction of two alternative or disjunctive causality conditions. The domain of (disjunctive) causality conditions $\boldsymbol{CC}$ is completely determined by the domain of elementary and conjunctive causality conditions $\boldsymbol{C}$ and the *or*-operator.

The disjunction of two causality conditions $\Gamma 1$ and $\Gamma 2$ of result action *a* is defined as follows:

> $\Gamma_a = \Gamma 1 \vee \Gamma 2$, defines that the alternative causality conditions in $\Gamma 1$ and $\Gamma 2$ are alternative conditions for the occurrence of the result action *a*, such that *a* is allowed to occur when at least one of these conditions is satisfied in an execution.

The *or*-operator is used to model that the occurrence of the result action is made possible by the satisfaction of one or more of the alternative conditions. The precise interpretation of the adjective ‘alternative’ is explained and formally defined below.

Alternative causes of action occurrences

In general, multiple alternative causality conditions that allow the occurrence of the result action can be satisfied simultaneously. For example, consider causality condition $\Gamma_a = b \vee c$ and the situation in which both *b* and *c* have occurred before *a* has occurred. In this case, action *a* is allowed to occur due to the satisfaction of condition *b* or due to the satisfaction of condition *c*.

We define that the occurrence of a result action is caused by (or depends on) only one of its alternative causality conditions. This definition is referred to as the *exclusive-or interpretation*. In case multiple alternative causality conditions are satisfied in an execution, a decision has to be made at execution time which alternative causality condition effectively will cause the occurrence of the result action, assuming that the result action occurs. This decision may be (partly) specified, or absolutely not. In the latter case, the decision is left to the implementer or is dictated by mechanisms of the implementation.

The alternative causality condition that causes the occurrence of the result action in an execution is called the *resulting causality condition*. In this execution, the result action depends on the actions that determine the satisfaction of the resulting causality condition, and is independent of other actions of the behaviour.

For example, consider behaviour $B = \{b \vee c \rightarrow a, \surd_! \rightarrow b, \surd_! \rightarrow c\}$. Figure 5.4(i) depicts this behaviour. The occurrence of *a* either depends on the occurrence of *b* and is then independent of *c*, or the occurrence of *a* depends on the occurrence of *c* and is then independent of *b*. Figure 5.4(ii) depicts the three partial executions that are allowed by $\Gamma_a = b \vee c$, when assuming that *b* and *c* occur. Executions in which the occurrence of *a* depends on the occurrences of both *b* and *c* are disallowed. These disallowed executions are represented by the partial execution in Figure 5.4(iii).

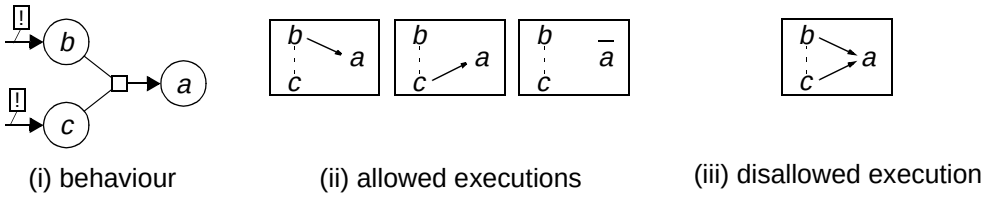


Figure 5.4: Causality condition $\Gamma_a = b \vee c$

Direct and indirect independencies

Figure 5.5 shows that the result action may depend indirectly on actions that are not defined in the resulting causality condition. In this example, action *a* indirectly depends on the occurrence of action *d*, since both the satisfaction of condition *b* and the satisfaction of condition *c* depend on the occurrence of *d*. The indirect dependency is a consequence of the transitivity of enabling relations.

In general, the set of actions on which some action *a* depends in an execution is defined by the following rules, where we assume γ_a and γ_b are the resulting causality conditions of actions *a* and *b*, respectively:

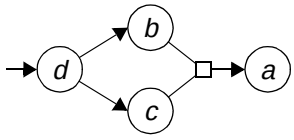


Figure 5.5: Example of indirect dependencies

1. action a depends on the actions in γ_a;
2. if b is an enabling action in γ_a then action a depends on the actions in γ_b. Enabling and synchronization actions in γ_b are called *indirect enabling actions* of a in case they are not defined as enabling actions in γ_a as well, since they indirectly enable a via b;
3. if b is a synchronization action in γ_a then action a depends on the actions in γ_b. Enabling actions in γ_b are indirect enabling actions of a in case they are not defined as enabling actions in γ_a as well. Synchronization actions in γ_b are called *indirect synchronization actions* of a in case they are not defined as synchronization actions in γ_a as well, since they synchronize indirectly with a via b;
4. if b is a disabling action in γ_a then action a only depends on the synchronization actions in γ_b;
5. if b is an indirect enabling action of a then rule 2 applies recursively;
6. if b is an indirect synchronization action of a then rule 3 applies recursively.

Figure 5.6(i) illustrates rules 2 and 5: action a depends on actions b, c and d, where c and d are indirect enabling actions of a. Figure 5.6(ii) illustrates rule 4: action a depends on actions b and c, since b occurs iff c occurs. Figures 5.6(iii) and (iv) illustrate that in rule 4 action a does not depend on enabling actions and disabling actions in γ_b, respectively: in both examples, condition $\gamma_a = \neg b$ can be satisfied either when c occurs or when c does not occur, which implies that action c does not determine the satisfaction of γ_a.

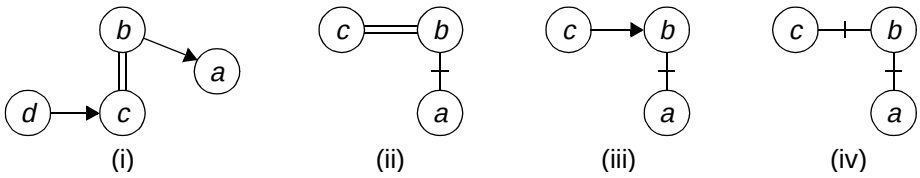


Figure 5.6: Examples of indirect dependencies (2)

Given that Ac_{Rel} denotes the set of actions on which action a depends in an execution, as obtained by the rules above, the distinction between direct and indirect dependencies is defined as follows:

- action a depends *directly* on action b in this execution, when b is defined in the resulting causality condition of a;
- action a depends *indirectly* on action b in this execution, when b is defined in Ac_{Rel} and action a does not depend directly on b.

Based on this distinction, an alternative causality condition γ_a defines that:

- action a depends directly on the actions defined in γ_a; and

- action *a* is independent of all other actions in the behaviour except for the actions on which *a* depends indirectly via γ_a in an execution,

when assuming γ_a is the resulting causality condition of result action *a*.

The set of actions on which action *a* depends indirectly may differ in different executions, since the actions on which *a* depends may have multiple alternative causality conditions. Figure 5.7 illustrates this situation. When assuming that condition *b* is the resulting causality condition of *a*, either *a* depends indirectly on *d* and is independent of *e* (and *c*), or *a* depends indirectly on *e* and is independent of *d* (and *c*). Alternatively, when condition *c* is the resulting causality condition of *a*, action *a* is independent of *b*, *d* and *e*.

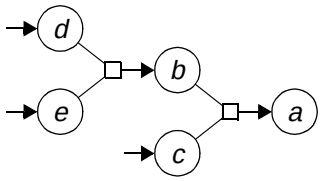


Figure 5.7: Example of indirect dependencies (3)

In general, different resulting causality conditions render different executions. The executions allowed by a disjunction of alternative causality conditions is equal to the union of the executions allowed by the individual alternative causality conditions, since only one of them can be the resulting causality condition in an execution.

Motivation

The reason for choosing the exclusive-or interpretation for the *or*-operator is that this interpretation is more expressive than an inclusive-or interpretation, in which a result action could be caused by multiple alternative causality conditions in an execution. The inclusive-or interpretation precludes the possibility to prescribe that an action may only be caused by individual alternative causality conditions. For example, in that case, the execution of Figure 5.4(iii) would be allowed. However, this alternative interpretation of the *or*-operator implies that we would not be able to define a behaviour in which action *a* is either dynamically related to *b* or is dynamically related to *c*.

A concrete example that corresponds to the abstract condition $\Gamma_a = b \vee c$ is when actions *b* and *c* represent independent requests for the reservation of some resource, and action *a* represents the acknowledgement of one of these requests. In this case, the cause for the acknowledgement is only one of the requests, since the resource can only be granted to one of them. The possible occurrence of another request is irrelevant to perform the acknowledgement once one request has already been selected.

Disjunction of multiple causality conditions

A consequence of the exclusive-or interpretation is that one has to define all possible alternative causes for the occurrence of the result action explicitly. For example, in case one

wants to allow the partial execution of Figure 5.4(iii), the causality condition of action a should be defined as $\Gamma_a = b \vee c \vee (b \wedge c)$.

The disjunction of multiple alternative conditions can be defined by repeated application of the *or*-operator. The *or*-operator is idempotent, commutative and associative. Parentheses can be used to structure disjunctions of causality conditions.

For example, causality condition $\Gamma_a = b \vee \neg c \vee (b \wedge \neg c)$ defines that action a is allowed to occur either (i) after action b has occurred, or (ii) when action c has not occurred before a nor occurs simultaneously with a, or (iii) after action b has occurred and action c has not occurred before a nor occurs simultaneously with a. In case (i), a is independent of c, and in case (ii), a is independent of b, assuming that b and c are independent.

Uncertainty attribute

The uncertainty attribute associates an uncertainty value with each alternative causality condition of a result action. Therefore, the uncertainty attribute of a result action consists of a set of uncertainty associations, one for each alternative causality condition. For example, given $\Gamma_a = b \vee \neg c$ the uncertainty attribute υ_a consists of two uncertainty associations, i.e., $\upsilon_a = \{\upsilon_a(b), \upsilon_a(\neg c)\}$.

5.2.2 Formal definition

Inadvertently, one might define the execution semantics of the disjunctive causality condition of some result action a in the following way:

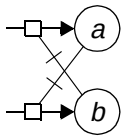
$$[\![\langle a, \Gamma 1 \cup \Gamma 2\rangle]\!] = [\![\langle a, \Gamma 1\rangle]\!] \cup [\![\langle a, \Gamma 2\rangle]\!] \;;$$

$$[\![\langle a, \{\gamma_a\}\rangle]\!] = [\![\langle a, \gamma_a\rangle]\!] \;.$$

The execution semantics of $\Gamma_a = \Gamma 1 \cup \Gamma 2$ would be equal to the union of the sets of partial executions allowed by causality conditions $\Gamma 1$ and $\Gamma 2$. Consequently, the executions allowed by the causality relation of action a would consist of the union of the partial executions allowed by the alternative causality conditions of a.

The above definition suffices for many behaviours, but not for all. Figure 5.8 illustrates a specific behaviour B consisting of two actions a and b for which this definition renders incorrect executions. Scheme 1 represents the calculation of the execution semantics of B using the rules above. According to this scheme $\boxed{a \rightarrow b}$ and $\boxed{b \rightarrow a}$ are possible executions of B. Both executions are disallowed by behaviour B, however, since actions a and b are defined to be either related via the disabling conditions $\neg b$ and $\neg a$, or are defined to be independent.

In case action a occurs due to $\surd$, the occurrence of a is independent of action b, which generally implies that the possible occurrence of b is either (i) independent of a or (ii) depends on the occurrence of a (via enabling condition a); behaviour B only contains option (i). The execution semantics of $\langle a, \{\surd\}\rangle$ should therefore forbid executions $\boxed{b \rightarrow a}$ and $\boxed{a = b}$, since



$B = \{ \surd \vee \neg b \rightarrow a,$
$\surd \vee \neg a \rightarrow b \}$

scheme 1:

$[\![\langle a, \surd\rangle]\!] \cup [\![\langle a, \neg b\rangle]\!]$: $\boxed{a}\ \boxed{\bar{a}}\ \boxed{a \rightarrow b}\ \boxed{a\ \bar{b}}\ \boxed{\bar{a}\ b}\ \boxed{\bar{a}\ \bar{b}}$

$[\![\langle b, \surd\rangle]\!]^{\chi}_{\chi} \cup [\![\langle b, \neg a\rangle]\!]^{\chi}_{\chi}$: $\boxed{b}\ \boxed{\bar{b}}\ \boxed{b \rightarrow a}\ \boxed{a\ \bar{b}}\ \boxed{\bar{a}\ b}\ \boxed{\bar{a}\ \bar{b}}$ $\otimes$

$\boxed{a\ b}\ \boxed{a \rightarrow b}\ \boxed{b \rightarrow a}\ \boxed{a\ \bar{b}}\ \boxed{\bar{a}\ b}\ \boxed{\bar{a}\ \bar{b}}$

scheme 2:

$[\![\langle a, \surd\rangle]\!] \otimes E\text{-}Ind(a, \{b\}) \cup [\![\langle a, \neg b\rangle]\!] \otimes E\text{-}Ind(a, \varnothing)$: $\boxed{a\ b}\ \boxed{a \rightarrow b}\ \boxed{a\ \bar{b}}\ \boxed{\bar{a}\ b}\ \boxed{\bar{a}\ \bar{b}}$

$[\![\langle b, \surd\rangle]\!]^{\chi}_{\chi} \otimes E\text{-}Ind(b, \{a\}) \cup [\![\langle b, \neg a\rangle]\!]^{\chi}_{\chi} \otimes E\text{-}Ind(b, \varnothing)$: $\boxed{a\ b}\ \boxed{b \rightarrow a}\ \boxed{a\ \bar{b}}\ \boxed{\bar{a}\ b}\ \boxed{\bar{a}\ \bar{b}}$ $\otimes$

$\boxed{a\ b}\ \boxed{a\ \bar{b}}\ \boxed{\bar{a}\ b}\ \boxed{\bar{a}\ \bar{b}}$

Figure 5.8: Application of naive definition

these executions are only possible when b occurs due to disabling condition $\neg a$ or synchronization condition $^{=}a$, respectively.

This extra constraint can be imposed by $E\text{-}Ind(a, \{b\})$, where function $E\text{-}Ind : \boldsymbol{A} \times \wp(\boldsymbol{A}) \rightarrow \wp(\boldsymbol{XX})$ defines the allowed partial executions for an action a when this action is independent of a set of other actions Ac:

$E\text{-}Ind(a, Ac) = \otimes\{E\text{-}Ind(a, \{b\}) \mid b \in Ac, b \neq a\}$;

$E\text{-}Ind(a, \{b\}) = \{\boxed{a\ b}, \boxed{a \rightarrow b}, \boxed{a\ \bar{b}}, \boxed{\bar{a}\ b}, \boxed{\bar{a}\ \bar{b}}\}$;

$E\text{-}Ind(a, \varnothing) = \{\boxed{a}, \boxed{\bar{a}}\}$.

This constraint should be added to $[\![\langle a, \surd\rangle]\!]$, such that the execution semantics of $\langle a, \{\gamma_a\}\rangle$ is defined as: $[\![\langle a, \{\surd\}\rangle]\!] = [\![\langle a, \surd\rangle]\!] \otimes E\text{-}Ind(a, \{b\})$. Analogously, constraint $E\text{-}Ind(b, \{a\})$ should be added to $[\![\langle b, \surd\rangle]\!]$. Scheme 2 of Figure 5.8 incorporates these additional constraints. This scheme renders the same executions as scheme 1, except for executions $\boxed{a \rightarrow b}$ and $\boxed{b \rightarrow a}$, which are made impossible by the combination of both constraints. For completeness, constraints $E\text{-}Ind(a, \varnothing)$ and $E\text{-}Ind(b, \varnothing)$ are added to $[\![\langle a, \neg b\rangle]\!]$ and $[\![\langle b, \neg a\rangle]\!]$, but they do not restrict the executions allowed by $[\![\langle a, \neg b\rangle]\!]$ and $[\![\langle b, \neg a\rangle]\!]$, respectively.

We conclude that, in general, when some action a occurs independently of action b, executions $\boxed{b \rightarrow a}$ and $\boxed{a = b}$ should be forbidden, since a dynamic two-sided relation may be defined between both actions either directly, such as in the above example, or indirectly.

Correct execution semantics

The execution semantics of some resulting causality condition γ_a should represent the following characteristics in terms of executions:

1. the direct dependency of action a on the actions defined in γ_a;

2. the independence of action a of all other actions of the behaviour, called B, except for the actions on which a indirectly depends via γ_a in an execution.

The execution semantics of the direct dependency of result action a on the actions in γ_a is defined by $[\![\langle a, \gamma_a\rangle]\!]$. Possible indirect dependencies of action a on actions that determine the satisfaction of γ_a, but are not defined in γ_a, are not considered explicitly by the execution semantics of γ_a. These indirect dependencies are defined implicitly by the composition of the execution semantics of γ_a and the executions semantics of the causality relations of the actions on which a depends indirectly via γ_a, due to the transitivity of ordering relation < and synchronization relation =, and the compositionality of < and =.

The execution semantics of the independence of result action a of the actions in B on which a depends neither directly nor indirectly via γ_a in an execution, is defined by $E\text{-}Ind(a, Ac_{Ind})$, where $Ac_{Ind} = Ac(B) - Ac_{Rel}$ and Ac_{Rel} represents the set of actions on which a depends directly or indirectly in this execution. In general, there may be many alternative sets Ac_{Rel}, which are defined by function Ac^+.

We introduce some functions in order to facilitate the definition of Ac^+:

- $Ac : \boldsymbol{C} \rightarrow \wp(\boldsymbol{A})$ defines the actions in some alternative causality condition, with $b \in \boldsymbol{A}$ and $\gamma 1, \gamma 2 \in \boldsymbol{C}$:

  $Ac(b) = Ac(\neg b) = Ac({}^{=}b) = \{b\}$;

  $Ac(\gamma 1 \wedge \gamma 2) = Ac(\gamma 1) \cup Ac(\gamma 2)$;

- $C : \boldsymbol{C} \rightarrow \wp(\boldsymbol{C})$ defines the elementary conditions in some alternative causality condition, with $b \in \boldsymbol{A}$ and $\gamma 1, \gamma 2 \in \boldsymbol{C}$:

  $C(b) = \{b\}$; $C(\neg b) = \{\neg b\}$; $C({}^{=}b) = \{{}^{=}b\}$;

  $C(\gamma 1 \wedge \gamma 2) = C(\gamma 1) \cup C(\gamma 2)$.

Function $Ac^+ : \boldsymbol{C} \times \boldsymbol{B} \rightarrow \wp(\wp(\boldsymbol{A}))$ defines for some alternative causality condition the alternative sets of actions on which the result action depends directly or indirectly in an execution. Variables Ac_{IndEna} represents the set of indirect enabling actions, and variable Ac_{IndSyn} represents the set of indirect synchronization actions.

We want to determine $Ac^+(\gamma_a, B)$, where behaviour B consists of action a and actions $a1$, .., an, such that $Ac(B) = \{a, a1, .., an\}$. We assume b, c and d range over $\{a1, .., an\}$.

For each combination of resulting causality conditions $\{\gamma_{a1}, .., \gamma_{an}\}$, such that $\gamma_{a1} \in \Gamma_{a1}$, .., $\gamma_{an} \in \Gamma_{an}$, set Ac_{Rel} can be determined recursively by the following algorithm:

1. $Ac_{Rel} = Ac(\gamma_a)$; $Ac_{IndEna} = \emptyset$; $Ac_{IndSyn} = \emptyset$;
2. if $b \in C(\gamma_a)$ then $\quad Ac_{Rel} = Ac_{Rel} \cup Ac(\gamma_b)$;
   $Ac_{IndEna} = Ac_{IndEna} \cup \{c \mid c \in C(\gamma_b) \text{ or } {}^{=}c \in C(\gamma_b)\}$;
3. if ${}^{=}b \in C(\gamma_a)$ then $\quad Ac_{Rel} = Ac_{Rel} \cup Ac(\gamma_b)$;
   $Ac_{IndEna} = Ac_{IndEna} \cup \{c \mid c \in C(\gamma_b)\}$;
   $Ac_{IndSyn} = Ac_{IndSyn} \cup \{c \mid {}^{=}c \in C(\gamma_b)\}$;
4. if $\neg b \in C(\gamma_a)$ then $\quad Ac_{Rel} = Ac_{Rel} \cup \{c \mid {}^{=}c \in C(\gamma_b)\}$;

5. for each $d \in Ac_{IndEna}$:
   if $b \in C(\gamma_d)$ then $Ac_{Rel} = Ac_{Rel} \cup Ac(\gamma_d)$;
   $Ac_{IndEna} = Ac_{IndEna} \cup \{c \mid c \in C(\gamma_d) \text{ or } {}^{=}c \in C(\gamma_d)\}$;
   (New actions may be added to Ac_{IndEna} in steps 5 and 6.)
6. for each $d \in Ac_{IndSyn}$:
   if ${}^{=}b \in C(\gamma_d)$ then $Ac_{Rel} = Ac_{Rel} \cup Ac(\gamma_d)$;
   $Ac_{IndEna} = Ac_{IndEna} \cup \{c \mid c \in C(\gamma_d)\}$;
   $Ac_{IndSyn} = Ac_{IndSyn} \cup \{c \mid {}^{=}c \in C(\gamma_d)\}$;
   (New actions may be added to Ac_{IndSyn} in this step.)

$Ac^{+}(\gamma_a, B)$ is equal to the set of all sets Ac_{Rel} that can be obtained using this algorithm. (Steps 1 to 6 of this algorithm implement rules 1 to 6 presented in Section 5.2.1.)

Based on the above, the incorrect definition of the execution semantics given in the beginning of this section can be replaced by the following definition.

Definition 5.3 The execution semantics of the disjunctive causality condition of a result action a is defined by the following rules:

$$[\![\langle a, \Gamma_1 \cup \Gamma_2\rangle]\!]\,(B) = [\![\langle a, \Gamma_1\rangle]\!]\,(B) \cup [\![\langle a, \Gamma_2\rangle]\!]\,(B)\ ;$$

$$[\![\langle a, \{\gamma_a\}\rangle]\!]\,(B) = \cup\{[\![\langle a, \gamma_a\rangle]\!] \otimes E\text{-}Ind(a, Ac_{Ind}) \mid Ac_{Ind} = Ac(B) - Ac_{Rel}, Ac_{Rel} \in Ac^{+}(\gamma_a, B)\};$$

where B represents the behaviour (context) in which action a is defined. ■

Optimization

Definition 5.3 can be made more efficient by replacing the definition of Ac_{Ind} by:

$Ac_{Ind}' = Ac(a, B) - Ac_{Rel}$,

where $Ac(a, B)$ consists of the following actions from $Ac(B)$:

- each action b for which an execution exists in which a depends directly on b; and
- each action b for which an execution exists in which b depends directly on a.

Consequently, this definition removes from the sets in Ac_{Ind} the actions $c \in Ac(B) - Ac(a, B)$, such that for all executions:

- action a is independent of c or a depends indirectly on c; and
- action c is independent of a or c depends indirectly on a.

The possible independence of action a on the actions in $Ac(B) - Ac(a, B)$ can be left undefined because the cross-conjunction of the execution semantics of all causality relations of B leaves the relation (or independence) between these actions undefined, unless indirect dependencies between these actions are defined. This implies that by leaving the relation between actions undefined in partial executions we represent the independence of actions. However, undefined relations are only interpreted as independence when the execution semantics of all causality relations of a behaviour have been considered.

Since $Ac(a, B) \subseteq Ac(B)$, the usage of $Ac(a, B)$ instead of $Ac(B)$ in Definition 5.3 potentially reduces the number of executions involved in the calculation of the execution semantics of a causality condition. Action set $Ac(a, B)$ is defined as follows:

$Ac(a, B) = Ac(\Gamma_a) \cup Ac^{\leftarrow}(a, B)$, where

- $Ac : \boldsymbol{CC} \rightarrow \wp(\boldsymbol{A})$ defines the actions in some causality condition, with $\Gamma \in \boldsymbol{CC}$:

  $Ac(\Gamma) = \{Ac(\gamma) \mid \gamma \in \Gamma\}$;

- $Ac^{\leftarrow} : \boldsymbol{A} \times \boldsymbol{B} \rightarrow \wp(\boldsymbol{A})$ renders the actions of a behaviour which may depend directly on some action a in an execution:

  $Ac^{\leftarrow}(a, B) = \{Ac(\rho) \mid \rho \in B, a \in Ac(\Gamma(\rho))\}$.

Example

Figure 5.9 depicts the execution semantics of the causality relations of a behaviour B consisting of four actions a, b, c and d. The execution semantics of these causality relations is equal to the execution semantics of the causality conditions of actions a, b, c and d, due to the default may-interpretation. Undefined relations are graphically represented by dotted lines, such as in Section 5.1.2. We use the optimization of Definition 5.3 as described above (and assume $\otimes$ has precedence over $\cup$) to infer the execution semantics of this behaviour:

$$[\![b \vee c \rightarrow a]\!] (B) = [\![\langle a, b\rangle]\!] \otimes E\text{-}Ind(a, \{c\}) \cup [\![\langle a, c\rangle]\!] \otimes E\text{-}Ind(a, \{b\}) ;$$

$$[\![d \rightarrow b]\!] (B) = [\![\langle b, d\rangle]\!] \otimes E\text{-}Ind(b, \{a\}) ;$$

$$[\![\surd \rightarrow c]\!] (B) = [\![\langle c, \surd\rangle]\!] \otimes E\text{-}Ind(c, \{a\}) ;$$

$$[\![\surd \rightarrow d]\!] (B) = [\![\langle d, \surd\rangle]\!] \otimes E\text{-}Ind(d, \{b\}) .$$

The execution semantics of B is obtained by the cross-conjunction of the execution semantics of the causality relations of a, b, c and d, i.e., $[\![B]\!] = [\![b \vee c \rightarrow a]\!] (B) \otimes [\![d \rightarrow b]\!] (B) \otimes [\![\surd \rightarrow c]\!] (B) \otimes [\![\surd \rightarrow d]\!] (B)$, which renders 11 different executions as depicted in Figure 5.9. The first execution defines the independence of b and c and of d and c, since the relation between b and c and the relation between d and c are left undefined. The second execution defines the independence of d and a, which is caused by the independence of b and a. The independence of b and a can not be composed with the ordering relation between d and b, and therefore the relation between d and a remains undefined. Instead, in the first execution, the initially undefined relation between a and d is removed during the calculation of the cross-conjunction, due to the composition of the ordering relations $d < b$ and $b < a$, which renders the indirect relation $d < a$.

Uncertainty attribute

The uncertainty attribute υ_a of result action a disallows the executions that are disallowed by one or more of its uncertainty associations φ_a. The set of executions disallowed by υ_a is defined as:

$$EE_{dis} = \cup\{Comp([\![\varphi_a]\!]) \mid \varphi_a \in \upsilon_a\} .$$

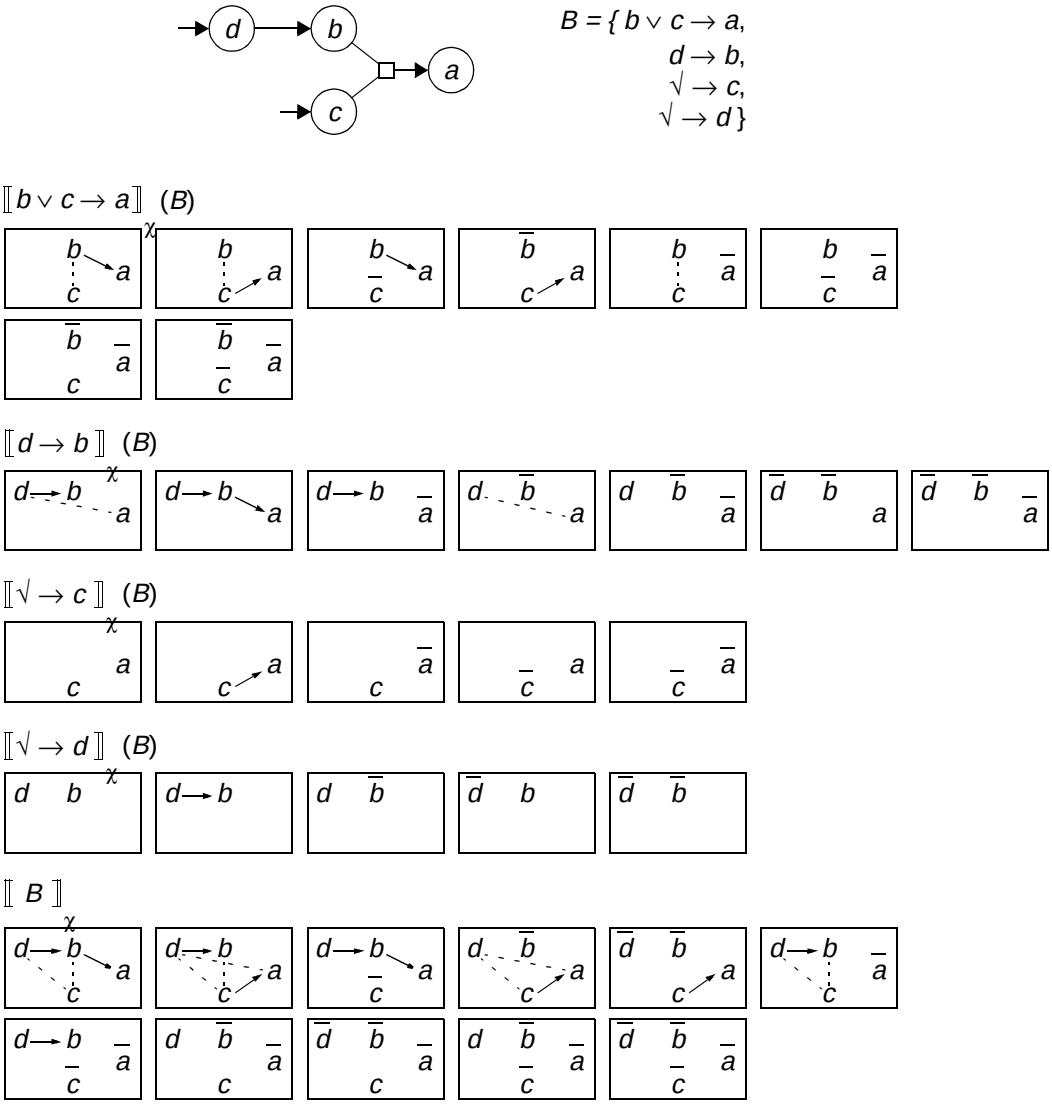


Figure 5.9: Example of execution semantics

Consequently, the set of executions allowed by υ_a is equal to the complement of EE_{dis}. In order to obtain the complement, all executions in EE_{dis} must have the same action domain. The maximal action domain of an execution in EE_{dis} is $Ac(\Gamma_a)$, which consists of all actions defined in the causality condition of action a. Therefore, each execution $e\chi$ in EE_{dis} is extended with the constraint $EE\text{-}Free(Ac_{dif})$, with $Ac_{dif} = Ac(\Gamma_a) - Ac(e\chi)$, which defines that the execution of the actions in Ac_{dif} is unconstrained. This extension is based on the property that an execution $e\chi_1 \in EE_{dis}$ disallows any execution $e\chi_2$, with $e\chi_1 \sqsubseteq e\chi_2$ and $Ac(e\chi_1) \subseteq Ac(e\chi_2)$. For example, if execution $\boxed{\bar{a}}$ is disallowed then executions $\boxed{\bar{a}\ b}$ and $\boxed{\bar{a}\ \bar{b}}$ are disallowed.

Definition 5.4 The execution semantics of the uncertainty attribute of result action a is defined as:

$$\llbracket \upsilon_a \rrbracket (\Gamma_a) = Comp(\cup\{EE_{dis}(\varphi, \Gamma_a) \mid \varphi \in \upsilon\}),$$

$$EE_{dis}(\varphi, \Gamma_a) = Comp(\llbracket \varphi \rrbracket) \otimes EE\text{-}Free(Ac_{dif}) \quad \text{if } Ac_{dif} \neq \varnothing;$$

$$= \; Comp(\llbracket \varphi \rrbracket) \qquad \text{if } Ac_{dif} = \varnothing,$$

where $Ac_{dif} = Ac(\Gamma_a) - Ac(\llbracket \varphi \rrbracket)$, which represents the actions defined in Γ_a but not contained in the action domain of $\llbracket \varphi \rrbracket$. ■

For example, given the causality relation $(b \wedge \neg c)_! \vee (b \wedge {}^{=}d)_! \vee e_? \rightarrow a$, the execution semantics of the uncertainty attribute of action a is defined as follows:

$$\begin{aligned}
\llbracket \upsilon_a \rrbracket (\Gamma_a) \;=\; & Comp(\; Comp(\llbracket \langle a, b \wedge \neg c, must \rangle \rrbracket) \otimes EE\text{-}Free(\{d, e\}) \\
& \cup Comp(\llbracket \langle a, b \wedge {}^{=}d, must \rangle \rrbracket) \otimes EE\text{-}Free(\{c, e\}) \\
& \cup Comp(\llbracket \langle a, e, may \rangle \rrbracket) \otimes EE\text{-}Free(\{b, c, d\}) \;\;) \\
= \; & Comp(\; Comp(\; Comp(Comp(\llbracket \langle a, b, must \rangle \rrbracket) \otimes Comp(\llbracket \langle a, \neg c, must \rangle \rrbracket))) \\
& \qquad \otimes EE\text{-}Free(\{d, e\}) \\
& \cup Comp(\; Comp(Comp(\llbracket \langle a, b, must \rangle \rrbracket) \otimes Comp(\llbracket \langle a, {}^{=}d, must \rangle \rrbracket))) \\
& \qquad \otimes EE\text{-}Free(\{c, e\}) \\
& \cup \varnothing \otimes EE\text{-}Free(\{b, c, d\}) \;\;) \\
= \; & Comp(\; \{ \boxed{b \;\; \overline{a} \;\; c} \} \otimes \{ \boxed{d - - e}, \boxed{d \;\; \overline{e}}, \boxed{\overline{d} \;\; e}, \boxed{\overline{d} \;\; \overline{e}} \} \\
& \cup \{ \boxed{b \;\; \overline{a} \;\; d} \} \otimes \{ \boxed{c - - e}, \boxed{c \;\; \overline{e}}, \boxed{\overline{c} \;\; e}, \boxed{\overline{c} \;\; \overline{e}} \} \;).
\end{aligned}$$

Execution semantics completion

The cross-conjunction of the execution semantics of all causality relations of a behaviour may contain partial executions in which undefined relations between action occurrences can be still found (e.g., see Figure 5.9). Therefore, a final operation that removes these undefined relations has to be performed. This operation is called *(execution semantics) completion*, and is represented by the operator ✔.

The completion operator ✔ transforms partial executions into regular executions by discarding the independence relation, such that action occurrences with undefined relations are transformed into independent action occurrences. Operator ✔ is defined below.

Definition 5.5 The completion operator ✔ : $\wp(\boldsymbol{XX}) \rightarrow \wp(\boldsymbol{X})$ is defined as:

$$✔(EE) = \{ \langle A, \overline{A}, <, = \rangle \mid \langle A, \overline{A}, <, =, | \rangle \in EE \} \; . \qquad ■$$

We adapt the definition of the execution semantics of a behaviour as presented in the previous chapter to incorporate the completion operator and the context parameters B and Γ_a.

Definition 5.6 The execution semantics of a behaviour B is defined as:

$$\llbracket B \rrbracket = ✔ \otimes \{ \llbracket \rho \rrbracket (B) \mid \rho \in B \} \; ;$$

$$\llbracket \langle a, \Gamma_a, \upsilon_a \rangle \rrbracket (B) = \llbracket \langle a, \Gamma_a \rangle \rrbracket (B) \otimes \llbracket \upsilon_a \rrbracket (\Gamma_a) \; . \qquad ■$$

Example

Figure 5.10 illustrates the application of the definitions presented above to obtain the execution semantics of behaviour B. For simplicity, this behaviour is defined using *must* uncertainty associations, in order to limit the number of possible executions. It is therefore convenient to assume that uncertainty associations get the value *must* by default in the

graphical notation in this example, which is represented by the statement default *must*. For brevity, executions containing the non-occurrence of *b* or the non-occurrence of *c* are omitted, since these executions are disallowed in this behaviour. The execution semantics of the causality relations of actions *a*, *b*, *c* and *d* are obtained using the following equations:

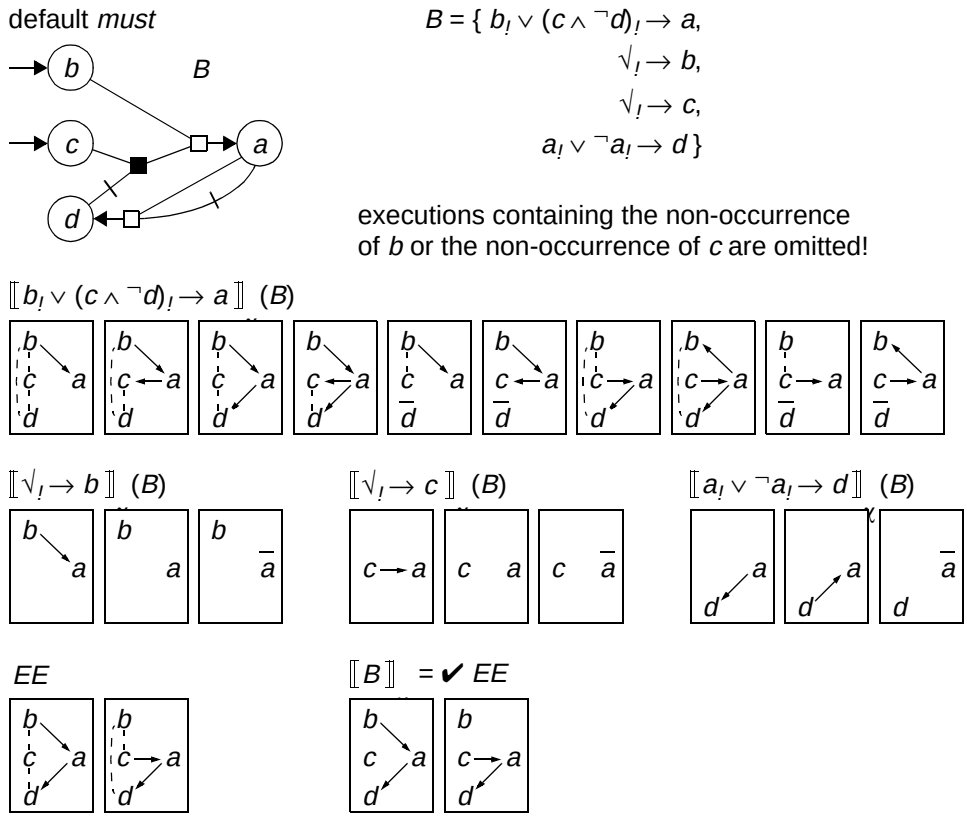


Figure 5.10: Example of execution semantics

$$
\begin{aligned}
[\![\, b_! \vee (c \wedge \neg d)_! \rightarrow a \,]\!]\,(B) \;=\; (\;& [\![\, \langle a, b\rangle \,]\!] \otimes \textit{E-Ind}(a, \{c, d\}) \\
& \cup [\![\, \langle a, c \wedge \neg d\rangle \,]\!] \otimes \textit{E-Ind}(a, \{b\})\;) \\
\otimes\; & \textit{Comp}(\;\textit{Comp}([\![\, \langle a, b, \textit{must}\rangle \,]\!]\;) \otimes \textit{EE-Free}(\{c, d\}) \\
& \cup \textit{Comp}([\![\, \langle a, c \wedge \neg d, \textit{must}\rangle \,]\!]\;) \otimes \textit{EE-Free}(\{b\})\;);
\end{aligned}
$$

$$[\![\, \surd_! \rightarrow b \,]\!]\,(B) = [\![\, \langle b, \surd\rangle \,]\!] \otimes \textit{E-Ind}(b, \{a\}) \otimes [\![\, \langle b, \surd, \textit{must}\rangle \,]\!]\;;$$

$$[\![\, \surd_! \rightarrow c \,]\!]\,(B) = [\![\, \langle c, \surd\rangle \,]\!] \otimes \textit{E-Ind}(c, \{a\}) \otimes [\![\, \langle c, \surd, \textit{must}\rangle \,]\!]\;;$$

$$
\begin{aligned}
[\![\, a_! \vee \neg a_! \rightarrow d \,]\!]\,(B) \;=\; & ([\![\, \langle d, a\rangle \,]\!] \otimes \textit{E-Ind}(d, \varnothing) \cup [\![\, \langle d, \neg a\rangle \,]\!] \otimes \textit{E-Ind}(d, \varnothing)) \\
& \otimes \textit{Comp}(\textit{Comp}([\![\, \langle d, a, \textit{must}\rangle \,]\!]\;) \cup \textit{Comp}([\![\, \langle d, \neg a, \textit{must}\rangle \,]\!]\;)).
\end{aligned}
$$

The execution semantics of *B* is obtained as follows:

$$
\begin{aligned}
\textit{EE} \;&=\; [\![\, b_! \vee (c \wedge \neg d)_! \rightarrow a \,]\!] \otimes [\![\, \surd_! \rightarrow b \,]\!] \otimes [\![\, \surd_! \rightarrow c \,]\!] \otimes [\![\, a_! \vee \neg a_! \rightarrow d \,]\!]\;; \\
[\![\, B \,]\!] \;&=\; \checkmark\, \textit{EE}\,.
\end{aligned}
$$

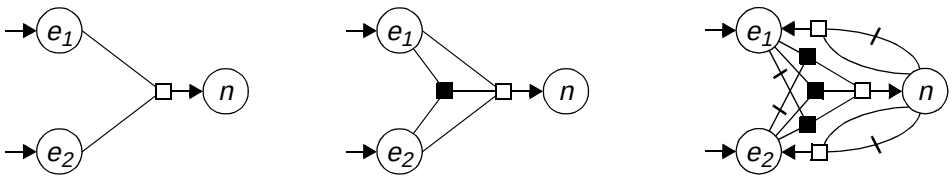
5.2.3 Examples

The use of the *or*-operator is illustrated below by means of some examples.

Example: error notification

Figure 5.11 depicts some examples of the disjunction of enabling (and disabling) conditions. Actions e_1 and e_2 model the occurrences of two distinct error situations, and action n models the notification of these error situations by means of its information attribute. The information attribute is omitted in Figure 5.11. The following cases are distinguished:

- in case of behaviour B_1, action n notifies the occurrence of only one error situation, i.e., either e_1 or e_2, but not both. In this case, the notification is caused by a single error situation and happens independently of the possible occurrence of another error situation;
- in case of behaviour B_2, action n notifies the occurrence of one or both error situations e_1 or e_2. In addition to behaviour B_1, behaviour B_2 allows both errors to be notified when e_1 and e_2 occur before n occurs. However, the notification of both errors is not required, even when they have both occurred before the notification occurs;
- in case of behaviour B_3, action n notifies the occurrence of any error situation that has occurred before n occurs. In order to guarantee this, a two-sided relation has to be defined between action n and actions e_1 and e_2.



$B_1 = \{ e_1 \vee e_2 \rightarrow n,$
$\surd \rightarrow e_1,$
$\surd \rightarrow e_2 \}$

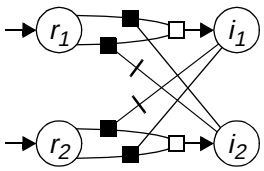
$B_2 = \{ e_1 \vee e_2 \vee$
$\vee (e_1 \wedge e_2) \rightarrow n,$
$\surd \rightarrow e_1,$
$\surd \rightarrow e_2 \}$

$B_3 = \{ (e_1 \wedge e_2) \vee (\neg e_1 \wedge e_2) \vee$
$\vee (e_1 \wedge \neg e_2) \rightarrow n,$
$n \vee \neg n \rightarrow e_1,$
$n \vee \neg n \rightarrow e_2 \}$

Figure 5.11: Disjunction of enabling (and disabling) conditions

Example: unique connection end-point identifiers

Figure 5.12 depicts an example of the disjunction of enabling and disabling conditions. Actions r_1 and r_2 model two independent requests for a unique connection end-point identifier, and actions i_1 and i_2 model the corresponding issues of identifiers. In order to guarantee the uniqueness of both connection end-point identifiers, the occurrences of actions i_1 and i_2 are interleaved, such that in case i_2 occurs after i_1, action i_2 knows which identifier has been issued by i_1, and vice versa.



$B = \{ \quad \surd \rightarrow r_1,$
$\surd \rightarrow r_2,$
$(r_1 \wedge i_2) \vee (r_1 \wedge \neg i_2) \rightarrow i_1,$
$(r_2 \wedge i_1) \vee (r_2 \wedge \neg i_1) \rightarrow i_2 \}$

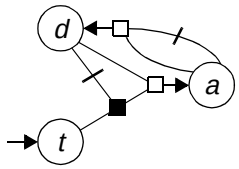
Figure 5.12: Disjunction of enabling and disabling conditions

Example: acknowledgement

Figure 5.13 depicts another example of the disjunction of enabling and disabling conditions. Action *a* models the sending of an acknowledgement, e.g., to inform a remote protocol entity that the last data unit has been received. An acknowledgement is sent

- either after a data unit is received, which is modelled by the occurrence of action *d*;
- or after a time-out occurs, which is modelled by the occurrence of action *t*, and no data unit has been received within the time-out interval.

In the former case, the acknowledgement is sent independently of the time-out. In the latter case, the data unit may still arrive after the acknowledgement has been sent.

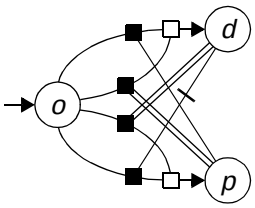


$$B = \{ \quad a \vee \neg a \rightarrow d, \quad \surd \rightarrow t, \quad d \vee (t \wedge \neg d) \rightarrow a \}$$

Figure 5.13: Disjunction of enabling and disabling conditions (2)

Example: pay before or at delivery

Figure 5.14 depicts an example of the disjunction of enabling, disabling and synchronization conditions. Actions *o*, *d* and *p* model the ordering, delivery and payment of a certain product. The two-sided relation between *d* and *p* defines that the ordered product must be paid for either before or at the same time when the product is delivered.



$$B = \{ \quad \surd \rightarrow o, \quad (o \wedge p) \vee (o \wedge {}^{=}p) \rightarrow d, \quad (o \wedge \neg d) \vee (o \wedge {}^{=}d) \rightarrow p \}$$

Figure 5.14: Disjunction of enabling, disabling and synchronization conditions

5.2.4 Disjunctive normal form

An expression representing the causality condition of an action has been represented so far in the so called disjunctive normal form. This is a consequence of the definition of a causality condition as a disjunction of alternative causality conditions, and the requirement to associate an uncertainty value with each alternative causality condition. An expression such as, e.g., $b \wedge (c \vee d)$ is consequently not allowed, since we can not associate uncertainty values to conditions which are not alternative causality conditions.

However, when developing causality relations it may be convenient to describe expressions of causality conditions in different forms, and possibly rewrite them later into disjunctive normal form. Two requirements to allow expressions in non-disjunctive normal form are:

- the possibility to rewrite these expressions into an equivalent disjunctive normal form;

- a default interpretation of uncertainty association values when a causality condition is expressed in non-disjunctive normal form.

Distributivity laws

An expression of a causality condition in non-disjunctive normal form can be rewritten in disjunctive normal form because the *and*-operator distributes over the *or*-operator. A proof of this distributivity law is given in Property 5.7 below.

The complementary distributivity law, i.e., the distribution of the *or*- over the *and*-operator, does not hold. This can be illustrated with the causality conditions $\Gamma 1_a = b \vee (c \wedge d)$ and $\Gamma 2_a = (b \vee c) \wedge (b \vee d)$. Condition $\Gamma 2_a$ allows an execution in which action *a* depends on actions *b* and *c* and is independent of action *d*, whereas condition $\Gamma 1_a$ does not allow such an execution. This is also shown by rewriting condition $\Gamma 2_a$ into disjunctive normal form, using the distributivity of the *and*-operator over the *or*-operator, i.e., $\Gamma 2_a = b \vee (b \wedge c) \vee (b \wedge d) \vee (c \wedge d)$. This implies that $\Gamma 2_a$ is not equivalent to $\Gamma 1_a$.

Default uncertainty association values

Since expressions in non-disjunctive normal form can be rewritten in disjunctive normal form, the alternative causality conditions that are implicitly defined by these expressions can be determined unambiguously. The values of the uncertainty associations of these implicitly defined alternative causality conditions can be defined unambiguously if we assume a default uncertainty association value.

For example, the causality condition $b \wedge (c \vee d)$ implicitly defines the alternative causality conditions $b \wedge c$ and $b \wedge d$. By assuming the default uncertainty association value *may*, this causality condition is equivalent to causality condition $(b \wedge c)_? \vee (b \wedge d)_?$.

The expressive power of causality conditions in non-disjunctive normal form is limited however, since a single uncertainty value is assumed. For example, the condition $(b \wedge c)_? \vee (b \wedge d)_!$ can not be defined in non-disjunctive normal form.

Formal definitions

The distributivity of the *and*-operator over the *or*-operator is formally defined below.

Property 5.7 The *and*-operator distributes over the *or*-operator.

Proof: The execution semantics of $\langle a, \gamma \wedge \Gamma \rangle$, with $\Gamma = \{\gamma_1,.., \gamma_n\}$, and the execution semantics of $\langle a, \{\gamma \wedge \gamma_i \mid \gamma_i \in \Gamma\}\rangle$ are the same for any $\gamma, \gamma_1,.., \gamma_n \in \boldsymbol{C}$.

$[\![\langle a, \gamma \wedge \Gamma\rangle]\!]\ (B)$
$= \ ([\![\langle a, \gamma\rangle]\!] \otimes [\![\langle a, \Gamma\rangle]\!])(B)$, using Def. 5.1 (extended with parameter *B*);
$= \ \cup\{[\![\langle a, \gamma\rangle]\!] \otimes \cup\{[\![\langle a, \gamma_i\rangle]\!] \otimes E\text{-}Ind(a, Ac_{Ind}) \mid Ac_{Ind} = Ac(B) - Ac_{Rel},$
$Ac_{Rel} \in Ac^+(\gamma \wedge \gamma_i, B)\} \mid \gamma_i \in \Gamma\},$
using Def. 5.3, such that behaviour *B* defines $\Gamma_a = \gamma \wedge \Gamma$, and function $Ac^+(\gamma \wedge \gamma_i, B)$ is generalized to causality conditions in non-disjunctive normal form;

$= \cup\{[\![\langle a, \gamma \wedge \gamma_i\rangle]\!] \otimes E\text{-}Ind(a, Ac_{Ind}) \mid Ac_{Ind} = Ac(B) - Ac_{Rel}, Ac_{Rel} \in Ac^{+}(\gamma \wedge \gamma_i, B)\} \mid \gamma_i \in \Gamma\}$, using Def. 5.1;

$= [\![\langle a, \{\gamma \wedge \gamma_i \mid \gamma_i \in \Gamma\}\rangle]\!] (B)$, using Def. 5.3. ■

Syntax

The syntax of our design notation $L_{causality}$ as introduced so far is defined by the following rules, where we assume that causality conditions are defined in disjunctive normal form and uncertainty values are represented as subscripts of alternative causality conditions:

behaviour	=	behaviour_id "= {" causality_relation [","causality_relation]* "}"
causality_relation	=	causality_condition "→" action_id
causality_condition	=	alternative_condition $_{\text{uncertainty\_value}}$ [or-operator alternative_condition $_{\text{uncertainty\_value}}$]*
alternative_condition	=	elementary_condition | composite_condition | "(" composite_condition ")"
elementary_condition	=	start_condition | basic_condition
start_condition	=	√
basic_condition	=	enabling_condition | disabling_condition | synchronization_condition
enabling_condition	=	action_id
disabling_condition	=	"¬"action_id
synchronization_condition	=	"="action_id
composite_condition	=	elementary_condition [and_operator elementary_condition]*
or_operator	=	∨
and_operator	=	∧
uncertainty_value	=	must_value | may_value
must_value	=	!
may_value	=	?
behaviour_id	=	identifier
action_id	=	identifier
identifier	=	character [character]*
character	=	"a" ... "z"

The syntax of the pre-formal design notation $L'_{causality}$ has already been defined in the sections containing formal definitions. Function $[\![\ ]\!]_{\hat{}}$ defines the mapping of expressions in $L_{causality}$ onto expressions in $L'_{causality}$. This mapping is straightforward, except for disjunctive causality conditions which map onto a set of conditions as follows:

$[\![\ \gamma 1 \vee .. \vee \gamma n\ ]\!]_{\hat{}} = \{\gamma 1, .., \gamma n\}$, with $\gamma 1, .., \gamma n \in \boldsymbol{C}$.

5.3 Consistency of causality conditions

This section presents rules for the consistent definition of the causality condition part of behaviours. Behaviours are considered inconsistent when they contain alternative causality conditions that can never be satisfied. Impossible actions are actions that can never occur because none of their alternative causality conditions can ever be satisfied. Impossible actions are undesirable in behaviour definitions and thus should be eliminated when inadvertently specified.

Two types of combination rules for causality conditions are distinguished: (i) rules for defining direct dependencies between actions, and (ii) rules for defining indirect dependencies between actions. These rules are (partly) defined using the notion of alternative behaviour introduced in Section 4.7. First, we generalize this notion to be applicable to behaviours with multiple actions. Subsequently, this notion is applied to define the combination rules for causality conditions.

5.3.1 Alternative behaviours

An alternative behaviour *BA* consisting of multiple actions *Ac* is defined as a set of causality relations, one per action in *Ac*. The causality condition of each action consists of a single alternative causality condition and the corresponding uncertainty association has the value *may*:

$$BA(Ac) = \{\ \gamma_a \rightarrow a\ [\upsilon_a(\gamma_a) = may] \mid a \in Ac,\ \gamma_a \in \boldsymbol{C}\ \}\ .$$

An alternative behaviour *BA* defines one possible combination of alternative causality conditions that enable one or more actions in *Ac* to occur in an execution. The possible combinations are constrained by the combination rules of Section 4.5. These rules are generalized for the case of multiple actions below. For this purpose, the following set of elementary conditions is introduced: $CE(b) = \{b, \neg b, {}^{=}b\}$ for any $b \in \boldsymbol{A}$.

Combination Rule 5:
The alternative causality conditions γ_a and γ_b of two actions *a* and *b* in an alternative behaviour must obey the following rules:

(i)	if $CE(b) \cap C(\gamma_a) = \varnothing$	then	$a \in C(\gamma_b)$ or $CE(a) \cap C(\gamma_b) = \varnothing$;
(ii)	if $b \in C(\gamma_a)$	then	$\neg a \in C(\gamma_b)$ or $CE(a) \cap C(\gamma_b) = \varnothing$;
(iii)	if ${}^{=}b \in C(\gamma_a)$	then	${}^{=}a \in C(\gamma_b)$;
(iv)	if $\neg b \in C(\gamma_a)$	then	$\{a, \neg a\} \cap C(\gamma_b) \neq \varnothing$ or $\gamma_b = \dagger$.

A necessary condition for an alternative behaviour *BA* to be consistent is that the alternative causality conditions of any pair of actions in *Ac*(*BA*) obey these rules. Domain ***BA*** denotes the universe of alternative behaviours which satisfy this necessary condition. Sub-domain $\boldsymbol{BA_{basic}}$ is introduced to denote the domain of alternative behaviours consisting of two actions as identified in Section 4.7, such that $\boldsymbol{BA_{basic}} = \cup\{\boldsymbol{BA(a, b)} \mid a, b \in \boldsymbol{A}, a \neq b\}$, which are called *basic alternative behaviours*.

Alternatively, this necessary consistency condition for some *BA* can be defined in terms of its composition from basic alternative behaviours.

An alternative behaviour *BA* consists of the conjunction of one or more basic alternative behaviours BA_i, such that:

$$BA = \blacksquare_i\, BA_i, \text{ with } BA_i \in \boldsymbol{BA_{basic}},\ Ac(BA) = \cup_i\, Ac(BA_i) \text{ and } i \neq j \Rightarrow Ac(BA_i) \neq Ac(BA_j).$$

The *and*-operator ■ represents the conjunction of alternative behaviours. The conjunction of two alternative behaviours is defined as:

$$BA_1 \blacksquare BA_2 = \{ \begin{array}{ll} \gamma 1_a \wedge \gamma 2_a \rightarrow a\,, & \text{if } \gamma 1_a \rightarrow a \in BA_1 \text{ and } \gamma 2_a \rightarrow a \in BA_2\,; \\ \gamma 1_a \rightarrow a\,, & \text{if } \gamma 1_a \rightarrow a \in BA_1 \text{ and } a \notin Ac(BA_2)\,; \\ \gamma 2_a \rightarrow a\,, & \text{if } \gamma 2_a \rightarrow a \in BA_2 \text{ and } a \notin Ac(BA_1) \\ \mid a \in Ac(BA_1) \cup Ac(BA_2) \}. & \end{array}$$

The conjunction of alternative behaviours is idempotent, commutative and associative.

The set of basic alternative behaviours of an alternative behaviour *BA* is unique, when considering that the choice relation is decomposed into asymmetric exclusion relations (strictly, the choice relation is not an elementary behaviour). This set is denoted by the function *Bas* : $\boldsymbol{BA} \rightarrow \boldsymbol{BA}'_{basic}$, with $\boldsymbol{BA}'_{basic} = \boldsymbol{BA}_{basic} - \{ ⓐ \vdash\!\dashv ⓑ \mid a, b \in \boldsymbol{A}\}$, which must obey the following property:

$$Bas(BA) = \{BA_i \mid i\} \Leftrightarrow BA = \blacksquare_i\, BA_i, \quad \text{with } BA \in \boldsymbol{BA} \text{ and } BA_i \in \boldsymbol{BA}'_{basic}.$$

Example

Figure 5.15(i) depicts the alternative behaviour $BA = \{\surd \rightarrow a, a \wedge \neg c \rightarrow b, a \wedge \neg b \rightarrow c, b \rightarrow d, c \rightarrow e\}$. Figure 5.15(ii) depicts the elements of *Bas*(*BA*). Figure 5.15(iii) shows a graphical shorthand notation for conjunctions of causality conditions in which the symbol ■ is omitted. This shorthand notation allows a straightforward and intuitive composition of alternative behaviours from basic alternative behaviours. Furthermore, it enables the use of the more concise shorthand notation of the choice relation shown in Figure 4.15(i) in Chapter 4, to represent the choice between actions *b* and *c* in Figure 5.15(iii).

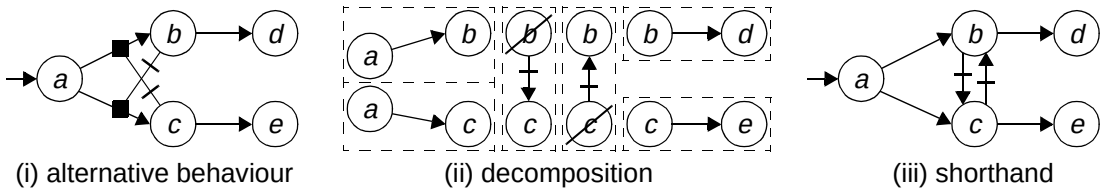


Figure 5.15: Example of an alternative behaviour, its decomposition and its shorthand notation

5.3.2 Disjunctions of alternative behaviours

The causality condition part of a behaviour *B* can be decomposed into a disjunction of one or more alternative behaviours, i.e.:

$$B_\Gamma = \square\, Alt(B_\Gamma),$$

where function $Alt : \boldsymbol{B} \rightarrow \wp(\boldsymbol{BA})$ denotes the set of alternative behaviours in which a behaviour can be decomposed, with $Ac(B_\Gamma) = \{a1, .., an\}$:

$$Alt(B_\Gamma) = \{BA = \{\gamma 1 \rightarrow a1, .. , \gamma n \rightarrow an\} \mid \gamma 1 \in \Gamma_{a1}, .., \gamma n \in \Gamma_{an}, BA \in \boldsymbol{BA}\}.$$

The *or*-operator ❑ represents the disjunction of behaviours. The disjunction of two behaviours is defined as:

$$B_1 \square B_2 = \{ \Gamma 1_a \vee \Gamma 2_a \rightarrow a\,[\upsilon 1_a, \upsilon 2_a], \text{ with } \Gamma 1_a \rightarrow a\,[\upsilon 1_a] \in B_1 \text{ and } \Gamma 2_a \rightarrow a\,[\upsilon 2_a] \in B_2$$

$\mid a \in Ac, Ac = Ac(B_1) = Ac(B_2)\ \}$;
if for any $a \in Ac$: $\upsilon 1_a$ and $\upsilon 2_a$ contain no conflicting uncertainty associations;

$=$ undefined, if otherwise.

The disjunction operation on behaviours is idempotent, commutative and associative.

The possible disjunctions of alternative behaviours are constrained by the following combination rule, which generalizes the combination rules of Section 4.5 on the disjunction of alternative causality conditions towards the case of multiple actions:

Combination Rule 6:
The causality conditions Γ_a and Γ_b of two actions a and b must obey the following rule:

if $\exists \gamma_a \in \Gamma_a \mid \neg b \in C(\gamma_a)$ then $\exists \gamma_b \in \Gamma_b \mid \neg a \in C(\gamma_b)$ or $^{=}a \in C(\gamma_b)$.

Alternatively, this rule can be defined directly in terms of rules on the disjunction of alternative behaviours.

The causality condition part of a behaviour B must obey the following rules for two actions a and b:

(i) if $\exists BA_1 \in Alt(B_\Gamma) \mid a \xrightarrow{-2} b \in Bas(BA_1)$ then
$\exists BA_2, BA_3 \in Alt(B_\Gamma) \mid \not{a} \rightarrow b \in Bas(BA_2)$ or $a = b \in Bas(BA_3)$, or both;

(ii) if $\exists BA_1 \in Alt(B_\Gamma) \mid a \leftarrow \not{b} \in Bas(BA_1)$ then
$\exists BA_2, BA_3 \in Alt(B_\Gamma) \mid \not{a} \rightarrow b \in Bas(BA_2)$ or $a = b \in Bas(BA_3)$, or both.

A necessary condition for the causality condition part of some behaviour B to be consistent is that the causality conditions of any pair of actions in $Ac(B)$ obey Combination Rules 5 and 6.

Alternative behaviours $Alt(B_\Gamma)$ represent all possible combinations of alternative causality conditions of behaviour B_Γ due to which one or more actions can occur. In case the causality condition part B_Γ of a behaviour definition can not be decomposed into a disjunction of alternative behaviours, this behaviour is considered inconsistent. This is because in that case one or more alternative causality conditions are defined for some action(s) which can not be combined with alternative causality conditions of all other actions.

Execution semantics

When the choice relation is also decomposed into the more elementary asymmetric exclusion relations and assuming that all possible actions in an alternative behaviour occur, an alternative behaviour in $Alt(B_\Gamma)$ directly represents the *maximal execution* allowed by this alternative behaviour. The maximal executions of the alternative behaviours in $Alt(B_\Gamma)$ and their possible prefixes constitute the executions that are allowed by B_Γ.

This implies that the execution semantics of the causality condition part of a behaviour definition B_Γ is equal to the union of the execution semantics of the alternative behaviours $Alt(B_\Gamma)$ of this behaviour. This property holds since any combination of alternative causality conditions from B_Γ which is not an alternative behaviour in $Alt(B_\Gamma)$, does not allow any exe-

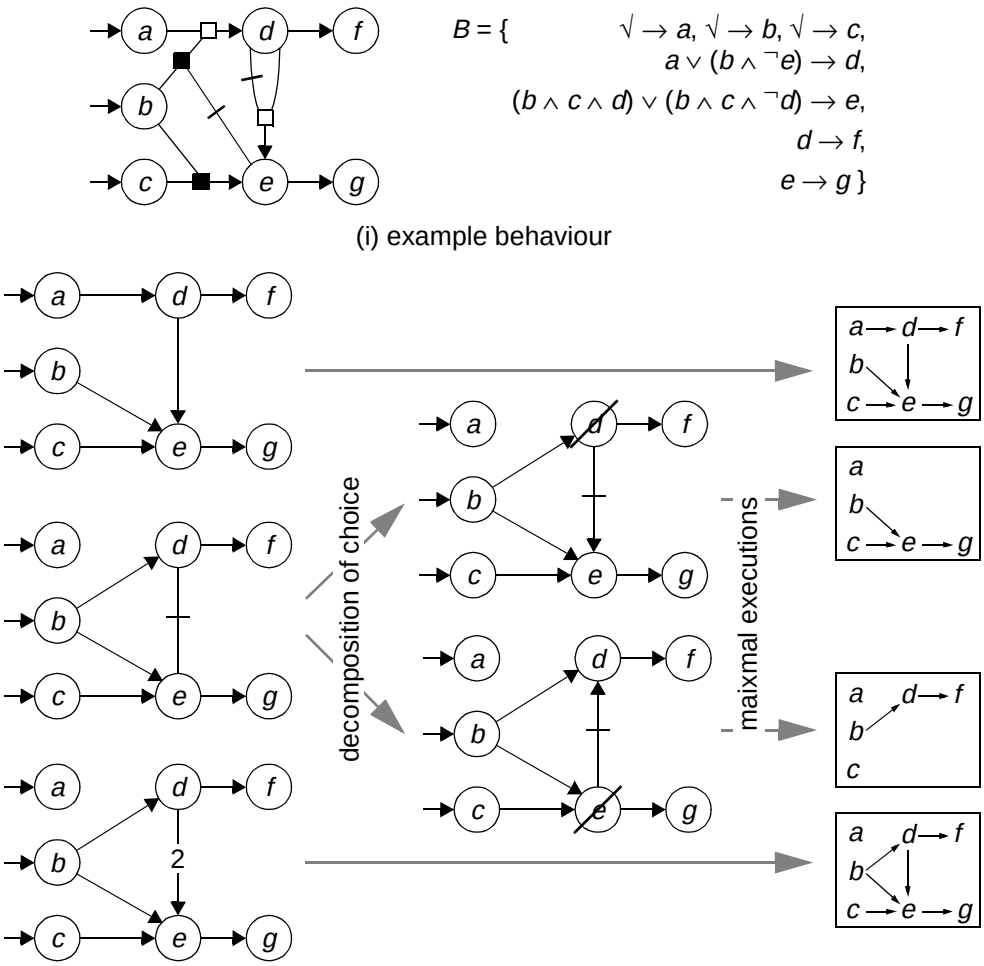
cution that is not allowed by one or more alternative behaviours in $Alt(B_\Gamma)$. A proof of this property is given in Section 5.3.4.

Example

Figure 5.16(i) depicts a behaviour B which defines, amongst others, the disjunction of

- the conjunction of a (one-sided) enabling relation between actions d and e, and a (one-sided) enabling relation between actions a and d; and
- the conjunction of a disabling relation between actions d and e, and a (one-sided) enabling relation between actions b and d.

Figure 5.16(ii) depicts the decomposition of B into alternative behaviours $Alt(B)$ and their corresponding maximal executions.



(i) example behaviour

(ii) decomposition in alternative behaviours

Figure 5.16: Example of disjunction of alternative behaviours

5.3.3 Indirect action relations

Despite the combination rules of the previous sections, the causality condition part of behaviours may still contain inconsistencies caused by the definition of indirect action relations. An *indirect action relation* between two actions *a* and *c* is implicitly defined via a third action *b* by the composition of a relation between *a* and *b* and a relation between *b* and *c*. Figure 5.17 illustrates the composition of an enabling relation between *a* and *b* and an enabling relation between *b* and *c*, which implicitly defines an enabling relation between *a* and *c*. Indirect action relations are represented in grey in the figures that follow.

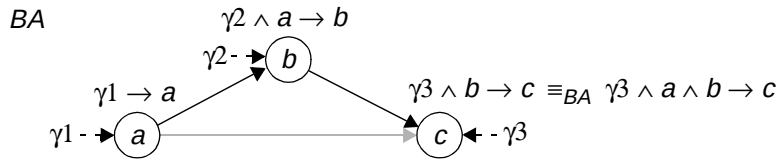


Figure 5.17: Indirect action relations

Indirect action relations can be explicitly added to a behaviour definition, which renders another yet equivalent behaviour definition. For example, alternative causality condition $\gamma 3 \wedge b$ is equivalent to causality condition $\gamma 3 \wedge a \wedge b$ in Figure 5.17. The symbol $\equiv_{BA}$ is used to represent that two causality conditions are equivalent in some (alternative) behaviour *BA*, w.r.t. the explicit or implicit definition of indirect action relations.

The explicit definition of indirect action relations can be used to reveal inconsistencies. For example, consider the following causality relations of actions *a*, *b* and *c*:

$\gamma 1 \wedge {}^{=}c \rightarrow a$,

$\gamma 2 \wedge a \rightarrow b$,

$\gamma 3 \wedge {}^{=}a \wedge b \rightarrow c$.

These causality relations implicitly define an enabling relation between *a* and *c*. The explicit definition of this enabling relation renders the following causality relation of *c*:

$\gamma 3 \wedge a \wedge {}^{=}a \wedge b \rightarrow c$,

which is equivalent to $\dagger \rightarrow c$, since conditions *a* and ${}^{=}a$ are conflicting causality conditions. Consequently, action *c* is an impossible action in this alternative behaviour. Figure 5.18 illustrates this example.

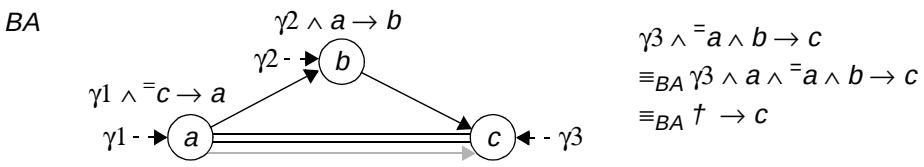


Figure 5.18: Revealing inconsistencies

The presence of an impossible action (inconsistency) in an alternative behaviour *BA* of some behaviour *B* does not necessarily imply that behaviour *B* is inconsistent. This is illustrated by behaviour *B* in Figure 5.19. This behaviour is decomposed into two alternative behaviours BA_1 and BA_2. Alternative behaviour BA_2 contains the impossible action *d*. Despite that, behaviour *B* is not considered inconsistent, since the occurrence of action *d* is allowed by alternative behaviour BA_1.

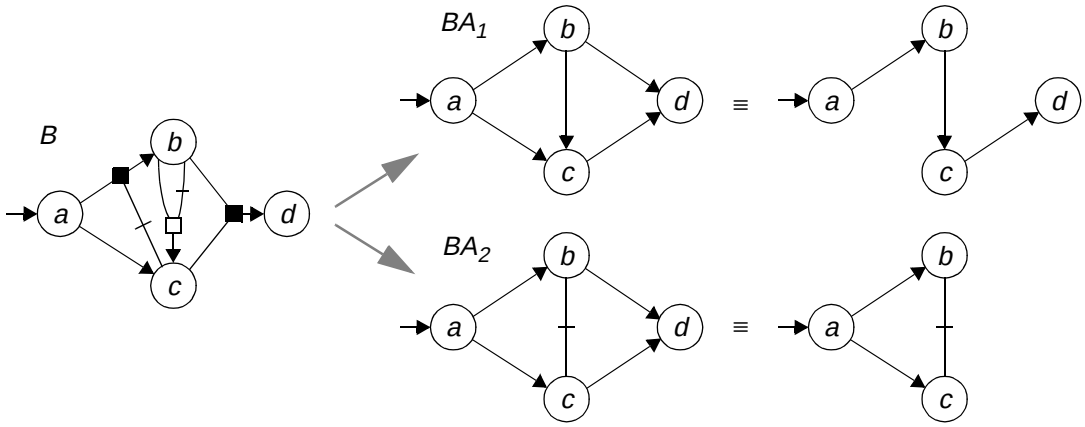


Figure 5.19: Example of consistent behaviour with an inconsistent alternative behaviour

Based on the above, the following minimal consistency requirement is imposed on the causality condition part B_Γ of a behaviour B in order to guarantee that no impossible actions are defined:

> a necessary condition for behaviour B_Γ to be consistent is that for each action of B_Γ at least one alternative behaviour exists in $Alt(B_\Gamma)$ which allows the occurrence of this action.

Identification

Table 5.20 identifies the possible indirect action relations between actions a and c that follow from the composition of an alternative action relation between actions a and b and an alternative action relation between actions b and c.

Table 5.20: Indirect action relations

	$b \rightarrow c$	$b \leftarrow c$	$b = c$	$b \leftarrow\!\!\dashv \cancel{c}$	$\cancel{b} \dashv\!\!\rightarrow c$		$b \dashv c$
$a \rightarrow b$	$a \rightarrow c$	$a - - c$	$a \rightarrow c$	$a - - c$	$a - - c$		$a - - c$
$a \leftarrow b$	$a - - c$	$a \leftarrow c$	$a \leftarrow c$	$a \leftarrow\!\!\dashv \cancel{c}$	$\cancel{a} \dashv\!\!\rightarrow c$		$a \dashv c$
$a = b$	$a \rightarrow c$	$a \leftarrow c$	$a = c$	$a \leftarrow\!\!\dashv \cancel{c}$	$\cancel{a} \dashv\!\!\rightarrow c$		$a \dashv c$
$a \leftarrow\!\!\dashv \cancel{b}$	$a \leftarrow\!\!\dashv \cancel{c}$	$a - - c$	$a \leftarrow\!\!\dashv \cancel{c}$	$a - - c$	$a - - c$		$a - - c$
$\cancel{a} \dashv\!\!\rightarrow b$	$\cancel{a} \dashv\!\!\rightarrow c$	$a - - c$	$\cancel{a} \dashv\!\!\rightarrow c$	$a - - c$	$a - - c$		$a - - c$
$a \dashv b$	$a \dashv c$	$a - - c$	$a \dashv c$	$a - - c$	$a - - c$		$a - - c$

The dashed line between actions a and c in $a - - c$, denotes that no indirect action relation between a and c is implied by the composition of the action relation between a and b and the action relation between b and c. Enabling relation $a \rightarrow b$ can denote a one-sided or a two-sided enabling relation between actions a and b. Given some behaviour B, indirect enabling relation $a \rightarrow c$ should be interpreted as a two-sided enabling relation in case distinct (indirect) action relations between actions a and c are defined in B, and should be interpreted as a one-sided enabling relation otherwise.

Table 5.20 shows the following properties of alternative action relations:

- *identity of the synchronization relation*: the composition of a synchronization relation between actions *a* and *b* with an arbitrary alternative action relation *Rel* between actions *b* and *c* implicitly defines the same action relation *Rel* between actions *a* and *c*;
- *transitivity of the enabling relation*: the composition of the enabling of action *b* by action *a* and the enabling of action *c* by action *b* implicitly defines the enabling of action *c* by action *a*;
- *propagation of the choice relation*: the composition of a choice between actions *a* and *b* and the enabling of action *c* by action *a* (or *b*) implicitly defines a choice between actions *a* (or *b*) and *c*.

Figure 5.21 illustrates these properties for alternative behaviour *BA*. Alternative behaviours BA_1, BA_2 and BA_3 define explicitly the indirect action relations caused by the identity of the synchronization relation, the transitivity of the enabling relation, and the propagation of the choice relation, respectively. Alternative behaviour BA_4 combines these indirect action relations, which introduces additional indirect action relations. These relations are depicted in darker grey with thicker lines.

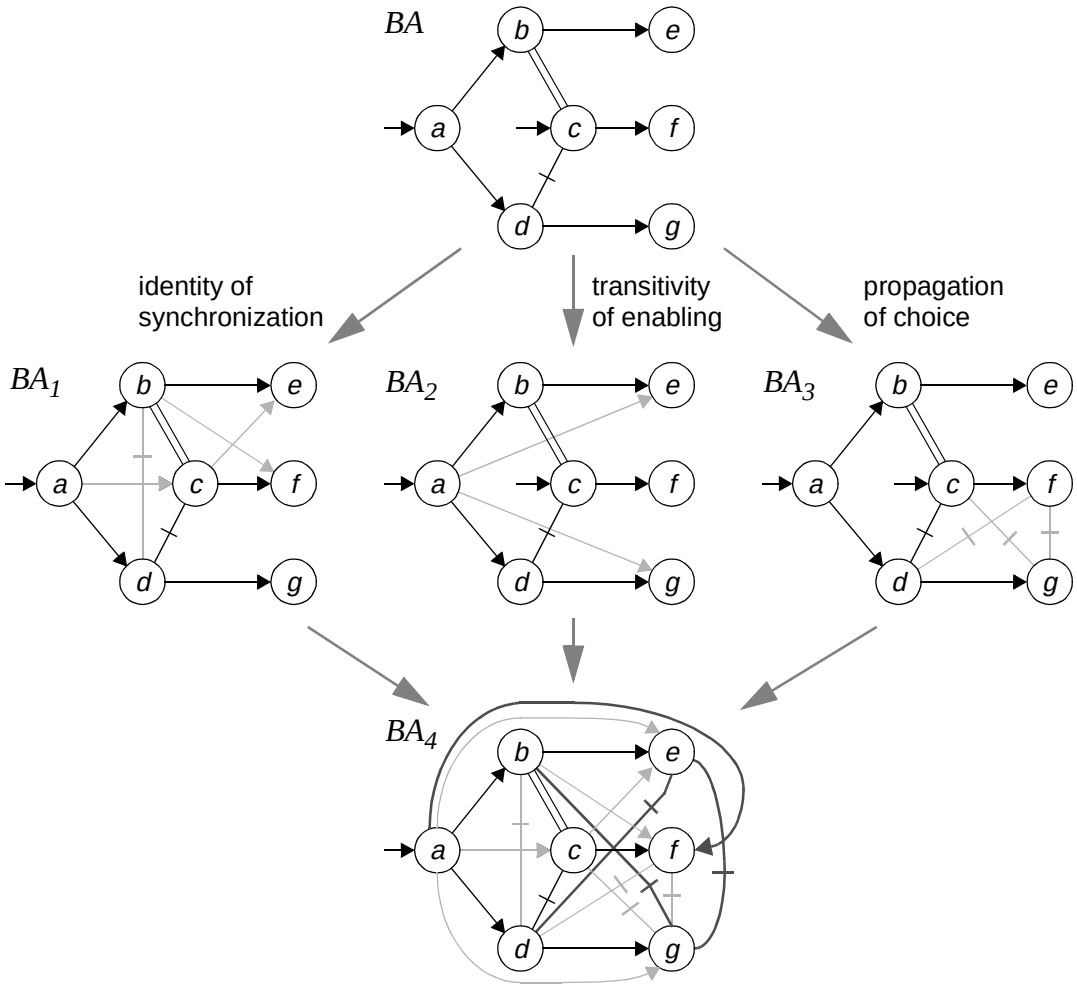


Figure 5.21: Properties of action relations

Dependency equivalent behaviours

The alternative behaviours BA_1, .., BA_4 in Figure 5.21 are obtained from alternative behaviour *BA* by making one or more indirect action relations in *BA* explicit. Table 5.20 represents the rules for making indirect action relations explicit. These rules are applied to alternative behaviours, since indirect action relations are composed exclusively of conjunctions of action relations.

The relationships between BA_1, .., BA_4 and *BA* are denoted as $BA \leqq BA_1$, .., $BA \leqq BA_4$, respectively. Relation $\leqq$ is a partial ordering relation, which is defined as follows:

> $BA1 \leqq BA2$ defines that *BA2* can be obtained from *BA1* by making zero, one or more indirect action relations in *BA1* explicit, using the rules in Table 5.20.

Alternative behaviour *BA1* is considered smaller than (or equal to) alternative behaviour *BA2*, in the sense that *BA1* defines less (or the same) indirect action relations explicitly, when compared to *BA2*. Nonetheless, both *BA1* and *BA2* define the same dependencies (between the same actions), either explicitly or implicitly. Therefore, the alternative behaviours *BA1* and *BA2* are called (*indirect*) *dependency equivalent*, which is denoted as $BA1 \equiv BA2$.

Two alternative behaviours *BA1* and *BA2* are dependency equivalent when the same alternative behaviour *BA* can be obtained from both *BA1* and *BA2*, using the rules in Table 5.20. The (indirect) dependency equivalence relation $\equiv$ is defined as follows:

> $BA1 \equiv BA2 \Leftrightarrow \exists BA \in \boldsymbol{BA} \mid BA1 \leqq BA$ and $BA2 \leqq BA$.

Relation $\equiv$ partitions domain ***BA*** into equivalence classes, where an equivalence class $[BA]_{\equiv}$ consists of all alternative behaviours in ***BA*** that are dependency equivalent with *BA*, i.e., $[BA]_{\equiv} = \{BA1 \in \boldsymbol{BA} \mid BA1 \equiv BA\}$. Partial ordering relation $\leqq$ defines a set of one or more minimal elements, which is denoted as $mins([BA]_{\equiv})$, and defines one maximal element, which is denoted as $max([BA]_{\equiv})$, for each equivalence class $[BA]_{\equiv}$. The alternative behaviours in $mins([BA]_{\equiv})$ define a minimal number of action relations, leaving indirect action relations implicit, while alternative behaviour $max([BA]_{\equiv})$ defines a maximal number of action relations, making all indirect action relations explicit. Alternative behaviour $max([BA]_{\equiv})$ represents the closure of all $BA1 \in [BA]_{\equiv}$ obtained by applying the rules in Table 5.20. The closure of some *BA1* is called the *(indirect) dependency closure* of *BA1*, and is briefly denoted as $BA1^+$, such that $BA1^+ = max([BA1]_{\equiv})$. For example, BA_4 in Figure 5.21 is the indirect dependency closure of *BA*, BA_1, .., BA_4, i.e., $BA_4 = BA^+ = BA_1{}^+ = .. = BA_4{}^+$.

The definitions given above can be generalized to behaviours consisting of disjunctions of alternative behaviours, as follows:

$$
\begin{aligned}
B1 \leqq B2 \Leftrightarrow \quad & \forall BA1 \in Alt(B1), BA2 \in Alt(B2) \mid BA1 \equiv BA2 \Rightarrow BA1 \leqq BA2 \text{ and} \\
& \forall BA1 \in Alt(B1), \exists BA2 \in Alt(B2) \mid BA1 \equiv BA2 \text{ and} \\
& \forall BA2 \in Alt(B2), \exists BA1 \in Alt(B1) \mid BA2 \equiv BA1;
\end{aligned}
$$

$$B1 \equiv B2 \Leftrightarrow \exists B \in \boldsymbol{B} \mid B1 \leqq B \text{ and } B2 \leqq B;$$

$$[B]_{\equiv} = \{B1 \in \boldsymbol{BA} \mid B1 \equiv B\};$$

$B^+ = max([B]_{\equiv})$.

Dependency equivalent causality conditions

Based on the above, the equivalence of causality conditions in alternative behaviours is defined as follows:

$\gamma 1_a \trianglelefteq_{BA} \gamma 2_a \Leftrightarrow \exists BA1 \in [BA]_{\equiv} \mid \gamma_a(BA1) = \gamma 1_a$ and
$\forall BA1 \in [BA]_{\equiv}, \gamma_a(BA1) = \gamma 1_a, \exists BA2 \in [BA]_{\equiv} \mid \gamma_a(BA2) = \gamma 2_a \wedge BA1 \trianglelefteq BA2$;

where $\gamma_a(BA1)$ denotes the alternative causality condition of action a in $BA1$;

$\gamma 1_a \equiv_{BA} \gamma 2_a \Leftrightarrow \exists \gamma_a \in \boldsymbol{C} \mid \gamma 1_a \trianglelefteq_{BA} \gamma_a \wedge \gamma 2_a \trianglelefteq_{BA} \gamma_a$;

Relation $\gamma 1_a \trianglelefteq_{BA} \gamma 2_a$ represents that alternative condition $\gamma 2_a$ can be obtained from $\gamma 1_a$ in equivalence class $[BA]_{\equiv}$, when for any $BA1 \in [BA]_{\equiv}$ in which $\gamma 1_a$ is the alternative condition of a, some $BA2 \in [BA]_{\equiv}$ can be obtained from $BA1$ by making zero, one or more indirect action relations explicit, such that $\gamma 2_a$ is the alternative condition of a in $BA2$. For example, consider alternative conditions $\gamma_g = d$, $\gamma 1_g = d$, $\gamma 2_g = d \wedge a$, $\gamma 3_g = d \wedge \neg c \wedge \neg f$ and $\gamma 4_g = d \wedge a \wedge \neg b \wedge \neg c \wedge \neg f$ of action g in alternative behaviours BA, $BA1$, $BA2$, $BA3$ and $BA4$ of Figure 5.21, respectively. The following relations hold between these conditions:

$\gamma_g \trianglelefteq_{BA} \gamma 1_g$; $\gamma_g \trianglelefteq_{BA} \gamma 2_g$; $\gamma_g \trianglelefteq_{BA} \gamma 3_g$; $\gamma_g \trianglelefteq_{BA} \gamma 4_g$;
$\gamma 2_g \trianglelefteq_{BA} \gamma 4_g$;
$\gamma 3_g \trianglelefteq_{BA} \gamma 4_g$.

Relation $\gamma 1_a \equiv_{BA} \gamma 2_a$ represents that alternative conditions $\gamma 1_a$ and $\gamma 2_a$ are dependency equivalent in the class of alternative behaviours $[BA]_{\equiv}$, when an alternative behaviour $BA1 \in [BA]_{\equiv}$ and an alternative behaviour $BA2 \in [BA]_{\equiv}$ exist, in which $\gamma 1_a$ and $\gamma 2_a$ are defined as alternative condition of action a, respectively. For example, considering the alternative conditions of action g in Figure 5.21, the following holds:

$\gamma_g \equiv_{BA} \gamma 1_g \equiv_{BA} \gamma 2_g \equiv_{BA} \gamma 3_g \equiv_{BA} \gamma 4_g$.

The above implies that relation $\equiv_{BA}$ defines a class of equivalent alternative causality conditions for each result action $a \in Ac(BA)$ in the context of alternative behaviour class $[BA]_{\equiv}$. Assuming γ_a is an element of this set, this set is defined as:

$[\gamma_a]_{\equiv BA} = \{\gamma_a(BA1) \mid BA1 \in [BA]_{\equiv}\}$.

Partial ordering relation $\trianglelefteq_{BA}$ defines a set of one or more minimal elements, which is denoted as $mins([\gamma_a(BA)]_{\equiv BA})$, and defines one maximal element, which is denoted as $max([\gamma_a]_{\equiv BA})$, in class $[\gamma_a]_{\equiv BA}$. The alternative conditions in $mins([\gamma_a]_{\equiv BA})$ represent the *minimal definitions* of all $\gamma 1_a \in [\gamma_a]_{\equiv BA}$. Alternative condition $max([\gamma_a]_{\equiv BA})$ represents the *maximal definition*, or closure, of all $\gamma 1_a \in [\gamma_a]_{\equiv BA}$ w.r.t. the application of the rules in Table 5.20. The closure of $\gamma 1_a$ is called the *(indirect) dependency closure* of $\gamma 1_a$ and is briefly denoted as $\gamma 1_a^{+BA}$, such that: $\gamma 1_a^{+BA} = \gamma_a(BA^+)$. For example, consider action g in Figure 5.21:

$\{\gamma_g, \gamma 1_g, \gamma 2_g, \gamma 3_g, \gamma 4_g\} \subseteq [\gamma_g]_{\equiv BA}$ and $\gamma_g^{+BA} = \gamma 4_g$.

In case equivalence class $[BA]_{\equiv}$ is clear from the context, the closure of some alternative causality condition γ_a may be denoted as γ_a^+.

The above definitions can be extended to behaviours consisting of disjunctions of alternative behaviours, as follows:

$$\gamma 1_a \trianglelefteq_B \gamma 2_a \Leftrightarrow \exists BA \in Alt(B) \mid \gamma 1_a \in [\gamma_a]_{\equiv BA} \text{ and } \forall BA \in Alt(B) \mid \gamma 1_a \in [\gamma_a]_{\equiv BA} \Rightarrow \gamma 1_a \trianglelefteq_{BA} \gamma 2_a;$$

$$\gamma 1_a \equiv_B \gamma 2_a \Leftrightarrow \exists BA \in Alt(B) \mid \gamma 1_a, \gamma 2_a \in [\gamma_a]_{\equiv BA} \text{ and } \forall BA \in Alt(B) \mid \gamma 1_a \in [\gamma_a]_{\equiv BA} \Leftrightarrow \gamma 2_a \in [\gamma_a]_{\equiv BA};$$

$$[\Gamma_a]_{\equiv B} = \{\Gamma_a(B1) \mid B1 \in [B]_{\equiv}\};$$

$$\Gamma_a{}^{+B} = \Gamma_a(B^+),$$

where $\Gamma_a(B1)$ denotes the causality condition of action a in $B1$.

Super- and sub-conditions

The notion of *sub-condition* is introduced to denote implications between the satisfaction of distinct alternative causality conditions. The 'sub-condition of' relationship between two alternative conditions $\gamma 1_a$ and $\gamma 2_a$ of some action a is defined as:

$\gamma 1_a \vdash \gamma 2_a$ defines that $\gamma 2_a$ is a *sub-condition* of $\gamma 1_a$, such that the satisfaction of $\gamma 1_a$ implies the satisfaction of $\gamma 2_a$.

Alternatively, condition $\gamma 1_a$ is called a *super-condition* of condition $\gamma 2_a$. A distinction is made between the 'sub-condition of' (or 'super-condition of') relationship in alternative behaviours and in behaviours consisting of disjunctions of alternative behaviours.

Alternative condition $\gamma 2_a$ is a sub-condition of alternative condition $\gamma 1_a$ in any alternative behaviour of $[BA]_{\equiv}$, in case $\gamma 2_a$ is completely contained in the dependency closure of $\gamma 1_a$, assuming $\gamma 1_a \in [\gamma_a]_{\equiv BA}$. In this case, the satisfaction of $\gamma 1_a$ implies the satisfaction of $\gamma 2_a$ for any $BA1 \in [BA]_{\equiv}$. This is defined as follows:

$$\gamma 1_a \vdash_{BA} \gamma 2_a \Leftrightarrow \gamma 1_a \in [\gamma_a]_{\equiv BA} \wedge C(\gamma 2_a) \subseteq C(\gamma 1_a{}^{+BA}) .$$

Alternative condition $\gamma 2_a$ is a sub-condition of alternative condition $\gamma 1_a$ in any behaviour of $[B]_{\equiv}$, in case the dependency closure of $\gamma 2_a$ is completely contained in the dependency closure of $\gamma 1_a$ for any pair of equivalence classes $[BA1]_{\equiv}$ and $[BA2]_{\equiv}$, with $BA1, BA2 \in Alt(B)$, such that $\gamma 1_a \in [\gamma_a]_{\equiv BA1}$ and $\gamma 2_a \in [\gamma_a]_{\equiv BA2}$. In this case, the satisfaction of $\gamma 1_a$ implies the satisfaction of $\gamma 2_a$ for any pair of alternative behaviours $BA1 \in [BA1]_{\equiv}$ and $BA2 \in [BA2]_{\equiv}$, respectively. This is defined as follows:

$$\gamma 1_a \vdash_B \gamma 2_a \Leftrightarrow \exists BA1, BA2 \in Alt(B) \mid \gamma 1_a \in [\gamma_a]_{\equiv BA1} \wedge \gamma 2_a \in [\gamma_a]_{\equiv BA2} \text{ and } \forall BA1, BA2 \in Alt(B) \mid \gamma 1_a \in [\gamma_a]_{\equiv BA1} \wedge \gamma 2_a \in [\gamma_a]_{\equiv BA2} \Rightarrow C(\gamma 2_a{}^{+BA2}) \subseteq C(\gamma 1_a{}^{+BA1}).$$

The 'sub-condition of' relationships obey the following properties:

1. $\gamma 1_a \vdash_{BA} \gamma 2_a \wedge \gamma 2_a \vdash_{BA} \gamma 1_a \Leftrightarrow \gamma 1_a \equiv_{BA} \gamma 2_a$;
2. $\gamma 1_a \vdash_B \gamma 2_a \wedge \gamma 2_a \vdash_B \gamma 1_a \Leftrightarrow \gamma 1_a \equiv_B \gamma 2_a$.

The proof of the first property is straightforward. The proof of the second property is derived as follows:

$\gamma 1_a \vdash_B \gamma 2_a$ and $\gamma 2_a \vdash_B \gamma 1_a$
$\Leftrightarrow \exists BA1, BA2 \in Alt(B) \mid \gamma 1_a \in [\gamma_a]_{\equiv BA1} \wedge \gamma 2_a \in [\gamma_a]_{\equiv BA2}$ and
$\forall BA1, BA2 \in Alt(B) \mid \gamma 1_a \in [\gamma_a]_{\equiv BA1} \wedge \gamma 2_a \in [\gamma_a]_{\equiv BA2} \Rightarrow C(\gamma 2_a{}^{+BA2}) = C(\gamma 1_a{}^{+BA1})$
$\Leftrightarrow \exists BA1, BA2 \in Alt(B) \mid \gamma 1_a \in [\gamma_a]_{\equiv BA1} \wedge \gamma 2_a \in [\gamma_a]_{\equiv BA2}$ and
$\forall BA1, BA2 \in Alt(B) \mid \gamma 1_a \in [\gamma_a]_{\equiv BA1} \wedge \gamma 2_a \in [\gamma_a]_{\equiv BA2} \Rightarrow [\gamma_a]_{\equiv BA2} = [\gamma_a]_{\equiv BA1}$
$\Leftrightarrow \exists BA \in Alt(B) \mid \gamma 1_a, \gamma 2_a \in [\gamma_a]_{\equiv BA}$ and
$\forall BA \in Alt(B) \mid \gamma 1_a \in [\gamma_a]_{\equiv BA} \Leftrightarrow \gamma 2_a \in [\gamma_a]_{\equiv BA}$
$\Leftrightarrow \gamma 1_a \equiv_B \gamma 2_a$.

5.3.4 Formal definition

The decomposition of a behaviour definition consisting of multiple actions *Ac* into a causality condition part and an uncertainty attribute part is formally defined below.

Property 5.8 A behaviour definition $B(Ac)$ can be decomposed into

- a causality condition part $B_\Gamma(Ac) = \{\langle a, \Gamma_a, \upsilon_a' \rangle \mid a \in Ac\}$, with: $\forall \gamma_a \in \Gamma_a \mid \upsilon_a'(\gamma_a) = may$;
- an uncertainty attribute part $B_\upsilon(Ac) = \{\upsilon_a \mid a \in Ac\}$,

such that: $[\![B(Ac)]\!] = \checkmark ([\![B_\Gamma(Ac)]\!] \otimes [\![B_\upsilon(Ac)]\!])$,

with $[\![B_\upsilon(Ac)]\!] = \otimes\{[\![\upsilon]\!] \mid \upsilon \in B_\upsilon(Ac)\}$.

Proof: This property follows immediately from Definition 5.6, and the may-interpretation of causality conditions. ■

The *and*-operator ■ on alternative behaviours is defined below.

Definition 5.9 The conjunction of two alternative behaviours BA_1 and BA_2 is defined as:

$$
\begin{aligned}
BA_1 \blacksquare BA_2 = \{ \; & \langle a, \{\gamma 1 \wedge \gamma 2\}, \{\langle a, \gamma 1 \wedge \gamma 2, may\rangle\}\rangle, \\
& \quad \text{if } \langle a, \{\gamma 1\}, \{\langle a, \gamma 1, may\rangle\}\rangle \in BA_1, \langle a, \{\gamma 2\}, \{\langle a, \gamma 2, may\rangle\}\rangle \in BA_2 ; \\
& \langle a, \{\gamma 1\}, \{\langle a, \gamma 1, may\rangle\}\rangle, \\
& \quad \text{if } \langle a, \{\gamma 1\}, \{\langle a, \gamma 1, may\rangle\}\rangle \in BA_1 \text{ and } a \notin Ac(BA_2) ; \\
& \langle a, \{\gamma 2\}, \{\langle a, \gamma 2, may\rangle\}\rangle, \\
& \quad \text{if } \langle a, \{\gamma 2\}, \{\langle a, \gamma 2, may\rangle\}\rangle \in BA_2 \text{ and } a \notin Ac(BA_1) \\
& \mid a \in Ac(BA_1) \cup Ac(BA_2) \}.
\end{aligned}
$$
■

Property 5.10 The *and*-operator $\blacksquare : \boldsymbol{BA} \times \boldsymbol{BA} \rightarrow \boldsymbol{BA}$ is idempotent, commutative and associative.

Proof: These properties are inherited from the *and*-operator $\wedge$. ■

The *or*-operator ❑ on behaviours with the same action domain is defined below.

Definition 5.11 The disjunction of two behaviours B_1 and B_2, with $Ac(B_1) = Ac(B_2)$, is defined as:

$B_1 \square B_2 = \{\langle a, \Gamma 1_a \cup \Gamma 2_a, \upsilon 1_a \cup \upsilon 2_a\rangle \mid a \in Ac\}$, with $\langle a, \Gamma 1_a, \upsilon 1_a\rangle \in B_1$ and $\langle a, \Gamma 2_a, \upsilon 2_a\rangle \in B_2$ if $\forall a \in Ac(B_1), \gamma \in \Gamma 1_a \cap \Gamma 2_a \mid \upsilon 1_a(\gamma) = \upsilon 2_a(\gamma)$;

$= \text{undefined}, \quad \text{otherwise.}$ ■

Property 5.12 The *or*-operator $\square : \boldsymbol{B} \times \boldsymbol{B} \rightarrow \boldsymbol{B}$ is idempotent, commutative and associative.

Proof: These properties are inherited from the union operator $\cup$ on sets. ■

The following property formally defines that the execution semantics of the causality condition part of a behaviour definition is equal to the union of the execution semantics of the alternative behaviours of this behaviour definition.

Property 5.13 Assuming the causality condition part B_Γ of behaviour B obeys the combination rules of Sections 5.3.1, 5.3.2 and 5.3.3, the following property holds:

$[\![B_\Gamma]\!] = \cup\{[\![BA]\!] \mid BA \in Alt(B_\Gamma)\}$.

Proof: The proof of this property consists of the following steps:

$[\![B_\Gamma]\!]$

$= \checkmark \otimes\{[\![\rho]\!] (B_\Gamma) \mid \rho \in B_\Gamma\}$, using Def. 5.6;

$= \checkmark \otimes\{[\![\langle a, \Gamma_a, \upsilon_a\rangle]\!] (B_\Gamma) \mid a \in Ac(B_\Gamma)\}$, with $\rho = \langle a, \Gamma_a, \upsilon_a\rangle$;

$= \checkmark \otimes\{[\![\langle a, \Gamma_a\rangle]\!] (B_\Gamma) \otimes [\![\upsilon_a]\!] (\Gamma_a) \mid a \in Ac(B_\Gamma)\}$, using Def. 5.6;

$= \checkmark \otimes\{[\![\langle a, \Gamma_a\rangle]\!] (B_\Gamma) \mid a \in Ac(B_\Gamma)\}$, using the may-interpretation of Γ_a;

$= \checkmark \otimes\{\cup\{[\![\langle a, \{\gamma_a\}\rangle]\!] (B_\Gamma) \mid \gamma_a \in \Gamma_a\} \mid a \in Ac(B_\Gamma)\}$, using Def. 5.3;

$= \checkmark \otimes\{\cup\{[\![\langle a, \{\gamma_a\}\rangle]\!] (B_\Gamma) \mid \gamma_a \in \Gamma_a\} \mid a \in \{a1, .., an\}\}$, with $Ac(B_\Gamma) = \{a1, .., an\}$;

$= \checkmark \cup\{\otimes\{[\![\langle a1, \{\gamma_{a1}\}\rangle]\!] (B_\Gamma), .., [\![\langle an, \{\gamma_{an}\}\rangle]\!] (B_\Gamma) \mid \gamma_{a1} \in \Gamma_{a1}, .., \gamma_{an} \in \Gamma_{an}\}$, using the distributivity of $\otimes$ over $\cup$;

$= \checkmark \cup\{\otimes\{[\![\langle a1, \{\gamma_{a1}\}\rangle]\!] (BA(\gamma_{a1}, .., \gamma_{an})), .., [\![\langle an, \{\gamma_{an}\}\rangle]\!] (BA(\gamma_{a1}, .., \gamma_{an}))$ $\mid \gamma_{a1} \in \Gamma_{a1}, .., \gamma_{an} \in \Gamma_{an}\}$, with: $BA(\gamma_{a1}, .., \gamma_{an}) = \{\langle a1, \{\gamma_{a1}\}\rangle, .., \langle an, \{\gamma_{an}\}\rangle\}$;

$BA(\gamma_{a1}, .., \gamma_{an})$ ranges over all possible behaviours consisting of a causality relation for all actions $a \in Ac(B_\Gamma)$, such that the causality condition of a consists of a single alternative causality condition $\gamma_a \in \Gamma_a$. Observe that $BA(\gamma_{a1}, .., \gamma_{an})$ is not necessarily an element of domain $\boldsymbol{BA}$, since it may disobey some combination rules.

$= \checkmark \cup\{[\![BA(\gamma_{a1}, .., \gamma_{an})]\!] \mid \gamma_{a1} \in \Gamma_{a1}, .., \gamma_{an} \in \Gamma_{an}\}$, using Def. 5.6;

$= \checkmark \cup\{[\![BA(\gamma_{a1}, .., \gamma_{an})]\!] \mid \gamma_{a1} \in \Gamma_{a1}, .., \gamma_{an} \in \Gamma_{an}, BA(\gamma_{a1}, .., \gamma_{an}) \in \boldsymbol{BA}\}$,

This step is based on the property that there exists no $BA(\gamma_{a1}, .., \gamma_{an})$, with $\gamma_{a1} \in \Gamma_{a1}, .., \gamma_{an} \in \Gamma_{an}$, such that $BA(\gamma_{a1}, .., \gamma_{an}) \notin \boldsymbol{BA}$ and $BA(\gamma_{a1}, .., \gamma_{an})$ allows an execution which is not allowed by one of the alternative behaviours $BA \in Alt(B_\Gamma) \subset \boldsymbol{BA}$. The proof of this property is by contradiction.

Assume that there exists a $BA(\gamma_{a1}, .., \gamma_{an}) \notin \boldsymbol{BA}$, which is briefly denoted as $BA1$, such that $BA1$ allows an execution χ which is not allowed by one of the alternative behaviours $BA \in Alt(B_\Gamma)$. This implies that execution χ is not a prefix of the maximal executions allowed by all $BA \in Alt(B_\Gamma)$, which are denoted as E_{max}. This is only possible

when one of the following conditions are satisfied.

(i) execution χ contains additional action occurrences when compared to the executions in E_{max}; or

(ii) one or more pairs of action occurrences in execution χ are related differently when compared to the executions in E_{max}; or

(iii) both *(i)* and *(ii)* apply.

This would imply that $BA1 \in \boldsymbol{BA}$, since additional action occurrences or differently related action occurrences are not allowed by combinations of causality conditions which do not obey Combination Rule 5. Instead, for each action pair with causality conditions which do not obey these combination rules, either the occurrence of one of them or the occurrences of both of them are disallowed (and the actions which depend on their occurrences). This implies that execution χ is impossible, which contradicts the assumption that *BA1* exists;

$= \checkmark \cup \{[\![BA]\!] \mid BA \in Alt(B_\Gamma)\}$. ■

5.4 Consistency of uncertainty attribute

This section presents rules for the consistent definition of the uncertainty attribute part of behaviours. The uncertainty attribute part is concerned with the additional constraints defined by *must*-uncertainty associations on top of the constraints defined by the causality condition part of behaviours, which is defined assuming the default *may*-interpretation for uncertainty.

Inconsistencies are normally possible when combinations of *must*- and *may*-uncertainty associations are defined. Consistency rules are presented for the following cases: (i) uncertainty associations of the same result action, (ii) uncertainty associations of synchronized actions, and (iii) uncertainty associations of mutually exclusive actions.

5.4.1 Uncertainty associations of the same result action

Figure 5.22 shows the inconsistent definition of uncertainty associations $\upsilon_a(c)$ and $\upsilon_a(b \wedge c)$ of action *a*. The *may*-uncertainty association $\upsilon_a(b) = ?$ defines that *a* may occur when *b* has occurred. This uncertainty association does not impose any restriction on the possible values of $\upsilon_a(c)$ and $\upsilon_a(b \wedge c)$. The *must*-uncertainty association $\upsilon_a(c) = !$ defines that *a* must occur when *c* has occurred, irrespective of any other conditions that are satisfied. This implies that action *a* must also occur when actions *b* and *c* have occurred. Consequently, $\upsilon_a(c) = !$ implies $\upsilon_a(b \wedge c) = !$, since the satisfaction of alternative causality condition $b \wedge c$ implies the satisfaction of alternative causality condition *c*, i.e., $b \wedge c \vdash_B c$.

Figure 5.23 shows another example of the inconsistent definition of the uncertainty associations of a single action. In this behaviour, the satisfaction of condition $\gamma 1_a = b \wedge c$ implies the satisfaction of $\gamma 2_a = c \wedge d$. However, this can not be derived from $\gamma 1_a$ and $\gamma 2_a$ directly, since $C(c \wedge d)$ is not a subset of $C(b \wedge c)$. Instead the indirect dependency closures of $\gamma 1_a$ and $\gamma 2_a$ have to be determined for each alternative behaviour *BA1* and *BA2* in *B* in which $\gamma 1_a$ and $\gamma 2_a$

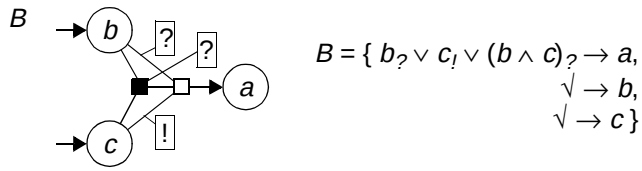


Figure 5.22: Must uncertainty associations implied by sub-conditions

are defined as alternative conditions of *a*, respectively. In this case only one such *BA1* and one such *BA2* exist:

$BA1 = \{b \wedge c \rightarrow a,\ d \rightarrow b,\ d \rightarrow c,\ \surd \rightarrow d\}$;

$BA2 = \{c \wedge d \rightarrow a,\ d \rightarrow b,\ d \rightarrow c,\ \surd \rightarrow d\}$.

The indirect dependency closures of $\gamma 1_a$ and $\gamma 2_a$ in $[BA1]_{\equiv}$ and $[BA2]_{\equiv}$ are:

$\gamma 1_a^{+BA1} = b \wedge c \wedge d$;

$\gamma 2_a^{+BA2} = c \wedge d$.

This implies $C(\gamma 2_a^{+BA2}) \subseteq C(\gamma 1_a^{+BA1})$ for any pair of alternative behaviours *BA2* and *BA1* in which $\gamma 2_a$ and $\gamma 1_a$ are defined, respectively. Consequently, we conclude that $b \wedge c \vdash_B c \wedge d$, and therefore $\upsilon_a(c \wedge d) = !$ implies $\upsilon_a(b \wedge c) = !$.

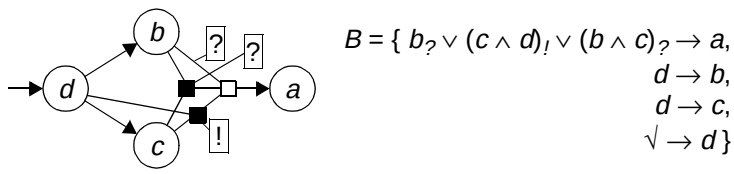


Figure 5.23: Use of indirect dependency closure

Consistency rule

In general, the *must*-uncertainty association of an alternative causality condition $\gamma 2_a$ implies the *must*-uncertainty association of another alternative causality condition $\gamma 1_a$ of the same result action *a*, when $\gamma 2_a$ is a sub-condition of $\gamma 1_a$. This follows from the fact that the situations represented by condition $\gamma 2_a$ in which action *a* must occur comprise all situations represented by condition $\gamma 1_a$ in which action *a* must occur.

Assuming that $\gamma 1_a$ and $\gamma 2_a$ are alternative causality conditions of action *a*, the uncertainty associations $\upsilon_a(\gamma 1_a)$ and $\upsilon_a(\gamma 2_a)$ must obey the following consistency rule:

if $\gamma 1_a \vdash_B \gamma 2_a$ then $\upsilon_a(\gamma 1_a) \geq \upsilon_a(\gamma 2_a)$, with *must* > *may*.

Alternative consistency rule

Alternatively, the above consistency rule can be based on a weaker notion of sub-condition. Instead of using $\gamma 1_a \vdash_B \gamma 2_a$, which states that the satisfaction of $\gamma 1_a$ implies the satisfaction of $\gamma 2_a$ for any pair of alternative behaviours *BA1*, *BA2* $\in$ *Alt*(*B*) in which $\gamma 1_a$ and $\gamma 2_a$ are defined, respectively, one may use a weaker condition which states that the satisfaction of $\gamma 1_a$ implies the satisfaction of $\gamma 2_a$ for some (at least one) of these pairs. The consequences of this weaker condition for the above consistency rule is illustrated by means of an example.

Figure 5.24 depicts a behaviour *B* in which action *a* either depends on action *b* or depends on action *c*, and actions *b* and *c* are interleaved. In addition, some alternative behaviours of *B* are shown.

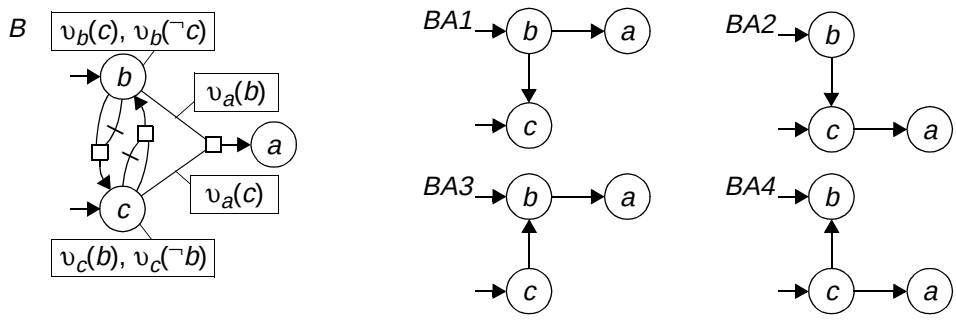


Figure 5.24: Interleaving of *b* and *c*

We consider the alternative causality conditions $\gamma 1_a = b$ and $\gamma 2_a = c$ of action *a*. According to the original interpretation of the notion of sub-condition the following holds: $b \not\vdash_B c$ and $c \not\vdash_B b$, such that $\upsilon_a(b)$ and $\upsilon_a(c)$ can be defined independently of each other. For example, condition $\gamma 1_a = b$ in *BA1* does not imply condition $\gamma 2_a = c$ in *BA4*, and condition $\gamma 2_a = c$ in *BA4* does not imply condition $\gamma 1_a = b$ in *BA1*. However, according to the alternative interpretation of the notion of sub-condition the following holds: $b \vdash'_B c$ and $c \vdash'_B b$, such that $\upsilon_a(b) = \upsilon_a(c)$. For example, condition $\gamma 1_a = b$ in *BA3* implies condition $\gamma 2_a = c$ in *BA2*, and condition $\gamma 2_a = c$ in *BA2* implies condition $\gamma 1_a = b$ in *BA3*.

The weaker notion of sub-condition restricts the possible variations of *B* w.r.t. the uncertainty attribute. For example, it does not allow one to define a behaviour in which (i) *a* must occur when *b* occurs and *c* does not occur, and (ii) *a* may (not) occur when *c* occurs and *b* does not occur. Since we see no motivation for such restrictions, we adopt the original notion of sub-condition in the remainder of this thesis.

Formal definition

A proof of the (original) consistency rule presented on page 174 is given in terms of the execution model. The consequence for the execution semantics of $\upsilon_a(\gamma 1_a) \geq \upsilon_a(\gamma 2_a)$ is that any execution disallowed by $\upsilon_a(\gamma 1_a) = !$ is also disallowed by $\upsilon_a(\gamma 2_a) = !$. This gives the following alternative definition of the consistency rule.

Property 5.14 Assuming $\gamma 1_a$ and $\gamma 2_a$ are alternative causality conditions of result action *a* in behaviour *B*, the following holds:

if $\gamma 1_a \vdash_B \gamma 2_a$ then for each pair $BA1, BA2 \in Alt(B)$, such that $\gamma_a(BA1) = \gamma 1_a$ and $\gamma_a(BA2) = \gamma 2_a$, the following holds:

$$Comp([\![\langle a, \gamma 1_a{}^+, must\rangle]\!]) \subseteq Comp([\![\langle a, \gamma 2_a{}^+, must\rangle]\!]) \otimes EE, \quad \text{if } Ac(\gamma 2_a{}^+) \subset Ac(\gamma 1_a{}^+)$$
$$Comp([\![\langle a, \gamma 1_a{}^+, must\rangle]\!]) = Comp([\![\langle a, \gamma 2_a{}^+, must\rangle]\!]) \quad \text{if otherwise}$$

where

- $Comp([\![\langle a, \gamma 1_a{}^+, must\rangle]\!])$ and $Comp([\![\langle a, \gamma 2_a{}^+, must\rangle]\!])$ represent the sets of executions disallowed by $\upsilon_a(\gamma 1_a) = !$ and $\upsilon_a(\gamma 2_a) = !$, respectively;

- $EE = EE\text{-}Free(Ac(\gamma 1_a^+) - Ac(\gamma 2_a^+))$ is needed to equalize the action domains of the execution sets at both sides of the $\subset$ operator. The cross-conjunction $Comp(\llbracket\langle a, \gamma 2_a^+, must\rangle\rrbracket) \otimes EE$ is based on the property that an execution $e\chi_1 \in Comp(\llbracket\langle a, \gamma 2_a^+, must\rangle\rrbracket)$ disallows any execution $e\chi_2$, with $e\chi_1 \sqsubseteq e\chi_2$ and $Ac(e\chi_1) \subseteq Ac(e\chi_2)$. For example, if execution $\boxed{a\quad}$ is disallowed then executions $\boxed{\bar{a}\quad b}$ and $\boxed{\bar{a}\quad \bar{b}}$ are also disallowed.

Proof: The proof of this property is as follows:

For each pair $BA1, BA2 \in Alt(B)$, such that $\gamma_a(BA1) = \gamma 1_a$ and $\gamma_a(BA2) = \gamma 2_a$, the following holds:

$C(\gamma 2_a^+) \subseteq C(\gamma 1_a^+)$, which implies $\gamma 1_a^+ = \gamma 2_a^+ \wedge \gamma$, with $\gamma \in \boldsymbol{C}$;

if $\gamma \neq \surd$:

$Comp(\llbracket\langle a, \gamma 1_a^+, must\rangle\rrbracket) = Comp(\llbracket\langle a, \gamma 2_a^+, must\rangle\rrbracket) \otimes Comp(\llbracket\langle a, \gamma, must\rangle\rrbracket)$,
using Definition 5.1;

$\Rightarrow Comp(\llbracket\langle a, \gamma 1_a^+, must\rangle\rrbracket) \subseteq Comp(\llbracket\langle a, \gamma 2_a^+, must\rangle\rrbracket) \otimes EE\text{-}Free(Ac(\gamma))$,
using the property: $Comp(\llbracket\langle a, \gamma, must\rangle\rrbracket) \subseteq EE\text{-}Free(Ac(\gamma))$;

$\Rightarrow Comp(\llbracket\langle a, \gamma 1_a^+, must\rangle\rrbracket) \subseteq Comp(\llbracket\langle a, \gamma 2_a^+, must\rangle\rrbracket) \otimes EE\text{-}Free(Ac(\gamma 1_a^+)\text{-}Ac(\gamma 2_a^+))$,
using the property: $C(\gamma 2_a^+) \cap C(\gamma) = \varnothing$;

if $\gamma = \surd$:

$$\begin{aligned} Comp(\llbracket\langle a, \gamma 1_a^+, must\rangle\rrbracket) &= Comp(\llbracket\langle a, \gamma 2_a^+, must\rangle\rrbracket) \otimes Comp(\llbracket\langle a, \surd, must\rangle\rrbracket) \\ &= Comp(\llbracket\langle a, \gamma 2_a^+, must\rangle\rrbracket). \end{aligned}$$ ■

Execution semantics of uncertainty associations

Based on the above, we observe that:

$$\begin{aligned} Comp(\llbracket\langle a, \gamma 1_a^+, must\rangle\rrbracket) &= Comp(\llbracket\langle a, \gamma 1_a, must\rangle\rrbracket) \otimes Comp(\llbracket\langle a, \gamma 1_a', must\rangle\rrbracket) \\ &\subseteq Comp(\llbracket\langle a, \gamma 1_a, must\rangle\rrbracket) \otimes EE\text{-}Free(Ac(\gamma 1_a')), \end{aligned}$$

with: $\gamma 1_a^+ = \gamma 1_a \wedge \gamma 1_a'$ and $\gamma 1_a'$ being introduced by the indirect dependency closure of $\gamma 1_a$.

This implies that $Comp(\llbracket\langle a, \gamma 1_a, must\rangle\rrbracket)$ disallows, in principle, more executions than $Comp(\llbracket\langle a, \gamma 1_a^+, must\rangle\rrbracket)$, which may raise the question whether the execution semantics of $\langle a, \gamma 1_a, must\rangle$ should be based on $\gamma 1_a$ or $\gamma 1_a^+$. Furthermore, the additional disallowed executions in $Comp(\llbracket\langle a, \gamma 1_a, must\rangle\rrbracket) - Comp(\llbracket\langle a, \gamma 1_a^+, must\rangle\rrbracket)$ are also disallowed by the execution semantics of the causality condition part of the behaviour, which considers all indirect dependencies of a behaviour. Therefore, both $\gamma 1_a$ and $\gamma 1_a^+$ can be used to define the execution semantics of $\upsilon_a(\gamma 1_a)$. An advantage of using $\gamma 1_a$ is that it makes it unnecessary to determine the indirect dependency closures of alternative causality conditions before their execution semantics can be established. A similar reasoning applies to $\upsilon_a(\gamma 2_a)$.

5.4.2 Uncertainty associations of synchronized actions

Figure 5.25 shows an example of the inconsistent definition of the uncertainty associations of two synchronized actions *a* and *b*. A shorthand notation is used in Figure 5.25, allowing

the symbol ■ to be omitted. In this shorthand notation, an uncertainty value is linked to the result action.

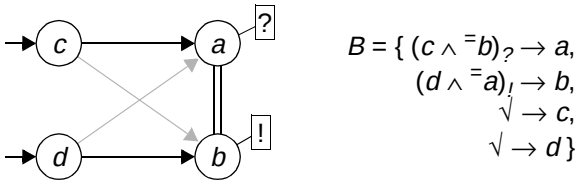


Figure 5.25: Uncertainty associations of synchronized actions

The *must*-uncertainty association $\upsilon_b(d \wedge {}^{=}a) = !$ defines that action *b* must synchronize with action *a* when action *d* has occurred. Since action *a* can only synchronize with action *b* when action *c* has occurred, the occurrence of *c* is also a condition for *b*. Consequently, alternative causality conditions $c \wedge {}^{=}b$ and $d \wedge {}^{=}a$ of actions *a* and *b* are equivalent to conditions $c \wedge d \wedge {}^{=}b$ and $c \wedge d \wedge {}^{=}a$, respectively. This implies that the alternative causality conditions of actions *a* and *b* are identical, except for the synchronization conditions.

Because the synchronization conditions define a reciprocal dependency between *a* and *b*, the uncertainty associations of $c \wedge d \wedge {}^{=}b$ and $c \wedge d \wedge {}^{=}a$ must be the same. Otherwise, one of them would define that *a* and *b* must synchronize when *c* and *d* have occurred, while the other would define that *a* and *b* may (not) synchronize when *c* and *d* have occurred, which renders an inconsistency. Consequently, the values of uncertainty associations $\upsilon_a(c \wedge {}^{=}b)$ and $\upsilon_b(d \wedge {}^{=}a)$ must be the same, i.e., $\upsilon_a(c \wedge {}^{=}b) = !$.

Consistency rule

The following property of two alternative causality conditions γ_a and γ_b which define two synchronized actions *a* and *b* in behaviour *B* normally holds:

> if $\gamma_a = {}^{=}b \wedge \gamma 1$, $\gamma_b = {}^{=}a \wedge \gamma 2$ and γ_a and γ_b can be combined in an alternative behaviour then $\gamma_a \equiv_B {}^{=}b \wedge \gamma 1 \wedge \gamma 2$ and $\gamma_b \equiv_B {}^{=}a \wedge \gamma 2 \wedge \gamma 1$.

This property follows from the identity property of the synchronization relation, which is explained in Section 5.3.3. The reciprocal dependency as defined by conditions ${}^{=}b$ and ${}^{=}a$ implies that the uncertainty associations $\upsilon_a(\gamma_a)$ and $\upsilon_b(\gamma_b)$ both define the uncertainty that *a* and *b* synchronize when condition $\gamma 1 \wedge \gamma 2$ is satisfied. A consequence of this property is that the uncertainty associations $\upsilon_a(\gamma_a)$ and $\upsilon_b(\gamma_b)$ must be the same.

Assuming that γ_a and γ_b are alternative causality conditions of actions *a* and *b*, which can be combined in an alternative behaviour, the uncertainty associations $\upsilon_a(\gamma_a)$ and $\upsilon_b(\gamma_b)$ must obey the following consistency rule:

> if $\gamma_a = {}^{=}b \wedge \gamma 1$ and $\gamma_b = {}^{=}a \wedge \gamma 2$ then $\upsilon_a(\gamma_a) = \upsilon_b(\gamma_b)$.

Formal definition

A proof of this consistency rule is given in terms of the execution model. The consequence for the execution semantics of $\upsilon_a(\gamma 1_a) = \upsilon_a(\gamma 2_a)$ is that the same executions are disallowed

by $\upsilon_a(\gamma_a)$ = ! and $\upsilon_b(\gamma_b)$ = !. This enables the following alternative definition of the consistency rule.

Property 5.15 Assuming γ_a and γ_b are alternative causality conditions of result actions *a* and *b* in behaviour *B*, which can be combined in an alternative behaviour, the following holds:

if $\gamma_a = {}^{=}b \wedge \gamma 1$ and $\gamma_b = {}^{=}a \wedge \gamma 2$ then for each $BA \in Alt(B)$, such that $\gamma_a(BA) = \gamma_a$ and $\gamma_b(BA) = \gamma_b$, the following holds:

$$Comp([\![\langle a, \gamma_a{}^{+}, must\rangle]\!]\ ) = Comp([\![\langle b, \gamma_b{}^{+}, must\rangle]\!]\ ),$$

where

- $Comp([\![\langle a, \gamma_a{}^{+}, must\rangle]\!]\ )$ and $Comp([\![\langle b, \gamma_b{}^{+}, must\rangle]\!]\ )$ represent the sets of executions disallowed by $\upsilon_a(\gamma_a)$ and $\upsilon_b(\gamma_b)$, respectively.

Proof: The proof of this property is as follows:

For each $BA \in Alt(B)$, such that $\gamma_a(BA) = \gamma_a$ and $\gamma_b(BA) = \gamma_b$, the following holds:

$\gamma_a \equiv {}^{=}b \wedge \gamma 1 \wedge \gamma 2$ and $\gamma_b \equiv {}^{=}a \wedge \gamma 2 \wedge \gamma 1$

$\Rightarrow \exists \gamma \in \boldsymbol{C} \mid \gamma_a{}^{+} = {}^{=}b \wedge \gamma$ and $\gamma_b{}^{+} = {}^{=}a \wedge \gamma$

$$\begin{aligned}\Rightarrow\ & Comp([\![\langle a, \gamma_a{}^{+}, must\rangle]\!]\ ) \\ &= Comp([\![\langle a, {}^{=}b, must\rangle]\!]\ ) \otimes Comp([\![\langle a, \gamma, must\rangle]\!]\ ) \\ &= Comp([\![\langle b, {}^{=}a, must\rangle]\!]\ ) \otimes Comp([\![\langle b, \gamma, must\rangle]\!]\ ) \\ &= Comp([\![\langle b, \gamma_b{}^{+}, must\rangle]\!]\ ),\end{aligned}$$

using the properties that $[\![\langle a, {}^{=}b, must\rangle]\!] = [\![\langle b, {}^{=}a, must\rangle]\!]$ and actions *a* and *b* are related in the same way to the actions in γ. ■

5.4.3 Uncertainty associations of mutually exclusive actions

Figure 5.26 shows an example of the inconsistent definition of the uncertainty associations of two mutually exclusive actions *a* and *b*. The example defines a choice between the occurrences of actions *a* and *b* after actions *c* and *d* have occurred. The *must*-uncertainty association $\upsilon_a(c \wedge \neg b)$ = ! defines that action *a* must occur when action *b* does not occur. This implies that $\upsilon_b(c \wedge d \wedge \neg a)$ = ? is inconsistent, since uncertainty association $\upsilon_b(c \wedge d \wedge \neg a)$ = ? defines that neither *a* nor *b* may occur after *c* and *d* have occurred. Consequently, the *must* value of $\upsilon_a(c \wedge \neg b)$ implies the *must* value of $\upsilon_b(c \wedge d \wedge \neg a)$.

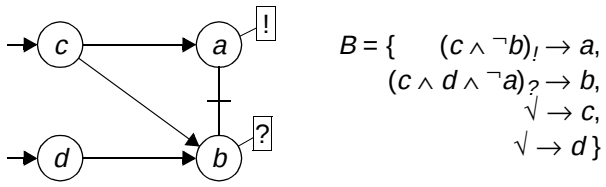


Figure 5.26: Uncertainty associations of mutually exclusive actions

Alternatively, we consider the causality relations $(c \wedge \neg b)_\mu \rightarrow a$ and $(c \wedge d \wedge \neg a)_! \rightarrow b$. In this case, both μ = ! and μ = ? are allowed, since the situations that comply to condition $c \wedge d$ in which action *a* is allowed to occur is a subset of the situations that comply to condition *c* in

which action a is allowed to occur, such that $\upsilon_b(c \wedge d \wedge \neg a)$ can not prescribe the uncertainty value of $\upsilon_a(c \wedge \neg b)$.

Figure 5.27 depicts another example of the inconsistent definition of the uncertainty associations of two mutually exclusive actions a and b. After action d has occurred, either b can occur or a and c can occur independently. The *may*-uncertainty association $\upsilon_a(d \wedge \neg b) = ?$ defines that action a may occur when action b does not occur, which can be combined with any value of $\upsilon_b(d \wedge \neg a \wedge \neg c)$. Instead, the *must*-uncertainty association $\upsilon_c(d \wedge \neg b) = !$ defines that action c must occur when action b does not occur, which can only be combined with the *must* value of $\upsilon_b(d \wedge \neg a \wedge \neg c)$. The uncertainty association $\upsilon_b(d \wedge \neg a \wedge \neg c) = ?$ is inconsistent, since it allows that neither b nor c occur. Consequently, the *must* value of $\upsilon_a(c \wedge \neg b)$ implies the *must* value of $\upsilon_b(d \wedge \neg a \wedge \neg c)$.

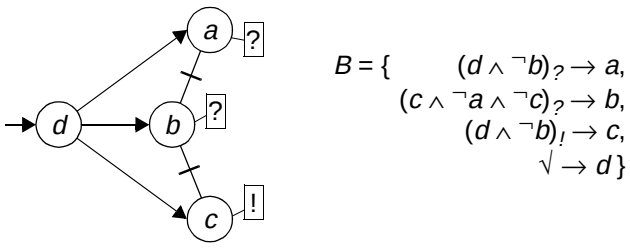


Figure 5.27: Uncertainty associations of mutually exclusive actions (2)

Observe that the *may* value of $\upsilon_a(d \wedge \neg b)$ is not inconsistent with the *must* values of $\upsilon_b(d \wedge \neg a \wedge \neg c)$ and $\upsilon_c(d \wedge \neg b)$, since execution $\boxed{d \quad \overline{b} \quad \overline{c}}$ being disallowed by $\upsilon_c(d \wedge \neg b) = !$ and execution $\boxed{d \quad \overline{a} \quad \overline{c} \quad \overline{b}}$ being disallowed by $\upsilon_b(d \wedge \neg a \wedge \neg c) = !$ do not imply that execution $\boxed{d \quad \overline{b} \quad \overline{a}}$ must be necessarily disallowed by $\upsilon_a(d \wedge \neg b)$.

Consistency rule

In general, the uncertainty associations of alternative causality conditions $\gamma_a = \neg b \wedge \gamma 1$ and $\gamma_b = \neg a \wedge \gamma 2$ define the uncertainty that at least action a or action b occurs when assuming condition $\gamma 1$ and condition $\gamma 2$ are satisfied, and γ_a and γ_b can be combined in an alternative behaviour. In this case, the *must*-uncertainty association of γ_b implies the *must*-uncertainty association of γ_a, when for each alternative behaviour BA in which γ_a and γ_b can be combined, their dependency closures satisfy the following constraint:

$\gamma_a^+ = \neg b \wedge \gamma 1'$, $\gamma_b^+ = \neg a \wedge \gamma 2'$ and $C(\gamma 2') \subseteq C(\gamma 1')$.

This can be understood by considering that whenever $\gamma 1$ is satisfied in BA, $\gamma 2$ is also satisfied. In other words, the situations that comply to $\gamma 2$ in which either a or b can occur, comprise all situations that comply to $\gamma 1$ in which either a or b can occur. This implies that when either a or b must occur when $\gamma 2$ is satisfied, at least a or b must occur when $\gamma 1$ is satisfied.

Assuming that γ_a and γ_b are alternative causality conditions of actions a and b in behaviour B, which can be combined in an alternative behaviour of B, the uncertainty associations $\upsilon_a(\gamma_a)$ and $\upsilon_b(\gamma_b)$ must obey the following consistency rule:

if $\gamma_a = \neg b \wedge \gamma 1$, $\gamma_b = \neg a \wedge \gamma 2$ and
$\forall BA \in Alt(B)$, $\gamma_a(BA) = \gamma_a$, $\gamma_b(BA) = \gamma_b \mid \gamma_a^+ = \neg b \wedge \gamma 1'$, $\gamma_b^+ = \neg a \wedge \gamma 2'$ and $C(\gamma 2') \subseteq C(\gamma 1')$
then $\upsilon_a(\gamma_a) \geq \upsilon_b(\gamma_b)$, with *must* > *may*.

The proof of this consistency rule in terms of the execution model is left as an exercise for the reader. This proof strongly resembles the proof of Property 5.14.

The application of the combination and consistency rules presented in Sections 5.3 and 5.4 is necessary, but probably not sufficient, to obtain consistent behaviour definitions, which are free from impossible actions, impossible alternative causality conditions or contradicting uncertainty associations. Further investigation of additional rules needed to guarantee consistent behaviour definitions falls outside the scope of this thesis.

5.5 Related work

This section briefly reviews some languages known from literature that support the modelling of causal dependencies (relations) between actions. In particular, we investigate to what extent the modelling of the following behaviour characteristics is supported:

- conjunctions and disjunctions of enabling and disabling conditions;
- independent actions;
- the modelling of uncertainty of action occurrences.

The modelling of synchronization conditions is not considered, since none of the reviewed languages are capable of directly expressing synchronization relations between actions.

The following languages are considered below: languages based on partially ordered sets (posets), causal and geometric automata, (dual) event structures and petri nets.

Poset-based languages

Languages based on posets support, in principle, the modelling of each of the behaviour characteristics mentioned above. The execution semantics of our design model in this (and the previous) chapter confirms this observation, since executions may be considered as posets extended with a synchronization relation.

Conjunctions of enabling conditions (often denoted as AND-causality) can be expressed directly by posets. Disjunctions of (conjunctions of) enabling conditions (often denoted as OR-causality) and conjunctions of disabling conditions (often denoted as NOT-causality) can be modelled by sets of posets. In addition, disjunctions of disabling conditions and arbitrary combinations of conjunctions and disjunctions of enabling and disabling conditions can be modelled by sets of posets. Independence is modelled by the absence of an ordering relation. Uncertainty is modelled by additional posets which are prefixes of the posets representing the behaviour executions in which all actions must occur unless they are disabled.

Examples of languages based on posets are reported by Pratt ([55]) and Gaifman ([22]), which have been reviewed in Section 3.10. Mazurkiewicz ([48]) presents a language based on traces which are isomorphic to posets.

Causal and geometric automata

Gunawardena introduces causal automata in [27, 29] and the more expressive geometric automata in [28] to support the modelling of causality.

A *causal automaton* is defined as a tuple $\langle E, \rho \rangle$, where E is a set of events and $\rho : E \rightarrow BE$ is a function to the free boolean algebra BE generated by E. We have omitted the labelling function here, since we assume an action can occur only once. Boolean algebra BE is restricted to the boolean operators $\vee$ and $\wedge$, such that $BE = \langle E, \vee, \wedge, \mathrm{F}, \mathrm{T} \rangle$. Figure 5.28 depicts some examples of causal automata. The first column represents all $e \in E$, while the second column represents $\rho(e)$.

C1 =

a	T
b	T
c	$a \wedge b$

C2 =

a	T
b	T
c	$a \vee b$

Figure 5.28: Examples of causal automata

An event $e \in E$ is enabled if the valuation of $\rho(e)$ renders the value T (true), i.e., $v(\rho(e)) = \mathrm{T}$, where v represents the valuation function. Function v is defined such that $v(e) = \mathrm{F}$ (false) for all $e \in E$. If an event e' occurs than e' is replaced by T in all $\rho(e)$, with $e \in E\text{-}\{e'\}$. For example, if a occurs in automaton C2 of Figure 5.28, then c becomes enabled, since $\rho(c) = \mathrm{T} \vee b$. Automata C1 and C2 show that causal automata support the representation of AND- and OR-causality.

A *geometric automaton* is defined as a triple $\langle E, \rho, \sigma \rangle$, where E is a set of events and $\rho, \sigma : E \rightarrow Fr(E)$ are functions from E to the free frame generated by E. The free frame $Fr(E)$ replaces the free boolean algebra BE to cope with the case of infinitely many events. This distinction is not important here. Figure 5.29 depicts an example of a geometric automaton. The three columns represent from left to right: function ρ, the events in E and function σ, respectively.

G =

T	a	F
T	b	F
$a \vee b$	c	$a \wedge b$

Figure 5.29: Example of a geometric automaton

An event $e \in E$ is enabled if $v(\rho(e)) = \mathrm{T}$ and $v(\sigma(e)) = \mathrm{F}$. Function ρ defines the actions that must have occurred and function σ defines the actions that must not have occurred to allow the occurrence of e. Consequently, geometric automata extend causal automata with the possibility to express NOT-causality. For example, automaton G of Figure 5.29 defines that c is allowed to occur after either a or b has occurred and becomes disabled after both a and b have occurred.

Geometric automata only allow the modelling of conjunctions of (disjunctions or conjunctions of) enabling conditions and (disjunctions or conjunctions of) disabling conditions. For example, the direct modelling of $a \vee (b \wedge \neg c) \rightarrow d$ is impossible using geometric automata. The modelling of independence is not properly supported, since an interleaving semantics is

defined for causal and geometric automata. Although the syntax suggests the independence of actions *a* and *b* in the above examples, both actions are forced to occur in arbitrary order. In addition, the modelling of uncertainty is not supported. Actions are assumed to occur once they are enabled, unless they are disabled by the occurrences of one or more other actions.

(Dual) event structures

In [37], Katoen introduces dual event structures, which are obtained from extended bundle event structures ([44, 43]) by dropping the stability constraint. An extended bundle event structure is defined as a four-tuple $\langle E, \rightsquigarrow, \mapsto, l\rangle$, where E is a set of events, $\rightsquigarrow \subseteq E \times E$ is the asymmetric conflict relation, $\mapsto \subseteq \wp(E) \times E$ is the bundle relation and $l : E \rightarrow \boldsymbol{A}$ is the action labelling function, with $\boldsymbol{A}$ being a set of actions. The asymmetric conflict $e_1 \rightsquigarrow e_2$ is equivalent to the causality relation $\neg e_1 \rightarrow e_2$ in our design model. A bundle relation $X \mapsto e$, with $X \in \wp(E)$, is equivalent to causality relation $e_1 \vee .. \vee e_n \rightarrow e$, with $X = \{e_1, .., e_n\}$, where a choice relation is defined between the events in X such that only one event can occur in an execution. The latter constraint is denoted as the *stability constraint*.

The stability constraint implies that (extended) bundle event structures (and stable event structures) only support a restricted form of OR-causality, which is denoted as XOR-causality (exclusive OR-causality) in [37]. By dropping the stability constraint, dual event structures support the modelling of OR-causality, although it is assumed that an event must occur due to the first possible cause that happens, which reduces the non-determinism of the OR-causality. The modelling of NOT-causality is supported by the asymmetric conflict relation. The modelling of AND-causality is supported by any type of event structure known to us.

Dual event structures only allow the modelling of conjunctions of (disjunctions of) enabling conditions and (conjunctions of) disabling conditions. This is a larger limitation than the one mentioned for geometric automata. For example, geometric automata allow the modelling of $\neg a \vee \neg b \rightarrow c$, whereas dual event structures do not. A common type of relation that is included in this limitation is the interleaving relation. Therefore, an ad-hoc extension of dual event structures is defined in [37] to support the modelling of the interleaving relation. Strictly, this limitation may be overcome (partly or completely) by allowing an action to be modelled by distinct events. However, this solution quickly becomes incomprehensible and leads to an explosion of the number of events and should be avoided.

We consider dual event structures above because they are more expressive than other known event structures, such as (extended) bundle, stable, flow or prime event structures, w.r.t. the modelling of different types of causality conditions. In any of these types of event structures, the independence of two actions is modelled by the absence of a relation between these actions. The modelling of uncertainty is however not supported, since event structures do not define whether an event must or may occur once it is enabled and it is not disabled by other events.

We also conclude that the stability constraint has been introduced to avoid the specification of behaviours containing *causal ambiguity* ([45, 43, 37]), which implies that one can not derive the causal relations between the actions of this behaviour from its possible traces.

Causal ambiguity is only a problem when one is interested in deriving the specification of a behaviour from its possible traces, which sounds like a weak motivation for adding the stability constraint and limiting designers in defining what they want to build. From a designer's point of view there is no ambiguity at all when prescribing and refining (implementing) a disjunction of causality conditions. The term causal ambiguity can be considered as misleading and surely over-emphasizes some behaviour property which is only interesting from a trace theoretical perspective.

Petri Nets

In [36, 35], Katoen presents a compositional semantics of the basic design language defined in [16, 67] in terms of labelled place/transitions nets. The considered basic design language includes the design language presented in this (and the previous) chapter, except for synchronization conditions and *must*-occurrences. Katoen showed that labelled place/transition nets can be used to model conjunctions and disjunctions of enabling and disabling conditions, independent actions and *may*-occurrences.

5.6 Conclusions

In this chapter we provide a complete architectural and formal definition of the *and*- and the *or*-operator on causality conditions introduced in Chapter 4. The *and*-operator allows the representation of a conjunction of multiple elementary causality conditions. The *or*-operator allows the representation of a causality condition consisting of multiple alternative causality conditions. An alternative causality condition is defined using only the *and*-operator. The adjective 'alternative' denotes that an action occurrence is caused by (or depends on) only one of its alternative conditions, although multiple of them can be satisfied at the same time.

The complete definitions of the *and*- and the *or*-operator allow one to model relations between multiple actions. Accordingly, the combination and consistency rules introduced in Chapter 4 are generalized towards behaviours with multiple actions.

We compared the expressive power of our basic design language defined so far with some other languages known from literature that support the modelling of causal dependencies (relations) between actions. From this comparison, we conclude that our language is highly expressive. We believe however that the most important benefit of our language is its basic design concepts, allowing one to directly represent (compositions of) elementary and essential behaviour characteristics, and the consistent and modular elaboration of these basic design concepts, based on the structuring of these characteristics in an abstraction hierarchy.

Chapter 6

Information, time and location attributes

This chapter extends causality relations with the information, time and location attributes introduced in Chapter 2. This allows one to relate information, time and location values established in different action occurrences.

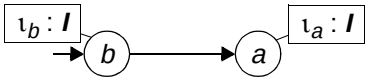
The structure of this chapter is as follows. Sections 6.1, 6.2 and 6.3 discuss the modelling of reference relations between information, location and time attributes of related actions, respectively. Section 6.4 addresses the modelling of mixed reference relations involving information, location and time attributes. Section 6.5 discusses related work. And Section 6.6 presents the conclusions.

6.1 Information references

This section extends causality relations with an information attribute. This allows one to model (i) the establishment of information values in action occurrences and (ii) dependencies between information values established in different action occurrences.

6.1.1 Information attribute

Normally action occurrences represent activities that establish some result. In order to represent such results we introduce an information attribute, extending the causality relations defined so far. Figure 6.1 illustrates this for actions *a* and *b*. The information attributes are represented in separate text-boxes in the graphical notation, and are represented between brackets to the right side of the result action in the textual notation. Symbols ι_a and ι_b represent the information values that are established in actions *a* and *b*, respectively, in an execution. Symbol ***I*** represents the universe of information values.



$$B = \{ \; b \rightarrow a\,(\iota_a : \boldsymbol{I}),\; \surd \rightarrow b\,(\iota_b : \boldsymbol{I}) \; \}$$

Figure 6.1: Information attribute

The definition of information attributes is optional in this work. In case the information attribute of some action is irrelevant at the considered abstraction level, it is omitted.

The possible values that can be established in the information attribute of a result action *a* are determined by three types of constraints:

- the *information value domain*, which defines the information values that are allowed by result action *a* itself;
- *information reference relations*, which define how the information value of *a* depends on the information values of the actions in the causality condition of *a*;
- *information causality conditions*, which define how the occurrence of *a* depends on the information values of the actions in the causality condition of *a*.

Information value domains

Often, an activity establishes a specific type of result. For example, a car assembly line produces cars, an insurance company sells insurances, and a communication service delivers data units at remote network access points. This implies that an action only allows a certain range of information values to be established. The restriction of the universe of information values to this particular range is called the *information value domain* of a result action (see Section 2.4.2). Figure 6.2 represents the information value domains of result actions *a* and *b*, which define that *a* and *b* are only allowed to establish a natural number from the ranges [3..7] and [0..9], respectively.

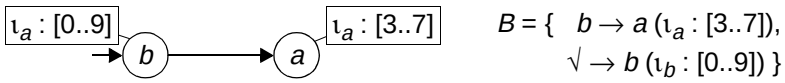


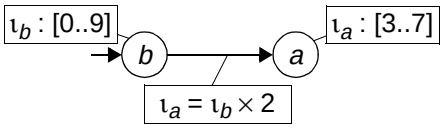
Figure 6.2: Information value domains

Information reference relations

The information value established by some action occurrence *a* may depend on the information values established in other action occurrences. In this case we say that action occurrence *a refers* to the information values of other action occurrences.

We define that action occurrence *a* may refer to the information value of another action occurrence *b*, when *a* depends on *b* and *a* occurs after *b*, i.e., $\tau_b < \tau_a$. The motivation for this definition is that an information value reference from one action occurrence to another implies a dependency (relation) between these action occurrences. Furthermore, at first, we consider referring in its common meaning as referring to something that has been established in the past.

Figure 6.3 depicts an example behaviour in which action *a* refers to the information value of action *b*. The information value of *a* depends on the information value established in *b* such that the value of ι_a must be equal to two times the value of ι_b. The relation between ι_a and ι_b is called an *information reference relation*. The conjunction of this information reference relation with the information value domains of *a* and *b* allows one of two possible values to be established in *a*: (i) $\iota_a = 4$ in case $\iota_b = 2$ and (ii) $\iota_a = 6$ in case $\iota_b = 3$. In case $\iota_b \in [0..9]-\{2, 3\}$, action *a* can not occur, since there is no value for ι_a that satisfies reference relation $\iota_a = \iota_b \times 2$ and lies within the information value domain [3..7].



$B = \{\ b \rightarrow a\ (\iota_a : [3..7])\ [\iota_a = \iota_b \times 2],$
$\surd \rightarrow b\ (\iota_b : [0..9])\ \}$

Figure 6.3: Information reference relations

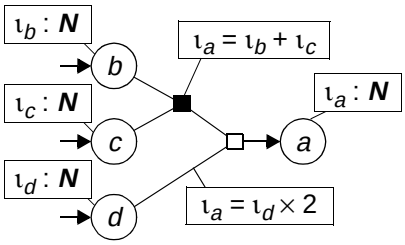
Information reference relation $\iota_a = \iota_b \times 2$ is represented within a text-box linked to the enabling relation between *b* and *a* in the graphical notation, since this constraint can be considered as a characteristic of this enabling relation. Alternatively, we allow this relation to be represented within the text-box linked to the referring action (here *a*).

In the textual notation, information reference relations are represented as constraints between square brackets to the right side of the information attribute of the result action. In the same way, we allow information value domains to be represented as constraints. For example, an alternative textual representation of the causality relation of action *a* is:

$b \rightarrow a\ (\iota_a : \boldsymbol{I})\ [\iota_a = \iota_b \times 2,\ \iota_a \in [3..7]]$.

An information reference relation defines how the information value of the result action depends on the information values of its enabling actions in an execution. This implies that a reference relation must be associated with each alternative causality condition of a result action.

Figure 6.4 shows an example that contains a result action *a*, which has two alternative causality conditions: $b \wedge c$ and *d*. Clauses "if $b \wedge c \rightarrow a$ then $\iota_a = \iota_b + \iota_c$" and "if $d \rightarrow a$ then $\iota_a = \iota_d \times 2$" represent that the value of ι_a is constrained by reference relations $\iota_a = \iota_b + \iota_c$ and $\iota_a = \iota_d \times 2$ in case the occurrence of *a* is caused by the satisfaction of alternative conditions $b \wedge c$ and *d*, respectively. Symbol ***N*** represents the domain of natural numbers.



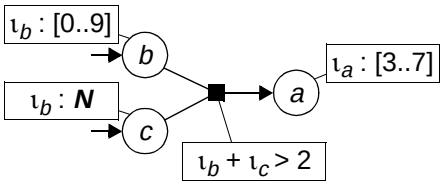
$B = \{\ (b \wedge c) \vee d \rightarrow a\ (\iota_a : \boldsymbol{N})$
$[\text{if } b \wedge c \rightarrow a \text{ then } \iota_a = \iota_b + \iota_c,$
$\text{if } d \rightarrow a \text{ then } \iota_a = \iota_d \times 2],$
$\surd \rightarrow b\ (\iota_b : \boldsymbol{N}),$
$\surd \rightarrow c\ (\iota_c : \boldsymbol{N}),$
$\surd \rightarrow d\ (\iota_d : \boldsymbol{N})\ \}$

Figure 6.4: Information reference relations (2)

Information causality conditions

An alternative causality condition can be extended to define conditions on the information values of its enabling actions. These conditions are called *information causality conditions*, and must be satisfied in order to allow the occurrence of the result action.

Figure 6.5 depicts an example behaviour in which information causality condition $\iota_b + \iota_c > 2$ represents that action *a* is only allowed to occur when the sum of the information values established in *b* and *c* is larger than 2. For example, in case $\iota_b = 0$ and $\iota_c = 1$, action *a* is not allowed to occur.



$$B = \{\ b \wedge c\ [\iota_b + \iota_c > 2] \rightarrow a\ (\iota_a : [3..7]),\ \surd \rightarrow b\ (\iota_b : [0..9]),\ \surd \rightarrow c\ (\iota_c : \boldsymbol{N})\ \}$$

Figure 6.5: Information causality conditions

Information causality condition $\iota_b + \iota_c > 2$ is represented within a text-box linked to the enabling relation between *b*, *c* and *a* in the graphical notation, since this condition can be considered a characteristic of this enabling relation. Alternatively, we allow this condition to be represented within the text-box linked to the result action (here *a*).

In the textual notation, information causality conditions are represented as constraints between square brackets to the right side of the corresponding alternative causality condition. Alternatively, we allow this condition to be represented between square brackets to the right side of the result action. For example, an alternative textual representation of the causality relation of action *a* in Figure 6.5 is:

$b \wedge c \rightarrow a\ (\iota_a : \boldsymbol{I})\ [\iota_b + \iota_c > 2,\ \iota_a \in [3..7]]$.

An essential difference between an information causality condition and an information reference relation is that the former only prescribes the required information values of enabling actions, whereas the latter prescribes the dependency of the information value of the result action on the information values of enabling actions. Therefore, in the textual notation it is advisable to place each information causality condition beside its corresponding causality condition and each information reference relation beside its result action.

6.1.2 Combination of information and uncertainty attributes

Information attribute constraints can be interpreted independently of uncertainty attribute constraints. However, uncertainty attribute constraints can *not* be interpreted independently of information attribute constraints.

For example, assume alternative condition $\gamma_a = b \wedge c$ in Figure 6.5 is associated with the *must* uncertainty value, i.e., $\upsilon_a(b \wedge c) = !$. The original interpretation that *a* must occur when *b* and *c* have occurred does not hold here, since *b* and *c* may establish information values which do not satisfy the information attribute constraints of *a*. In this case no information value can be established in *a*, and *a* does not occur. Consequently, the definition of the *must* uncertainty association $\upsilon_a(b \wedge c) = !$ should be extended to define that *a* must occur when *b* and *c* have occurred and the information attribute constraints of *a* are satisfied.

In general, the condition for the occurrence of some result action *a* consists of the conjunction of its causality condition and its information attribute constraints. Consequently, the interpretation of uncertainty association $\pi_a(\gamma_a) = \mathit{must}$ is defined as follows:

$\pi_a(\gamma_a) = \mathit{must}$ defines that action *a* must occur when

(i) alternative causality condition γ_a is satisfied; and

(ii) the information attribute constraints of a are satisfied, consisting of the information value domain of a, the information reference relations associated with γ_a and the information causality conditions associated with γ_a.

6.1.3 Pre-formal notation

For the sake of conciseness, we only represent information attribute constraints in an intuitive way in our design notation $L_{causality}$ (see Section 4.1.3), without describing its syntax precisely. Instead, notational elements and syntax rules for the representation of information attribute constraints in the pre-formal notation $L'_{causality}$ are introduced below. This enables and facilitates the definition of the execution semantics of information value domains, information reference relations and information causality conditions. We assume our design notation is sufficiently precise to leave function $[\![\ ]\!]$ undefined.

Information value domain

The information value domain of an action is defined as part of the action specification of this action (see Chapter 2, Definition 2.1). When abstracting from the location and time attributes, an action specification is defined as a tuple $\langle a, \mathrm{I}_a\rangle$, where a identifies the action and I_a represents the information value domain of a. For example, in case of the behaviour of Figure 6.5, the specifications of actions a, b and c are: $\langle a, [3..7]\rangle$, $\langle b, [0..9]\rangle$ and $\langle c, \boldsymbol{N}\rangle$, respectively.

Information reference relations

The information reference relation associated with alternative causality condition γ_a of result action a is denoted as $I\text{-}Ref_a(\gamma_a)$, or alternatively as $I\text{-}Ref(a, \gamma_a)$. Reference relation $I\text{-}Ref_a(\gamma_a)$ is defined by the set of all possible information references between the information value of a and the information values of the referred enabling actions in γ_a. Such an *information reference* defines a possible information value that can be established by a for a possible combination of information values that can be established by the referred enabling actions in γ_a. Consequently, the set of information references defined by $I\text{-}Ref_a(\gamma_a)$ represents the possible information values that can be established by a for all possible combinations of information values that can be established by the referred enabling actions in γ_a. An information reference is represented by a set of tuples, one tuple $\langle b, \iota_b\rangle$ for each referred enabling action b in γ_a and one tuple $\langle a, \iota_a\rangle$ for result action a. We assume $I\text{-}Ref_a(\gamma_a)$ only refers to enabling actions in γ_a for which the information attribute is defined.

For example, in case of the example behaviour in Figure 6.4, the information reference relation associated with alternative causality condition $b \wedge c$ of result action a is defined as:

$$I\text{-}Ref_a(b \wedge c) = \{\ \{\langle b, \iota_b\rangle, \langle c, \iota_c\rangle, \langle a, \iota_a\rangle\} \mid \iota_a = \iota_b + \iota_c,\ \ \iota_a, \iota_b, \iota_c \in \boldsymbol{I}\}.$$

The information references in $I\text{-}Ref_a(b \wedge c)$ represent the possible information values that can be established by a for all possible combinations of information values that can be established by actions b and c, when assuming $b \wedge c$ is the resulting causality condition of a.

Depending on the specific information reference relation, multiple alternative information values of a may be acceptable for the same combination of information values of the referred enabling actions in γ_a. This is represented by distinct information references, one for each possible information value of a.

For example, consider the following information reference relation:

$$I\text{-}Ref_a(b \wedge c) = \{ \{\langle b, \iota_b\rangle, \langle c, \iota_c\rangle, \langle a, \iota_a\rangle\} \mid \iota_a > \iota_b + \iota_c,\ \iota_a, \iota_b, \iota_c \in \boldsymbol{I}\}.$$

In case $\iota_b = 2$ and $\iota_c = 3$, any information value ι_a larger than 5 may be established. This is represented by the infinite set of information references $\{\{\langle b, 2\rangle, \langle c, 3\rangle, \langle a, 6\rangle\}, \{\langle b, 2\rangle, \langle c, 3\rangle, \langle a, 7\rangle\}, \{\langle b, 2\rangle, \langle c, 3\rangle, \langle a, 8\rangle\}, ...\}$, which is just a subset of $I\text{-}Ref_a(b \wedge c)$.

An information reference relation $I\text{-}Ref_a(\gamma_a)$ is equal to the empty set in case no information value can be established by a for any combination of information values of the referred enabling actions in γ_a.

An information reference relation $I\text{-}Ref_a(\gamma_a)$ is equal to $\{\{\langle a, \iota_a\rangle\} \mid \iota_a \in \boldsymbol{I}\}$, in case the reference relation for condition γ_a is undefined in the behaviour definition. In this case, we assume no constraint is imposed on the information value of a.

The set of information reference relations of result action a, consisting of one information reference relation for each alternative causality condition of a, is denoted as $I\text{-}Refs_a(\Gamma_a)$, or alternatively as $I\text{-}Refs(a, \Gamma_a)$.

Information causality conditions

The information causality condition associated with alternative causality condition γ_a of result action a is denoted as $I\text{-}Cau_a(\gamma_a)$, or alternatively as $I\text{-}Cau(a, \gamma_a)$. Information causality condition $I\text{-}Cau_a(\gamma_a)$ is defined by the set of all possible combinations of information values of the enabling actions in γ_a that allow the occurrence of a. Such a combination is defined by a set of tuples, one tuple $\langle b, \iota_b\rangle$ for each referred enabling action b in γ_a. We assume $I\text{-}Cau_a(\gamma_a)$ only refers to enabling actions in γ_a for which the information attribute is defined.

For example, in case of the example behaviour in Figure 6.5, the information causality condition associated with alternative causality condition $b \wedge c$ of result action a is defined as:

$$I\text{-}Cau_a(b \wedge c) = \{ \{\langle b, \iota_b\rangle, \langle c, \iota_c\rangle\} \mid \iota_b + \iota_c > 2,\ \iota_b, \iota_c \in \boldsymbol{I}\}.$$

An information causality condition $I\text{-}Cau_a(\gamma_a)$ is equal to $\{\{\langle b1, \iota_{b1}\rangle, .., \langle bn, \iota_{bn}\rangle\} \mid \iota_{b1} \in \boldsymbol{I}, .., \iota_{bn} \in \boldsymbol{I}\}$, with $b1, .., bn$ representing the referred enabling actions in γ_a, in case the information causality condition for γ_a is undefined in the behaviour definition. In this case, we assume that any combination of information values of the enabling actions in γ_a for which the information attribute is defined allows the occurrence of result action a.

The set of information causality conditions of result action a, consisting of one information causality condition for each alternative causality condition of a, is denoted as $I\text{-}Caus_a(\Gamma_a)$, or alternatively as $I\text{-}Caus(a, \Gamma_a)$.

Causality relations

The pre-formal notation of causality relations extended with an information attribute is defined below.

Definition 6.1 The *causality relation* of an action a is defined as a five-tuple $\langle\langle a, \mathrm{I}_a\rangle, \Gamma, \upsilon, \textit{I-Refs}, \textit{I-Caus}\rangle$, where

- $\langle a, \mathrm{I}_a\rangle$, is the specification of result action a, where
  - $a \in \boldsymbol{A}$, identifies result action a; and
  - I_a is the information value domain of a;
- $\Gamma \in \boldsymbol{CC}$, is the causality condition of a;
- $\upsilon : \{a\} \times \Gamma \rightarrow \boldsymbol{U}$, is the uncertainty attribute of a;
- $\textit{I-Refs} : \{a\} \times \{\Gamma\} \rightarrow \wp(\wp(\wp(\boldsymbol{A} \times \boldsymbol{I})))$, defines the information reference relations of a,

  with: $\textit{I-Refs}(a, \Gamma) = \{\textit{I-Ref}(a, \gamma) \mid \gamma \in \Gamma\}$, where

  $\textit{I-Ref} : \{a\} \times \Gamma \rightarrow \wp(\wp(\boldsymbol{A} \times \boldsymbol{I}))$, defines the information reference relation associated with some $\gamma \in \Gamma$. The following consistency rules must hold for $\textit{I-Ref}(a, \gamma)$:
  - $\forall \textit{i-ref} \in \textit{I-Ref}(a, \gamma) \mid Ac(\textit{i-ref}) = Ac_{ref}(\gamma)$, which demands that each information reference *i-ref* in $\textit{I-Ref}(a, \gamma)$ defines a combination of information values of the actions in $Ac_{ref}(\gamma)$, where
    - $Ac(\textit{i-ref}) = \{b \mid \langle b, \iota_b\rangle \in \textit{i-ref}\}$; and
    - $Ac_{ref}(\gamma)$ consists of the enabling actions in γ that are referred to by a, and a itself;
  - $\forall \textit{i-ref} \in \textit{I-Ref}(a, \gamma) \mid \forall \langle b, \iota_b\rangle, \langle c, \iota_c\rangle \in \textit{i-ref} \mid b = c \Rightarrow \iota_b = \iota_c$, which demands that each information reference *i-ref* in $\textit{I-Ref}(a, \gamma)$ defines one and only one information value for any action in $Ac(\textit{i-ref})$;
- $\textit{I-Caus} : \{a\} \times \{\Gamma\} \rightarrow \wp(\wp(\wp(\boldsymbol{A} \times \boldsymbol{I})))$, defines the information causality conditions of a,

  with: $\textit{I-Caus}(a, \Gamma) = \{\textit{I-Cau}(a, \gamma) \mid \gamma \in \Gamma\}$, where

  $\textit{I-Cau} : \{a\} \times \Gamma \rightarrow \wp(\wp(\boldsymbol{A} \times \boldsymbol{I}))$, defines the information causality condition associated with some $\gamma \in \Gamma$. The following consistency rules must hold for $\textit{I-Cau}(a, \gamma)$:
  - $\forall \textit{i-cau} \in \textit{I-Cau}(a, \gamma) \mid Ac(\textit{i-cau}) = Ac_{cau}(\gamma)$, which demands that each information condition *i-cau* in $\textit{I-Cau}(a, \gamma)$ defines a combination of information values of the actions in $Ac_{cau}(\gamma)$, where
    - $Ac(\textit{i-cau}) = \{b \mid \langle b, \iota_b\rangle \in \textit{i-cau}\}$; and
    - $Ac_{cau}(\gamma)$ consists of the enabling actions in γ that are referred to by a;
  - $\forall \textit{i-cau} \in \textit{I-Cau}(a, \gamma) \mid \forall \langle b, \iota_b\rangle, \langle c, \iota_b\rangle \in \textit{i-cau} \mid b = c \Rightarrow \iota_b = \iota_c)$, which demands that each information condition *i-cau* in $\textit{I-Cau}(a, \gamma)$ defines one and only one information value for any action in $Ac(\textit{i-cau})$. ■

Behaviours

The extension of causality relations to support information attributes adds an *information attribute part* to behaviour definitions. Therefore, a behaviour definition $B(Ac)$ that models information attributes consists of:

- a causality condition part $B_\Gamma(Ac) = \{\langle a, \Gamma_a, \upsilon_a'\rangle \mid a \in Ac\}$,
  with: $\forall\gamma_a \in \Gamma_a \mid \upsilon_a'(\gamma_a) = may$;
- an uncertainty attribute part $B_\upsilon(Ac) = \{\upsilon_a \mid a \in Ac\}$; and
- an information attribute part $B_\iota(Ac) = \{\langle\langle a, \mathrm{I}_a\rangle, I\text{-}Refs_a, I\text{-}Caus_a\rangle \mid a \in Ac\}$.

6.1.4 Formal definition

The definition of the execution semantics of causality relations that support information attributes, is divided into three parts:

1. the definition of the execution semantics of $\langle a, \Gamma_a, \upsilon_a\rangle$, which is explained in Chapters 4 and 5;
2. the definition of the execution semantics resulting from the combination of the information attribute constraints of action a, i.e., $\langle \mathrm{I}_a, I\text{-}Refs_a, I\text{-}Caus_a\rangle$. These constraints define the possible information references between a and its enabling actions;
3. the integration of the execution semantics of $\langle a, \Gamma_a, \upsilon_a\rangle$ and $\langle \mathrm{I}_a, I\text{-}Refs_a, I\text{-}Caus_a\rangle$, which is performed in three steps:
   (i) expansion of the executions allowed by $\langle a, \gamma_a\rangle$, for any $\gamma_a \in \Gamma_a$, with the information references allowed by $\langle \mathrm{I}_a, I\text{-}Refs_a, I\text{-}Caus_a\rangle$;
   (ii) expansion of the executions disallowed by $\langle a, \gamma_a, \upsilon_a(\gamma_a)\rangle$, for any $\gamma_a \in \Gamma_a$, with all possible combinations of information values of the enabling actions in γ_a for which an information value can be established in a;
   (iii) cross-conjunction of the expanded executions sets determined in (i) and (ii).

Information attribute constraints

The conjunction of the constraints imposed by the information value domain I_a, information reference relations $I\text{-}Refs_a$ and information causality conditions $I\text{-}Caus_a$ on the information attribute of result action a is called the *information attribute constraint* of a, and is denoted as $I\text{-}Atts_a(\Gamma_a)$, or alternatively as $I\text{-}Atts(a, \Gamma_a)$. Information attribute constraint $I\text{-}Atts(a, \Gamma_a)$ is decomposed into alternative constraints that are associated with each alternative causality condition γ_a in Γ_a, such that: $I\text{-}Atts(a, \Gamma_a) = \{I\text{-}Att(a, \gamma_a) \mid \gamma_a \in \Gamma_a\}$.

Constraint $I\text{-}Att(a, \gamma_a)$ defines the possible information references between action a and one or more of its enabling actions in γ_a. These references are allowed by the causality relation of a, when assuming that γ_a is the resulting causality condition of a. $I\text{-}Att(a, \gamma_a)$ is obtained from $I\text{-}Ref(a, \gamma_a)$ by eliminating all information references with values for ι_a that are not in I_a, and by eliminating all information references with combinations of information values of enabling actions that are not allowed by $I\text{-}Cau(a, \gamma_a)$. This is defined below.

Definition 6.2 Given $\langle\langle a, \mathrm{I}\rangle, \Gamma, \upsilon, \textit{I-Refs}, \textit{I-Caus}\rangle$, the information attribute constraint of result action a is defined as:

$$\textit{I-Atts}(a, \Gamma_a) = \{\textit{I-Att}(a, \gamma_a) \mid \gamma_a \in \Gamma_a\} ;$$

$$\begin{aligned}\textit{I-Att}(a, \gamma_a) = \{\textit{i-ref} \mid\ & \textit{i-ref} \in \textit{I-Ref}(a, \gamma_a),\\ & \langle a, \iota_a\rangle \in \textit{i-ref} \Rightarrow \iota_a \in \mathrm{I}_a\\ & \qquad\qquad \wedge\\ & \qquad\qquad \exists \textit{i-cau} \in \textit{I-Cau}(a, \gamma_a) \mid (\textit{i-ref} - \{\langle a, \iota_a\rangle\}) \subseteq \textit{i-cau}\}. \blacksquare\end{aligned}$$

Consider the following causality relation of action a as an example:

$$(b \wedge c)\ [\iota_b + \iota_c > 2] \rightarrow a\ (\iota_a : [3..7])\ [\iota_a = \iota_b + \iota_c\ ].$$

Supposing that the information attributes of enabling actions b and c are defined, the information attribute constraint $\textit{I-Att}(a, b \wedge c)$ can be determined based on Definition 6.2 as follows:

$$\textit{I-Att}(a, b \wedge c) = \{\ \{\langle b, \iota_b\rangle, \langle c, \iota_c\rangle, \langle a, \iota_a\rangle\} \mid \iota_a = \iota_b + \iota_c, \iota_b + \iota_c > 2, \iota_a \in [3..7]\}$$

with: $\textit{I-Ref}(a, b \wedge c) = \{\ \{\langle b, \iota_b\rangle, \langle c, \iota_c\rangle, \langle a, \iota_a\rangle\} \mid \iota_a = \iota_b + \iota_c\ \}$,
$\textit{I-Cau}(a, b \wedge c) = \{\ \{\langle b, \iota_b\rangle, \langle c, \iota_c\rangle\} \mid \iota_b + \iota_c > 2\ \}$,
$\mathrm{I}_a = [3..7]$.

Information attribute expansion of executions allowed by $\langle a, \gamma_a\rangle$

The information references defined by $\textit{I-Att}(a, \gamma_a)$ may be considered as alternative constraints on the combinations of information values that can be established by result action a and the enabling actions that are referred to by a. These constraints should be added to the constraints represented by the executions allowed by $\langle a, \gamma_a\rangle$. However, these constraints have to be added only to the executions in $[\![\langle a, \{\gamma_a\}\rangle]\!]\ (B)$ which contain the occurrence of a and, thus, the occurrences of its enabling actions. No information values have to be defined for other action occurrences or action occurrences in executions in which a does not occur, since the absence of an information value represents that no constraint on this value is imposed.

We assume EE_a represents the executions in $[\![\langle a, \{\gamma_a\}\rangle]\!]\ (B)$ in which a occurs, i.e., $EE_a = \{e\chi \mid e\chi \in [\![\langle a, \{\gamma_a\}\rangle]\!]\ (B), a \in A(e\chi)\}$. Each execution $e\chi_1$ in EE_a must be expanded to a set of executions EE_{exp}, one per information reference in $\textit{I-Att}(a, \gamma_a)$, such that each $e\chi_2$ in EE_{exp} is composed of execution $e\chi_1$ and an information reference *i-ref* from $\textit{I-Att}(a, \gamma_a)$. Information reference *i-ref* defines the information values of the occurrences of action a and the enabling actions referred to by a in execution $e\chi_2$.

The addition of an information reference to an execution is represented by the information attribute addition operator $+_\iota$. This operator is defined below.

Definition 6.3 The information attribute addition operator $+_\iota : \boldsymbol{XX} \times \wp(\boldsymbol{A} \times \boldsymbol{I}) \rightarrow \boldsymbol{XX}_\iota$ is defined as:

$$\begin{aligned}\langle A, \overline{A}, <, =, |\rangle +_\iota \textit{i-ref} &= \langle A, \overline{A}, <, =, |, \textit{i-ref}\rangle && \text{if } Ac(\textit{i-ref}) \subseteq A;\\ &= \text{undefined} && \text{otherwise.} \qquad \blacksquare\end{aligned}$$

The information attribute expansion operator $\oplus_\iota$ represents the expansion of a subset of the executions in some execution set *EE*, with the information references in *I-Att*(a, γ_a). This subset is denoted by action set Ac_{ref}, which defines that only those executions in *EE* in which actions Ac_{ref} occur are expanded. Typically, Ac_{ref} consists of result action a and the referred enabling actions in γ_a. The use of parameter Ac_{ref} allows the expansion operator to be applied to any execution set, without restricting this set first to all executions which contain the action occurrences in Ac_{ref}. Expansion operator $\oplus_\iota$ is defined below.

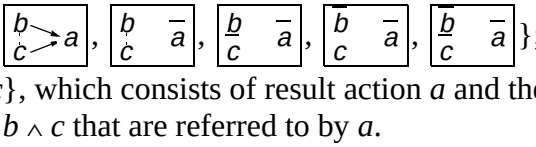
Definition 6.4 The information attribute expansion operator $\oplus_\iota : \wp(\boldsymbol{XX}) \times (\wp(\boldsymbol{A}) \times \wp(\wp(\boldsymbol{A} \times \boldsymbol{I}))) \rightarrow \wp(\boldsymbol{XX}_\iota)$ is defined as:

$$
\begin{aligned}
EE \oplus_\iota \langle Ac_{ref}, \textit{I-Ref}\rangle &= \{e\chi +_\iota \textit{i-ref} \mid e\chi \in EE, Ac_{ref} \subseteq A(e\chi), \textit{i-ref} \in \textit{I-Ref}\} \\
&\quad \cup \{e\chi \mid e\chi \in EE, Ac_{ref} \not\subseteq A(e\chi)\} && \text{if } Ac_{ref} \neq \emptyset; \\
&= EE && \text{otherwise,}
\end{aligned}
$$

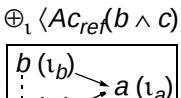
where $EE \in \wp(\boldsymbol{XX})$, $\textit{I-Ref} \in \wp(\wp(\boldsymbol{A} \times \boldsymbol{I}))$ and Ac_{ref} represents the action occurrences that have an information value. ■

Figure 6.6 illustrates the application of Definitions 6.3 and 6.4 to the causality relation of action a:

$(b \wedge c)\ [\iota_b + \iota_c > 2] \rightarrow a\ (\iota_a : [3..7])\ [\iota_a = \iota_b + \iota_c]$, with:

- $EE = [\![\langle a, b \wedge c\rangle]\!] = \{$  $\}$; and
- $Ac_{ref}(b \wedge c) = \{a, b, c\}$, which consists of result action a and the enabling actions in alternative condition $b \wedge c$ that are referred to by a.

$EE \oplus_\iota \langle Ac_{ref}(b \wedge c), \textit{I-Att}(a, b \wedge c)\rangle =$

$\{$  $\mid \iota_a = \iota_b + \iota_c, \iota_b + \iota_c > 2, \iota_a \in [3..7]\} \cup$

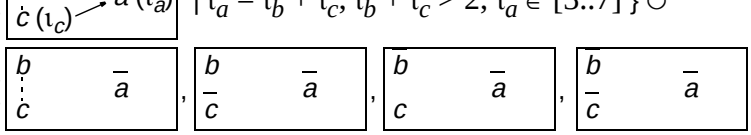
$\{$  $\}$

Figure 6.6: Information attribute expansion

Information attribute expansion of executions disallowed by $\langle a, \gamma_a, \upsilon_a(\gamma_a)\rangle$

So far the uncertainty association $\langle a, \gamma_a, \upsilon_a(\gamma_a)\rangle$ disallows the executions defined by execution set *Comp*($[\![\langle a, \gamma_a, \textit{must}\rangle]\!]$). In case $\pi_a(\gamma_a) = \textit{may}$, this set is empty. In case $\pi_a(\gamma_a) = \textit{must}$, this set consists of a single execution.

With the information attribute, we have to consider that the condition for the occurrence of a consists of the conjunction of alternative causality condition γ_a and information attribute constraint *I-Att*(a, γ_a), as explained in Section 6.1.2. This implies that the execution defined by *Comp*($[\![\langle a, \gamma_a, \textit{must}\rangle]\!]$) is only disallowed in case information attribute constraint *I-Att*(a, γ_a) can be satisfied in this execution, i.e., in case *I-Att*(a, γ_a) allows an information value to be established for a, given the combination of information values established in the ena-

bling actions in γ_a that are referred to by a. Therefore, the set of disallowed executions $Comp(\llbracket\langle a, \gamma_a, must\rangle\rrbracket)$ is expanded with $\langle Ac_{cau}(\gamma_a), I\text{-}Att^{-}(a, \gamma_a)\rangle$, where

- $Ac_{cau}(\gamma_a)$ represents the enabling actions in γ that are referred to by a;
- $I\text{-}Att^{-}(a, \gamma_a)$ is obtained from $I\text{-}Att(a, \gamma_a)$ by removing the information value of a from the information references in $I\text{-}Att(a, \gamma_a)$:

$$I\text{-}Att^{-}(a, \gamma_a) = \{i\text{-}ref' \mid i\text{-}ref' = \{\langle b, \iota_b\rangle \mid \langle b, \iota_b\rangle \in i\text{-}ref, b \neq a\}, i\text{-}ref \in I\text{-}Att(a, \gamma_a)\}.$$

Expansion $Comp(\llbracket\langle a, \gamma_a, must\rangle\rrbracket) \oplus_\iota \langle Ac_{cau}(\gamma_a), I\text{-}Att^{-}(a, \gamma_a)\rangle$ defines the executions in which action a can occur due to the satisfaction of γ_a and $I\text{-}Att(a, \gamma_a)$, while a does not occur. Consequently, this expansion represents the executions that are disallowed by $\langle a, \gamma_a, must\rangle$ and $I\text{-}Att(a, \gamma_a)$.

For any value of $\upsilon_a(\gamma_a)$, the executions disallowed by uncertainty association $\langle a, \gamma_a, \upsilon_a(\gamma_a)\rangle$ and information attribute constraint $I\text{-}Att(a, \gamma_a)$ are defined by:

$$EE_{dis} = Comp(\llbracket\langle a, \gamma_a, \upsilon_a(\gamma_a)\rangle\rrbracket) \oplus_\iota \langle Ac_{cau}(\gamma_a), I\text{-}Att^{-}(a, \gamma_a)\rangle.$$

EE_{dis} is empty in case $\upsilon_a(\gamma_a) = may$, since $Comp(\llbracket\langle a, \gamma_a, \upsilon_a(\gamma_a)\rangle\rrbracket)$ is empty in this case.

In order to derive the allowed executions from EE_{dis}, function $Comp$ has to be extended to handle executions with information values. This extension is straightforward and is denoted as $Comp_\iota$, but is not presented here for brevity. This implies that the execution semantics of the combination $\langle a, \gamma_a, \upsilon_a(\gamma_a)\rangle$ and $I\text{-}Att(a, \gamma_a)$ is defined as:

$$Comp_\iota(Comp(\llbracket\langle a, \gamma_a, \upsilon_a(\gamma_a)\rangle\rrbracket) \oplus_\iota \langle Ac_{cau}(\gamma_a), I\text{-}Att^{-}(a, \gamma_a)\rangle).$$

For example, assume the causality relation of action a in the previous example is extended with uncertainty association $\upsilon_a(b \wedge c) = must$, i.e.:

$$(b \wedge c)\ [\iota_b + \iota_c > 2] \rightarrow a\ (\iota_a : [3..7])\ [\iota_a = \iota_b + \iota_c, \upsilon_a(b \wedge c) = must].$$

Figure 6.7 depicts the executions disallowed by the combination of uncertainty association $\upsilon_a(b \wedge c)$ and information attribute constraint $I\text{-}Att(a, b \wedge c)$, with:

- $EE_{dis} = \{\boxed{\begin{matrix} b & \bar{a} \\ \vdots & \\ c & \end{matrix}}\}$;
- $Ac_{cau}(b \wedge c) = \{b, c\}$; and
- $I\text{-}Att^{-}(a, b \wedge c) = \{\ \{\langle b, \iota_b\rangle, \langle c, \iota_c\rangle\} \mid \iota_b + \iota_c > 2, \iota_b + \iota_c \in [3..7]\}$.

$$EE_{dis} \oplus_\iota \langle Ac_{cau}(b \wedge c), I\text{-}Att^{-}(a, b \wedge c)\rangle = \{\boxed{\begin{matrix} b\ (\iota_b) & \bar{a} \\ \vdots & \\ c\ (\iota_c) & \end{matrix}} \mid \iota_b + \iota_c > 2, \iota_b + \iota_c \in [3..7]\}$$

Figure 6.7: Information attribute expansion (2)

Consequently, the execution defined in EE_{dis} is only allowed when the information values of b and c do not satisfy the constraints $\iota_b + \iota_c > 2$ and $\iota_b + \iota_c \in [3..7]$. This is the case when the following holds: $\iota_b + \iota_c \leq 2$ or $\iota_b + \iota_c \notin [3..7]$, which can be simplified to: $\iota_b + \iota_c \notin [3..7]$.

Execution semantics

The execution semantics of causality relations that support information attributes is defined below, using the definitions above.

Definition 6.5 The execution semantics of causality relation $\langle\langle a, \mathrm{I}\rangle, \Gamma, \upsilon, \textit{I-Refs}, \textit{I-Caus}\rangle$ is defined as:

$$\begin{aligned}
&[\![\langle\langle a, \mathrm{I}\rangle, \Gamma, \upsilon, \textit{I-Refs}, \textit{I-Caus}\rangle]\!]\ (B)\\
&= \cup\{[\![\langle a, \{\gamma\}\rangle]\!]\ (B) \oplus_{\iota} \langle Ac_{ref}(\gamma), \textit{I-Att}(a, \gamma)\rangle \mid \gamma \in \Gamma\}\\
&\quad \otimes \textit{Comp}_{\iota}(\ \cup\{EE_{dis}(\varphi, \Gamma_a) \mid \varphi \in \upsilon\}\ ),
\end{aligned}$$

$$\begin{aligned}
&EE_{dis}(\langle a, \gamma, \upsilon(\gamma)\rangle, \Gamma_a)\\
&= (\textit{Comp}([\![\langle a, \gamma, \upsilon(\gamma)\rangle]\!]\ ) \oplus_{\iota} \langle Ac_{cau}(\gamma), \textit{I-Att}^{-}(a, \gamma)\rangle) \otimes \textit{EE-Free}(Ac_{dif}), && \text{if } Ac_{dif} \neq \varnothing;\\
&= \textit{Comp}([\![\langle a, \gamma, \upsilon(\gamma)\rangle]\!]\ ) \oplus_{\iota} \langle Ac_{cau}(\gamma), \textit{I-Att}^{-}(a, \gamma)\rangle, && \text{if } Ac_{dif} = \varnothing,
\end{aligned}$$

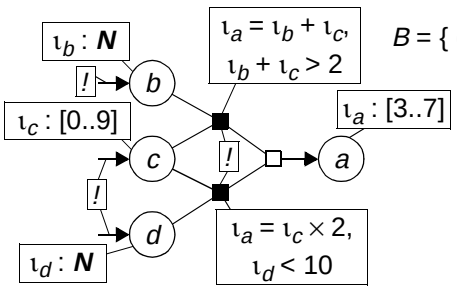
where:

- B represents the behaviour in which a is defined;
- $Ac_{cau}(\gamma)$ represents the enabling actions in γ that are referred to by a;
- $Ac_{ref}(\gamma) = Ac_{cau}(\gamma) \cup \{a\}$;
- $Ac_{dif} = Ac(\Gamma_a) - Ac([\![\langle a, \gamma, \upsilon(\gamma)\rangle]\!]\ )$, which represents the actions that are defined in Γ_a but are not contained in the action domain of $[\![\ \langle a, \gamma, \upsilon(\gamma)\rangle\ ]\!]$. This has been explained in Section 5.2.2. ■

The execution semantics of a behaviour B is defined as $[\![\ B\ ]\!] = \checkmark \otimes\{\ [\![\ \rho\ ]\!]\ (B) \mid \rho \in B\}$, as explained in Section 5.2.2. We assume that the cross-conjunction operator $\otimes$ is extended to executions with information values, as it has been presented in Section 3.7.

Example 1

Figure 6.8 depicts behaviour B consisting of actions a, b, c and d. This example illustrates the application of Definition 6.5 to a disjunction of two (composite) enabling conditions.



$$\begin{aligned}
B = \{\ &(b \wedge c)_{!}[\iota_b + \iota_c > 2] \vee (c \wedge d)_{!}[\iota_d < 10] \rightarrow a\ (\iota_a : [3..7])\\
&[\text{if } b \wedge c \rightarrow a \text{ then } \iota_a = \iota_b + \iota_c,\\
&\ \text{if } c \wedge d \rightarrow a \text{ then } \iota_a = \iota_c \times 2],\\
&\checkmark_{!} \rightarrow b\ (\iota_b : \boldsymbol{N}),\\
&\checkmark_{!} \rightarrow c\ (\iota_c : [0..9]),\\
&\checkmark_{!} \rightarrow d\ (\iota_d : \boldsymbol{N})\ \}
\end{aligned}$$

Figure 6.8: Example 1: behaviour B

The information attribute constraints associated with the alternative causality conditions of actions a, b, c and d are defined as:

$$\textit{I-Att}(a, b \wedge c) = \{\ \{\langle b, \iota_b\rangle, \langle c, \iota_c\rangle, \langle a, \iota_a\rangle\} \mid \iota_a = \iota_b + \iota_c, \iota_b + \iota_c > 2, \iota_a \in [3..7]\};$$

$$\textit{I-Att}(a, c \wedge d) = \{\ \{\langle c, \iota_c\rangle, \langle d, \iota_d\rangle, \langle a, \iota_a\rangle\} \mid \iota_a = \iota_c \times 2, \iota_d < 10, \iota_a \in [3..7]\};$$

$I\text{-}Att(b, \surd) = \{ \{\langle b, \iota_b\rangle\} \mid \iota_b \in \boldsymbol{N}\}$;

$I\text{-}Att(c, \surd) = \{ \{\langle c, \iota_c\rangle\} \mid \iota_c \in [0..9]\}$;

$I\text{-}Att(d, \surd) = \{ \{\langle d, \iota_d\rangle\} \mid \iota_d \in \boldsymbol{N}\}\}$.

The execution semantics of behaviour B is defined as:

$$[\![B]\!] = \checkmark \otimes \{ [\![\rho_z]\!] (B) \mid z \in Ac(B)\}, \text{ with}$$

$$\begin{aligned}
[\![\rho_a]\!] (B) = \; & (\; ([\![b \wedge c \rightarrow a]\!] (B) \oplus_\iota \langle\{a, b, c\}, I\text{-}Att(a, b \wedge c)\rangle) \\
& \quad \cup ([\![c \wedge d \rightarrow a]\!] (B) \oplus_\iota \langle\{a, c, d\}, I\text{-}Att(a, c \wedge d)\rangle) \;) \\
& \otimes \\
& Comp_\iota(\; ((Comp([\![\upsilon_a(b \wedge c) = !]\!] \;) \oplus_\iota \langle\{b, c\}, I\text{-}Att^{-}(a, b \wedge c)\rangle) \otimes EE\text{-}Free(\{d\})) \\
& \quad \cup ((Comp([\![\upsilon_a(c \wedge d) = !]\!] \;) \oplus_\iota \langle\{c, d\}, I\text{-}Att^{-}(a, c \wedge d)\rangle) \otimes EE\text{-}Free(\{b\})) \;);
\end{aligned}$$

$$\begin{aligned}
[\![\rho_b]\!] (B) = \; & ([\![\surd \rightarrow b]\!] (B) \oplus_\iota \langle\{b\}, I\text{-}Att(b, \surd)\rangle) \\
& \otimes Comp_\iota(\; (Comp([\![\upsilon_b(\surd) = !]\!] \;) \oplus_\iota \langle\varnothing, I\text{-}Att^{-}(b, \surd)\rangle) \;) \\
= \; & ([\![\surd \rightarrow b]\!] (B) \oplus_\iota \langle\{b\}, I\text{-}Att(b, \surd)\rangle) \otimes [\![\upsilon_b(\surd) = !]\!] \text{ , with } I\text{-}Att^{-}(b, \surd)) = \{\varnothing\};
\end{aligned}$$

$$[\![\rho_c]\!] (B) = ([\![\surd \rightarrow c]\!] (B) \oplus_\iota \langle\{c\}, I\text{-}Att(c, \surd)\rangle) \otimes [\![\upsilon_c(\surd) = !]\!] \text{ , with } I\text{-}Att^{-}(c, \surd)) = \{\varnothing\};$$

$$[\![\rho_d]\!] (B) = ([\![\surd \rightarrow d]\!] (B) \oplus_\iota \langle\{d\}, I\text{-}Att(d, \surd)\rangle) \otimes [\![\upsilon_d(\surd) = !]\!] \text{ , with } I\text{-}Att^{-}(d, \surd)) = \{\varnothing\}.$$

Figure 6.9 illustrates the construction of the execution semantics of B from the execution semantics of its causality relations.

Example 2

Figure 6.10 shows an example where the transitivity property of enabling relations is inherited by reference relations. In this example the indirect dependency of the information value of action a on the information value of action d in the first set of executions, i.e., $\iota_a = f(\iota_c)$ and $\iota_c = \iota_d \times 2$, propagates in the semantics, becoming a direct dependency, i.e., $\iota_a = f(\iota_d \times 2)$.

Example 3

Figure 6.11 depicts a behaviour B consisting of the sequential composition of actions b and a and depicts the construction of the execution semantics of B. Action a can never occur, since its information attribute constraints can not be satisfied for any information value of b, i.e.: $I\text{-}Att(a, b) = \{\{\langle b, \iota_b\rangle, \langle a, \iota_a\rangle\} \mid \iota_a = \iota_b \times 2, \iota_b > 3, \iota_a \in [3..7]\} = \varnothing$.
Consequently, execution $\boxed{b\,(\iota_b) \longrightarrow a\,(\iota_a)}$ is not allowed by $[\![\rho_a]\!] (B)$.

6.1.5 Language requirements

The pre-formal notation for information attribute constraints is not really appealing from a designer's point of view, since it is verbose and lacks conciseness. Nonetheless, this notation was chosen because its simple mathematical structures allow one to define the execution semantics of information reference relations and information causality conditions in a rather straightforward way.

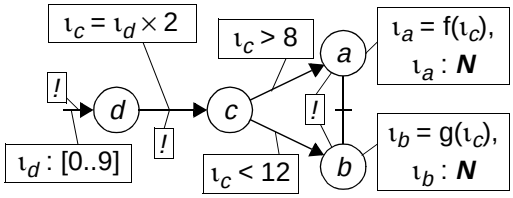
$$
\begin{array}{ll}
[\![\, \rho_a \,]\!]_{\chi}(B) = \{ & \boxed{\begin{array}{l} b\,(\iota_b) \searrow \\ \vdots\ c\,(\iota_c) \longrightarrow a\,(\iota_a) \\ \vdots\ d \end{array}}\ ,\ \boxed{\begin{array}{l} b\,(\iota_b) \searrow \\ \vdots\ c\,(\iota_c) \longrightarrow a\,(\iota_a) \\ \overline{d} \end{array}}\ \mid \iota_a = \iota_b + \iota_c,\ \iota_b + \iota_c > 2,\ \iota_a \in [3..7]\ \} \cup \\
\{ & \boxed{\begin{array}{l} b \\ \vdots\ c\,(\iota_c) \longrightarrow a\,(\iota_a) \\ \vdots\ d\,(\iota_d) \nearrow \end{array}}\ ,\ \boxed{\begin{array}{l} \overline{b} \\ c\,(\iota_c) \longrightarrow a\,(\iota_a) \\ \vdots\ d\,(\iota_d) \nearrow \end{array}}\ \mid \iota_a = \iota_c \times 2,\ \iota_d < 10,\ \iota_a \in [3..7]\ \} \cup \\
\{ & \boxed{\begin{array}{ll} b\,(\iota_b) & \\ \vdots\ c\,(\iota_c) & \overline{a} \\ \vdots\ d\,(\iota_d) & \end{array}}\ \mid \iota_b + \iota_c \notin [3..7],\ \iota_d \geq 10\} \cup \\
\{ & \boxed{\begin{array}{ll} b\,(\iota_b) & \\ \vdots\ c\,(\iota_c) & \overline{a} \\ \overline{d} & \end{array}}\ \mid \iota_b + \iota_c \notin [3..7]\} \cup \{\ \boxed{\begin{array}{ll} \overline{b} & \\ c & \overline{a} \\ \vdots\ d\,(\iota_d) & \end{array}}\ \mid \iota_d \geq 10\} \cup \\
\{ & \boxed{\begin{array}{ll} b & \\ \overline{c} & \overline{a} \\ d & \end{array}}\ ,\ \boxed{\begin{array}{ll} b & \\ \overline{c} & \overline{a} \\ \overline{d} & \end{array}}\ ,\ \boxed{\begin{array}{ll} \overline{b} & \\ c & \overline{a} \\ \overline{d} & \end{array}}\ ,\ \boxed{\begin{array}{ll} \overline{b} & \\ \overline{c} & \overline{a} \\ d & \end{array}}\ ,\ \boxed{\begin{array}{ll} \overline{b} & \\ \overline{c} & \overline{a} \\ \overline{d} & \end{array}}\ \}
\end{array}
$$

$$
\begin{array}{l}
[\![\, \rho_b \,]\!]_{\chi}(B) = \{ \boxed{b\,(\iota_b) \longrightarrow a} \mid \iota_b \in \boldsymbol{N}\} \cup \{ \boxed{b\,(\iota_b) \quad a} \mid \iota_b \in \boldsymbol{N}\} \cup \{ \boxed{b\,(\iota_b) \quad \overline{a}} \mid \iota_b \in \boldsymbol{N}\} \\
[\![\, \rho_c \,]\!]_{\chi}(B) = \{ \boxed{c\,(\iota_c) \longrightarrow a} \mid \iota_c \in [0..9]\} \cup \{ \boxed{c\,(\iota_b) \quad a} \mid \iota_c \in [0..9]\} \cup \{ \boxed{c\,(\iota_b) \quad \overline{a}} \mid \iota_c \in [0..9]\ \} \\
[\![\, \rho_d \,]\!]_{\chi}(B) = \{ \boxed{d\,(\iota_d) \longrightarrow a} \mid \iota_d \in \boldsymbol{N}\} \cup \{ \boxed{d\,(\iota_d) \quad a} \mid \iota_d \in \boldsymbol{N}\} \cup \{ \boxed{d\,(\iota_d) \quad \overline{a}} \mid \iota_d \in \boldsymbol{N}\}
\end{array}
$$

$$
\begin{array}{ll}
[\![\, B \,]\!]_{\chi} = \{ & \boxed{\begin{array}{l} b\,(\iota_b) \searrow \\ c\,(\iota_c) \longrightarrow a\,(\iota_a) \\ d \end{array}} \mid \iota_a = \iota_b + \iota_c,\ \iota_b + \iota_c > 2,\ \iota_a \in [3..7],\ \iota_b \in \boldsymbol{N},\ \iota_c \in [0..9]\ \} \cup \\
\{ & \boxed{\begin{array}{l} b \\ c\,(\iota_c) \longrightarrow a\,(\iota_a) \\ d\,(\iota_b) \nearrow \end{array}} \mid \iota_a = \iota_c \times 2,\ \iota_d < 10,\ \iota_a \in [3..7],\ \iota_d \in \boldsymbol{N},\ \iota_c \in [0..9]\ \} \cup \\
\{ & \boxed{\begin{array}{ll} b\,(\iota_b) & \\ c\,(\iota_c) & \overline{a} \\ d\,(\iota_d) & \end{array}} \mid \iota_b + \iota_c \notin [3..7],\ \iota_d \geq 10\}
\end{array}
$$

Figure 6.9: Example 1: Execution semantics of *B*

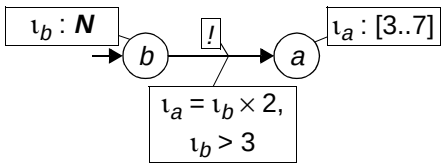
A design language is expected to support the specification of data types in a concise and intuitive way. The definition of data types involves the definition of one or more sorts of information values, such as, e.g., integers, characters, strings and sets, and operations on these sorts, such as, e.g., the multiplication operator on integers, the length operator on strings and the containment relation on sets. Furthermore, mechanisms to compose new data type definitions from existing ones should be provided by the design language.



$$B = \{ \quad (c \wedge \neg b)_! [\iota_c > 8] \rightarrow a\,(\iota_a : \boldsymbol{N})[\iota_a = f(\iota_c)],$$
$$(c \wedge \neg a)_! [\iota_c < 12] \rightarrow b\,(\iota_b : \boldsymbol{N})[\iota_b = g(\iota_c)],$$
$$d_! \rightarrow c\,(\iota_c : \boldsymbol{N})[\iota_c = \iota_d \times 2],$$
$$\surd_! \rightarrow d\,(\iota_d : [0..9]) \}$$

$$[\![B]\!]_\chi = \{ \boxed{d\,(\iota_d) \longrightarrow c\,(\iota_c) \nearrow a\,(\iota_a),\ \overline{b}} \mid \iota_a = f(\iota_c),\ \iota_c = \iota_d \times 2,\ \iota_c > 8,\ \iota_a, \iota_c \in \boldsymbol{N},\ \iota_d \in [0..9] \} \cup$$
$$\{ \boxed{d\,(\iota_d) \longrightarrow c\,(\iota_c) \searrow \overline{a},\ b\,(\iota_b)} \mid \iota_b = g(\iota_c),\ \iota_c = \iota_d \times 2,\ \iota_c < 12,\ \iota_a, \iota_c \in \boldsymbol{N},\ \iota_d \in [0..9] \}$$

Figure 6.10: Example 2



$$B = \{ b_! [\iota_b > 3] \rightarrow a\,(\iota_a : [3..7])[\iota_a = \iota_b \times 2],$$
$$\surd_! \rightarrow b\,(\iota_b : \boldsymbol{N}) \}$$

$$[\![\rho_a]\!]_\chi (B) = \{ \boxed{b\,(\iota_b) \longrightarrow a\,(\iota_a)} \mid \iota_a = \iota_b \times 2,\ \iota_b > 3,\ \iota_a : [3..7] \} \cup$$
$$\{ \boxed{b\,(\iota_b) \quad \overline{a}} \mid \iota_b : \boldsymbol{N} \} \cup \{ \boxed{\overline{b} \quad \overline{a}} \}$$
$$= \{ \boxed{b\,(\iota_b) \quad \overline{a}} \mid \iota_b : \boldsymbol{N} \} \cup \{ \boxed{\overline{b} \quad \overline{a}} \}$$

$$[\![\rho_b]\!]_\chi (B) = \{ \boxed{b\,(\iota_b) \longrightarrow a} \mid \iota_b \in \boldsymbol{N} \} \cup \{ \boxed{b\,(\iota_b) \quad a} \mid \iota_b \in \boldsymbol{N} \} \cup \{ \boxed{b\,(\iota_b) \quad \overline{a}} \mid \iota_b \in \boldsymbol{N} \}$$

$$[\![B]\!] = \{ \boxed{b\,(\iota_b) \quad \overline{a}} \mid \iota_b : \boldsymbol{N} \} \cup \{ \boxed{\overline{b} \quad \overline{a}} \}$$

Figure 6.11: Example 3

In order to facilitate and speed up the specification of data types, a library of basic data types (e.g., boolean, integer and character) and commonly used composite data types (e.g., set, list and queue) should be part of the design language. The basic data types form the elementary building blocks for the construction of composite data types. The specification language LOTOS ([4, 71]) does not provide such elementary building blocks, but requires that any data type is specified using its data type language. This property contributes to voluminous and time consuming data type specifications, despite the availability of standard libraries of commonly used data type specifications.

6.1.6 Information references between synchronized actions

The notions of information reference relation and information causality condition can be extended to synchronized actions. Figure 6.12(i) depicts an example behaviour *B* in which actions *a* and *b* must synchronize and reference relation $\iota_a = \iota_b \times 2$ defines that the information value of *a* must be equal to two times the information value of *b*.

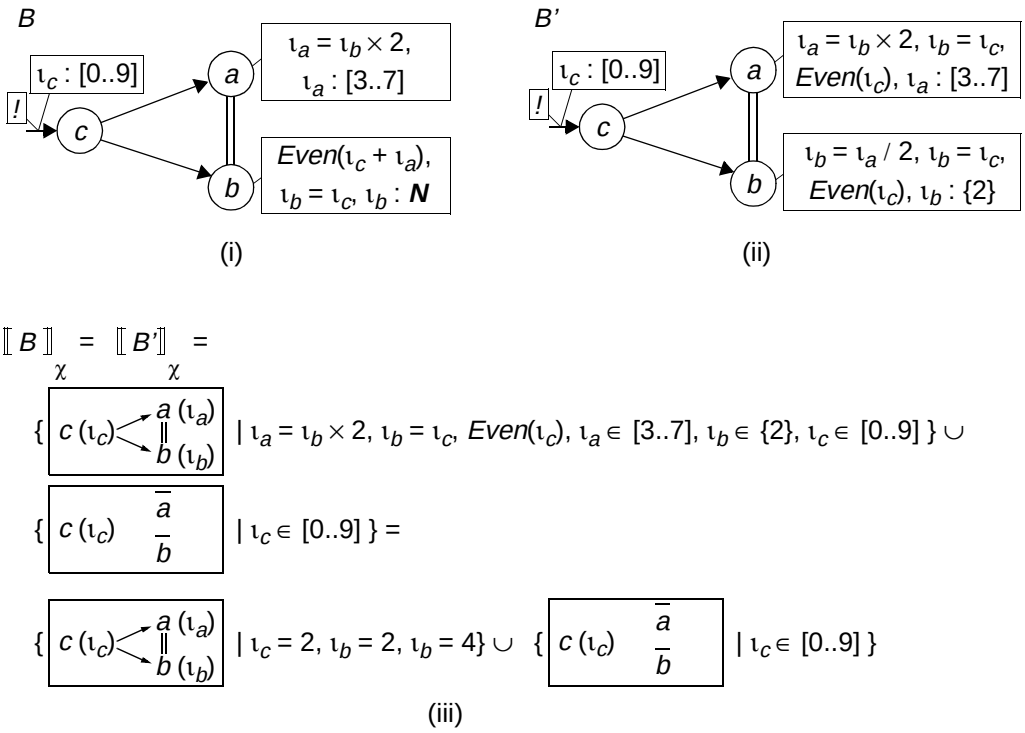


Figure 6.12: Reference relation between synchronized actions

Reference relation $\iota_a = \iota_b \times 2$ defines a reciprocal constraint, in the sense that the information value of *b* is also determined by this reference relation. This reciprocal aspect is inherited from the synchronization conditions $^{=}b$ and $^{=}a$, which define that either both *a* and *b* occur (simultaneously), or none of them occurs. This implies that an information value can be established in *b* if and only if an information value can be established in *a*.

Figure 6.12(ii) depicts an equivalent behaviour *B'*, in which some reciprocal constraints are made explicit:

- information causality condition $Even(\iota_c)$ represents that the value of ι_c must be even. This constraint follows from constraints $Even(\iota_c + \iota_a)$ and $\iota_a = \iota_b \times 2$;
- information value domain $\iota_b : \{2\}$ follows from the conjunction of $\iota_a : [3..7]$, $\iota_a = \iota_b \times 2$, $\iota_b = \iota_c$ and $Even(\iota_c)$.

Figure 6.12(iii) depicts the executions allowed by *B* and *B'*.

The extension of the pre-formal notation and its execution semantics with information reference relations and information causality conditions involving synchronized actions is straightforward and is not further elaborated in this thesis.

6.2 Location references

The extension of causality relations with a location attribute allows one to model (i) the locations where action occurrences make their results available and (ii) dependencies between the locations of different action occurrences. This extension can be performed

analogously to the extension of causality relations with an information attribute presented in Section 6.1. Location values can be considered as specific types of information values, and therefore, can be dealt with in the same way.

Figure 6.13 illustrates the modelling of location attribute constraints for a simple behaviour *B*. This behaviour consists of actions *ordering*, *production* and *delivery*, which model the ordering, production and delivery of a car, respectively. The following location attribute constraints are defined:

- the location value domain λ : *World* of actions *ordering* and *delivery* models that a car can be ordered by and delivered to any dealer in the world, respectively. Data type *World* represents a data-base containing the addresses of all car dealers in the world;
- the location causality condition $\lambda_{ordering} \neq Iraq$ models that the production of cars for Iraq is not allowed (e.g., due to some UN resolution);
- the location value domain λ : {*Italy*, *Japan*} of *production* models that a car is either produced in Italy or in Japan;
- the location reference relation associated with the enabling relation between actions *ordering* and *production* models that a car ordered by a dealer in Europe is produced in Italy, and a car ordered by a dealer outside Europe is produced in Japan;
- the location reference relation $\lambda_{delivery} = \lambda_{ordering}$ models that a car is delivered at the address of the dealer that ordered the car.

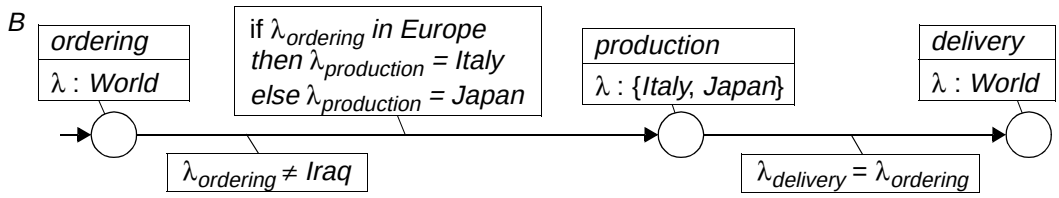


Figure 6.13: Location attribute constraints

6.3 Time references

The extension of causality relations with a time attribute allows one to model (i) the time moments when actions occur and (ii) dependencies between the time moments of different action occurrences. This extension can be performed analogously to the extension of causality relations with an information attribute presented in Section 6.1, except for two additional constraints which are explained later on in this section. Time moments (time values) can be considered as specific types of information values, and therefore, can be dealt with in the same way.

The time attribute of some result action *a* is modelled by extending the causality relation of action *a* with the definition of the time value domain of *a*, and possibly, the definition of some time reference relations and time causality conditions. Figure 6.14 illustrates the representation of time attribute constraints for two simple behaviours *B1* and *B2*.

Behaviour *B1* in Figure 6.14 illustrates the modelling of the following time constraints:

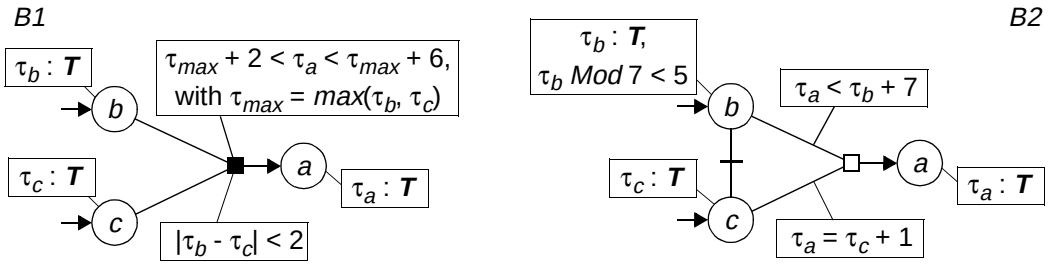


Figure 6.14: Time attribute constraints

- time reference relation $\tau_{max} + 2 < \tau_a < \tau_{max} + 6$, with $\tau_{max} = max(\tau_b, \tau_c)$, models that in case action *a* occurs, *a* must occur not earlier than 2 time units and not later than 6 time units after both actions *b* and *c* have occurred;
- time causality condition $|\tau_b - \tau_c| < 2$ models that actions *b* and *c* must both occur within a time frame of 2 time units in order to allow the occurrence of *a*.

Behaviour *B2* models a choice between actions *b* and *c* followed by action *a*. This behaviour could represent the normal and expedited ordering of an article followed by the delivery of the ordered article. In this behaviour, a time unit represents an entire day, and time value 0 represents a Monday. The following time constraints are defined:

- time value domain τ_c *Mod* 7 < 5 models that the normal ordering of an article can only be performed from Monday till Friday;
- time reference relation $\tau_a < \tau_b + 7$ models that an article is delivered within a week after a normal ordering; and
- time reference relation $\tau_a = \tau_c + 1$ models that an article is delivered the next day after an expedited ordering. An expedited ordering can also be performed in the weekend.

6.3.1 Implicit time references

In the modelling of time attributes one has to consider that causality conditions may already define time constraints implicitly. In this sense the modelling of time attributes differs from the modelling of information attributes. Figure 6.15 illustrates the modelling of an implicit time constraint by two behaviours *B1* and *B2*. In case we treat the time attributes of actions *a* and *b* as information attributes, the time reference relations $\tau_a < \tau_b + 2$ and $\tau_a = \tau_b - 3$ define that τ_a may be smaller than τ_b in *B1* and that τ_a must be smaller than τ_b in *B2*. In other words, action *b* may occur before *a* occurs. This is in conflict with the implicit time constraint $\tau_a > \tau_b$ prescribed by the enabling relation between *b* and *a*.

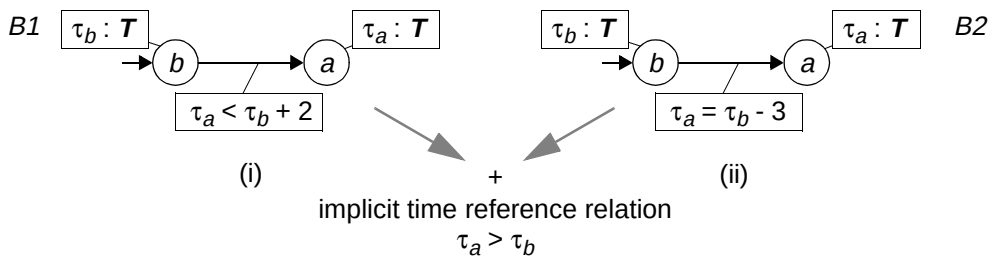


Figure 6.15: Implicit time reference relations

An enabling relation contains an implicit time constraint, which defines that the enabling actions must occur before the result action. This time constraint is called an *implicit time reference relation*. Implicit time reference relations must be made explicit in the semantics in order to determine the complete time attribute constraint of some result action.

For example, the enabling relation between actions *b* and *a* in Figure 6.15 defines the implicit time reference relation $\tau_a > \tau_b$. The combination of this time constraint with time constraints $\tau_a < \tau_b + 2$ and $\tau_a = \tau_b - 3$ defines that action *a* is allowed to occur within two time units after *b* has occurred in *B1*, i.e., $\tau_b < \tau_a < \tau_b + 2$, and that action *a* will never occur in *B2*, respectively.

An analogous reasoning applies to synchronized actions, in which the synchronization relation defines an implicit time reference relation that imposes that all synchronized actions must occur at the same time moment.

6.3.2 Pre-formal notation and execution semantics

The pre-formal notation and execution semantics of causality relations extended with a time attribute can be defined analogously to the pre-formal notation and execution semantics of causality relations extended with an information attribute, except for the replacement of the symbols ι, I, *I* and ***I*** by the symbol τ, T, *T* and ***T***, respectively. This leaves only the pre-formal notation and execution semantics of implicit time reference relations to be considered.

Implicit time reference relations are made explicit by adding to each time reference relation *T-Ref*(a, γ_a), the time constraint $\tau_a > \tau_b$ for every enabling action *b* that is referred to by *a*. This implies that the following consistency rule must hold for every time reference relation *T-Ref*(a, γ_a):

- $\forall t\text{-}ref \in T\text{-}Ref(a, \gamma_a) \mid \forall \langle b, \tau_b \rangle \in t\text{-}ref \mid \tau_b < \tau_a$.

A similar consistency rule can be defined for implicit time reference relations of synchronized actions.

The addition of the above consistency rule(s) implies that Definition 6.5 also applies to causality relations extended with a time attribute, whenever the symbols ι, I, *I* and ***I*** are replaced by the symbol τ, T, *T* and ***T***, respectively.

6.3.3 Operational interpretation

A specific operational interpretation of how action relations are (to be) implemented can originate additional implicit time constraints. This is explained for two example operational interpretations below.

Suppose that we adopt the following operational interpretation: at any moment during the execution of a behaviour, the decision if and when an action occurs can only be based on the knowledge of which actions have occurred (or not) so far. This interpretation is illustrated by behaviours *B1* and *B2* in Figure 6.16.

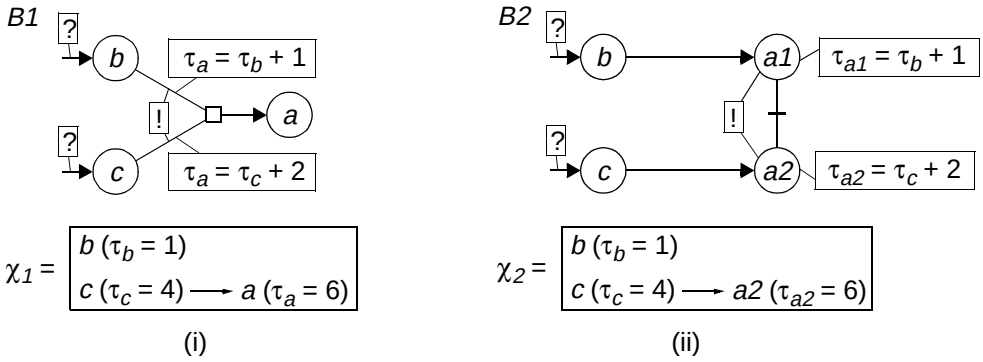


Figure 6.16: Operational interpretation

Figure 6.16(i) depicts behaviour *B1* in which action *a* is either allowed to occur due to the occurrence of action *b* or due to the occurrence of action *c*. If *a* occurs due to *b* then *a* must occur at $\tau_a = \tau_b + 1$, and if *a* occurs due to *c* then *a* must occur at $\tau_a = \tau_c + 2$. Figure 6.16(i) also depicts execution χ_1, which is a possible execution of *B1* according to the semantics we have been considering so far. However, according to the operational interpretation mentioned above, action *a* should have occurred due to the occurrence of *b* at $\tau_a = 2$ in χ_1. This operational interpretation implies that the implementation can not know before (or at) $\tau_a = 2$, whether *c* will occur after this moment or not, and therefore the implementation should decide to execute *a* one time unit after *b* due to the *must* uncertainty association $\upsilon_a(b) = !$. This interpretation also implies that execution χ_1 would not be valid.

Analogously, execution χ_2 of behaviour *B2* in Figure 6.16(ii) would not be allowed according to the same operational interpretation applied to *B2*. After *b* has occurred at $\tau_b = 1$, the implementation must decide within one time unit whether *a1* or *a2* must occur. Since *c* has not occurred before moment $\tau_b + 1$, the implementation should decide to let *a1* occur at $\tau_{a1} = \tau_b + 1$ (and thus disable the occurrence of *a2*).

As a second example, we suppose the following operational interpretation: a choice between two (or more) actions should be resolved through the occurrence of one of these actions before the explicitly defined time constraints disable one or more of the involved actions. This interpretation is illustrated by behaviour *B* in Figure 6.17, which defines a choice between actions *a* and *b*, such that either *a* must occur at $\tau_a < \tau_c + 1$, or *b* must occur at $\tau_b < \tau_c + 4$. According to the operational interpretation, execution χ depicted in Figure 6.17 would not be a valid execution of *B*, since the explicitly defined time constraint of *a* expires at $\tau_c + 1$, disabling the occurrence of *a*, while *b* only occurs 3 time units later. Instead, either *a* or *b* should have occurred before $\tau_c + 1$. This operational interpretation is rather restrictive. For example, consider that actions *c*, *a* and *b* represent the submission of a question and the return of a negative and a positive answer, respectively. The operational interpretation does not allow one to model that a positive answer may be returned after the deadline for a negative answer has expired.

The operational interpretations discussed above represent specific assumptions about how choices between alternative enabling relations are implemented. However, it is not difficult to find implementation strategies for the above examples that do not comply with these operational interpretations, but are still acceptable. For example in case of behaviours *B1*

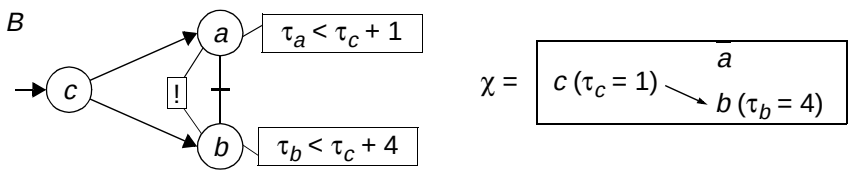


Figure 6.17: Operational interpretation (2)

and *B2* in Figure 6.16, we may consider an implementation in which from the beginning of the behaviour the occurrences of *b* and *c* are scheduled to occur at $\tau_b = 1$ and $\tau_c = 4$. This implies that the implementation knows before time moment $\tau_b + 1$ that *c* will surely occur at $\tau_c = 4$, such that the implementation can safely decide to let *a* occur due to *c* at $\tau_a = 6$ in *B1*, and to let *a2* occur due to *c* at $\tau_{a2} = 6$ in *B2*.

We conclude therefore that one should be careful to consider operational interpretations when defining the semantics of a design model. Although they may seem intuitive at first sight, some alternative implementation strategies that do not comply with these interpretations may still be acceptable. Therefore, the adoption of these interpretations may unnecessarily constrain the expressive power of the design model. Nonetheless, developers of a design model are free to add operational interpretations if necessary. This can be achieved by imposing additional constraints on function $[\![\ ]\!]$. We choose however not to prescribe any operational interpretations of this sort in this thesis.

6.4 Mixed references

This section extends causality relations with an information, a location and a time attribute. This allows one to model (i) the establishment of values for these attributes in action occurrences and (ii) dependencies between these values in different action occurrences.

6.4.1 Mixed attribute constraints

The extension of causality relations with multiple types of action attributes is straightforward in case each of these attribute types are used independently. However, relations can also be defined between values of different attribute types. These relations are called *mixed attribute constraints*, since they relate different attributes and constrain the possible combinations of values that can be established by these attributes.

Assuming that the information, time and location attributes of result action *a* are defined, the following mixed attribute constraints of action *a* are distinguished:

- the *mixed value domain*, which defines the combinations of information, time and location values that can be established in action *a*;
- *mixed reference relations*, which define how the combination of information, time and location values established in action *a* depends on the combinations of information, time and location values established in the actions in the causality condition of *a*;
- *mixed causality conditions*, which define how the occurrence of *a* depends on the com-

binations of information, time and location values established in the actions in the causality condition of *a*.

Mixed value domains

A result action *a* may restrict the possible combinations of information, time and location values that can be established in this action to a subset of the combinations allowed by the cartesian product of the information, time and location value domains of this action. This subset is called the mixed value domain of *a*. Figure 6.18 illustrates the mixed value domain for an instance of a postal mail delivery service.

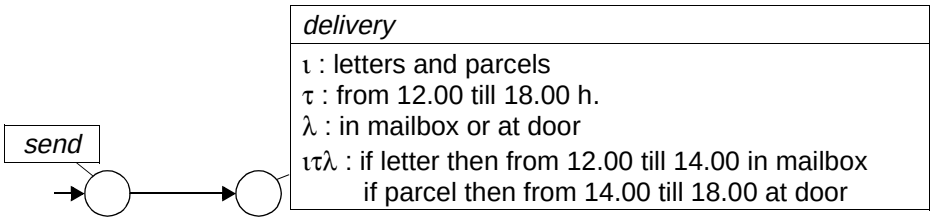


Figure 6.18: Mixed value domain

Actions *send* and *delivery* represent the sending and delivery of postal mail, respectively. The cartesian product of the information, time and location value domains of action *delivery* defines that letters and parcels can be delivered to the recipient's mailbox or can be personally handed at the recipient's door, sometime in the afternoon. A mixed value domain, which is ranged over by ιτλ, is defined to represent constraints involving different attributes. This mixed value domain prescribes letters to be delivered in the recipient's mailbox sometime between 12.00 and 14.00 hours and parcels to be personally handed at the recipient's door sometime between 14.00 and 18.00 hours.

In principle, the sole definition of the mixed value domain suffices, since it comprises the information, time and location value domains. However, for reasons of clarity, we also define each of the latter domains explicitly.

Mixed reference relations

The combination of information, time and location values established by some result action *a* may depend on the combinations of information, time and location values established in the enabling actions referred to by *a*. The relation between the combination of attribute values of *a* and the combinations of attribute values of the referred enabling actions in the resulting causality condition of *a*, is called a *mixed reference relation*. A mixed reference relation should be associated with each alternative causality condition of *a*.

Figure 6.19 illustrates the definition of a mixed reference relation by means of an instance of an e-mail service.

Actions *send* and *receive* represent the sending and receiving of an e-mail message, respectively. The information values of action *send* and *receive* involve the following fields: *To* : *Address*, which represents the destination address of the e-mail message; *From* : *Address*,

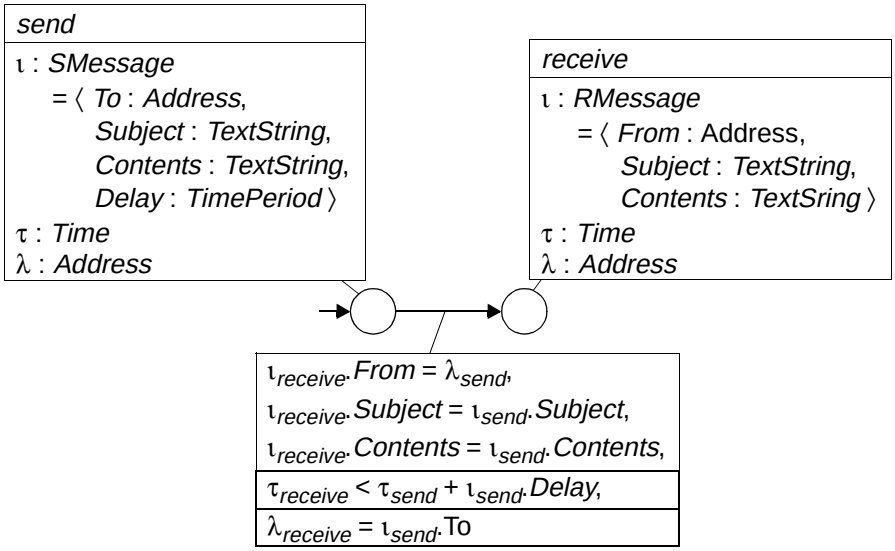


Figure 6.19: Mixed reference relation

which represents the source address of the e-mail message; *Subject* : *TextString*, which represents the subject of the e-mail message; *Contents* : *TextString*, which represents the actual e-mail message; and *Delay* : *TimePeriod*, which represents the maximal transfer delay of the e-mail message.

The mixed reference relation of action *receive*, which is represented in the middle text-box linked to the enabling relation in Figure 6.19, defines the following dependencies:

- the information value of action *receive* is composed of the location value and the *Subject* and *Contents* fields in the information value of action *send*;
- the maximal time value of action *receive* is defined by the sum of the time value and the *Delay* field in the information value of action *send*;
- the location value of action *receive* is determined by the *To* field in the information value of action *send*.

Mixed causality conditions

An alternative causality condition may define conditions on the combinations of information, time and location values of its enabling actions. These conditions are called *mixed causality conditions*. Figure 6.20 illustrates the definition of a mixed causality condition by means of an instance of the overseas delivery of articles.

Actions *shipment*, *notification* and *local_transport* represent the shipment of articles to a local harbour, the notification of this shipment indicating the harbour where the article is delivered, and the local transport of the article to its final destination, respectively. The mixed causality condition, which is represented in the text-box linked to the conjunction symbol in Figure 6.20, defines the following conditions on the attribute values of actions *notification* and *shipment*:

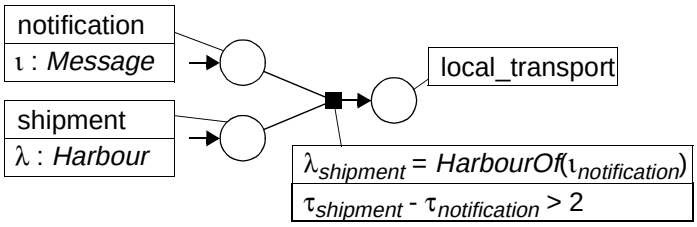


Figure 6.20: Mixed causality condition

- $\lambda_{shipment} = HarbourOf(\iota_{notification})$ represents that local transport is only possible when the actual harbour to which the article is being shipped is the same as the harbour being indicated in the notification;
- $\tau_{shipment} - \tau_{notification} > 2$ represents that local transport can only be arranged when (the arrival of) the shipment is notified at least two days in advance.

6.4.2 Pre-formal notation

The pre-formal notation of mixed value domains, mixed reference relations, mixed causality conditions and causality relations extended with an information, a time and a location attribute is defined below. For this purpose, we consider some result action *a*, and we assume that the information, time and location attributes of action *a* are defined.

Mixed value domains

The mixed value domain of some action is defined as part of the action specification of this action (see Chapter 2, Definition 2.1). An action specification is defined as a five-tuple $\langle a, \mathrm{I}_a, \mathrm{T}_a, \Lambda_a, \varsigma_a \rangle$, where *a* identifies the action and I_a, T_a, Λ_a and ς_a represent the information, time, location and mixed value domains of *a*, respectively, with $\varsigma_a \subseteq \mathrm{I}_a \times \mathrm{T}_a \times \Lambda_a$. For example, in case of the example in Figure 6.18, the mixed value domain of action *delivery* can be defined as:

$$\begin{aligned}\varsigma_{delivery} = \{ \langle \iota, \tau, \lambda \rangle \mid\ & \iota \in \text{Letters} \Leftrightarrow \tau \in [12.00, 14.00] \wedge \lambda \in \text{Mailboxes}, \\ & \iota \in \text{Parcels} \Leftrightarrow \tau \in [14.00, 18.00] \wedge \lambda \in \text{Doors}, \\ & \iota \in \text{Letters} \cup \text{Parcels}, \tau \in [12.00, 18.00], \lambda \in \text{Mailboxes} \cup \text{Doors} \} .\end{aligned}$$

Mixed reference relations

The mixed reference relation associated with alternative causality condition γ_a of result action *a* is denoted as $ITL\text{-}Ref_a(\gamma_a)$, or alternatively as $ITL\text{-}Ref(a, \gamma_a)$. Mixed reference relation $ITL\text{-}Ref_a(\gamma_a)$ is defined as a subset of the cartesian product of the information, time and location reference relations $I\text{-}Ref_a(\gamma_a)$, $T\text{-}Ref_a(\gamma_a)$ and $L\text{-}Ref_a(\gamma_a)$, by adding constraints on references involving different attribute types. This implies that $ITL\text{-}Ref_a(\gamma_a)$ defines all possible mixed references between the attribute values of result action *a* and the attribute values of the referred enabling actions in γ_a, where a mixed reference is defined as a three-tuple consisting of an information reference, a time reference and a location reference. We assume that $ITL\text{-}Ref_a(\gamma_a)$ only refers to defined attributes of enabling actions in γ_a.

For example, in case of the behaviour in Figure 6.19, the mixed reference relation of action *receive* is defined as:

$$
\begin{aligned}
&\mathit{ITL\text{-}Ref}_{receive}(\mathit{send}) \\
&= \{ \ \langle \ \{\langle \mathit{send}, \iota_{send}\rangle, \langle \mathit{receive}, \iota_{receive}\rangle\}, \{\langle \mathit{send}, \tau_{send}\rangle, \langle \mathit{receive}, \tau_{receive}\rangle\}, \\
&\qquad \{\langle \mathit{send}, \lambda_{send}\rangle, \langle \mathit{receive}, \lambda_{receive}\rangle\} \ \rangle \\
&\qquad \mid \ \iota_{receive}.\mathit{From} = \lambda_{send}, \iota_{receive}.\mathit{Subject} = \iota_{send}.\mathit{Subject}, \\
&\qquad \iota_{receive}.\mathit{Contents} = \iota_{send}.\mathit{Contents}, \tau_{receive} < \tau_{send} + \iota_{send}.\mathit{Delay}, \\
&\qquad \lambda_{receive} = \iota_{send}.\mathit{To} \ \} \\
&\subseteq \mathit{I\text{-}Ref}_{receive}(\mathit{send}) \times \mathit{T\text{-}Ref}_{receive}(\mathit{send}) \times \mathit{L\text{-}Ref}_{receive}(\mathit{send}),
\end{aligned}
$$

where

$$
\begin{aligned}
&\mathit{I\text{-}Ref}_{receive}(\mathit{send}) \\
&= \{ \ \{\langle \mathit{send}, \iota_{send}\rangle, \langle \mathit{receive}, \iota_{receive}\rangle\} \mid \\
&\qquad \iota_{receive}.\mathit{Subject} = \iota_{send}.\mathit{Subject}, \iota_{receive}.\mathit{Contents} = \iota_{send}.\mathit{Contents} \ \}; \\
&\mathit{T\text{-}Ref}_{receive}(\mathit{send}) \\
&= \{ \ \{\langle \mathit{send}, \tau_{send}\rangle, \langle \mathit{receive}, \tau_{receive}\rangle \ \} \mid \tau_{receive} > \tau_{send} \ \}; \\
&\mathit{L\text{-}Ref}_{receive}(\mathit{send}) \\
&= \{ \ \{\langle \mathit{send}, \lambda_{send}\rangle, \langle \mathit{receive}, \lambda_{receive}\rangle \ \} \mid \lambda_{send}, \lambda_{receive} \in \boldsymbol{L}\}.
\end{aligned}
$$

A mixed reference relation $\mathit{ITL\text{-}Ref}_a(\gamma_a)$ is equal to the cartesian product of $\mathit{I\text{-}Ref}_a(\gamma_a)$, $\mathit{T\text{-}Ref}_a(\gamma_a)$ and $\mathit{L\text{-}Ref}_a(\gamma_a)$ in case no references are defined between different attribute types. The set of mixed reference relations of action a, consisting of one mixed reference relation for each alternative causality condition of a, is denoted as $\mathit{ITL\text{-}Refs}_a(\gamma_a)$, or alternatively as $\mathit{ITL\text{-}Refs}(a, \gamma_a)$.

In principle, the sole definition of the mixed reference relations suffices, since they comprise the information, time and location reference relations. However, for reasons of clarity, we also define each of the latter reference relations explicitly.

Mixed causality conditions

The mixed causality condition associated with alternative causality condition γ_a of result action a is denoted as $\mathit{ITL\text{-}Cau}_a(\gamma_a)$, or alternatively as $\mathit{ITL\text{-}Cau}(a, \gamma_a)$. The mixed causality condition $\mathit{ITL\text{-}Cau}_a(\gamma_a)$ is defined as a subset of the cartesian product of the information, time and location causality conditions $\mathit{I\text{-}Cau}_a(\gamma_a)$, $\mathit{T\text{-}Cau}_a(\gamma_a)$ and $\mathit{L\text{-}Cau}_a(\gamma_a)$, by adding constraints on the possible combinations of values of different attribute types. This implies that $\mathit{ITL\text{-}Cau}_a(\gamma_a)$ defines all possible combinations of information, time and location values of the referred enabling actions in γ_a that allow the occurrence of a. We assume $\mathit{ITL\text{-}Cau}_a(\gamma_a)$ only refers to defined attributes of enabling actions in γ_a.

For example, in case of the behaviour in Figure 6.20, the mixed causality condition of action *local_transport* is defined as:

$$
\begin{aligned}
&\mathit{ITL\text{-}Cau}_{local\_transport}(\mathit{notification} \wedge \mathit{shipment}) \\
&= \{ \ \langle \ \{\langle \mathit{notification}, \iota_{notification}\rangle\}, \{\langle \mathit{notification}, \tau_{notification}\rangle, \langle \mathit{shipment}, \tau_{shipment}\rangle\}, \\
&\qquad \{\langle \mathit{shipment}, \lambda_{shipment}\rangle\} \ \rangle
\end{aligned}
$$

$\mid \quad \lambda_{shipment} = HarbourOf(\iota_{notification}), \tau_{shipment} - \tau_{notification} > 2 \}$

$\subseteq \; I\text{-}Cau_{local\_transport}(\gamma) \times T\text{-}Cau_{local\_transport}(\gamma) \times L\text{-}Cau_{local\_transport}(\gamma)$,
with: $\gamma = notification \wedge shipment$,

where

$I\text{-}Cau_{local\_transport}(notification \wedge shipment)$
$= \{ \{\langle notification, \iota_{notification}\rangle\} \mid \iota_{notification} \in \boldsymbol{I} \}$;

$T\text{-}Cau_{local\_transport}(notification \wedge shipment)$
$= \{ \{\langle notification, \tau_{notification}\rangle, \langle shipment, \tau_{shipment}\rangle\} \mid \tau_{shipment} - \tau_{notification} > 2 \}$;

$L\text{-}Cau_{local\_transport}(notification \wedge shipment)$
$= \{ \{\langle shipment, \lambda_{shipment}\rangle\} \mid \lambda_{shipment} \in \boldsymbol{L} \}$.

A mixed causality condition $ITL\text{-}Cau_a(\gamma_a)$ is equal to the cartesian product of $I\text{-}Cau_a(\gamma_a)$, $T\text{-}Cau_a(\gamma_a)$ and $L\text{-}Cau_a(\gamma_a)$ in case no constraints are defined on the combination of different attribute types. The set of mixed causality conditions of action a, consisting of one mixed causality condition for each alternative causality condition of a, is denoted as $ITL\text{-}Caus_a(\gamma_a)$, or alternatively as $ITL\text{-}Caus(a, \gamma_a)$.

In principle, the sole definition of the mixed causality conditions suffices, since they comprise the information, time and location causality conditions. However, for reasons of clarity, we also define each of the latter conditions explicitly.

Causality relations

The pre-formal notation of causality relations extended with an information, a time and a location attribute is defined below.

Definition 6.6 The *causality relation* of an action a is defined as a five-tuple $\langle\langle a, \mathrm{I}, \mathrm{T}, \Lambda, \varsigma\rangle, \Gamma, \upsilon, \langle I\text{-}Refs, T\text{-}Refs, L\text{-}Refs, ITL\text{-}Refs\rangle, \langle I\text{-}Caus, T\text{-}Caus, L\text{-}Caus, ITL\text{-}Caus\rangle\rangle$, where

- $\langle a, \mathrm{I}, \mathrm{T}, \Lambda, \varsigma\rangle$, is the specification of result action a, where
  - $a \in \boldsymbol{A}$, identifies result action a;
  - $\mathrm{I}, \mathrm{T}, \Lambda$ are the information, time and location value domains of a, respectively;
  - $\varsigma \subseteq \mathrm{I} \times \mathrm{T} \times \Lambda$ is the mixed value domain of a;
- $\Gamma \in \boldsymbol{CC}$, is the causality condition of a;
- $\upsilon : \{a\} \times \Gamma \rightarrow \boldsymbol{U}$, is the uncertainty attribute of a;
- $\langle I\text{-}Refs, T\text{-}Refs, L\text{-}Refs, ITL\text{-}Refs\rangle$ defines the information, time, location and mixed reference relations of a, respectively, with:
  - $I\text{-}Refs : \{a\} \times \{\Gamma\} \rightarrow \wp(\wp(\wp(\boldsymbol{A} \times \boldsymbol{I})))$, defined as in Definition 6.1;
  - $T\text{-}Refs : \{a\} \times \{\Gamma\} \rightarrow \wp(\wp(\wp(\boldsymbol{A} \times \boldsymbol{T})))$, is defined analogously to $I\text{-}Refs$, including the additional consistency rule concerning implicit time reference relations as explained in Section 6.3;
  - $L\text{-}Refs : \{a\} \times \{\Gamma\} \rightarrow \wp(\wp(\wp(\boldsymbol{A} \times \boldsymbol{L})))$, is defined analogously to $I\text{-}Refs$;

- $ITL\text{-}Caus \subseteq I\text{-}Caus \times T\text{-}Caus \times L\text{-}Caus$;

• $\langle I\text{-}Caus, T\text{-}Caus, L\text{-}Caus, ITL\text{-}Caus\rangle$ defines the information, time, location and mixed causality conditions of a, respectively, with:

- $I\text{-}Caus : \{a\} \times \{\Gamma\} \rightarrow \wp(\wp(\wp(\boldsymbol{A} \times \boldsymbol{I})))$, defined as in Definition 6.1;
- $T\text{-}Caus : \{a\} \times \{\Gamma\} \rightarrow \wp(\wp(\wp(\boldsymbol{A} \times \boldsymbol{T})))$, is defined analogously to $I\text{-}Caus$;
- $L\text{-}Caus : \{a\} \times \{\Gamma\} \rightarrow \wp(\wp(\wp(\boldsymbol{A} \times \boldsymbol{L})))$, is defined analogously to $I\text{-}Caus$;
- $ITL\text{-}Caus \subseteq I\text{-}Caus \times T\text{-}Caus \times L\text{-}Caus$. ■

In case one of the attributes of an action is omitted in the design notation, the corresponding attribute constraints can be left undefined in the pre-formal notation.

Behaviours

The extension of causality relations with an information, a time and a location attribute implies that, in addition to a causality condition part and an uncertainty attribute part (see Section 6.1.3), a *mixed attribute part* $B_{\iota\tau\lambda}(Ac)$ can be distinguished in behaviour definitions, with:

$$B_{\iota\tau\lambda}(Ac) = \{\langle\langle a, \mathrm{I}, \mathrm{T}, \Lambda, \varsigma\rangle, \langle I\text{-}Refs, T\text{-}Refs, L\text{-}Refs, ITL\text{-}Refs\rangle, \langle I\text{-}Caus, T\text{-}Caus, L\text{-}Caus, ITL\text{-}Caus\rangle\rangle \mid a \in Ac\}.$$

6.4.3 Formal definition

The execution semantics of causality relations extended with an information, a time and a location attribute can be defined analogously to the execution semantics defined in Section 6.1.4, by replacing the information attribute constraints I_a, $I\text{-}Refs_a$ and $I\text{-}Caus_a$ by the mixed attribute constraints ς_a, $ITL\text{-}Refs_a$ and $ITL\text{-}Caus_a$, respectively. Observe that it is sufficient to consider the mixed attribute constraints, since they comprise the information, location and time attribute constraints.

The definitions of mixed attribute constraint $ITL\text{-}Atts_a(\Gamma_a)$, mixed attribute addition operator $+_{\iota\tau\lambda}$ and mixed attribute expansion operator $\oplus_{\iota\tau\lambda}$ that replace $I\text{-}Atts_a(\Gamma_a)$, $+_\iota$ and $\oplus_\iota$, respectively, are presented below.

Definition 6.7 Given causality relation $\langle\langle a, \mathrm{I}, \mathrm{T}, \Lambda, \varsigma\rangle, \Gamma, \upsilon, \langle I\text{-}Refs, T\text{-}Refs, L\text{-}Refs, ITL\text{-}Refs\rangle, \langle I\text{-}Caus, T\text{-}Caus, L\text{-}Caus, ITL\text{-}Caus\rangle\rangle$, the mixed attribute constraint of result action a is defined as:

$$ITL\text{-}Atts(a, \Gamma_a) = \{ITL\text{-}Att(a, \gamma_a) \mid \gamma_a \in \Gamma_a\} ;$$

$$\begin{aligned} ITL\text{-}Att(a, \gamma_a) = \{ & \langle i\text{-}ref, t\text{-}ref, l\text{-}ref\rangle \\ & \mid \langle i\text{-}ref, t\text{-}ref, l\text{-}ref\rangle \in ITL\text{-}Ref(a, \gamma_a), \\ & \quad \langle a, \iota_a\rangle \in i\text{-}ref, \langle a, \tau_a\rangle \in t\text{-}ref, \langle a, \lambda_a\rangle \in l\text{-}ref \\ & \quad \Rightarrow \langle \iota_a, \tau_a, \lambda_a\rangle \in \varsigma \\ & \qquad \wedge \\ & \qquad \exists\langle i\text{-}cau, t\text{-}cau, l\text{-}cau\rangle \in ITL\text{-}Cau(a, \gamma_a) \mid \end{aligned}$$

$$(\textit{i-ref} - \{\langle a, \iota_a\rangle\}) \subseteq \textit{i-cau}, (\textit{t-ref} - \{\langle a, \tau_a\rangle\}) \subseteq \textit{t-cau},$$
$$(\textit{l-ref} - \{\langle a, \lambda_a\rangle\}) \subseteq \textit{l-cau}\ \}. \qquad \blacksquare$$

Definition 6.8 The mixed attribute addition operator $+_{\iota\tau\lambda} : \boldsymbol{XX} \times (\wp(\boldsymbol{A} \times \boldsymbol{I}) \times \wp(\boldsymbol{A} \times \boldsymbol{T}) \times \wp(\boldsymbol{A} \times \boldsymbol{L})) \rightarrow \boldsymbol{XX}_{\iota\tau\lambda}$ is defined as:

$$\begin{aligned}\langle A, \overline{A}, <, =, |\rangle +_{\iota\tau\lambda} \langle \textit{i-ref}, \textit{t-ref}, \textit{l-ref}\rangle &= \langle A, \overline{A}, <, =, |, \textit{i-ref}, \textit{t-ref}, \textit{l-ref}\rangle \\ &\qquad \text{if } Ac(\textit{i-ref}) \cup Ac(\textit{t-ref}) \cup Ac(\textit{l-ref}) \subseteq A\ ; \\ &= \text{undefined} \\ &\qquad \text{otherwise}\ . \qquad \blacksquare\end{aligned}$$

Definition 6.9 The information expansion operator $\oplus_{\iota\tau\lambda} : \wp(\boldsymbol{XX}) \times (\wp(\wp(\boldsymbol{A} \times \boldsymbol{I})) \times \wp(\wp(\boldsymbol{A} \times \boldsymbol{T})) \times \wp(\wp(\boldsymbol{A} \times \boldsymbol{L}))) \rightarrow \wp(\boldsymbol{XX}_{\iota\tau\lambda})$ is defined as:

$$\begin{aligned}EE \oplus_{\iota\tau\lambda} \langle Ac_{ref}, \textit{ITL-Ref}\rangle &= \{e\chi +_{\iota\tau\lambda} \textit{itl-ref} \mid e\chi \in EE, Ac_{ref} \subseteq A(e\chi), \textit{itl-ref} \in \textit{ITL-Ref}\} \\ &\quad \cup \{e\chi \mid e\chi \in EE, Ac_{ref} \not\subseteq A(e\chi)\} \qquad \text{if } Ac_{ref} \neq \emptyset; \\ &= EE \qquad \text{otherwise,}\end{aligned}$$

where $EE \in \wp(\boldsymbol{XX})$, $\textit{ITL-Ref} \in \wp(\wp(\boldsymbol{A} \times \boldsymbol{I})) \times \wp(\wp(\boldsymbol{A} \times \boldsymbol{T})) \times \wp(\wp(\boldsymbol{A} \times \boldsymbol{L}))$ and Ac_{ref} represents the action occurrences that have an information, time and location value. $\blacksquare$

6.4.4 Additional remarks

The notions of mixed reference relations and mixed causality conditions can be extended rather straightforwardly for synchronized actions, as indicated for the information attribute in Section 6.1.6. This extension is not further elaborated in this thesis for the sake of conciseness.

6.5 Related work

This section relates the modelling of time in our work to some time extensions that have been proposed for process algebras, in particular LOTOS, and to time extensions that have been proposed for (extended bundle) event structures. We have not considered property-oriented techniques for behaviour specification, such as temporal logic, because we concentrate on the use of constructive techniques for distributed systems design ([24]).

This analysis can be performed from an architectural and from a formal perspective. From an architectural perspective, we are interested in the expressive power of the time modelling techniques used in the extensions mentioned above, i.e., the type of time constraints that can be modelled with these techniques. Furthermore, we abstract as much as possible from aspects that are related to or are a consequence of the modelling of interactions, since we focus in this thesis on the action concept.

From a formal perspective, we only consider how the semantical models used in the time extensions mentioned above determine the modelling of the passing of time. In some time extensions, the passing of time has to be explicitly modelled due to the underlying interleav-

ing semantics ([18]), e.g., by a function *MakeOlder()* such as in [58], by specific time transitions and function *age()* in [9], or by the occurrence of a special action χ in [53, 30]. Since actions in interleaved processes have to be ordered, the passing of time in these processes has to be coordinated, i.e., some mechanism is needed to ensure that time passes equally in these processes. In models based on a true concurrency semantics, such as our model, this problem does not appear, since actions in independent behaviour parts can occur independently, and temporal relations are implied by causal (enabling) relations.

6.5.1 Process algebras

When considering the time extensions of process algebras proposed in [9, 8, 7, 60, 59, 58, 46, 47, 50, 51, 12, 18, 52, 53, 30], we can divide them into two categories:

1. time extensions that correspond to defining a time constraint associated with an enabling condition;
2. time extensions that correspond to defining a time constraint associated with a conjunction of an enabling and a disabling condition.

Disjunctions of enabling or disabling conditions can not be expressed in traditional process algebras; neither can synchronization relations between distinct actions.

Time constraints associated with enabling conditions

This category comprises the various time extensions of the action prefix *a*; *B* (sometimes denoted as *a*.*B* or *aB*). These extensions can be modelled by a combination of the following elementary constraints on the time attribute of action *a*:

- $C \rightarrow a\ [\tau_a > \tau_C + \Delta T]$, with $\Delta T \geq 0$, which defines a lowerbound on the time moments at which *a* is allowed to occur;
- $C \rightarrow a\ [\tau_a = \tau_C + \Delta T]$, with $\Delta T > 0$, which defines one specific time moment at which *a* is allowed to occur;
- $C \rightarrow a\ [\tau_a < \tau_C + \Delta T]$, with $\Delta T > 0$ and $\tau_C < \tau_a$, which defines an upperbound on the time moments at which *a* is allowed to occur,

where *C* represents the enabling condition of action *a* and τ_C denotes the first moment at which the enabling condition is satisfied. One possibility is that *C* represents the occurrence of a single action *b* and τ_C is equal to τ_b in case of a timed action policy, i.e., when time constraints apply to individual interaction contributions. Another possibility is that *C* represents the conjunction of multiple action occurrences $\{b1, .., bn\}$ and τ_C is equal to $\max(\tau_{b1}, .., \tau_{bn})$ in case of a timed interaction policy, i.e., when time constraints apply to the conjunction of interaction contributions. The distinction between a timed action and a timed interaction policy is discussed in [7, 9].

Figure 6.21 illustrates how some of the proposed time extensions of the action prefix operator can be modelled in terms of our basic design language, by combining one or more of the time constraints indicated above. Only the time constraints of action *a* are considered. We assume that the enabling condition of action *a* has a *must* uncertainty value by default (rep-

resented by the statement: default *must*), which corresponds to the common architectural interpretation of LOTOS that an enabled action eventually occurs, unless it is disabled by another action.

	default *must*
timed choice [60]: *a {t in T}; B*	
b; a {2..4}; B	$b \rightarrow a\ [\tau_b + 2 \leq \tau_a \leq \tau_b + 4]$
timed action prefix [50]: *a {t}; B*	
b; a {3}; B	$b \rightarrow a\ [\tau_a = \tau_b + 3]$
timing [9]: *time a (t1, t2) in B*	
time a {2..4} in (b1; a; B1 |[a]| b2; a; B2)	$(b1 \wedge b2) \rightarrow a\ [\tau_{max} + 2 \leq \tau_a \leq \tau_{max} + 4]$ with: $\tau_{max} = \max(\tau_{b1}, \tau_{b2})$
arbitrary waiting [30]	
	$b \rightarrow a\ [\tau_a > \tau_b]$

Figure 6.21: Time extensions of action prefix

Urgency is a notion frequently encountered in time modelling. This notion is also known as maximal progress and minimal delay, and is sometimes used in combination with hiding. In the context of enabling conditions, we interpret urgency as follows: action *a* is urgent if *a* must occur immediately or within a finite time period after it has been enabled. This can be modelled by defining an upperbound on the moments at which action *a* can occur, i.e.:

$C \rightarrow a\ [\tau_a < \tau_C + \Delta T]$, with $\Delta T > 0$ and $\tau_C < \tau_a$.

In order to model that action *a* must occur immediately, the value of ΔT should be near zero. In contrast to (most of) the considered time extensions, we do not allow action *a* to occur at the same time as its enabling action(s). Although allowing an enabling action and its result action to occur at the same time may be convenient for mathematical reasons, we consider this as an incorrect abstraction of an enabling relation, since it results in enablings that can not be implemented in practice.

Time constraints associated with conjunctions of enabling and disabling conditions

The second category comprises various time extensions of the disabling operator, such as the time-out ($\triangleright^{d}$ or $\overset{t}{\triangleright}$), start delay ($\lfloor P \rfloor^{d}(Q)$) and the execution delay or watchdog ($\lceil P \rceil^{d}(Q)$ or $\overset{t}{\blacktriangleright}$) operators described in [46, 53, 39]. In a consistent process algebra, these time extensions are shorthand notations, which can be expanded to behaviour definitions involving only the time extended action prefix and the choice operators as basic building blocks. In ATP [53] though, the unit delay operator, which models an elementary time-out, is used as a basic building block.

The time extensions mentioned above can be modelled by associating time constraints with conjunctions of enabling and disabling conditions in terms of our basic design language. Figure 6.22 illustrates the modelling of a time-out and a watchdog mechanism.

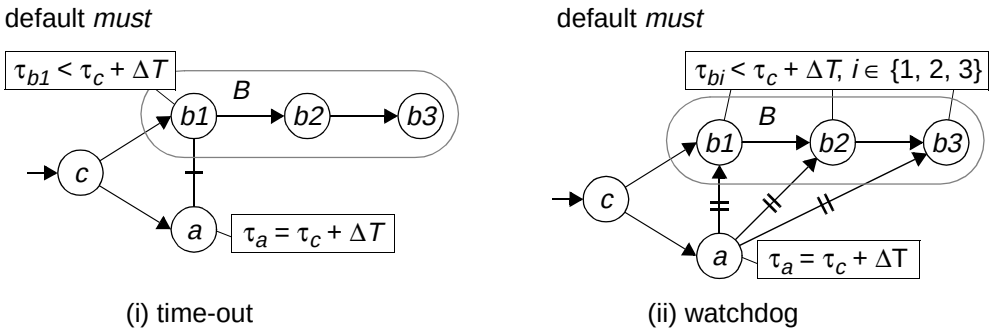


Figure 6.22: Modelling of time-out and watchdog

Figure 6.22(i) depicts a time-out mechanism for behaviour *B*. Parameter ΔT represents parameters *d* and *t* in the operators mentioned above. The start of behaviour *B* can be disabled by the occurrence of action *a* at time moment $\tau_c + \Delta T$ in case initial action *b1* of *B* has not occurred before this time moment. Action *c* initiates both behaviour *B* and its time-out. In case *B* has multiple initial actions, the choice relation between *a* and these initial actions should be defined for each one of them.

Figure 6.22(ii) depicts a watchdog mechanism for behaviour *B*, with parameter ΔT. Behaviour *B* is allowed to be executed for a maximal duration of ΔT time units after time moment τ_c. At time moment $\tau_c + \Delta T$, the occurrence of *a* disrupts the execution of *B*, provided *B* has not finished yet. Time constraint $\tau_a = \tau_c + \Delta T$ is associated with every alternative causality condition of *a*. In case action *a* is not allowed to occur after *B* has finished, the relation between *a* and *b3* should be replaced by a choice relation. Action *c* initiates both behaviour *B* and its watchdog. A feature supported by the execution delay or watchdog operators ($\lceil P \rceil^d(Q)$ or $\overset{t}{\blacktriangleright}$) in [53, 39] is to cancel the watchdog mechanism by the execution of a special action in behaviour *B*. This can be modelled in our basic design language, for example, by defining a choice relation between action *a* and the special action.

The urgency notion is also used in the context of conjunctions of enabling and disabling conditions. In this context, we interpret urgency as follows: action *a* is urgent if *a* is forced to occur immediately or within a finite time period after it has been enabled, unless it is disabled by the occurrence of another action before this time constraint expires (see e.g. [9]). For example, when we assume action *b* is urgent in the behaviour of Figure 6.23, action *b* is forced to occur at time moment $\tau_c + 6$ if action *a* has not occurred before or does not occur at this time moment.

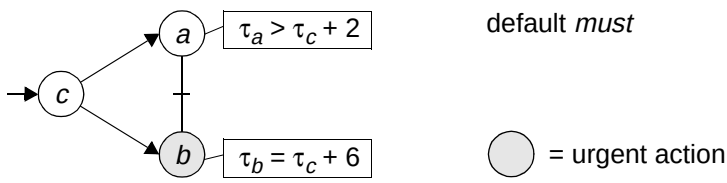


Figure 6.23: Example of urgent action

Our model does not support this notion of urgency. In our design model, action *a* is allowed to occur after time moment $\tau_c + 6$ if action *b* does not occur at this time moment. This choice has been motivated in Section 6.3.3. Furthermore, we remark that the above type of

urgency is generally introduced to model time-outs. However, we can perfectly model time-outs without needing this notion of urgency. For example, the time-out of action *a* at time moment $\tau_c + 6$ can be modelled by replacing the time constraint of action *a* by the time constraint $\tau_c + 2 < \tau_a < \tau_c + 6$.

6.5.2 Real-time event structures

Katoen et al. present in [39, 37, 10] real-time event structures, which are timed extensions of (extended) bundle event structures (see Section 5.5). A real-time event structure is defined as a four-tuple $\langle E, D, T, U\rangle$, where E is an (extended bundle) event structure $\langle E, \rightsquigarrow, \mapsto, l\rangle$, $D : E \rightarrow \wp(Time)$ is the *event delay* function, $T : \mapsto \rightarrow \wp(Time)$ is the *bundle delay* function, $U : E \rightarrow \text{Bool}$ is the *urgency* predicate and *Time* represents the time domain. Two conditions are defined for the urgency predicate in order to limit the global impact of urgent events and to ensure that urgent events are enabled at a single time moment, if ever.

The event delay function D defines the time moments at which an event *e* is allowed to occur since the start of the system. In our basic design language, this is modelled by adding a corresponding time constraint to action *e* that refers to the time moment of the start condition. This is possible since every action depends (indirectly) on the start condition. For example, $D(e) = [t_1..t_2]$ corresponds to the definition of time constraint $\tau_\surd + t_1 \leq \tau_e \leq \tau_\surd + t_2$. In correspondence to [39, 37], we assume $\tau_\surd = 0$ in this analysis.

The bundle delay function T defines for an event *e* and a bundle *X* the time moments at which *e* is allowed to occur relative to the moment at which an event in *X* occurs. In our basic design language, this is modelled by adding a corresponding time constraint to action *e* for every alternative enabling action of *e*, which refers to the time moment of this enabling action. For example, $T((\{e_1, e_2\}, e)) = [t_1..t_2]$ corresponds to the definition of the following causality relation:

$$e_1 \vee e_2 \rightarrow e \text{ [if } e_i \rightarrow e : \tau(e_i) + t_1 \leq \tau_e \leq \tau(e_i) + t_2, \text{ with } i \in \{1, 2\}],$$

when assuming $\{e_1, e_2\}$ is the only bundle of *e*. For example, the definition of additional bundle $\{e_3\}$ for event *e*, with $T((\{e_3\}, e)) = [t_3..t_4]$, renders the causality relation:

$$(e_3 \wedge e_1) \vee (e_3 \wedge e_2) \rightarrow e \text{ [if } e_3 \wedge e_i \rightarrow e : \tau(e_i) + t_1 \leq \tau_e \leq \tau(e_i) + t_2 \wedge \tau(e_3) + t_3 \leq \tau_e \leq \tau(e_3) + t_4 \text{ , with } i \in \{1, 2\} \text{].}$$

The bundle delay function does not allow one to define time constraints per event in a bundle.

The urgency predicate U defines whether an event is urgent or not. If an event is urgent it is forced to happen at the single time moment at which it is enabled, unless it is disabled by another event. The use of urgent events is restricted in [39, 37] to the modelling of time-outs and watchdogs, although these references show that watchdogs can be modelled without urgent events. The concept of urgency is needed in [39, 37] to model time-outs because event structures do not allow one to prescribe whether an event must or may occur when it is enabled. Whether an event must or may occur once it is enabled is left undefined by the event structure semantics.

6.6 Conclusions

In this chapter we extend the causality relation concept introduced in Chapter 4 with an information, a location and a time attribute. This extension allows a designer to model constraints on the possible information, location and time values that can be established in related action occurrences in a uniform and consistent way. The uniform handling of action attributes is lightly disturbed in the case of implicit time constraints and in case operational interpretations are considered, as discussed in Sections 6.3.1 and 6.3.3.

The information, location and time attributes can in principle be modelled independently from each other, since they represent independent (orthogonal) behaviour characteristics. Through the definition of mixed attribute constraints, however, it is possible to generalize the design model even more by allowing designers to constrain the possible combinations of values that can be established by distinct attribute types. Furthermore, it is possible, when referring to an established time or location value, to use this value as an information value, or when referring to an established information value, to use this value as a location or time value.

The extensions presented in this chapter result in an expressive basic design language for the modelling of information, location and time attribute constraints. We do not know of any other work that adopts a comparable approach to uniformity and consistency and renders a language with comparable expressive power.

De Weger shows in [80] that the information, time and location attribute constraints discussed in this chapter can be modelled in terms of actions and causality relations without using the information, time and location attributes. In this respect, we should consider the extension of actions and causality relations with the information, time and location attribute as providing shorthand notations to designers for the concise modelling of behaviours.

Chapter 7

Probability attributes

This chapter discusses the modelling of the integral and stochastic probability of actions, based on the integral and stochastic probability attributes introduced in Chapter 2.

The integral probability attribute is a refinement of the uncertainty attribute, meant to quantify the uncertainty of actions. This chapter presents two alternative definitions of the integral probability attribute. The simple integral probability attribute directly quantifies the uncertainty of actions as modelled by the uncertainty attribute. The extended integral probability attribute extends the simple probability attribute in order to model the individual probabilities of actions in two-sided relations, such as choice, disabling, interleaving and synchronization relations. Under certain conditions, the simple and extended probability attributes can be used in combination in a single behaviour definition.

The stochastic probability attribute is a refinement of the integral probability attribute and the time attribute, which models the probability of an action as a function of the time moments at which this action is allowed to occur. Similarly to the integral probability attribute, this chapter presents a simple and an extended definition of the stochastic probability attribute.

The structure of this chapter is as follows. Section 7.1 introduces the integral probability attribute. Section 7.2 defines the simple interpretation of the integral probability attribute and Section 7.3 formalizes this interpretation. Section 7.4 defines the extended interpretation of the integral probability attribute and Section 7.5 formalizes this interpretation. Section 7.6 discusses the combined usage of integral probability attributes and information, time or location attributes. Section 7.7 discusses the modelling of the stochastic probability of actions. And Section 7.8 presents the conclusions.

7.1 Integral probability

The integral probability attribute refines the uncertainty attribute by quantifying the uncertainty of actions. The *must* and *may* uncertainty values are replaced by the (infinite) set of integral probability values $(0..1] \subset \boldsymbol{R}$, such that the *must* value corresponds to value 1 and the *may* value corresponds to a value in the range (0..1).

A range (or set) of integral probability values is associated with each alternative causality condition γ_a of some result action a. This association is called an *integral probability asso-*

ciation, and is denoted as $\pi_a(\gamma_a)$, or alternatively as $\pi(a, \gamma_a)$. This association is defined as follows.

> The integral probability association $\pi_a(\gamma_a)$ defines the integral probability that result action a occurs when assuming that alternative causality condition γ_a is satisfied.

The integral probability of result action a as defined by $\pi_a(\gamma_a)$ is a *conditional* probability, since it assumes the satisfaction of γ_a.

In case $\pi_a(\gamma_a)$ consists of multiple probability values, this integral probability association defines a family of behaviours, one behaviour for each possible value of $\pi_a(\gamma_a)$. In this case the implementer is free to implement any behaviour from this family.

(Pre-formal) notation

Figure 7.1 illustrates the notation of integral probability associations. Integral probability associations are graphically represented in text-boxes linked to the corresponding alternative causality conditions, or alternatively in text-boxes linked to the result action. In the textual notation, integral probability associations are represented between square brackets to the right side of the corresponding alternative causality condition, or alternatively to the right side of the result action.

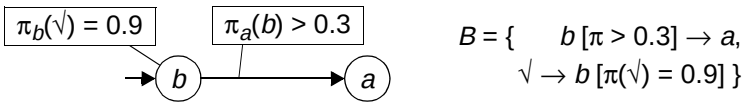


Figure 7.1: Notation

For the sake of conciseness, we represent a probability association that consists of a single value in terms of this value, refraining from representing a range of values. For example, the probability association of action b in Figure 7.1 is represented as $\pi_b(\surd) = 0.9$ instead of $\pi_b(\surd) \in \{0.9\}$ or $\pi_b(\surd) \in [0.9..0.9]$.

In the definition of formulas involving probability associations, a probability association represents a variable that ranges over the possible values of this probability association. For example, the equation $\pi_b(\surd) \times \pi_b(a) > 0.27$ holds in Figure 7.1, since $\pi_b(\surd)$ represents the value 0.9 and $\pi_b(a)$ represents any value larger than 0.3 and not larger than 1.

The pre-formal notation of causality relations using the integral probability attribute is obtained from the notation for causality relations considered so far, by replacing the uncertainty attribute by the integral probability attribute. This notation is defined below, for causality relations without information, time and location attributes.

Definition 7.1 The *causality relation* of an action a is defined as a triple $\langle a, \Gamma, \pi \rangle$, where

- $a \in \boldsymbol{A}$, is the result action;
- $\Gamma \in \boldsymbol{CC}$, is the causality condition of a; and
- $\pi : \{a\} \times \Gamma \rightarrow \wp(\boldsymbol{P})$, is the integral probability attribute of a. ■

Domain $\boldsymbol{P}$ represents the range of possible probability values, such that $\boldsymbol{P} = [0..1] \subset \boldsymbol{R}$.

Absolute probabilities

Besides the conditional probability, one is generally interested in the absolute probability of an action. The absolute probability of an action is the probability that an action occurs in an execution, only assuming the start condition is satisfied (i.e., the behaviour is executed). The absolute probability of action a is denoted as Π_a, or alternatively as $\Pi(a)$.

7.2 Simple integral probability attribute

This section defines the integral probability attribute as a straightforward refinement of the uncertainty attribute, which directly quantifies the uncertainty of action occurrences. This attribute is called the *simple (integral) probability attribute*. The notation introduced in the previous section is used to represent the simple (integral) probability attribute.

The simple probability attribute allows one to model the conditional probability of an individual action when this action is independent of or one-sided related to other actions. This attribute also allows one to model the conditional probability that at least one action occurs from a group of two-sided (probability) related actions. The exact meaning of such relations is discussed later on.

This section defines the interpretation of simple integral probability associations for elementary, conjunctive and disjunctive causality conditions.

7.2.1 Elementary causality conditions

The interpretation of the simple probability association $\pi_a(\gamma_a)$, with $\gamma_a \in \{\surd, b, \neg b, \overline{}b\}$ being an elementary causality condition of result action a, is defined as follows.

- $\pi_a(\surd)$ defines the integral probability that action a occurs when the start condition is satisfied. This corresponds to the probability that a occurs at the beginning of the behaviour, since the start condition is by definition always satisfied;
- $\pi_a(b)$ defines the integral probability that action a occurs when action b occurs before action a. This corresponds to the probability that a occurs after b, when assuming that enabling condition b is satisfied;
- $\pi_a(\neg b)$ defines the integral probability that action a occurs when action b does not occur before nor simultaneously with action a. This corresponds to the probability that either a occurs before b, or a occurs and b does not occur, when assuming that disabling condition $\neg b$ is satisfied;
- $\pi_a(\overline{}b)$ defines the integral probability that action a occurs when action b occurs simultaneously with action a. This corresponds to the probability that actions a and b occur simultaneously, since the assumption that synchronization condition $\overline{}b$ is satisfied implies the occurrence of a.

The interpretation of the definitions above is elaborated for the cases in which actions *a* and *b* are independent, one-sided related or two-sided related. This interpretation abstracts from any other actions of the behaviour in which *a* and *b* are defined.

Independence

Consider the independence of actions *a* and *b* as defined by causality relations:

$\surd \rightarrow a\ [\pi_a(\surd)]$,
$\surd \rightarrow b\ [\pi_b(\surd)]$.

The (absolute) probabilities that actions *a* and *b* occur are equal to $\pi_a(\surd)$ and $\pi_b(\surd)$, respectively. The probability that both actions *a* and *b* occur in the same execution is equal to $\pi_a(\surd) \times \pi_b(\surd)$, since the occurrences of *a* and *b* are independent. For example, assume $\pi_a(\surd) = 0.8$ and $\pi_b(\surd) = 0.9$. Theoretically, when considering 'infinitely many' executions we would observe that actions *a* and *b* both occur in 72 percent of these executions.

One-sided relations

Consider the one-sided relation between actions *a* and *b* as defined by causality relations:

$\surd \rightarrow b\ [\pi_b(\surd)]$,
$b \rightarrow a\ [\pi_a(b)]$.

The conditional probability that *a* occurs when assuming that *b* occurs is equal to $\pi_a(b)$. For example, assume $\pi_b(\surd) = 0.9$ and $\pi_a(b) > 0.3$, such as in Figure 7.1. When considering 'infinitely many' executions in which *b* occurs, action *a* occurs in more than 30 percent of these executions. The absolute probability that *a* occurs in an execution is:

$$\Pi_a = \Pi_b \times \pi_a(b) = \pi_b(\surd) \times \pi_a(b) > 0.27.$$

This example shows that a range of multiple probability values can be associated with an alternative causality condition. The larger this range is the more freedom is left to the implementer.

Two-sided relations

We consider the temporal freedom relation between actions *a* and *b* in order to discuss the interpretation of the simple probability attribute for the case of two-sided relations. The temporal freedom relation contains any two-sided relation that can be defined between *a* and *b*. This means that the discussion below applies to any two-sided relation between *a* and *b*, by assuming that the value of a simple probability association is equal to zero for each alternative causality condition that is absent in the definition of this two-sided relation.

The temporal freedom relation between actions *a* and *b* is defined by causality relations:

$\neg b \vee {}^{=}b \vee b \rightarrow a\ [\pi_a(\neg b), \pi_a({}^{=}b), \pi_a(b)]$,
$\neg a \vee {}^{=}a \vee a \rightarrow b\ [\pi_b(\neg a), \pi_b({}^{=}a), \pi_b(a)]$.

The occurrences of actions *a* and *b* are enabled at the beginning of the behaviour by the two-sided conditions $\gamma_a = \neg b$, $\gamma_b = \neg a$, $\gamma_a = {}^{=}b$ and $\gamma_b = {}^{=}a$. Since synchronization conditions

$\gamma_a = {}^{=}b$ and $\gamma_b = {}^{=}a$ are reciprocal conditions, i.e., they both represent the synchronization of a and b, they are denoted by $\gamma_a = {}^{=}b$ in the sequel. Furthermore, the probability values associated with reciprocal synchronization conditions should be the same, i.e., $\pi_a({}^{=}b) = \pi_b({}^{=}a)$.

Disabling conditions $\gamma_a = \neg b$ and $\gamma_b = \neg a$ and synchronization condition $\gamma_a = {}^{=}b$ are exclusive conditions, i.e., only one of these conditions can be satisfied at the same time. This implies that these two-sided conditions represent a choice between the following alternatives:

1. action a is allowed to occur due to $\gamma_a = \neg b$, assuming the satisfaction of $\gamma_a = \neg b$;
2. action b is allowed to occur due to $\gamma_b = \neg a$, assuming the satisfaction of $\gamma_b = \neg a$;
3. actions a and b are allowed to occur due to $\gamma_a = {}^{=}b$, assuming the satisfaction of $\gamma_a = {}^{=}b$.

The choice between the alternatives above is not quantified, in the sense that the probabilities of these alternatives are undefined. Consequently, the probability of the occurrence of each of the actions a and b can not be derived from probability associations $\pi_a(\neg b)$, $\pi_b(\neg a)$ and $\pi_b({}^{=}a)$, since the probabilities of the satisfaction of their corresponding conditions are unknown.

However, the probability that neither action a nor action b occurs can be determined. Figure 7.2 shows the reasoning used to determine this probability. We assume that any implementation of actions a and b has to consider the alternatives above in a particular order. p_1, p_2 and p_3 represent the probabilities with which alternatives 1, 2 and 3 are considered first, respectively, such that $p_1 + p_2 + p_3 = 1$. For example, a is allowed to occur due to $\gamma_a = \neg b$ with probability p_1, which is represented as $\gamma_a = \neg b : \mathrm{p}_1$ in Figure 7.2. In this case, action a actually occurs due to $\gamma_a = \neg b$ with probability $\pi_a(\neg b)$, which is represented as $a : \pi_a(\neg b)$, and does not occur due to $\gamma_a = \neg b$ with probability $1\text{-}\pi_a(\neg b)$, which is represented as $\text{-} : 1\text{-}\pi_a(\neg b)$, where '-' denotes that no action has occurred (yet). In case of $\text{-} : 1\text{-}\pi_a(\neg b)$ the choice is made that b is allowed to occur due to $\gamma_b = \neg a$ with probability p_{11}, or a is allowed to occur due to $\gamma_a = {}^{=}b$ with probability of p_{12}, such that $p_{11} + p_{12} = 1$. In case of $\gamma_b = \neg a : \mathrm{p}_{11}$, action b actually occurs due to $\gamma_b = \neg a$ with probability $\pi_b(\neg a)$, and does not occur due to $\gamma_b = \neg a$ with probability $1\text{-}\pi_b(\neg a)$. The case of $\gamma_a = {}^{=}b : 1$ leaves only the possibility that a occurs due to $\gamma_a = {}^{=}b$ with probability $\pi_a({}^{=}b)$ and does not occur due to $\gamma_a = {}^{=}b$ with probability $1\text{-}\pi_a({}^{=}b)$.

The probability that a and b both do not occur can be determined by eliminating probabilities p_x and p_{xy}, with $x \in \{1, 2, 3\}$ and $y \in \{1, 2\}$, in the following expression:

$$
\begin{aligned}
& p_1 \times (\,(1\text{-}\pi_a(\neg b)) \times (p_{11} \times (1\text{-}\pi_b(\neg a)) \times (1\text{-}\pi_a({}^{=}b))) + (p_{12} \times (1\text{-}\pi_a({}^{=}b)) \times (1\text{-}\pi_b(\neg a)))\,) + \\
& p_2 \times (\,(1\text{-}\pi_b(\neg a)) \times (p_{21} \times (1\text{-}\pi_a(\neg b)) \times (1\text{-}\pi_a({}^{=}b))) + (p_{22} \times (1\text{-}\pi_a({}^{=}b)) \times (1\text{-}\pi_a(\neg b)))\,) + \\
& p_3 \times (\,(1\text{-}\pi_a({}^{=}b)) \times (p_{31} \times (1\text{-}\pi_a(\neg b)) \times (1\text{-}\pi_b(\neg a))) + (p_{32} \times (1\text{-}\pi_b(\neg a)) \times (1\text{-}\pi_a(\neg b)))\,) \\
& = (1\text{-}\pi_a(\neg b)) \times (1\text{-}\pi_b(\neg a)) \times (1\text{-}\pi_a({}^{=}b)).
\end{aligned}
$$

The probability of the individual occurrences of actions a and b can not be determined, since they depend on probabilities p_x and p_{xy}. Therefore, the current interpretation of probability associations $\pi_a(\neg b)$, $\pi_b(\neg a)$ and $\pi_b({}^{=}a)$ does not allow one to specify the probabilities of individual occurrences of two-sided related actions. It only allows one to define the probability that neither a nor b occurs. This probability is complementary to the probability that at least action a or b occurs, which is equal to:

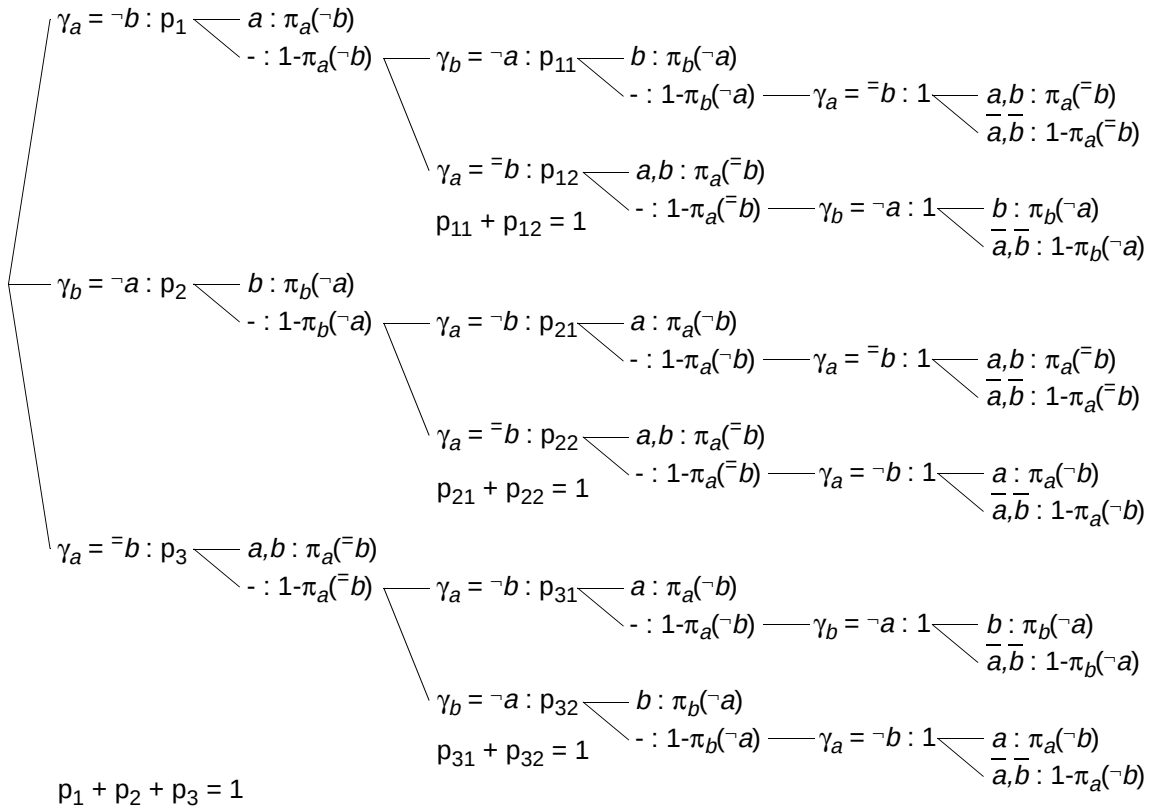


Figure 7.2: Simple probability associations of the temporal freedom relation

$$\pi_a(\neg b) + \pi_b(\neg a) + \pi_a(^{=}b) - \pi_a(\neg b) \times \pi_b(\neg a) - \pi_a(\neg b) \times \pi_a(^{=}b) - \pi_b(\neg a) \times \pi_a(^{=}b) + \pi_a(\neg b) \times \pi_b(\neg a) \times \pi_a(^{=}b).$$

The integral probability association $\pi_a(b)$ represents the probability that action a occurs once action b has occurred due to disabling condition $\neg a$. Once action b has occurred, the occurrence of a is only enabled by enabling condition $\gamma_a = b$. This implies that the conditional probability that a occurs due to enabling condition $\gamma_a = b$ can be considered independently of the conditional probability that at least a or b occurs due to the two-sided conditions $\gamma_a = \neg b$, $\gamma_b = \neg a$ and $\gamma_a = {}^{=}b$. An analogous reasoning applies to $\pi_b(a)$.

We can apply the results obtained above to any two-sided relation between actions a and b.

For example, consider the choice between actions a and b as defined by:
$\neg b \rightarrow a$ [$\pi_a(\neg b) = 0.7$],
$\neg a \rightarrow b$ [$\pi_b(\neg a) = 0.4$].
The probability that a or b occurs is: $\pi_a(\neg b) + \pi_b(\neg a) - \pi_a(\neg b) \times \pi_b(\neg a) = 0.82$.

For example, consider the synchronization of actions a and b as defined by:
${}^{=}b \rightarrow a$ [$\pi_a(^{=}b) = 0.6$],
${}^{=}a \rightarrow b$ [$\pi_b(^{=}a) = 0.6$].
The probability that a or b occurs is: $\pi_a(^{=}b) = \pi_b(^{=}a) = 0.6$.

For example, consider the interleaving of actions a and b as defined by:

$\neg b \vee b \rightarrow a$ [$\pi_a(\neg b) = 0.7$, $\pi_a(b) = 1$],

$\neg a \vee a \rightarrow b$ [$\pi_b(\neg a) = 0.4$, $\pi_b(a) = 0.8$].

The probability that a or b occurs is: $\pi_a(\neg b) + \pi_b(\neg a) - \pi_a(\neg b) \times \pi_b(\neg a) = 0.82$. This probability is completely determined by $\pi_a(\neg b)$ and $\pi_b(\neg a)$, since disabling conditions $\neg b$ and $\neg a$ define if and which action occurs at the beginning of the behaviour.

Decomposition of uncertainty

The scheme of Figure 7.2 illustrates a decomposition of the uncertainty of the occurrence of some action a into two distinct sources:

- the uncertainty caused by the choice between alternative two-sided relations in which action a is involved, as represented by probabilities p_x and p_{xy};
- the uncertainty caused by (the implementation of) action a itself, as represented by probability associations π_a.

The simple integral probability attribute defined in this section only allows one to quantify the latter source of uncertainty.

The notion of enabling

The notion of enabling has been used quite intuitively so far. We define that an alternative causality condition γ_a *enables* the occurrence of result action a when

- the enabling conditions in γ_a are satisfied, i.e., the enabling actions in γ_a have occurred;
- the disabling and synchronization conditions in γ_a may be satisfied, i.e., the disabling and synchronization actions in γ_a have not occurred yet.

Actions that are enabled by one or more alternative causality conditions are called *enabled actions*. The definition of the enabling notion implies the following rules:

1. action a is allowed to occur due to condition γ_a if and only if action a is enabled by condition γ_a, in case γ_a consists only of enabling conditions;
2. action a is not necessarily allowed to occur due to condition γ_a if action a is enabled by condition γ_a, in case γ_a contains disabling and synchronization conditions.

The enabling notion distinguishes between the satisfaction of one-sided conditions, i.e., enabling conditions, and the satisfaction two-sided conditions, i.e., disabling and synchronization conditions, allowing the occurrence of some action. This distinction is important in the elaboration of the execution semantics of the probability attribute.

In terms of the execution model, probability association $\pi_a(\gamma_a)$ is defined as the ratio between the probability of the executions in which condition γ_a is satisfied and result action a occurs, and the probability of the executions in which condition γ_a is satisfied and action a occurs or does not occur.

The satisfaction of an enabling condition γ_a of result action *a* is determined by the occurrence of the corresponding enabling action(s) in γ_a. Since action occurrences are represented in the execution model, the executions in which γ_a is satisfied can always be determined. This implies that the conditional probability of action *a* can be precisely defined in terms of the ratio above. Furthermore, the absolute probability of the occurrence of *a* can be determined, whenever the probability of the executions in which γ_a is satisfied is known.

The satisfaction of a two-sided condition γ_a of result action *a* is determined by a negotiation between action *a* and the disabling or synchronization action(s) in γ_a. The probabilities of the possible outcomes of this negotiation are not specified by the simple probability attribute. Consequently, the executions in which γ_a is satisfied can not be determined. This implies that the conditional probability of action *a* can not be defined in terms of the ratio above, neither can the absolute probability of the occurrence of *a* be determined. Given the current interpretation of the probability attribute, only the probability that at least one of a group of two-sided related actions occurs can be defined. This has been explained for the case of two actions above, and is discussed for multiple actions below.

7.2.2 Conjunctive causality conditions

The interpretation of integral probability associations of conjunctive causality conditions is defined as follows.

> The integral probability association $\pi_a(\gamma 1 \wedge \gamma 2)$ defines the integral probability that result action *a* occurs when assuming (sub-)conditions $\gamma 1$ and $\gamma 2$ are satisfied.

This definition is illustrated for two specific instances of $\gamma 1 \wedge \gamma 2$:

- $\pi_a(b \wedge \neg c)$ defines the integral probability that action *a* occurs when assuming that action *b* has occurred and action *c* does not occur before *a* nor simultaneously with *a*;
- $\pi_a(b \wedge {}^{=}c)$ defines the integral probability that action *a* occurs when assuming that action *b* has occurred and action *c* synchronizes with action *a*.

Example: enabling conditions

Figure 7.3 depicts a behaviour in which action *a* is allowed to occur when both actions *b* and *c* have occurred. The conditional probability that *a* occurs when assuming that *b* and *c* have occurred is larger than 0.1, i.e., when considering ‘infinitely many’ executions in which *b* and *c* occur, action *a* occurs in at least 10 percent of these executions. Since actions *b* and *c* are independent, the absolute probability that both *b* and *c* occur in the same execution is equal to $\Pi_{b \wedge c} = \pi_b(\surd) \times \pi_c(\surd)$. Consequently, the absolute probability that *a* occurs is equal to $\Pi_a = \Pi_{b \wedge c} \times \pi_a(b \wedge c)$, which renders $0.008 < \Pi_a < 0.4$; i.e., when considering ‘infinitely many’ executions, *a* occurs in at least 0.8 percent of these executions and in at most 40 percent of these executions.

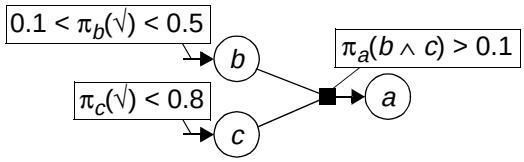


Figure 7.3: Enabling conditions

Example: choice

Figure 7.4 depicts the choice between actions *a* and *b* with common enabling action *c*. When assuming that *c* occurs, both actions *a* and *b* are enabled. The two-sided choice relation defines that either *a* or *b* is allowed to occur. This choice is equivalent to the choice relation between *a* and *b* as discussed in the previous section, with enabling condition *c* replacing the start condition. Consequently, only the conditional probability that at least *a* or *b* occurs can be determined, which is denoted as $\pi'_{a \vee b}(c)$. Conditional probability $\pi'_{a \vee b}(c)$ is equal to the complement of the probability that neither *a* nor *b* occurs, such that:

$$\begin{aligned}\pi'_{a \vee b}(c) &= 1 - (1-\pi_a(c \wedge \neg b)) \times (1-\pi_b(c \wedge \neg a)) \\ &= \pi_a(c \wedge \neg b) + \pi_b(c \wedge \neg a) - \pi_a(c \wedge \neg b) \times \pi_b(c \wedge \neg a) = 0.9\end{aligned}$$

The absolute probability that at least *a* or *b* occurs is defined as:

$$\Pi_{a \vee b} = \Pi_c \times \pi'_{a \vee b}(c) = 0.6 \times 0.9 = 0.54$$

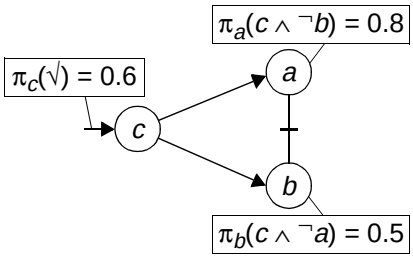


Figure 7.4: Choice

Enabling contexts

Figure 7.5 depicts the choice between actions *a* and *b* with distinct enabling actions *c* and *d*, respectively. In order to determine the conditional probability that at least *a* or *b* occurs, we have to make assumptions about whether the enabling actions of *a* and *b* occur or do not occur in an execution. Such an assumption is called an *enabling context*. For example, enabling context $\langle\{c\}, \{d\}\rangle$ represents the assumption that *c* occurs and *d* does not occur in an execution. Consequently, this enabling context represents all possible executions of the behaviour in Figure 7.4(ii) in which *c* occurs and *d* does not occur.

An enabling context may enable the occurrence of action *a* due to its alternative causality condition $c \wedge \neg b$, if (i) enabling action *c* is assumed to occur and (ii) disabling action *b* is assumed not to occur, or no assumption on the (non-)occurrence of *b* is made. An analogous rule applies to the enabling of action *b*. The restriction of the causality relation of an action to the alternative causality conditions that may enable this action given a certain enabling

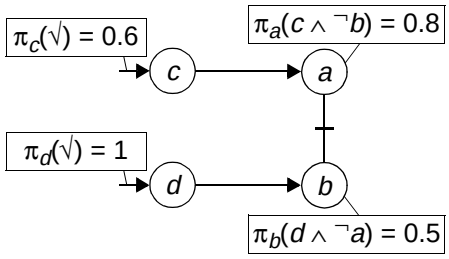


Figure 7.5: Choice with distinct enabling contexts

context, is called the *enabling part of the causality relation* of this action. For example, in case of enabling context $\langle\{c\}, \{d\}\rangle$, the enabling part of the causality relation of action a is $c \wedge \neg b$ $[\pi = 0.8] \rightarrow a$, and the enabling part of the causality relation of action b is empty, i.e., b can not occur. The *enabled behaviour* consists of the enabling part of each causality relation of the behaviour.

In Figure 7.5 two enabling contexts in which action a may be enabled by alternative causality condition $\gamma_a = c \wedge \neg b$ can be distinguished:

1. enabling context $\langle\{c, d\}, \varnothing\rangle$; and
2. enabling context $\langle\{c\}, \{d\}\rangle$.

The first enabling context represents that actions c and d occur. This implies that both actions a and b may be enabled by conditions $\gamma_a = c \wedge \neg b$ and $\gamma_b = d \wedge \neg b$, respectively, in the executions represented by this enabling context. Similarly to the case of a single enabling action, the choice relation between actions a and b determines which action is allowed to occur. Consequently, only the conditional probability that a or b happens given that c and d happen, which is denoted as $\pi_{a \vee b}'(c \wedge d)$, can be determined:

$$\pi_{a \vee b}'(c \wedge d) = \pi_a(c \wedge \neg b) + \pi_b(d \wedge \neg a) - \pi_a(c \wedge \neg b) \times \pi_b(d \wedge \neg a) = 0.9$$

π' is used to denote the conditional probability of (a disjunction of) one or more action occurrences, when assuming a certain enabling context. The prime indicates that these conditional probabilities are derived from probability attribute π.

The result above is independent of the relative time moments of the occurrences of c and d. This can be understood by considering a possible scenario as depicted in Figure 7.6. This scenario assumes that c occurs 3 time units before d, whereas a must occur within 5 time units after c occurs and b must occur 2 time units after d occurs.

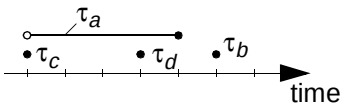


Figure 7.6: A scenario

After c has occurred, action a becomes enabled and the implementation decides between the following two alternative options:

1. action a is scheduled to occur within 3 time units after c in $\pi_a(c \wedge \neg b) \times 100$ percent of the executions. Subsequently, the following cases are distinguished:

- action d occurs before a occurs, such that action b becomes also enabled. In this case the implementation may still choose between the occurrence of a or b, i.e., either a remains scheduled, or a is de-scheduled and b is scheduled to occur 2 time units after d;
- action d occurs after a has occurred, such that action b does not become enabled, since the occurrence of a disables the occurrence of b.

Independent of the actual case, the probability that a or b occurs due to this option is equal to $\pi_a(c \wedge \neg b)$;

2. action a is not scheduled to occur in 100 - $\pi_a(c \wedge \neg b) \times 100$ percent of the executions. Subsequently, action d occurs and action b becomes enabled. Action b is scheduled to occur 2 time units after d in $\pi_b(d \wedge \neg b) \times 100$ percent of these executions. This implies the probability that b occurs due to this option is equal to $(1\text{-}\pi_a(c \wedge \neg b)) \times \pi_b(d \wedge \neg b)$.

The sum of the probabilities of both options renders the same value for $\pi_{a \vee b}'(c \wedge d)$ as defined above. The same probability value is obtained when interchanging the time moments of c and d, or when assuming different time intervals.

The second enabling context represents that action c occurs and action d does not occur. This implies that only action a is enabled in the executions represented by this enabling context, such that a is always allowed to occur. No conflict with action b is possible in this case. Consequently, the conditional probability that a occurs given that c happens and d does not happen, which is denoted as $\pi_a'(c \wedge \overline{d})$, can be determined:

$$\pi_a'(c \wedge \overline{d}) = \pi_a(c \wedge \neg b) = 0.8,$$

where $\overline{d}$ represents the assumption that d does not occur in an execution.

Analogously, conditional probabilities $\pi_{a \vee b}'(c \wedge d) = 0.9$ and $\pi_b'(\overline{c} \wedge d) = \pi_b(d \wedge \neg a) = 0.5$ can be determined by considering the probability contexts in which action b may be enabled by $\gamma_b = d \wedge \neg a$.

Since conditional probabilities $\pi_{a \vee b}'(c \wedge d)$, $\pi_a'(c \wedge \overline{d})$ and $\pi_b'(\overline{c} \wedge d)$ assume disjoint enabling contexts in which a or b is allowed to occur, the absolute probability that minimal a or b occurs is defined as:

$$\begin{aligned}\Pi_{a \vee b} &= \Pi_{c \wedge d} \times \pi_{a \vee b}'(c \wedge d) + \Pi_{c \wedge \overline{d}} \times \pi_{a \vee b}'(c \wedge \overline{d}) + \Pi_{\overline{c} \wedge d} \times \pi_{a \vee b}'(\overline{c} \wedge d) \\ &= \Pi_c \times \Pi_d \times \pi_{a \vee b}'(c \wedge d) + \Pi_c \times (1\text{-}\Pi_d) \times \pi_{a \vee b}'(c \wedge \overline{d}) + (1\text{-}\Pi_c) \times \Pi_d \times \pi_{a \vee b}'(\overline{c} \wedge d) \\ &= \Pi_c \times \pi_a(c \wedge \neg b) + \Pi_d \times \pi_b(d \wedge \neg a) - \Pi_{c \wedge d} \times \pi_a(c \wedge \neg b) \times \pi_b(d \wedge \neg a) \\ &= 0.48 + 0.5 - 0.24 = 0.74,\end{aligned}$$

with $\Pi_{c \wedge d} = \Pi_c \times \Pi_d$, $\Pi_c = \pi_c(\surd)$ and $\Pi_d = \pi_d(\surd)$, since actions c and d are independent.

Probability contexts

The notion of probability context is introduced to analyse the probability information in behaviour definitions. A *probability context pc* is defined as the combination of:

- an *enabling context EC*, which consists of action sets EA and $\overline{EA}$ representing the actions that are assumed to occur and the actions that are assumed not to occur in an execution, respectively. EA and $\overline{EA}$ are disjoint sets;

- an *enabled behaviour EB*, which consists of the enabling part of the causality relation of each action in the behaviour. The enabling part of a causality relation consists of the alternative causality conditions, and their probability associations, which may enable the result action when enabling context *EC* is assumed.

Enabling context $EC = \langle EA, \overline{EA} \rangle$ represents the executions χ of a behaviour for which the following holds: $EA \subseteq A_\chi$ and $\overline{EA} \subseteq \overline{A}_\chi$. Enabled behaviour *EB* represents the actions that may be executed next in these executions, when assuming that the actions in *EA* have occurred. These actions are denoted as the actions that may be enabled by enabling context *EC*.

An alternative causality condition $\gamma_b \in \Gamma_b$ may enable result action *b* given enabling context $\langle EA, \overline{EA} \rangle$, if (i) each enabling action in γ_b is contained in *EA*, (ii) each disabling action in γ_b is not contained in *EA*, and (iii) each synchronization action in γ_b is neither contained in *EA* nor in $\overline{EA}$. In case none of the alternative conditions in Γ_b may enable action *b*, the causality relation of this action is omitted from *EB*. The fact that an action may be enabled by multiple alternative causality conditions is reflected in the definition above. The examples in this section, however, are restricted to actions with a single alternative condition. Section 7.2.3 presents examples involving actions with multiple alternative causality conditions.

Conditional probabilities associated with a probability context

The probability information associated with a probability context $pc = \langle EC, EB \rangle$, with $EC = \langle EA, \overline{EA} \rangle$, depends on the actions that may be enabled in this probability context and the relationships between these actions. The following cases are distinguished:

1. *EB* is empty. In this case, *pc* can be discarded, since no action can be enabled given enabling context *EC*;
2. *EB* consists of a single action *a*, i.e., $EB = \{\gamma_a\ [\pi_a(\gamma_a)] \rightarrow a\}$. In this case, *pc* defines the conditional probability that *a* occurs when assuming that actions *EA* occur and actions $\overline{EA}$ do not occur in an execution. This probability is denoted as $\pi'_a(pc)$ and is defined as $\pi'_a(pc) = \pi_a(\gamma_a)$. Its complement is the conditional probability that *a* does not occur, which is defined as $\pi'_{\overline{a}}(pc) = 1 - \pi_a(\gamma_a)$.

   For clarity, the parameters of π'_a and $\pi'_{\overline{a}}$ denote the assumed probability context instead of the corresponding enabling context.

   For example, consider behaviour *B1* in Figure 7.7 and assume $pc = \langle \langle \{b\}, \varnothing \rangle, \{b \rightarrow a\ [\pi_a(b)]\} \rangle$. In this case *pc* defines the conditional probability $\pi'_a(pc) = \pi_a(b)$, representing the conditional probability that *a* occurs once *b* has occurred. For conciseness, the probability associations are omitted in Figure 7.7.
3. *EB* consists of multiple independent actions. In this case, the probability information associated with *pc* consists of $\pi'_a(pc)$ and $\pi'_{\overline{a}}(pc)$, as defined in case 2 above, for each action *a* in *EB*. Furthermore, the conditional probability of each combination of occurrences and non-occurrences of the actions in *EB* can be determined.

   For example, consider behaviour *B2* in Figure 7.7 and assume $pc = \langle \langle \{c\}, \varnothing \rangle, \{c \rightarrow a\ [\pi_a(c)],\ c \rightarrow b\ [\pi_b(c)]\} \rangle$. In this case *pc* defines the conditional probabilities $\pi'_a(pc) = \pi_a(c)$, $\pi'_b(pc) = \pi_b(c)$, $\pi'_{a \wedge b}(pc) = \pi_a(c) \times \pi_b(c)$, $\pi'_{a \wedge \overline{b}}(pc) = \pi_a(c) \times (1 - \pi_b(c))$, etc.

4. *EB* consists of multiple two-sided (probability) related actions. In this case, *pc* defines the conditional probability that at least one or none of these actions occur(s). This case is elaborated below.
5. A combination of cases 3 and 4 above. In this case, *pc* defines for each independent group of two-sided (probability) related actions, and their combinations, the conditional probability that at least one or none of the actions in each group occur(s).

   For example, consider behaviour *B3* in Figure 7.7 and assume $pc = \langle \langle \{d\}, \emptyset \rangle, \{d \wedge \neg b \rightarrow a\ [\pi_a(d \wedge \neg b)], d \wedge \neg a \rightarrow b\ [\pi_b(d \wedge \neg a)], d \rightarrow c\ [\pi_c(d)]\} \rangle$. In this case *pc* defines the conditional probabilities $\pi'_c(pc) = \pi_c(d)$, $\pi'_{a \vee b}(pc)$, $\pi'_{c \wedge (a \vee b)}(pc)$, $\pi'_{\bar{c} \wedge (a \vee b)}(pc)$, etc.

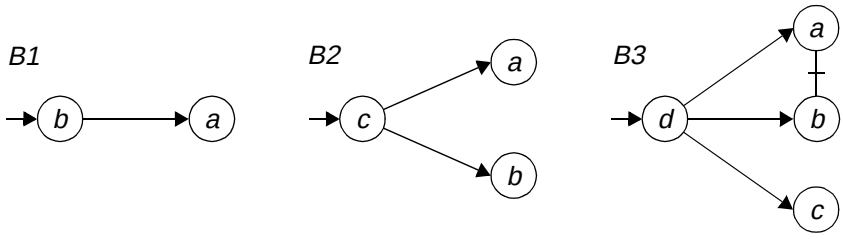


Figure 7.7: Example behaviours

Two-sided probability related actions

Whether an action *a* is allowed to occur due to the alternative condition in the enabling part of its causality relation depends on whether this condition defines a two-sided relation between action *a* and another action *b*. This action *b* may also be enabled within enabling context *EC*, such that this two-sided relation may represent a negotiation for the occurrences of actions *a* and *b* with an undetermined outcome. This implies that the conditional probabilities of the individual occurrences of actions *a* and *b* can not be determined. These actions are called *two-sided probability related*.

A two-sided relation may be defined between action *b* and a third action *c*, such that *c* may be enabled given the current enabling context, whereas *c* is not two-sided related with action *a*. In this case, the negotiations involving the occurrence of actions *a* and *b* and the occurrence of actions *b* and *c* are related. For example, assuming that a choice relation is defined between actions *a* and *b* and between actions *b* and *c*, the decision to allow the occurrence of *a* in the former negotiation implies that the latter negotiation can only decide to allow the occurrence of *c*. This implies that actions *a*, *b* and *c* are two-sided probability related, since neither the conditional probability of the individual occurrence of *a*, nor the ones of *b* and *c*, can be determined within the current enabling context.

Based on the above, two actions *a* and *b* are two-sided probability related assuming a certain enabling context, if (i) two alternative causality conditions γ_a and γ_b exist such that *a* and *b* may be enabled by γ_a and γ_b, respectively, and γ_a and γ_b define a two-sided relation between *a* and *b*, or (ii) a third action *c* exists such that *a* and *c* are two-sided probability related and *c* and *b* are two-sided probability related. The two-sided probability relation between actions is symmetric and transitive, in contrast to a two-sided relation which is symmetric but not necessarily transitive. Furthermore, a two-sided relation is stronger than a two-sided probability relation, since a two-sided relation implies a two-sided probability

relation, whereas a two-sided probability relation does not necessarily imply a two-sided relation.

Assuming that $pc = \langle EC, EB \rangle$ consists of multiple two-sided probability related actions $Ac(EB)$, the following complementary conditional probabilities are defined by pc:

$$\pi'_{\overline{a1} \wedge .. \wedge \overline{an}}(pc) = (1 - \pi_{a1}(\gamma_{a1})) \times .. \times (1 - \pi_{an}(\gamma_{an}));$$

$$\pi'_{a1 \vee .. \vee an}(pc) = 1 - \pi'_{\overline{a1} \wedge .. \wedge \overline{an}}(pc),$$

where $\{a1, .., an\} \subseteq Ac(EB)$, such that $\{a1, .., an\}$ is the maximal subset of $Ac(EB)$ that contains only one synchronized action in case EB contains multiple synchronized actions. This is explained below.

The negotiation for the occurrences of the actions in EB can be explained by a scheme similar to the one in Figure 7.2. None of the actions in EB occur, if the implementation mechanisms decide that each action does not occur when it is allowed to occur due to its alternative condition. The probability that the implementation mechanisms decide that action ai does not occur is equal to $1 - \pi_{ai}(\gamma_{ai})$. Since the implementation mechanisms may decide this independently for each action, and the order in which these decisions are made is irrelevant, the conditional probability that none of the actions in EB occur is equal to the product of $1 - \pi_{ai}(\gamma_{ai})$ for all actions $ai \in \{a1, .., an\}$.

In case EB contains multiple synchronized actions, only one of these actions is included in $\{a1, .., an\}$. This is because the probability association of only one of these synchronized actions should be considered in the formula above, due to the reciprocal aspect of synchronization conditions. Which synchronized action is included in $\{a1, .., an\}$ is irrelevant, since the values of the probability associations of all synchronized actions should be the same. The latter rule is a refinement of the corresponding rule for the uncertainty attribute in Section 5.4.

Example: choice between three actions

Figure 7.8(i) depicts a behaviour B representing a choice of one out of three between actions a, b and c. The following probability contexts are identified to determine all information on the conditional probability of actions a, b and c:

$$pc_1 = \langle\langle\{d, e\}, \emptyset\rangle, \{d \wedge \neg b \wedge \neg c \rightarrow a, e \wedge \neg a \wedge \neg c \rightarrow b, e \wedge \neg a \wedge \neg b \rightarrow c\}\rangle,$$

$$pc_2 = \langle\langle\{d\}, \{e\}\rangle, \{d \wedge \neg b \wedge \neg c \rightarrow a\}\rangle,$$

$$pc_3 = \langle\langle\{e\}, \{d\}\rangle, \{e \wedge \neg a \wedge \neg c \rightarrow b, e \wedge \neg a \wedge \neg b \rightarrow c\}\rangle.$$

For brevity, probability associations are omitted in the enabled behaviours.

The conditional probabilities associated with probability contexts pc_1, pc_2 and pc_3 are:

$$\pi'_{a \vee b \vee c}(pc_1) = \pi'_{a \vee b \vee c}(d \wedge e) = 1 - ((1-\pi_a(\gamma_a)) \times (1-\pi_b(\gamma_b)) \times (1-\pi_c(\gamma_c))) = 0.999,$$

$$\pi'_a(pc_2) = \pi'_a(d \wedge \overline{e}) = 1 - (1-\pi_a(\gamma_a)) = 0.9,$$

$$\pi'_{b \vee c}(pc_3) = \pi'_{b \vee c}(\overline{d} \wedge e) = 1 - ((1-\pi_b(\gamma_b)) \times (1-\pi_c(\gamma_c))) = 0.99$$

The absolute probability that either a, b or c occurs is derived as follows:

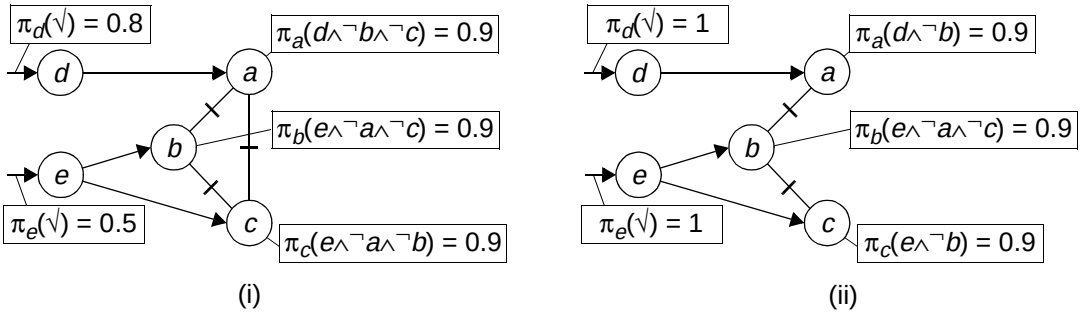


Figure 7.8: Choice between three actions

$$\Pi_{a \vee b \vee c} = \Pi_{d \wedge e} \times \pi'_{a \vee b \vee c}(d \wedge e) + \Pi_{d \wedge \overline{e}} \times \pi'_{a}(d \wedge \overline{e}) + \Pi_{\overline{d} \wedge e} \times \pi'_{b \vee c}(\overline{d} \wedge e),$$

with $\Pi_{d \wedge e} = \Pi_d \times \Pi_e$, $\Pi_{d \wedge \overline{e}} = \Pi_d \times (1 - \Pi_e)$, $\Pi_{\overline{d} \wedge e} = (1 - \Pi_d) \times \Pi_e$.

Figure 7.8(ii) depicts a choice between actions *a*, *b* and *c* which allows either action *b* to occur or allows actions *a* and *c* to occur independently. Despite *a* and *c* can occur independently of each other when assuming *d* and *e* occur, the conditional probabilities of the occurrences of both actions are indirectly related via action *b*, due to the choice relations between *a* and *b* and between *b* and *c*. This implies that actions *a*, *b* and *c* are two-sided probability related when it is assumed that *d* and *e* occur, such that only the probability that at least one of the actions *a*, *b* or *c* occurs can be determined.

The following probability contexts are distinguished to derive all information on the conditional probability of actions *a*, *b* and *c*:

$$pc_1 = \langle\langle\{d, e\}, \varnothing\rangle, \{d \wedge \neg b \rightarrow a, e \wedge \neg a \wedge \neg c \rightarrow b, e \wedge \neg b \rightarrow c\}\rangle,$$

$$pc_2 = \langle\langle\{d\}, \{e\}\rangle, \{d \wedge \neg b \rightarrow a\}\rangle,\ pc_3 = \langle\langle\{e\}, \{d\}\rangle, \{e \wedge \neg a \wedge \neg c \rightarrow b, e \wedge \neg b \rightarrow c\}\rangle,$$

$$pc_4 = \langle\langle\{d, e, c\}, \varnothing\rangle, \{d \wedge \neg b \rightarrow a\}\rangle,\ pc_5 = \langle\langle\{d, e, a\}, \varnothing\rangle, \{e \wedge \neg b \rightarrow c\}\rangle.$$

Absolute probability $\Pi_{a \vee b \vee c}$ is determined by probability contexts pc_1, pc_2 and pc_3, and is the same as in the previous example. Probability contexts pc_4 and pc_5 define additional probability information for the executions in which *a* and *c* are allowed to occur independently.

Example: consecutive choice

Figure 7.9 depicts a behaviour consisting of two disjoint groups of two-sided probability related actions: a group consisting of actions *a* and *b* and a group consisting of actions *c* and *d*. Therefore, the conditional probability that either *c* or *d* occurs can be determined independently of the conditional probability that either *a* or *b* occurs, i.e.:

$$\pi'_{c \vee d}(e) = 1 - (\,(1-\pi_c(e \wedge \neg d)) \times (1-\pi_d(e \wedge \neg c))\,) = 1,$$

$$\pi'_{a \vee b}(d) = 1 - (\,(1-\pi_a(d \wedge \neg b)) \times (1-\pi_b(d \wedge \neg a))\,) = 1 - 0.04 = 0.96$$

The absolute probability that either *c* or *d* occurs is equal to $\Pi_{c \vee d} = \Pi_e \times \pi'_{c \vee d}(e) = 1$. However, the absolute probability that either *a* or *b* occurs can not be determined, since the abso-

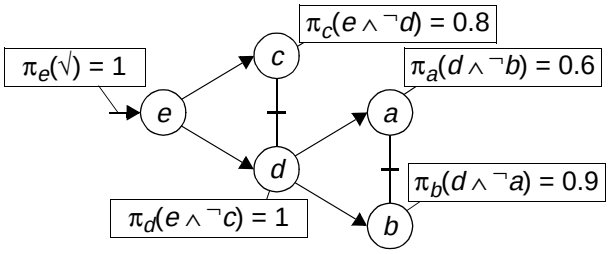


Figure 7.9: Consecutive choice

lute probability of the occurrence of *d* can not be determined using the simple probability attribute.

Example: synchronization

Figure 7.10 depicts a behaviour *B* consisting of the synchronization of actions *a* and *b*. Due to the identity property of the synchronization relation, enabling conditions *c* and *d* are conditions of both actions *a* and *b*. In other words, the indirect dependency closures of $\gamma_a = c \wedge {}^{=}b$ and $\gamma_b = d \wedge {}^{=}a$ are $\gamma_a{}^{+} = c \wedge d \wedge {}^{=}b$ and $\gamma_b{}^{+} = c \wedge d \wedge {}^{=}a$. The following probability context is identified to determine the conditional probability of actions *a* and *b*:

$$pc = \{\langle\langle\{c, d\}, \varnothing\rangle, \{c \wedge {}^{=}b \rightarrow a, d \wedge {}^{=}a \rightarrow b\}\rangle\}.$$

The associated conditional probability that at least *a* or *b* occurs is defined as:

$$\pi'_{a \vee b}(c \wedge d) = \pi_a(c \wedge {}^{=}b) = \pi_b(d \wedge {}^{=}a) = 0.8,$$

which is equivalent to the conditional probability that *a* and *b* synchronize. The absolute probability that *a* and *b* synchronize is equal to $\Pi_{c \wedge d} \times \pi'_{a \vee b}(c \wedge d) = 0.48$.

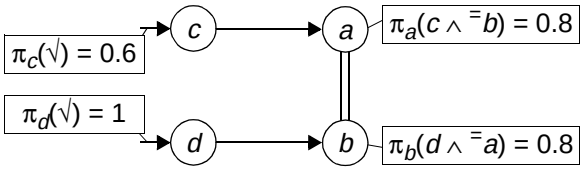


Figure 7.10: Synchronization

This example illustrates a specific case in which the conditional probability of individual action occurrences can be determined, despite that these actions are two-sided related. This is possible in case a probability context consists only of synchronized actions, since the conditional probability of all involved actions is the same in this case.

7.2.3 Disjunctive causality conditions

In case an action depends on multiple alternative causality conditions, the probability of the occurrence of this action is the same or larger when compared to the case in which this action depends on a subset of these conditions. Additional alternative causality conditions create more opportunities for the occurrence of an action.

An action may be enabled by multiple alternative causality conditions in a probability context *pc*. In order to determine the conditional probabilities associated with *pc* using the rules defined in Section 7.2.2, we have to determine, for each action *a* in *pc*, the probability that *a* does not occur when assuming that *a* is allowed to occur due to, possibly, multiple alternative causality conditions. This probability is determined by the relationship between the alternative causality conditions of *a*. This is illustrated by some examples below. The systematic analysis of these relationships falls outside the scope of this thesis.

Example: dependency on actions in choice

Figure 7.11 depicts a behaviour in which action *a* depends on the occurrence of action *b* or depends on the occurrence of action *c*, and actions *b* and *c* are in a choice relation. For brevity, we omit *d* in the discussion, since *d* is a necessary condition for *b* and *c*. The following probability contexts are identified to determine all information on the conditional probability of action *a*:

$pc_1 = \langle\langle\{b\}, \{c\}\rangle, \{b \rightarrow a\}\rangle$ and $pc_2 = \langle\langle\{c\}, \{b\}\rangle, \{c \rightarrow a\}\rangle$.

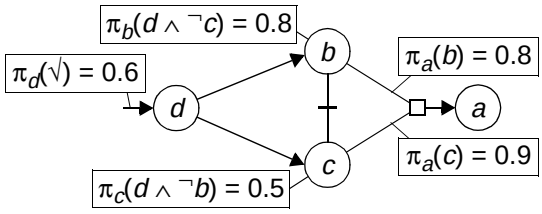


Figure 7.11: Dependency on actions in choice

The associated conditional probabilities are: $\pi'_a(b \wedge \overline{c}) = \pi_a(b)$ and $\pi'_a(\overline{b} \wedge c) = \pi_a(c)$. The absolute probability that *a* occurs is defined as:

$$\begin{aligned}\Pi_a &= \Pi_{b \wedge \overline{c}} \times \pi'_a(b \wedge \overline{c}) + \Pi_{\overline{b} \wedge c} \times \pi_a'(\overline{b} \wedge c)\\ &= \Pi_b \times \pi_a(b) + \Pi_c \times \pi_a(c).\end{aligned}$$

The value of Π_a can not be determined, since the values of the individual probabilities Π_b and Π_c can not be determined using the simple probability attribute.

Probability context $\langle\langle\{b, c\}, \varnothing\rangle, \{b \vee c \rightarrow a\}\rangle$ is invalid for this behaviour, since actions *b* and *c* can not occur in the same execution.

Example: dependency on independent actions

Figure 7.12 depicts a behaviour in which action *a* depends on the occurrence of action *b* or depends on the occurrence of action *c*, and actions *b* and *c* may occur independently. The probability contexts defining information on the conditional probability of action *a* are:

$pc_1 = \langle\langle\{b, c\}, \varnothing\rangle, \{b \vee c \rightarrow a\}\rangle$, $pc_2 = \langle\langle\{b\}, \{c\}\rangle, \{b \rightarrow a\}\rangle$ and $pc_3 = \langle\langle\{c\}, \{b\}\rangle, \{c \rightarrow a\}\rangle$.

The conditional probabilities $\pi'_a(b \wedge \overline{c}) = \pi_a(b)$ and $\pi'_a(\overline{b} \wedge c) = \pi_a(b)$ are associated with pc_2 and pc_3, respectively.

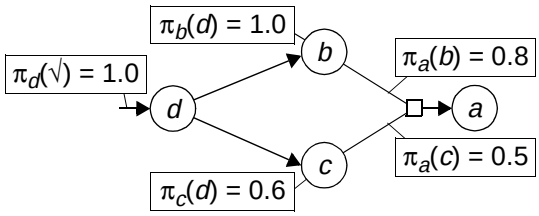


Figure 7.12: Dependency on independent actions

The conditional probability that a occurs (either due to b or due to c) when assuming that b and c occur is associated with pc_1. This probability is denoted as $\pi'_a(b \wedge c)$. The choice of the resulting causality condition of a is not specified by probability associations $\pi_a(b)$ and $\pi_a(c)$. Consequently, only the probability that a occurs can be determined, and no distinction can be made between the probability that a occurs due to b and the probability that a occurs due to c. Since actions b and c are independent, enabling conditions b and c can be satisfied independently. Therefore, probability associations $\pi_a(b)$ and $\pi_a(c)$ are also considered independently. This gives the following result:

$$\pi'_a(b \wedge c) = \pi_a(b) + \pi_a(c) - \pi_a(b) \times \pi_a(c) = 0.9$$

Based on the above, the absolute probability that a occurs can be derived as follows:

$$\begin{aligned}\Pi_a &= \Pi_{b \wedge \bar{c}} \times \pi'_a(b \wedge \bar{c}) + \Pi_{\bar{b} \wedge c} \times \pi'_a(\bar{b} \wedge c) + \Pi_{b \wedge c} \times \pi'_a(b \wedge c) \\ &= \Pi_b \times \pi_a(b) + \Pi_c \times \pi_a(c) - \Pi_{b \wedge c} \times \pi_a(b) \times \pi_a(c) = 0.86,\end{aligned}$$

with $\Pi_{b \wedge c} = \Pi_b \times \Pi_c$ and $\pi'_a(b \wedge c) = \pi_a(b) + \pi_a(c) - \pi_a(b) \times \pi_a(c)$.

Example: super- and sub-conditions

Figure 7.13 depicts behaviour B, which extends the behaviour in Figure 7.12 by allowing action a to occur also due to the occurrences of both b and c. The probability contexts that determine the conditional probability of action a are:

$$pc_1 = \langle\langle\{b, c\}, \varnothing\rangle, \{b \vee c \vee (b \wedge c) \rightarrow a\}\rangle,$$

$$pc_2 = \langle\langle\{b\}, \{c\}\rangle, \{b \rightarrow a\}\rangle \text{ and } pc_3 = \langle\langle\{c\}, \{b\}\rangle, \{c \rightarrow a\}\rangle.$$

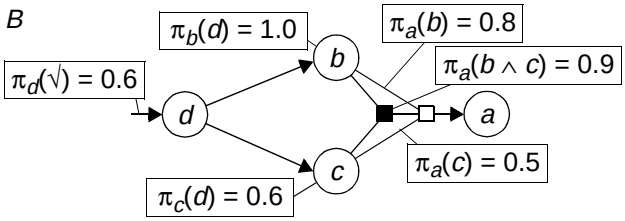


Figure 7.13: Super- and sub-conditions

Enabling conditions b and c are sub-conditions of enabling condition $b \wedge c$, i.e., $b \wedge c \vdash_B b$ and $b \wedge c \vdash_B c$, since the satisfaction of condition $b \wedge c$ implies the satisfaction of conditions b and c. However, the (independent) satisfaction of both b and c also implies the satisfaction of $b \wedge c$. Therefore, the corresponding probability associations must satisfy the following equation: $\pi_a(b \wedge c) = \pi_a(b) + \pi_a(c) - \pi_a(b) \times \pi_a(c)$. Furthermore, the probability that a occurs

when it is assumed that b and c occur is completely determined by $\pi_a(b \wedge c)$, such that the conditional probability associated with common probability context pc_1 is defined as:

$$\pi'_a(pc_1) = \pi'_a(b \wedge c) = \pi_a(b \wedge c).$$

Since $\pi'_a(b \wedge c)$ is the same as in the example of Figure 7.12, the absolute probability of the occurrence of a is also the same in this case.

7.2.4 Conclusion

The simple integral probability attribute allows one to model

- the conditional probability of individual action occurrences that depend only on enabling conditions; and
- the conditional probability of the occurrence of at least one action from a group of two-sided probability related actions.

The modelling of the conditional and absolute probability of individual action occurrences is mainly restricted to behaviours defining the sequential and independent composition of actions. For example, the simple probability attribute does allow one to model the conditional probability of individual action occurrences in two-sided relations, such as choice or disabling relations. Therefore, we conclude that the expressive power of the simple integral probability attribute is limited.

A complex set of rules is needed to obtain all probability information from a behaviour definition. In order to obtain this probability information, one should have (i) rules to identify all possible probability contexts in this behaviour and (ii) rules to determine the conditional probabilities associated with each probability context. Particularly, the systematic analysis of rules to determine the conditional probabilities of actions having multiple alternative causality conditions in a probability context is a complex task.

We do not develop any additional rules in this thesis to perform steps (i) and (ii) than the ones presented in Section 7.2.2, since the probability information that can be gained with these extra rules is of limited use as observed above. The rules discussed above (in this section) are sufficient to analyse the most common cases for which the use of the simple probability attribute is meaningful.

7.3 Formal definition (1)

This section defines the execution semantics of the conditional probabilities $\pi'_A(pc)$ and $\pi'_{\overline{A}}(pc)$ that are associated with probability context $pc = \langle\langle EA, \overline{EA}\rangle, EB\rangle$, where:

- $\pi'_{\overline{A}}(pc)$ represents the probability that none of the actions in A occur, with $A \subseteq Ac(EB)$ being a set of actions that may be enabled in pc as determined by the rules presented in Section 7.2.2 (see page 230);
- $\pi'_A(pc)$ represents the probability that at least one of the actions in A occurs, which is the complement of $\pi'_{\overline{A}}(pc)$, such that $\pi'_A(pc) = 1 - \pi'_{\overline{A}}(pc)$.

We assume $\pi'_A(pc)$ and $\pi'_{\overline{A}}(pc)$ are derived from behaviour B. The execution semantics of each of these conditional probabilities is defined in terms of the constraints they impose on probability function $\pi_E : \wp(E) \rightarrow \wp(\boldsymbol{P})$, where E represents the execution semantics of the causality condition part of behaviour B, i.e., $E = [\![B_\Gamma]\!]$. These constraints are represented in terms of a set of tuples $\langle E1, \pi_E(E1)\rangle$, where $E1$ represents a subset of E and $\pi_E(E1)$ represents the probability of the executions in $E1$. $\pi_E(E1)$ may be defined in terms of the probability of other execution sets. For example, the constraints imposed by equation $\pi_E(E1)/\pi_E(E2) = 0.8$ is represented as:

$\{ \langle E1, \pi_E(E2) \times 0.8\rangle, \langle E2, \pi_E(E1) / 0.8\rangle \}$.

Conditional probabilities $\pi'_A(pc)$ and $\pi'_{\overline{A}}(pc)$ constrain the probability of the following subsets of executions of behaviour B_Γ:

- $E_{pc} = \{\chi \mid \chi \in [\![B_\Gamma]\!] , AE \subseteq A_\chi, \overline{AE} \subseteq \overline{A}_\chi\}$, which defines the executions of behaviour B_Γ that satisfy the enabling context of pc;
- $E_{\overline{A}} = \{\chi \mid \chi \in E_{pc}, A \subseteq \overline{A}_\chi\}$, which defines the subset of E_{pc} in which none of the actions in A occur;
- $E_A = E_{pc} - E_{\overline{A}}$, which defines the subset of E_{pc} in which at least one of the actions in A occurs,

such that the following ratio's must hold:

$\pi'_{\overline{A}}(pc) = \pi_E(E_{\overline{A}}) / \pi_E(E_{pc})$; and

$\pi'_A(pc) = \pi_E(E_A) / \pi_E(E_{pc})$.

Definition 7.2 The execution semantics of the conditional probabilities $\pi'_A(pc)$ and $\pi'_{\overline{A}}(pc)$ that are derived from behaviour B, with $\pi'_A(pc) = 1 - \pi'_{\overline{A}}(pc)$, is defined as:

$[\![\pi'_{\overline{A}}(pc)]\!] = \{ \langle E_{\overline{A}}, \pi_E(E_{pc}) \times \pi'_{\overline{A}}(pc)\rangle, \langle E_{pc}, \pi_E(E_{\overline{A}}) / \pi'_{\overline{A}}(pc)\rangle \}$;

$[\![\pi'_A(pc)]\!] = \{ \langle E_A, \pi_E(E_{pc}) \times \pi'_A(pc)\rangle, \langle E_{pc}, \pi_E(E_A) / \pi'_A(pc)\rangle \}$,

with: $pc = \langle\langle EA, \overline{EA}\rangle, EB\rangle$,
$E = [\![B_\Gamma]\!]$,
$E_{pc} = \{\chi \mid \chi \in [\![B_\Gamma]\!] , AE \subseteq A_\chi, \overline{AE} \subseteq \overline{A}_\chi\}$,
$E_{\overline{A}} = \{\chi \mid \chi \in E_{pc}, A \subseteq \overline{A}_\chi\}$,
$E_A = E_{pc} - E_{\overline{A}}$. ■

Example: sequential composition

Figure 7.14 depicts behaviour B, which consists of the sequential composition of actions d, c and a and the sequential composition of actions d, c and b. Figure 7.14 also depicts the executions of B_Γ, with $E = [\![B_\Gamma]\!] = \{\chi_1, \chi_2, \chi_3, \chi_4, \chi_5, \chi_6\}$.

The probability of these executions is constrained by the equations below. These equations are generated by the conditional probabilities associated with the probability contexts that can be identified in behaviour B. For brevity, these probability contexts are omitted.

$\pi'_d(\surd) = \pi_d(\surd) = \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_4, \chi_5\}) / \pi_E(E)$;

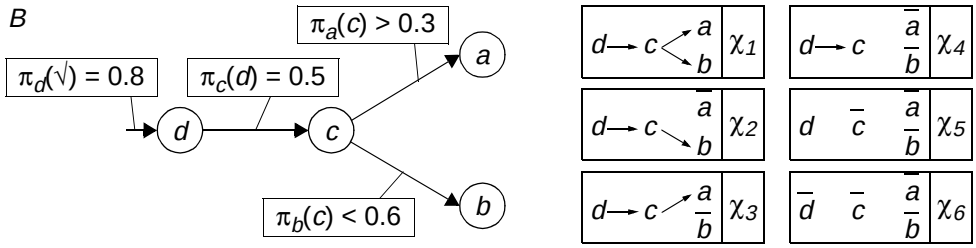


Figure 7.14: Sequential composition

$\pi'_c(d) = \pi_c(d) = \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_4\}) / \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_4, \chi_5\})$;

$\pi'_b(c \wedge d) = \pi_b(c) = \pi_E(\{\chi_1, \chi_2\}) / \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_4\})$;

$\pi'_a(c \wedge d) = \pi_a(c) = \pi_E(\{\chi_1, \chi_3\}) / \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_4\})$;

$\pi'_{a \wedge b}(c \wedge d) = \pi'_a(c \wedge d) \times \pi'_b(c \wedge d) = \pi_E(\{\chi_1\}) / \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_4\})$;

$\pi'_{\bar{a} \wedge b}(c \wedge d) = (1 - \pi'_a(c \wedge d)) \times \pi'_b(c \wedge d) = \pi_E(\{\chi_2\}) / \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_4\})$;

$\pi'_{a \wedge \bar{b}}(c \wedge d) = \pi'_a(c \wedge d) \times (1 - \pi'_b(c \wedge d)) = \pi_E(\{\chi_3\}) / \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_4\})$;

$\pi'_{\bar{a} \wedge \bar{b}}(c \wedge d) = (1 - \pi'_a(c \wedge d)) \times (1 - \pi'_b(c \wedge d)) = \pi_E(\{\chi_4\}) / \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_4\})$; and

$\pi_E(E) = 1$.

These equations render the following solution:

$\pi_E(\{\chi_6\}) = 0.2$; $\pi_E(\{\chi_5\}) = 0.4$;
$\pi_E(\{\chi_1, \chi_2\}) < 0.24$; $0.12 < \pi_E(\{\chi_1, \chi_3\}) \leq 0.4$;
$\pi_E(\{\chi_1\}) < 0.24$; $\pi_E(\{\chi_2\}) < 0.168$; $0.048 < \pi_E(\{\chi_3\}) \leq 0.4$; $\pi_E(\{\chi_4\}) \leq 0.28$;
with $\pi_E(\{\chi_1, \chi_2, \chi_3, \chi_4\}) = 0.4$

Example: disjunction of enabling conditions

Figure 7.15 depicts a behaviour B, in which action a depends on the disjunction of super-condition $b \wedge c$ and sub-conditions b and c. Figure 7.15 also shows the executions of B_Γ, with $E = [\![B_\Gamma]\!] = \{\chi_1, \chi_2, \chi_3, \chi_4, \chi_5, \chi_6, \chi_7, \chi_8, \chi_9\}$.

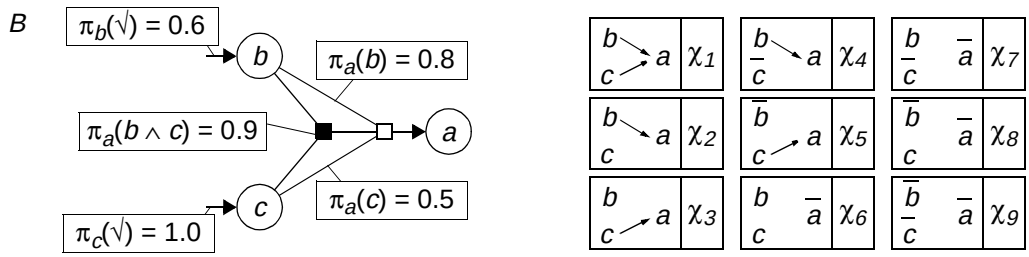


Figure 7.15: Disjunction of enabling conditions

The probability of these executions is constrained by the following equations:

$\pi'_b(\surd) = \pi_b(\surd) = \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_4, \chi_6, \chi_7\}) / \pi_E(E)$;

$\pi'_c(\surd) = \pi_c(\surd) = \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_5, \chi_6, \chi_8\}) / \pi_E(E)$;

$\pi'_{b \wedge c}(\surd) = \pi_b(\surd) \times \pi_c(\surd) = \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_6\}) / \pi_E(E)$;

$\pi'_{b \wedge \overline{c}}(\surd) = \pi_b(\surd) \times (1 - \pi_c(\surd)) = \pi_E(\{\chi_4, \chi_7\}) / \pi_E(E)$;

$\pi'_{\overline{b} \wedge c}(\surd) = (1 - \pi_b(\surd)) \times \pi_c(\surd) = \pi_E(\{\chi_5, \chi_8\}) / \pi_E(E)$;

$\pi'_{\overline{b} \wedge \overline{c}}(\surd) = (1 - \pi_b(\surd)) \times (1 - \pi_c(\surd)) = \pi_E(\{\chi_9\}) / \pi_E(E)$;

$\pi'_a(b \wedge c) = \pi_a(b \wedge c) = \pi_E(\{\chi_1, \chi_2, \chi_3\}) / \pi_E(\{\chi_1, \chi_2, \chi_3, \chi_6\})$;

$\pi'_a(b \wedge \overline{c}) = \pi_a(b) = \pi_E(\{\chi_4\}) / \pi_E(\{\chi_4, \chi_7\})$;

$\pi'_a(\overline{b} \wedge c) = \pi_a(c) = \pi_E(\{\chi_5\}) / \pi_E(\{\chi_5, \chi_8\})$; and

$\pi_E(E) = 1$.

These equations render the following solution:

$\pi_E(\{\chi_9\}) = 0$; $\pi_E(\{\chi_8\}) = 0.2$; $\pi_E(\{\chi_7\}) = 0$; $\pi_E(\{\chi_6\}) = 0.06$;
$\pi_E(\{\chi_5\}) = 0.2$; $\pi_E(\{\chi_4\}) = 0$; $\pi_E(\{\chi_1, \chi_2, \chi_3\}) = 0.54$

The probabilities of the individual executions χ_1, χ_2 and χ_3 can not be determined, since the probability of the choice of the resulting causality condition of action *a* is undefined. Furthermore, the probability context associated with the conditional probability $\pi'_a(b \wedge \overline{c})$ represents an empty set of executions, since executions χ_4 and χ_7 are impossible. Therefore, this probability context is invalid, and the corresponding conditional probability should be eliminated from the list above.

Absolute probabilities

The execution semantics of a behaviour *B* can be used to derive the absolute probability of the actions in this behaviour. The absolute probability of an action or group of actions is equal to the sum of the probability of the executions in which these actions occur. For example, in Figure 7.15 the absolute probability of the occurrence of action *a* can be calculated as follows:

$\Pi_a = \pi_E(\{\chi_1, \chi_2, \chi_3\}) + \pi_E(\{\chi_4\}) + \pi_E(\{\chi_5\}) = 0.74$.

Example: two-sided probability related actions

Figure 7.16 depicts a behaviour *B* consisting of actions *a*, *b*, *c* and *d*, such that actions *a*, *b* and *c* are two-sided probability related. Figure 7.16 also depicts the executions of behaviour B_Γ, with $E = [\![B_\Gamma]\!] = \{\chi_1, \chi_2, \chi_3, \chi_4\}$.

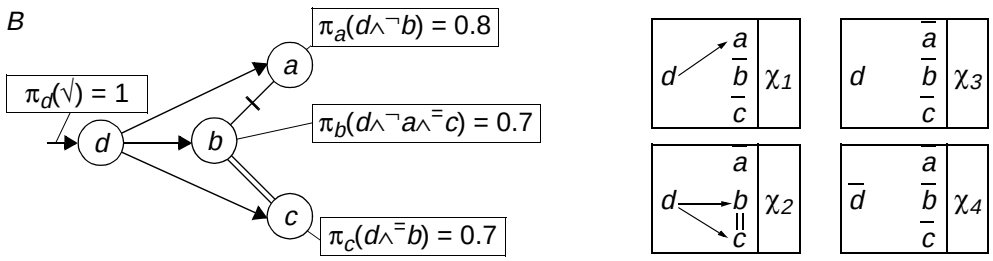


Figure 7.16: Two-sided probability related actions

The probability of these executions is constrained by the following equations:

$\pi'_d(\surd) = \pi_d(\surd) = \pi_E(\{\chi_1, \chi_2, \chi_3\}) / \pi_E(E)$;

$\pi'_{a \vee b \vee c}(d) = 1 - (1 - \pi_a(d \wedge \neg b)) \times (1 - \pi_b(d \wedge \neg b \wedge {}^{=}c)) = \pi_E(\{\chi_1, \chi_2\}) / \pi_E(\{\chi_1, \chi_2, \chi_3\})$;

$\pi_E(E) = 1$.

These equations render the following solution:

$\pi_E(\{\chi_4\}) = 0$; $\pi_E(\{\chi_3\}) = 0.06$; $\pi_E(\{\chi_1, \chi_2\}) = 0.94$.

The absolute probability of either the occurrence of *a* or the occurrences of *b* and *c* is derived as follows:

$\Pi_{a \vee (b \wedge c)} = \pi_E(\{\chi_1, \chi_2\}) = 0.94$.

7.4 Extended integral probability attribute

The simple integral probability attribute does not allow one to quantify the probability of the choice between alternative two-sided relations or alternative dynamic relations. Particularly in case of two-sided relations this is a severe limitation, since they comprise common behaviour patterns such as choice, disabling and interleaving.

This section presents the *extended integral probability attribute*, which extends the simple integral probability attribute to quantify the probability of the choice between alternative two-sided relations. We start by selecting the type of (sub-)behaviours to which this probability attribute can be applied. Subsequently, we define the interpretation of the extended probability attribute for elementary, conjunctive and disjunctive causality conditions, and we present consistency rules for the consistent definition of this probability attribute. We also discuss the application of the extended probability attribute to other behaviour types than the selected ones.

7.4.1 Selection

The application of the extended integral probability attribute is limited to groups of two-sided related actions, such that each group Ac_{two} obeys the following characteristics:

1. every action *a* in group Ac_{two} is two-sided related with every other action in this group;
2. every action *a* in group Ac_{two} depends on the same set of enabling actions outside this group, called Ac_{one}, by means of a conjunction of one-sided enabling relations.

Figure 7.17 represents a group of two-sided related actions Ac_{two} and its one-sided dependency on a group of actions Ac_{one}.

The motivation for limiting the application of the extended integral probability attribute is twofold:

- *complexity of consistency rules*. The first characteristic of group Ac_{two} defines that Ac_{two} neither contains independent actions nor alternative dynamic relations. The

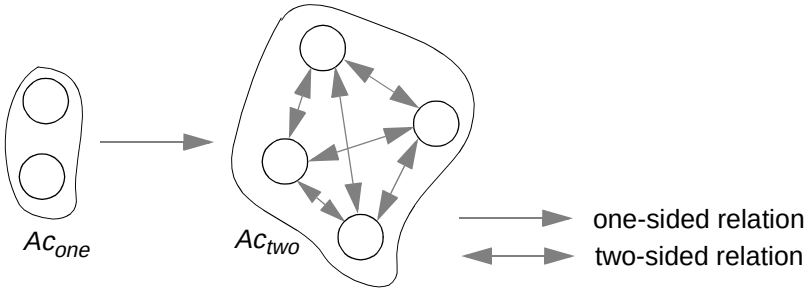


Figure 7.17: Group of two-sided related actions

application of the extended probability attribute to groups of two-sided related actions involving independent actions or dynamically related actions would significantly increase the complexity of consistency rules for this attribute. Furthermore, the application of the extended attribute to some dynamic relations is not always wanted. This is elaborated in Section 7.4.7.

- *correspondence with simple attribute*. The second characteristic of group Ac_{two} is a sufficient condition to satisfy a correspondence relation between the simple and extended integral probability attributes, in which the conditional probabilities defined by the simple attribute are preserved by the extended attribute. The satisfaction of this correspondence relation guarantees that a behaviour using the extended probability attribute defines more detailed probability information than the corresponding behaviour using the simple probability attribute. This correspondence relation is explained in Sections 7.4.2, 7.4.3 and 7.4.4.

7.4.2 Elementary causality conditions

The extended integral probability attribute is denoted by the symbol π^*. The interpretation of the extended probability association $\pi^*_a(\gamma_a)$, with $\gamma_a \in \{\surd, b, \neg b, {}^{=}b\}$ being an elementary causality condition of result action a, is defined as follows.

- $\pi^*_a(\surd)$ defines the integral probability that action a occurs due to the start condition, when assuming that this condition enables the occurrence of a. This corresponds to the probability that a occurs and the start condition is the resulting causality condition;
- $\pi^*_a(b)$ defines the integral probability that action a occurs due to enabling condition b, when assuming that this condition enables the occurrence of a. This corresponds to the probability that a occurs and is related to the occurrence of b, such that a occurs after b, when assuming that b occurs;
- $\pi^*_a(\neg b)$ defines the integral probability that action a occurs due to disabling condition $\neg b$, when assuming that this condition enables the occurrence of a. This corresponds to the probability that a occurs and is related to the (non-)occurrence of b, such that a occurs before b, or a occurs and b does not occur;
- $\pi^*_a({}^{=}b)$ defines the integral probability that action a occurs due to synchronization condition ${}^{=}b$, when assuming that this condition enables the occurrence of a. This corresponds to the probability that a occurs and is related to the occurrence of b, such that actions a and b synchronize.

The difference between the simple and extended interpretation is twofold:

1. the extended interpretation defines the conditional probability that result action a occurs *due to* a certain alternative causality condition γ_a, whereas the simple interpretation does not. This implies that the extended interpretation prescribes the probability that alternative causality condition γ_a becomes the resulting causality condition, when this condition enables the result action;
2. the extended interpretation assumes that some alternative causality condition γ_a enables result action a, whereas the simple interpretation assumes γ_a is satisfied. In the case of the start condition and enabling condition, both assumptions are equivalent. However, in the case of two-sided conditions, the extended interpretation defines implicitly the probability that these conditions are satisfied once they enable a, since $\pi^*_a(\gamma_a)$ prescribes the probability that γ_a is the resulting causality condition, when assuming that γ_a enables a.

We illustrate the extended probability attribute by means of the temporal freedom relation between two actions a and b. We consider this relation with the same goals as for the simple probability attribute.

Example: temporal freedom between two actions

Using the extended probability attribute, the temporal freedom relation between actions a and b is defined by causality relations:

$\neg b\ [\pi^*_a(\neg b)] \vee {}^{=}b\ [\pi^*_a({}^{=}b)] \vee b\ [\pi^*_a(b)] \rightarrow a,$

$\neg a\ [\pi^*_b(\neg a)] \vee {}^{=}a\ [\pi^*_b({}^{=}a)] \vee a\ [\pi^*_b(a)] \rightarrow b\ .$

Figure 7.18 depicts the probability of the executions allowed by the temporal freedom relation.

execution:	probability:
$\boxed{a \rightarrow b}$	$\pi^*_a(\neg b) \times \pi^*_b(a)$
$\boxed{b \rightarrow a}$	$\pi^*_b(\neg a) \times \pi^*_a(b)$
$\boxed{a = b}$	$\pi^*_b({}^{=}a) = \pi^*_a({}^{=}b)$
$\boxed{a \quad \bar{b}}$	$\pi^*_a(\neg b) \times (1 - \pi^*_b(a))$
$\boxed{\bar{a} \quad b}$	$\pi^*_b(\neg a) \times (1 - \pi^*_a(b))$
$\boxed{\bar{a} \quad \bar{b}}$	$\dfrac{1 - (\pi^*_a(\neg b) + \pi^*_b(\neg a) + \pi^*_a({}^{=}b))}{1}$ +

Figure 7.18: Extended probability associations of the temporal freedom relation

At the beginning of the execution of this behaviour, action a is enabled by conditions $\gamma_a = \neg b$ and $\gamma_a = {}^{=}b$, and action b is enabled by conditions $\gamma_b = \neg a$ and $\gamma_b = {}^{=}a$. Since only one of the conditions $\gamma_a = \neg b$, $\gamma_b = \neg a$ and $\gamma_a = {}^{=}b$ ($\gamma_b = {}^{=}a$) can be satisfied, i.e., either a can occur due to $\gamma_a = \neg b$, or b can occur due to $\gamma_b = \neg a$, or both a and b can occur due to $\gamma_a = {}^{=}b$ and $\gamma_b = {}^{=}a$, respectively, the corresponding extended probability associations must obey the following rule:

$$\pi^*_a(\neg b) + \pi^*_b(\neg a) + \pi^*_a({}^{=}b) \leq 1, \qquad \text{with } \pi^*_a({}^{=}b) = \pi^*_b({}^{=}a).$$

Extended probability associations $\pi^*_a(\neg b)$, $\pi^*_b(\neg a)$ and $\pi^*_a(\bar{\bar{b}})$ combine the corresponding simple probability associations $\pi_a(\neg b)$, $\pi_b(\neg a)$ and $\pi_a(\bar{\bar{b}})$, and the probabilities p_x and p_{xy} in Figure 7.2. Consequently, the extended probability attribute quantifies implicitly the choice between alternative two-sided relations.

In the case of the simple probability attribute, the following probability context determines the probability of the occurrence of *a* or *b*:

$$pc = \langle\langle\varnothing, \varnothing\rangle, \{\neg b\ [\pi^*_a(\neg b)] \vee \bar{\bar{b}}\ [\pi^*_a(\bar{\bar{b}})] \rightarrow a, \neg a\ [\pi^*_b(\neg a)] \vee \bar{\bar{a}}\ [\pi^*_b(\bar{\bar{a}})] \rightarrow b\},$$

The conditional probability $\pi'_{a \vee b}(\surd)$ is associated with *pc*. The extended probability attribute allows one to define this probability in more detail, by defining the probability that action *a* or *b*, or both, occur due to one of their alternative conditions in *pc*. Consequently, the following conditional probabilities are associated with *pc* in case of the extended probability attribute: $\pi^*_a(\neg b)$, $\pi^*_b(\neg a)$ and $\pi^*_a(\bar{\bar{b}})$.

The extended probability attribute preserves the probability information defined by the simple probability attribute, if both attributes define the same value for the probability that at least *a* or *b* occurs. The probability that at least *a* or *b* occurs in the current example is equal to $\pi^*_a(\neg b) + \pi^*_b(\neg a) + \pi^*_a(\bar{\bar{b}})$. Therefore, the following correspondence relation must hold:

$$\begin{aligned}
\pi^*_a(\neg b) + \pi^*_b(\neg a) + \pi^*_a(\bar{\bar{b}}) &= \pi'_{a \vee b}(\surd) \quad \Leftrightarrow \\
\pi^*_a(\neg b) + \pi^*_b(\neg a) + \pi^*_a(\bar{\bar{b}}) &= \pi_a(\neg b) + \pi_b(\neg a) + \pi_a(\bar{\bar{b}}) \\
&\quad - \pi_a(\neg b) \times \pi_b(\neg a) - \pi_a(\neg b) \times \pi_a(\bar{\bar{b}}) - \pi_b(\neg a) \times \pi_a(\bar{\bar{b}}) \\
&\quad + \pi_a(\neg b) \times \pi_b(\neg a) \times \pi_a(\bar{\bar{b}}).
\end{aligned}$$

This correspondence relation can be satisfied for any values of $\pi_a(\neg b)$, $\pi_b(\neg a)$ and $\pi_a(\bar{\bar{b}})$. This implies that the probability information in any definition of a temporal freedom relation using the simple probability attribute can be defined in more detail using the extended attribute.

The semantics of the simple probability associations of enabling conditions $\gamma_a = b$ and $\gamma_b = a$ is equivalent to the semantics of the corresponding extended probability associations. This is because the probability contexts in which *a* and *b* are enabled by $\gamma_a = b$ and $\gamma_b = a$, respectively, do not contain additional alternative conditions of actions *a* and *b*.

The probability of the executions of other two-sided relations between actions *a* and *b* can be obtained from the equations in Figure 7.18, by assuming that the value of an extended probability association is equal to zero when the corresponding alternative causality condition is absent.

7.4.3 Conjunctive causality conditions

The interpretation of extended integral probability associations of conjunctive causality conditions is defined as follows.

> The extended integral probability association $\pi^*_a(\gamma 1 \wedge \gamma 2)$ defines the integral probability that result action *a* occurs due to alternative causality condition $\gamma 1 \wedge \gamma 2$ when assuming (sub-)conditions $\gamma 1$ and $\gamma 2$ enable result action *a*.

This definition is illustrated for two specific instances of $\gamma 1 \wedge \gamma 2$:

- $\pi^*_a(b \wedge \neg c)$ defines the integral probability that action *a* occurs and is related to actions *b* and *c*, such that *a* occurs after *b* has occurred and *c* does not occur before nor simultaneously with *a*, when assuming that *b* occurs;
- $\pi^*_a(b \wedge {}^{=}c)$ defines the integral probability that action *a* occurs and is related to actions *b* and *c*, such that *a* occurs after *b* has occurred and *a* synchronizes with action *c*, when assuming that action *b* occurs.

Example: choice

Figure 7.19 depicts the choice between actions *a* and *b* with common enabling action *c*. This behaviour is identical to the one in Figure 7.4, except for using the extended probability attribute instead of the simple probability attribute and using different probability values. The following probability context in which *a* and *b* are enabled by *c* is identified:

$$pc = \langle\langle\{c\}, \varnothing\rangle, \{c \wedge \neg b\ [\pi^*_a(c \wedge \neg b)] \rightarrow a,\ c \wedge \neg a\ [\pi^*_b(c \wedge \neg a)] \rightarrow b\}\rangle.$$

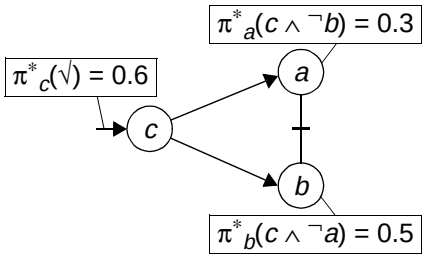


Figure 7.19: Choice (2)

The conditional probabilities associated with this probability context are $\pi^*_a(c \wedge \neg b)$ and $\pi^*_b(c \wedge \neg a)$, which define the probability of the choice between actions *a* and *b*, such that *a* and *b* occur in 30 and 50 percent of the executions in which *c* occurs, respectively. Their values must obey the consistency rule $\pi^*_a(c \wedge \neg b) + \pi^*_b(c \wedge \neg a) \leq 1$, which reflects that only one of two actions in a choice relation can occur.

The definition of $\pi^*_a(c \wedge \neg b)$ and $\pi^*_b(c \wedge \neg a)$ must satisfy the following correspondence relation, in order to preserve the probability information in the behaviour of Figure 7.4:

$$\pi'_{a \vee b}(c) = \pi^*_a(c \wedge \neg b) + \pi^*_b(c \wedge \neg a) \quad \Leftrightarrow$$
$$\pi_a(c \wedge \neg b) + \pi_b(c \wedge \neg a) - \pi_a(c \wedge \neg b) \times \pi_b(c \wedge \neg a) = \pi^*_a(c \wedge \neg b) + \pi^*_b(c \wedge \neg a).$$

For any values of $\pi_a(c \wedge \neg b)$ and $\pi_b(c \wedge \neg a)$, we can find values for $\pi^*_a(c \wedge \neg b)$ and $\pi^*_b(c \wedge \neg a)$ that satisfy the equation above, such that the behaviour in Figure 7.19 defines more detailed probability information than the behaviour in Figure 7.4.

Example: choice between three actions

Figure 7.20(i) depicts a behaviour *B* representing a choice of one out of three between actions *a*, *b* and *c*. The following probability context in which *a*, *b* and *c* are enabled by *d* is identified:

$pc = \langle\langle\{d\}, \varnothing\rangle, \{d \wedge \neg b \wedge \neg c \rightarrow a, d \wedge \neg a \wedge \neg c \rightarrow b, d \wedge \neg a \wedge \neg b \rightarrow c\}\rangle.$

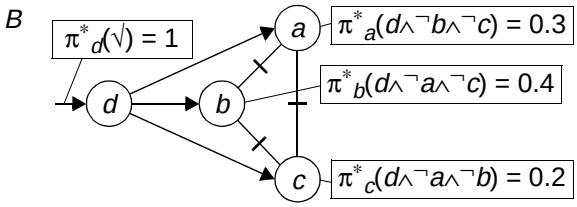


Figure 7.20: Choice between three actions (2)

The associated conditional probabilities $\pi^*_a(d \wedge \neg b \wedge \neg c)$, $\pi^*_b(d \wedge \neg a \wedge \neg c)$ and $\pi^*_c(d \wedge \neg a \wedge \neg b)$ define the probability of the choice between actions *a*, *b* and *c*. The summation of their values must be smaller than or equal to 1, since at most one action can occur. The following correspondence relation must be obeyed to obtain a corresponding behaviour with the simple integral probability attribute:

$\pi'_{a \vee b \vee c}(d) = \pi^*_a(d \wedge \neg b \wedge \neg c) + \pi^*_b(d \wedge \neg a \wedge \neg c) + \pi^*_c(d \wedge \neg a \wedge \neg b),$
with $\pi^*_a(d \wedge \neg b \wedge \neg c) + \pi^*_b(d \wedge \neg a \wedge \neg c) + \pi^*_c(d \wedge \neg a \wedge \neg b) \leq 1$

Consequently, for any allowed values of $\pi^*_a(d \wedge \neg b \wedge \neg c)$, $\pi^*_b(d \wedge \neg a \wedge \neg c)$ and $\pi^*_c(d \wedge \neg a \wedge \neg b)$, a corresponding value for $\pi'_{a \vee b \vee c}(d)$ can be found.

Example: independent groups of two-sided related actions

Figure 7.21 depicts a behaviour *B* consisting of two independent groups of two-sided related actions with common enabling action *e*: (i) the choice relation between actions *a* and *b* and (ii) the synchronization relation between actions *c* and *d*. Two probability contexts in which actions *a* and *b*, and actions *c* and *d* are enabled by *e*, respectively, are identified:

$pc_1 = \langle\langle\{e\}, \varnothing\rangle, \{e \wedge \neg b \rightarrow a, e \wedge \neg a \rightarrow b\}\rangle$ and $pc_2 = \langle\langle\{e\}, \varnothing\rangle, \{e \wedge {}^{=}d \rightarrow c, e \wedge {}^{=}c \rightarrow d\}\rangle.$

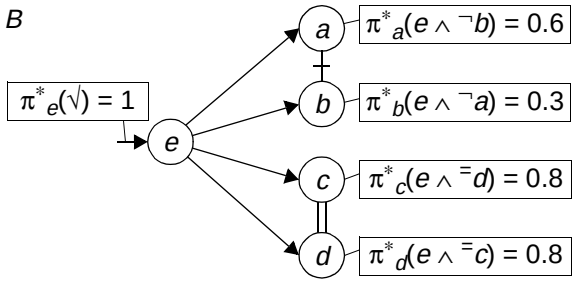


Figure 7.21: Independent groups of two-sided related actions

The conditional probabilities $\pi^*_a(e \wedge \neg b)$ and $\pi^*_b(e \wedge \neg a)$ are associated with pc_1. These probabilities define the probability of the occurrence of *a* and the probability of the occurrence of *b* after *e* has occurred, respectively, and are independent of the conditional probability of the occurrences of *c* and *d*. The reciprocal conditional probabilities $\pi^*_c(e \wedge {}^{=}d)$ and $\pi^*_d(e \wedge {}^{=}c)$ are associated with pc_2. These probabilities define the probability of the synchronized occurrences of *c* and *d* after *e* has occurred, and are independent of the conditional probability of the occurrences of *a* and *b*. The values of the extended probability associations of actions *a*, *b*, *c* and *d* must obey the following consistency rules:

$\pi^*_a(e \wedge \neg b) + \pi^*_b(e \wedge \neg a) \leq 1$ and $\pi^*_c(e \wedge {}^{=}d) = \pi^*_d(e \wedge {}^{=}c) \leq 1$.

Based on the above, the following correspondence relations must hold between this behaviour and the corresponding behaviour with the simple probability attribute:

$\pi'_{a \vee b}(e) = \pi^*_a(e \wedge \neg b) + \pi^*_b(e \wedge \neg a) = 0.9$;

$\pi'_{c \vee d}(e) = \pi'_c(e) = \pi'_d(e) = \pi^*_c(e \wedge {}^{=}d) = \pi^*_d(e \wedge {}^{=}c) = 0.8$;

$\pi'_{(a \vee b) \wedge (c \vee d)}(e) = (\pi^*_a(e \wedge \neg b) + \pi^*_b(e \wedge \neg a)) \times \pi^*_c(e \wedge {}^{=}d) = 0.72$;

$\pi'_{(a \vee b) \wedge (\overline{c} \wedge \overline{d})}(e) = (\pi^*_a(e \wedge \neg b) + \pi^*_b(e \wedge \neg a)) \times (1 - \pi^*_c(e \wedge {}^{=}d))\ 0.18$;

$\pi'_{(\overline{a} \wedge \overline{b}) \wedge (c \vee d)}(e) = (1 - \pi^*_a(e \wedge \neg b) - \pi^*_b(e \wedge \neg a)) \times \pi^*_c(e \wedge {}^{=}d) = 0.08$;

$\pi'_{\overline{a} \wedge \overline{b} \wedge (\overline{c} \wedge \overline{d})}(e) = (1 - \pi^*_a(e \wedge \neg b) - \pi^*_b(e \wedge \neg a)) \times (1 - \pi^*_c(e \wedge {}^{=}d)) = 0.02$

The conditional probability of the occurrences of two independent actions *x* and *z* due to alternative causality conditions γ_x and γ_z, respectively, which depend on the same enabling actions, is defined by the product of the values of $\pi^*_x(\gamma_x)$ and $\pi^*_z(\gamma_z)$. In case of the probability of the non-occurrences of *a* and *b*, these values should be replaced by $1 - \pi^*_x(\gamma_x)$ and $1 - \pi^*_z(\gamma_z)$, respectively.

Example: choice with distinct enabling contexts

Figure 7.22 depicts the choice between actions *a* and *b*, which depend on enabling actions *c* and *d*, respectively. This behaviour does not comply to the limitations introduced in Section 7.4.1.

Consider the probability contexts that are identified in the corresponding example in Figure 7.5, and the associated conditional probabilities $\pi'_{a \vee b}(c \wedge d)$, $\pi'_a(c \wedge \overline{d})$ and $\pi'_b(\overline{c} \wedge d)$. The definition of $\pi^*_a(c \wedge \neg b)$ and $\pi^*_b(d \wedge \neg a)$ should satisfy the following correspondence relations in order to preserve the probability information in Figure 7.5:

$$\begin{aligned}
\pi'_{a \vee b}(c \wedge d) &= \pi^*_a(c \wedge \neg b) + \pi^*_b(d \wedge \neg a),\\
\pi'_a(c \wedge \overline{d}) &= \pi^*_a(c \wedge \neg b),\\
\pi'_b(\overline{c} \wedge d) &= \pi^*_b(d \wedge \neg a),
\end{aligned}$$

which is equivalent to:

$$\begin{aligned}
&\pi_a(c \wedge \neg b) + \pi_b(d \wedge \neg a) - \pi_a(c \wedge \neg b) \times \pi_b(d \wedge \neg a) = \pi^*_a(c \wedge \neg b) + \pi^*_b(d \wedge \neg a),\\
&\pi_a(c \wedge \neg b) = \pi^*_a(c \wedge \neg b),\\
&\pi_b(d \wedge \neg a) = \pi^*_b(d \wedge \neg a).
\end{aligned}$$

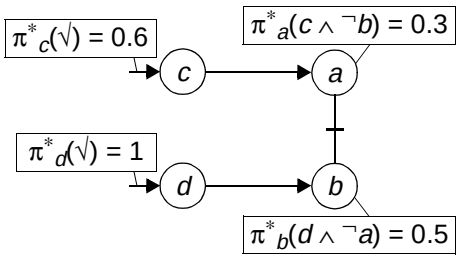


Figure 7.22: Choice with distinct enabling contexts (2)

The correspondence relations above can not be satisfied for $\pi_a(c \wedge \neg b) \neq 0$ and $\pi_b(d \wedge \neg a) \neq 0$. This implies that the probability information defined by $\pi'_{a \vee b}(c \wedge d)$, $\pi'_a(c \wedge \overline{d})$ and $\pi'_b(\overline{c} \wedge d)$ can not be preserved when using the extended probability attribute. For example, using the extended probability attribute, it is impossible to define that at least action *a* or *b* must occur when action *c* has occurred, since the definition of $\pi^*_a(c \wedge \neg b) = 1$ would imply that $\pi^*_b(d \wedge \neg a) = 0$ (and vice versa), such that action *b* (or *a*) can never occur.

In this example, the correspondence relations can not be satisfied, because actions *a* and *b* have distinct enabling contexts, in which the simple and extended probability associations are related differently. In general, when a group of actions related by (alternative) choice relations has distinct enabling contexts, the correspondence between the simple and extended probability attribute may not be found. A group of two-sided related actions has distinct enabling contexts in case they depend on distinct enabling actions. This illustrates the need for the second characteristic of group Ac_{two} as defined in Section 7.4.1. This example also illustrates that the extended probability attribute is not a refinement of the simple probability attribute. Instead, both attributes may be considered as orthogonal, which can satisfy the correspondence relations for certain type of behaviours, particularly for groups of two-sided related actions.

Usage of probability contexts

The (execution) semantics of an extended probability association $\pi^*_a(\gamma_a)$ is independent of any other extended probability association, since $\pi^*_a(\gamma_a)$ defines the conditional probability that action *a* occurs *due to* γ_a. Therefore, probability contexts are not necessary when defining the semantics of $\pi^*_a(\gamma_a)$; see also Section 7.5. Nonetheless, probability contexts are useful to define:

- consistency rules for the values of extended probability associations;
- correspondence relations between the simple and extended probability attributes.

The definition of consistency rules and correspondence relations is elaborated below for actions having a single alternative causality condition. Actions having multiple alternative causality conditions are considered in Section 7.4.4.

Consistency rules

Consider a group of two-sided related actions Ac_{two} and a probability context $pc = \langle EC, EB \rangle$, which consists of a subset of the actions in Ac_{two}, i.e., $Ac(EB) \subseteq Ac_{two}$. Each pair of actions in $Ac(EB)$ is related via either a choice or a synchronization relation, and *EB* defines the conjunction of these relations.

The above implies that only one alternative causality condition in *EB* can be satisfied, when considering reciprocal synchronization conditions as a whole, i.e., as a single condition. This can be understood as follows. Synchronized actions either occur all or none of them occur. When all synchronized actions in *EB* are replaced by a single action, the new behaviour *EB*' consists of a choice between one or more actions, such that only one of these actions can occur. This implies that only the alternative causality condition of one of these

actions can be satisfied. Consequently, in *EB* either a single alternative causality condition of some action *a* can be satisfied when this action is related to all other actions in *EB* via a choice relation, or the alternative causality conditions of a group of synchronized actions can be satisfied when this group is related to all other actions in *EB* via a choice relation. The latter alternative causality conditions are reciprocal synchronization conditions. When considering these reciprocal conditions as a whole, only a single alternative causality condition in *EB* can be satisfied.

Based on the above, the following consistency rule is imposed by *pc*:

$\Sigma\{\pi^*_a(\gamma_a) \mid a \in Ac(EB')\} \leq 1,$

where

- *EB'* is obtained from *EB* by eliminating reciprocal synchronization conditions such that only one synchronization condition remains. This implies that all synchronized actions are represented by a single action in *EB'*;
- γ_a is the alternative causality condition of *a* in *EB'*.

Correspondence relations

The summation in the consistency rule above defines the probability that at least one of the actions in *Ac*(*EB*) occurs when assuming enabling context *EC*. This implies that the following correspondence relation is associated with probability context *pc*:

$\pi'_{some}(pc) = \Sigma\{\pi^*_a(\gamma_a) \mid a \in Ac(EB')\},$

where

- *EB'* and γ_a are defined above; and
- $\pi'_{some}(pc)$ represents the conditional probability that at least one of the actions in *Ac*(*EB*) occurs.

7.4.4 Disjunctive causality conditions

Actions in a group of two-sided related actions may have multiple alternative causality conditions. Figure 7.23 illustrates this by means of behaviour *B*, which consists of the conjunction of a temporal freedom relation between actions *a* and *b*, and two disabling relations between actions *a* and *c* and actions *b* and *c*. Actions *a* and *b* are allowed to occur in any order, or simultaneously, as long as *c* has not occurred yet. The occurrence of *c* disables the occurrences of *a* and *b*, unless *a* and *b* have occurred before *c*. Figure 7.23 uses the graphical shorthand notations of the disabling relation and the temporal freedom relation, which can be found in Figure 4.15(iii) and (v), respectively.

In case an action has multiple extended probability associations, the execution semantics of these extended probability associations can be determined independently of each other, since each association defines the conditional probability that the action occurs due to a particular alternative causality condition. Consequently, the interpretation of the extended probability attribute for disjunctive causality conditions directly follows from the interpre-

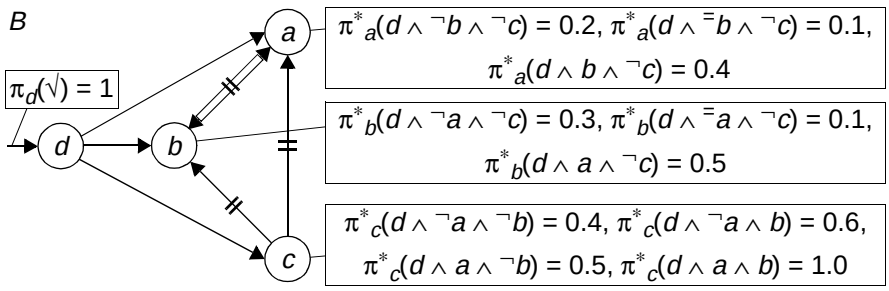


Figure 7.23: π^* applied to disjunctive causality conditions

tation for conjunctive causality conditions. The formulas of the consistency rule and correspondence relation presented in the previous section have to be reconsidered, however, in case actions may be enabled by multiple alternative causality relations.

The following probability contexts are distinguished in behaviour *B* of Figure 7.23:

$$pc_1 = \langle\langle\{d\}, \varnothing\rangle, \quad \{ \; (d \wedge \neg b \wedge \neg c) \vee (d \wedge {}^{=}b \wedge \neg c) \rightarrow a, \; (d \wedge \neg a \wedge \neg c) \vee (d \wedge {}^{=}a \wedge \neg c) \rightarrow b, \; d \wedge \neg a \wedge \neg b \rightarrow c \; \} \rangle ;$$

$$pc_2 = \langle\langle\{d, a\}, \varnothing\rangle, \quad \{ \; d \wedge a \wedge \neg c \rightarrow b, \; d \wedge a \wedge \neg b \rightarrow c \; \} \rangle ;$$

$$pc_3 = \langle\langle\{d, b\}, \varnothing\rangle, \quad \{ \; d \wedge b \wedge \neg c \rightarrow a, \; d \wedge \neg a \wedge b \rightarrow c \; \} \rangle ;$$

$$pc_4 = \langle\langle\{d, a, b\}, \varnothing\rangle, \; \{ \; d \wedge a \wedge b \rightarrow c \; \} \rangle.$$

Despite that an action may be enabled by multiple alternative causality conditions in a probability context, only one alternative causality condition can be satisfied (when considering reciprocal synchronizations as a whole), since any action is two-sided related with any other action in this context. This implies that the conditional probability of the occurrence of some action *a* in a certain probability context *pc* is defined by the summation of the extended probability association values of the alternative causality conditions by which *a* may be enabled in *pc*. Consequently, the following consistency rules are associated with pc_1, pc_2, pc_3 and pc_4, respectively:

$$\pi^*_a(d \wedge \neg b \wedge \neg c) + \pi^*_a(d \wedge {}^{=}b \wedge \neg c) + \pi^*_b(d \wedge \neg a \wedge \neg c) + \pi^*_c(d \wedge \neg a \wedge \neg b) \leq 1;$$

$$\pi^*_b(d \wedge a \wedge \neg c) + \pi^*_c(d \wedge a \wedge \neg b) \leq 1;$$

$$\pi^*_a(d \wedge b \wedge \neg c) + \pi^*_c(d \wedge \neg a \wedge b) \leq 1;$$

$$\pi^*_c(d \wedge a \wedge b) \leq 1.$$

Since the summations above define the probability that at least one action occurs in the corresponding probability context, the following correspondence relations are identified between behaviour *B* in Figure 7.23 and the corresponding behaviour with the simple probability attribute:

$$\pi'_{some}(pc_1) = \pi^*_a(d \wedge \neg b \wedge \neg c) + \pi^*_a(d \wedge {}^{=}b \wedge \neg c) + \pi^*_b(d \wedge \neg a \wedge \neg c) + \pi^*_c(d \wedge \neg a \wedge \neg b);$$

$$\pi'_{some}(pc_2) = \pi^*_b(d \wedge a \wedge \neg c) + \pi^*_c(d \wedge a \wedge \neg b);$$

$$\pi'_{some}(pc_3) = \pi^*_a(d \wedge b \wedge \neg c) + \pi^*_c(d \wedge \neg a \wedge b);$$

$\pi'_{some}(pc_4) = \pi^*_c(d \wedge a \wedge b)$.

Consistency rules and correspondence relations

In order to generalize the consistency rule and correspondence relation defined in Section 7.4.3, the summation in the corresponding formulas should include all (i.e., possibly multiple) extended probability associations of an action in *pc*, when considering reciprocal synchronizations as a whole.

The consistency rule imposed by *pc* is defined as:

$\Sigma\{\pi^*_a(\gamma) \mid a \in Ac(EB'), \gamma \in \Gamma_a\} \leq 1$,

where

- *EB'* is obtained from *EB* by eliminating reciprocal synchronization conditions such that a group of reciprocal synchronization conditions is represented by one synchronization condition. This may imply that some synchronized actions are removed from *EB'*;
- Γ_a is the causality condition of *a* in *EB'*.

The correspondence relation imposed by *pc* is defined as:

$\pi'_{some}(pc) = \Sigma\{\pi^*_a(\gamma) \mid a \in Ac(EB'), \gamma \in \Gamma_a\}$,

where

- *EB'* and Γ_a are defined above; and
- $\pi'_{some}(pc)$ represents the conditional probability that at least one of the actions in *Ac*(*EB*) occurs.

7.4.5 Additional consistency rules

Two additional consistency rules for extended probability associations are distinguished.

Extended probability associations of synchronized actions

The following rule defines the relation between extended probability associations of reciprocal synchronization conditions.

When assuming γ_a and γ_b are alternative causality conditions of actions *a* and *b*, which can be combined in an alternative behaviour, probability associations $\pi^*_a(\gamma_a)$ and $\pi^*_b(\gamma_b)$ must obey the following rule:

if $\gamma_a = {}^{=}b \wedge \gamma 1$ and $\gamma_b = {}^{=}a \wedge \gamma 2$ then $\pi^*_a(\gamma_a) = \pi^*_b(\gamma_b)$.

Combination of simple and extended probability attribute

The simple and extended probability attribute can be combined in a single behaviour definition, when the following rules are obeyed:

1. the use of the extended probability attribute is restricted to groups of two-sided related actions;
2. either the simple or the extended probability attribute must be used in a group of two-sided related actions, but not both.

Figure 7.24 depicts a behaviour *B*, in which both the simple and extended probability attribute are used. The executions of this behaviour that have a probability larger than zero are also shown in Figure 7.24. The probabilities of these executions are, with $E = [\![B_\Gamma]\!]$:

$$\pi_E(\chi_1) = 0.48;\ \pi_E(\chi_2) = 0.16;\ \pi_E(\chi_3) = 0.12;\ \pi_E(\chi_4) = 0.16;\ \pi_E(\chi_5) = 0.04;\ \pi_E(\chi_6) = 0.04.$$

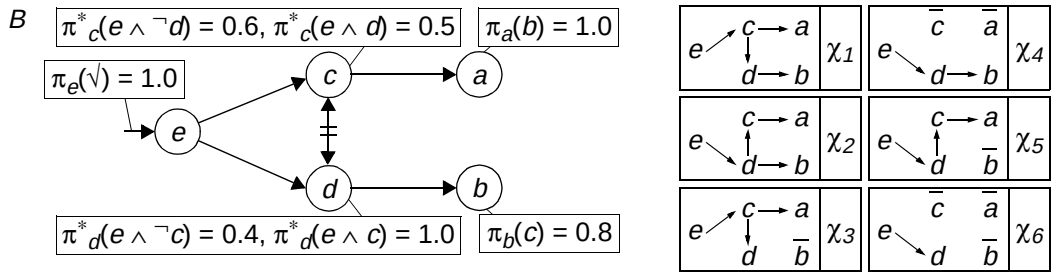


Figure 7.24: Combined use of simple and extended probability attribute

7.4.6 Absolute probabilities

The absolute probability of an individual action *a* in a group of two-sided probability related actions Ac_{two} as depicted in Figure 7.17, can be determined if (i) the extended probability attribute is used to define the conditional probabilities associated with the alternative causality conditions of the actions in Ac_{two}, and (ii) the absolute probability of the occurrence of all enabling actions in Ac_{one} is known. This absolute probability is denoted as $\Pi(Ac_{one})$.

In this case, the absolute probability of action *a* is defined as:

$$\Pi_a = \Sigma\{\pi^*_a(\gamma) \times \Pi_{EC(pc)} \mid pc \in PC, \gamma \in \Gamma_a\},$$

where

- $\Pi_{EC(pc)}$ represents the absolute probability that the enabling context of *pc* is satisfied, with $EC(pc)$ representing the enabling context of *pc*;
- *PC* consists of the probability contexts in which one or more of the actions in group Ac_{two} are enabled.

The values of $\Pi_{EC(pc)}$ can be determined relative to the value of $\Pi(Ac_{one})$. This is illustrated for the behaviour in Figure 7.23 of Section 7.4.4. The group of two-sided related actions *a*, *b* and *c* has four probability contexts pc_1, pc_2, pc_3 and pc_4. Figure 7.25 depicts the relationships between their enabling contexts. The satisfaction of $EC(pc_1)$ corresponds to the occurrence of action *d*, with $Ac_{one} = \{d\}$. Once $EC(pc_1)$ is satisfied, enabling contexts $EC(pc_2)$ and $EC(pc_3)$ are satisfied when *a* and *b* occur due to conditions $\gamma 1_a$ and $\gamma 1_b$, respectively. Enabling context $EC(pc_4)$ can be satisfied in three alternative cases: (i) after $EC(pc_2)$ is sat-

isfied, b occurs due to $\gamma 3_b$, (ii) after $EC(pc_1)$ is satisfied, a and b synchronize due to conditions $\gamma 2_a$ and $\gamma 2_b$, or (iii) after $EC(pc_3)$ is satisfied, a occurs due to $\gamma 3_a$.

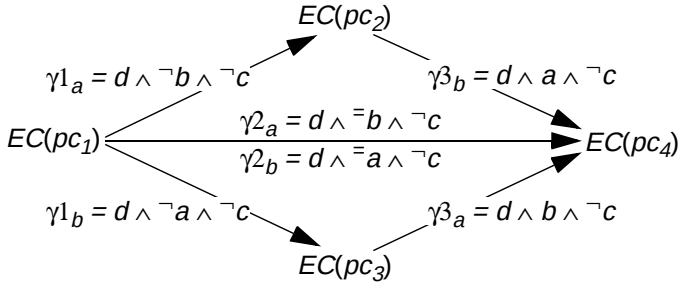


Figure 7.25: Relation between enabling contexts

Based on the scheme in Figure 7.25, the absolute probabilities of the enabling contexts $EC(pc_1)$, $EC(pc_2)$, $EC(pc_3)$ and $EC(pc_4)$ are defined as:

$$\Pi_{EC(pc1)} = \Pi(Ac_{one}) = \Pi_d = 1;$$

$$\Pi_{EC(pc2)} = \Pi_{EC(pc1)} \times \pi^*_a(d \wedge \neg b \wedge \neg c) = 0.2;$$

$$\Pi_{EC(pc3)} = \Pi_{EC(pc1)} \times \pi^*_b(d \wedge \neg a \wedge \neg c) = 0.3;$$

$$\Pi_{EC(pc4)} = \Pi_{EC(pc2)} \times \pi^*_b(d \wedge a \wedge \neg c) + \Pi_{EC(pc3)} \times \pi^*_a(d \wedge b \wedge \neg c) + \Pi_{EC(pc1)} \times \pi^*_a(d \wedge {}^{=}b \wedge \neg c) = 0.32.$$

This renders the following values for the absolute probabilities of actions a, b, c and d:

$$\Pi_d = 1,$$

$$\Pi_a = \Pi_{EC(pc1)} \times (\pi^*_a(d \wedge \neg b \wedge \neg c) + \pi^*_a(d \wedge {}^{=}b \wedge \neg c)) + \Pi_{EC(pc3)} \times \pi^*_a(d \wedge b \wedge \neg c) = 0.42,$$

$$\Pi_b = \Pi_{EC(pc1)} \times (\pi^*_b(d \wedge \neg a \wedge \neg c) + \pi^*_b(d \wedge {}^{=}a \wedge \neg c)) + \Pi_{EC(pc2)} \times \pi^*_a(d \wedge a \wedge \neg c) = 0.5,$$

$$\Pi_c = \Pi_{EC(pc1)} \times \pi^*_c(d \wedge \neg a \wedge \neg b) + \Pi_{EC(pc2)} \times \pi^*_c(d \wedge a \wedge \neg b) + \Pi_{EC(pc3)} \times \pi^*_c(d \wedge \neg a \wedge b) + \Pi_{EC(pc4)} \times \pi^*_c(d \wedge a \wedge b) = 1$$

7.4.7 Extended application

The (execution) semantics of extended probability associations can be defined independently of the type of relations between the involved actions. In this respect, the extended probability attribute can be applied to any behaviour. However, when the extended probability attribute is applied to types of (sub-)behaviours other than groups of two-sided related actions, one should take the following observations into account:

- the definition of consistency rules and correspondence relations may become (very) complicated in case of groups of partly two-sided related actions that contain independent actions or dynamically related actions;
- the correspondence relations associated with certain types of behaviours can not be satisfied or can only be satisfied for specific values of the simple probability attribute, such that the application of the extended probability attribute may be undesirable for these behaviours;
- additional rules may be needed to obtain all probability information from a behaviour

definition.

These observations are illustrated by means of two examples below.

Example: group of partly two-sided related actions

Figure 7.26 depicts a behaviour B that defines a choice between actions a, b and c, such that either b is allowed to occur, or a and c are allowed to occur independently. Since actions a and c are independent, actions a, b and c are called partly two-sided related. Figure 7.26 also depicts the executions of B_Γ, with $E = [\![B_\Gamma]\!] = \{\chi_1, \chi_2, \chi_3, \chi_4, \chi_5, \chi_6\}$.

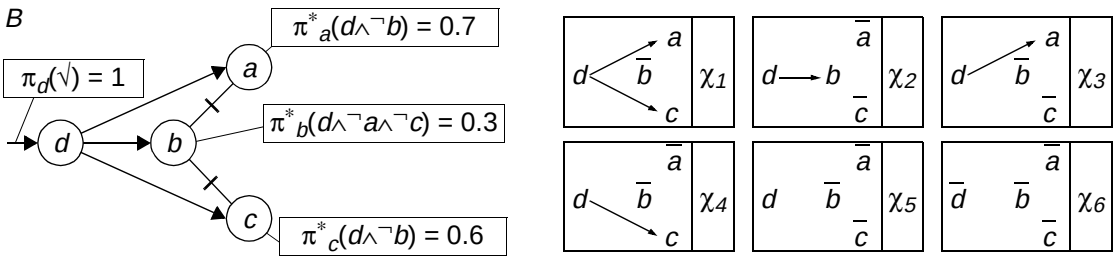


Figure 7.26: Group of partly two-sided related actions

Three probability contexts pc_1, pc_2 and pc_3 are distinguished:

$$pc_1 = \langle\langle\{d\}, \varnothing\rangle, \{d \wedge \neg b \rightarrow a, d \wedge \neg a \wedge \neg c \rightarrow b, d \wedge \neg b \rightarrow c\}\rangle;$$

$$pc_2 = \langle\langle\{d, a\}, \varnothing\rangle, \{d \wedge \neg b \rightarrow c\}\rangle \text{ and } pc_3 = \langle\langle\{d, c\}, \varnothing\rangle, \{d \wedge \neg b \rightarrow a\}\rangle.$$

In case of pc_1, either condition $\gamma_b = d \wedge \neg a \wedge \neg c$ or conditions $\gamma_a = d \wedge \neg b$ and $\gamma_c = d \wedge \neg b$ can be satisfied. This renders the following consistency rules:

$$\pi^*_a(d \wedge \neg b) + \pi^*_b(d \wedge \neg a \wedge \neg c) \leq 1 \text{ and } \pi^*_c(d \wedge \neg b) + \pi^*_b(d \wedge \neg a \wedge \neg c) \leq 1.$$

The probabilities that actions a, b and c occur after action d has occurred, are equal to $\pi^*_a(d \wedge \neg b)$, $\pi^*_b(d \wedge \neg a \wedge \neg c)$ and $\pi^*_c(d \wedge \neg b)$, respectively. Since $\pi_d(\surd) = 1$, this is equivalent to the following equations in terms of the execution model, with $\pi_E(\{\chi_6\}) = 0$:

$$\pi_E(\{\chi_1, \chi_3\}) = \pi^*_a(d \wedge \neg b); \pi_E(\{\chi_1, \chi_4\}) = \pi^*_c(d \wedge \neg b); \pi_E(\{\chi_2\}) = \pi^*_b(d \wedge \neg a \wedge \neg c).$$

The probability that at least one of the actions a, b or c occurs is equal to $\pi_E(\{\chi_1, \chi_2, \chi_3, \chi_4\})$. In order to determine this value from $\pi_E(\{\chi_1, \chi_3\})$, $\pi_E(\{\chi_1, \chi_4\})$ and $\pi_E(\{\chi_2\})$, we have to determine the probability of execution χ_1, i.e., the probability that both a and c occur. Although actions a and c may occur independently, this probability is *not* equal to $\pi^*_a(d \wedge \neg b) \times \pi^*_c(d \wedge \neg b)$. Actions a and c can only occur independently after it has been decided with a probability of $1 - \pi^*_b(d \wedge \neg a \wedge \neg c)$ that b does not occur. Consequently, the product $\pi^*_a(d \wedge \neg b) \times \pi^*_c(d \wedge \neg b)$ should be normalized w.r.t. the probability that b does not occur, such that the probability that both a and c occur is defined as:

$$\pi_E(\{\chi_1\}) = \pi^*_a(d \wedge \neg b) \times \pi^*_c(d \wedge \neg b) / (1 - \pi^*_b(d \wedge \neg a \wedge \neg c)).$$

The above renders the following correspondence relations:

$$\pi'_{a \vee b \vee c}(d) = \pi^*_a(d \wedge \neg b) + \pi^*_b(d \wedge \neg a \wedge \neg c) + \pi^*_c(d \wedge \neg b)$$

$$- \pi^*_a(d \wedge \neg b) \times \pi^*_c(d \wedge \neg b) / (1 - \pi^*_b(d \wedge \neg a \wedge \neg c));$$

$$\pi'_a(d \wedge c) = \pi^*_a(d \wedge \neg b) / (1 - \pi^*_b(d \wedge \neg a \wedge \neg c));$$

$$\pi'_c(d \wedge a) = \pi^*_c(d \wedge \neg b) / (1 - \pi^*_b(d \wedge \neg a \wedge \neg c)),$$

which can be reduced to:

$$\pi_b(d \wedge \neg a \wedge \neg c) = \pi^*_b(d \wedge \neg a \wedge \neg c));$$

$$\pi_a(d \wedge \neg b) = \pi^*_a(d \wedge \neg b) / (1 - \pi^*_b(d \wedge \neg a \wedge \neg c));$$

$$\pi_c(d \wedge \neg b) = \pi^*_c(d \wedge \neg b) / (1 - \pi^*_b(d \wedge \neg a \wedge \neg c)).$$

The equations above have a solution for any value of $\pi_a(d \wedge \neg b)$, $\pi_b(d \wedge \neg a \wedge \neg c)$ and $\pi_a(d \wedge \neg b)$, except for behaviours in which $\pi_b(d \wedge \neg a \wedge \neg c) = 1$.

This example illustrates that the extended probability attribute can be used to partly two-sided relations. Furthermore, it shows that consistency rules and correspondence relations can be defined for this example, at the expense of an additional rule to derive the conditional probability of the occurrences of two independent actions.

However, the generalization of the correspondence relations and additional semantics rules that are needed for these type of behaviours, having any number of actions and any pairs of independent actions, is a rather complex task. Furthermore, the resulting definitions are difficult to understand. This becomes even worse for groups of partly two-sided related actions that also contain dynamic relations. Fortunately, these type of behaviours are not commonly used. Therefore, we choose not to develop general definitions of consistency rules, correspondence relations and additional semantics rules for these types of behaviours, in this thesis. Instead, these definitions are developed for individual behaviours when necessary.

Example: disjunction of enabling conditions

Figure 7.27 depicts a behaviour *B*, in which action *a* depends on the occurrence of action *b* or the occurrence of action *c*.

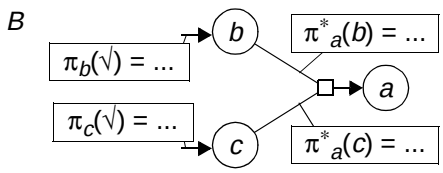


Figure 7.27: Dependency on independent actions (2)

Extended probability associations $\pi^*_a(b)$ and $\pi^*_a(c)$ define the conditional probabilities that *a* occurs due to the occurrences of *b* and *c*, respectively. Since the probability of the occurrence of *a* can be maximal one, the following consistency rule must be obeyed:

$$\Pi_b \times \pi^*_a(b) + \Pi_c \times \pi^*_a(c) \leq 1.$$

According to this rule, the possible values of $\pi^*_a(b)$ and $\pi^*_a(c)$ depend on the values of Π_b and Π_c. This is, however, in conflict with our interpretation of conditional probabilities used so far. Therefore, the following more conservative consistency rule is chosen:

$\pi^*_a(b) + \pi^*_a(c) \leq 1.$

Based on the probability contexts pc_1, pc_2 and pc_3 identified in the corresponding example in Section 7.2.3 (except for initial action d, see Figure 7.12), the following correspondence relations are identified:

$\pi'_a(b \wedge c) = \pi_a(b) + \pi_a(c) - \pi_a(b) \times \pi_a(c) = \pi^*_a(b) + \pi^*_a(c);$

$\pi'_a(b \wedge \overline{c}) = \pi_a(b) = \pi^*_a(b);$

$\pi'_a(\overline{b} \wedge c) = \pi_a(c) = \pi^*_a(c).$

These correspondence relations, combined with the consistency rule given before, have no solution for any values of $\pi_a(b)$ and $\pi_a(c)$. Consequently, this behaviour can not preserve the probability information of the same behaviour with the simple probability attribute.

Despite the dissatisfaction of the correspondence relations, one may prefer the use of the extended attribute, e.g., because it allows one to define a distribution between the probability that a is related to b and the probability that a is related to c, in the executions in which both b and c occur. Due to this choice it becomes impossible to model, e.g., the requirement that action a must occur when b or c occurs in case $\Pi_b < 1$ or $\Pi_c < 1$. This requirement can be modelled using the simple probability attribute.

The use of the extended probability attribute in behaviours for which the associated correspondence relations can not be satisfied, may allow probability requirements to be modelled that could not be modelled when using the simple probability attribute. However, the use of the extended probability attribute in this case, may at the same time disallow the modelling of probability requirements that could be modelled when using the simple probability attribute.

7.5 Formal definition (2)

This section defines the execution semantics of the extended integral probability attribute. We assume that this attribute is applied only to groups of two-sided related actions that obey the characteristics of Ac_{two} as discussed in Section 7.4.1.

Extended probability association $\langle a, \gamma, \pi^*_a(\gamma)\rangle$ in a behaviour B constrains the probability of the following subsets of executions of B_Γ:

- $E_\gamma = \{\chi \mid \chi \in (\llbracket B_\Gamma \rrbracket \setminus \llbracket Ena(\gamma) \rightarrow a \rrbracket), Ac_{ena}(\gamma) \subseteq A_\chi\}$, which defines the executions of B_Γ in which action a may be enabled by alternative causality condition γ, where:
  - $Ena(\gamma) = \wedge\{b \mid b \in C(\gamma)\}$ denotes the conjunction of enabling conditions in γ;
  - $Ac_{ena}(\gamma) = Ac(Ena(\gamma))$ denotes the enabling actions in γ;
  - the restriction operator \ has been explained in Section 3.6.3;
- $E_{\gamma \rightarrow a} = E_\gamma \setminus (\llbracket \gamma \rightarrow a \rrbracket \setminus \{\boxed{a}\})$, which defines the subset of E_γ in which action a occurs due to alternative causality condition γ;
- $\overline{E}_{\gamma \rightarrow a} = E_\gamma - E_{\gamma \rightarrow a}$, which defines the subset of E_γ in which action a does not occur due

to alternative causality condition γ.

such that the following ratio's must hold, with $E = [\![B_\Gamma]\!]$ and $\pi_E : \wp(E) \to \wp(\boldsymbol{P})$:

$\pi^*_a(\gamma) = \pi_E(E_{\gamma \to a}) / \pi_E(E_\gamma)$;

$1 - \pi^*_a(\gamma) = \pi_E(\overline{E}_{\gamma \to a}) / \pi_E(E_\gamma)$.

Definition 7.3 The execution semantics of extended probability association $\langle a, \gamma, \pi^*_a(\gamma)\rangle$ in a behaviour B is defined as:

$[\![\langle a, \gamma, \pi^*_a(\gamma)\rangle]\!] = \{ \langle E_{\gamma \to a}, \pi_E(E_\gamma) \times \pi^*_a(\gamma)\rangle, \langle E_\gamma, \pi_E(E_{\gamma \to a}) / \pi^*_a(\gamma)\rangle \}$,

with: $E = [\![B_\Gamma]\!]$,
$E_\gamma = \{\chi \mid \chi \in ([\![B_\Gamma]\!] \setminus [\![Ena(\gamma) \to a]\!]), Ac_{ena}(\gamma) \subseteq A_\chi\}$,
$E_{\gamma \to a} = E_\gamma \setminus ([\![\gamma \to a]\!] \setminus \{\boxed{a}\})$,
$\overline{E}_{\gamma \to a} = E_\gamma - E_{\gamma \to a}$. ■

Example

Figure 7.28 depicts a behaviour B consisting of a group of two-sided related actions a, b and c, which are enabled by action d. Actions a, b and c are related by the conjunction of a temporal freedom relation between actions a and c and two choice relations between actions a and b and actions b and c. Figure 7.28 also shows the executions of B_Γ, with $E = [\![B_\Gamma]\!] = \{\chi_1, \chi_2, \chi_3, \chi_4, \chi_5, \chi_6, \chi_7, \chi_8\}$.

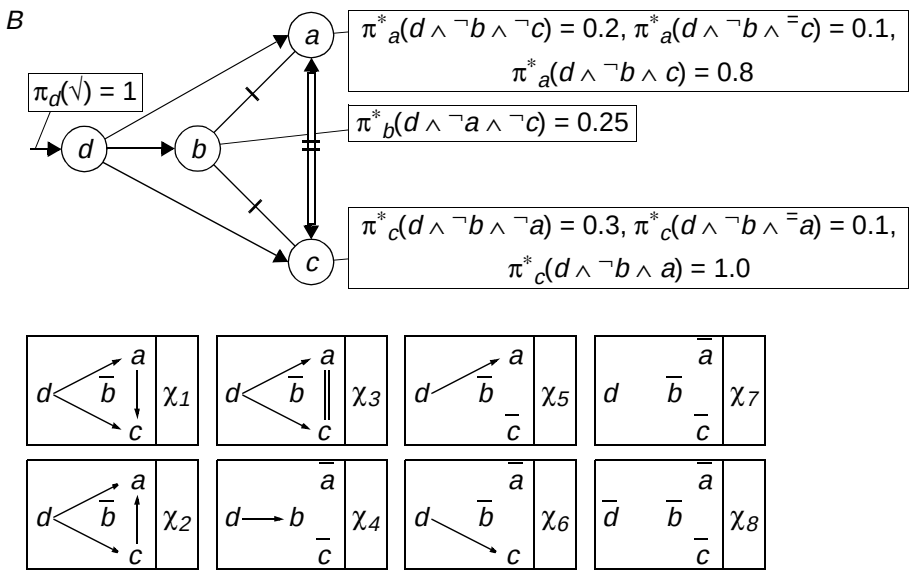


Figure 7.28: Execution semantics of groups of two-sided related actions

The probability of these executions is constrained by the following equations:

$\pi_d(\surd) = \pi_E(E_d) / \pi_E(E)$, with $E_d = E - \{\chi_8\}$;

$\pi^*_a(d \wedge \neg b \wedge \neg c) = \pi_E(\{\chi_1, \chi_5\}) / \pi_E(E_d)$;

$\pi^*_a(d \wedge \neg b \wedge {}^{=}c) = \pi^*_c(d \wedge \neg b \wedge {}^{=}a) = \pi_E(\{\chi_3\}) / \pi_E(E_d)$;

$\pi^*_a(d \wedge \neg b \wedge c) = \pi_E(\{\chi_2\}) / \pi_E(\{\chi_2, \chi_6\})$;

$\pi^*_b(d \wedge \neg a \wedge \neg c) = \pi_E(\{\chi_4\}) / \pi_E(E_d)$;

$\pi^*_c(d \wedge \neg b \wedge \neg a) = \pi_E(\{\chi_2, \chi_6\}) / \pi_E(E_d)$;

$\pi^*_c(d \wedge \neg b \wedge a) = \pi_E(\{\chi_1\}) / \pi_E(\{\chi_1, \chi_5\})$;

$\pi_E(E) = 1$,

which render the following solution:

$\pi_E(\{\chi_8\}) = 0$; $\pi_E(\{\chi_7\}) = 0.15$; $\pi_E(\{\chi_6\}) = 0.06$; $\pi_E(\{\chi_5\}) = 0$;
$\pi_E(\{\chi_4\}) = 0.25$; $\pi_E(\{\chi_3\}) = 0.1$; $\pi_E(\{\chi_2\}) = 0.24$; $\pi_E(\{\chi_1\}) = 0.2$.

7.6 Combination with information, time and location attributes

This section briefly considers the specification of the integral probability attribute in combination with the information, time and location attributes by means of two examples.

Example: time-out

Information, time and location attribute constraints can be modelled independently of simple and extended integral probability attribute constraints. Figure 7.29 illustrates this for the modelling of the time-out (action *time-out*) of the receipt of an acknowledgement (action *ack*) of a previously sent message (action *send*). Action *ack* is allowed to occur within ΔT time units after the occurrence of action *send*, whereas action *time-out* is only allowed to occur precisely at ΔT time units after *send*. These time constraints can be interpreted independently of the probability constraints on actions *send*, *ack* and *time-out*. These probability constraints define that action *ack* must occur after *send* in minimal 90 percent of the executions in which *send* occurs, whereas action *time-out* must occur in maximal 10 percent of these executions. Furthermore, the constraint $\pi^*_{ack}(send \wedge \neg time\text{-}out) + \pi^*_{time\text{-}out}(send \wedge \neg ack) = 1$ imposes that either *ack* or *time-out* must occur after *send* has occurred.

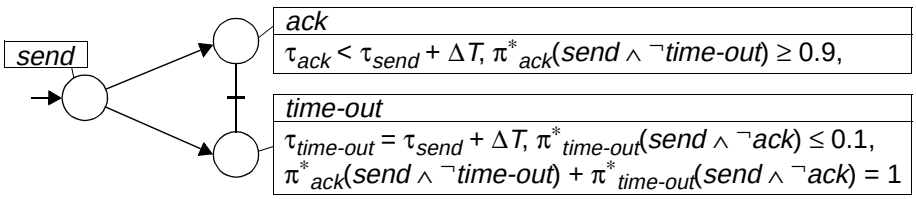


Figure 7.29: Time-out

Simple and extended probability attribute constraints can not always be modelled independently of information, time or location attribute constraints. This property is inherited from the uncertainty attribute, which has been explained in Section 6.1.2. For example, assume alternative causality condition $send \wedge \neg time\text{-}out$ of action *ack* in Figure 7.29 is associated with the information causality condition [ι_{send} = 'yes'], representing that an acknowledgement is only returned when this is requested in the sent message, i.e.:

$send \wedge \neg time\text{-}out$ [ι_{send} = 'yes'] $\rightarrow$ *ack* [...].

In this case, $\pi^*_{ack}(send \wedge \neg time\text{-}out) \geq 0.9$ defines that *ack* must occur after *send* in minimal 90 percent of the executions in which *send* occurs with information value ι_{send} = 'yes'. In

case *send* occurs with ι_{send} = ‘no’, action *ack* is not allowed to occur. Consequently, the probability attribute constraints of action *ack* can not be interpreted independently of the information causality condition [ι_{send} = ‘yes’].

Yet, probability constraint $\pi^*_{time\text{-}out}(send \wedge \neg ack) \leq 0.1$ defines that *time-out* occurs after *send* in at most 10 percent of the executions in which *send* occurs, independently of the value of ι_{send}. However, the conjunction of constraints $\pi^*_{ack}(send \wedge \neg time\text{-}out) \geq 0.9$, $\pi^*_{time\text{-}out}(send \wedge \neg ack) \leq 0.1$ and $\pi^*_{ack}(send \wedge \neg time\text{-}out) + \pi^*_{time\text{-}out}(send \wedge \neg ack) = 1$ renders a conflict in case ι_{send} = ‘no’:

- in case *send* occurs with ι_{send} = ‘yes’, the following equations hold:
  $\pi^*_{ack}(send \wedge \neg time\text{-}out) \geq 0.9$, $\pi^*_{time\text{-}out}(send \wedge \neg ack) \leq 0.1$ and
  $\pi^*_{ack}(send \wedge \neg time\text{-}out) + \pi^*_{time\text{-}out}(send \wedge \neg ack) = 1$,
  which allow many values for $\pi^*_{ack}(send \wedge \neg time\text{-}out)$ and $\pi^*_{time\text{-}out}(send \wedge \neg ack)$;
- in case *send* occurs with ι_{send} = ‘no’, the following equations hold:
  $\pi^*_{ack}(send \wedge \neg time\text{-}out) = 0$, $\pi^*_{time\text{-}out}(send \wedge \neg ack) \leq 0.1$ and
  $\pi^*_{ack}(send \wedge \neg time\text{-}out) + \pi^*_{time\text{-}out}(send \wedge \neg ack) = 1$
  $\Rightarrow \pi^*_{time\text{-}out}(send \wedge \neg ack) \leq 0.1$ and $\pi^*_{time\text{-}out}(send \wedge \neg ack) = 1$,
  which allow no value for $\pi^*_{time\text{-}out}(send \wedge \neg ack)$.

The cause of this conflict is that $\pi^*_{ack}(send \wedge \neg time\text{-}out)$ in the modified example applies to a subset of the executions to which $\pi^*_{time\text{-}out}(send \wedge \neg ack)$ applies. For a better understanding, this could be made explicit, e.g., by representing $\pi^*_{ack}(send \wedge \neg time\text{-}out)$ as follows:
$\pi^*_{ack}(send \wedge \neg time\text{-}out \wedge [\iota_{send} = \text{‘yes’}])$.
However, this conflict can easily be resolved by removing constraint $\pi^*_{time\text{-}out}(send \wedge \neg ack) \leq 0.1$. Any way, this constraint is redundant in the original behaviour of Figure 7.29.

Example: dynamic establishment of probability

Figure 7.30 depicts behaviour *B*, which defines the sequential composition of actions *b* and *a*. The conditional probability that action *a* occurs after action *b* has occurred is established dynamically (and non-deterministically) by action *b* via its information attribute.

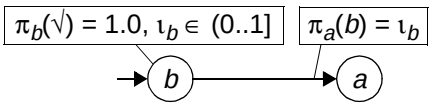


Figure 7.30: Dynamic establishment of probability

The execution semantics of behaviour *B* is defined as follows:

$$[\![B_\Gamma]\!] = \langle E, \pi_E \rangle,$$

with: $E = [\![B_\Gamma]\!] = \{\chi_1 = \boxed{b \rightarrow a}, \chi_2 = \boxed{b \quad \bar{a}}, \chi_3 = \boxed{\bar{b} \quad \bar{a}}\}$;

$\pi_E(\{\chi_1\}) = \iota_b$, $\pi_E(\{\chi_2\}) = 1 - \iota_b$, $\pi_E(\{\chi_3\}) = 0$.

This example illustrates that an integral probability attribute constraint and an information attribute constraint can be related explicitly. The modelling of the integral probability

attribute in combination with the information, time and location attributes is not further elaborated in this thesis.

7.7 Stochastic probability

This section discusses a refinement of the integral probability attribute and the time attribute, which is called the *stochastic probability attribute*. Similarly to the integral probability attribute, a simple and extended stochastic probability attribute are distinguished. We define both attributes below, and illustrate their use by means of some examples.

7.7.1 Introduction

Suppose that one wants to model the (conditional) probability of the occurrence of an action as a function of time. This implies that the integral probability of an action must be distributed over the time interval(s) in which this action is allowed to occur.

For example, consider the time-out example in Figure 7.29. Figure 7.31(i) depicts a uniform distribution of the integral probability of action *ack* over time interval $(\tau_{send}..\tau_{send} + \Delta T]$, assuming $\pi^*_{ack}(send \wedge \neg time\text{-}out) = 0.9$. This distribution function is denoted as $\pi^*_{ack}(send \wedge \neg time\text{-}out, \tau)$, and models the probability that *ack* occurs within interval $(\tau_{send}..\tau]$, i.e., $\tau_{send} < \tau_{ack} \leq \tau$, with $\tau_{send} < \tau < \tau_{send} + \Delta T$. Time moment τ_{send} represents the moment when the occurrence of *ack* becomes enabled by condition $send \wedge \neg time\text{-}out$. Figure 7.31(ii) depicts the corresponding uniform density function, which is the derivative of the uniform distribution function in Figure 7.31(i).

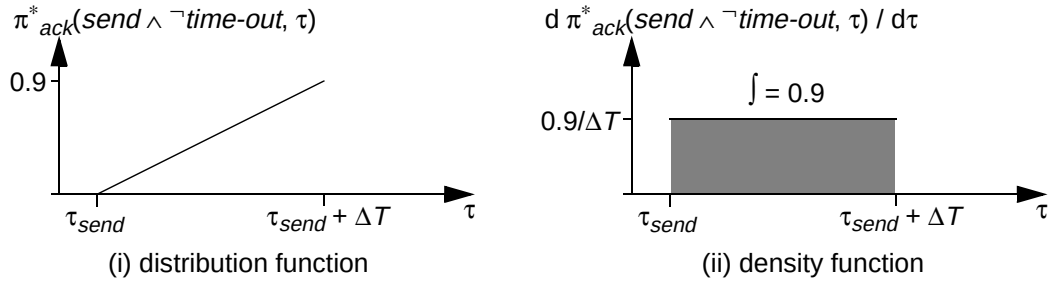


Figure 7.31: Uniform distribution and density function

In order to support the modelling requirement stated above, the time attribute and the simple and extended integral probability attributes are refined, such that the conditional probability of an action *a* is defined as a function of the time moments at which *a* is allowed to occur. These refinements are called the simple and extended *stochastic* probability attributes, respectively, since the time attributes of actions are modelled as stochastic variables.

The information and location attributes can also be modelled as stochastic variables if found necessary. This option is not elaborated in this thesis, but can be performed similarly to the modelling of the time attribute as stochastic variable.

7.7.2 Simple stochastic probability attribute

The simple stochastic probability attribute associates a probability distribution function with an alternative causality condition. Such an association is called a *simple stochastic probability association*, and is denoted as a simple integral probability association extended with a time parameter.

> The simple stochastic probability association $\pi_a(\gamma, \tau)$ defines the probability that action *a* occurs within time interval $(\tau_\gamma..\tau]$, i.e., $\tau_\gamma < \tau_a \leq \tau$, when assuming that γ enables *a* from moment τ_γ, and γ is satisfied when *a* occurs.

The possible time moments τ in $\pi_a(\gamma, \tau)$ are defined by the time attribute constraints of *a*.

The definition above is illustrated by means of three examples. For convenience, we only consider the discrete time case. Therefore, time domain $\boldsymbol{T}$ is restricted to discrete time domain $\boldsymbol{Z} \subset \boldsymbol{T}$, and we assume a sampling period of 1, such that:

$\boldsymbol{Z}$ = {.., -1, 0, 1, 2, ..} represents a two-sided infinite discrete time axis.

Since we consider discrete time in the examples below, we use probability *density* functions instead of probability distribution functions. We believe this is more intuitive, because in the discrete time case a probability density function models the probability that an action occurs *at* a certain time moment.

Example: sequential composition

Figure 7.32 depicts the sequential composition of actions *c*, *b* and *a*, and their associated probability density functions. For example, the probability density function of action *b* models that *b* occurs 1 time unit after *c* with probability 0.7, 2 time units after *c* with probability 0.2, or 3 time units after *c* with probability 0.1.

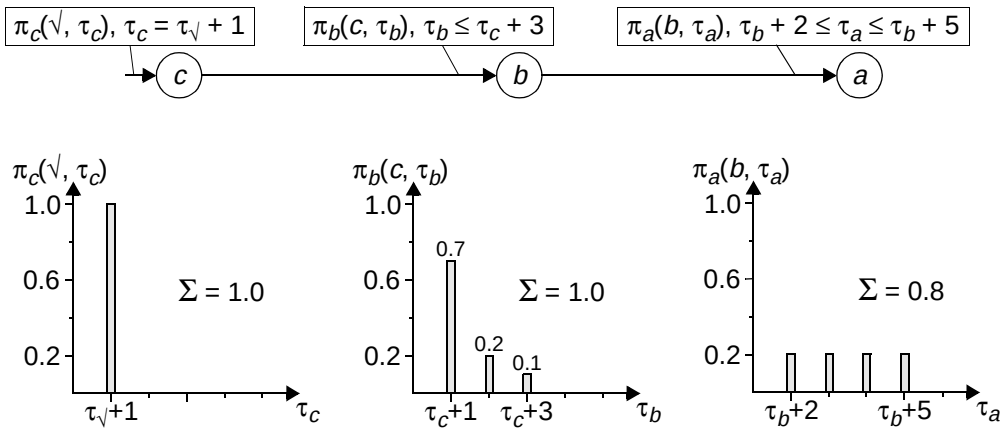


Figure 7.32: Sequential composition

The semantics of the simple stochastic probability attribute in this behaviour is explained in terms of the absolute probability density functions $\Pi_c(\tau_c)$, $\Pi_b(\tau_b)$ and $\Pi_a(\tau_a)$, which represent the absolute probability of the occurrences of actions *c*, *b* and *a* as a function of time. Figure 7.33 depicts these absolute probability density functions.

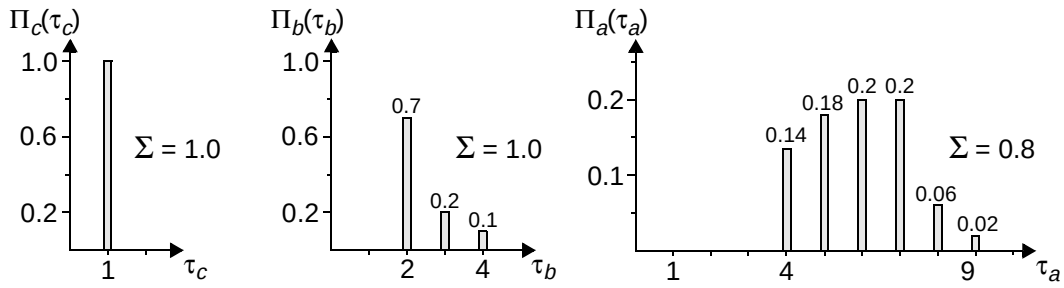


Figure 7.33: Absolute probability density functions

Absolute probability density function $\Pi_c(\tau_c)$ is composed as follows:

$\Pi_c(\tau_c<1) = 0$, since c is not allowed to occur;
$\Pi_c(\tau_c=1) = \Pi_\surd(\tau_\surd=0) \times \pi_c(\surd, \tau_\surd+1) = 1.0$;
$\Pi_c(\tau_c>1) = 0$, since c is not allowed to occur,

where we assume the start condition is satisfied at $\tau_\surd=0$, i.e.:

$$\Pi_\surd(\tau) = \begin{cases} 1, \text{ if } \tau = 0 \\ 0, \text{ if } \tau \neq 0. \end{cases}$$

Absolute probability density function $\Pi_b(\tau_b)$ is composed as follows:

$\Pi_b(\tau_b<2) = 0$, since b is not allowed to occur;
$\Pi_b(\tau_b=2) = \Pi_c(\tau_c=1) \times \pi_b(c, \tau_c+1) = 0.7$;
$\Pi_b(\tau_b=3) = \Pi_c(\tau_c=1) \times \pi_b(c, \tau_c+2) = 0.2$;
$\Pi_b(\tau_b=4) = \Pi_c(\tau_c=1) \times \pi_b(c, \tau_c+3) = 0.1$;
$\Pi_b(\tau_b>4) = 0$, since b is not allowed to occur.

Absolute probability density function $\Pi_a(\tau_a)$ is composed as follows:

$\Pi_a(\tau_a<4) = 0$, since a is not allowed to occur;
$\Pi_a(\tau_a=4) = \Pi_b(\tau_b=2) \times \pi_a(b, \tau_b+2) = 0.14$;
$\Pi_a(\tau_a=5) = \Pi_b(\tau_b=2) \times \pi_a(b, \tau_b+3) + \Pi_b(\tau_b=3) \times \pi_a(b, \tau_b+2) = 0.18$;
$\Pi_a(\tau_a=6) = \Pi_b(\tau_b=2) \times \pi_a(b, \tau_b+4) + \Pi_b(\tau_b=3) \times \pi_a(b, \tau_b+3) + \Pi_b(\tau_b=4) \times \pi_a(b, \tau_b+2)= 0.2$;
$\Pi_a(\tau_a=7) = \Pi_b(\tau_b=2) \times \pi_a(b, \tau_b+5) + \Pi_b(\tau_b=3) \times \pi_a(b, \tau_b+4) + \Pi_b(\tau_b=4) \times \pi_a(b, \tau_b+3)= 0.2$;
$\Pi_a(\tau_a=8) = \Pi_b(\tau_b=3) \times \pi_a(b, \tau_b+5) + \Pi_b(\tau_b=4) \times \pi_a(b, \tau_b+4) = 0.06$;
$\Pi_a(\tau_a=9) = \Pi_b(\tau_b=4) \times \pi_a(b, \tau_b+5) = 0.02$;
$\Pi_a(\tau_a>9) = 0$, since b is not allowed to occur.

We conclude that $\Pi_a(\tau_a)$, $\Pi_b(\tau_b)$ and $\Pi_c(\tau_c)$ are obtained by the following discrete time convolutions:

$\Pi_a(\tau_a) = (\Pi_b * \pi_a(b))(\tau_a)$;
$\Pi_b(\tau_b) = (\Pi_c * \pi_b(c))(\tau_b)$;
$\Pi_c(\tau_c) = (\Pi_\surd * \pi_c(\surd))(\tau_c)$,

where the discrete time convolution operator $*$ is defined as follows ([41]):

$$(x * y)(\tau) = \Sigma_{\tau' \in \boldsymbol{Z}}\, x(\tau - \tau') \times y(\tau'),\ \tau \in \boldsymbol{Z},$$

provided the sum exists for all $\tau \in \boldsymbol{Z}$.

Example: disjunction of enabling conditions

Figure 7.34 depicts a behaviour of three actions *a*, *b* and *c*, in which action *a* depends on the disjunction of enabling conditions *b* and *c*. Figure 7.34 also depicts the probability density functions associated with the alternative causality conditions of *a*, *b* and *c*.

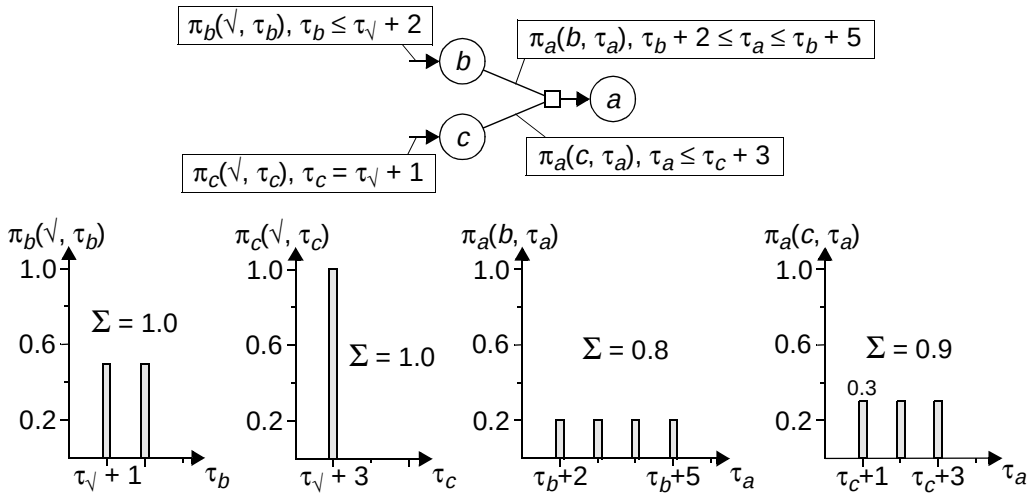


Figure 7.34: Disjunction of enabling conditions

Figure 7.35 depicts absolute probability density functions $\Pi_c(\tau_c)$, $\Pi_b(\tau_b)$ and $\Pi_a(\tau_a)$ that are used to explain the semantics of the simple stochastic probability attribute.

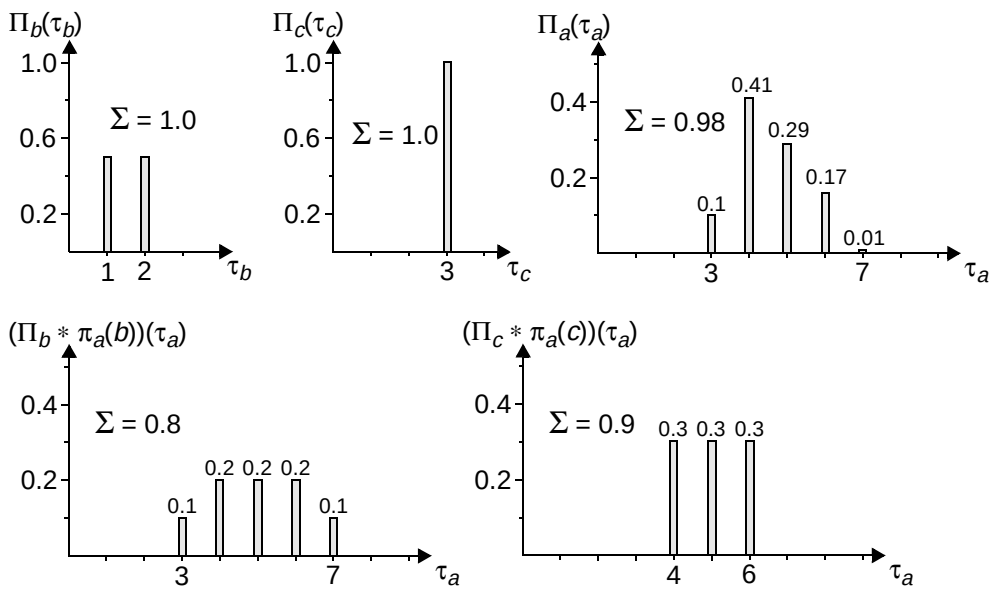


Figure 7.35: Absolute probability density functions (2)

Absolute probability density functions $\Pi_b(\tau_b)$ and $\Pi_c(\tau_c)$ are determined in the same way as in the previous example. Absolute probability density function $\Pi_a(\tau_a)$ is defined as follows for the cases $\tau_a < 3$, $\tau_a = 3$ and $\tau_a = 4$:

$\Pi_a(\tau_a<3)$ = 0, since *a* is not allowed to occur;

$\Pi_a(\tau_a=3)$ = $(\Pi_b * \pi_a(b))(3)$, since *a* is only allowed to occur due to *b*;

$$\begin{aligned}\Pi_a(\tau_a{=}4) \quad &= (\Pi_b * \pi_a(b))(4) \times (1 - \Sigma_{\tau' \leq 4} (\Pi_c * \pi_a(c))(\tau')) + && \#1\\ &\quad (\Pi_c * \pi_a(c))(4) \times (1 - \Sigma_{\tau' \leq 4} (\Pi_b * \pi_a(b))(\tau')) + && \#2\\ &\quad (\Pi_b * \pi_a(b))(4) \times (\Pi_c * \pi_a(c))(4) && \#3\\ &= 0.41. \text{ This is explained below.}\end{aligned}$$

Any implementation of this behaviour has to consider for each possible time moment τ_a the decision whether *a* should occur or not occur at τ_a when condition *b* is satisfied, and the decision whether *a* should occur or not occur at τ_a when condition *c* is satisfied. Both decisions can be considered independently, because conditions *b* and *c* are independent. Action *a* occurs at τ_a when the outcome of one or both decisions is that *a* should occur.

Term #1 in the definition of $\Pi_a(\tau_a{=}4)$ represents the probability of the decision that *a* should occur at $\tau_a = 4$ when condition *b* is satisfied, and the decision that *a* should not occur at any $\tau_a \leq 4$ when condition *c* is satisfied. Since these decisions are considered independently, this probability can be decomposed into the product of the probability that *a* occurs at $\tau_a = 4$ due to the satisfaction of condition *b* when we would assume that *a* depends only on *b*, which is represented by the sub-term $(\Pi_b * \pi_a(b))(4)$, and the probability that *a* does not occur at $\tau_a \leq 4$ due to the satisfaction of condition *c* when we would assume that *a* depends only on *c*, which is represented by the sub-term $(1 - \Sigma_{\tau' \leq 4} (\Pi_c * \pi_a(c))(\tau'))$.

Term #2 in the definition of $\Pi_a(\tau_a{=}4)$ is analogous to term #1 with the roles of conditions *b* and *c* being interchanged. Term #3 represents the probability of the decision that *a* should occur at $\tau_a = 4$ when condition *b* is satisfied, and the decision that *a* should occur at $\tau_a = 4$ when condition *c* is satisfied. In this case, it is undefined whether *a* occurs due to *b* or due to *c*. This decision is completely left to the implementer.

Probability $\Pi_a(\tau_a{=}4)$ is equal to the sum of terms #1, #2 and #3, since these terms are complete, i.e., they represent all possible situations in which *a* can occur at $\tau_a = 4$, and these terms are disjoint, i.e., they represent distinct situations.

The values of $\Pi_a(\tau_a{=}5)$, $\Pi_a(\tau_a{=}6)$, $\Pi_a(\tau_a{=}7)$ and $\Pi_a(\tau_a{>}7)$ can be determined analogously to $\Pi_a(\tau_a{=}4)$. In order to facilitate this, Figure 7.35 also depicts absolute probability density functions $(\Pi_b * \pi_a(b))(\tau_a)$ and $(\Pi_c * \pi_a(c))(\tau_a)$.

Based on the above, a general definition of $\Pi_a(\tau_a)$ is:

$$\begin{aligned}\Pi_a(\tau_a) \quad &= (\Pi_b * \pi_a(b))(\tau_a) \times (\ 1 - \Sigma_{\tau' \leq \tau(a)}(\Pi_c * \pi_a(c))(\tau')\ ) +\\ &\quad (\Pi_c * \pi_a(c))(\tau_a) \times (\ 1 - \Sigma_{\tau' \leq \tau(a)}(\Pi_b * \pi_a(b))(\tau')\ ) +\\ &\quad (\Pi_b * \pi_a(b))(\tau_a) \times (\Pi_c * \pi_a(c))(\tau_a), \qquad \text{with } \tau(a) = \tau_a.\end{aligned}$$

The value of $\Sigma_{\tau(a)} \Pi_a(\tau_a)$ is equal to the value of Π_a, where Π_a represents the absolute integral probability of action *a* in the abstraction of the behaviour in Figure 7.34, which uses the simple *integral* probability attribute instead of the simple *stochastic* probability attribute. In this abstraction, the values of the simple integral probability associations are equal to the integral (summation) of the probability density functions defined by the corresponding simple stochastic probability associations in Figure 7.34, i.e.:

$\pi_a(b) = \Sigma_{\tau(a)} \pi_a(b, \tau_a) = 0.8$, $\pi_a(c) = \Sigma_{\tau(a)} \pi_a(c, \tau_a) = 0.9$,
$\pi_b(\surd) = \Sigma_{\tau(b)} \pi_b(\surd, \tau_b) = 1$, $\pi_c(\surd) = \Sigma_{\tau(c)} \pi_c(\surd, \tau_c) = 1$.

Example: choice

Figure 7.36 depicts the choice between actions *a* and *b*, where both actions depend on action *c*. Figure 7.36 also depicts the probability density functions associated with the alternative causality conditions of *a*, *b* and *c*.

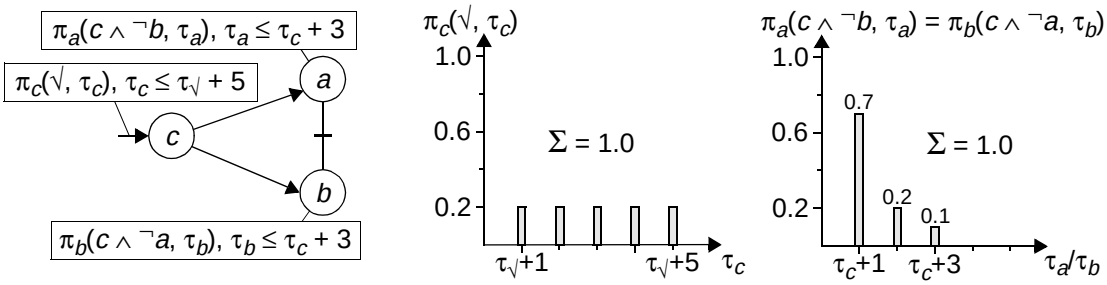


Figure 7.36: Choice (3)

The semantics of the simple stochastic probability attribute is explained in terms of absolute probability density functions $\Pi_c(\tau_c)$ and $\Pi_{a \vee b}(\tau)$. Figure 7.37 depicts these functions, where absolute probability density function $\Pi_{a \vee b}(\tau)$ is defined as follows:

$$\begin{aligned}\Pi_{a \vee b}(\tau) = \; & (\Pi_c * \pi_a(c \wedge \neg b))(\tau) \times (\, 1 - \Sigma_{\tau' \le \tau}(\Pi_c * \pi_b(c \wedge \neg a))(\tau') \,) + \\ & (\Pi_c * \pi_b(c \wedge \neg a))(\tau) \times (\, 1 - \Sigma_{\tau' \le \tau}(\Pi_b * \pi_a(c \wedge \neg b))(\tau') \,) + \\ & (\Pi_b * \pi_a(b))(\tau) \times (\Pi_c * \pi_a(c))(\tau).\end{aligned}$$

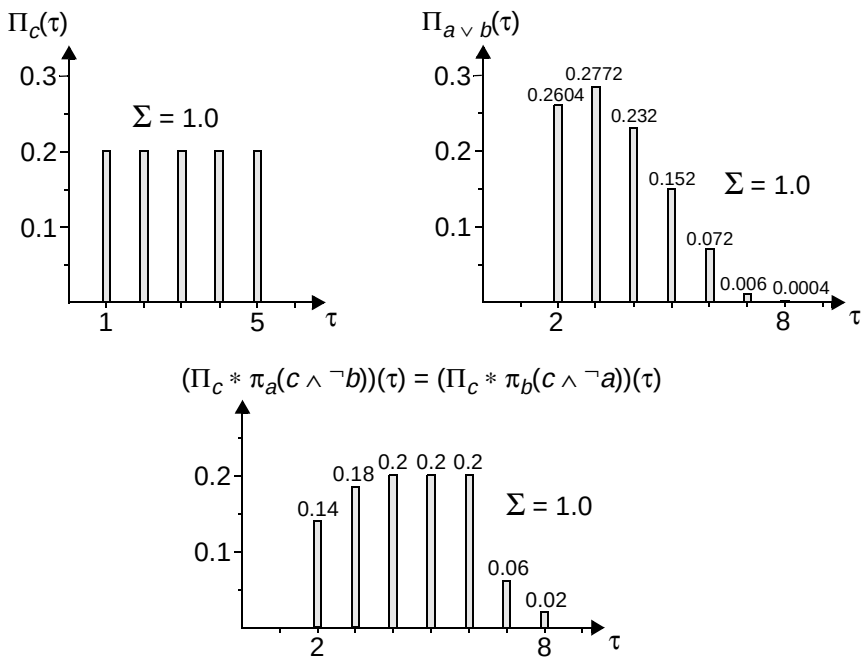


Figure 7.37: Absolute probability density functions (3)

$\Pi_{a \vee b}(\tau)$ is obtained in the same way as $\Pi_a(\tau_a)$ in the previous example. This similarity originates from the common property that the occurrence of $a \vee b$ in this example and the occurrence of *a* in the previous example are both enabled by two alternative causality conditions. The occurrence of $a \vee b$ in this example is enabled by conditions $\gamma_a = c \wedge \neg b$ and $\gamma_b = c \wedge \neg a$, and the occurrence of *a* in the previous example is enabled by conditions $\gamma_a = b$ and $\gamma_a = c$.

The absolute probability density functions $\Pi_a(\tau_a)$ and $\Pi_b(\tau_b)$ can not be determined in this example, since actions *a* and *b* are enabled simultaneously by alternative conditions $\gamma_a = c \wedge \neg b$ and $\gamma_b = c \wedge \neg a$, and the probability of the choice between these conditions is undefined.

7.7.3 Extended stochastic probability attribute

The extended stochastic probability attribute associates a probability distribution function with an alternative causality condition. Such an association is called an *extended stochastic probability association*, and is denoted as an extended integral probability association extended with a time parameter.

> The extended stochastic probability association $\pi^*_a(\gamma, \tau)$ defines the probability that action *a* occurs due to γ within time interval $(\tau_\gamma..\tau]$, i.e., $\tau_\gamma < \tau_a \leq \tau$, when assuming that γ enables *a* from moment τ_γ.

The possible time moments τ in $\pi^*_a(\gamma, \tau)$ are defined by the time attribute constraints of *a*.

The definition above is illustrated by means of a single example. Again, discrete time is considered, and probability density functions are used instead of probability distribution functions.

Example: choice

Figure 7.38 depicts the choice between actions *a* and *b*, where both actions depend on action *c*. Figure 7.38 also depicts the probability density functions associated with the alternative causality conditions of *a* and *b*. The probability density function associated with the start condition of *c* is assumed to be the same as the one in Figure 7.36.

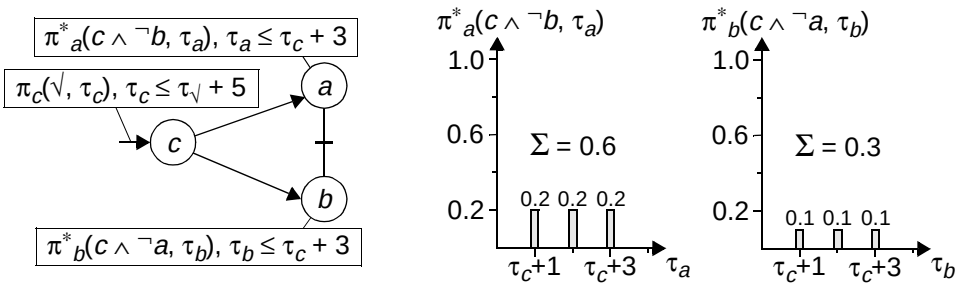


Figure 7.38: Choice (4)

The semantics of the simple stochastic probability attribute is explained in terms of absolute probability density functions $\Pi_c(\tau_c)$, $\Pi_a(\tau_a)$ and $\Pi_b(\tau_b)$. Function $\Pi_c(\tau_c)$ is depicted in Figure 7.37. Figure 7.39 depicts $\Pi_a(\tau_a)$ and $\Pi_b(\tau_b)$, which are defined as follows:

$$\Pi_a(\tau_a) = (\Pi_c * \pi^*_a(c \wedge \neg b))(\tau_a);$$

$$\Pi_b(\tau_b) = (\Pi_c * \pi^*_b(c \wedge \neg a))(\tau_b).$$

The consistency rule $\Sigma_{\tau(a)}\, \pi_a(c \wedge \neg b, \tau_a) + \Sigma_{\tau(b)}\, \pi_b(c \wedge \neg a, \tau_b) \leq 1$ must hold, since the probability that either *a* or *b* occurs can be maximal 1, i.e.: $\Sigma_{\tau(a)}\, \Pi_a(\tau_a) + \Sigma_{\tau(b)}\, \Pi_b(\tau_b) \leq 1$.

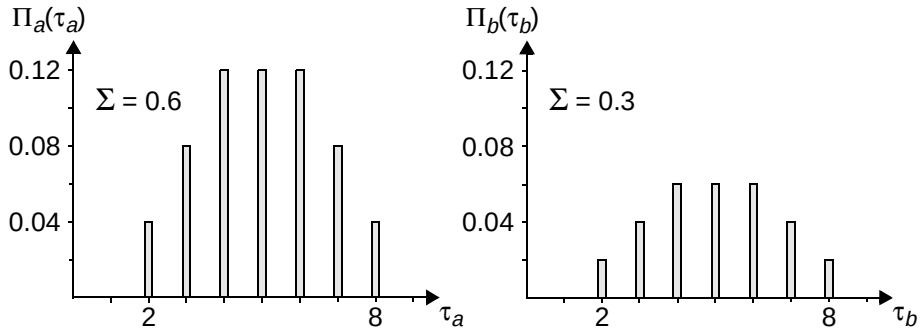


Figure 7.39: Absolute probability density functions (4)

7.7.4 Concluding remarks

The definition of rules to determine stochastic probability information from a behaviour definition and consistency rules to guarantee the consistent definition of the stochastic probability attribute are left for further study. Neither is the execution semantics of the stochastic probability attribute elaborated in this thesis, although Section 3.8.3 gives an indication of how this can be done.

7.8 Conclusions and related work

In this chapter we define the integral and stochastic probability attribute. These probability attributes can be used instead of the uncertainty attribute to quantify the uncertainty of action occurrences. We model the integral probability of an action by associating a range of integral probability values with each alternative causality condition of this action, and model the stochastic probability by associating a probability distribution function with each alternative condition. The integral probability attribute quantifies the uncertainty values of the uncertainty attribute, such that the *must* value corresponds to probability value 1, and the *may* value corresponds to a probability value in the range (0..1). The stochastic probability attribute uses the time attribute of an action as a stochastic variable, such that a probability distribution function defines for each time interval in which the action can occur the probability that the action actually occurs.

Two interpretations of an (integral or stochastic) probability association are distinguished: the *simple* interpretation defines the probability that the action occurs when the associated condition is satisfied, and the *extended* interpretation defines the probability that the action occurs due to the associated condition when this condition enables the action. These two interpretations in combination with the possibility to associate probability values with causality conditions involving enabling, disabling and synchronization conditions, renders an expressive basic language for modelling probability.

The complete architectural and formal definition of the integral and stochastic probability attributes proved to be a too ambitious goal for this thesis and needs further elaboration. The following topics should be addressed in more depth in future work:

- the application of the extended interpretation beyond groups of two-sided related

actions, and its combined use with the simple interpretation;

- the modelling of probability attribute constraints in combination with information, time and location attribute constraints;
- the execution semantics of the stochastic probability attribute.

Furthermore, intuitive and easy-to-use shorthand notations to facilitate the modelling of probability constraints in common behaviour patterns should be developed.

Most probability extensions of process algebras and event structures we encountered, e.g., in [1, 10, 11, 15, 23, 26, 37, 38, 49, 68, 78], are limited to the choice relation and the enabling relation. Extensions that assign probabilities to actions involved in a choice relation normally demand that the sum of these probabilities is equal to 1, and extensions that assign a probability distribution function to the enabling condition of an action normally demand that the integral of this function is equal to 1. This implies, for instance, that the uncertainty of the occurrence of an action can only be modelled using an additional action that disables the uncertain action with some probability. For example, in process algebras and event structures, the loss of a data unit is generally modelled by means of an internal event, which disables the action that models the receipt of this data unit.

The execution concept is sufficiently expressive to define the formal semantics of the integral (and stochastic) probability attribute. A behaviour is considered as an experiment, and the executions of this behaviour are considered as the possible outcomes of this experiment. An execution gets a (set of) probability value(s), which represents the probability of this execution being the outcome of the behaviour experiment. The sum of the probability of all possible executions (outcomes) is always equal to 1. The probability of an action occurrence can be derived as the sum of the probability of the executions in which this action occurs.

Although our model allows the representation of stochastic and performance characteristics, the coupling between this model and performance analysis models should be investigated in order to allow performance analysis to be directly based on behaviour specifications. This topic falls outside the scope of this thesis.

Chapter 8

Behaviour refinement

During the top-down design of distributed systems, abstract designs have to be replaced by more concrete designs that are closer to available implementation mechanisms. Behaviour refinement is a design operation in which abstract behaviours are replaced by more concrete behaviours. Methods that guide designers and enforce the correctness of these replacements are necessary in order to increase the effectiveness of the design process. This chapter presents a set of methods to perform behaviour refinement, based on a careful consideration of the basic design concepts of action and causality relation.

The structure of this chapter is as follows. Section 8.1 discusses behaviour refinement in detail. Section 8.2 presents a method to abstract from actions that are inserted during behaviour refinement. Section 8.3 presents a method for handling actions that are refined by activities of arbitrary complexity. Section 8.4 presents methods with the same goals as the ones in Sections 8.2 and 8.3 for behaviours represented using the execution model. Section 8.5 applies behaviour refinement to a concrete case study. And Section 8.6 presents the conclusions.

8.1 Definition

This section defines the purpose, underlying principles and approach to behaviour refinement. Two basic types of behaviour refinement are identified: action refinement and causality refinement. This section also explains the use of abstraction to assess the conformance relation between an abstract behaviour and the corresponding concrete behaviour. Finally, we discuss the role of correctness relations in the comparison between the abstraction of a concrete behaviour and the original abstract behaviour.

8.1.1 Purpose

During the design of distributed systems we may replace abstract designs by more concrete designs. We consider the relation between an abstract design and a more concrete design based on the assumption that an abstract design is a prescription for implementation. An abstract design prescribes *what* should be implemented, whereas a more concrete design prescribes *how* this abstract design should be implemented. The notions of abstract design and concrete design are relative, since a more concrete design can be considered as an abstract design in a next design step.

The objective of behaviour refinement is to replace an abstract behaviour by a more concrete behaviour that conforms to this abstract behaviour. Behaviour refinement allows a designer to add more detail to the abstract behaviour, such that the concrete behaviour is closer to the real system behaviour. Figure 8.1 depicts an example of behaviour refinement.

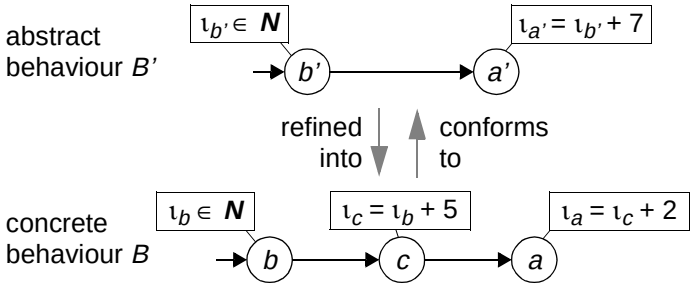


Figure 8.1: Behaviour refinement design operation

Actions of abstract behaviours are called *abstract actions* and actions of concrete behaviours are called *concrete actions*. We assume that the occurrence of each abstract action corresponds to the occurrence of one or more concrete actions. This assumption makes it possible to compare the abstract behaviour with the concrete behaviour, by comparing the abstract actions with their corresponding concrete actions. This comparison is needed in order to assess whether the concrete behaviour conforms to the abstract behaviour.

Concrete actions that correspond to abstract actions are called *reference actions*, since they are considered as reference points in the concrete behaviour for assessing conformance. Concrete actions that are not reference actions are called *inserted actions*, since they are inserted during behaviour refinement. For example, in Figure 8.1 actions *b'* and *a'* are abstract actions and actions *b*, *c* and *a* are concrete actions. Actions *b* and *a* are reference actions that correspond to abstract actions *b'* and *a'*. Action *c* is an inserted action. Abstract actions are denoted by the action identifiers of their corresponding reference actions appended with a prime.

A *conformance relation* defines which concrete behaviours are valid refinements, or implementations, of the abstract behaviour. This conformance relation should guarantee that what is prescribed in the abstract behaviour is preserved by the concrete behaviour. The following requirements for conformance are identified:

1. *preservation of action relations*: the structure of relations between abstract actions is preserved by the structure of relations between their corresponding concrete actions;
2. *preservation of attribute values*: attribute values of abstract actions are preserved by the attributes of their corresponding concrete actions;
3. *preservation of attribute value relations*: relations between attribute values of abstract actions are preserved by the relations between the attributes of the corresponding concrete actions.

For example, in Figure 8.1 the enabling relation between abstract actions *b'* and *a'* is preserved by the conjunction of the enabling relation between concrete actions *b* and *c* and the enabling relation between concrete actions *c* and *a*. The information values of abstract actions *b'* and *a'* are preserved by reference actions *b* and *a*, respectively, since they can

establish the same information values, respectively. Furthermore, information reference relation $\iota_{a'} = \iota_{b'} + 7$ between abstract actions *b'* and *a'* is preserved by the conjunction of information reference relation $\iota_c = \iota_b + 5$ between concrete actions *c* and *b* and information reference relation $\iota_a = \iota_c + 2$ between concrete actions *b* and *a*.

The scope of behaviour refinement in this thesis is restricted to behaviour definitions consisting of actions and their causality relations. The refinement of actions into interactions or the refinement of behaviours involving interactions is not elaborated in this thesis.

8.1.2 Basic types of refinement

Two basic types of behaviour refinement are distinguished:

- *causality refinement*, which consists of replacing causality relations between abstract actions by causality relations involving their corresponding concrete actions and some inserted actions;
- *action refinement*, which consists of replacing an abstract action by a concrete activity involving multiple concrete actions and their causality relations.

Instances of behaviour refinement may consist of one of these basic types of refinement or a combination of both. The essential difference between causality refinement and action refinement is the way attributes of abstract actions are distributed over the attributes of concrete actions. This difference is reflected in distinct specializations of the preservation of attribute values conformance requirement, one for each basic type of behaviour refinement.

Causality refinement

Causality refinement allows one to model the relations between abstract actions in more detail through the introduction of inserted actions. Inserted actions model additional activities in the concrete behaviour that were not relevant in the elaboration of the abstract behaviour. An essential characteristic of causality refinement is that the attributes of an abstract action are preserved by the attributes of a single concrete action. Therefore, each abstract action corresponds to a *single* reference action.

Action refinement

Action refinement allows one to model an activity that is represented by a single abstract action in more detail. The activity is decomposed into multiple related sub-activities which are represented by concrete actions and their causality relations. An essential characteristic of action refinement is that at least one of the attributes of the abstract action is distributed over the attributes of multiple concrete actions in the activity.

An activity that replaces an abstract action makes its attribute values available through the occurrence of one or more of its final actions. These final actions are the reference actions that correspond to the abstract action. The following generic cases are distinguished:

- *single final action*: an activity has a single final action, such that this activity makes all

its attribute values available when this final action occurs;

- *conjunction of final actions*: an activity has multiple independent or synchronized final actions, such that this activity makes all its attribute values available when all these final actions occur;
- *disjunction of final actions*: an activity has multiple alternative final actions, such that this activity makes all its attribute values available when one of these final actions occurs. This action is called the actual final action;
- any combination of conjunctions and disjunctions of final actions.

Considering the cases of a single final action, conjunction of final actions and disjunction of final actions, which are indicated as *(sf)*, *(cf)* and *(df)*, respectively, the conformance requirement concerning the preservation of attribute values applies as follows:

- the information values of an abstract action should be preserved in:

  (sf) the information attribute of the final action and the information attribute(s) of inserted action(s) that can be referred to via the final action;

  (cf) the union of the information attributes of the final actions and the information attributes of inserted actions that can be referred to via the final actions;

  (df) the information attribute of the actual final action and the information attribute(s) of inserted action(s) that can be referred to via this final action.

  Information values of the abstract action are established in the final action(s) or are established in the inserted actions of an activity that can be referred to via the final actions;

- the time moment of an abstract action should be preserved by:

  (sf) the time moment of the final action;

  (cf) the time moment of the latest final action;

  (df) the time moment of the actual final action.

  The abstract action occurs when all (information) values of the activity are available;

- the location of an abstract action should be preserved by:

  (sf) the location of the final action;

  (cf) the collection of the locations of the final actions;

  (df) the location of the actual final action.

  The location of the abstract action represents the location(s) of the final action(s);

- the probability of an abstract action should be preserved by:

  (sf) the probability that the final action occurs;

  (cf) the probability that all final actions occur;

  (df) the probability that one of the final actions occurs.

  The probability that the abstract action occurs is the probability that the entire activity

terminates successfully.

Example

The difference between causality refinement and action refinement in which an action is replaced by an activity with a single final action is rather subtle. Figure 8.2 illustrates this difference, by considering an abstract behaviour that consists of abstract actions *b'* and *a'*. Abstract action *a'* establishes a pair of (sub-)information values ι1 and ι2, such that: $\iota_{a'} = \langle \iota 1, \iota 2 \rangle$, with $\iota 1 = f(\iota_{b'})$ and $\iota 2 = g(\iota_{b'})$. Abstract action *b'* corresponds to reference action *b* and abstract action *a'* corresponds to reference actions *a* and *a2*. We assume that the information values of abstract action *b'* are preserved by reference action *b*.

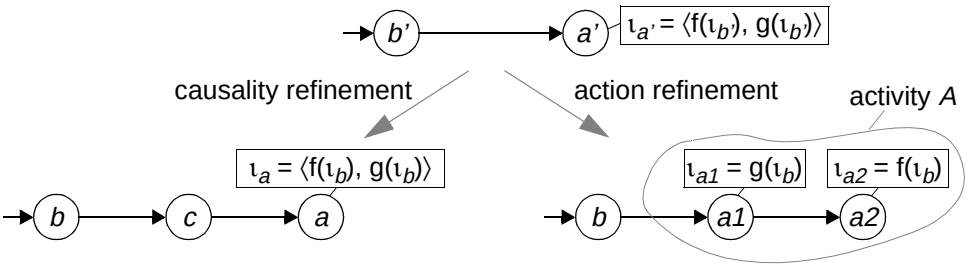


Figure 8.2: Difference between causality refinement and action refinement

Performing causality refinement we can, e.g., replace the enabling relation between abstract actions *b'* and *a'* by an enabling relation between reference action *b* and inserted action *c* and an enabling relation between inserted action *c* and reference action *a*. In this way *a* can refer via *c* to the attribute values of *b*. Reference action *a* should establish the same attribute values as abstract action *a'*.

Performing action refinement we can, e.g., replace abstract action *a'* by concrete activity *A*, consisting of inserted action *a1* and reference action *a2*. The establishment of information values ι1 and ι2 can be distributed over actions *a1* and *a2*, respectively. The information attributes of actions *a2* and *a1* preserve the information values of abstract action *a'*, since information value ι2 is established in final action *a2* and information value ι1 is established in *a1* which can be referred to via *a2*.

In principle, we could consider both refinements in Figure 8.2 as possible refinements of abstract action *a'*, i.e., we could consider them as distinct instances of a generic action refinement operation. However, we deliberately chose to consider the refinements in Figure 8.2 as distinct types of refinement in order to stress the following differences between them:

- in case of causality refinement the attributes of an abstract action are not distributed over the attributes of multiple concrete actions, whereas in case of action refinement at least one of the attributes of an abstract action is distributed over multiple concrete actions. Consequently, when determining the abstraction of a concrete behaviour obtained by causality refinement we only have to abstract from inserted actions. When determining the abstraction of a concrete behaviour obtained by action refinement, in contrast, we have to abstract from groups of (one or more) final actions and may have to abstract from inserted actions. This distinction justifies the separation of methods to abstract from inserted actions in Section 8.2, and methods to abstract from groups of

final actions in Section 8.3;

- causality refinement is more concerned with the decomposition of causality relations, whereas action refinement is more concerned with the decomposition of actions. This distinction motivated the names for both types of refinement.

Examples of action refinement with multiple final actions are presented in Section 8.3 and can be found in [56].

8.1.3 Use of abstraction

An abstract behaviour can be replaced by many alternative concrete behaviours. Depending on the choice of a concrete behaviour, different concrete actions and their causality relations are added to the abstract behaviour. Since this choice is determined by specific design objectives, behaviour refinement can not be automated in its totality.

In contrast, the abstraction of a concrete behaviour is unique. When abstracting from certain concrete actions and their causality relations, the abstraction of this concrete behaviour is completely determined by the remaining concrete actions and their causality relations. Rules can be provided to calculate this abstraction. These rules can, in principle, be automated.

The uniqueness of an abstraction allows one to assess the conformance between an abstract behaviour and a concrete behaviour, by comparing the abstraction of the concrete behaviour with the original abstract behaviour. Therefore, we distinguish the following subsequent design activities in an instance of behaviour refinement:

1. *delimitation of the abstract behaviour*: we only consider the refinement of behaviours that are influenced by a finite number of abstract actions. For example, in case of recursive behaviours one should identify the finite behaviour parts that are (infinitely) repeated;
2. *refinement of the abstract behaviour into a concrete behaviour*: in this activity we determine how the abstract behaviour is implemented by the concrete behaviour;
3. *determination of the abstraction of the concrete behaviour*: a method to perform this activity is presented below;
4. *comparison of the abstraction of the concrete behaviour with the original abstract behaviour*: both behaviours should comply to a certain correctness relation. If this is not the case, the concrete behaviour is not considered as a correct implementation of the abstract behaviour. In this case the designer must return to design activity 2.

Determination of the abstraction of a concrete behaviour

The following steps define a method to determine the abstraction of a concrete behaviour:

1. *identify reference actions and inserted actions in the concrete behaviour*: particularly, identified reference actions have to be considered as:
   - (single) reference actions, which are obtained when causality refinement has been

applied; or

- groups of reference actions, which are formed by grouping the final actions of each activity that is obtained when action refinement has been applied;

2. *abstract from inserted actions*: an abstraction method for doing this is presented in Section 8.2;
3. *replace each group of reference actions by an abstract action*: an abstraction method for doing this is presented in Section 8.3.

8.1.4 Correctness relations

The conformance between an abstract behaviour and its corresponding concrete behaviour is assessed by assessing a correctness relation between this abstract behaviour and the abstraction of the concrete behaviour. Depending on the specific conformance requirements, this correctness relation can be:

- an equivalence relation, which defines that the concrete behaviour should preserve all behaviour properties of the abstract behaviour; or
- a partial ordering relation, which defines that the concrete behaviour should preserve a subset of the behaviour properties of the abstract behaviour.

Example: strong and weak preservation of attribute values

Two alternatives of the preservation of attribute values conformance requirement are considered:

- *strong preservation of attribute values*: all attribute values that are possible for an abstract action are also possible for the corresponding concrete actions; and
- *weak preservation of attribute values*: some attribute values that are possible for an abstract action are not possible for the corresponding concrete actions.

Strong preservation can be assessed in terms of an equivalence relation $\approx$ on the attribute values of two abstract actions, whereas weak preservation can be assessed in terms of a partial ordering relation $\angle$ on the attribute values of two abstract actions. Figure 8.3 depicts an example of strong and weak preservation of attribute values. Behaviours B_1' and B_2' represent the abstractions of two alternative refinements B_1 and B_2 of abstract behaviour *B'*. Since abstract actions *b'* and *b1'* can occur at the same time moments, behaviours *B'* and B_1 obey the strong preservation of attribute values. In contrast, abstract action *b2'* can only occur at a subset of the time moments at which abstract action *b'* can occur. Consequently, behaviours *B'* and B_2 obey the weak preservation of attribute values.

Example: strong and weak preservation of independence

The preservation of independence between abstract actions is complementary to the preservation of action relations conformance requirement. Two alternatives for the conformance requirements on the preservation of independence are considered:

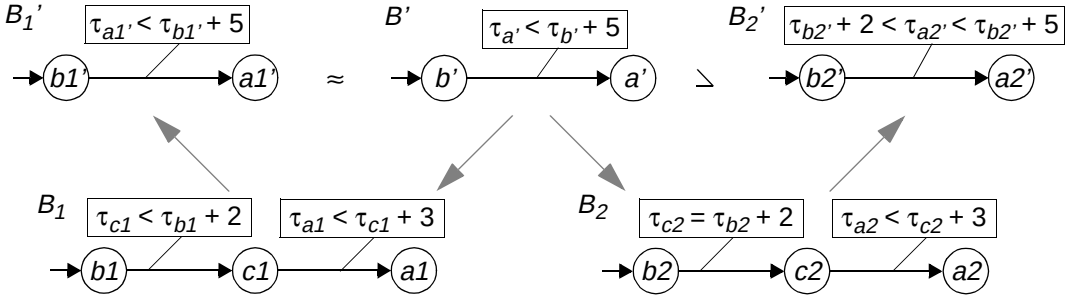


Figure 8.3: Strong and weak preservation of attribute values

- *strong preservation of independence*: all reference actions corresponding to abstract actions that are independent in the abstract behaviour are also independent in the concrete behaviour; all reference actions corresponding to abstract actions that are related in the abstract behaviour are also related in the concrete behaviour;
- *weak preservation of independence*: some reference actions corresponding to abstract actions that are independent in the abstract behaviour are related in the concrete behaviour; all reference actions corresponding to abstract actions that are related in the abstract behaviour are also related in the concrete behaviour.

Strong preservation can be assessed in terms of an equivalence relation $\approx$ on the characteristic of two abstract actions being related or independent, whereas weak preservation can be assessed in terms of a partial ordering relation $\angle$ on the characteristic of two abstract actions being related or independent (e.g., with 'independent' $\angle$ 'related'). Figure 8.4 depicts an example of strong and weak preservation of independence. Abstract behaviour *B'* consists of two independent abstract actions *a'* and *c'*, which model activities *A* and *C*, respectively. Concrete behaviours B_1 and B_2 represent two alternative refinements in which the initiation of activity *A* and the initiation of activity *C* are explicitly represented by concrete actions *b1* and *d1* in B_1, respectively, and by concrete actions *b2* and *d2* in B_2, respectively. In case of B_1, activities *A* and *C* are executed independently, e.g., because they are performed on distinct processors. In case of B_2, activities *A* and *C* are executed in arbitrary order, e.g., because they must be performed on the same processor. For this purpose, a choice between the initiation of *A* and the initiation of *C* is defined. In case activity *A* is initiated first, activity *C* is only initiated after *A* is completed, and vice versa.

Behaviours B_1*'* and B_2*'* represent the abstractions of B_1 and B_2, respectively. Since actions *a1*' and *c1*' are independent, behaviours *B'* and B_1 obey the strong preservation of independence. In contrast, actions *a2*' and *c2*' are related by an interleaving relation. Consequently, behaviours *B'* and B_2 obey the weak preservation of independence.

8.2 Abstraction from inserted actions

This section presents a method that allows one to deduce the abstract behaviour of a given concrete behaviour, by abstracting from the inserted actions and their influence on the reference actions in the concrete behaviour. This method contains steps and rules that have to be followed in order to abstract from a *single* inserted action, say *z*. We assume that the

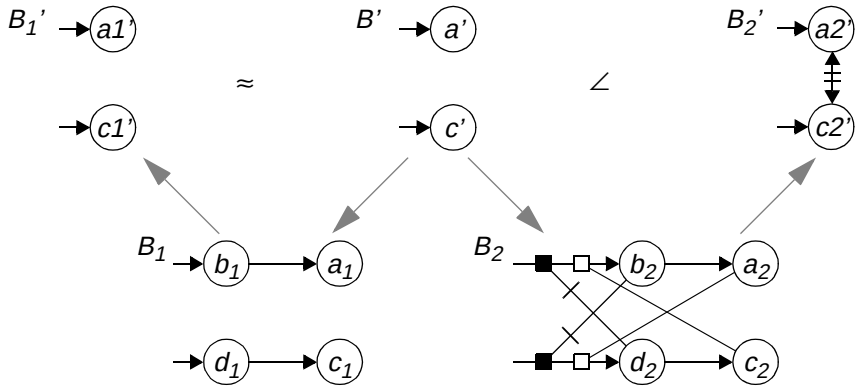


Figure 8.4: Strong and weak preservation of independence

abstraction of multiple inserted actions can be performed by consecutively abstracting from each single inserted action in any order.

8.2.1 Causality context

When abstracting from a single inserted action *z* in a concrete behaviour *B*, the remaining actions in *B* are considered as reference actions. The influence of inserted action *z* on these reference actions can be delimited to the causality context of *z*. The causality context of *z* only considers the actions in *B* that are directly related to *z*. An action *a* is directly related to *z* if (i) action *z* is mentioned in the causality condition of *a*, or (ii) action *a* is mentioned in the causality condition of *z*.

The causality context of inserted action *z* is defined as the part of behaviour *B* that consists of:

- the causality relation of *z*;
- parts of the causality relations of all actions in *B* that are directly related to *z*. For each action *a* these parts consist of the alternative causality conditions of *a* in which *z* is mentioned, and the attribute constraints that are associated with these conditions. In case *z* is not mentioned in any alternative causality condition of *a*, the start condition √ is defined as the causality condition of *a* in this causality context.

The causality context of *z* in behaviour *B* is denoted as *Con*(*B*, *z*). The actions in *Con*(*B*, *z*) are called the *context actions* of *z* and are denoted as *Ac*(*Con*(*B*, *z*)). Figure 8.5 depicts an example behaviour *B*. The causality contexts of actions *d* and *e* are:

$Con(B, e) = \{ \surd \rightarrow e,\ e \wedge \neg d \rightarrow c,\ e \wedge \neg c \rightarrow d \};$

$Con(B, d) = \{ \surd \rightarrow e,\ e \wedge \neg d \rightarrow c,\ e \wedge \neg c \rightarrow d,\ d \rightarrow b \}.$

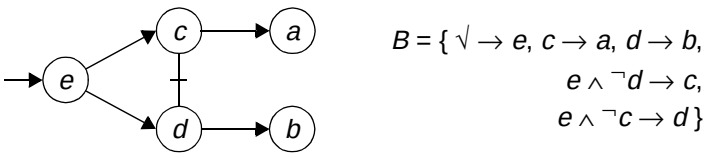


Figure 8.5: Example behaviour

An action may indirectly relate the occurrences of its context actions. Furthermore, an action may allow one of its context actions to indirectly refer to the attributes of another of its context actions. For example in Figure 8.5, action *d* defines an indirect relation between actions *e* and *b*, such that the occurrence of *b* indirectly depends on the occurrence of *e*. Furthermore, action *b* can refer indirectly to the attributes of action *e*, via action *d*.

In order to abstract from the influence of an inserted action *z* on the concrete behaviour one should be able to abstract from indirect relations between context actions of this inserted action, from indirect references between information, time and location attributes of these context actions and from probability associations of these context actions that involve inserted action *z*, i.e.:

- alternative causality conditions of context actions of inserted action *z* that define indirect relations between these actions via *z*, should be replaced by equivalent conditions that define these action relations directly;
- information, time and location attribute constraints of context actions of inserted action *z* that define indirect references between the attributes of these actions via the attributes of *z*, should be replaced by equivalent constraints that define these references directly;
- probability attribute constraints of context actions of inserted action *z* that define probability associations involving *z*, should be replaced by equivalent constraints that are defined in terms of probability associations only involving these context actions.

8.2.2 Method

The following steps define a method to determine the abstraction of some concrete behaviour *B*, called *B'*, which abstracts from inserted action *z*:

1. Determine the causality context of inserted action *z*, which is briefly denoted as *BC*, such that: $BC = Con(B, z)$;
2. Determine the alternative behaviours in *BC*, which are denoted as *BAs*, using the method presented in Section 8.2.3;
3. For each alternative behaviour $BA \in BAs$, determine abstract behaviour *BA'* which abstracts from inserted action *z*. This step involves the following activities:
   (i) determine BA_{Γ}' from BA_{Γ}, which abstracts from indirect action relations via *z*, using the method presented in Section 8.2.4;
   (ii) determine $BA_{\iota\tau\lambda}'$ from $BA_{\iota\tau\lambda}$, which abstracts from indirect references between information, time and location attributes via the attributes of *z*, using the method presented in Section 8.2.5;
   (iii) determine BA_{υ}' from BA_{υ}, which abstracts from indirect uncertainty associations involving *z*, using the method presented in Section 8.2.6;
   (iv) compose *BA'* from BA_{Γ}', $BA_{\iota\tau\lambda}'$ and BA_{υ}';
4. Determine the abstraction of *BC*, denoted as *BC'*, which abstracts from inserted action *z*. The abstraction of *BC* is defined in two steps:

(i) determine the disjunction of all *BA'*, with $BA \in BAs$, such that:
$BC' = \square_{\iota\tau\lambda\upsilon}\{BA' \mid BA \in BAs\}$,
where the disjunction operator $\square_{\iota\tau\lambda\upsilon}$ is defined in Section 8.2.7;

(ii) simplify *BC'* by integrating equivalent alternative causality conditions;

5. Determine abstract behaviour *B'*, which is obtained by replacing causality context *BC* in behaviour *B* by its abstraction *BC'*.

The method outlined above defines the abstraction of behaviour *B* as the disjunction of the abstractions of each alternative behaviour in *B*. This approach is based on the property that a behaviour can be decomposed into alternative behaviours, as discussed in Chapter 5. It is easier to define abstraction rules for conjunctions of action relations than for combinations of conjunctions and disjunctions of action relations. Since alternative behaviours consist exclusively of conjunctions of action relations, we first determine the abstraction of each alternative behaviour in *B*, and subsequently combine these abstractions to obtain the abstraction of *B*.

This chapter uses an extended interpretation of the notion of alternative behaviour w.r.t. the definitions in Chapters 4 and 5. An alternative behaviour is extended with the information, time, location and uncertainty attribute constraints that are associated with the alternative causality condition of each action in this behaviour. We use the term alternative behaviour in this chapter to denote this extended interpretation.

The method outlined above is restricted to behaviours using the uncertainty attribute. Sections 8.2.9 and 8.2.10 discuss additional methods for the abstraction of behaviours using the simple and extended integral probability attribute, respectively.

8.2.3 Alternative behaviours

The following steps define a method to determine the alternative behaviours *BAs* in *BC*:

1. define $BAs_\Gamma = Alt(BC_\Gamma)$, using function *Alt* as defined in Section 5.3.2. BAs_Γ defines the causality condition part of each alternative behaviour that can be identified in *BC*;
2. define for each $BA_\Gamma \in BAs_\Gamma$, the corresponding uncertainty attribute part BA_υ as follows: $BA_\upsilon = \{\langle a, \gamma_a, \pi_a(\gamma_a)\rangle \mid \langle a, \{\gamma_a\}\rangle \in BA_\Gamma\}$;
3. define for each $BA_\Gamma \in BAs_\Gamma$, the corresponding mixed attribute part $BA_{\iota\tau\lambda}$ as follows: $BA_{\iota\tau\lambda} = \{\langle\langle a, \mathrm{I}_a, \mathrm{T}_a, \Lambda_a, \varsigma_a\rangle, \langle I\text{-}Refs_a, T\text{-}Refs_a, L\text{-}Refs_a, ITL\text{-}Refs_a\rangle, \langle I\text{-}Caus_a, T\text{-}Caus_a, L\text{-}Caus_a, ITL\text{-}Caus_a\rangle \mid \langle a, \{\gamma_a\}\rangle \in BA_\Gamma\}$;
4. for each $BA_\Gamma \in BAs_\Gamma$, define *BA* as the composition of BA_Γ and the corresponding $BA_{\iota\tau\lambda}$ and BA_υ. Set *BAs* consists of all *BA* that can be obtained this way.

In the decomposition of some behaviour *BC* into alternative behaviours, we can use the choice relation as a basic action relation, or decompose this relation further into asymmetric exclusion relations. This makes no difference for the abstract behaviour that is obtained using the method of Section 8.2.2. We use the choice relation as a basic action relation as much as possible, since it is more concise and intuitive.

Figure 8.6 illustrates the decomposition of behaviour *BC*, which defines the interleaving of two actions *c* and *d*. We can represent the combination of disabling conditions $\neg d$ and $\neg c$ of actions *c* and *d*, respectively, by a choice relation, as in alternative behaviour *BA2*, or instead, represent this combination of conditions by the disjunction of two complementary asymmetric exclusion relations as in alternative behaviours *BA4* and *BA5*.

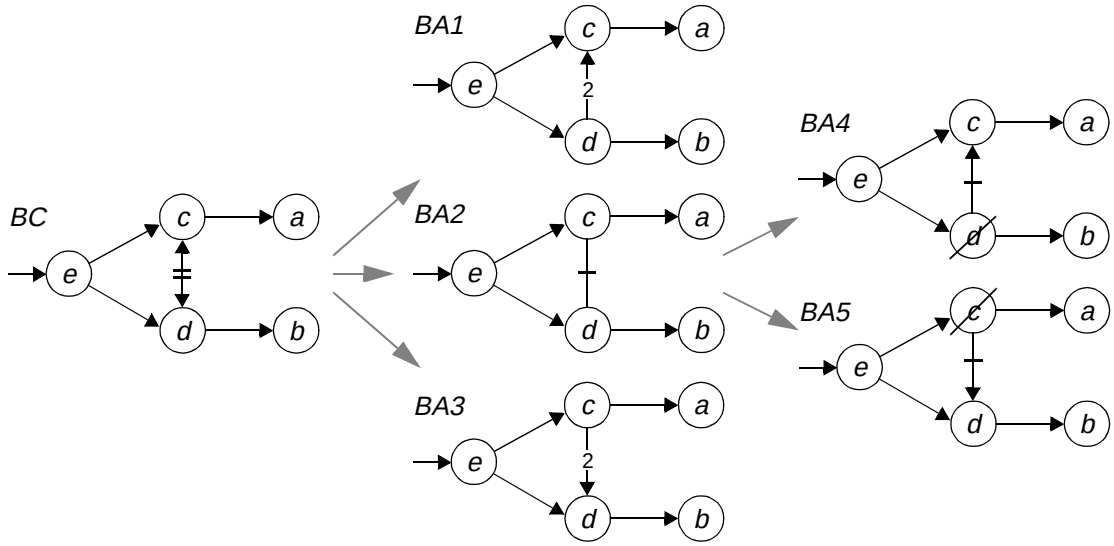


Figure 8.6: Alternative behaviours

Impossible conditions

The occurrence of action *d* in alternative behaviour *BA4* of Figure 8.6 is impossible. This has been represented so far in terms of our textual notation as $\dagger \rightarrow d$. In this chapter, we use the following notation to represent the causality relation of *d* in this case:

$e \wedge \neg c \rightarrow d\ [\upsilon_d(e \wedge \neg c) = imp]$,

where uncertainty value *imp* is introduced to explicitly represent that the occurrence of *d* due to condition $e \wedge \neg c$ is impossible, i.e., *d* is disabled by *c* in this alternative behaviour.

Similar to the graphical notation, this textual notation allows one to represent the cause for the impossibility of the occurrence of action *d*. This information is used in the method to abstract from uncertainty associations involving inserted actions, as discussed in Section 8.2.6.

8.2.4 Indirect action relations

The following steps define a method to determine the abstraction of the causality condition part of some alternative behaviour *BA*, by abstracting from indirect action relations via inserted action *z*:

1. define $BA_\Gamma' = BA_\Gamma$, such that each action name in BA_Γ' is appended with a prime, except for inserted action *z*;

2. replace the alternative causality conditions of actions in BA_Γ' that involve the elementary conditions z, $\neg z$ or $^{=}z$, by their indirect dependency closure, using the rules presented in Section 5.3 (Table 5.20);
3. remove inserted action z, which implies:
   (i) the elimination of the causality relation of z from BA_Γ'; and
   (ii) the elimination of elementary conditions z, $\neg z$ or $^{=}z$ from any alternative causality condition in BA_Γ'.

For example, consider causality context *Con*(*B*, *d*) as defined in Section 8.2.1, which contains a single alternative behaviour BA_Γ, where BA_Γ = *Con*(*B*, *d*). Figure 8.7(i) depicts the initial definition of BA_Γ', which is the result of the first step of the method given above. Figure 8.7(ii) depicts an intermediate definition of BA_Γ', which explicitly defines all indirect dependencies in BA_Γ' via inserted action *d*. Figure 8.7(iii) depicts the final definition of BA_Γ', in which inserted action *d* has been removed.

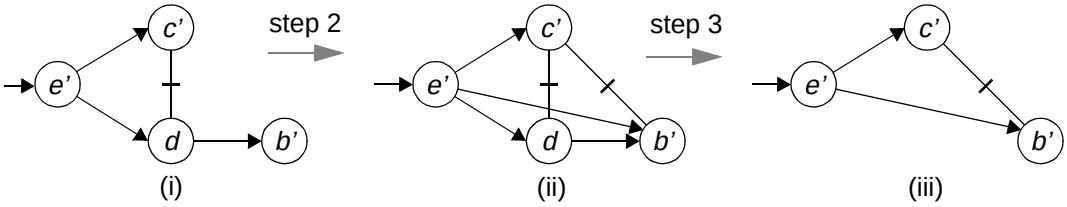


Figure 8.7: Abstraction from indirect action relations via inserted action *d*

The following example illustrates the embedding of the method outlined above in the method of Section 8.2.2.

Figure 8.8(i) depicts the step-wise abstraction from inserted action *z* in behaviour *B*, which defines a disabling relation between actions *a* and *z*. The causality context of *z* is equal to behaviour *B*, i.e., *Con*(*B*, *z*) = *B*. Step I in Figure 8.8(i) identifies alternative behaviours *BA1* and *BA2*, appends the reference actions in *BA1* and *BA2* with a prime, and defines the indirect dependency closures of *BA1* and *BA2*. This step comprises steps 1 and 2 of the method in Section 8.2.2, and steps 1 and 2 of the method outlined above. Step II in Figure 8.8(i) defines the abstract alternative behaviours *BA1'* and *BA2'* by removing inserted action *z* from *BA1* and *BA2*, respectively. This step corresponds to step 3 in the method outlined above. Finally, step III in Figure 8.8(i) defines the abstraction of *B* as the disjunction of *BA1'* and *BA2'*. This step corresponds to step 4 of the method in Section 8.2.2; step 5 of this method can be omitted.

Figure 8.8(ii) depicts behaviour *BS*, which defines a sync-disabling relation between actions *a* and *z*. This relation extends the disabling relation between *a* and *z* in behaviour *B* of Figure 8.8(i) with an alternative synchronization relation. This renders one additional alternative behaviour *BA3* and its abstraction *BA3'*. Consequently, the abstraction of *BS* is obtained as the disjunction of *BA1'*, *BA2'* and *BA3'*. At this point, a choice exists between interpreting the enabling relation between actions *a'* and *b'* in *BA3'* as a one-sided enabling relation, which renders abstract behaviour *BS'*, or as a two-sided enabling relation, which renders abstract behaviour *B'*. The textual representations of *B'* and *BS'* are:

$B' = \{\surd \rightarrow c', c' \wedge \neg b' \rightarrow a', (c' \wedge \neg a') \vee (c' \wedge a') \rightarrow b'\}$; and

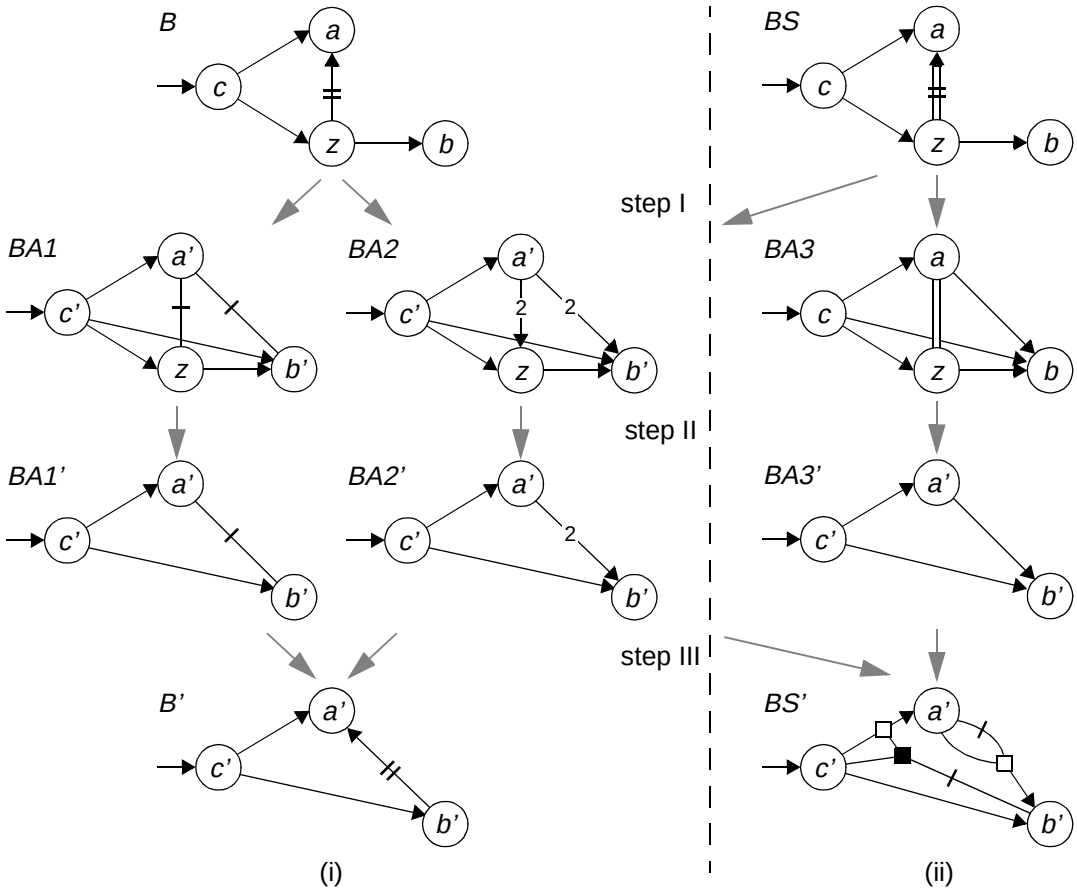


Figure 8.8: Abstraction from indirect action relations via inserted action *z*

$BS' = \{\surd \rightarrow c', c' \vee (c' \wedge \neg b') \rightarrow a', (c' \wedge \neg a') \vee (c' \wedge a') \rightarrow b'\}$.

Behaviours *B'* and *BS'* allow the same executions. However, *B'* defines a two-sided relation between *a'* and *b'*, whereas *BS'* defines a combination of a one-sided and a two-sided relation between *a'* and *b'*.

In general, we assume that an enabling relation between two actions *a* and *b* should be interpreted as a two-sided enabling relation in case another alternative behaviour exists that defines a distinct relation between *a* and *b*, and should be interpreted as a one-sided enabling relation otherwise. This rule is motivated by current experience in the application of our method. A more fundamental argumentation is left for further study. Based on this rule, behaviour *B'* is considered the abstraction of behaviour *BS* in Figure 8.8.

8.2.5 Indirect information, time and location references

This section presents a method to abstract from indirect references between attributes of the context actions of inserted action *z*. Two examples are presented to introduce and motivate some of the steps and rules of this method. The complete method is presented afterwards.

Figure 8.9 illustrates the abstraction of inserted action *z*, which is assumed to be obtained by causality refinement. We abstract from the reference in the information attribute of action *a* to the information attribute of *z* by simply substituting the information attribute constraint

of z in the information attribute constraint of a. However, such a simple substitution in case of the reference in the time attribute of a to the time attribute of inserted action z would render the time constraint $[\tau_{a'} < \tau_{b'} + 5]$, which is an incorrect abstraction. In order to obtain correct abstractions, implicit time constraints have to be considered when abstracting from references to the time attribute of inserted actions. In the example of Figure 8.9, the constraint $[\tau_z = \tau_{b'} + 2]$ should also be substituted in the implicit time constraint $\tau_z < \tau_{a'}$, which renders the additional time constraint $[\tau_{b'} + 2 < \tau_{a'}]$.

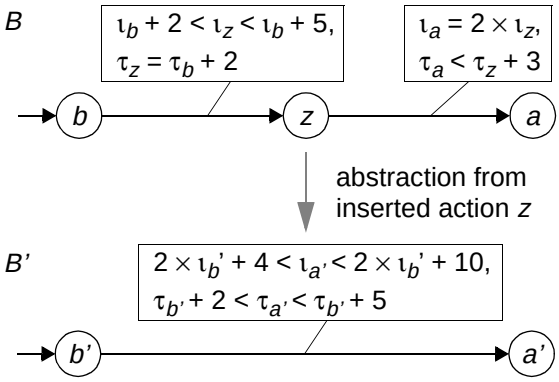


Figure 8.9: Abstraction from indirect information and time references

Figure 8.10 illustrates the abstraction of inserted action z, which is assumed to be obtained by action refinement. Reference actions a, b and c correspond to abstract actions a', b' and c', respectively, and concrete actions a and z model abstract action a' in more detail. We abstract from the reference in the information attribute of final action a to the information attribute of z by substituting $\iota_z = g(\iota_c)$ in $\iota_a = f(\iota_z)$. In addition, the information attribute of a' is replaced by tuple $\langle \iota 1_a, \iota 2_a \rangle$, which represents the information values established in a and z, respectively, since z is referred to by b via a. Consequently, the refinement of abstract action a' into activity A is characterized by the distribution of the establishment of values $\iota 1_{a'}$ and $\iota 2_{a'}$ over concrete actions a and z, respectively. Accordingly, the information references of action b to the information attributes of actions a and z are replaced by references to $\iota 1_{a'}$ and $\iota 2_{a'}$ of abstract action a', respectively.

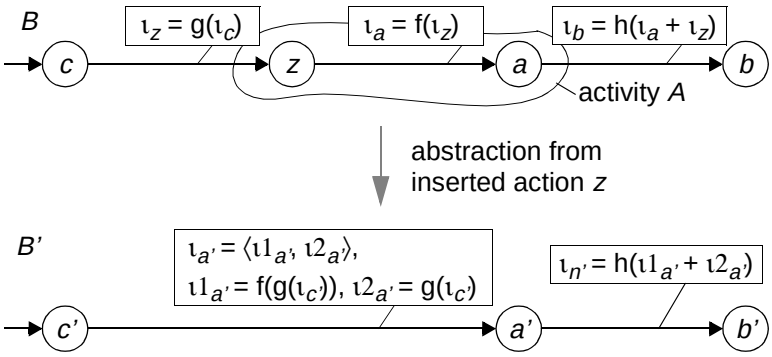


Figure 8.10: Abstraction from indirect information references

The following steps define a method to determine the abstraction of the mixed attribute part of some alternative behaviour BA, by abstracting from indirect references between information, time and location attributes of the context actions of inserted action z via the attributes of z:

1. define $BA_{\iota\tau\lambda}' = BA_{\iota\tau\lambda}$, such that each action name in $BA_{\iota\tau\lambda}'$ is appended with a prime, except for inserted action *z*;
2. remove the information, time and location attributes of *z* from $BA_{\iota\tau\lambda}'$, and eliminate disabling condition $\neg z$ and synchronization condition $^{=}z$ from any information, time and location attribute association in $BA_{\iota\tau\lambda}'$;

In case inserted action *z* is obtained by causality refinement:

3. for each information, time and location attribute association in $BA_{\iota\tau\lambda}'$ that involves enabling condition *z*, perform the following activities:
   (i) eliminate enabling condition *z*;
   (ii) replace references to the information and location attributes of *z* by their possible values or constraints;
   (iii) replace references to the time attribute of *z* by its possible values or constraints, taking into account implicit time constraints;

In case inserted action *z* is part of some activity *A* obtained by action refinement:

4. for each information, time and location attribute association of some action *a* in $BA_{\iota\tau\lambda}$, that involves enabling condition *z*, perform the following activities:
   (i) eliminate enabling condition *z*;

   in case action *a* is an *inserted action* of activity *A*:

   (ii) replace references to the information and location attributes of *z* by their possible values or constraints;
   (iii) replace references to the time attribute of *z* by its possible values or constraints, taking into account implicit time constraints;

   in case action *a* is a *final action* of activity *A*:

   (iv) replace references to the information and location attributes of *z* by their possible values or constraints;
   (v) replace references to the time attribute of *z* by its possible values or constraints, taking into account implicit time constraints;
   (vi) extend the information attribute of *a'* with the information, time and location attribute of *z*, in case these attributes are referred to via *a*, such that:
   $\iota_{a'} = \langle \iota_a, \langle \iota_z, \tau_z, \lambda_z \rangle \rangle$.
   In addition, add the corresponding attribute constraints to *a'*;

   in case action *a* is an action *outside* activity *A* that refers via some final action *b* of *A* to inserted action *z*:

   (vii) replace references to the information, time and location attributes of *z* by corresponding references to the information attribute of final action *b*, which is defined as $\iota_{b'} = \langle \iota_b, \langle \iota_z, \tau_z, \lambda_z \rangle \rangle$, such that:
   - a reference to ι_z is replaced by a reference to $\iota_{b'}$.2.1 (the first element of the second element of $\iota_{b'}$);
   - a reference to τ_z is replaced by a reference to $\iota_{b'}$.2.2 (the second element of

the second element of $\iota_{b'}$);
- a reference to λ_z is replaced by a reference to $\iota_{b'}$.2.3 (the third element of the second element of $\iota_{b'}$).

Figure 8.9 illustrates step 3 and Figure 8.10 illustrates step 4 of this method.

Indirect reference relations can only be defined in combination with indirect enabling relations. Therefore, the method given above should be applied in combination with the method in Section 8.2.4, as performed in the examples of Figures 8.9 and 8.10.

8.2.6 Uncertainty associations involving inserted action *z*

This section presents a method to determine the abstraction of uncertainty associations of the context actions of inserted action *z* in which *z* is mentioned (involved). Rules are developed to determine the abstraction of uncertainty associations involving (i) enabling condition *z*, (ii) synchronization condition $^{=}z$ and (iii) disabling condition $\neg z$. These rules are combined into a complete method at the end of this section.

The notions of minimal definition and (indirect) dependency closure of some alternative causality condition γ are used below. These notions have been explained in Section 5.3.

Uncertainty associations involving enabling condition *z*

Figure 8.11 depicts the abstraction of uncertainty association $\upsilon_a(z)$, which involves enabling condition *z*. Action *a* indirectly depends on enabling condition *b* via inserted action *z*, such that $\upsilon_a(z) = \upsilon_a(z \wedge b)$. Therefore, when abstracting from *z*, the uncertainty that *a'* occurs when condition *b'* is satisfied must be determined, which is represented by uncertainty association $\upsilon_{a'}(b')$. This uncertainty association models the uncertainty that *z* occurs after *b* has occurred and the uncertainty that *a* occurs after *z* has occurred. Action *a* must occur after *b* has occurred if and only if (i) *z* must occur after *b* has occurred, and (ii) *a* must occur after *z* has occurred. This implies that $\upsilon_{a'}(b')$ is determined by the composition of $\upsilon_a(z \wedge b)$ and $\upsilon_b(z)$, such that: $\upsilon_{a'}(b') = \mathit{must} \Leftrightarrow \upsilon_z(b) = \mathit{must} \wedge \upsilon_a(z) = \mathit{must}$.

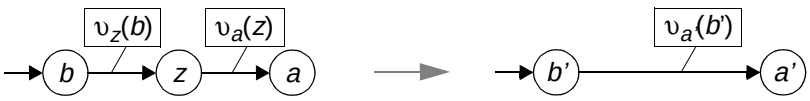


Figure 8.11: Abstraction of uncertainty associations involving enabling condition *z*

The general case of an uncertainty association involving enabling condition *z* is represented for some action *a* with alternative causality condition γ_a by the following causality relations:

$z \wedge \gamma \rightarrow a\ [\upsilon_a(z \wedge \gamma)]$, # with $\gamma_a^{+} = z \wedge \gamma$

$\gamma_z \rightarrow z\ [\upsilon_z(\gamma_z)]$.

The abstraction of the causality relation of action *a* is denoted as: $\gamma' \rightarrow a'[\upsilon_{a'}(\gamma')]$, where γ' is derived from γ by appending each action name with a prime. We distinguish the case in which enabling condition *z* is an element of (one of) the minimal definition(s) of γ_a, from the case in which it is not.

Enabling condition z is not an element of (one of) the minimal definition(s) of γ_a, in case a also depends indirectly on enabling condition z via a third action b. This indirect dependency is represented by an enabling or synchronization condition in γ_a that involves b. In this case, the value of uncertainty association $\upsilon_a(\gamma')$ is equal to the value of $\upsilon_a(z \wedge \gamma)$, since the uncertainty introduced by z and its causality condition is already represented by the enabling or synchronization condition in γ_a involving b, via which a depends indirectly on z.

In case enabling condition z is an element of (one of) the minimal definition(s) of γ_a, uncertainty association $\upsilon_a(\gamma')$ has to consider the following sources of uncertainty:

- the uncertainty introduced by reference action a, i.e., $\upsilon_a(z \wedge \gamma)$;
- the uncertainty introduced by inserted action z, i.e., $\upsilon_z(\gamma_z)$; and
- the possible increase in uncertainty w.r.t. γ_a caused by considering γ_z.

The following properties hold for the alternative causality conditions of a and z:

1. enabling conditions of inserted action z are also enabling conditions of action a, i.e., $\{b \mid b \in C(\gamma_z)\} \subseteq C(\gamma_a^+)$;
2. synchronization conditions of inserted action z are enabling conditions of action a, i.e., $\{b \mid {}^{=}b \in C(\gamma_z)\} \subseteq C(\gamma_a^+)$;
3. disabling conditions of inserted action z are also disabling conditions of action a, i.e., $\{\neg b \mid \neg b \in C(\gamma_z)\} \subseteq C(\gamma_a^+)$, in case these disabling conditions are part of an (a)symmetric exclusion relation between z and a third action b.

The first property implies that when considering the enabling conditions in γ_z the uncertainty w.r.t. γ_a does not have to be increased, since these conditions are also defined as enabling conditions in γ_a^+.

The second property implies that any synchronization action in γ_z is an enabling action in γ_a^+, including z itself. The increase in uncertainty caused by these synchronization conditions w.r.t. γ_a, is completely defined by $\upsilon_z(\gamma_z)$ and by $\upsilon_b(\gamma_b)$ for any b in $\{b \mid {}^{=}b \in C(\gamma_z)\}$, since $\upsilon_z(\gamma_z) = \upsilon_b(\gamma_b)$ due to the reciprocal aspect of the synchronization conditions. Consequently, when abstracting from z, the uncertainty introduced by the synchronization conditions in γ_z is still represented by the uncertainty associations of enabling actions $\{b' \mid {}^{=}b \in C(\gamma_z)\}$ in $\gamma_a'^{+}$, and therefore it does not have to be considered again in the uncertainty association of action a' itself.

The third property implies that a disabling condition in γ_z, which is part of an (a)symmetric exclusion relation between z and a third action b, does not increase the uncertainty w.r.t. γ_a. In case γ_z is part of a two-sided enabling relation between z and a third action b, the corresponding disabling condition in γ_z does not increase the uncertainty w.r.t. γ_a, since b depends on z and therefore it can not disable z.

Consequently, condition γ_z does not increase the uncertainty w.r.t. causality condition γ_a. This implies that uncertainty association $\upsilon_a(\gamma')$ is completely determined by $\upsilon_a(z \wedge \gamma)$ and $\upsilon_z(\gamma_z)$. Therefore, $\upsilon_a(\gamma')$ is defined by the following rule:

$\upsilon_a(\gamma') = \upsilon_a(z \wedge \gamma)$, if (i) $\upsilon_z(\gamma_z) = \mathit{must}$; or

(ii) some b exists such that ${}^{=}b \in C(\gamma_z^{+})$, ${}^{=}z \in C(\gamma_b^{+})$ and $b \in C(\gamma_a^{+})$;

$= imp$, if $\upsilon_a(z \wedge \gamma) = imp$ or $\upsilon_z(\gamma_z) = imp$;

$= may$, otherwise.

Figure 8.12 depicts an example of the abstraction of an uncertainty association $\upsilon_a(z)$. Enabling condition z is equivalent to condition $z \wedge c \wedge \neg b \wedge d$, such that the abstraction of $\upsilon_a(z)$ is defined as:

$\upsilon_a(c' \wedge \neg b' \wedge d') = \upsilon_a(z)$, since γ_z does not increase the uncertainty w.r.t. γ_a, and the uncertainty introduced by z is represented by $\upsilon_d(c' \wedge \neg b')$.

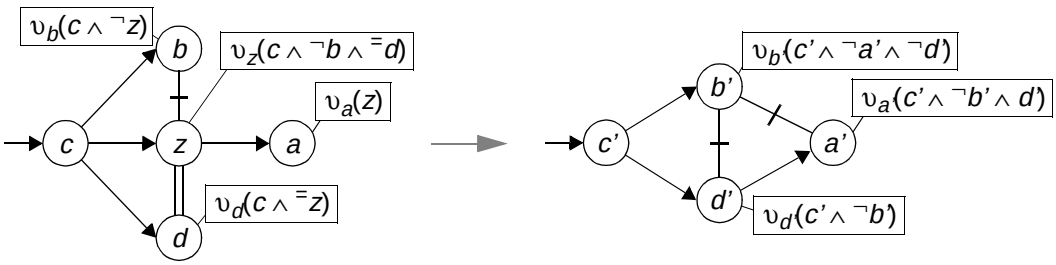


Figure 8.12: Abstraction of uncertainty associations involving enabling condition z (2)

Uncertainty associations involving synchronization condition ${}^{=}z$

Figure 8.12 also depicts the abstraction of uncertainty association $\upsilon_d(c \wedge {}^{=}z)$, which involves synchronization condition ${}^{=}z$. Causality condition $c \wedge {}^{=}z$ is equivalent to condition $c \wedge \neg b \wedge {}^{=}z$, such that $\upsilon_d(c \wedge {}^{=}z) = \upsilon_d(c \wedge \neg b \wedge {}^{=}z)$. Therefore, when abstracting from z, the abstraction of uncertainty association $\upsilon_d(c \wedge {}^{=}z)$ is defined as $\upsilon_d(c' \wedge \neg b')$. The value of $\upsilon_d(c' \wedge \neg b')$ is equal to the value of $\upsilon_d(c \wedge \neg b \wedge {}^{=}z)$, since the uncertainty introduced by synchronization conditions ${}^{=}z$ and ${}^{=}d$ is contained in $\upsilon_d(c \wedge \neg b \wedge {}^{=}z)$, and is the same for both actions d and z, i.e., $\upsilon_d(c \wedge \neg b \wedge {}^{=}z) = \upsilon_z(c \wedge \neg b \wedge {}^{=}d)$.

The general case of an uncertainty association involving synchronization condition ${}^{=}z$ is represented for some action a with alternative causality condition γ_a by the following causality relations:

${}^{=}z \wedge \gamma \rightarrow a\ [\upsilon_a({}^{=}z \wedge \gamma)]$, # with $\gamma_a^{+} = {}^{=}z \wedge \gamma$

${}^{=}a \wedge \gamma \rightarrow z\ [\upsilon_z({}^{=}a \wedge \gamma)]$ # with $\gamma_z^{+} = {}^{=}a \wedge \gamma$.

The abstraction of the causality relation of action a is denoted as: $\gamma' \rightarrow a'\ [\upsilon_a(\gamma')]$. Uncertainty association $\upsilon_a(\gamma')$ has to consider the following sources of uncertainty:

- the uncertainty introduced by reference action a, i.e., $\upsilon_a({}^{=}z \wedge \gamma)$;
- the uncertainty introduced by inserted action z, i.e., $\upsilon_z({}^{=}a \wedge \gamma)$;
- the uncertainty introduced by synchronization conditions ${}^{=}z$ and ${}^{=}a$.

The uncertainty introduced by synchronization conditions ${}^{=}z$ and ${}^{=}a$ is contained in $\upsilon_a({}^{=}z \wedge \gamma)$ and $\upsilon_z({}^{=}a \wedge \gamma)$, with $\upsilon_a({}^{=}z \wedge \gamma) = \upsilon_z({}^{=}a \wedge \gamma)$, due to the reciprocal aspect of synchronization conditions. This implies the following rule:

$\upsilon_{a'}(\gamma') = \upsilon_a(\bar{z} \wedge \gamma) = \upsilon_z(\bar{a} \wedge \gamma)$.

Uncertainty associations involving disabling condition ¬z

Figure 8.13 depicts the abstraction of uncertainty association $\upsilon_c(d \wedge \neg z)$, which involves disabling condition ¬z. Action *c* indirectly depends on disabling condition ¬*b* via inserted action *z*, such that $\upsilon_c(d \wedge \neg z) = \upsilon_c(d \wedge \neg z \wedge \neg b)$, with $d \wedge \neg z \equiv d \wedge \neg z \wedge \neg b$. Therefore, when abstracting from *z*, the uncertainty that *c'* occurs when condition $d' \wedge \neg b'$ is satisfied must be determined, i.e., $\upsilon_{c'}(d' \wedge \neg b')$.

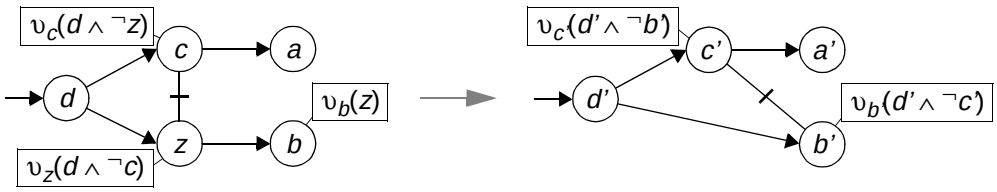


Figure 8.13: Abstraction from uncertainty associations involving disabling condition ¬*z*

Uncertainty association $\upsilon_{c'}(d' \wedge \neg b')$ comprises the uncertainty of the occurrence of *c* due to the possible disabling of *c* by inserted action *z*. This uncertainty is (partly) represented by disabling condition ¬*b'*, since *z* enables *b*. The following two cases are distinguished:

1. action *b* occurs in every execution in which action *z* occurs, i.e., $\upsilon_b(z) = \mathit{must}$. In this case actions *b* and *c* completely inherit the choice between actions *c* and *z*. This implies that the uncertainty of the occurrence of *c* due to its possible disabling by *z* is completely represented by condition ¬*b'*, such that: $\upsilon_{c'}(d' \wedge \neg b') = \upsilon_c(d \wedge \neg z \wedge \neg b)$;
2. action *b* may occur in some, but not all executions in which *z* occurs, i.e., $\upsilon_b(z) = \mathit{may}$. In this case actions *b* and *c* only partly inherit the choice between actions *c* and *z*. This implies that the uncertainty of the occurrence of *c* due to its possible disabling by *z* is only partly represented by condition ¬*b*', and the 'remaining' uncertainty has to be compensated for by $\upsilon_{c'}(d' \wedge \neg b')$, such that: $\upsilon_{c'}(d' \wedge \neg b') = \mathit{may}$.

Two general cases of an uncertainty association involving disabling condition ¬z are distinguished for some action *a* with alternative causality condition γ_a:

1. an (a)symmetric exclusion relation is defined between *z* and *a*:
   $\neg z \wedge \gamma 1 \rightarrow a\ [\upsilon_a(\neg z \wedge \gamma 1)]$, # with $\gamma_a^+ = \neg z \wedge \gamma 1$
   $\neg a \wedge \gamma 2 \rightarrow z\ [\upsilon_z(\neg a \wedge \gamma 2)]$ # with $\gamma_z^+ = \neg a \wedge \gamma 2$
   (asymmetric exclusion relations are represented using uncertainty value *imp*); or
1. a two-sided enabling relation is defined between *z* and *a*:
   $\neg z \wedge \gamma 1 \rightarrow a\ [\upsilon_a(\neg z \wedge \gamma 1)]$, # with $\gamma_a^+ = \neg z \wedge \gamma 1$
   $a \wedge \gamma 2 \rightarrow z\ [\upsilon_z(\neg a \wedge \gamma 2)]$ # with $\gamma_z^+ = a \wedge \gamma 2$.

The abstraction of the causality relation of action *a* is denoted as: $\gamma 1' \rightarrow a'\ [\upsilon_{a'}(\gamma 1')]$. We distinguish the case in which disabling condition ¬z is an element of (one of) the minimal definition(s) of γ_a, from the case in which it is not.

Disabling condition ¬z is not an element of (one of) the minimal definition(s) of γ_a, in case *a* also depends indirectly on disabling condition *z* via a third action *b*. This indirect depend-

ency is represented by an enabling or synchronization condition in γ_a that involves *b*. In this case, the value of uncertainty association $\upsilon_a(\gamma 1')$ is equal to the value of $\upsilon_a(\neg z \wedge \gamma 1)$, since the uncertainty introduced by *z* and its causality condition is already represented by the enabling or synchronization condition in γ_a involving *b*, via which *a* depends indirectly on disabling condition $\neg z$.

In case disabling condition $\neg z$ is an element of (one of) the minimal definition(s) of γ_a, uncertainty association $\upsilon_a(\gamma 1')$ has to consider the following sources of uncertainty:

- the uncertainty introduced by reference action *a*, i.e., $\upsilon_a(\neg z \wedge \gamma 1)$;
- the uncertainty introduced by disabling condition $\neg z$, representing the possible disabling of *a* by *z*.

In the case of a two-sided enabling relation or an asymmetric exclusion relation between *z* and *a* in which *z* is disabled, the latter source of uncertainty does not have to be considered, since *z* depends on the occurrence of *a* or *z* is impossible, respectively. In this case, $\upsilon_a(\gamma 1')$ is completely determined by $\upsilon_a(\neg z \wedge \gamma 1)$.

In the case of a symmetric exclusion relation between *z* and *a* or an asymmetric exclusion relation between *z* and *a* in which *a* is disabled, the uncertainty introduced by $\neg z$ can only be represented completely by another disabling condition $\neg b$, for an action *b* that completely inherits the (a)symmetric exclusion relation between *a* and *z*, if such an action exists. This is possible: (i) in case *b* is enabled by *z*, such that *b* must occur in every execution in which *z* occurs (corresponding to case 1 in the example of Figure 8.13), or (ii) in case a synchronization relation is defined between *b* and *z*. In both cases, actions *a'* and *b'* completely inherit the exclusion relation between *a* and *z*, such that the value of $\upsilon_a(\gamma 1')$ is determined by $\upsilon_a(\neg z \wedge \gamma 1)$. However, in case no such action *b* exists, the uncertainty introduced by $\neg z$ should be considered by assigning the value *may* to $\upsilon_a(\gamma 1')$.

The above implies that $\upsilon_a(\gamma 1')$ is defined by the following rule:

$\upsilon_a(\gamma') = \upsilon_a(\neg z \wedge \gamma)$, if (i) action *z* depends on action *a*, i.e., $a \in C(\gamma_z^+)$; or
(ii) action *z* is impossible, i.e., $\upsilon_z(\gamma_z) = imp$; or
(iii) a third action *b* exists, such that:
$z \in C(\gamma_b^+)$, $\neg b \in C(\gamma_a^+)$, and *b* must occur after *z*; or
$^{=}z \in C(\gamma_b^+)$, $^{=}b \in C(\gamma_z^+)$ and $\neg b \in C(\gamma_a^+)$; or
(iv) $\upsilon_a(\neg z \wedge \gamma) = imp$;

$= may$, otherwise.

Figure 8.14 depicts two additional examples of the abstraction of uncertainty associations involving disabling condition $\neg z$.

The abstraction of $\upsilon_a(c \wedge \neg z)$ in Figure 8.14(i) is defined as: $\upsilon_a(c') = may$. Uncertainty association $\upsilon_a(c')$ represents the possibility that the occurrence of action *a* is disabled by inserted action *z*.

The abstraction of $\upsilon_c(d \wedge \neg z)$ in Figure 8.14(ii) is defined as:

$\upsilon_c(d' \wedge \neg b') = \upsilon_c(d \wedge \neg z)$, if $\upsilon_b(z \wedge e) = must$ and $\upsilon_e(\surd) = must$;

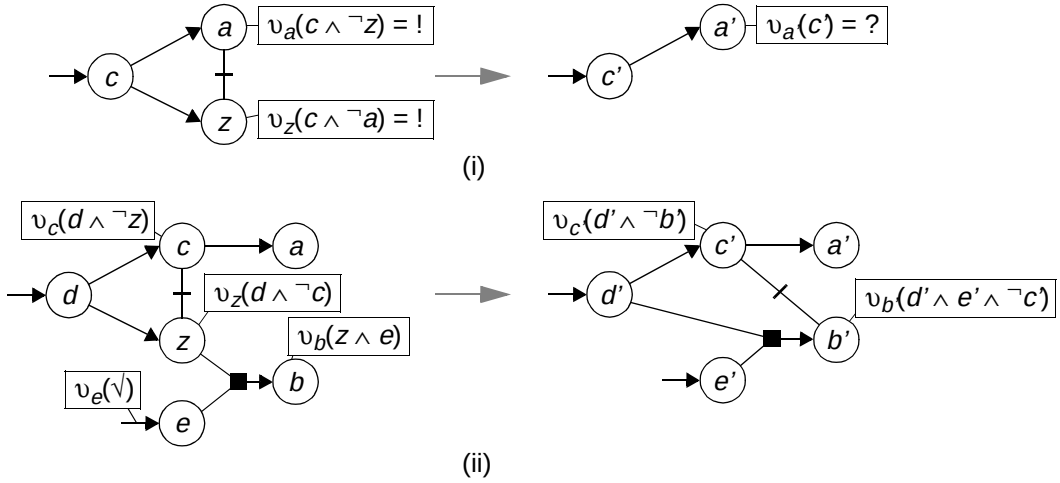


Figure 8.14: Abstraction of uncertainty associations involving disabling condition $\neg z$ (2)

$= \mathit{may}$ otherwise.

In this behaviour, a necessary condition for action *b* to occur in every execution in which inserted action *z* occurs is that action *e* must occur.

Complete method

Based on the rules presented above, the following steps define a method to determine the abstraction of the uncertainty attribute part of some alternative behaviour *BA*, which abstracts from inserted action *z*:

1. define $BA_\upsilon' = BA_\upsilon$, such that each action name in BA_υ' is appended with a prime, except for inserted action *z*;
2. remove the uncertainty association of *z* from BA_υ';
3. for each uncertainty association $\langle a', \gamma_{a'}, \upsilon_{a'}(\gamma_{a'})\rangle$ in BA_υ' that involves enabling condition *z*, such that $\gamma_{a'}^{+} = \gamma \wedge z$, replace $\upsilon_{a'}(\gamma_{a'})$ by $\upsilon_{a'}(\gamma)$, where the value of $\upsilon_{a'}(\gamma)$ is defined by the following rule:

   $\upsilon_{a'}(\gamma) = \upsilon_a(\gamma_a)$, if (i) $\upsilon_z(\gamma_z) = \mathit{must}$; or
   (ii) γ_z contains a synchronization condition; or
   (iii) enabling condition *z* is not an element of (one of) the minimal definition(s) of γ_a;

   $= \mathit{imp}$, if $\upsilon_a(\gamma_a) = \mathit{imp}$ or $\upsilon_z(\gamma_z) = \mathit{imp}$;

   $= \mathit{may}$, if otherwise;
4. for each uncertainty association $\langle a', \gamma_{a'}, \upsilon_{a'}(\gamma_{a'})\rangle$ in BA_υ' that involves synchronization condition $^{=}z$, such that $\gamma_{a'}^{+} = \gamma \wedge {}^{=}z$, replace $\upsilon_{a'}(\gamma_{a'})$ by $\upsilon_{a'}(\gamma)$, where the value of $\upsilon_{a'}(\gamma)$ is defined by the following rule:

   $\upsilon_{a'}(\gamma) = \upsilon_a(\gamma_a)$;

5. for each uncertainty association $\langle a', \gamma_{a'}, \upsilon_{a'}(\gamma_{a'})\rangle$ in BA_{υ}' that involves disabling condition $\neg z$, such that $\gamma_{a'}^{+} = \gamma \wedge \neg z$, replace $\upsilon_{a'}(\gamma_{a'})$ by $\upsilon_{a'}(\gamma)$, where the value of $\upsilon_{a'}(\gamma)$ is defined by the following rule:

   $\upsilon_{a'}(\gamma) = \upsilon_{a}(\gamma_{a})$, if (i) action z depends on the occurrence of a; or
   (ii) the occurrence of z is impossible; or
   (iii) a third action b exists which is related to z such that a' and b' inherit the (a)symmetric exclusion relation between a and z, and b occurs in every execution in which z occurs; or
   (iv) disabling condition $\neg z$ is not an element of (one of) the minimal definition(s) of γ_a; or
   (v) if $\upsilon_a(\gamma_a) = imp$;

   may, if otherwise.

8.2.7 Disjunctions of alternative behaviours

The determination of the abstraction of causality context *BC* as defined in step 5 of the method in Section 8.2.2, consists of the following two activities:

1. determine the disjunction of the abstractions of the alternative behaviours in *BC*;
2. integrate (disjunctions of) equivalent alternative causality conditions.

Disjunction operator

In order to be able to apply the disjunction operator defined in Section 5.3.2 in step 5 of the method of Section 8.2.2, we extend this operator to include behaviours with information, time, location and uncertainty attributes. The extended definition of disjunction operator $\square$ is denoted as $\square_{\iota\tau\lambda\upsilon}$.

Consider the disjunction of behaviours *B1* and *B2*, which contain action *a*. Both *B1* and *B2* may define the same alternative causality condition γ for *a*, but associate different attribute constraints [*C1*] and [*C2*], respectively, with this condition. In behaviour $B = B1 \square_{\iota\tau\lambda\upsilon} B2$ we want the disjunction of both conditions, and their associated attribute constraints, to be defined as $\gamma\,[C1] \vee \gamma\,[C2]$. This implies that the disjunction $\gamma \vee \gamma$ in B_{Γ} must not be integrated into the single condition γ, which would be performed implicitly in the pre-formal notation, since it uses the union operator to represent the disjunction of conditions. For this purpose, we extend alternative causality conditions with a unique label to make identical conditions distinguishable from each other. For example, the causality condition of *a* in *B* could be represented as $\langle\gamma, l1\rangle\,[C1] \vee \langle\gamma, l2\rangle\,[C2]$, where *l1* and *l2* represent unique labels.

The above implies that even in case [*C1*] = [*C2*], the disjunction $\gamma\,[C1] \vee \gamma\,[C2]$ is not integrated into the single condition $\gamma\,[C]$, with [*C*] = [*C1*] = [*C2*]. We deliberately separate the integration of disjunctions of equivalent conditions from the disjunction operation itself. The integration of disjunctions of equivalent conditions is discussed later on.

The disjunction operator $\square_{\iota\tau\lambda\upsilon}$ is defined below, assuming alternative causality conditions are extended with unique labels.

Definition 8.1 The disjunction of two behaviours *B1* and *B2*, with $Ac(B1) = Ac(B2)$, is defined as:

$$
\begin{aligned}
B1 \square_{\iota\tau\lambda\upsilon} B2 = \{ \; & \langle\langle a, \mathrm{I1} \cup \mathrm{I2}, \mathrm{T1} \cup \mathrm{T2}, \Lambda 1 \cup \Lambda 2, \varsigma 1 \cup \varsigma 2\rangle, \Gamma 1 \cup \Gamma 2, \upsilon 1 \cup \upsilon 2, \\
& \quad \langle \mathit{I\text{-}Refs1} \cup \mathit{I\text{-}Refs2}, \mathit{T\text{-}Refs1} \cup \mathit{T\text{-}Refs2}, \mathit{L\text{-}Refs1} \cup \mathit{L\text{-}Refs2}, \\
& \qquad \mathit{ITL\text{-}Refs1} \cup \mathit{ITL\text{-}Refs2}\rangle, \\
& \quad \langle \mathit{I\text{-}Caus1} \cup \mathit{I\text{-}Caus2}, \mathit{T\text{-}Caus1} \cup \mathit{T\text{-}Caus2}, \mathit{L\text{-}Caus1} \cup \mathit{L\text{-}Caus2}, \\
& \qquad \mathit{ITL\text{-}Caus1} \cup \mathit{ITL\text{-}Caus2}\rangle\rangle \\
& \mid a \in Ac(B1)\},
\end{aligned}
$$

with: $\langle\langle a, \mathrm{I1}, \mathrm{T1}, \Lambda 1, \varsigma 1\rangle, \Gamma 1, \upsilon 1, \langle \mathit{I\text{-}Refs1}, \mathit{T\text{-}Refs1}, \mathit{L\text{-}Refs1}, \mathit{ITL\text{-}Refs1}\rangle, \langle \mathit{I\text{-}Caus1}, \mathit{T\text{-}Caus1}, \mathit{L\text{-}Caus1}, \mathit{ITL\text{-}Caus1}\rangle\rangle \in B1$; and $\langle\langle a, \mathrm{I2}, \mathrm{T2}, \Lambda 2, \varsigma 2\rangle, \Gamma 2, \upsilon 2, \langle \mathit{I\text{-}Refs2}, \mathit{T\text{-}Refs2}, \mathit{L\text{-}Refs2}, \mathit{ITL\text{-}Refs2}\rangle, \langle \mathit{I\text{-}Caus2}, \mathit{T\text{-}Caus2}, \mathit{L\text{-}Caus2}, \mathit{ITL\text{-}Caus2}\rangle\rangle \in B2$. ■

Property 8.2 The disjunction operator $\square_{\iota\tau\lambda\upsilon} : \boldsymbol{B} \times \boldsymbol{B} \rightarrow \boldsymbol{B}$ is idempotent, commutative and associative.

Proof: These properties are inherited from the union operator $\cup$ on sets. ■

Integration of disjunctions of equivalent alternative causality conditions

The causality relation of an action *a* obtained by the application of the *or*-operator $\square_{\iota\tau\lambda\upsilon}$ may contain equivalent alternative causality conditions. In certain cases, a disjunction of equivalent alternative causality conditions can be integrated into a single alternative causality condition. These cases are determined, amongst others, by the information, time, location and uncertainty associations of the involved alternative causality conditions, and the relationship between these conditions in the concrete behaviour. Therefore, rules are needed to perform this integration. The systematic development of general integration rules, however, goes beyond the scope of this chapter. In the sequel, we consider the integration of equivalent alternative causality conditions on an ad-hoc basis by means of examples.

8.2.8 Examples

This section illustrates the method of Section 8.2.2 by means of two examples.

Example 1: unreliable connection

Figure 8.15 depicts the abstraction of concrete behaviour *B* into abstract behaviour *B'''*. Concrete action *c* models the establishment of a connection. Concrete actions *b* and *a* model the transfer of data segments. Concrete action *d* models the release of the connection, which may disable actions *b* and *a*, or only action *a*.

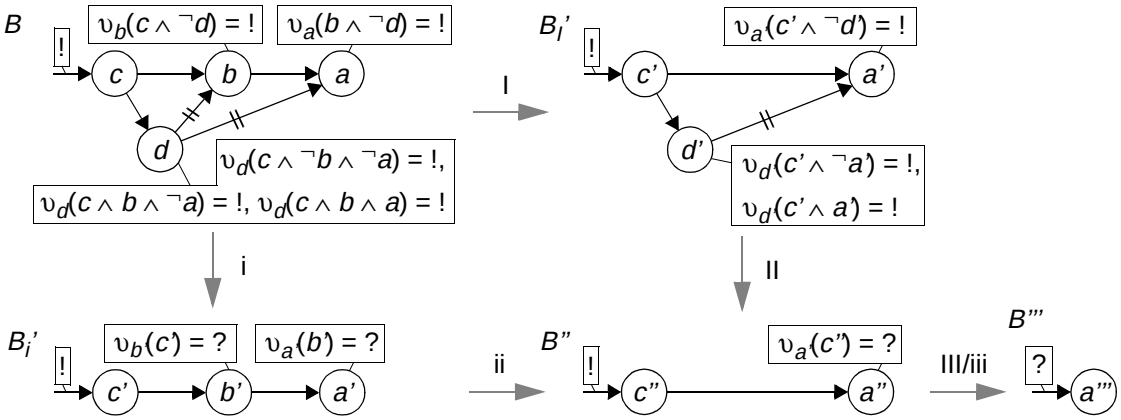


Figure 8.15: Unreliable connection

We want to abstract from actions *c*, *b* and *d* in behaviour *B* to determine the result that is established by *B* when this behaviour is considered as a single action. In this abstraction, the data segments modelled by actions *a* and *b* can be seen as a single data unit.

The abstraction of concrete behaviour *B* is determined via two alternative paths: one path consisting of abstraction steps I, II and III and the other path consisting of abstraction steps i, ii and iii. Both paths render the same abstract behaviour *B'''*.

Abstraction step I determines the abstraction of *B*, which is called B_I', by abstracting from inserted action *b*. The causality context of action *b* is equal to *B*, i.e., *Con*(*B*, *b*) = *B*. Behaviour *B* is decomposed into three alternative behaviours *BA1*, *BA2* and *BA3*. Figure 8.16 depicts these alternative behaviours and their abstractions $BA1_I$', $BA2_I$' and $BA3_I$', respectively. Abstract behaviour B_I' is defined by the disjunction of $BA1_I$', $BA2_I$' and $BA3_I$'. Furthermore, B_I' is simplified by integrating equivalent alternative causality conditions, which is straightforward in this case, since all uncertainty associations are assigned the *must* value. Behaviour B_I' models that either the entire data unit is delivered, as represented by abstract action *a'*, and the connection is subsequently released, or the disconnect activity disables the delivery of the entire data unit.

Abstraction step i determines the abstraction of *B*, which is called B_i', by abstracting from inserted action *d*. The causality context of *d* is also equal to *B*, i.e., *Con*(*B*, *d*) = *B*. Figure 8.16 depicts the corresponding abstract alternative behaviours $BA1_i$', $BA2_i$' and $BA3_i$'. Abstract behaviour B_i' is defined by the disjunction of $BA1_i$', $BA2_i$' and $BA3_i$', and the integration of equivalent alternative causality conditions. In this case, the integration of the alternative causality conditions of action *b'* is based on the property that these conditions are abstractions of the same alternative causality condition of concrete action *b*. Since an alternative behaviour exists in which *b* may be disabled by *d* and the uncertainty introduced by disabling condition ¬*d* is not (completely) represented by another disabling condition of *b*, the occurrence of *b'* after *c'* is uncertain. An analogous reasoning applies to action *a'*. Behaviour B_i' models that the delivery of the entire data unit is uncertain.

Abstraction step II and ii determine the abstraction of B_I' and B_i', by abstracting from inserted actions *d'* and *b'*, respectively. Both steps render the same (more) abstract behav-

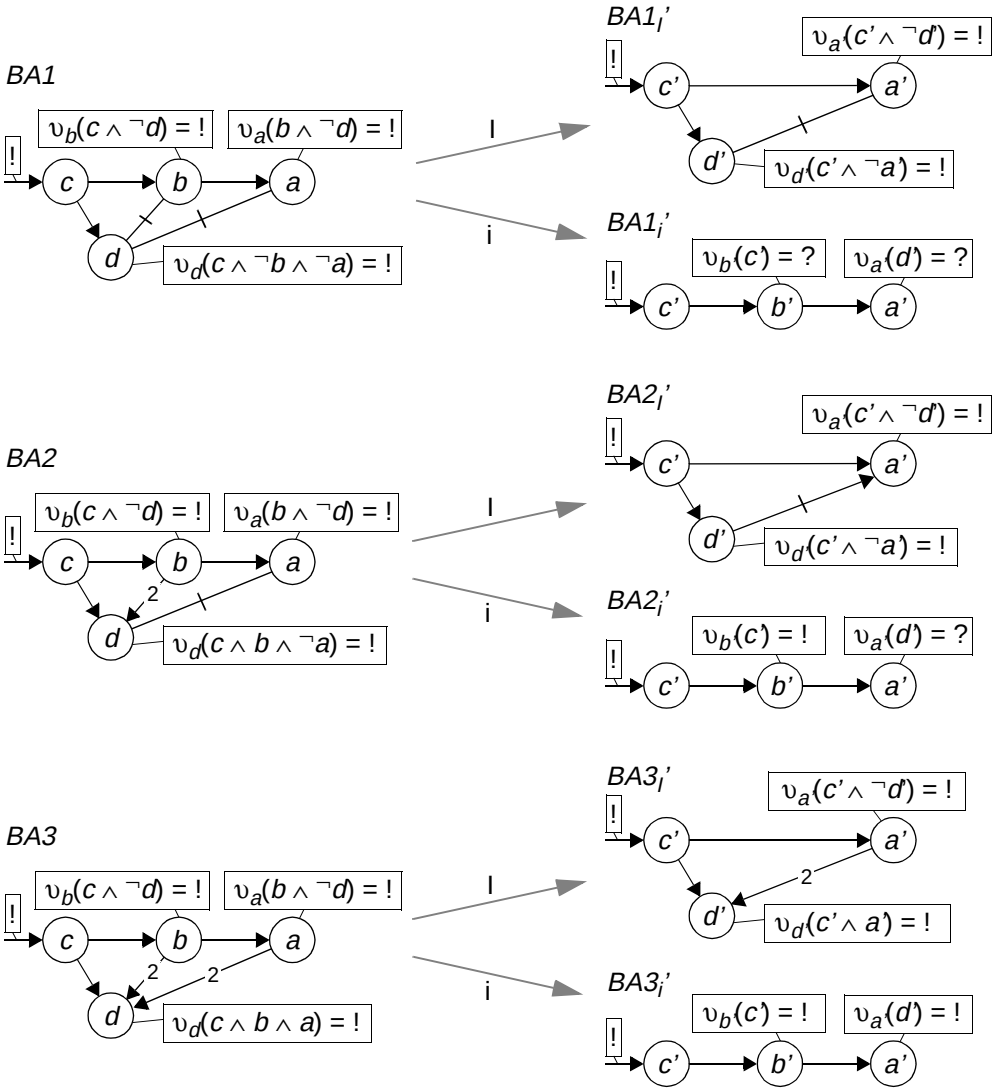


Figure 8.16: Abstractions of alternative behaviours

iour *B”*, which models the uncertain delivery of the entire data unit after the connection has been established.

Abstraction step III (iii) determines the abstraction of *B”*, which is called *B”’*, by abstracting from inserted action *c”*. Behaviour *B”’* models the highest abstraction of concrete behaviour *B*: the uncertain delivery of the entire data unit. The example shows how this uncertainty can be made explicit by applying behaviour refinement (the introduction of inserted actions and their causality relations).

Example 2: choice

Figure 8.17 depicts the step-wise abstraction from inserted actions *z1* and *z2* in concrete behaviour *B*, which results in abstract behaviour *B”*.

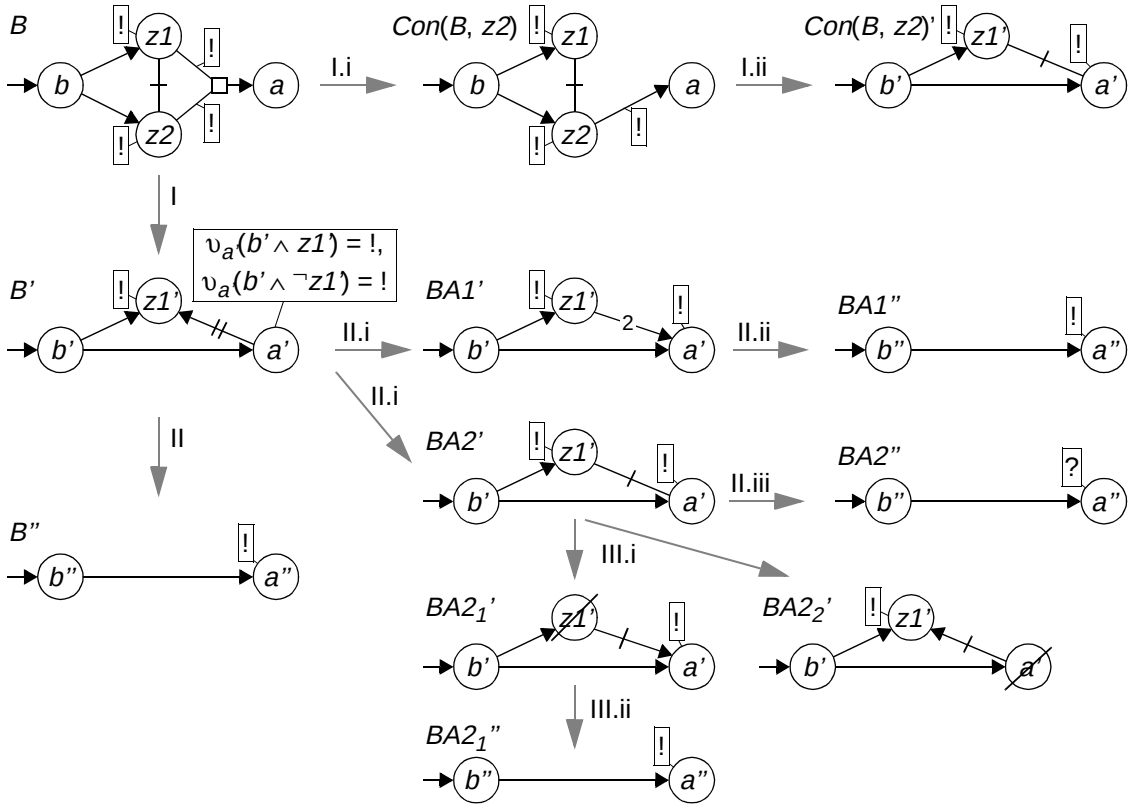


Figure 8.17: Choice

Step I determines behaviour *B'*, which abstracts from inserted action *z2* in *B*. Step I.i determines the causality context of *z2*, i.e., *Con*(*B*, *z2*). Subsequently, step I.ii determines the causality context *Con*(*B*, *z2*)', which abstracts from *z2*. Finally, *B'* is obtained through the replacement of *Con*(*B*, *z2*) by *Con*(*B*, *z2*)' in concrete behaviour *B*.

Step II determines behaviour *B"*, which abstracts from inserted action *z1'* in *B'*. The causality context of *z1'* is equal to *B'*, i.e., *Con*(*B'*, *z1'*) = *B'*. Step II.i determines the alternative behaviours *BA1'* and *BA2'* in *B'*. Steps II.ii and II.iii determine the alternative behaviours *BA1"* and *BA2"*, which abstract from inserted action *z1'*, respectively. Behaviour *B"* is obtained by the disjunction of *BA1"* and *BA2"*, resulting in:

$B" = \{\surd \rightarrow b", b"[\upsilon_{a'}(b") = !] \vee b"[\upsilon_{a'}(b") = ?] \rightarrow a"\}$.

This behaviour can be simplified by combining $b"[\upsilon_{a'}(b") = !]$ and $b"[\upsilon_{a'}(b") = ?]$, resulting in:

$B" = \{\surd \rightarrow b", b"[\upsilon_{a'}(b") = !] \rightarrow a"\}$.

Even though *a" may* occur in *BA2"*, action *a" must* occur in *B"*. The possible disabling of *a'* in *BA2'* is disallowed by *BA1'*, since *a' must* occur after *z1'* has occurred in *BA1'*. Therefore, *BA2"* is considered impossible. This should be anticipated when developing integration rules.

Alternatively, the same result can be derived by decomposing *BA2'* into alternative behaviours $BA2_1$' and $BA2_2$' (see step III.i in Figure 8.17), where $BA2_2$' is considered impossible because *BA1'* forces *a'* to occur after *z1'* has occurred. This implies that *B"* is obtained by

the disjunction of *BA1"* and $BA2_1$*"* (see step III.ii), resulting in:

$B" = \{\surd \rightarrow b", b"[\upsilon_{a'}(b") = !] \vee b"[\upsilon_{a'}(b") = !] \rightarrow a"\}$.

This behaviour can be simplified to:

$B" = \{\surd \rightarrow b", b"[\upsilon_{a'}(b") = !] \rightarrow a"\}$.

In order to support this alternative abstraction step, the method in Section 8.2.3 could be extended with a step in which impossible alternative behaviours are removed.

8.2.9 Abstraction of simple integral probability associations

This section extends the method of Section 8.2.6 to support the abstraction from an inserted action z in some alternative behaviour BA_π that uses the simple integral probability attribute instead of the uncertainty attribute. For this purpose, the abstraction rules in steps 3, 4 and 5 have to be replaced by rules defining the abstraction of some simple probability association $\langle a, \gamma_a, \pi_a(\gamma_a)\rangle$ in BA_π, which involves enabling condition z, synchronization condition $^{=}z$, and disabling condition $\neg z$, respectively.

In the definition of alternative behaviours, the value of the probability association of a disabled action in an asymmetric exclusion relation corresponds to zero. For example, the causality relation of action d in alternative behaviour *BA4* of Figure 8.6 is represented as:

$e \wedge \neg c \rightarrow d\ [\pi_d(e \wedge \neg c) = 0]$.

Simple probability associations involving enabling condition *z*

Consider the example in Figure 8.11, while assuming $\upsilon_z(b)$ and $\upsilon_a(z \wedge b)$ are replaced by simple probability associations $\pi_z(b)$ and $\pi_a(z \wedge b)$, respectively. In this case, $\pi_{a'}(b')$ is defined by the product of the amount of uncertainty introduced by $\pi_z(b)$ and $\pi_a(z \wedge b)$, such that:

$\pi_{a'}(b') = \pi_z(b) \times \pi_a(z \wedge b)$.

The general case of a simple probability association involving enabling condition z is represented for some action a with alternative causality condition γ_a by the following causality relations:

$z \wedge \gamma \rightarrow a\ [\pi_a(z \wedge \gamma)]$, # with $\gamma_a^+ = z \wedge \gamma$

$\gamma_z \rightarrow z\ [\pi_z(\gamma_z)]$.

The abstraction of the causality relation of action a is denoted as: $\gamma' \rightarrow a'[\pi_{a'}(\gamma')]$, where γ' is derived from γ by appending each action name with a prime. In case enabling condition z is not an element of (one of) the minimal definition(s) of γ_a, probability association $\pi_{a'}(\gamma')$ is equal to $\pi_a(z \wedge \gamma)$. Otherwise, probability association $\pi_{a'}(\gamma')$ is defined by the following rule:

$\pi_{a'}(\gamma') = \pi_a(z \wedge \gamma)$, if γ_z contains a synchronization condition, i.e., some b exists such that $^{=}b \in C(\gamma_z^+)$, $^{=}z \in C(\gamma_b^+)$ and $b \in C(\gamma_a^+)$;

$= \pi_z(\gamma_z) \times \pi_a(z \wedge \gamma)$, otherwise.

This rule is analogous to the corresponding rule for uncertainty associations involving enabling condition z presented in Section 8.2.6.

For example, consider Figure 8.12, assuming that uncertainty associations are replaced by simple probability associations. In this case, $\pi_a(c' \wedge \neg b' \wedge d') = \pi_a(z)$, since alternative condition γ_z does not increase the uncertainty w.r.t. alternative condition γ_a, and the uncertainty introduced by z is represented by $\pi_d(c' \wedge \neg b')$.

Simple probability associations involving synchronization condition $^{=}z$

Consider again the example in Figure 8.12, assuming simple probability associations. The amount of uncertainty introduced by synchronization conditions $^{=}z$ and $^{=}d$ is defined by $\pi_d(c \wedge \neg b \wedge {}^{=}z)$, and is the same for both actions z and d, such that: $\pi_d(c \wedge \neg b \wedge {}^{=}z) = \pi_z(c \wedge \neg b \wedge {}^{=}d)$. Therefore, when abstracting from z, the abstraction of $\pi_d(c \wedge \neg b \wedge {}^{=}z)$ is defined as:

$\pi_d(c' \wedge \neg b') = \pi_d(c \wedge \neg b \wedge {}^{=}z)$.

The general case of a simple probability association involving synchronization condition $^{=}z$ is represented for some action a with alternative causality condition γ_a by the following causality relations:

$^{=}z \wedge \gamma \rightarrow a\ [\pi_a({}^{=}z \wedge \gamma)]$, # with $\gamma_a^+ = {}^{=}z \wedge \gamma$
$^{=}a \wedge \gamma \rightarrow z\ [\pi_z({}^{=}a \wedge \gamma)]$ # with $\gamma_z^+ = {}^{=}a \wedge \gamma$.

The abstraction of the causality relation of action a is denoted as: $\gamma' \rightarrow a'\ [\pi_a(\gamma')]$. Simple probability association $\pi_a(\gamma')$ is defined by the following rule:

$\pi_a(\gamma') = \pi_a({}^{=}z \wedge \gamma) = \pi_z({}^{=}a \wedge \gamma)$.

This rule is analogous to the corresponding rule for uncertainty associations involving synchronization condition $^{=}z$ presented in Section 8.2.6.

Simple probability associations involving disabling condition $\neg z$

Consider the example in Figure 8.13, assuming uncertainty associations are replaced by simple probability associations. The abstraction of $\pi_c(d \wedge \neg z)$ is defined as follows:

$\pi_c(d' \wedge \neg b') = \pi_c(d \wedge \neg z)$, if $\pi_b(z) = 1$, such that b occurs after z in every execution in which z occurs;
$< \pi_c(d \wedge \neg z)$, otherwise.

In case $\pi_b(z) < 1$, we only know that the value of $\pi_c(d' \wedge \neg b')$ is smaller than the value of $\pi_c(d \wedge \neg z)$, since the precise amount of uncertainty caused by the possible disabling of c by z, i.e., the probability that z disables c, can not be determined.

Two general cases of a simple probability association involving disabling condition $\neg z$ are distinguished for some action a with alternative causality condition γ_a:

1. an (a)symmetric exclusion relation is defined between z and a:
   $\neg z \wedge \gamma 1 \rightarrow a\ [\pi_a(\neg z \wedge \gamma 1)]$, # with $\gamma_a^+ = \neg z \wedge \gamma 1$
   $\neg a \wedge \gamma 2 \rightarrow z\ [\pi_z(\neg a \wedge \gamma 2)]$ # with $\gamma_z^+ = \neg a \wedge \gamma 2$;
2. a two-sided enabling relation is defined between z and a:
   $\neg z \wedge \gamma 1 \rightarrow a\ [\pi_a(\neg z \wedge \gamma 1)]$, # with $\gamma_a^+ = \neg z \wedge \gamma 1$
   $a \wedge \gamma 2 \rightarrow z\ [\pi_z(\neg a \wedge \gamma 2)]$ # with $\gamma_z^+ = a \wedge \gamma 2$.

The abstraction of the causality relation of action a is denoted as: $\gamma 1' \rightarrow a'[\pi_a(\gamma 1')]$. In case disabling condition $\neg z$ is not an element of (one of) the minimal definition(s) of γ_a, probability association $\pi_a(\gamma')$ is equal to $\pi_a(\neg z \wedge \gamma)$. Otherwise, probability association $\pi_a(\gamma 1')$ is defined as:

$\pi_a(\gamma') = \pi_a(\neg z \wedge \gamma)$, if (i) action z depends on action a, i.e., $a \in C(\gamma_z^+)$; or
(ii) action z is impossible, i.e., $\pi_z(\gamma_z) = 0$; or
(iii) a third action b exists, such that:
$z \in C(\gamma_b^+)$, $\neg b \in C(\gamma_a^+)$, and b must occur after z; or
$^{=}z \in C(\gamma_b^+)$, $^{=}b \in C(\gamma_z^+)$ and $\neg b \in C(\gamma_a^+)$; or
(iv) $\pi_a(\neg z \wedge \gamma) = 0$;

$< \pi_a(\neg z \wedge \gamma)$, otherwise.

This rule is analogous to the corresponding rule for uncertainty associations involving synchronization condition $\neg z$ presented in Section 8.2.6.

For example, consider Figure 8.14, while assuming uncertainty associations are replaced by simple probability associations. The abstraction of $\pi_a(c \wedge \neg z)$ in the modified example of Figure 8.14(i), i.e., $\pi_a(c')$, is defined as: $\pi_a(c') < \pi_a(c \wedge \neg z)$, since z may disable a, but the precise probability with which a is disabled is undefined.

The abstraction of $\pi_c(d \wedge \neg z)$ in the modified example of Figure 8.14(ii) is defined as:

$\pi_c(d' \wedge \neg b') = \pi_c(d \wedge \neg z)$, if $\pi_e(\surd) = 1$ and $\pi_b(z \wedge e) = 1$;
$< \pi_c(d \wedge \neg z)$, otherwise.

Example: unreliable connection (2)

Figure 8.18 depicts the abstraction of concrete behaviour B into abstract behaviour B''' via two alternative paths. This example is similar to the example in Figure 8.15, except that the uncertainty attribute is replaced by the simple probability attribute, and uncertainty values are replaced by specific probability values.

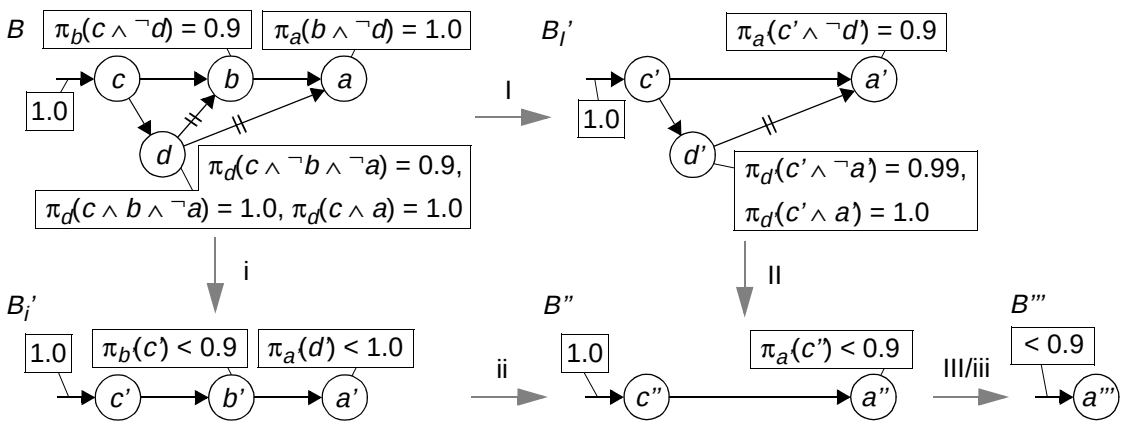


Figure 8.18: Unreliable connection (2)

Abstraction step I can be performed similarly to step I in Figure 8.15, using the abstraction rules for simple probability associations presented above. For brevity, we do not redraw

Figure 8.16 for this case, but instead present the definition of abstract behaviour B_I' as the disjunction of $BA1_I'$, $BA2_I'$ and $BA3_I'$:

$B_I' = \{ \ \surd\,[\pi = 1.0] \vee \surd\,[\pi = 1.0] \vee \surd\,[\pi = 1.0] \rightarrow c',$
$(c' \wedge \neg d')\,[\pi = 0.9] \vee (c' \wedge \neg d')\,[\pi = 0.9] \vee (c' \wedge \neg d')\,[\pi = 0.9] \rightarrow a',$
$(c' \wedge \neg a')\,[\pi = 0.9] \vee (c' \wedge \neg a')\,[\pi = 0.9] \vee (c' \wedge a')\,[\pi = 1.0] \rightarrow d'\},$

which can be simplified by integrating equivalent alternative conditions to:

$B_I' = \{ \ \surd\,[\pi = 1.0] \rightarrow c',$
$(c' \wedge \neg d')\,[\pi = 0.9] \rightarrow a',$
$(c' \wedge \neg a')\,[\pi = 0.99] \vee (c' \wedge a')\,[\pi = 1.0] \rightarrow d'\}.$

The integration of the alternative causality conditions of action a' is straightforward, since in each alternative behaviour abstract condition $\gamma_{a'} = c' \wedge \neg d'$ is the abstraction of the same concrete condition $\gamma_a = b \wedge \neg d$, and the uncertainty association of $\gamma_{a'}$ must compensate for the same amount of uncertainty introduced by inserted action b.

The integration of $(c' \wedge \neg a')\,[\pi = 0.9] \vee (c' \wedge \neg a')\,[\pi = 0.9]$ into $(c' \wedge \neg a')\,[\pi = 0.99]$ in the causality relation of action d' is less straightforward, since abstract condition $\gamma_{d'} = c' \wedge \neg a'$ is the abstraction of concrete condition $\gamma 1_d = c \wedge \neg b \wedge \neg a$ in *BA1* and the abstraction of concrete condition $\gamma 2_d = c \wedge b \wedge \neg a$ in *BA2* (see Figure 8.16), where $\gamma 1_d$ and $\gamma 2_d$ are distinct (non-equivalent) alternative conditions of concrete action d. The value of $\pi_{d'}(c' \wedge \neg a')$ is, amongst others, determined by the relationship between alternative conditions $\gamma 1_d$ and $\gamma 2_d$. This is comparable to the way in which this relationship determines the value of the probability that d does not occur when it is assumed that d is allowed to occur due to $\gamma 1_d$ and $\gamma 2_d$, as discussed in Section 7.2.3. In this specific case, however, the value of $\pi_{d'}(c' \wedge \neg a')$ can be determined. Since $\pi_d(c \wedge b \wedge \neg a) = 1.0$, we can deduce that the probability that action d' must occur when condition $c' \wedge \neg a'$ is satisfied is equal to the probability that action b or d must occur after action c has occurred. Consequently, the following holds:

$\pi_{d'}(c' \wedge \neg a') = \pi_b(c \wedge \neg d) + \pi_d(c \wedge \neg b \wedge \neg a) - \pi_b(c \wedge \neg d) \times \pi_d(c \wedge \neg b \wedge \neg a) = 0.99.$

However, in case $\pi_d(c \wedge b \wedge \neg a) = p$, with $p < 1$, only an upper and lower boundary on the possible values of $\pi_{d'}(c' \wedge \neg a')$ can be established.

Similarly to the above, the following definition of abstract behaviour B_i' is obtained via abstraction step i:

$B_i' = \{ \ \surd\,[\pi = 1.0] \vee \surd\,[\pi = 1.0] \vee \surd\,[\pi = 1.0] \rightarrow c',$
$c'[\pi < 0.9] \vee c'[\pi = 0.9] \vee c'[\pi = 0.9] \rightarrow b',$
$b'[\pi < 1.0] \vee b'[\pi < 1.0] \vee b'[\pi = 1.0] \rightarrow a'\},$

which can be simplified by integrating equivalent alternative conditions to:

$B_i' = \{ \ \surd\,[\pi = 1.0] \rightarrow c',\ c'[\pi < 0.9] \rightarrow b',\ b'[\pi < 1.0] \rightarrow a'\}.$

The value of $\pi_{b'}(c')$ is determined by the formula

$\pi_{b'}(c') = p_1 \times \pi_{b'}(\gamma 1_{c'}) + p_2 \times \pi_{b'}(\gamma 2_{c'}) + p_2 \times \pi_{b'}(\gamma 3_{c'})$, with $p_1 + p_2 + p_3 = 1$,

where $\gamma 1_{c'}$, $\gamma 2_{c'}$ and $\gamma 3_{c'}$ represent the alternative causality conditions of c' in $BA1_i'$, $BA2_i'$ and $BA3_i'$, and p_1, p_2 and p_3 represent the probability of $BA1_i'$, $BA2_i'$ and $BA3_i'$, respectively. Since p_1, p_2 and p_3 can not be determined, but are assumed to be larger than zero, we can only derive that $\pi_{b'}(c')$ must be smaller than 0.9. $\pi_{a'}(b')$ is determined analogously.

The analysis of abstraction steps II, ii and III/iii is left to the reader. In the above, we assumed the disjunction operator $\square_{\iota\tau\lambda\pi}$, which can be obtained from Definition 8.1 by replacing uncertainty attribute υ by simple probability attribute π.

8.2.10 Abstraction of extended integral probability associations

This section extends the method of Section 8.2.6 in order to support the abstraction from inserted action *z* in some alternative behaviour $BA_{\pi*}$, which uses the extended integral probability attribute instead of the uncertainty attribute. Analogously to the previous section, rules are presented to determine the abstraction of some extended probability association $\langle a, \gamma_a, \pi^*_a(\gamma_a)\rangle$ in $BA_{\pi*}$, which involves enabling condition *z*, synchronization condition $^{=}z$, and disabling condition $\neg z$, respectively.

The development of these rules is easier than for the simple probability attribute, because $\langle a, \gamma_a, \pi^*_a(\gamma_a)\rangle$ defines the probability that *a* occurs *due to* γ_a in $BA_{\pi*}$ when *a* is enabled by γ_a. Without further explanation, the abstraction rules are defined as follows:

1. for each $\langle a', \gamma_{a'}, \pi^*_{a'}(\gamma_{a'})\rangle$ in $BA_{\pi*}'$, which involves enabling condition *z*, such that $\gamma_{a'}^{+} = \gamma \wedge z$, replace $\pi^*_{a'}(\gamma_{a'})$ by $\pi^*_a(\gamma)$, where the value of $\pi^*_a(\gamma)$ is defined by the following rule:

   $\pi^*_a(\gamma) = \pi^*_a(\gamma_a)$, if (i) γ_z contains a synchronization condition; or
   (ii) enabling condition *z* is not an element of (one of) the minimal definition(s) of γ_a;

   $= \pi^*_a(\gamma_a) \times \pi^*_z(\gamma_z)$, otherwise.

2. for each $\langle a', \gamma_{a'}, \pi^*_{a'}(\gamma_{a'})\rangle$ in $BA_{\pi*}'$, which involves synchronization condition $^{=}z$, such that $\gamma_{a'}^{+} = \gamma \wedge {}^{=}z$, replace $\pi^*_{a'}(\gamma_{a'})$ by $\pi^*_a(\gamma)$, where $\pi^*_a(\gamma)$ is defined by the following rule:

   $\pi^*_a(\gamma) = \pi^*_a(\gamma_a)$;

3. for each $\langle a', \gamma_{a'}, \pi^*_{a'}(\gamma_{a'})\rangle$ in $BA_{\pi*}'$, which involves disabling condition $\neg z$, such that $\gamma_{a'}^{+} = \gamma \wedge \neg z$, replace $\pi^*_{a'}(\gamma_{a'})$ by $\pi^*_a(\gamma)$, where $\pi^*_a(\gamma)$ is defined by the following rule:

   $\pi^*_a(\gamma) = \pi^*_a(\gamma_a)$.

The development of abstraction rules for the combined specification of the simple and extended probability attribute falls outside the scope of this thesis.

Example: choice (2)

Figure 8.19 depicts the step-wise abstraction from inserted actions *z1* and *z2* in concrete behaviour *B*, which renders abstract behaviour *B"*. This example is similar to the example in Figure 8.17 in Section 8.2.8, except that the uncertainty attribute is replaced by the extended probability attribute, and uncertainty values are replaced by specific probability values.

Abstraction steps I and II can be performed similarly to the corresponding steps discussed in Section 8.2.8, using the abstraction rules for extended probability associations presented above. For brevity, only some intermediate results are shown in Figure 8.19. Alternative

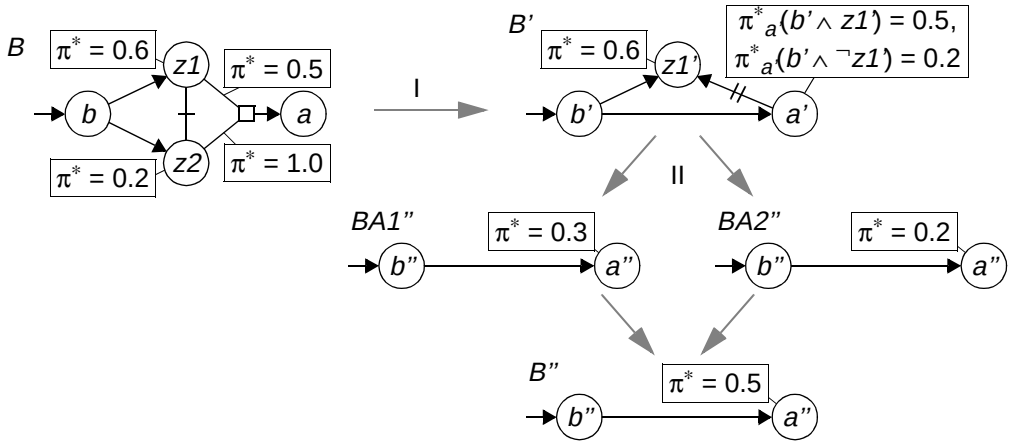


Figure 8.19: Choice (2)

condition $\gamma_{a''} = b''$ of abstract action *a"* in *B"* is the abstraction of alternative conditions $\gamma 1_a = z1$ and $\gamma 2_a = z2$ of concrete action *a* in *B*. These concrete conditions are exclusive conditions due to the choice between *z1* and *z2*, such that *a* is either allowed to occur due $\gamma 1_a$ or is allowed to occur due to $\gamma 2_a$, but not both. Therefore, the value of $\pi^*_{a''}(b'')$ in *B"* is defined as the sum of the values of $\pi^*_{a''}(b'')$ in *BA1"* and $\pi^*_{a''}(b'')$ in *BA2"*, where enabling conditions *b"* in *BA1"* and *b"* in *BA2"* are the abstractions of $\gamma 1_a$ and $\gamma 2_a$, respectively.

8.3 Abstraction from final actions

The abstraction method of Section 8.1.3 divides the abstraction of an activity in some concrete behaviour into two steps: (i) the abstraction from its inserted actions as explained in Section 8.2, and (ii) the replacement of its final actions by an abstract action that models the completion of the entire activity. This section presents a method to perform second step (ii).

8.3.1 Method

The following steps are distinguished when replacing the final actions of some activity *A* by abstract action *a'*:

1. determine the causality relation of abstract action *a'*. This consists of two sub-steps:
   (i) determine the causality condition of abstract action *a'* by integrating the causality conditions of the corresponding final actions;
   (ii) determine the abstraction of the attribute values of the final actions in terms of the possible values or constraints of the attributes of abstract action *a'*;
2. determine the causality relations of the abstract actions outside *A*, denoted by *b'*, which depend on abstract action *a'*. This consists of two sub-steps:
   (i) determine the *completion condition* of activity *A*, in terms of the occurrences of its final actions that correspond to the occurrence of abstract action *a'*. Replace this completion condition by a corresponding condition in terms of abstract action *a'* in the causality relations of abstract actions *b'*;

(ii) replace references of abstract actions *b'* to the information, time and location attributes of the final actions of *A* by references to the corresponding attributes of abstract action *a'* as defined in step 1(ii).

Rules to perform the above steps are presented in the next sections for the following generic cases of activity ending, which are denoted as (*sf*), (*cf*) and (*df*), respectively:

(sf) activity *A* consists of a single final action *a*;

(cf) activity *A* consists of a conjunction of final actions a_i, with $1 \le i \le n$;

(df) activity *A* consists of a disjunction of final actions a_i, with $1 \le i \le n$.

8.3.2 Causality condition of abstract action *a'*

The following rules define the causality condition of abstract action *a'* by integrating the causality conditions of the corresponding final actions:

(sf) $\Gamma(a') = \Gamma(a)^{[']} = \{\gamma^{[']} \mid \gamma \in \Gamma(a)\}$, which defines that the causality condition of *a'* is derived from the causality condition of final action *a*, by relabelling each action name with a prime. This rule represents that abstract action *a'* is allowed to occur when final action *a* is allowed to occur. The relabelling operation is represented by postfix-operator $^{[']}$;

(cf) $\Gamma(a') = \wedge\{\Gamma(a_i) \mid 1 \le i \le n\}^{[']} = \vee\{\gamma_1^{[']} \wedge .. \wedge \gamma_n^{[']} \mid \gamma_1 \in \Gamma(a_1), .., \gamma_n \in \Gamma(a_n)\}$, which defines that the causality condition of *a'* is derived from the causality conditions of all a_i's, as the disjunction of all possible conjunctions of *n* alternative causality conditions γ_i, involving one γ_i of each final action a_i, while relabelling each action name with a prime. This rule represents that abstract action *a'* is allowed to occur when all final actions a_i are allowed to occur;

In case of synchronized final actions, the corresponding synchronization conditions $^{=}a_i$ should be removed from any $\gamma' \in \Gamma(a')$.

(df) $\Gamma(a') = \vee\{\Gamma(a_i) \mid 1 \le i \le n\}^{[']} = \vee\{\gamma^{[']} \mid \gamma \in \Gamma(a_i), 1 \le i \le n\}$, which defines that the causality condition of *a'* is derived from the causality conditions of all a_i's, as the disjunction of the alternative causality conditions of all final actions a_i, while relabelling each action name with a prime. This rule represents that abstract action *a'* is allowed to occur when minimal one final action a_i is allowed to occur.

In case of exclusive final actions, the corresponding disabling conditions $\neg a_i$ should be removed from any $\gamma' \in \Gamma(a')$.

In case activity *A* consists of a combination of conjunctions and disjunctions of final actions, the causality condition of abstract action *a'* is defined by a proper combination of the rules above.

8.3.3 Attribute values of abstract action *a'*

Rules to determine the abstraction of the attribute values of the final actions of activity *A* are discussed below.

Single final action

In case activity A consists of a *single final action* a, activity A makes its attribute values available through the occurrence of a. Therefore, the attribute values of abstract action a' are the same as the attribute values of a. This implies that the attribute constraints associated with each $\gamma' \in \Gamma(a')$ are derived directly from the attribute constraints associated with the corresponding $\gamma \in \Gamma(a)$ by relabelling action names with a prime.

Conjunction of final actions

In case activity A consists of a *conjunction of final actions* a_i, activity A makes its attribute values available through the occurrence of all final actions. The following rules define how the information, time and location values of abstract action a' are determined by the corresponding attribute values of the final actions:

- $\iota(a') = \langle \iota(a_1), .., \iota(a_n) \rangle$, which defines that the information value of a' consists of the information values of all final actions of A;
- $\tau(a') = \max(\{\tau(a_i) \mid 1 \leq i \leq n\}$, which defines that the time moment of a' corresponds to the time moment of the latest final action of A;
- $\lambda(a') = \{\lambda(a_i) \mid 1 \leq i \leq n\}$, which defines that the location of a' corresponds to the collection of locations of all final actions of A;

The information, time and location attribute constraints associated with each $\gamma' \in \Gamma(a')$ are derived from the corresponding attribute constraints associated with conditions $\gamma_i \in \Gamma(a_i)$, with $\gamma' = \gamma_1^{[']} \wedge .. \wedge \gamma_n^{[']}$, by integrating these constraints according to the above rules.

The uncertainty and probability attribute constraint associated with each $\gamma' \in \Gamma(a')$ is derived from the corresponding attribute constraints associated with conditions $\gamma_i \in \Gamma(a_i)$, with $\gamma' = \gamma_1^{[']} \wedge .. \wedge \gamma_n^{[']}$, according to the following rule:

- $\upsilon(a', \gamma') = \times(\{\upsilon(a_i, \gamma_i) \mid 1 \leq i \leq n\}$, where $\varphi_1 \times \varphi_2 = \mathit{must} \Leftrightarrow \varphi_1 = \varphi_2 = \mathit{must}$, which defines that the uncertainty that a' occurs when γ' is satisfied corresponds to the uncertainty that all final actions occur when all conditions γ_i are satisfied;
- $\pi(a', \gamma') = \times(\{\pi(a_i, \gamma_i) \mid 1 \leq i \leq n\}$, which defines that the probability that a' occurs when γ' is satisfied corresponds to the probability that all final actions occur when all conditions γ_i are satisfied;
- $\pi^*(a', \gamma') = \times(\{\pi^*(a_i, \gamma_i) \mid 1 \leq i \leq n\}$, which defines that the probability that a' occurs due to γ' when γ' enables a' corresponds to the probability that each final action a_i occurs due to condition γ_i when γ_i enables a_i.

In case activity A contains a group of synchronized final actions, the above equations should be adjusted to consider the simple or extended probability association of only one action from this group, due to the reciprocal aspect of the synchronization condition.

Disjunction of final actions

In case activity A consists of a *disjunction of final actions* a_i, activity A makes its attribute values available through the occurrence of one of these final actions. Therefore, the attribute values of abstract action a' are the same as the attribute values of this final action. This implies that the attribute constraints associated with each $\gamma' \in \Gamma(a')$ are derived directly from the attribute constraints associated with the corresponding $\gamma \in \Gamma(a_i)$, while relabelling action names with a prime.

In case distinct final actions a_i and a_j have alternative causality conditions γ_i and γ_j, respectively, which are equivalent in the abstract behaviour, i.e., $\gamma_i \equiv_{B'} \gamma_j$, and have the same information, time and location attribute constraints associated with γ_i and γ_j, the disjunction of both conditions in $\Gamma(a')$, i.e.,

$$\gamma_i^{[']}[I\text{-}Att(a', \gamma_i^{[']})^{[']}] \vee \gamma_j^{[']}[I\text{-}Att(a', \gamma_j^{[']})^{[']}]$$

may be simplified by defining only one of both alternative conditions, which is denoted as γ' hereafter. In this case, the uncertainty and probability constraints associated with γ' are derived from the uncertainty and probability constraints associated with γ_i and γ_j, according to the following rules:

- $\upsilon(a', \gamma') = must$, if $\upsilon(a_i, \gamma_i) = must$ or $\upsilon(a_j, \gamma_j) = must$, which defines that abstract action a' must occur when final action a_i or a_j must occur when γ_i and γ_j are satisfied;
- $\pi(a', \gamma') = \pi(a_i, \gamma_i) + \pi(a_j, \gamma_j) - \pi(a_i, \gamma_i) \times \pi(a_j, \gamma_j)$, which defines that the probability that action a' occurs when γ' is satisfied corresponds to (i) the probability that a_i or a_j, or both, occur when assuming γ_i and γ_j are satisfied, in case a_i and a_j are independent actions, or (ii) the probability that either a_i or a_j occurs when assuming γ_i and γ_j are (subsequently) satisfied, in case a_i and a_j are exclusive actions;
- $\pi^*(a', \gamma') = \pi^*(a_i, \gamma_i) + \pi^*(a_j, \gamma_j)$, which defines that the probability that action a' occurs due to γ' when γ' enables a' corresponds to the probability that either a_i occurs due to γ_i when γ_i enables a_i or a_j occurs due to γ_j when γ_j enables a_j, in case a_i and a_j are exclusive actions. The case in which actions a_i or a_j are independent is not considered in this work for the extended probability attribute.

Completion condition of activity *A*

The completion condition of activity A represents the occurrence (or successful termination) of A. For the generic cases of activity ending identified in Section 8.3.1, this completion condition is defined as:

(sf) a, which defines that the successful termination of activity A corresponds to the occurrence of the single final action a;

(cf) $a_1 \wedge .. \wedge a_n$, which defines that the successful termination of activity A corresponds to the occurrences of all final actions of A;

(df) $a_1 \vee .. \vee a_n$, which defines that the successful termination of activity A corresponds to the occurrence of a single final action of A.

The replacement of the completion condition in the causality condition of some abstract action b' is defined by the following rules:

(sf) in the case of a single final action, the following replacements have to be made:

- a is replaced by a', which represents that abstract action b' depends on the occurrence of a';
- $\neg a$ is replaced by $\neg a'$, which represents that abstract action b' depends on the non-occurrence of a', i.e., a' does not occur before nor simultaneously with b';
- $^{=}a$ is replaced by $^{=}a'$, which represents that abstract action b' depends on the synchronization with a';

(cf) in the case of a conjunction of final actions, the following replacements have to be made:

- $a_1 \wedge .. \wedge a_n$ is replaced by a', which represents that abstract action b' depends on the occurrence of a';
- $\neg a_1 \vee .. \vee \neg a_n$ is replaced by $\neg a'$, which represents that abstract action b' depends on the non-occurrence of a';
- $^{=}a_1 \wedge .. \wedge {}^{=}a_n$ is replaced by $^{=}a'$, which represents that abstract action b' depends on the synchronization with a';

(df) in the case of a disjunction of final actions, the following replacements have to be made:

- $a_1 \vee .. \vee a_n$ is replaced by a', which represents that abstract action b' depends on the occurrence of a';
- $\neg a_1 \wedge .. \wedge \neg a_n$ is replaced by $\neg a'$, which represents that abstract action b' depends on the non-occurrence of a';
- $^{=}a_1 \vee .. \vee {}^{=}a_n$ is replaced by $^{=}a'$, which represents that abstract action b' depends on the synchronization with a'.

In case activity A consists of a combination of conjunctions and disjunctions of final actions, the replacement of the completion condition of this activity is defined by a proper combination of the rules above.

8.3.4 Examples

This section illustrates the method presented in the previous sections by means of three examples.

Example 1: reception of segmented data unit

Figure 8.20 depicts an example of abstraction of a conjunction of final actions. Concrete action s models the sending of a data unit, which is divided into three data segments. Concrete actions $r1$, $r2$ and $r3$ model the reception of these three segments at communication ports 1, 2 and 3, such that each segment either arrives within ΔT time units or is lost. Concrete action a models the reassembly of the original data unit.

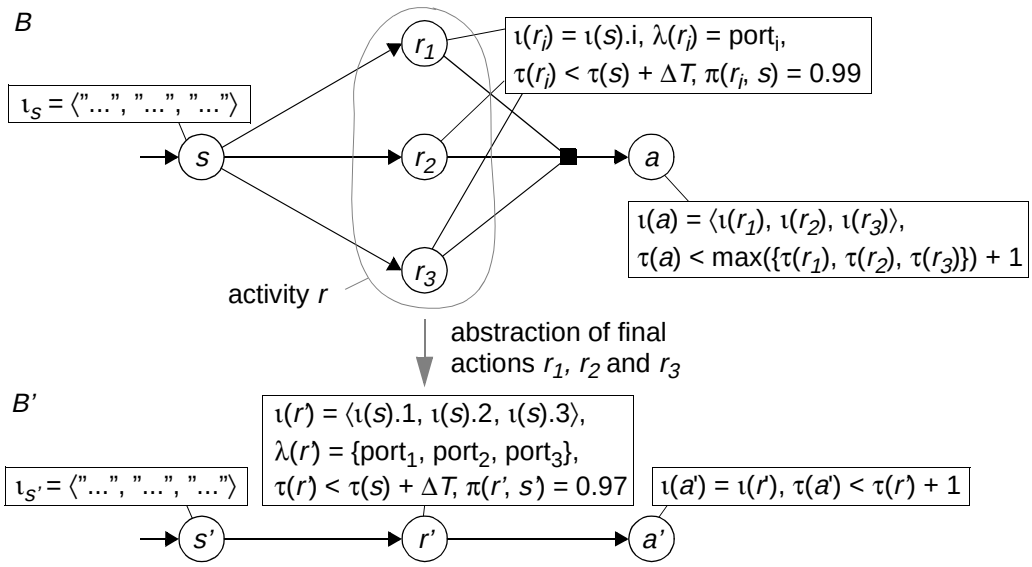


Figure 8.20: Reception of segmented data unit

Concrete actions *r1*, *r2* and *r3* are the final actions of activity *r*, which represents the reception of the entire data unit. This activity is replaced by abstract action *r'* in the abstract behaviour. The causality condition of *r'* corresponds to the conjunction of the conditions of the final actions, i.e., $s \wedge s \wedge s$, which is simplified to *s*. The time moment of *r'* corresponds to the time moment at which the last data segment is received. The information value of *r'* consists of all segments of the original data unit, since activity *r* can only terminate when all segments have been received. Consequently, the probability that *r'* occurs is equal to the probability that all final actions *r1*, *r2* and *r3* occur, i.e.: $\pi_{r'}(s') = \pi_{r1}(s) \times \pi_{r2}(s) \times \pi_{r3}(s)$. The location of *r'* can be considered as a single interface that comprises ports 1 to 3.

The completion condition of activity *r* corresponds to the occurrences of all final actions, i.e., $r_1 \wedge r_2 \wedge r_3$. This condition is replaced by enabling condition *r'* in the causality relation of abstract action *a'*. The time reference of action *a'* to the maximum of $\tau(r_1)$, $\tau(r_2)$ and $\tau(r_3)$ is replaced by $\tau(r')$, and information references of action *a'* to $\iota(r_1)$, $\iota(r_2)$ and $\iota(r_3)$ are replaced by references to the first, second and third element of $\iota(r')$, i.e., $\iota(r').1$, $\iota(r').2$ and $\iota(r').3$. The composition of these references, i.e., $\langle\iota(r').1, \iota(r').2, \iota(r').3\rangle$ is simplified to $\iota(r')$.

Example 2: alternative routing of a data unit

Figure 8.21 depicts an example of abstraction of a disjunction of final actions. Concrete action *s* models the sending of a data unit. Concrete actions *r1*, *r2* and *r3* model the reception of this data unit at communication ports 1, 2 or 3, respectively, via three alternative routes. Each route is characterized by a distinct maximal delay ΔT_i and a distinct cost value $\$_i$. Concrete action *a* models the acknowledgement of the reception of the data unit.

Concrete actions *r1*, *r2* and *r3* are alternative final actions of activity *r*, such that only one of them can occur. This activity is replaced by abstract action *r'* in the abstract behaviour. The causality condition of *r'* corresponds to the disjunction of the conditions of the final actions, i.e., $s \vee s \vee s$, which is simplified to *s*. The information, time and location attribute con-

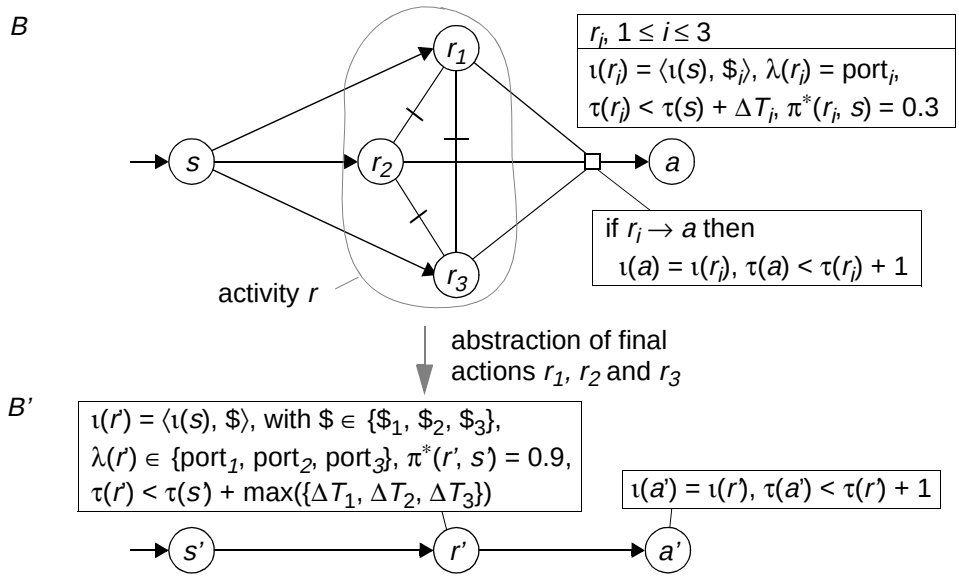


Figure 8.21: Alternative routing of a data unit

straints associated with condition *s'* are defined as the disjunction of the corresponding attribute constraints of alternative condition *s* of *r1*, *r2* and *r3*, such that:

- the information value of *r'* is equal to one of the information values of *r1*, *r2* or *r3*;
- the time moment of *r'* lies within the time intervals in which *r1*, *r2* or *r3* can occur;
- the location value of *r'* comprises the locations of *r1*, *r2* or *r3*.

The probability that *r'* occurs is equal to the probability that either *r1*, *r2* or *r3* occurs. Observe that extended probability association $\pi^*(r', s') = 0.9$ can be replaced by simple probability association $\pi(r', s') = 0.9$.

The completion condition of activity *r* corresponds to the occurrences of one of the final actions, i.e., $r_1 \vee r_2 \vee r_3$. This condition is replaced by enabling condition *r'* in the causality relation of abstract action *a'*. The references of action *a'* to $\iota(r_1)$, $\iota(r_2)$ and $\iota(r_3)$ and to $\tau(r_1)$, $\tau(r_2)$ and $\tau(r_3)$ are replaced by references to $\iota(r')$ and to $\tau(r')$, respectively.

Example 3: connection termination

Figure 8.22 depicts another example of abstraction of a disjunction of final actions. Concrete action *c* models the establishment of a connection. This connection can be terminated in two alternative ways: (i) concrete action *r* models the orderly release of the connection, and (ii) activity *a* models the abrupt abortion of the connection. Activity *a* consists of a disjunction of two final actions a_1 and a_2, which model the occurrence of a protocol error and the occurrence of a QoS failure, respectively. The specific protocol error or QoS failure are modelled by the information attributes of a_1 and a_2. Concrete action *n* notifies the abortion. In case both a_1 and a_2 occur, *n* notifies only one of them.

Activity *a* is terminated when one or both actions a_1 and a_2 occur. This activity is replaced by action *a'* in the abstract behaviour. The causality condition of *a'* corresponds to the dis-

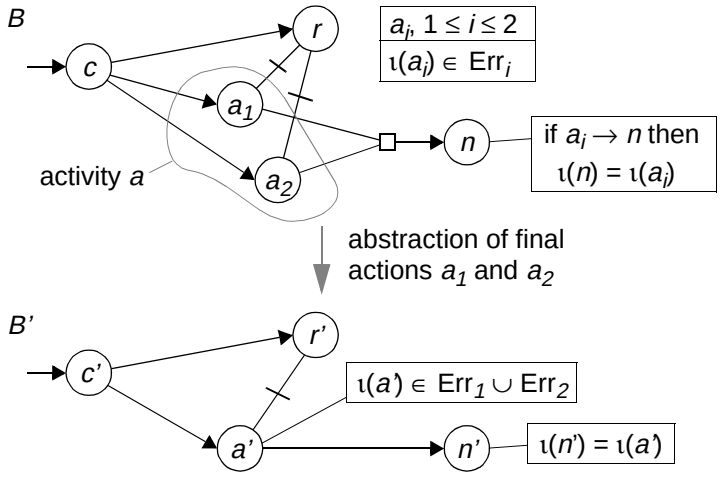


Figure 8.22: Connection termination

junction of the conditions of a_1 and a_2, i.e., $c \vee c$, which is simplified to c. The information attribute constraint associated with a' consists of the disjunction of the corresponding attribute constraints associated with a_1 and a_2, such that the information value of a' indicates either one of the protocol errors or one of the QoS failures indicated by a_1 and a_2, respectively. The uncertainty that a' occurs is equal to the uncertainty that a_1 or a_2 occurs.

The completion condition of activity a corresponds to the occurrences of one of the final actions, i.e., $a_1 \vee a_2$. This condition is replaced by enabling condition a' in the causality relation of abstract action n'. References of action n' to $\iota(a_1)$ and $\iota(a_2)$ are replaced by a reference to $\iota(a')$.

Furthermore, abstract action r' depends on condition $\neg a_1 \wedge \neg a_2$. This condition is replaced by disabling condition $\neg a'$, since the non-occurrence of a' corresponds to the non-occurrences of both a_1 and a_2.

8.3.5 Impossibility of abstraction

From our experience applying the method presented in this chapter, we observed that it is not always possible to abstract from an (entire) activity. For example, if we assume that a concrete action b exists in the example of Figure 8.20, such that b depends on the reception of a single data segment, it would be impossible to replace the condition of this action by a condition in terms of the (non-)occurrence of abstract action r'. In this case, no abstraction of activity r as a single abstract action r' would be possible. Similarly, if we assume that a concrete action exists in the example of Figure 8.21, such that this action depends on only one final action or depends on the disjunction of only two final actions, no abstraction of activity r would be possible.

However, the examples above do not invalidate the method. The impossibility to abstract from an (entire) activity is either caused by an incorrect refinement of an abstract action or is caused by an incorrect identification of the reference actions in the concrete behaviour. The method applies when these two concerns are consistently addressed.

8.4 Use of the execution model

This section presents a method to assess the conformance between an abstract behaviour and a concrete behaviour in case both behaviours are represented in terms of their executions.

8.4.1 Method outline

The assessment of the conformance between an abstract behaviour *B1'* and a concrete behaviour *B* is performed in two steps: (1) the determination of the abstraction of *B*, which is called *B2'*, and (2) the assessment of the correctness relation between *B1'* and *B2'*. Figure 8.23(i) depicts these steps, including the refinement step itself.

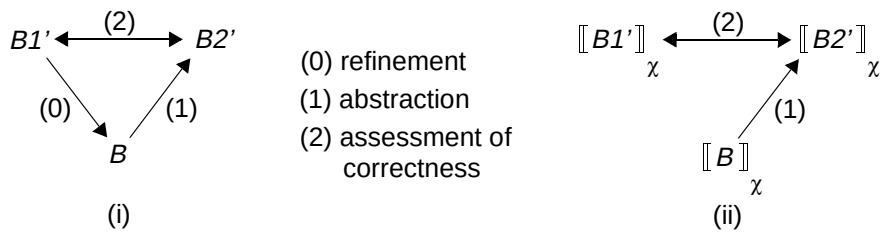


Figure 8.23: Conformance assessment

The methods that have been presented so far in this chapter to support steps (1) and (2) are defined in terms of manipulations on causality-based behaviour definitions. Alternatively, methods can be developed to perform steps (1) and (2) in terms of manipulations of execution-based behaviour definitions such as illustrated in Figure 8.23(ii). Methods that manipulate causality-based or execution-based behaviour definitions are denoted as causality-based and execution-based methods, respectively.

The purpose of execution-based methods that support step (1) in Figure 8.23(ii) is to determine the abstractions of the executions of concrete behaviour *B*. The executions of *B* are called *concrete executions* and their abstractions are called *abstract executions*. The set of abstract executions obtained in this step should be equal to the execution semantics of *B2'*. This requires that the abstraction rules of the causality-based methods presented in the preceding sections are properly translated to abstraction rules of the execution-based methods applied in step (1). The satisfaction of this requirement is informally motivated during the development of these execution-based methods in Sections 8.4.2 and 8.4.3. A formal proof of correctness is beyond the scope of this thesis.

The purpose of an execution-based method that supports step (2) in Figure 8.23(ii) is to compare the abstract executions of *B2'* with the abstract executions of *B1'* w.r.t. a certain correctness relation. In this section, we assume that the executions of *B1'* and *B2'* must be the same, i.e., $[\![B1']\!] = [\![B2']\!]$.

Abstraction of an execution-based behaviour

The following steps define a method to determine the abstraction of a concrete execution-based behaviour:

1. *identify reference actions and inserted actions in the concrete behaviour*, similar to the corresponding step of the causality-based method presented in Section 8.1.3;
2. *abstract from inserted actions*, using the method presented in Section 8.4.2;
3. *replace each group of final actions by an abstract action*, using the method presented in Section 8.4.3.

8.4.2 Abstraction from inserted actions

This section presents a method to abstract from inserted actions in an execution-based behaviour. The causality-based method of Section 8.2 abstracts from the influence of some inserted action *z* on a concrete behaviour by subsequently abstracting from indirect action relations via *z*, indirect information, time and location references via *z*, and the influence of *z* on the uncertainty and probability of other actions. How these abstractions are performed in terms of the execution model is explained below.

Indirect action relations

The method of Section 8.2.4 replaces indirect action relations by direct action relations, using the rules for determining the indirect dependency closures of causality conditions presented in Section 5.3. The determination of the indirect dependency closure of the causality conditions of some behaviour corresponds to the determination of the transitive closures of the ordering relation $<_\chi$ and synchronization relation $=_\chi$ in all executions of this behaviour. This can be proven for each individual rule in Table 5.20.

Consequently, we can abstract from indirect action relations caused by inserted action *z* for execution-based behaviours as follows:

1. determine for each execution the transitive closure of relations $<_\chi$ and $=_\chi$;
2. eliminate the occurrence and non-occurrence of *z* from each execution.

Indirect information, time and location references

References to information, time or location values are modelled in the execution model by enumerating all possible combinations of attribute values that can be established by the involved actions in an execution. This implies that inserted action *z* can be simply removed from all executions, since the possible combinations of attribute values that are established by its context actions in these executions remain unaffected by this removal.

Uncertainty and probability

The probability or uncertainty of the occurrence of an action is modelled implicitly in the execution model by the sum of the probability of the executions in which this action occurs. The relation between the probabilities of directly or indirectly related action occurrences is modelled implicitly by execution probabilities, since the executions of a behaviour represent all possible combinations of action occurrences and their relations. This implies that

when inserted action z is removed from all executions, the relation between the probabilities of its context actions remains unaffected.

In case the abstractions of two or more concrete executions result in the same abstract execution, the sum of the probabilities of these concrete executions should be assigned to the probability of the resulting abstract execution.

Method

The following steps define a method to abstract from inserted action z in a concrete execution-based behaviour $\langle E, \pi_E \rangle$, resulting in behaviour $\langle E', \pi_{E'} \rangle$:

1. determine the abstraction of each execution $\chi \in E$, which is called χ', such that:

   $E' = \{\chi' \mid \chi \in E\}$, where χ' is defined as:

   $\underline{A}_{\chi'} = \underline{A}_{\chi} - \{z\}$;
   $\overline{A}_{\chi'} = \overline{A}_{\chi} - \{z\}$;
   $<_{\chi'} = \{\langle a, b\rangle \mid \langle a, b\rangle \in <_{\chi}^{+}, a \neq z, b \neq z\}$;
   $=_{\chi'} = \{\langle a, b\rangle \mid \langle a, b\rangle \in =_{\chi}^{+}, a \neq z, b \neq z\}$;
   $\iota_{\chi'} = \iota_{\chi} - \{\langle z, \iota_{\chi}(z)\rangle\}$;
   $\tau_{\chi'} = \tau_{\chi} - \{\langle z, \tau_{\chi}(z)\rangle\}$;
   $\lambda_{\chi'} = \lambda_{\chi} - \{\langle z, \lambda_{\chi}(z)\rangle\}$;

   in addition, append each action name with a prime;

2. determine the abstraction of probability function $\pi_E : \wp(E) \rightarrow \wp(\boldsymbol{P})$, such that:

   $\pi_{E'} : \wp(E') \rightarrow \wp(\boldsymbol{P})$, while for each $\chi' \in E'$ the following holds:

   $\pi_{E'}(\{\chi'\}) = \Sigma\{\pi_E(\{\chi_1\}) \mid \chi_1 \in E, \chi_1' = \chi'\}$, where χ_1' is the abstraction of χ_1.

Example: unreliable connection (3)

Figure 8.24 depicts the behaviour of an unreliable connection, called B, which is a variant of the examples in Figures 8.15 and 8.18. Behaviour B uses the simple and extended probability attribute in combination. Figure 8.24 also shows the abstraction of B, called B', which abstracts from inserted action b.

In Figure 8.24 the execution model is used to establish that B' is a correct abstraction of B. This is performed in two steps:

1. determine the execution semantics of B and B', such that:

   $[\![B]\!] = \langle E, \pi_E \rangle$, with $E = \{\chi_1, \chi_2, \chi_3\}$ and
   $\pi_E = \{\pi_E(\{\chi_1\}) = 0.81, \pi_E(\{\chi_2\}) = 0.09, \pi_E(\{\chi_3\}) = 0.1\}$;
   $[\![B']\!] = \langle E', \pi_{E'} \rangle$, with $E' = \{\chi_1', \chi_{23}'\}$ and
   $\pi_{E'} = \{\pi_{E'}(\{\chi_1'\}) = 0.81, \pi_{E'}(\{\chi_{23}'\}) = 0.19\}$;

2. determine the abstraction of E when abstracting from b and compare this abstraction with E'. In this case, execution χ_1' is equal to the abstraction of χ_1 and execution χ_{23}' is equal to the abstractions of χ_2 and χ_3. Consequently, the probability of χ_1' and χ_{23}' are defined as: $\pi_{E'}(\{\chi_1'\}) = \pi_E(\{\chi_1\})$ and $\pi_{E'}(\{\chi_{23}'\}) = \pi_E(\{\chi_2\}) + \pi_E(\{\chi_3\})$.

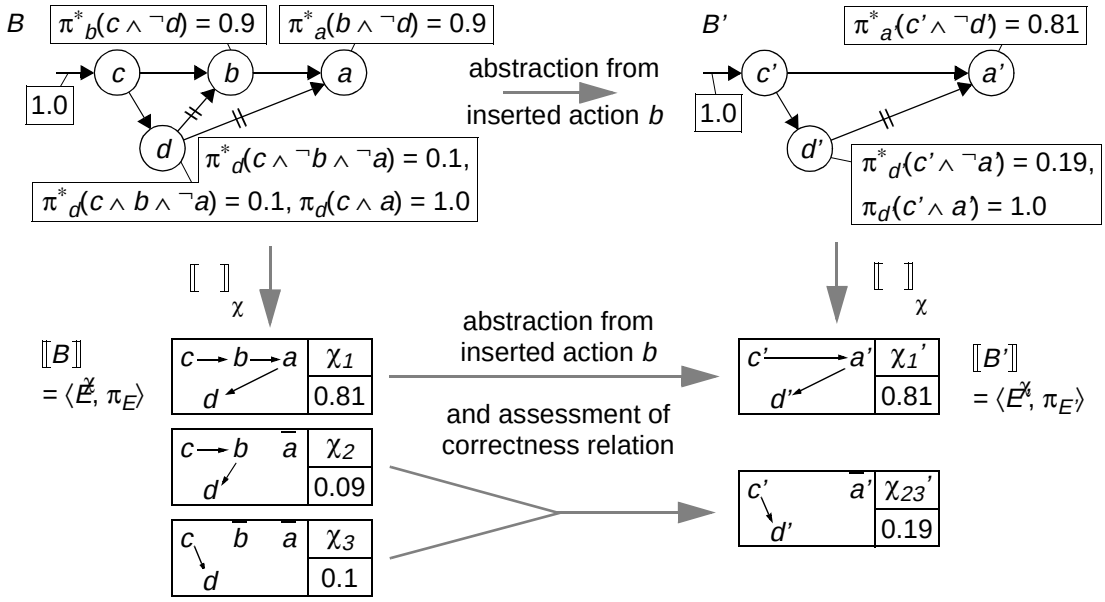


Figure 8.24: Unreliable connection (3)

8.4.3 Abstraction from final actions

This section presents a method to determine the abstraction of the final actions of an activity in an execution-based behaviour. The correspondence of this method with the causality-based method of Section 8.3 is also explained.

Method

The following rules must be followed when replacing the final actions of some activity A by abstract action a' in execution-based behaviour $\langle E, \pi_E \rangle$:

(sf) in case activity A consists of a single final action a, determine the abstraction χ' of each $\chi \in E$ by replacing occurrence a in χ by occurrence a' in χ' or replacing non-occurrence $\overline{a}$ in χ by non-occurrence $\overline{a}'$ in χ'.

This corresponds to relabelling each action name a in χ by a'. This operation is denoted as $\chi' = \chi\ [a'/a]$;

(cf) in case activity A consists of a conjunction of final actions $a_1, .., a_n$, determine the abstraction χ' of each $\chi \in E$ by performing one of the following activities, where set $A_{final} = A_\chi \cap \{a_1, .., a_n\}$ represents the final actions that occur in χ:

- if $|A_{final}| = n$ then replace each occurrence $a_i \in A_{final}$ by occurrence a' and define the attribute values of a' as $\iota(a') = \langle\, \iota(a_1), .., \iota(a_n)\, \rangle$, $\tau(a') = \max(\{\tau(a_1), .., \tau(a_n)\})$, and $\lambda(a') = \{\lambda(a_1), .., \lambda(a_n)\}$, such that:

$$A_{\chi'} = A_\chi\ [a'/a_1, .., a'/a_n];\ \overline{A}_{\chi'} = \overline{A}_\chi;$$
$$<_{\chi'} = <_\chi\ [a'/a_1, .., a'/a_n];\ =_{\chi'} = =_\chi\ [a'/a_1, .., a'/a_n];$$
$$\iota_{\chi'} = (\ \iota_\chi - \{\langle b, \iota_\chi(b)\rangle \mid b \in A_{final}\}\ ) \cup \{\langle a', \iota(a')\rangle\};$$
$$\tau_{\chi'} = (\ \tau_\chi - \{\langle b, \tau_\chi(b)\rangle \mid b \in A_{final}\}\ ) \cup \{\langle a', \tau(a')\rangle\};$$
$$\lambda_{\chi'} = (\ \lambda_\chi - \{\langle b, \lambda_\chi(b)\rangle \mid b \in A_{final}\}\ ) \cup \{\langle a', \lambda(a')\rangle\};$$

- if $|A_{final}| < n$ then replace each occurrence $a_i \in A_{final}$ and each non-occurrence $\overline{a}_j$, with $a_j \in \{a_1, .., a_n\} - A_{final}$, by non-occurrence $\overline{a}'$, such that:

  $A_{\chi'} = A_\chi - A_{final}$; $\overline{A}_{\chi'} = \overline{A}_\chi - (\{a_1, .., a_n\} - A_{final}) \cup \{a'\}$;
  $<_{\chi'} = <_\chi - \{\langle b, c\rangle \mid \langle b, c\rangle \in <_\chi^+, (b \in A_{final} \vee c \in A_{final})\}$;
  $=_{\chi'} = =_\chi - \{\langle b, c\rangle \mid \langle b, c\rangle \in =_\chi^+, (b \in A_{final} \vee c \in A_{final})\}$;
  $\iota_{\chi'} = \iota_\chi - \{\langle b, \iota_\chi(b)\rangle \mid b \in A_{final}\}$, $\tau_{\chi'} = \tau_\chi - \{\langle b, \tau_\chi(b)\rangle \mid b \in A_{final}\}$,
  $\lambda_{\chi'} = \lambda_\chi - \{\langle b, \lambda_\chi(b)\rangle \mid b \in A_{final}\}$;

  If there exists a $\langle b, c\rangle \in (<_\chi^+ \cup =_\chi^+)$ such that $b \in A_{final}$ and $c \notin A_{final}$, then the abstraction of χ is impossible. This is explained later on.

(df) in case activity A consists of a disjunction of final actions $a_1, .., a_n$, determine the abstraction χ' of each $\chi \in E$ by performing one of the following activities, where set $A_{final} = A_\chi \cap \{a_1, .., a_n\}$ represents the final actions that occur in χ:

- if $|A_{final}| = 1$ then replace occurrence a_i by occurrence a' and remove non-occurrences $\overline{a}_{j\ (j \neq i)}$, when assuming $A_{final} = \{a_i\}$, such that:

  $A_{\chi'} = A_\chi[a'/a_i]$; $\overline{A}_{\chi'} = \overline{A}_\chi - (A_{final} - \{a_i\})$;
  $<_{\chi'} = <_\chi[a'/a_i]$; $=_{\chi'} = =_\chi[a'/a_i]$; $\iota_{\chi'} = \iota_\chi[a'/a_i]$; $\tau_{\chi'} = \tau_\chi[a'/a_i]$; $\lambda_{\chi'} = \lambda_\chi[a'/a_i]$;

- if $|A_{final}| = 0$ then replace non-occurrences $\overline{a}_{j\ (1 \leq j \leq n)}$ by non-occurrence $\overline{a}'$. This corresponds to the relabelling operation: $\chi' = \chi\ [a'/a_1, .., a'/a_n]$;

- if $|A_{final}| > 1$ then create a new instance of χ for each $a_i \in A_{final}$ in which occurrence a_i is replaced by occurrence a', and occurrences $a_j \in A_{final} - \{a_i\}$ and non-occurrences $a_k \in \{a_1, .., a_n\} - A_{final}$ are removed, such that:

  $A_{\chi'} = (A_\chi - (A_{final} - \{a_i\}))[a'/a_i]$; $\overline{A}_{\chi'} = \overline{A}_\chi - (\{a_1, .., a_n\} - A_{final})$;
  $<_{\chi'} = <_\chi - \{\langle b, c\rangle \mid \langle b, c\rangle \in <_\chi^+, (b \in (A_{final} - \{a_i\}) \vee c \in (A_{final} - \{a_i\}))\}$;
  $=_{\chi'} = =_\chi - \{\langle b, c\rangle \mid \langle b, c\rangle \in =_\chi^+, (b \in (A_{final} - \{a_i\}) \vee c \in (A_{final} - \{a_i\}))\}$;
  $\iota_{\chi'} = \iota_\chi - \{\langle b, \iota_\chi(b)\rangle \mid b \in (A_{final} - \{a_i\})\}$, $\tau_{\chi'} = \tau_\chi - \{\langle b, \tau_\chi(b)\rangle \mid b \in (A_{final} - \{a_i\})\}$,
  $\lambda_{\chi'} = \lambda_\chi - \{\langle b, \lambda_\chi(b)\rangle \mid b \in (A_{final} - \{a_i\})\}$.

  If there exists a $\langle b, c\rangle \in (<_\chi^+ \cup =_\chi^+)$ such that $b \in (A_{final} - \{a_i\})$ and $c \notin (A_{final} - \{a_i\})$, then the abstraction of χ is impossible. This is explained later on.

In case activity A consists of a combination of conjunctions and disjunctions of final actions a proper combination of the rules above must be applied.

Completion condition of activity *A*

The replacement rules for the occurrences and non-occurrences of the final actions of activity A given above correspond to the rules for defining the completion condition of activity A presented in Section 8.3.1, when considered per execution. The successful or non-successful termination of activity A in a concrete execution is represented by the occurrence or non-occurrence of abstract action a' in the abstract execution, respectively.

Causality condition of abstract action *a'*

The rules for the integration of the causality conditions of the final actions of *A* are reflected in the abstraction of each execution as follows:

(sf) abstract action *a'* occurs iff final action *a* occurs. Occurrence *a'* is involved in the same ordering and synchronization relations in which occurrence *a* is involved;

(cf) abstract action *a'* occurs iff all final actions a_1, .., a_n occur. Occurrence *a'* is involved in the conjunction of the ordering and synchronization relations in which occurrences a_1, .., a_n are involved;

(df) abstract action *a'* occurs iff at least one final action occurs. Occurrence *a'* corresponds to the occurrence of one final action, i.e., the actual final action a_i, such that *a'* is involved in the same ordering and synchronization relations in which a_i is involved.

Attribute values of abstract action *a'*

The rules for the abstraction of the attribute values of the final actions of *A* are reflected in the abstraction of each execution as follows:

(sf) the information, time and location attribute values of *a'* and *a* are the same. The probability of the abstract executions in which *a'* occurs is equal to the probability of the concrete executions in which *a* occurs;

(cf) the information value of *a'* consists of the combination of the information values of a_1, .., a_n, the time value of *a'* is equal to the time value of the latest final action that occurs, and the location value of *a'* comprises the collection of the location values of a_1, .., a_n. The probability of the abstract executions in which *a'* occurs is equal to the probability of the concrete executions in which all a_1, .., a_n occur;

(df) the information, time and location attribute values of *a'* is equal to the corresponding values of the actual final action that occurs. The probability of the abstract executions in which *a'* occurs is equal to the probability of the concrete executions in which the actual final actions occur.

Example: reception of segmented data unit

Figure 8.25 depicts the abstraction of the execution semantics of behaviour *B* in Figure 8.20, when assuming actions *c* and *a* must occur once they are allowed to occur and ignoring the information, time and location attributes.

Abstract execution χ_1' is the abstraction of concrete execution χ_1 in which the conjunction of action occurrences r_1, r_2 and r_3 is replaced by the occurrence of abstract action *r'*. Abstract execution χ_2' is the abstraction of the remaining concrete actions, since in each of these executions one or more of the actions r_1, r_2 and r_3 do not occur, such that the corresponding occurrences and non-occurrences are replaced by the non-occurrence of action *r'*.

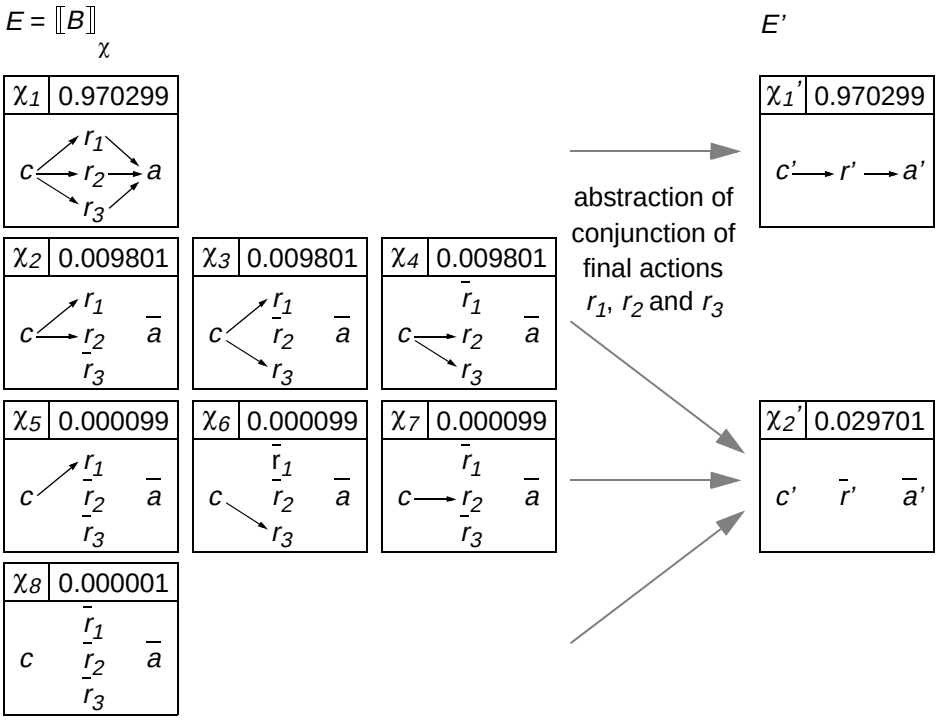


Figure 8.25: Reception of segmented data unit (2)

Example: alternative routing of a data unit (2)

Figure 8.26 depicts the abstraction of the execution semantics of behaviour *B* in Figure 8.21, when assuming actions *c* and *a* must occur once they are allowed to occur and ignoring the information, time and location attributes.

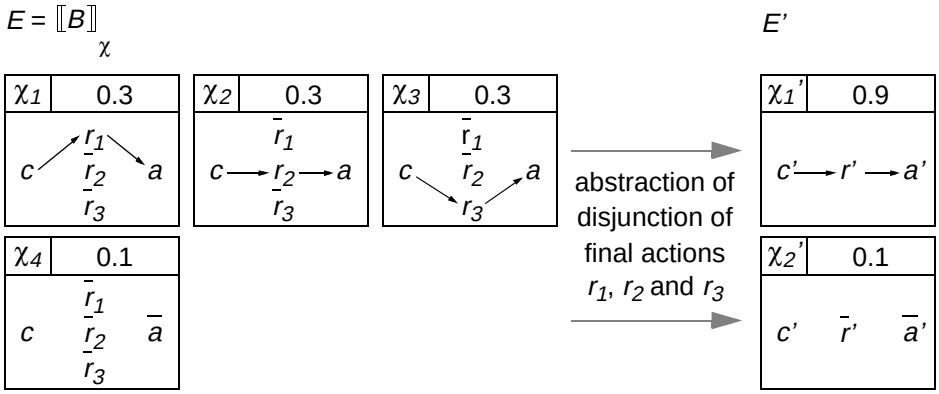


Figure 8.26: Alternative routing of a data unit (2)

Abstract execution χ_1' is the abstraction of concrete executions χ_1, χ_2 and χ_3 in which the disjunction of action occurrences r_1, r_2 and r_3 is replaced by the occurrence of abstract action *r'*. Abstract execution χ_2' is the abstraction of concrete executions χ_4 in which the non-occurrences of actions r_1, r_2 and r_3 is replaced by the non-occurrence of action *r'*.

Impossibility of abstraction

In the case of a conjunction of final actions, the impossibility of abstraction can be identified under some circumstances. For example, assume that a concrete action *b* exists in the example of Figure 8.20, such that *b* depends on the reception of a single data segment. In this case, an execution exists in which action *b* occurs, but not all actions r_1, r_2 and r_3 occur. This implies that when replacing the corresponding occurrences and non-occurrences of r_1, r_2 and r_3 by the non-occurrence of *r'* in this execution, the occurrence of *b* has to be replaced by the non-occurrence of *b*. This implies that the abstraction as defined in our method is impossible.

In the case of a disjunction of (independent) final actions, the impossibility of an abstraction can also be identified under some circumstances. For example, consider behaviour *B* in Figure 8.22, where action *n* is replaced by two independent actions n_1 and n_2, such that n_1 is enabled by a_1 and n_2 is enabled by a_2. This implies that an execution in which a_1, a_2, n_1 and n_2 occur is possible. The abstraction of this execution is impossible, since n_1 and n_2 depend on distinct final actions of activity *a*, whereas activity *a* defines a disjunction of final actions.

8.5 Example: client-server interactions

This section illustrates some of the refinement rules discussed in the previous sections by the design of a client-server interaction. In this example, we concentrate on the refinement of action relations and do not consider quantitative aspects.

We adopt the following conventions in graphical behaviour representations: an action identifier starts with the name of the (inter)action point at which the action happens, e.g., *Areq* is an action that happens at action point *A*, and information attributes are represented by their sorts, e.g., *server_type* represents an information value *v* of sort *server_type*. Furthermore, we place action identifiers next to their corresponding actions, instead of placing them inside text-boxes.

8.5.1 Initial design

The initial design focuses on the interaction between a client application and a data server, abstracting from the identification of the data server and from the remote communication with the data server. During the design process this abstraction should be preserved for the client application, i.e., the design of the internal operation of the client application can be based on the interaction (pattern) as established in this initial design.

Figure 8.27 depicts the composition of entities distinguished in this design. The *client_application* entity interacts with the *data_server* entity at an interaction point *A*.

Figure 8.28 depicts the behaviour at interaction point *A*. The client application requests certain data from the data server through the *Areq* action. This action has two information attributes: the *server_type* attribute that specifies the type of data server that is required, and

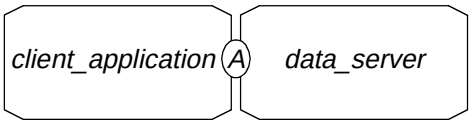


Figure 8.27: Entity domain representation of the initial design

the *question* attribute that specifies the properties of the data requested. The *Areq* action is followed by either an *Arsp* action, in case the requested data was found, or an *Arej* action, in case the data server failed to find the requested data. The information value attribute *answer* of the *Arsp* action contains the requested data. Although this is not indicated by the attributes of the *Arej* action, there may be two main causes for failure: the data server does not have the requested data, or a time-out occurs before the data server is able to respond. The specific rejection reason is not explicitly indicated in the sequel. This also applies to other 'reject' actions that are identified later on.

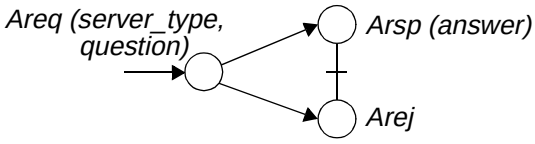


Figure 8.28: Behaviour domain representation of the initial design

8.5.2 Introduction of a trader component

The first design step focuses on the mechanism to locate a suitable data server. We assume that there are potentially multiple data servers that can provide the requested data. Hence, we introduce a trader component, which is able to provide the name of a specific data server, given the type of the data server and the name of the client application. Since we want to hide the existence of a trader from the client application, we introduce a fourth component, called the interface handler, which provides the original interface to the client application.

Figure 8.29 shows the entities identified above. At this abstraction level, the *client_application* entity interacts with the *interface_handler* entity, instead of with the *data_server* entity, at interaction point *A*. The *interface_handler* entity interacts with the *trader* entity and with the *data_server* entity at interaction points *B* and *C*, respectively.

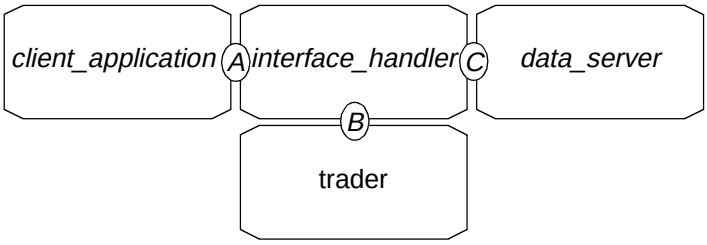


Figure 8.29: Entity domain representation after the introduction of a trader

Figure 8.30 depicts the refined behaviour. After the *Areq* action, the interface handler asks the trader for a suitable data server through a *Breq* action. The *Breq* action has two information attributes, viz. *server_type* and *client_name*. These are used by the trader to determine a

suitable data server. If the search for a suitable server is successful, the *Breq* action is followed by a *Brsp* action with information attribute *server_name*. Otherwise, a *Brej* action is performed. The *Brsp* action is followed by a *Creq* action, in which data is requested from the data server. The properties of the requested data are defined in the information attribute *question* and the data server is identified by means of the information attribute *server_name*. The *Creq* action is followed by either a *Crsp* action, in case the requested data was found, or a *Crej* action, in case the data server failed to find the requested data. The information attribute *answer* of the *Crsp* action contains the requested data. The *Crsp* action enables the *Arsp* action, which has *answer* as its information attribute. An *Arej* action happens after a *Brej* or a *Crej*.

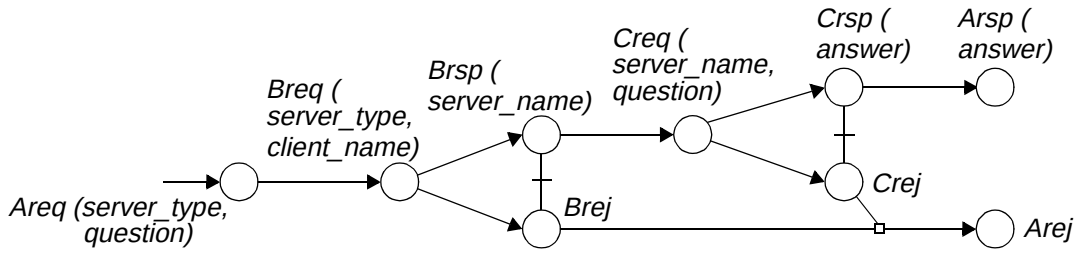


Figure 8.30: Behaviour domain representation after the introduction of a trader

Conformance assessment

The concrete behaviour of Figure 8.30 conforms to the abstract behaviour of Figure 8.28. This can be assessed by applying the method of Section 8.1.3 as follows:

1. identify concrete actions *Areq*, *Arsp* and *Arej* of Figure 8.30 as single reference actions and identify the remaining actions as inserted actions;
2. abstract from the inserted actions (in any arbitrary order) using the method of Section 8.2;
3. compare the abstract behaviour obtained in step 2 with the behaviour of Figure 8.28. These two behaviours are equivalent.

Alternative applications of the method of Section 8.1.3 that render the same result are possible. For example, one may consider concrete actions *Creq*, *Crsp* and *Crej* as an activity with two final actions, obtained through action refinement of an abstract action *C*. In this case, one replaces the activity by the abstract action *C* first, and subsequently abstracts from *C* by considering this action as an inserted action.

8.5.3 Federation of traders

This design step focuses on trader federation and concurrent use of different traders. This step assumes that there is not a single trader that knows all possible data servers, but actually multiple traders and multiple trading domains. A trader in the domain associated with the client application (local domain) passes a request to a trader in a remote domain in case it can not find a data server. This process is based on agreed procedures known as trader federation. In order to speed up the search process, the local trader may contact multiple

traders at the same time. In the following, we consider the situation where a local trader concurrently contacts two remote traders.

Figure 8.31 depicts the entities involved in the trader federation considered in this example.

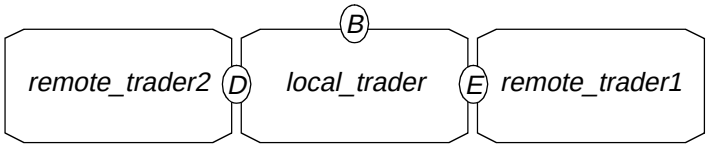


Figure 8.31: Entity domain representation after the introduction of trader federation

Figure 8.32 shows the behaviour of the three federated traders.

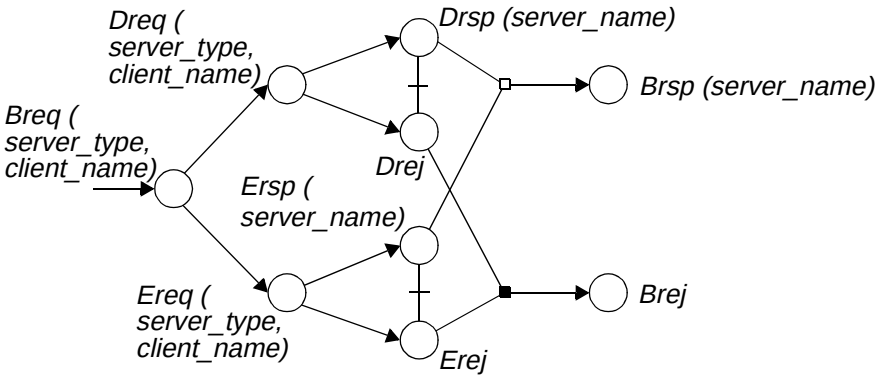


Figure 8.32: Behaviour domain representation after the introduction of trader federation

Conformance assessment

The concrete behaviour of Figure 8.32 conforms to the abstract sub-behaviour of Figure 8.30 consisting of actions *Breq*, *Brsp* and *Brej*, and their relations. This can be assessed by applying the method of Section 8.1.3 analogously to the previous section. Figure 8.33 depicts the abstraction of the concrete behaviour of Figure 8.32.

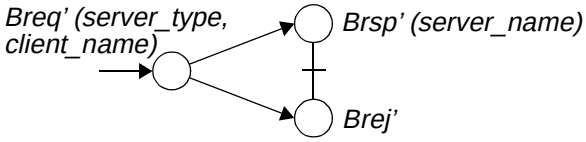


Figure 8.33: Abstraction of the concrete behaviour in Figure 8.32

8.5.4 Remote communication

The final design step focuses on the remote communication between components. We assume that the components identified so far reside on different end-systems of a distributed system. Hence, the communication between components is accomplished via an intermediate component that may not be reliable (i.e., messages may get lost).

Figure 8.34 depicts the entities involved in communication between the interface handler and the data server.

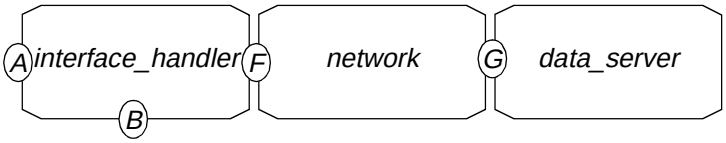


Figure 8.34: Entity domain representation after the consideration of remote communication

Figure 8.35 depicts the behaviour of an instance of communication between the interface handler and the data server.

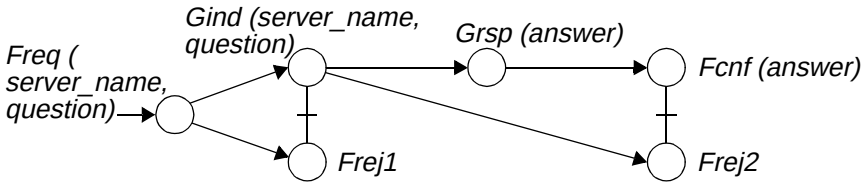


Figure 8.35: Behaviour domain representation after the consideration of remote communication

Conformance assessment

The concrete behaviour of Figure 8.35 conforms to the abstract sub-behaviour of Figure 8.30 consisting of actions *Creq*, *Crsp* and *Crej*, and their relations. This can be assessed by applying the method of Section 8.1.3 as follows:

1. identify concrete actions *Freq* and *Fcnf* as single reference actions and identify concrete actions *Frej1* and *Frej2* as a group of reference actions;
2. abstract from inserted actions *Gind* and *Grsp* using the method of Section 8.2;
3. replace final actions *Frej1* and *Frej2* by abstract action *Frej'*, which represents two alternative causes for rejection;
4. compare the abstract behaviour obtained in step 3 with the abstract sub-behaviour of Figure 8.30 consisting of actions *Creq*, *Crsp* and *Crej*. These two behaviours are equivalent.

Figures 8.36 (i) and (ii) depict the abstract behaviours obtained in step 2 and 3, respectively.

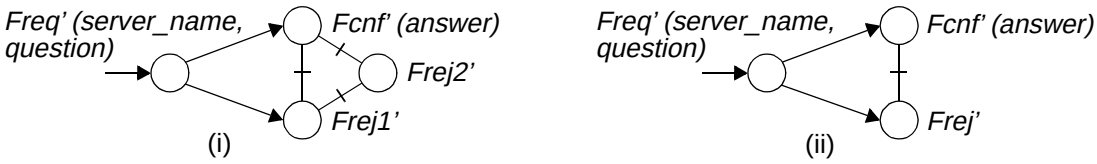


Figure 8.36: Abstraction of the concrete behaviour in Figure 8.35

8.6 Conclusions

In this chapter we present an integrated set of methods to perform behaviour refinement. The objective of behaviour refinement is to replace an abstract behaviour by a more concrete behaviour that conforms to this abstract behaviour. Our methods support two basic types of behaviour refinement: *causality refinement*, in which causality relations between

abstract actions are replaced by causality relations involving their corresponding concrete actions and some inserted actions, and *action refinement*, in which an abstract action is replaced by an activity involving multiple concrete actions and their causality relations. The methods are based on assessing the conformance relation between the abstract behaviour and the corresponding concrete behaviour. This assessment involves the determination of the abstraction of the concrete behaviour and the comparison of this abstraction with the original abstract behaviour. Rules to perform the abstraction and comparison operations are developed.

The methods presented in this chapter are sufficiently detailed to form a basis for the development of software tools that automate parts of these methods. In particular, a direct implementation of the execution-based methods of Section 8.4 seems possible. These methods are complete and rather simple when compared to the corresponding causality-based methods of Sections 8.2 and 8.3. A potential problem associated with a tool that supports an execution-based method is the large number of executions that may have to be manipulated. This number can be restricted, however, by delimiting concrete behaviours to the causality context of the inserted and final actions that have to be removed in order to obtain abstractions of these concrete behaviours.

The availability of tools that partly automate the abstraction from inserted and final actions allows one to develop case studies of more complex behaviours. These case studies can be used to verify, improve and elaborate the methods that have been presented in this chapter. For example, the causality-based methods of Sections 8.2 and 8.3 need further elaboration on abstraction rules for the combined use of the simple and extended probability attribute, and rules for the integration of equivalent alternative causality conditions.

Behaviours involving interactions

This chapter has not considered the refinement of behaviours involving interactions. Figure 8.37 illustrates a strategy to support the refinement of such behaviours, which (re-)uses the methods presented in this chapter. Step I represents the replacement of interactions by their corresponding (more abstract) definition as integrated interactions (see Section 2.4), which renders a behaviour consisting of actions only. Step II represents the behaviour refinement design operation as discussed in this chapter. Step III represents the decomposition of some (integrated inter)actions into interactions.

Examples of the refinement of behaviours involving interactions can be found in [57]. This paper presents, amongst others, a case study in which a financial transaction involving multiple banks is designed at subsequent abstraction levels. These abstraction levels correspond to the design milestones presented in Section 2.2.

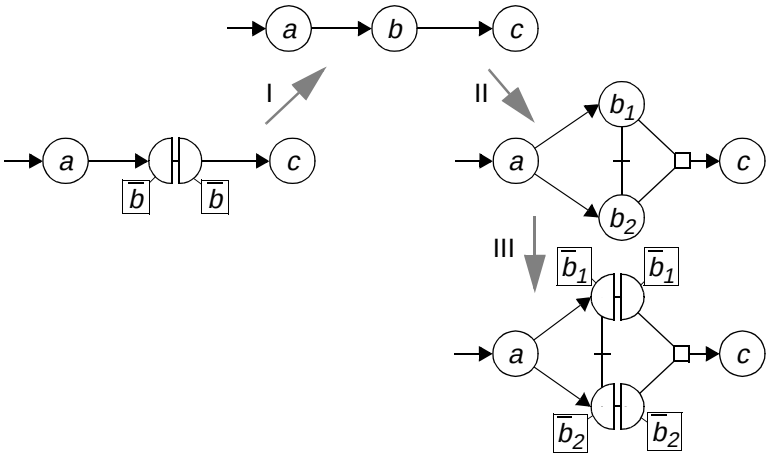


Figure 8.37: Refinement of behaviours involving interactions

Chapter 9
Causality-oriented behaviour composition

This chapter discusses the causality-oriented structuring technique, which is a technique to compose behaviours from sub-behaviours. The purpose of this chapter is to address the problem of modelling repetitive behaviours using the causality-oriented structuring technique, and to give a simple solution. For this purpose, the notions of behaviour type and behaviour instantiation are introduced, which allow designers to (dynamically) create new instances of a behaviour.

The causality-oriented structuring technique extended with behaviour instantiation provides the necessary means to develop case studies of large and complex behaviours, possibly involving repetitive behaviours, which require structuring in terms of smaller, less complex sub-behaviours. One such a case study is performed in Chapter 10.

The structure of this chapter is as follows. Section 9.1 explains the causality-oriented structuring technique. Section 9.2 defines the notions of behaviour type and behaviour instantiation and discusses the modelling of repetitive behaviours. Section 9.3 briefly considers the execution semantics of behaviour definitions using behaviour instantiation. And Section 9.4 presents the conclusions.

9.1 Causality-oriented structuring

The definition of causality relations between actions can be generalized to the definition of causality relations between behaviours. This allows the structuring of a complex behaviour in terms of less complex sub-behaviours and their relationships. Furthermore, this technique allows predefined sub-behaviours to be reused through instantiation, and repetitive behaviours to be represented through repeated behaviour instantiation. This structuring technique, which is called *causality-oriented structuring*, makes use of:

- *entry points*, which are points in a behaviour from which actions of that behaviour can be enabled by conditions involving actions of other behaviours;
- *exit points*, which are causality conditions in a behaviour that can be used to enable actions of other behaviours.

Behaviours can be composed by relating their exit and entry points, which are indicated by the keywords *exit* and *entry*, respectively. A behaviour may have multiple exit and entry points.

The remainder of this section illustrates the use of the causality-oriented structuring technique. For an elaborate discussion we refer to [16]. Although causality-oriented structuring is a technique to compose and decompose behaviour definitions, in the sequel we mainly use and refer to this technique as a means to compose behaviours.

9.1.1 Single entry and exit

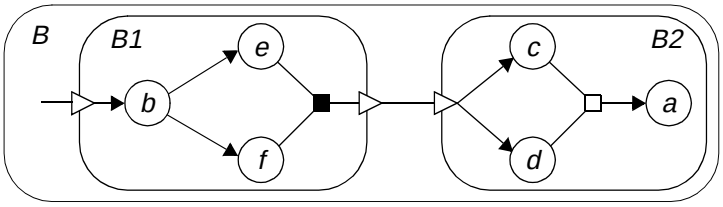
Figure 9.1(i) depicts the causality-oriented composition of behaviour *B* from sub-behaviours *B1* and *B2*. The corresponding textual representations of *B1*, *B2* and *B* are as follows:

$$B1 = \{ \; entry \rightarrow b,\ b \rightarrow e,\ b \rightarrow f,\ e \wedge f \rightarrow exit \; \};$$

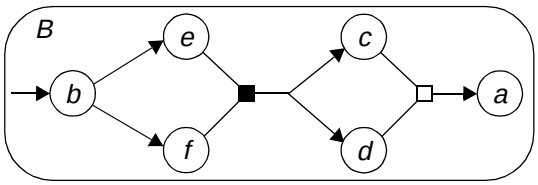
$$B2 = \{ \; entry \rightarrow c,\ entry \rightarrow d,\ c \vee d \rightarrow a \; \};$$

$$B = \{ \; \surd \rightarrow B1.entry,\ B1.exit \rightarrow B2.entry \; \}.$$

Figure 9.1(ii) depicts the corresponding monolithic definition of *B*.



(i) causality-oriented structure



(ii) monolithic structure

Figure 9.1: Single entry and exit

Sub-behaviours *B1* and *B2* both have one entry point. These entry points can be considered as place-holders for the causality conditions of the initial action(s) of *B1* and *B2*. This is represented by the statement $entry \rightarrow b$ in case of *B1* and the statements $entry \rightarrow c$ and $entry \rightarrow d$ in case of *B2*, where action *b* is the initial action of *B1* and actions *c* and *d* are the initial actions of *B2*.

Sub-behaviour *B1* has one exit point, which corresponds to causality condition $e \wedge f$. This is represented by the statement $e \wedge f \rightarrow exit$.

Statement $B1.exit \rightarrow B2.entry$ combines the exit point of *B1* with the entry point of *B2*. This implies that condition $e \wedge f$ in *B1* becomes the causality condition of actions *c* and *d* in *B2*.

This exit/entry combination models the sequential composition of *B1* and *B2*, since only enabling conditions are involved.

An exit/entry combination associates indirectly a causality condition with an entry point via an exit point. Instead, one may associate a causality condition with an entry point directly. For example, the statement $\surd \rightarrow B1.entry$ defines that action *b* is enabled by the start condition.

The association of a causality condition with an exit or entry point is represented as a causality relation. However, in contrast to a result action, action attributes can not be associated with an exit or entry point, since exit points and entry points do not model activities. Instead, exit and entry points are introduced as syntactic artefacts that allow one to define a result action and (parts of) its causality condition in different sub-behaviours.

Entry and exit points are graphically represented by the symbol $\triangleright$, pointing in the direction of the enabled behaviours. The combination of an exit and an entry point is graphically represented by linking the corresponding exit and entry symbols with a solid line.

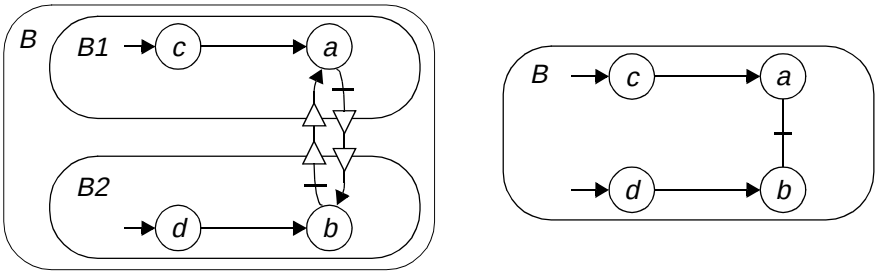
Figure 9.2(i) depicts the composition of behaviour *B* from sub-behaviours *B1* and *B2*. The corresponding textual representations of *B1*, *B2* and *B* are as follows:

$$B1 = \{ \surd \rightarrow c,\ c \wedge entry \rightarrow a,\ \neg a \rightarrow exit \};$$

$$B2 = \{ \surd \rightarrow d,\ d \wedge entry \rightarrow b,\ \neg b \rightarrow exit \};$$

$$B = \{ B1.exit \rightarrow B2.entry,\ B2.exit \rightarrow B1.entry \}.$$

Figure 9.2(ii) depicts the corresponding monolithic definition of *B*.



(i) causality-oriented structure (ii) monolithic structure

Figure 9.2: Single entry and exit (2)

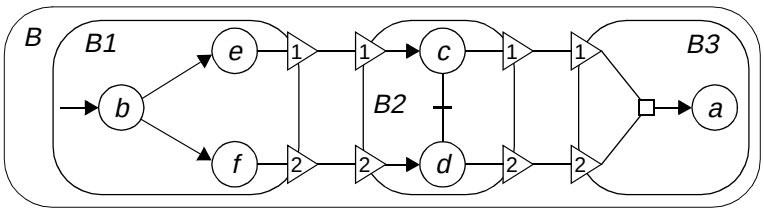
The exit/entry combinations as represented by the statements $B1.exit \rightarrow B2.entry$ and $B2.exit \rightarrow B1.entry$, model a choice between the successful termination of *B1* and the successful termination of *B2*. Because choice is a two-sided relation, two exit/entry combinations are needed.

9.1.2 Multiple entries and exits

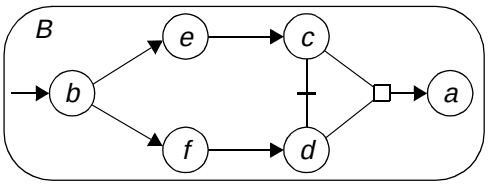
Figure 9.3(i) depicts the composition of behaviour *B* from sub-behaviours *B1*, *B2* and *B3*. The corresponding textual representations of *B1*, *B2*, *B3* and *B* are as follows:

$$
\begin{aligned}
B1 &= \{\ \surd \rightarrow b,\ b \rightarrow e,\ b \rightarrow f,\ e \rightarrow exit1,\ f \rightarrow exit2\ \};\\
B2 &= \{\ entry1 \wedge \neg d \rightarrow c,\ entry2 \wedge \neg c \rightarrow d,\ c \rightarrow exit1,\ d \rightarrow exit2\ \};\\
B3 &= \{\ entry1 \vee entry2 \rightarrow a\ \};\\
B &= \{\ B1.exit1 \rightarrow B2.entry1,\ B1.exit2 \rightarrow B2.entry2,\\
&\quad\ B2.exit1 \rightarrow B3.entry1,\ B2.exit2 \rightarrow B3.entry2\ \}.
\end{aligned}
$$

Figure 9.3(ii) depicts the corresponding monolithic definition of *B*.



(i) causality-oriented structure



(ii) monolithic structure

Figure 9.3: Multiple entries and exits

The sub-behaviours in Figure 9.3(i) have multiple entries or exits. In order to distinguish between multiple entries and exits of a single sub-behaviour, the keywords *exit* and *entry* are appended with a unique identifier. In the graphical representation, this identifier is depicted inside the symbol $\triangleright$. In this thesis, we attach different natural numbers to different exits and entries in order to distinguish them.

9.1.3 Parameterized entries and exits

In general, an entry or an exit can be parameterized with a list of information, time or location variables, which comprises all information passed from the ‘exit’ behaviour to the ‘entry’ behaviour. These variables contain a selection of the information, time and location values of the enabling actions that can be referred to when the causality condition associated with this entry or exit is satisfied. A requirement on the combination of an exit and an entry is that both have the same parameter list, i.e., the same number of variables, having the same type and specified in the same order. Only the variable names of these parameter lists may differ.

Consider the example of Figure 9.1. Assume that actions *e* and *f* establish an information, time and location value, and assume that actions *c* and *d* want to refer to the information values of *e* and *f* and the time value of *e*. For this purpose, the exit of *B1* and the entry of *B2* are extended with a parameter list consisting of two information variables *v1* and *v2* and a time variable *v3*. The causality relations obtained with this extension are defined below.

$$
\begin{aligned}
B1 &= \{ \ \ldots, \\
&\quad e \wedge f \rightarrow exit\ (v1, v2 : \boldsymbol{I}, v3 : \boldsymbol{T})\ [v1 = \iota_e, v2 = \iota_f, v3 = \tau_e]\ \}; \\
B2 &= \{ \ entry\ (v1, v2 : \boldsymbol{I}, v3 : \boldsymbol{T}) \rightarrow c\ (\iota_c : \mathrm{I}_c, \tau_c : \mathrm{T}_c)\ [\iota_c = v1 + v2, \tau_c < v3 + 1], \\
&\quad entry\ (v1, v2 : \boldsymbol{I}, v3 : \boldsymbol{T}) \rightarrow d\ (\iota_d : \mathrm{I}_d, \tau_d : \mathrm{T}_d)\ [\iota_c = v1 - v2, \tau_c < v3 + 2], \\
&\quad \ldots\ \}; \\
B &= \{ \ \surd \rightarrow B1.entry, B1.exit \rightarrow B2.entry\ \}.
\end{aligned}
$$

The statement $B1.exit \rightarrow B2.entry$ is allowed, since the parameter lists of $B1.exit$ and $B2.entry$ match. This statement implicitly defines that the parameters of $B2.entry$ get the same values as the corresponding parameters of $B1.exit$. Since the parameter list of an entry/exit combination needs to be specified only once, the causality relation of $B1.exit$ may be simplified to: $e \wedge f \rightarrow exit\ (\iota_e, \iota_f, \tau_e)$. Figure 9.4 illustrates the graphical representations of the causality relations of $B1.exit$ and action c.

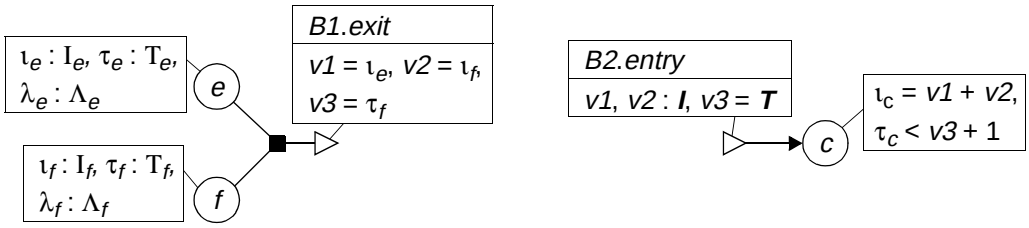


Figure 9.4: Parameterized exits and entries

In general, the parameterization of an exit/entry combination facilitates the composition of two sub-behaviours, since one does not have to know the enabling actions that establish the information, time and location values passed via this exit/entry when specifying the enabled sub-behaviour. One does not have to know the enabled actions that use these values when specifying the enabling sub-behaviour either.

9.1.4 Interpretation

The causality-oriented structuring technique is a pure syntactic operation on behaviour definitions. The semantics of a causality-oriented behaviour definition B_{caus} is equal to the semantics of the corresponding monolithic behaviour definition B_{mono} that consists of the union of the causality relations of the sub-behaviours in B_{caus}, in which entries are replaced by their corresponding causality conditions, causality relations of exits are removed, and references to exit/entry parameters are replaced by references to the action attribute values that are assigned to these parameters. We assume that action names are unique across different sub-behaviours of B_{caus}.

In the graphical representation, causality-oriented structuring can be considered as partitioning the monolithic behaviour structures into two or more sub-structures, where sub-structures are obtained by cutting through the arrows or lines between actions.

9.2 Behaviour instantiation

The definition of exit and entry points in a behaviour definition allows one to re-use this behaviour definition in the composition of larger behaviour definitions. The exit and entry points define the possible ways in which a behaviour definition can be combined with other behaviour definitions.

9.2.1 Multiple behaviour instances

The reuse of some pre-defined sub-behaviour can be considered as copying the definition of this sub-behaviour and relabelling the obtained behaviour copy, including its actions, with unique names. This uniquely relabelled behaviour copy is called a *behaviour instance*, and its actions are called *action instances*.

In the definition of behaviours, we do not want to explicitly copy sub-behaviours in order to create new instances. Instead, we want to have a single behaviour definition and a mechanism to (dynamically) create new instances from this definition. This single behaviour definition is called a *behaviour type* definition, since it abstracts from the identity of its instances. In other words, a behaviour type represents a collection of behaviour instances, which differ only w.r.t. their identity and the identity of their action instances. The actions of a behaviour type are called *action types*.

Type and instance names

When a new instance of a behaviour type definition is created, this behaviour instance and each of its action instances should be labelled with a unique identifier (name). In principle, any labelling strategy can be followed as long as it renders uniquely labelled behaviour and action instances. In this thesis, we want names of behaviour instances and action instances to reflect directly whether they are instances of the same type or not. For this purpose, *type names* and *instance names* are introduced.

Type names uniquely identify behaviour or action types. Instance names uniquely identify behaviour or action instances. A behaviour instance name is defined as a tuple ⟨behaviour type name, instance identifier⟩, and an action instance name is defined as a tuple ⟨action type name, instance identifier⟩. Alternatively, instance names are represented with the instance identifier as a superscript of the type name, e.g., instance names a^1 and B^2 are alternative representations of $\langle a, 1\rangle$ and $\langle B, 2\rangle$. In this thesis, we normally use instance identifiers from the set of natural numbers.

Behaviour type definitions

The textual representation of a behaviour type definition has the following syntax:

```
behaviour_type_definition   =  behaviour_type_name "= {" causality_relation_part
                               ["where" sub_behaviour_part] "}"
sub_behaviour_part          =  behaviour_type_definition ["; " behaviour_type_definition]* .
```

A behaviour type definition is divided into two parts, which are separated by the keyword *where*:

- a *causality-relation-part*, which represents the definition of the causality relations of this behaviour type;
- a *sub-behaviour-part*, which represents the definition of the sub-behaviour types that are instantiated in the causality-relation-part of this behaviour type.

Causality relations are defined in the same way as before, except that action names are replaced by action type names.

The addition of the sub-behaviour-part implies that a hierarchy of behaviour type definitions can be defined. In case behaviour type *B1* is defined in the sub-behaviour-part of behaviour type *B*, *B1* is called a sub-behaviour (type) of *B* and *B* is called a super-behaviour (type) of *B1*. The super- and sub-relationship between behaviour types are transitive.

Behaviour instantiation

A behaviour instance of some behaviour type *B* with instance identifier *id* is created in one of the following ways:

1. through the definition of the causality relation of one of its entry points $entry_i$, which is denoted as $B^{id}.entry_i$;
2. through the definition of the combination of one of its exit points $exit_i$, which is denoted as $B^{id}.exit_i$, with an entry point of another behaviour or with an exit point of its super-behaviour.

In case behaviour type *B* has multiple entry or exit points, only one instance B^{id} is created.

The creation of a new instance of behaviour type *B* implies the creation of new instances of all action types in *B*. We use the convention that all action instances of some behaviour instance B^{id} inherit instance identifier *id*.

Behaviour instances are created in the causality-relation-part of some behaviour type definition. Scope rules should be defined to determine which behaviour types can be instantiated. In this thesis, we assume that given some behaviour type definition *B*, the causality-relation-part of *B* may define instantiations of any super- or sub-types of *B*, or of *B* itself.

Examples

Figure 9.5 depicts sub-behaviour *B2* and behaviour *B*, which is composed from three instances of sub-behaviour *B2*. The textual definition of *B* is as follows:

$$
\begin{aligned}
B \quad = \quad \{ \quad & B.entry1 \rightarrow B2^1.entry1, \\
& B.entry2 \rightarrow B2^1.entry2, \\
& B.entry2 \rightarrow B2^2.entry1, \\
& B.entry3 \rightarrow B2^2.entry2, \\
& B2^1.exit \rightarrow B2^3.entry1,
\end{aligned}
$$

$B2^2.exit \rightarrow B2^3.entry2,$
$B2^3.exit \rightarrow B.exit$
where
$B2 = \{ entry1 \wedge entry2 \rightarrow a, a \rightarrow exit \}$
}.

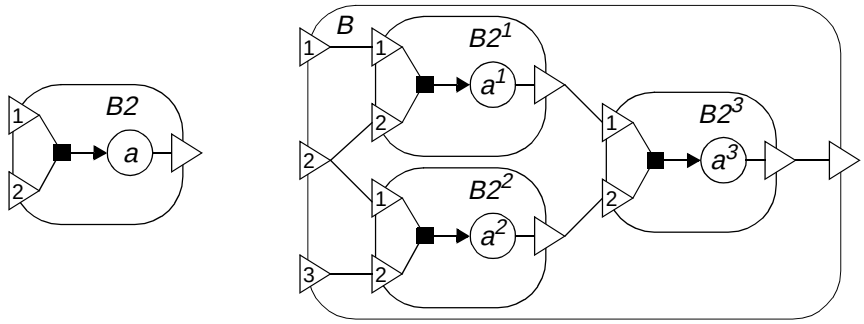


Figure 9.5: Multiple behaviour instances of *B2*

Figure 9.6 depicts behaviour *B1* which reuses behaviour *B* as defined above. The textual definition of *B1* is as follows:

$B1 = \{ \surd \rightarrow b, \surd \rightarrow c, \surd \rightarrow d,$
$b \rightarrow B.entry1, c \rightarrow B.entry2, d \rightarrow B.entry3,$
$B.exit \rightarrow d$
where
$B = \{$ # see above # $\}$
}.

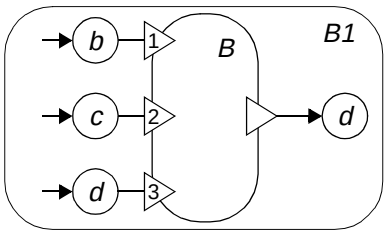


Figure 9.6: Reuse of behaviour *B* as defined in Figure 9.5

Since *B* is instantiated only once in *B1*, it is not necessary to extend behaviour type name *B* with an instance identifier to denote the instance of *B* in Figure 9.6. Behaviour *B1* can not be reused by any other behaviour, since it does not define any exit or entry points.

9.2.2 Repetitive behaviours

Repetitive behaviours are modelled through repeated behaviour instantiation. Two techniques for performing repeated behaviour instantiation are distinguished:

1. *replicated behaviour instantiation*, in which the repetition of some behaviour type *B* is modelled through the repeated instantiation of *B* by its super-behaviour type. In this case, the super-behaviour type of *B* is responsible for creating multiple instances of *B* and for defining the relationships between these instances;

2. *recursive behaviour instantiation*, in which the repetition of some behaviour type *B* is modelled through the (recursive) instantiation of *B* by *B* itself. In this case, behaviour type *B* itself is responsible for instantiating one or more new instances of *B* and for defining the relationship between the current instance and the new instance(s) and between new instances.

Example: unconfirmed service

This example illustrates the use of replicated and recursive behaviour instantiation for the modelling of an unconfirmed data transfer service, which supports the error-free and in-sequence delivery of data units. Since data loss is not allowed, we assume that uncertainty associations get the value *must* by default.

The data service is modelled by the repeated instantiation of a sub-behaviour, called *B*, which models the transfer of a single data unit. Actions *s* and *r* model the sending and receiving of this data unit, respectively.

Using replicated behaviour instantiation, the data service is modelled through the repeated instantiation of *B* by super-behaviour *B1*, such that all instances of *B* are composed sequentially. Behaviour *B1* is defined as follows:

$$
\begin{array}{lll}
B1 & = & \{ \ \surd \rightarrow B^{0}.entry1,\ \surd \rightarrow B^{0}.entry2, \\
 & & \quad i \in \boldsymbol{N} \mid B^{i}.exit1 \rightarrow B^{i+1}.entry1, \\
 & & \quad i \in \boldsymbol{N} \mid B^{i}.exit2 \rightarrow B^{i+1}.entry2, \\
 & & \quad \text{where} \\
 & & \qquad B \ = \ \{ \ entry1 \rightarrow s,\ entry2 \wedge s \rightarrow r, \\
 & & \qquad\qquad s \rightarrow exit1,\ r \rightarrow exit2 \ \} \\
 & & \}.
\end{array}
$$

This behaviour definition uses variable *i*, which ranges over the possible instance identifiers of *B*. Identical causality relation definitions with different values for *i*, are generalized into a single definition consisting of two parts, separated by the symbol "|": (i) a condition defining the possible values of *i*, and (ii) a causality relation parameterized with *i*. Such a generalized causality relation definition represents a set of causality relations, one for each possible value of *i*. For example, assume that condition $i \in \boldsymbol{N}$ in *B1* is replaced by condition $i \in \boldsymbol{N} \wedge i \leq 5$. In this case, *B1* models a data service that transfers six data units.

Using recursive behaviour instantiation, the data service is modelled by making behaviour *B* instantiate itself. In this case the super-behaviour of *B*, called *B2*, only instantiates the first instance of *B*. Behaviour *B2* is defined as follows:

$$
\begin{array}{lll}
B2 & = & \{ \ \surd \rightarrow B^{0}.entry1,\ \surd \rightarrow B^{0}.entry2, \\
 & & \quad \text{where} \\
 & & \qquad B \ = \ \{ \ entry1 \rightarrow s,\ entry2 \wedge s \rightarrow r, \\
 & & \qquad\qquad s \rightarrow B^{self+1}.entry1,\ r \rightarrow B^{self+1}.entry2 \ \} \\
 & & \}
\end{array}
$$

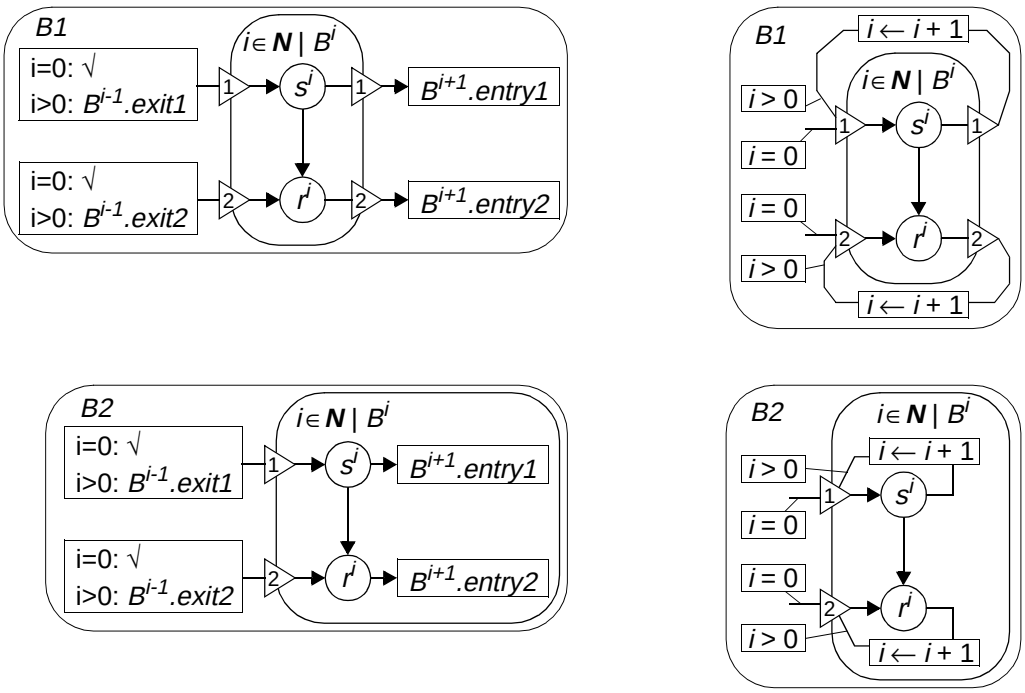
This behaviour definition uses function *self*, which renders the instance identifier of the current instance of the behaviour type in which it is used. For example, in order to model a data

service that transfers six data units, the causality relations of $B^{self+1}.entry1$ and $B^{self+1}.entry2$ would be modified as follows:

$self \leq 4 \mid s \rightarrow B^{self+1}.entry1,$

$self \leq 4 \mid r \rightarrow B^{self+1}.entry2.$

Figure 9.7(i) and (ii) depict two alternative graphical representations of behaviours *B1* and *B2*. In Figure 9.7(i), the relationship between B^i and other instances of *B* is represented using text-boxes. In Figure 9.7(ii), this relationship is represented by a 'cyclic' arrow labelled with a text-box, such that the arrow runs through the text-box. The label $i \leftarrow i + 1$ denotes that B^{i+1} depends on B^i. Furthermore, constraints $i = 0$ and $i > 0$ denote that the entry points of B^i are associated with the start condition in case $i = 0$, and are associated with the exit points of B^{i-1} in case $i > 0$.



(i) (ii)

Figure 9.7: Alternative graphical representations of *B1* and *B2*

Example: unconfirmed service with capacity M

The following behaviour defines an unconfirmed service with capacity *M*:

$$
\begin{aligned}
B3 \;=\; \{ \;& \surd \rightarrow B^0.entry1,\ \surd \rightarrow B^0.entry2, \\
& i \in \boldsymbol{N},\ i < M \mid \surd \rightarrow B^i.entry3, \\
& \text{where} \\
& \quad B \;=\; \{ \; entry1 \wedge entry3 \rightarrow s,\ entry2 \wedge s \rightarrow r, \\
& \qquad\quad s \rightarrow B^{self+1}.entry1,\ r \rightarrow B^{self+1}.entry2, \\
& \qquad\quad r \rightarrow B^{self+M}.entry3 \; \} \\
& \}.
\end{aligned}
$$

This behaviour is derived from behaviour *B2* of the previous example. A third entry point is added to behaviour *B*, which represents the condition that maximal *M* data units may be in transit. This entry point is added to the causality condition of action *s*. For example, entry point $B^i.entry3$ is associated with the start condition in case $i < M$, and is associated with causality condition r^{i-M} in case $i \geq M$, which models that action instance s^i can not occur before action instance r^{i-M} has occurred.

Capacity *M* can be considered as a parameter of behaviour *B3*. This can be made explicit in the design notation by extending behaviour definitions with a parameter list.

Repetitive causality conditions

The causality condition of an action *a* defined in some sub-behaviour B_1 may contain repetitive sub-conditions in case this action depends on an action type *b* in another sub-behaviour B_2 that may be instantiated multiple times. Furthermore, the causality condition of *a* may become infinitely large in case sub-behaviour *B2* may be instantiated infinitely many times.

For example, consider behaviour *B4*, composed from an adapted version of sub-behaviour *B* as defined in behaviour *B1* above, and from sub-behaviour B_{dis}, which models the possible disabling of action *s* by action *d*.

$$
\begin{aligned}
B4 \;=\; \{ \;& \surd \rightarrow B^0.entry1,\ \surd \rightarrow B^0.entry2,\\
& i \in \boldsymbol{N} \mid B^i.exit1 \rightarrow B^{i+1}.entry1,\\
& i \in \boldsymbol{N} \mid B^i.exit2 \rightarrow B^{i+1}.entry2,\\
& i \in \boldsymbol{N} \mid B_{dis}.exit \rightarrow B^i.entry3,\\
& \wedge_{i \in \boldsymbol{N}}\, B^i.exit3 \rightarrow B_{dis}.entry\\
& \text{where}\\
& \quad B \;=\; \{ \; entry1 \wedge entry3 \rightarrow s,\ entry2 \wedge s \rightarrow r,\\
& \qquad\quad s \rightarrow exit1,\ r \rightarrow exit2,\ s \vee \neg s \rightarrow exit3 \; \};\\
& \quad B_{dis} \;=\; \{ entry \rightarrow d,\ \neg d \rightarrow exit \; \}\\
\}.\ &
\end{aligned}
$$

Figure 9.8 depicts the first two instances of behaviour type *B* in *B4*. The remaining instances are indicated by dashed lines.

Behaviour type B_{dis} is instantiated only once. The behaviour and action instance are denoted as B_{dis} and *d*, respectively. Behaviour *B* is instantiated infinitely many times. The corresponding behaviour and action instances are denoted as B^i, and s^i and r^i, respectively, with $i \in \boldsymbol{N}$.

Behaviour types *B* and B_{dis} define that action type *d* may disable action type *s*. This implies that action instance *d* may disable any action instance s^i. Consequently, the causality relation of action instance *d* is defined as:

$$\wedge_i\, (s^i \vee \neg s^i) \rightarrow d\ .$$

Despite that action instance *d* depends on infinitely many action instances s^i, its implementation is, in principle, not more difficult than the case in which *d* disables a limited number of instances s^i. This is because action *d* can occur due to the non-occurrence of each

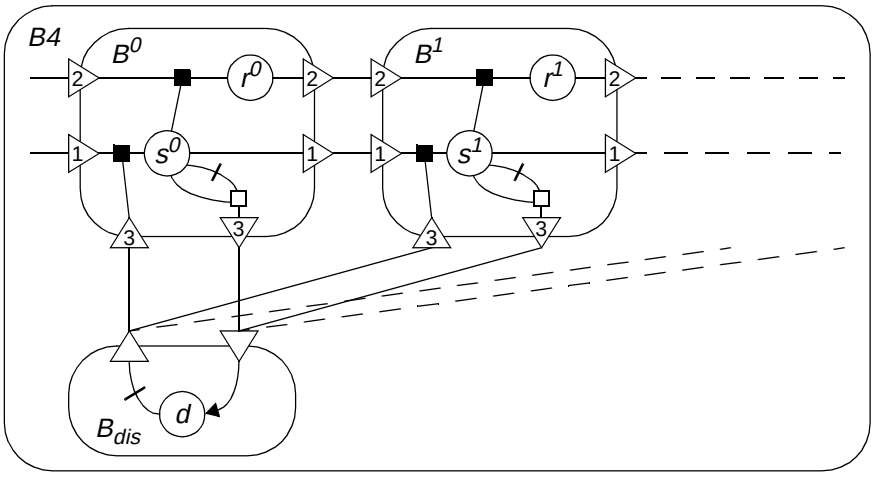


Figure 9.8: Behaviour *B4*

instance s^i. In contrast, if d depends only on the occurrences of s^i, i.e., $\wedge_i s^i \rightarrow d$, d would never occur since its condition would never be true.

9.3 Formal definition

The execution semantics of some causality-oriented structured behaviour definition B_{caus}, is defined by $[\![B_{mono}]\!]$, where B_{mono} is the corresponding monolithic definition of B_{caus}. The monolithic definition is obtained by (i) completely defining all behaviour instances in B_{caus}, and (ii) removing all exits and entries from B_{caus}.

This strategy to define the execution semantics of B_{caus} can not be applied, however, in case of large behaviour definitions that contain many (possibly infinitely) behaviour instances. In this case, we have to exploit the causality-oriented structure of B_{caus} and the compositionality of function $[\![\ ]\!]$. This is illustrated with two examples below.

Example: unconfirmed service (2)

Consider behaviour *B1* in Section 9.2.2. This behaviour defines the sequential composition of infinitely many instances of behaviour *B*. The execution semantics of *B1* can be defined as follows:

$$[\![B1]\!] = [\![B^0{}_{mono}]\!] \otimes [\![B^1{}_{mono}]\!] \otimes [\![B^2{}_{mono}]\!] \otimes \ldots$$
$$= \otimes_i [\![B^i{}_{mono}]\!] \ ,$$

where $B^i{}_{mono}$ is obtained from B^i through replacing each exit point by its associated causality condition, and removing the causality relations of exits and entries, i.e.:

$$B^0{}_{mono} = \{ \surd \rightarrow s^0, s^0 \rightarrow r^0 \},$$
$$B^i{}_{mono} = \{ s^{i-1} \rightarrow s^i, s^i \wedge r^{i-1} \rightarrow r^i \}, \text{ with } i > 0.$$

The decomposition of $[\![B1]\!]$ presented above is possible, because each behaviour instance B^i depends only on the preceding instances B^j ($j < i$) via one-sided enabling relations, and

does not depend on any following instances B^k ($k > i$). This decomposition allows a step-wise analysis of the semantics of *B1*, in which each step considers a subsequent behaviour instance of *B*, such that the execution semantics of the preceding instances can be reused.

A similar decomposition is possible for the execution semantics of behaviour *B3* in Section 9.2.2, which models the unconfirmed service with limited capacity.

Example: repetitive causality condition

Consider behaviour *B4* in Section 9.2.2. This behaviour also defines the sequential composition of an infinite number of instances of behaviour *B*, but in addition defines the possible disabling of these instances by behaviour instance B_{dis}. The execution semantics of *B4* can be defined as follows:

$$\begin{aligned} [\![B1]\!] &= [\![B_{dis\text{-}mono}]\!] \otimes [\![B^0{}_{mono}]\!] \otimes [\![B^1{}_{mono}]\!] \otimes [\![B^2{}_{mono}]\!] \otimes .... \\ &= [\![B_{dis\text{-}mono}]\!] \otimes \otimes_i [\![B^i{}_{mono}]\!] \ , \end{aligned}$$

where $B^i{}_{mono}$ and $B_{dis\text{-}mono}$ are defined as follows:

$$\begin{aligned} B^0{}_{mono} &= \{ \neg d \rightarrow s^0, s^0 \rightarrow r^0 \}, \\ B^i{}_{mono} &= \{ s^{i-1} \wedge \neg d \rightarrow s^i, s^i \wedge r^{i-1} \rightarrow r^i \}, \text{ with } i > 0, \\ B_{dis\text{-}mono} &= \{ \wedge_i (s^i \vee \neg s^i) \rightarrow d \}. \end{aligned}$$

Although the decomposition above allows a step-wise analysis of the semantics of behaviour *B4*, the execution semantics of $B_{dis\text{-}mono}$ has to be recalculated each time an additional instance of *B* is considered. However, intermediate results in the calculation of $[\![B_{dis\text{-}mono}]\!]$ can be reused.

The development of efficient strategies to determine the execution semantics of (repetitive) behaviours is left for further study.

9.4 Conclusions

In this chapter we extend our basic design language with the causality-oriented structuring technique defined in [16]. This technique allows the structuring of a behaviour as the composition of multiple smaller sub-behaviours, and their relationships. In order to support the modelling of (infinitely) repetitive behaviours, we have extended this technique with the notions of behaviour type and behaviour instantiation. Through behaviour instantiation one can (dynamically) create multiple instances of a single behaviour type definition. Furthermore, we have illustrated how the causality-oriented structure of a behaviour can be exploited to define the formal semantics of behaviours using repetitive behaviour instantiation.

Shorthand notations

In the examples of Section 9.2.2, behaviour instances are explicitly labelled with unique instance identifiers. In the example of the unconfirmed service with capacity *M*, this explicit

labelling is necessary and convenient, since it allows one to relate behaviour instance B^i to behaviour instances B^{i-1} and B^{i-M}.

However, in the other examples, behaviour instance B^i only depends on the preceding instance B^{i-1}. This represents a common situation in the modelling of repetitive (recursive) behaviours. For these (and other) common situations, shorthand notations can be defined, which define certain relationships between behaviour instances implicitly. In this way, a designer is not bothered with explicitly labelling and relating behaviour instances.

Although this chapter provides the conceptual means to model causality-oriented structured behaviours, including repetitive behaviours, more work is needed to determine how these means can be made available to designers in terms of suitable and easy-to-use notational elements. For example, the graphical notation of repetitive behaviours needs to be simplified, in particular for these common situations.

Chapter 10

Case study: OSI Connection-oriented Transport Service

This chapter applies our design model to the modelling of the OSI Connection-oriented Transport Service. This service is modelled in three subsequent steps: (i) the modelling of a single transport connection, (ii) the modelling of multiple concurrent transport connections, and (iii) the modelling of quality of service characteristics.

The structure of this chapter is as follows. Section 10.1 gives an overview of the transport service and discusses our approach to model this service. Sections 10.2, 10.3 and 10.4 present a model of the connection establishment phase, the connection release phase and the data transfer phase, respectively. Section 10.5 combines these models into a model of a single transport connection. Section 10.6 extends this model towards a model of the transport service supporting multiple transport connections. Section 10.7 adds the modelling of quality of service characteristics. Section 10.8 presents the conclusions.

10.1 Introduction

The OSI Connection-oriented Transport Service is provided by the transport layer of the OSI Basic Reference Model [31]. The informal definition of this service is published as an international standard in [32]. A formal definition of this service, using the formal specification language LOTOS [71, 33], has been published as a technical report in [34].

The OSI Connection-oriented Transport Service (TS) is briefly denoted as transport service or TS in the sequel. We assume the reader is acquainted with OSI concepts and terminology.

10.1.1 Transport service characteristics

The transport service supports the transfer of data units between transport service users. The general characteristics of this service are:

- *data transparency*: the transport service neither interprets nor constrains the contents of data units;
- *connection-oriented*: data transfer is preceded by the establishment of a connection and is ended by the unconditional release of this connection. Correspondingly, the service

behaviour is divided into three subsequent phases: a connection establishment phase, a data transfer phase and a release phase;

- *point-to-point*: a transport connection is established between two transport service users. Multiple connections can be established between the same or between distinct pairs of transport service users;
- *end-to-end addressing*: the calling user provides the address of the Transport Service Access Point (TSAP) of the called user, when requesting the establishment of a connection;
- *flow control by backpressure*: the rate at which the receiving user accepts data units may control the rate at which the sending user is allowed to submit data units;
- *expedited data transfer*: a user may send expedited data units, which have precedence over normal data units. Expedited data units may bypass previously sent normal data units, with the minimal guarantee that an expedited data unit is not delivered after a normal data unit submitted after the expedited data unit;
- *QoS selection*: the calling and called user may negotiate about the quality of the transport connection by selecting the values of the Quality of Service (QoS) parameters.

10.1.2 Transport service primitives

The transport service defines the interactions between the transport service users and the transport service provider, and the relationships between these interactions. These interactions are called *Transport Service Primitives* (TSPs). The transport service primitives are modelled in this chapter as actions, which implies that we do not distinguish between the interaction contributions of the service users and the service provider. The relationships between service primitives are modelled in terms of causality relations.

The action definitions of the TSPs are presented below.

Connection establishment

The connection establishment phase has four service primitives: *T-Connect request*, *T-Connect indication*, *T-Connect response* and *T-Connect confirm*.

The *T-Connect request* primitive models a request of the calling user to establish a transport connection to the called user. This primitive is represented by action *Creq*, which is defined as follows:

```
Creq (  ι : ⟨  cld  : TSAP,                        # called address
               clg  : TSAP,                        # calling address
               exp  : Bool,                        # expedited data option
               qos  : QoS,                         # quality of service
               usr  : UserData [Len(usr) ≤ 32]     # user data
           ⟩
        τ : Time,
        λ : ⟨  tsap : TSAP,                    # tsap address
```

```
            tcei   : TCEI,                    # transport connection endpoint identifier
        〉  ).
```

The information attribute of *Creq* is defined as a five tuple, which represents the parameters of the *T-Connect request* primitive:

- called address *cld*, which defines the TSAP address of the called user;
- calling address *clg*, which defines the TSAP address of the calling user;
- expedited data option *exp*, which defines whether expedited data transfer should be possible (value *True*) or not (value *False*);
- quality of service *qos*, which defines a list of QoS parameters. These parameters are explained in Section 10.7;
- TS-user data *usr*, which defines the data unit that is to be transferred to the remote user. This data unit can be at most 32 octets long, which is represented by the constraint [*Len*(*usr*) ≤ 32]. We assume function *Len* yields the length of the user data (element of *UserData*).

The parameters above are denoted as ι.*cld*, ι.*clg*, ι.*exp*, ι.*qos* and ι.*usr*, respectively. The data types of these parameters are not fully elaborated here. We assume these data types can be constructed from standard data types, such as boolean, natural number and string. For example, TSAP addresses can be represented by natural numbers and user data can be represented by strings of characters or octets.

The time attribute value of *Creq* is defined as an element of data type *Time*, which is a synonym for domain ***T***. The location attribute of *Creq* is defined as a tuple consisting of a TSAP address and a Transport Connection Endpoint Identifier (TCEI), denoted as λ.*tsap* and λ.*tcei*, respectively. A TCEI is used to uniquely identify distinct connections at a single TSAP. We assume that data type *TCEI* can be constructed from the natural number data type, similarly to data type *TSAP*.

The *T-Connect indication* primitive models the indication of the connect request to the called user. This primitive is represented by action *Cind*. Since this primitive has the same parameters as the *T-Connect request* primitive, the definitions of actions *Cind* and *Creq* are identical, except for their names.

The *T-Connect response* primitive models the acceptance of the requested transport connection by the called user. This primitive is represented by action *Crsp*, which is defined as follows:

```
Crsp (  ι : 〈  qos   : QoS,                      # quality of service
               rsp   : TSAP,                     # responding address
               exp   : Bool,                     # expedited data option
               usr   : UserData [Len(usr) ≤ 32]  # user data
            〉
        τ : Time,
        λ : 〈  tsap  : TSAP,                     # tsap address
               tcei  : TCEI                      # transport connection endpoint identifier
```

$\rangle$).

The information sub-attribute $\iota.rsp$ represents the responding address parameter of the *T-Connect response* primitive, which defines the address of the TSAP to which the transport connection has been established.

The *T-Connect confirm* primitive models the indication of the connect response to the calling user. This primitive is represented by action *Ccnf*. Since this primitive has the same parameters as the *T-Connect response* primitive, the definitions of actions *Ccnf* and *Crsp* are identical, except for their names.

Data transfer

The data transfer phase has four service primitives: *T-Data request*, *T-Data indication*, *T-Expedited-Data request* and *T-Expedited-Data indication*.

The *T-Data request* and *T-Data indication* primitives model the sending and delivery of normal data units, respectively. These primitives are represented by actions *Dreq* and *Dind*, which are defined as follows:

$$
\begin{array}{lll}
Dreq\ (\ \iota : UserData, & & \\
\quad \tau : Time, & & \\
\quad \lambda : \langle\ tsap & : TSAP, & \\
\qquad\quad tcei & : TCEI & \\
\quad \rangle\ ) & & \\
\end{array}
\qquad
\begin{array}{lll}
Dind\ (\ \iota : UserData & & \\
\quad \tau : Time, & & \\
\quad \lambda : \langle\ tsap & : TSAP, & \\
\qquad\quad tcei & : TCEI & \\
\quad \rangle\ ). & & \\
\end{array}
$$

The *T-Expedited-Data request* and *T-Expedited-Data indication* primitives model the sending and delivery of expedited data units. These primitives are represented by actions *Ereq* and *Eind*, respectively. Since these primitives have the same parameters as the *T-Data request* and *T-Data indication* primitives, the definitions of actions *Ereq* and *Eind* and the definitions of actions *Dreq* and *Dind* are identical, except for their names.

Connection release

The connection release phase consists of two service primitives: *T-Disconnect request* and *T-Disconnect indication*.

The *T-Disconnect request* primitive models a request for the unconditional release of a transport connection by some transport user. This primitive is represented by action *Rreq*, which is defined as follows:

$$
\begin{array}{lll}
Rreq\ (\ \iota : UserData\ [Len(usr) \leq 64], & & \\
\quad \tau : Time, & & \\
\quad \lambda : \langle\ tsap & : TSAP, & \\
\qquad\quad tcei & : TCEI & \\
\quad \rangle\ ). & & \\
\end{array}
$$

The *T-Disconnect indication* primitive models the indication of connection release to some user. This primitive is represented by action *Rind*, which is defined as follows:

$$
\begin{array}{lll}
\mathit{Rind}\ (\ \iota : \langle & \mathit{rsn} & : \mathit{Reason}, \\
 & \mathit{usr} & : \mathit{UserData} \\
\quad \rangle & & \\
\quad \tau : \mathit{Time}, & & \\
\quad \lambda : \langle & \mathit{tsap} & : \mathit{TSAP}, \\
 & \mathit{tcei} & : \mathit{TCEI} \\
\quad \rangle & & \ ).
\end{array}
$$

The information sub-attribute $\iota.\mathit{rns}$ represents the reason for the *T-Disconnect indication* primitive. One of the following reasons may be defined:

- *user invoked*, which defines that the connection release is requested by the remote user. This reason is represented by value *user*;
- *provider invoked*, which defines that the transport provider has initiated the connection release, e.g., because the provider can not maintain the current connection or is not able to establish a new connection. This reason is represented by value *prov*.

10.1.3 Transport service model

The development of a transport service model is divided into the modelling of the functional aspects of the transport service and the modelling of the QoS aspects of the transport service. Functional aspects are related to the modelling of (inter)actions and their relationships, without quantifying time and probability aspects. QoS aspects are related to the modelling of quality of service (performance) characteristics, requiring the quantification of time and probability aspects.

The modelling of the functional aspects of the transport service comprises the modelling of the service primitives, including the establishment of their information and location attributes, and the modelling of the temporal relations between these service primitives, including the reference relations between their information and location attributes. The uncertainty of service primitives is modelled using the uncertainty attribute, where we assume that uncertainty associations get the value *must* by default.

The functional aspects of the transport service are further divided into the functional aspects of a single transport connection and the functional aspects of the coordination between multiple transport connections. First, we develop a model of a single Transport Connection (TC), which is decomposed into sub-models of the connection establishment, data transfer and connection release phases. Subsequently, we model the administration of TCEIs at a single TSAP, which allows the establishment of multiple concurrent transport connections.

The QoS aspects of the transport service comprise the modelling of the QoS characteristics as defined by the QoS parameters established in the *T-Connect* primitives. These QoS characteristics are modelled by means of constraints on the time attribute and the (extended) integral probability attribute of the service primitives.

Figure 10.1 depicts the decomposition of the transport service model used in this chapter, and indicates the sections where the various parts of this model are presented.

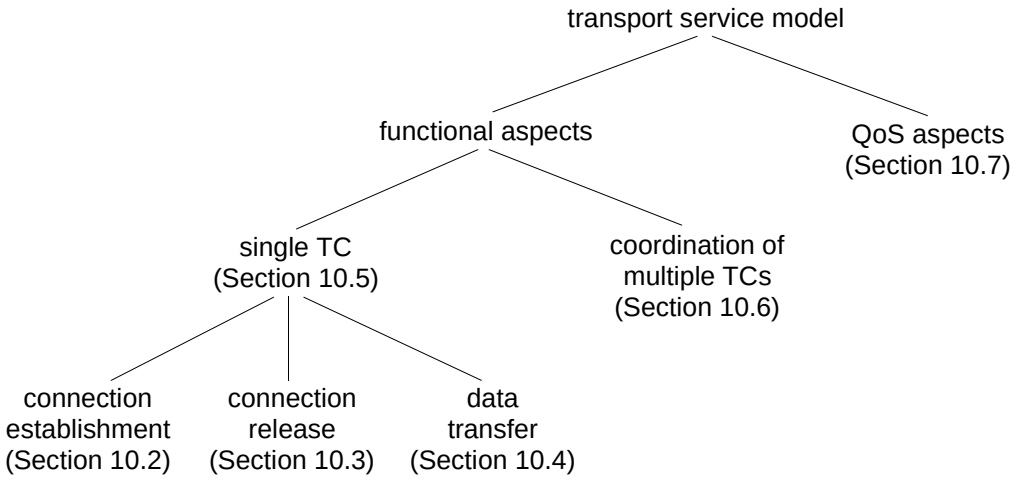


Figure 10.1: Decomposition of transport service model

10.1.4 Notational conventions

We use the following conventions for the textual representation of behaviour definitions:

- a behaviour definition begins and ends with the keywords **behaviour** and **endbehaviour**, which replace the set symbols { and }, respectively;
- a causality-oriented behaviour definition is divided into the following parts:
  - (i) an *entry part*, which defines the entry points of the behaviour and their associated parameter lists. This part is preceded by the keyword **entries**;
  - (ii) an *action part*, which defines the causality relations of the actions of the behaviour. This part is preceded by the keyword **actions**;
  - (iii) an *exit part*, which defines the causality relations of the exit points of the behaviour. This part is preceded by the keyword **exits**;
  - (iv) an *exit/entry part*, which defines the association of causality conditions with entry points, either directly or indirectly via exit points. This part is preceded by the keyword **exits/entries**;
  - (v) a *where part*, which contains the sub-behaviour definitions. This part is preceded by the keyword **where**.

  The parts above can be defined in any order;
- for brevity, the attribute list of an action is not represented, since it has been defined in Section 10.1.2. Instead, the existence of such a list is denoted by the symbol “( )”;
- comments are preceded by the symbol #.

10.2 Connection establishment

The Connection Establishment (CE) phase consists of the sequential composition of the *T-Connect* primitives. The behaviour of the CE phase is defined below.

behaviour *CE* =

entries

entry1	# *start* condition
entry2	# $\neg Dreq \wedge \neg Dind$ (called side)
entry3	# $\neg Dreq \wedge \neg Dind$ (calling side)

actions

entry1	$\rightarrow Creq\ (\ ),$
Creq [υ = ?]	$\rightarrow Cind\ (\ )\ [\iota.cld = \iota_{Creq}.cld,\ \iota.clg = \iota_{Creq}.clg,$ $\iota.exp = \iota_{Creq}.exp,\ \iota.qos \leq \iota_{Creq}.qos,$ $\iota.usr = \iota_{Creq}.usr,\ \lambda.tsap = \iota_{Creq}.cld\ ]$
$Cind \wedge entry2$	$\rightarrow Crsp\ (\ )\ [\iota.qos \leq \iota_{Cind}.qos,\ \iota.rsp = \lambda.tsap,$ $\iota.exp \Rightarrow \iota_{Cind}.exp,\ \lambda = \lambda_{Cind}\ ],$
$Crsp \wedge entry3$	$\rightarrow Ccnf\ (\ )\ [\iota.qos = \iota_{Cind}.qos,\ \iota.rsp = \iota_{Crsp}.rsp,$ $\iota.exp = \iota_{Crsp}.exp,\ \iota.usr = \iota_{Cind}.usr,$ $\lambda = \lambda_{Creq}\ ]$

exits

Creq	$\rightarrow exit1(\lambda_{Creq}.tsap),$
Cind	$\rightarrow exit2(\lambda_{Cind}.tsap),$
Crsp	$\rightarrow exit3(\lambda_{Crsp}.tsap,\ \iota_{Crsp}.exp,\ \iota_{Crsp}.qos),$
Ccnf	$\rightarrow exit4(\lambda_{Ccnf}.tsap,\ \iota_{Ccnf}.exp,\ \iota_{Ccnf}.qos),$
$\neg Crsp$	$\rightarrow exit5,$
$\neg Ccnf$	$\rightarrow exit6$

endbehaviour

Action *Creq* depends on the causality condition represented by *entry1*, which is assumed to be the start condition. The information attribute of *Creq* establishes the following SP parameter values: the calling and called user addresses, the value of the expedited data option, the QoS parameter values and some user data that is to be transferred to the called user. In principle, any SP parameter values from their corresponding domains are allowed.

Action *Cind* is enabled by *Creq*. The SP parameter values established in *Cind* and *Creq* are the same, except for the QoS parameter. The QoS requested by the calling user may be lowered or kept the same by the transport service provider (except for the QoS parameter TC protection, which should be kept the same). The possible lowering of the requested QoS is represented by operator ≤. The TSAP address of the location at which *Cind* occurs is determined by the called address parameter. The occurrence of action *Cind* is uncertain, which is represented by associating the *may* value with enabling condition *Creq*. This uncertainty models the possible inability of the transport service provider to establish a connection. In this case, a provider initiated disconnect occurs at the calling user side, provided that the calling user has not issued a disconnect request before. These situations are modelled in Section 10.3.

Action *Crsp* is enabled by *Cind*. The information attribute of *Crsp* establishes the following SP parameter values: the QoS parameter values accepted by the called user, the responding

user address, the expedited data option value and some user data that is to be transferred to the calling user. The called user may lower the QoS established in *Cind* (except for the QoS parameter TC protection), or keep it the same. The responding address is equal to the called address. The expedited data option can only be selected by the called user if it has been selected by the calling user; the called user is always allowed to de-select this option. Actions *Crsp* and *Cind* occur at the same location.

Action *Ccnf* is enabled by *Crsp*. The SP parameter values established in *Ccnf* and *Crsp* are the same. Action *Ccnf* occurs at the same location as *Creq*.

Actions *Crsp* and *Ccnf* must occur, unless they are disabled by the occurrence of a disconnect request or disconnect indication at the calling and called user side, respectively. These disabling conditions are represented by *entry2* and *entry3*, respectively.

Behaviour *CE* also defines six exit points. These exit points are used to define dependencies between the CE phase and the connection release and data transfer phases.

Figure 10.2 shows the graphical representation of behaviour *CE*.

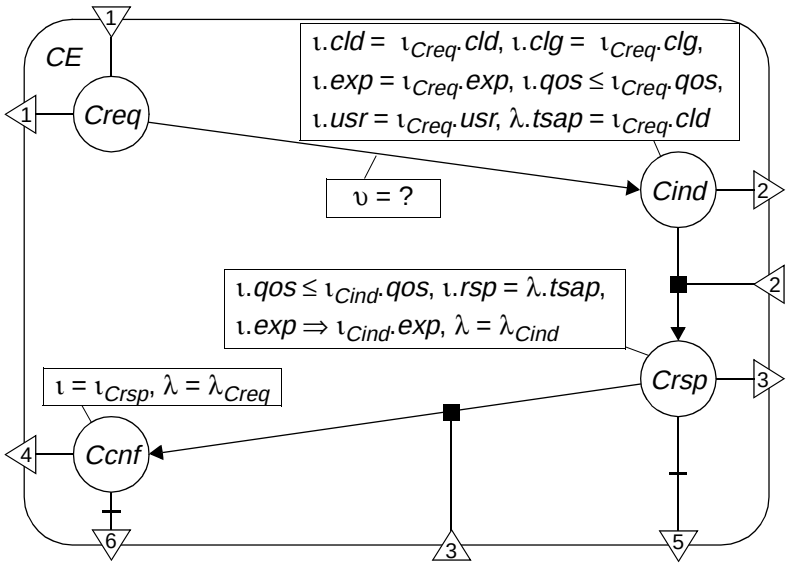
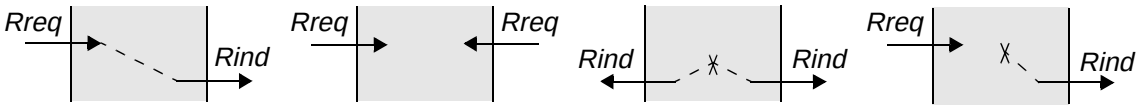


Figure 10.2: CE phase

10.3 Connection release

The Connection Release (CR) phase consists of the occurrence of either a disconnect request or a disconnect indication at the calling and called user side. Figure 10.3 depicts some scenario's of a connection release in terms of time-sequence diagrams.

The behaviour of the CR phase is decomposed into two related sub-behaviours, each of them modelling the occurrence of a disconnect request or disconnect indication at one of the sides of the transport connection. These sub-behaviours are defined as distinct instances of the same behaviour type, which is called *CR-half*. The relationships between the actions in



(i) invoked by one user (ii) invoked by both users (iii) invoked by provider (iv) invoked by user and provider

Figure 10.3: Connection release scenario's

both sub-behaviours are modelled via exit and entry points. Behaviour *CR-half* is defined below.

behaviour *CR-half* =

entries

entry1(v_{cl} : ⟨*TSAP*, *TCEI*⟩) # *Creq* / *Cind*
entry2(v_{cl} : ⟨*TSAP*, *TCEI*⟩, v_{exp} : *Bool*, v_{qos} : *QoS*) # *Ccnf* / *Crsp*
entry3 # ¬*Ccnf* / ¬*Crsp*
entry4(v_{usr} : *UserData*) # remote *Rreq*
entry5 # remote *Rind*
entry6 # disabling condition from data primitives at calling / called side

actions

entry1 ∧ *entry6* ∧ ¬*Rind* → *Rreq* ( ) [λ = v_{cl}],

alternative conditions of *Rind* satisfied during connection establishment
(*entry1* ∧ *entry3* ∧ *entry6* ∧ ¬*Rreq* ∧
(√ # 1: inability to establish connection
∨ *entry4* # 2: remote user invoked disconnect
∨ *entry5* # 3: (remote) provider invoked disconnect
)
alternative conditions of *Rind* satisfied after connection establishment
∨ (*entry2* ∧ *entry6* ∧ ¬*Rreq* ∧
(√ # 4: provider invoked disconnect
∨ *entry4* # 5: remote user invoked disconnect
∨ *entry5* # 6: (remote) provider invoked disconnect
)
→ *Rind* ( ) [if @1, @3, @4, @6 :
ι.*rsn* = *prov*, ι.*usr* = *undefined*,
if @2, @5:
ι.*rsn* = *user*, ι.*usr* = v_{usr},
λ = v_{cl}]

exits

¬*Rreq* ∧ ¬*Rind* → *exit1*,
Rreq → *exit2*(ι_{Rreq}.*usr*),
Rind → *exit3*

endbehaviour

Action *Rreq* is allowed to occur when *entry1* and *entry6* are satisfied and action *Rind* has not occurred yet. Entry point *entry1* either represents enabling condition *Creq* in case *CR-half* models a connection release at the calling side, or represents enabling condition *Cind* in case *CR-half* models a connection release at the called side. In order to simplify the discussion, we assume that *CR-half* models a connection release at the calling side in the sequel. In this case, *entry1* models that a connection release is not possible before a connect request at the calling side. The location of *Rreq* is the same as the location of *Creq*, which is passed via the parameter list of *entry1*.

Entry point *entry6* represents the condition that action *Rreq* at the calling side may not occur simultaneously with normal or expedited data primitives at this side. This condition originates from the requirement that the occurrence of *Rreq* at the calling side disables the occurrence of any new data primitive instance at this side. The corresponding (two-sided) disabling relation defines that action *Rreq* depends on the following condition:

$$\wedge_i(Dreq^i \vee \neg Dreq^i) \wedge \wedge_i(Dind^i \vee \neg Dind^i) \wedge \wedge_i(Ereq^i \vee \neg Ereq^i) \wedge \wedge_i(Eind^i \vee \neg Eind^i),$$

where we assume infinitely many instances of the data primitives can occur if the connection is never released. The location attributes of *Rreq* and of all instances of *Dreq*, *Dind*, *Ereq* and *Eind* are assumed to be the same. Action instances are identified by means of natural numbers.

The condition associated with *entry6* is represented by an exit point of the data transfer phase behaviour definition, which is presented in Section 10.4.

The causality condition of action *Rind* is much more complicated. The reason for this is twofold:

- the disconnect primitives are used to model the release of a connection during connection establishment as well as after connection establishment;
- a connection release may be initiated by the transport users, the transport service provider, or both.

Alternative causality conditions that allow action *Rind* to occur during connection establishment are characterized by sub-condition $entry1 \wedge entry3 \wedge entry6 \wedge \neg Rreq$. Entry point *entry3* represents that action *Ccnf* has not occurred yet. In conjunction with $entry1 \wedge entry3 \wedge entry6 \wedge \neg Rreq$, the following alternative sub-conditions are defined:

1. the start condition, which represents that no additional condition is defined. Alternative condition $entry1 \wedge entry3 \wedge entry6 \wedge \neg Rreq$ represents the possibility that the provider rejects the establishment of a connection, e.g., due to resource limitations;
2. *entry4*, which represents that action *Rreq* has occurred at the called side. Consequently, alternative condition $entry1 \wedge entry3 \wedge entry6 \wedge \neg Rreq \wedge entry4$ models the possibility that the called user rejects the establishment of a connection;
3. *entry5*, which represents that action *Rind* has occurred at the called side. *Rind* has been surely invoked by the provider, otherwise a *Rreq* must have occurred at the calling side. Consequently, alternative condition $entry1 \wedge entry3 \wedge entry6 \wedge \neg Rreq \wedge entry5$ models the possibility that the provider rejects the establishment of a connection, and that a *Rind* has already happened at the called side.

Alternative causality conditions that allow action *Rind* to occur after connection establishment are characterized by sub-condition $entry2 \wedge entry6 \wedge \neg Rreq$. Entry point *entry2* represents that action *Ccnf* has occurred. In conjunction with $entry2 \wedge entry6 \wedge \neg Rreq$, the same alternative sub-conditions are defined as the ones presented above. They only differ in the fact that *Rind* in this case occurs after instead of during connection establishment.

The causality conditions of *Rind* defined above are numbered 1 to 6 in behaviour definition *CR-half*. Each of these causality conditions defines infinitely many alternative causality conditions, due to the disabling condition represented by *entry6*. In case one of the alternative conditions in 1, 3, 4 or 6 causes the occurrence of *Rind*, the reason parameter of *Rind* must be equal to the value *prov*, and the user data parameter is undefined. In case one of the alternative conditions in 2 and 5 causes the occurrence of *Rind*, the reason parameter of *Rind* must be equal to the value *user*, and the user data parameter must be equal to the corresponding parameter established in the preceding *Rreq*. The condition that one of the alternative causality conditions in the *i*-th specified causality condition causes the occurrence of the result action, i.e., is the resulting causality condition, is represented as $@i$.

The discussion above equally applies to the case in which *CR-half* models a connection release at the called side, when reading ‘called’ instead of ‘calling’, ‘*Cind*’ instead of ‘*Creq*’, ‘*Crsp*’ instead of ‘*Ccnf*’, and vice versa.

The behaviour of the entire release phase is defined below.

behaviour *CR* =

entries

$entry1(v_{cl} : \langle TSAP, TCEI \rangle)$	# *Creq*
$entry2(v_{cl} : \langle TSAP, TCEI \rangle)$	# *Cind*
$entry3(v_{cl} : \langle TSAP, TCEI \rangle, v_{exp} : Bool, v_{qos} : QoS)$	# *Crsp*
$entry4(v_{cl} : \langle TSAP, TCEI \rangle, v_{exp} : Bool, v_{qos} : QoS)$	# *Ccnf*
entry5	# $\neg Crsp$
entry6	# $\neg Ccnf$
entry7	# disabling condition for data primitives at calling side
entry8	# disabling condition for data primitives at called side

exit/entries

# instance of *CR-half* at calling side	
entry1	$\rightarrow CR\text{-}half^1.entry1,$
entry4	$\rightarrow CR\text{-}half^1.entry2,$
entry6	$\rightarrow CR\text{-}half^1.entry3,$
$CR\text{-}half^2.exit2$	$\rightarrow CR\text{-}half^1.entry4,$
$CR\text{-}half^2.exit3$	$\rightarrow CR\text{-}half^1.entry5,$
entry7	$\rightarrow CR\text{-}half^1.entry6,$
# instance of *CR-half* at called side	
entry2	$\rightarrow CR\text{-}half^2.entry1,$
entry3	$\rightarrow CR\text{-}half^2.entry2,$
entry5	$\rightarrow CR\text{-}half^2.entry3,$
$CR\text{-}half^1.exit2$	$\rightarrow CR\text{-}half^2.entry4,$

CR-half^1*.exit3* → *CR-half*^{2}*.entry5*,
entry8 → *CR-half*^{2}*.entry6*

exits

CR-half^1*.exit1* → *exit1*, # $\neg$*Rreq* $\wedge$ $\neg$*Rind* calling side
CR-half^2*.exit1* → *exit2* # $\neg$*Rreq* $\wedge$ $\neg$*Rind* called side

where

behaviour *CR-half* = # see above # **endbehaviour**

endbehaviour

Exit points *CR.exit1* and *CR.exit2* represent the disabling conditions for the connect primitives *Crsp* and *Ccnf* and for the data primitives at the calling and called side, respectively.

Figure 10.4 shows the graphical representation of behaviour *CR*, without action attributes.

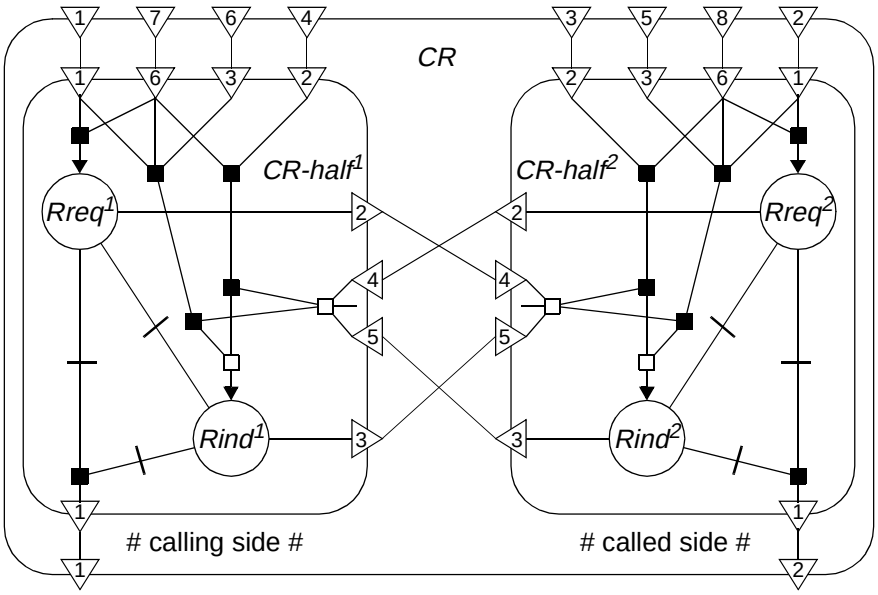


Figure 10.4: CR phase

Provider invoked connection reject

Figure 10.5 depicts a specific scenario of a connection release in which the TS provider rejects the connection after a connect request (*Creq*), which is indicated to the calling user by means of a disconnect indication (*Rind*). In this scenario, the TS provider blocks the connect indication at the called user side.

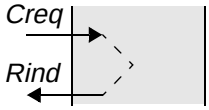


Figure 10.5: Provider invoked connection reject

This scenario is supported by our models of the *CE* and *CR* phases. The possible rejection of a connection by the TS provider and the blocking of the corresponding connect indication is modelled by the *may* uncertainty value associated with the causality condition of action *Cind* in behaviour *CE* of Section 10.2. The non-occurrence of *Cind* implies the non-occurrences of *Crsp* and *Ccnf* in behaviour *CE*, and implies that either *Rreq* or *Rind* occurs at the calling user side in behaviour *CR*.

10.4 Data transfer

The Data Transfer (DT) phase consists of the bi-directional transfer of normal and expedited data units. The transfer of a data unit is modelled by a data request at the sending user side, in which the data unit to be transferred is established, followed by a data indication at the receiving user side, in which the same data unit is established.

The behaviour of the DT phase is decomposed into two independent sub-behaviours, which model data transfer in one direction, i.e., either from the calling to the called user, or vice versa. These sub-behaviours are defined as distinct instances of the same behaviour type, which is called *DT-half*. Behaviour *DT-half* is defined below.

behaviour *DT-half* =

entries

$entry1(v_{snd}, v_{rcv}: \langle TSAP, TCEI\rangle, v_{exp} : Bool, v_{qos} : QoS)$ # *Ccnf* / *Crsp*
entry2 # *Crsp* / *Ccnf*
entry3 # $\neg Rreq \wedge \neg Rind$ (sending side)
entry4 # $\neg Rreq \wedge \neg Rind$ (receiving side)

exits/entries

$entry1 \rightarrow DT\text{-}inst^{1}.entry1(v_{snd}, v_{rcv}, v_{exp}, v_{qos}),$
$entry2 \rightarrow DT\text{-}inst^{1}.entry2,$
$entry2 \rightarrow DT\text{-}inst^{1}.entry3,$
$i > 1 \mid DT\text{-}inst^{i-1}.exit1 \rightarrow DT\text{-}inst^{i}.entry1(v_{snd}, v_{rcv}, v_{exp}, v_{qos}),$
$i > 1 \mid DT\text{-}inst^{i-1}.exit2 \rightarrow DT\text{-}inst^{i}.entry2,$
$i > 1 \mid DT\text{-}inst^{i-1}.exit3 \rightarrow DT\text{-}inst^{i}.entry3,$

$i \geq 1 \mid entry3 \rightarrow DT\text{-}inst^{i}.entry4,$
$i \geq 1 \mid entry4 \rightarrow DT\text{-}inst^{i}.entry5,$

exits

$\wedge_{i \geq 1}\ DT\text{-}inst^{i}.exit4 \rightarrow exit1,$
$\wedge_{i \geq 1}\ DT\text{-}inst^{i}.exit5 \rightarrow exit2$

where

behaviour *DT-inst* = # see below # **endbehaviour**

endbehaviour

Behaviour *DT-half* defines the sequential composition of infinitely many instances of sub-behaviour *DT-inst*. Each instance of *DT-inst* models the transfer of a single data unit from the sending side to the receiving side. Figure 10.6 depicts the structure of behaviour *DT-half*.

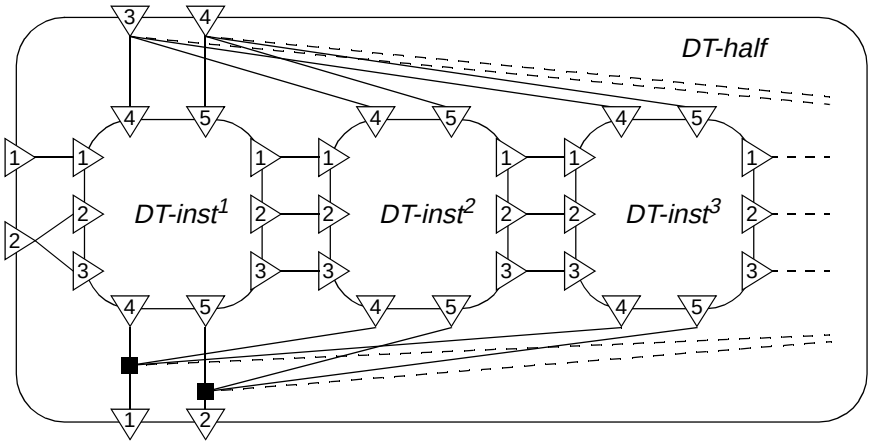


Figure 10.6: DT phase: one direction

Behaviour *DT-inst* is defined below.

behaviour *DT-inst* =

entries

$entry1(v_{snd}, v_{rcv}: \langle TSAP, TCEI\rangle, v_{exp} : Bool, v_{qos} : QoS)$	# last *Dreq* or *Ereq*
entry2	# last *Dind* or *Eind*
entry3	# last *Eind*
entry4	# $\neg Rreq \wedge \neg Rind$ (sending side)
entry5	# $\neg Rreq \wedge \neg Rind$ (receiving side)

actions

$entry1 \wedge entry4 \wedge \neg Ereq$	$\rightarrow Dreq\ (\ )\ [\lambda = v_{snd}]$,
$entry1 \wedge entry4 \wedge \neg Dreq\ [v_{exp}]$	$\rightarrow Ereq\ (\ )\ [\lambda = v_{snd}]$,
$entry2 \wedge entry5 \wedge Dreq$	$\rightarrow Dind\ (\ )\ [\iota = \iota_{Dreq}, \lambda = v_{rcv}]$,
$entry3 \wedge entry5 \wedge Ereq$	$\rightarrow Eind\ (\ )\ [\iota = \iota_{Ereq}, \lambda = v_{rcv}]$

exits

$Dreq \vee Ereq$	$\rightarrow exit1(v_{snd}, v_{rcv}, v_{exp}, v_{qos})$,
$Dind \vee Eind$	$\rightarrow exit2$,
$(Dreq \wedge entry3) \vee Eind$	$\rightarrow exit3$,
$(Dreq \vee \neg Dreq) \wedge (Ereq \vee \neg Ereq)$	$\rightarrow exit4$,
$(Dind \vee \neg Dind) \wedge (Eind \vee \neg Eind)$	$\rightarrow exit5$

endbehaviour

In each subsequent instance of *DT-inst* a choice is made between a data request or an expedited-data request at the sending user side, which is followed by a data indication or an expedited-data indication at the receiving user side, respectively. We assume the action instances of behaviour instance $DT\text{-}inst^{id}$ inherit instance identifier *id*.

Instances of actions *Dreq* and *Ereq* must occur sequentially, such that action instance $Dreq^i$ or $Ereq^i$ is allowed to occur after either $Dreq^{i-1}$ or $Ereq^{i-1}$ has occurred. This condition is represented by entry point *entry1*, which can alternatively be defined in terms of its possible instances as follows:

$Crsp \rightarrow entry1^1$ or $Ccnf \rightarrow entry1^1$;
$i > 1 \mid Dreq^{i-1} \vee Eind^{i-1} \rightarrow entry1^i$.

The first instance of *entry1* corresponds to enabling condition *Ccnf* or enabling condition *Crsp*, depending on whether *DT-inst* models data transfer from the calling to the called user or from the called to the calling user, respectively. This condition is passed via entry point *entry1* of behaviour *DT-half*. In addition, *entry1* of both *DT-inst* and *DT-half* is associated with a parameter list that passes all relevant information established during connection establishment:

- v_{snd} and v_{rcv} defines the location values of the sending and receiving user side;
- v_{exp} defines whether the expedited data option is selected, or not; and
- v_{qos} defines the QoS parameter values that have been established.

Actions *Dreq* and *Ereq* depend on *entry4*, which represents the possible disabling of these actions by the occurrence of a disconnect primitive at the sending user side. This condition is passed via entry point *entry3* of behaviour *DT-half*. Furthermore, constraint $[v_{exp}]$ is associated with the causality condition of action *Ereq*. This constraint is true iff the expedited data option is selected.

Action instance $Dind^i$ is allowed to occur after instance $Dreq^i$ has occurred and after all preceding data indications that correspond to a normal or expedited data request that has occurred before $Dreq^i$ have occurred. The latter condition is represented by *entry2*, which can alternatively be defined in terms of its possible instances as follows:

$Crsp \rightarrow entry2^1$ or $Ccnf \rightarrow entry2^1$;
$i > 1 \mid Dind^{i-1} \vee Eind^{i-1} \rightarrow entry2^i$.

The first instance of *entry2* corresponds to enabling condition *Crsp* or enabling condition *Ccnf*, depending on whether *DT-half* models data transfer from the calling to the called user or from the called to the calling user, respectively. This condition is passed via entry point *entry2* of behaviour *DT-half*.

Action instance $Eind^i$ is allowed to occur after instance $Ereq^i$ has occurred and after all expedited data indications that correspond to an expedited data request that has occurred before $Ereq^i$ have occurred. The latter condition is represented by *entry3*, which can alternatively be defined in terms of its possible instances as follows:

$Crsp \rightarrow entry3^1$ or $Ccnf \rightarrow entry3^1$;
$i > 1 \mid \wedge_{1 \leq j \leq i-1} (Dreq^j \vee Eind^j) \rightarrow entry3^i$.

Entry point $entry3^i$ depends on condition $Dreq^j$, with $j < i$, in case in the j-th instance of *DT-inst* a normal data request occurs. The dependency on $Dreq^j$ is used in order to model that action $Eind^i$ may, but does not have to, occur before action $Dind^j$ occurs. In this way the

requirement w.r.t. the precedence relation between normal and expedited data transfer as described in Section 10.1 is satisfied. The dependency on $Dreq^j$ is not introduced just for this purpose, but $Eind^i$ also depends indirectly on $Dreq^j$ via $Ereq^i$.

Actions *Dind* and *Eind* depend on *entry5*, which represents the possible disabling of these actions by the occurrence of a disconnect primitive at the receiving user side. This condition is passed via entry point *entry4* of behaviour *DT-half*.

Figure 10.7 shows the graphical representation of behaviour *DT-inst*.

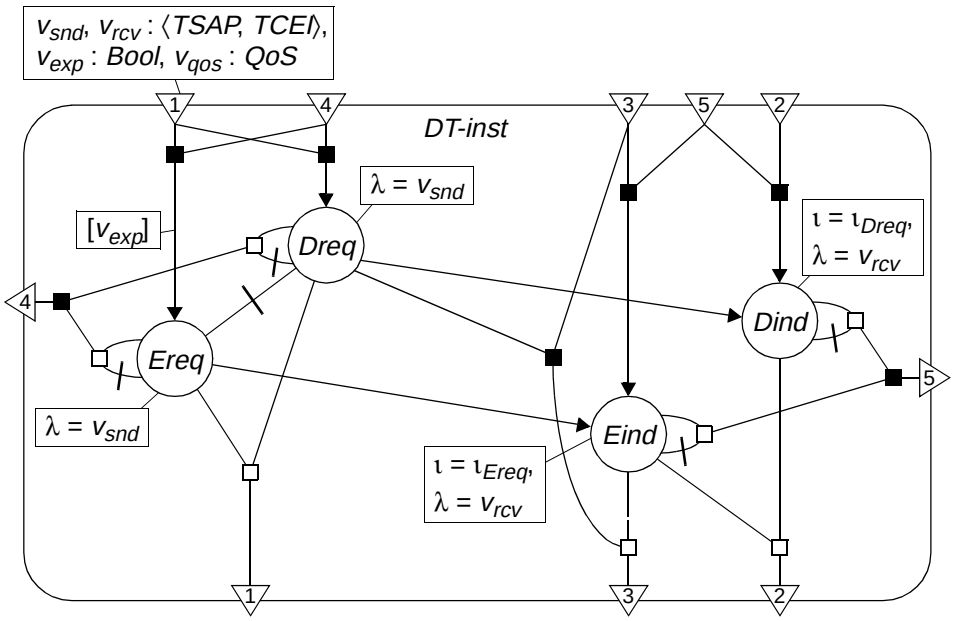


Figure 10.7: Instance of data transfer

The behaviour of the entire DT phase is defined as follows.

behaviour *DT* =

entries

entry1(v_{clg}, v_{cld}: ⟨*TSAP*, *TCEI*⟩, v_{exp} : *Bool*, v_{qos} : *QoS*) # *Crsp*
entry2(v_{clg}, v_{cld}: ⟨*TSAP*, *TCEI*⟩, v_{exp} : *Bool*, v_{qos} : *QoS*) # *Ccnf*
entry3 # $\neg Rreq \wedge \neg Rind$ (calling side)
entry4 # $\neg Rreq \wedge \neg Rind$ (called side)

exits/entries

data transfer from calling to called side
entry2 → $DT\text{-}half^1.entry1$,
entry1 → $DT\text{-}half^1.entry2$,
entry3 → $DT\text{-}half^1.entry3$,
entry4 → $DT\text{-}half^1.entry4$,
data transfer from called to calling side
entry1 → $DT\text{-}half^2.entry1$,
entry2 → $DT\text{-}half^2.entry2$,
entry4 → $DT\text{-}half^2.entry3$,

entry3 → *DT-half*^{2}*.entry4*,

exits

DT-half^1*.exit1* ∧ *DT-half*^{2}*.exit2* → *exit1*, # calling side
DT-half^1*.exit2* ∧ *DT-half*^{2}*.exit1* → *exit2* # called side

endbehaviour

Exit points *exit1* and *exit2* represent the conditions for the disconnect primitives, defining that an action is not allowed to occur simultaneously with one of the data primitives at the calling or called user side, respectively. This condition originates from the disabling relation between the disconnect primitives and the data primitives, which has been explained in Section 10.3.

In order to distinguish two instances *DT-inst*^{i} in *DT-half*^{1} and *DT-half*^{2}, instance identifier *i* can be extended with the instance identifier of *DT-half*, such that both instances are identified as *DT-inst*$^{\langle i,\,1\rangle}$ and *DT-inst*$^{\langle i,\,2\rangle}$, respectively. Whether this is needed or not depends on the scope rules that are being used.

10.5 Single transport connection

The behaviour of a single Transport Connection (TC) is composed from behaviour definitions *CE*, *CR* and *DT*, by combining their exit and entry points. This behaviour is defined below. We assume that the exit/entry combinations used in this behaviour need no further explanation.

behaviour *TC* =

entries

entry

exits/entries

entry conditions of *CE*
entry → *CE.entry1*,
CR.exit2 → *CE.entry2*, *CR.exit1* → *CE.entry3*,
entry conditions of *CR*
CE.exit1 → *CR.entry1*, *CE.exit2* → *CR.entry2*,
CE.exit3 → *CR.entry3*, *CE.exit4* → *CR.entry4*,
CE.exit5 → *CR.entry5*, *CE.exit6* → *CR.entry6*,
DT.exit1 → *CR.entry7*, *DT.exit2* → *CR.entry8*,
entry conditions of *DT*
CE.exit3 → *DT.entry1*, *CE.exit4* → *DT.entry2*,
CR.exit1 → *DT.entry3*, *CR.exit2* → *DT.entry4*

where

behaviour *CE* = # see Section 10.2 # **endbehaviour**

behaviour *CR* = # see Section 10.3 # **endbehaviour**

behaviour *DT* = # see Section 10.4 # **endbehaviour**

endbehaviour

10.6 Multiple transport connections

The transport service can support multiple transport connections simultaneously between the same or distinct pairs of transport users. Simultaneous transport connections are related as follows:

1. transport connections may have to compete for the resources of the TS provider;
2. transport connections at the same TSAP must be assigned distinct TCEIs.

The first relationship implies that the number of transport connections that can be supported by the transport service is limited and depends on the resource capacity of the transport service provider. This constraint on the transport service is not elaborated in the OSI standard, and is therefore not considered any further here.

The second relationship implies that each TSAP must perform some form of TCEI administration in order to ensure that distinct transport connections can be identified uniquely at this TSAP. This is modelled as follows. We assume that each TSAP maintains a list of free TCEIs, which is called v_{tcei}. When a *Creq* or *Cind* occurs at some TSAP, a TCEI from this list is assigned to the requested transport connection, and this TCEI is removed from v_{tcei}. Consistently, when a *Rreq* or *Rind* occurs, the TCEI of the released transport connection is added to v_{tcei}. In order to guarantee that the TCEIs established in actions *Creq* and *Cind* are unique and to keep the TCEI administration consistent, we assume that actions *Creq*, *Cind*, *Rreq* and *Rind* must occur interleaved.

The behaviour at a TSAP as discussed above is called *TCEIadm*. This behaviour may be considered as a constraint on the occurrences of actions *Creq*, *Cind*, *Rreq* and *Rind* at some TSAP, which has to be added to the constraints on these actions as defined by any instance of behaviour *TC* that (partly) occurs at this TSAP. This is most intuitively modelled using the constraint-oriented composition technique explained in [16, 57, 67]. For this purpose, actions *Creq*, *Cind*, *Rreq* and *Rind* are modelled as interactions by decomposing each of these actions into an interaction contribution from *TCEIadm* and an interaction contribution from *TC*. Consequently, the behaviour of the transport service is defined as the constraint-oriented composition of an instance of behaviour *TCEIadm* for each TSAP and a, possibly, infinite number of instances of behaviour *TC*. This behaviour is defined below.

behaviour *TCs* (v_{tsaps} : *TSAPs*, $v_{endpoints}$: *Endpoints*) =

interaction structures

TCEIadm, *TC*: **interactions** <u>*Creq*</u>, <u>*Cind*</u>, <u>*Rreq*</u>, <u>*Rind*</u>
interaction points λ.*tsap*

exits/entries

tsap $\in v_{tsaps}$ | $\surd \rightarrow$ *TCEIadm.entry*(*tsap*, *TCEIsOf*(*tsap*, $v_{endpoints}$)),
any | $\surd \rightarrow$ *TC.entry*

where

behaviour *TCEIadm* =

entries

entry (v_{tsap} : *TSAP*, v_{tceis} : *TCEIs*)

interactions

$entry \wedge \neg \underline{Cind} \wedge \neg \underline{Rreq} \wedge \neg \underline{Rind} \quad \rightarrow \underline{Creq}\ (\ )\ [\lambda.tsap = v_{tsap},\ \lambda.tcei \in v_{tceis}],$
$entry \wedge \neg \underline{Creq} \wedge \neg \underline{Rreq} \wedge \neg \underline{Rind} \quad \rightarrow \underline{Cind}\ (\ )\ [\lambda.tsap = v_{tsap},\ \lambda.tcei \in v_{tceis}],$
$entry \wedge \neg \underline{Creq} \wedge \neg \underline{Cind} \wedge \neg \underline{Rind} \quad \rightarrow \underline{Rreq}\ (\ )\ [\lambda.tsap = v_{tsap}],$
$entry \wedge \neg \underline{Creq} \wedge \neg \underline{Cind} \wedge \neg \underline{Rreq} \quad \rightarrow \underline{Rind}\ (\ )\ [\lambda.tsap = v_{tsap}],$

exits/entries

$\underline{Creq} \quad \rightarrow TCEIadm.entry\ (v_{tsap}, v_{tceis} - \{\lambda_{Creq}.tcei\}),$
$\underline{Cind} \quad \rightarrow TCEIadm.entry\ (v_{tsap}, v_{tceis} - \{\lambda_{Cind}.tcei\}),$
$\underline{Rreq} \quad \rightarrow TCEIadm.entry\ (v_{tsap}, v_{tceis} \cup \{\lambda_{Rreq}.tcei\}),$
$\underline{Rind} \quad \rightarrow TCEIadm.entry\ (v_{tsap}, v_{tceis} \cup \{\lambda_{Rind}.tcei\})$

endbehaviour *TCEIadm*

behaviour *TC* =

see Section 10.5,
with *Creq*, *Cind*, *Rreq*, *Rind* being defined as interaction contributions

endbehaviour *TC*

endbehaviour *TCs*

Behaviour *TCs* is parameterized with v_{tsaps} and $v_{endpoints}$. Parameter v_{tsaps} defines the set of TSAPs of the transport service. Data type *TSAPs* represents a set of TSAPs. Parameter $v_{endpoints}$ defines for each TSAP of the transport service the set of TCEIs that are available at this TSAP. Data type *Endpoints* represents a set of pairs with signature ⟨*TSAP*, *TCEIs*⟩, where data type *TCEIs* represents a set of TCEIs. We assume that function *TCEIsOf* : *TSAP* × *Endpoints* → *TCEIs* is defined in data type *Endpoints*, such that *TCEIsOf*(*tsap*, $v_{endpoints}$) renders the set of TCEIs related to *tsap* in $v_{endpoints}$.

Behaviour part **interaction structures** of *TCs* defines which behaviours interact with each other and their means of interaction, which comprise:

- the **interactions** in which these behaviours participate at their common interaction points; and
- the **interaction points** of these behaviours, which are defined in terms of the part of the location attribute of the interactions of these behaviours that must match.

In this case, a behaviour instance of *TCEIadm* must participate in interactions *Creq*, *Cind*, *Rreq* and *Rind* with a behaviour instance of *TC*, when they have an interaction point in com-

mon. An instance of *TCEIadm* and an instance of *TC* have a common interaction point if and only if the TSAP part of the location values of their interactions, i.e., $\lambda.tsap$, is the same.

Behaviour part **exits/entries** of *TCs* defines an instance of behaviour *TCEIadm* for each TSAP in v_{tsap}. An infinite number of instances of behaviour *TC* are defined as well. The keyword **any** represents the creation of infinitely many instances of the causality relation to the right of symbol '|'.

The interaction contributions of a behaviour are defined in behaviour part **interactions**. This is illustrated for behaviour *TCEIadm*. Behaviour *TC* should be adapted correspondingly, i.e., actions *Creq*, *Cind*, *Rreq*, *Rind* should be removed from behaviour part **actions** and defined as interaction contributions in behaviour part **interactions**.

10.7 Quality of Service

The quality characteristics of a transport connection are negotiated between the calling user, the called user and the transport service provider in the CE phase by means of a number of QoS parameters. These QoS parameters are modelled by the information sub-attribute $\iota.qos$: *QoS* of actions *Creq*, *Cind*, *Crsp* and *Ccnf*. Data type *QoS* is defined below, using an ad-hoc notation for data types.

```
data type QoS = ⟨
    CEdelay   : Delay,        # TC establishment delay
    CEfail    : Probability,  # TC establishment failure probability
    CRdelay   : Delay,        # TC release delay
    CRfail    : Probability,  # TC release failure probability
    DTdelay   : Delay,        # Transit delay
    DTthrou   : Throughput,   # Throughput
    DTrer     : Probability,  # Residual error rate
    DTfail    : Probability,  # Transfer failure probability
    TCprot    : Protection,   # TC protection
    TCprior   : Priority,     # TC priority
    TCresil   : Probability   # TC resilience
    ⟩
endtype QoS
```

QoS parameter x is denoted as $\iota.qos.x$, or more briefly as $\iota.x$, since no confusion with other SP parameters is possible. For example, $\iota_{Cind}.DTdelay$ denotes the transit delay value established in the connect indication primitive.

We assume that a delay, a probability and a throughput can be represented by a positive real number, and therefore data types *Delay*, *Probability* and *Throughput* can be constructed from the standard data type of real numbers. Similarly, we assume that the data types *Protection* and *Priority* can be constructed from the standard data type of natural numbers. Conforming to the TS standard, we do not impose any constraints on the possible delay, probability, throughput, protection or priority values that can be specified.

In this section, we discuss how the quality (performance) characteristics associated with each QoS parameter can be modelled. We also discuss the modelling of flow control by backpressure. This section only presents the extensions necessary to incorporate these characteristics in the transport service model defined so far.

10.7.1 Connection establishment phase

Two QoS parameters apply to the CE phase: TC establishment delay (*CEdelay*) and TC establishment failure probability (*CEfail*). It seems paradoxical that these parameters determine the quality characteristics of the CE phase, since they are established during the CE phase. The QoS negotiation strategy of the transport service allows this, however, since the requested QoS may not be raised during QoS negotiation. This implies that values of some QoS parameters may not be decreased, e.g., a delay value, and values of other QoS parameters may not be increased, e.g., a throughput value.

TC establishment delay

Parameter *CEdelay* defines the maximal acceptable delay between the occurrences of actions *Creq* and *Ccnf*. The calling user requests a value for this parameter in *Creq*. This value may be increased or kept the same by the transport service provider and by the called user in actions *Cind* and *Crsp*, respectively. The delay between actions *Creq* and *Ccnf* is defined as $\tau_{Ccnf} - \tau_{Creq}$.

The QoS characteristic defined by *CEdelay* is modelled by adding the following time constraints to actions *Cind*, *Crsp* and *Ccnf* in behaviour *CE* (see Section 10.2):

$$Cind : [\tau_{Cind} < \tau_{Creq} + \iota_{Creq}.CEdelay];$$

$$Crsp : [\tau_{Crsp} < \tau_{Creq} + \iota_{Cind}.CEdelay];$$

$$Ccnf : [\tau_{Ccnf} \leq \tau_{Creq} + \iota_{Crsp}.CEdelay].$$

Since the negotiation strategy of the transport service implies that $\iota_{Creq}.CEdelay \leq \iota_{Cind}.CEdelay \leq \iota_{Crsp}.CEdelay$, the conjunction of the constraints above always allows the occurrences of actions *Creq*, *Cind*, *Crsp* and *Ccnf*, such that: $\tau_{Creq} < \tau_{Cind} < \tau_{Crsp} < \tau_{Ccnf}$. This property would not be guaranteed if the transport provider or the called user were allowed to decrease the requested value of *CEdelay*.

For example, assume $\iota_{Creq}.CEdelay = 4$, $\iota_{Cind}.CEdelay = 2$ and $\tau_{Creq} = 1$. The time constraint of action *Crsp*, including implicit time constraint $\tau_{Cind} < \tau_{Crsp}$, is defined as:

$$\tau_{Cind} < \tau_{Crsp} < \tau_{Creq} + 2.$$

In case action *Cind* occurs at $\tau_{Cind} = 4$, this time constraint renders $4 < \tau_{Crsp} < 3$, which implies that *Cind* can not occur.

TC establishment failure probability

Parameter *CEfail* defines the ratio of total connection establishment failures to total connection establishment attempts. A connection establishment attempt is characterized by the

occurrence of action *Creq* and an attempt fails when its corresponding *Ccnf* does not occur. Parameter *CEfail* is negotiated analogously to parameter *CEdelay* above.

We assume that the transport service is allowed to perform better than the specified value of *CEfail*, since it makes no sense to deliberately implement unreliability to meet this value. This implies that *CEfail* should be interpreted as a maximal acceptable ratio.

The probability of actions *Cind*, *Crsp* and *Ccnf* is defined as:

$\Pi_{Cind} = \pi^*_{Cind}(Creq) \times \Pi_{Creq}$;

$\Pi_{Crsp} = \pi^*_{Crsp}(Cind \wedge entry2) \times \Pi_{Cind}$;

$\Pi_{Ccnf} = \pi^*_{Ccnf}(Crsp \wedge entry3) \times \Pi_{Crsp}$.

This implies that the establishment failure probability is equal to

$1 - (\pi^*_{Ccnf}(Crsp \wedge entry3) \times \pi^*_{Crsp}(Cind \wedge entry2) \times \pi^*_{Cind}(Creq))$.

The extended integral probability attribute is used to model the conditional probability of TSPs, since their causality conditions involve two-sided conditions. Actions *Creq* and *Cind* are exceptions. In the case of these two actions, the use of the simple or extended probability attribute would make no difference.

The QoS characteristic defined by *CEfail* is modelled by adding the following integral probability constraints to actions *Cind*, *Crsp* and *Ccnf*:

$Cind : [\pi^*_{Cind}(Creq) \geq 1 - \iota_{Creq}.CEfail]$;

$Crsp : [\pi^*_{Crsp}(Cind \wedge entry2) \geq (1 - \iota_{Cind}.CEfail) / \pi^*_{Cind}(Creq)]$;

$Ccnf : [\pi^*_{Ccnf}(Crsp \wedge entry3) \geq (1 - \iota_{Crsp}.CEfail) / (\pi^*_{Crsp}(Cind \wedge entry2) \times \pi^*_{Cind}(Creq))]$.

The constraint of *Cind* represents that *Cind* must occur after *Creq* has occurred with a probability equal to or larger than the complement of the failure probability established in *Creq*. The constraint of *Crsp* represents that *Crsp* must occur after *Creq* has occurred with a probability equal to or larger than the complement of the failure probability established in *Cind*. This probability comprises the uncertainty of the occurrences of *Creq* and *Cind*. Therefore, the conditional probability that *Crsp* must occur assuming *Cind* has occurred is equal to the complement of the failure probability established in *Cind* divided by the probability that *Cind* occurs. The constraint of *Ccnf* is defined analogously.

Since the negotiation strategy of the transport service implies that $\iota_{Creq}.CEfail \leq \iota_{Cind}.CEfail \leq \iota_{Crsp}.CEfail$, the above probability constraints are consistent for any possible values of $\pi^*_{Cind}(Creq)$, $\pi^*_{Crsp}(Cind \wedge entry2)$ and $\pi^*_{Ccnf}(Crsp \wedge entry3)$. This would not be guaranteed if the transport provider or the called user were allowed to decrease the value of *CEfail*.

For example, assume $\iota_{Creq}.CEfail = 0.2$, $\iota_{Cind}.CEfail = 0.1$ and $\pi_{Cind}(Creq) = 0.8$. In this case the probability constraint of action *Crsp* is defined as: $\pi^*_{Crsp}(Cind) \geq (1 - 0.1) / 0.8 > 1$. This constraint is incorrect, since probability values must be in the range [0..1].

The OSI standard excludes failures as a result of error, TC refusal, or excessive delay caused by a TS user in the calculation of the TC establishment failure probability. This requirement stems from viewing the transport service as the behaviour of the transport serv-

ice provider. Indeed, if we consider the TS behaviour specified so far as the behaviour of the transport service provider, this requirement is satisfied.

However, we define the transport service as the common behaviour of the transport service users and the transport service provider. This implies that the contribution of the transport service users in the service behaviour is modelled explicitly, and the requirement of the OSI standard discussed above should be ignored. The explicit modelling of the contribution of the transport service users can be seen as a prescription of their behaviour in using the transport service provider.

10.7.2 Connection release phase

The following QoS parameters apply to the CR phase: TC release delay (*CRdelay*) and TC release failure probability (*CRfail*). These parameters are established separately for the calling and called user. Therefore, both *CRdelay* and *CRfail* are defined as a tuple ⟨*clg*, *cld*⟩, such that *CRdel.clg* and *CRfail.clg* represent the release delay and failure probability in the direction of the calling user, respectively, and *CRdel.cld* and *CRfail.cld* represent the release delay and failure probability in the direction of the called user, respectively.

These QoS parameters are negotiated in the CE phase, whereas the CR phase can already be initiated during the CE phase. Figure 10.8 depicts the possible scenario's of a user invoked connection reject.

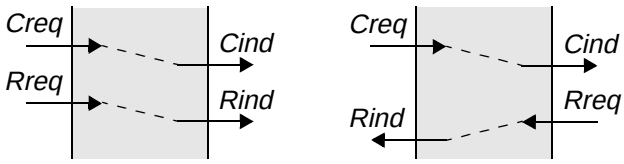


Figure 10.8: User invoked connection reject

The OSI standard does not define whether the *CRdelay* and *CRfail* values established in action *Creq* or the ones established in action *Cind* have to be used in the scenario's of Figure 10.8. We assume the *CRdelay* and *CRfail* values of action *Cind* have to be used in this case, since they comprise the constraints imposed by the calling user and the transport service provider.

In case a connection release is invoked during the DT phase, the *CRdelay* and *CRfail* values established in actions *Crsp* and *Ccnf* are used.

TC release delay

Parameter *CRdelay* defines the maximal acceptable delay between the occurrence of a *Rreq* at one side of the connection and the occurrence of a corresponding *Rind* at the other side. This delay is defined as $\tau_{Rind} - \tau_{Rreq}$.

The QoS characteristic defined by *CRdelay* is modelled by adding the following time constraints to action *Rind* in behaviour *CR-half* (see Section 10.3):

Rind at calling side: (in *CR-half*[1]) [if @2 : $\tau_{Rind} \leq \tau_{Rreq} + \iota_{Cind}.CRdelay.clg$,
if @5 : $\tau_{Rind} \leq \tau_{Rreq} + \iota_{Ccnf}.CRdelay.clg$],

Rind at called side: (in *CR-half*[2]) [if @2 : $\tau_{Rind} \leq \tau_{Rreq} + \iota_{Cind}.CRdelay.cld$,
if @5 : $\tau_{Rind} \leq \tau_{Rreq} + \iota_{Crsp}.CRdelay.cld$].

Action *Rind* at the calling side can refer to the value of *CRdelay.cld* established in actions *Cind* and *Ccnf*, and action *Rind* at the called side can refer to the value of *CRdelay.clg* established in actions *Cind* and *Crsp*. The entry and exit points of behaviours *CE* and *CR-half* involved in the causality condition of *Rind* at the calling side or in the causality condition of *Rind* at the called side should be adapted in order to allow that references are made via their parameter lists.

TC release failure probability

Parameter *CRfail* defines the ratio of total connection release requests resulting in release failure to total release requests. A release request fails when the corresponding release indication does not occur.

The QoS characteristic defined by *CRfail* is modelled by adding the following integral probability attribute constraints to action *Rind* in behaviour *CR-half* (see Section 10.3):

Rind at calling side: (in *CR-half*[1]) [$\pi^*_{Rind}(@2) \geq 1 - \iota_{Cind}.CRfail.clg$,
$\pi^*_{Rind}(@5) \geq 1 - \iota_{Ccnf}.CRfail.clg$],

Rind at called side: (in *CR-half*[2]) [$\pi^*_{Rind}(@2) \geq 1 - \iota_{Cind}.CRfail.cld$,
$\pi^*_{Rind}(@5) \geq 1 - \iota_{Crsp}.CRfail.cld$].

Causality conditions @2 and @5 represent infinitely many alternative causality conditions, due to the condition represented by entry point *entry6* of behaviour *CR-half*. Expressions $\pi^*_{Rind}(@2)$ and $\pi^*_{Rind}(@5)$ denote that each alternative causality condition in @2 and @5 should be associated the above probability constraints, respectively. The alternative causality conditions in @2 and @5 are exclusive, in the sense that action *Rind* can be enabled by only one of them simultaneously. This property guarantees that the conditional probability that *Rind* occurs at the calling (called) side once *Rreq* has occurred at the remote side is equal to or larger than the complement of the specified release failure probability *CRfail.cld* (*CRfail.clg*).

Considering the references of action *Rind* to the values of *CRfail.clg* and *CRfail.cld*, a similar remark applies here as the one that has been made for the *CRdelay* parameter.

10.7.3 Data transfer phase

The following QoS parameters apply to the DT phase: Transit delay (*DTdelay*), Throughput (*DTthrou*), Residual Error Rate (*DTrer*) and Transfer failure probability (*DTfail*). The values of each of these parameters that are effectively used during a connection are established in actions *Crsp* and *Ccnf*. Parameters *DTdelay*, *DTthrou* and *DTfail* are defined as a tuple ⟨*clg*, *cld*⟩, where *clg* and *cld* define separate QoS values for the quality of data transfer in the direction of the calling and called user, respectively.

Transit delay

Parameter *DTdelay* defines the maximal transit delay of normal data units for each direction of a transport connection separately. Transit delay is defined as the elapsed time between the occurrence of an instance of action *Dreq* and the occurrence of the corresponding instance of action *Dind*, i.e., $\tau_{Dind} - \tau_{Dreq}$.

The QoS requirement defined by *DTdelay* is modelled by adding the following time constraint to each alternative causality condition of action *Dind* in behaviour *DT-inst* (see Section 10.4):

$[\tau_{Dind} \leq \tau_{Dreq} + v_{qos}.DTdelay]$,

where we assume that $v_{qos}.DTdelay$ gets the value $\iota_{Crsp}.DTdelay.clg$ or $\iota_{Ccnf}.DTdelay.cld$, depending on whether *DT-inst* models data transfer in the direction of the calling or called user, respectively.

The OSI standard also defines an average transit delay parameter. This parameter can be modelled using the stochastic probability attribute, by defining $x = \tau_{Dind} - \tau_{Dreq}$ as a stochastic variable and associating a probability distribution function with this variable, such that the average and integral of this function are equal to the average and maximal transit delay, respectively. We do not further elaborate the modelling of this stochastic probability attribute in this thesis.

Throughput

Parameter *DTthrou* defines the maximal throughput of normal data units for each direction of a transport connection separately. A maximal throughput value represents the maximal rate at which data units are sent and delivered in one transfer direction. Given a sequence of n successfully transferred normal data units, with $n \geq 2$, the sending and receiving rates are defined as:

- *sending rate*: the number of octets contained in the last n-1 data units divided by the time period between the first and last data *request* in the sequence;
- *receiving rate*: the number of octets contained in the last n-1 data units divided by the time period between the first and last data *indication* in the sequence.

The QoS requirement defined by *DTthrou* is modelled by adding the following combination of information and time constraints to each alternative causality condition of action *Dind* in behaviour *DT-inst* (see Section 10.4):

action *Dreq*:

$[\ (\ \Sigma_{(first+1 \leq i \leq self)}\ Len(\iota(Dreq^{i}).usr)\ )\ /\ (\ \tau(Dreq^{self}) - \tau(Dreq^{first})\ ) \leq v_{qos}.DTthrou]$;

action *Dind*:

$[\ (\ \Sigma_{(first+1 \leq i \leq self)}\ Len(\iota(Dind^{i}).usr)\ )\ /\ (\ \tau(Dind^{self}) - \tau(Dind^{first})\ ) \leq v_{qos}.DTthrou\ ]$,

where we assume that

- function *Len* denotes the number of octets of a user data unit;

- function *self* denotes the current instance of *Dreq*/*Dind*;
- function *first* denotes the first instance of *Dreq*/*Dind* in the current sequence of successfully transferred normal data units; and
- $v_{qos}.DTthrou$ gets the value $\iota_{Crsp}.DTthrou.clg$ or $\iota_{Ccnf}.DTthrou.cld$, depending on whether *DT-inst* models data transfer in the direction of the calling or called user, respectively.

Action instances $Dreq^{self}$ and $Dind^{self}$ can refer to $Dreq^{i}$ and $Dind^{i}$, respectively, with $first \leq i < self$, since they depend (indirectly) on $Dreq^{i}$ and $Dind^{i}$ via enabling relations.

The OSI standard also defines an average throughput parameter. The modelling of this parameter is not elaborated in this thesis for the sake of brevity.

Residual error rate

Parameter *DTrer* is defined as the ratio of the total number of lost, duplicated and incorrectly received normal data units to the total number of normal data units submitted for transfer. This QoS parameter applies to both directions of transfer.

According to this parameter, the probability that a normal data request is followed by a corresponding normal data indication, which satisfies all attribute constraints considered so far, is equal to or larger than the complement of *DTrer*. This characteristic is modelled by adding the following integral probability attribute constraint to each alternative causality condition γ of action *Dind* in behaviour *DT-inst* (see Section 10.4):

$[\pi^{*}_{Dind}(\gamma) \geq 1 - v_{qos}.DTrer]$,

where we assume that $v_{qos}.DTrer$ gets the value $\iota_{Crsp}.DTrer$ or $\iota_{Ccnf}.DTrer$, depending on whether *DT-inst* models data transfer in the direction of the calling or called user, respectively.

We do not explicitly model data corruption, data loss or data duplication in the service behaviour, since we consider a design as a prescription for implementation, which defines what we want to happen. In this case, we want normal data units to be delivered correctly with a minimal probability of $1 - v_{qos}.DTrer$. This implies that a normal data unit is not delivered with a maximal probability of $v_{qos}.DTrer$. Unreliability reasons are considered irrelevant at the abstraction level of the transport service.

In addition to the modification above, the causality condition of action *Dind* has to be adapted. We explain this by considering instance $Dind^{i}$. Due to the possible non-occurrence of instances of action *Dind*, instance $Dind^{i}$ should depend directly on condition $(Eind^{i-1} \vee Dind^{i-1} \vee (\neg Eind^{i-1} \wedge \neg Dind^{i-1}))$, which is briefly denoted as Γ^{i-1}, and should depend indirectly on the conjunction of conditions $\wedge_{j < i-1} \Gamma^{j}$, with $\Gamma^{1} = Ccnf$ or $\Gamma^{1} = Crsp$ depending on whether data delivery at the calling or called side is modelled, respectively. The complete adapted causality condition of $Dind^{i}$ is defined below.

$$(Eind^{i-1} \vee Dind^{i-1} \vee (\neg Eind^{i-1} \wedge \neg Dind^{i-1})) \wedge \wedge_{j < i-1} \Gamma^{j} \wedge Dreq^{i} \wedge (\neg Rreq \wedge \neg Rind) \wedge \wedge_{k > i} \neg Dind^{k} \rightarrow Dind^{i},$$

where the locations of actions *Rreq* and *Rind* and actions *Dind* and *Eind* are the same.

The above causality condition defines a disabling relation between instance $Dind^i$ and all instances $Dind^j$ and $Eind^j$, with $j < i$, such that $Dind^i$ may disable the occurrence of previous instances of *Dind* and *Eind*. This implies that $Dind^i$ may be disabled by subsequent instances of *Dind*, which is represented by condition $\wedge_{k > i} \neg Dind^k$.

Action instance $Dind^i$ should only disable preceding instances of *Dind* and *Eind*, when the occurrences of these instances become impossible due to the expiration of the maximal transit delay. The last one of these instances that becomes impossible due to the expiration of the maximal transit delay is $Dind^{i-1}$ or $Eind^{i-1}$. The occurrence of this instance is impossible when the maximal transit delay has passed since the occurrence of $Dreq^{i-1}$ or $Ereq^{i-1}$, respectively. Consequently, instance $Dind^i$ should not occur before Δt time units have passed since the occurrence of $Dreq^i$, with:

$$\Delta t = \begin{cases} v_{qos}.DTdelay - \Delta t_{req}, & \text{if } \Delta t_{req} < v_{qos}.DTdelay; \\ 0, & \text{otherwise,} \end{cases}$$

where $\Delta t_{req} = \tau(Dreq^i) - \tau(Dreq^{i-1} \text{ or } Ereq^{i-1})$ represents the time interval between the occurrence of $Dreq^i$ and the occurrence of the preceding normal or expedited data request.

This implies that the following time constraint is associated with alternative condition $\neg Eind^{i-1} \wedge \neg Dind^{i-1} \wedge \wedge_{j < i-1} \Gamma^j \wedge Dreq^i \wedge (\neg Rreq \wedge \neg Rind) \wedge \wedge_{k > i} \neg Dind^k$:

$$[\tau(Dind^i) > \tau(Dreq^i) + v_{qos}.DTdelay - \Delta t'_{req}],$$

with $\Delta t'_{req} = min(v_{qos}.DTdelay, \tau(Dreq^i) - \tau(Dreq^{i-1} \text{ or } Ereq^{i-1}))$.

Based on the above, the causality relation of action instance $Dind^i$ in behaviour *DT-inst* is defined as follows, including the characteristic imposed by the maximal transit delay parameter:

$$\begin{aligned} & entry5^i \wedge Dreq^i \wedge entry7^i \wedge \\ & \quad (\quad entry2^i\ [\pi^*(Dind^i) \geq 1 - v_{qos}.DTrer] \\ & \qquad \vee \\ & \qquad entry6^i\ [\tau(Dind^i) > \tau(Dreq^i) + v_{qos}.DTdelay - \Delta t'_{req}, \pi^*(Dind^i) \geq 1 - v_{qos}.DTrer] \\ & \quad) \qquad \rightarrow Dind^i\ (\ )\ [\ \tau(Dind^i) < \tau(Dreq^i) + v_{qos}.DTdelay, \\ & \qquad\qquad \iota = \iota(Dreq^i), \lambda = v_{rcv}\ ]\ . \end{aligned}$$

Entry points *entry6* and *entry7*, and the corresponding exit points, are added to behaviour *DT-inst*, which represent causality condition $\neg Eind^{i-1} \wedge \neg Dind^{i-1}$ and causality condition $\wedge_{k > i} \neg Dind^k$, respectively. Probability constraint $[\pi^*(Dind^i) \geq 1 - v_{qos}.DTrer]$ should be associated with each alternative causality condition defined by $entry5 \wedge Dreq \wedge entry7 \wedge entry2$. These alternative conditions are exclusive conditions, such that only one of them can be satisfied.

Finally, the causality relation of action instance $Eind^i$ is adapted to define its possible disabling by action instances $Dind^k$, with $k > i$. This completes the definition of the disabling relations between certain instances of *Dind* and *Eind* mentioned before. Furthermore, we assume that expedited data transfer should also obey the maximal transit delay requirement:

$$entry3 \wedge entry5 \wedge entry7 \wedge Ereq^i \rightarrow Eind^i\ (\ )\ [\quad \tau < \tau(Ereq^i) + v_{qos}.DTdelay,\ \iota = \iota(Ereq^i),\ \lambda = v_{rcv}]$$

In the transport service model defined so far, an instance of *Eind* is never disabled by an instance of *Dind*, since expedited data transfer is assumed to be reliable.

Transfer failure probability

Parameter *DTfail* defines the ratio of total transfer failures to total transfer attempts. A transfer attempt corresponds to the sending of a data unit. An attempt fails when the requirements as specified by the throughput, transit delay and residual error rate QoS parameters are not satisfied.

A *DTfail* value smaller than 1 implies that data units may be delivered:

- with a higher throughput than the maximal throughput;
- after the maximal transit delay has expired; and
- with a higher residual error rate.

We do not model these type of transfer failures explicitly in this thesis, since we do not want to prescribe their implementation. In our specification of the transport service, the occurrence of a data indication models the correct delivery of a data unit to the called user, satisfying the specified maximal throughput and transit delay.

The occurrence of one or more of the failures above should result in a provider initiated connection release, since the negotiated quality of the transport connection could not be guaranteed any more. Therefore, we model the transfer failure probability as the probability of a provider initiated release. The latter probability is denoted as the resilience of a transport connection, which is discussed in Section 10.7.4.

10.7.4 Miscellaneous

The following QoS parameters do not apply to a specific phase, but to a transport connection as a whole: TC protection (*TCprot*), TC priority (*TCprior*) and TC Resilience (*TCresil*).

TC protection and TC priority

Parameter *TCprot* defines to which extent the transport service prevents unauthorized monitoring or manipulation of data units. The following types of protection can be selected: (i) no protection, (ii) protection against passive monitoring, (iii) protection against modification, replay, addition or deletion, or (iv) both (ii) and (iii). The provision of this QoS parameter can not be modelled in the transport service, since it does not affect the interactions between the transport service provider and the (intended) transport service users, except for the negotiation of the type of protection.

Parameter *TCprior* defines the relative importance of a TC w.r.t. (i) the order in which the QoS of TCs is degraded (if necessary), and (ii) the order in which TCs are aborted to

recover resources (if necessary). The modelling of the provision of this QoS parameter is only meaningful if resource management functions are explicitly defined in terms of the resource capacity and resource allocation strategy. Implementation information is necessary to do this.

Resilience of the TC

Parameter *TCresil* defines the probability that the TS provider initiates a connection release during a time interval of a specified size ΔT, without a prior user invoked disconnect. This implies that, independent of the duration of a connection at some time moment $\tau 1$, the probability that the provider initiates a disconnect in time interval $(\tau 1..\tau 2]$, with $\Delta T = \tau 2 - \tau 1$, is equal to *TCresil*, provided that no disconnect request has occurred yet.

The most intuitive and straightforward way to model the resilience parameter is through the introduction of an additional action *Rprov*, which models the internal decision of the transport provider to initiate a connection release. This action does not represent a service primitive, which implies that the transport service specification would partly become an intensional specification. The causality relation of action *Rprov* would be defined as:

$$Creq \wedge \neg Rreq^1 \wedge \neg Rreq^2 \rightarrow Rprov\ [\ \pi^*(Creq \wedge \neg Rreq^1 \wedge \neg Rreq^2, \tau) = 1 - e^{-\mu \cdot \tau}\ ].$$

Action *Rprov* is enabled by a connect request, and is disabled by a disconnect request at one or both sides of the connection. The exponential distribution function $1 - e^{-\mu \cdot \tau}$, with parameter μ, is associated with this condition. This distribution function defines the conditional probability that *Rprov* occurs within time interval $(\tau_{Creq}..\tau]$, i.e., $\tau_{Creq} < \tau_{Rprov} \leq \tau$. The conditional probability that *Rprov* occurs in an arbitrary interval $(\tau 1..\tau 2]$ during the connection is equal to $\text{Prob}(\tau \leq \tau 2 \mid \tau > \tau 1) = 1 - e^{-\mu \cdot \Delta T}$ ([69]). This probability is independent of the duration of the connection at $\tau 1$, with $\tau 1 - \tau_{Creq} > 0$, due to the memoryless property of exponential distribution functions. Consequently, parameter μ should be chosen such that the following equation holds: $1 - e^{-\mu \cdot \Delta T} = TCResil$.

In order to model that a provider initiated release is indicated to the transport users within $\Delta T1$ time units, the causality relation of action *Rind* should be adapted as follows (see behaviour *CR-half* in Section 10.3): (i) replace start condition $\surd$ by enabling condition *Rprov* in @1 and @4, and (ii) add time-constraint $[\tau_{Rind} < \tau_{Rprov} + \Delta T1]$ to the alternative conditions in @1 and @4. In addition, disabling condition $\neg Rprov$ should be added (in conjunction) to the causality condition of action *Rreq* in order to model the choice between *Rreq* and *Rprov.*

10.7.5 Flow control by backpressure

The rate at which a sending user executes (normal or expedited) data requests may be limited by the rate at which the corresponding receiving user executes data indications. This can be understood as follows. When the sending user submits new data units to the transport provider at a higher rate than the transport provider can deliver these data units to the receiving user, the buffering capacity of the transport connection will eventually become completely utilized. From this moment, the sending user can only submit new data units

after the receiving user has released some buffer by accepting data units from the provider. This characteristic of the transport service is called *flow control by backpressure*.

Flow control by backpressure for normal data units can be modelled by adding condition $Dind^{i-M} \vee Eind^{i-M}$ to the causality condition of action instance $Dreq^i$, (see behaviour *DT-inst* in Section 10.4), with $i > M$, resulting in the following causality relation:

$$entry1^i \wedge entry4^i \wedge \neg Ereq^i \wedge (Dind^{i-M} \vee Eind^{i-M}) \rightarrow Dreq^i\ (\ )\ [\lambda = v_{snd}]\ ,$$

where constraint $[\lambda = v_{snd}]$ is associated with each alternative causality condition of $Dreq^i$.

Condition $Dind^{i-M} \vee Eind^{i-M}$ represents that $Dreq^i$ is not allowed to occur before the previous M normal and expedited data units have been delivered. For convenience, we assume no data units are lost and therefore either a normal or expedited data unit is delivered in the (i-M)th instance of *DT-inst*.

Similarly, flow control by backpressure for expedited data units can be modelled by adding condition $Dind^{i-N} \vee Eind^{i-N}$ to the causality condition of action instance $Ereq^i$, resulting in the following causality relation:

$$entry1^i \wedge entry4^i \wedge \neg Dreq^i \wedge (Dind^{i-N} \vee Eind^{i-N})\ [v_{exp}] \rightarrow Ereq^i\ (\ )\ [\lambda = v_{snd}],$$

where constraints $[v_{exp}]$ and $[\lambda = v_{snd}]$ are associated with each alternative causality condition of $Ereq^i$.

The OSI standard prescribes that the execution of a normal data request may not prevent the subsequent execution of an expedited data request (see clause 9.2f in [32]). This requirement can be satisfied by prescribing that $N < M$. In this case, the (buffer) capacity of a transport connection is equivalent to $2 \times M - N$ data units. For example, when $N = M - 1$ it is always possible to submit one expedited data unit after the submission of normal data units is blocked, which implies that maximal $M + 1$ data units can be in transit. The values of N and M can be defined as parameters of a transport connection, which can be established by the transport provider during connection establishment.

10.8 Conclusions

Our basic design concepts are sufficiently expressive and general to model the functional and QoS aspects of the transport service. Service primitives and relations between service primitives, including information, location and time aspects, can be modelled directly and properly at the considered abstraction level, without using specification artifices or being forced to change the abstraction level. Probability aspects of the transport service can also be modelled using our design concepts, but sometimes less directly than we desired, particularly in case actions have multiple alternative causality conditions. However, these alternative conditions are exclusive, in the sense that only one of them can enable the result action (simultaneously), facilitating the use and comprehension of the (extended) probability attribute.

This case study clearly illustrates the need for shorthand notations for composite design concepts in order to obtain concise behaviour specifications. Particularly, shorthand nota-

tions for disabling relations and for exit/entry combinations are useful. The disabling of an action having many distinct instances renders many alternative causality conditions for the disabling action. This is inconvenient and obscures the modelling of probability. Therefore, a shorthand for disabling relations should be accompanied with an easy-to-use probability attribute. Exit/entry combinations can be used more concisely by introducing shorthands for frequently used compositions of sub-behaviours, such as sequential composition, disabling and choice.

Despite its lack of conciseness, our specification has several advantages over the LOTOS specification of the transport service as presented in [34]:

- besides the negotiation of QoS parameter values, the behaviour characteristics imposed by these values during the transport connection are also modelled;
- flow control by backpressure is modelled more properly by means of a causal relationship between the acceptance of data units at the receiving side and the submission of new data units at the sending side;
- the precedence relationship between normal and expedited data units is modelled directly in terms of causal relations between the corresponding data primitives. The LOTOS specification models this indirectly in terms of a queue in which the ordering of data units may be changed;
- the administration of TCEIs is modelled here as a local constraint of a TSAP. In the LOTOS specification this is modelled as a global constraint of the transport service;
- the relationship between provider initiated disconnect indications at both sides of the connection is modelled explicitly.

Chapter 11

Conclusions

In this chapter we summarize the main results of our research and indicate some directions for further work. This chapter is structured as follows. Section 11.1 addresses the issue of architectural versus formal reasoning, Section 11.2 discusses the main results of this thesis and Section 11.3 suggests further work.

11.1 Architectural versus formal reasoning

An important idea underlying our research is that the development of an effective design methodology, including an effective design language, should be based on properly chosen and precisely defined basic design concepts. Properly chosen design concepts represent essential system conceptions that are derived from the real world, and not from mathematical models. This thesis further elaborates the work of Ferreira Pires reported in [16], since this work identifies basic design concepts that support, amongst others, causality-based behaviour definition, the modelling of timing and probability aspects, and behaviour refinement.

This thesis carefully refines the basic concepts identified in [16] and precisely defines these concepts using a formal model. Only after the careful identification and elaboration of the pragmatically necessary design concepts in architectural terms, the formal elaboration of these concepts becomes relevant. A formal design language that is not based on a careful consideration of the architectural purpose of its underlying concepts usually presents severe limitations on its practical applicability and therefore misses the essential prerequisites for using it.

Therefore, this thesis provides formal elaboration subordinate to preceding architectural work. Quite often, we can experience that familiarity with existing formal models, including their limitations, prevents one from accepting new improved design concepts, in particular when they do not fit within these formal models. This, however, is a weak reason for not using new concepts. Although formal methods are important means to improve the correctness and reliability of systems, the use of proper and expressive concepts is equally, perhaps even more, important to achieve these goals. One should not forget that many successful systems have been built without the support of formal methods.

We consider the approach of this thesis, in which the architectural elaboration of basic design concepts is followed by a formal elaboration, a useful one.

11.2 Main results

The main achievements of this thesis are the development of an expressive basic design language for distributed system behaviours and a general method to support the refinement of such behaviours.

11.2.1 Basic design language

Chapters 4 to 7 present an expressive basic design language that supports the conception and specification of distributed system behaviours, based on a careful consideration of the action concept and the concept of causality relation. This language has been developed using a modular approach. The following basic design (language) modules are distinguished:

- the *basic module*, which supports the design of a large variety of temporal ordering relations, based on the elementary causality conditions of enabling, disabling and synchronization, and their composition using the conjunction and disjunction operators. This module also allows one to model the uncertainty of the occurrence of an action once its causality condition is satisfied, using the uncertainty attribute;
- the *information, location and time modules*, which support the design of constraints on the information, time and location attribute of an action, respectively, including constraints that are local to the action and constraints that involve references to the information, time and location attribute values established in causally related actions. The *mixed module* combines these modules to support the design of mixed constraints involving the information, location and time attribute;
- the *integral and stochastic probability modules*, which support the design of the probability of the occurrence of an action once its causality condition is satisfied. The integral probability attribute specifies this probability in terms of a set of probability values in the range [0..1] of real numbers. The stochastic probability attribute specifies this probability as a function of the moments at which the action is allowed to occur, in terms of a probability distribution function.

Abstraction hierarchy

Section 2.5 presents an abstraction hierarchy that structures the characteristics of action relations that we consider essential in the design of distributed system behaviours, at subsequent abstraction levels. This hierarchy facilitated the systematic and modular development of the causality relation concept, which resulted in the basic design modules described above.

The basic module is an abstraction of all the other modules. The basic module abstracts from the information, location and time attribute and restricts the modelling of probability to the presence or absence of uncertainty. The information, time and location module refine the basic module, allowing one to design information, time and location attribute constraints independently of each other and in a uniform way. The mixed module relates the

information, time and location module, allowing one to relate information, time and location attribute constraints.

The integral probability module is a refinement of the basic module, which allows one to model the uncertainty of actions in more detail. Information, time and location attribute constraints can be modelled independently of integral probability attribute constraints, but the opposite in not necessarily true, only under certain conditions. The stochastic probability module is a refinement of the time module and the integral probability module.

Execution model

Chapter 3 presents the execution model. This model defines a behaviour by enumerating all possible executions of this behaviour. The execution concept proved to be an intuitive and expressive notion to define the formal semantics of causality relations in a compositional way. A constraint-oriented approach has been used, in which the semantics of (part of) a causality relation is defined as a constraint on the possible executions of the behaviour. This constraint enumerates the executions that are allowed by (part of) the causality relation. This approach enables one to define the semantics of multiple (parts of) causality relations as the conjunction of the constraints they impose on the executions of the behaviour.

Expressive power

Our basic design language supports the design of many types of relations between actions, including commonly used relations such as sequential composition, disabling, choice, interleaving and independence. This language also supports the design of information, location, time and probability aspects in a general and flexible way.

Chapter 9 introduces the causality-oriented structuring technique of [16, 57] as a technique to structure behaviours into compositions of sub-behaviours and their relationships. This technique is extended to support the design of (infinitely) repetitive behaviours by means of the repeated instantiation of finite sub-behaviour definitions.

The constraint-oriented structuring technique defined in [16, 57] is not elaborated in this thesis, but is used informally in Chapter 10 to model part of the OSI Connection-oriented Transport Service. Once this technique is added to our basic design language, we believe this language is highly expressive, comparable to, or even stronger than, process algebras and event structures (see also the concluding remarks of Chapters 5, 6 and 7).

11.2.2 Behaviour refinement

Chapter 8 presents an integrated set of methods to support the refinement of behaviours that can be expressed using our basic design language. Two basic types of refinement are supported: action refinement and causality refinement.

Behaviour refinement is supported indirectly by means of assessing the conformance relation between an abstract behaviour and the concrete behaviour that is obtained from the abstract behaviour by an instance of causality refinement or action refinement. This assess-

ment involves two steps: (i) the abstraction step, which determines the abstraction of the concrete behaviour, and (ii) the comparison step, which compares this abstraction with the original abstract behaviour. We developed rules to perform the abstraction and comparison steps. These rules do not impose constraints on the type of behaviours that can be validated.

We also present an integrated set of execution-based methods to perform the assessment of the conformance relation based on the executions of the abstract behaviour and the concrete behaviour. The correspondence between the presented execution-based methods and causality-based methods is discussed informally.

11.3 Further work

We identify a number of topics for further research in the following directions: extension of the basic design language, additional support for behaviour refinement and the development of techniques for behaviour analysis. Part of this work is being carried out in the Testbed project ([19]).

Design language

The basic design language presented in this thesis should be extended in several ways to become a full-grown operationally applicable design language. We propose the following extensions:

- addition of the constraint-oriented structuring technique, enabling the specification of interactions (partitioned actions) between multiple sub-behaviours;
- integration of entity domain concepts in the design language;
- development of concise and easy-to-use shorthand notations for frequently used composite design concepts (shorthand notations could be developed for different application domains as well);
- elaboration of the integral and stochastic probability attributes, in particular the combined use of the simple and extended interpretations of these attributes and the use of the probability attributes in combination with the information, time and location attributes (see also the concluding remarks in Chapter 7).

Behaviour refinement

The assessment of the conformance relation between an abstract and a concrete behaviour is performed in two steps: the abstraction step and the comparison step. The abstraction step is further divided into three (sub-)steps: (i) the decomposition of the concrete behaviour into alternative behaviours, (ii) the determination of the abstraction of each alternative behaviour, and (iii) the definition of the abstract behaviour as the disjunction of the abstract alternative behaviours. A complete set of abstraction rules has been presented to perform step (ii), except for the case in which an alternative behaviour uses the simple and extended integral probability attributes in combination. Abstraction rules have to be developed for this

case. Rules to simplify the abstract behaviour definition obtained in step (iii), by integrating disjunctions of equivalent alternative causality conditions, are also necessary.

Rules to perform the comparison step have not been defined in this thesis. However, examples of correctness relations have been considered on which these rules can be based. Correctness relations determine the type of refinements (implementations) of a behaviour that are considered correct. For convenience, we have assumed a strong correctness relation, in which all relations and attribute values of the abstract behaviour are preserved by the concrete behaviour. However, the use of weaker correctness relations during the design process may be useful and should be investigated.

Further research on behaviour refinement should concentrate on the development of software tools to (partly) automate the abstraction and comparison steps. For this purpose, the execution-based methods seem very suitable, since their definitions are complete and rather straightforward. The availability of tools facilitate the elaboration of more (complex) case studies in order to verify, improve and elaborate the presented methods for behaviour refinement. These case studies can also be used to verify the correspondence between the execution-based and causality-based methods.

Behaviour analysis

This thesis has focused on the development of basic concepts and methods that facilitate a designer to conceive, structure and refine system behaviours. It is important to spend attention in future research to the development of methods that facilitate the analysis of system behaviours. These methods should also be supported by automated tools. For example, tools could be developed to support syntax checking, semantical analysis, simulation, behaviour abstraction, validation of correctness relations (behaviour equivalencies) and performance analysis.

References

[1] M. Ajmone Marsan, A. Bianco, L. Ciminiera, R. Sisto and A. Valenzano, A LOTOS extension for the performance analysis of distributed systems, in: *IEEE/ACM Transactions on Networking* 2, 2 (1994) 151-165.

[2] J.W. de Bakker, W.-P. de Roever and G. Rozenberg, eds., *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS 354 (Springer-Verlag, Berlin, 1989).

[3] G.A. Blaauw and F.P. Brooks Jr., *Computer Architecture*, Lecture Notes, University of Twente, The Netherlands, and University of Carolina at Chapel Hill, U.S.A., 1985.

[4] T. Bolognesi and E. Brinksma, Introduction to the ISO specification language LOTOS, in: *Computer Networks and ISDN Systems* 14 (1987) 25-59.

[5] T. Bolognesi, D. de Frutos, R. Langerak and D. Latella, Correctness preserving transformations for the early phases of software development, in: [6] 161-180.

[6] T. Bolognesi, J. van de Lagemaat and C.A. Vissers, eds., *LOTOSphere: Software Development with LOTOS* (Kluwer Academic Publisher, Dordrecht, 1995).

[7] T. Bolognesi and F. Lucidi, LOTOS-like process algebras with urgent or timed interactions, in: K.R. Parker and G.A. Rose, eds., *Formal Description Techniques IV* (North-Holland, Amsterdam, 1992) 249-264.

[8] T. Bolognesi and F. Lucidi, A timed full LOTOS with time/action tree semantics, in: [62] 205-237.

[9] T. Bolognesi, F. Lucidi and S. Trigila, Converging towards a timed LOTOS standard, in: *Computer Standards & Interfaces* 16 (1994) 87-118.

[10] E. Brinksma, J.-P. Katoen, R. Langerak and D. Latello, Performance analysis and true concurrency semantics, in: [62] 309-337.

[11] E. Brinksma, J.-P. Katoen, R. Langerak and D. Latello, A stochastic causality-based process algebra, in: *The Computer Journal* 38, 7 (1995) 552-565.

[12] J.-P. Courtiat, R.C. de Oliveira and L. Andriantsiferana, Specification and validation of multimedia protocols using RT-LOTOS, in: *Proceedings of the 5th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems* (IEEE Computer Society Press, Los Alamitos, CA, 1995) 354-362.

[13] A. Danthine, G. Leduc and P. Wolper, eds., *Protocol Specification, Testing and Verification XIII* (North-Holland, Amsterdam, 1993).

[14] M. Diaz and R. Groz, eds., *Formal Description Techniques V*, IFIP Transactions C-10 (Elsevier Science Publishers B.V., Amsterdam, 1993).

[15] M. Fang, C.J. Ho-Stuart and H.S.M. Zedan, Specification of real-time probabilistic behaviour, in: [13] 143-157.

[16] L. Ferreira Pires, *Architectural notes: a framework for distributed systems development*, Ph.D. Thesis, University of Twente, Enschede, The Netherlands, 1994.

[17] L. Ferreira Pires and C.A. Vissers, Overview of the Lotosphere design methodology, in: *ESPRIT 1990, Conference Proceedings* (Kluwer Academic Publishers, Dordrecht 1990) 371-387.

[18] C. Fidge, A constraint-oriented real-time process calculus, in: [14] 363-378.

[19] H.M. Franken, A virtual test environment for business processes, in: ACM Bulletin of Special interest group on supporting group work 18, 1 (April 1997) 63-67.

[20] H.M. Franken, M.K. de Weger, D.A.C. Quartel and L. Ferreira Pires, On engineering support for business process modelling and redesign, in: *Business Process Re-engineering and Benchmarking. Proceedings of the International Workshop on Modelling Techniques*, Bordeaux, France, April 18-19, 1996, 81-98.

[21] H.M. Franken and M.K. de Weger, A modelling framework for capturing business process dynamics, Accepted for publication in: *Journal of Business Change and Re-engineering. The journal of corporate transformation*.

[22] H. Gaifman, Modelling concurrency by partial orders and nonlinear transition systems, in: [2] 467-488.

[23] R. van Glabbeek, S.A. Smolka, B. Steffen and C. Tofts, Reactive, generative, and stratified models of probabilistic processes, in: *Proceedings 5th Annual IEEE Symposium on Logic in Computer Science* (IEEE Computer Society Press, Los Alamitos, CA, 1992) 130-141.

[24] R. Gotzhein, Temporal logic and applications - a tutorial, in: *Computer Networks and ISDN Systems* 24 (1992) 203-218.

[25] R. Gotzhein, *Open distributed systems: on concepts, methods, and design from a logical point of view* (Vieweg, Wiesbaden, 1993).

[26] N. Götz, U. Herzog and M. Rettelbach, Multiprocessor and distributed system design: the integration of functional specification and performance analysis using stochastic process algebras, in: L. Donatiello and R. Nelson, eds., *Performance Evaluation of Computer and Communication Systems*, LNCS 729 (Springer-Verlag, Berlin, 1993) 121-146.

[27] J. Gunawardena, Causal automata I: Confluence ≡ {AND, OR} Causality, in: M.Z. Kwiatkowska, M.W. Shields and R.M. Thomas, eds., *Semantics for Concurrency* (Springer-Verlag, London, 1990) 137-155.

[28] J. Gunawardena, Geometric logic, causality and event structures, in: J.C.M. Baeten and J.F. Groote, eds., *Concur '91: Concurrency Theory*, LNCS 527 (Springer-Verlag, Berlin, 1991) 266-279.

[29] J. Gunawardena, Causal automata, in: *Theoretical Computer Science* 101 (1992) 265-288.

[30] H. Hansson and B. Jonsson, A calculus for communicating systems with time and probabilities, in: *Proceedings of 11th IEEE Real-Time Systems Symposium* (IEEE Computer Society Press, 1990) 278-287.

[31] ISO, Information Processing Systems - Open Systems Interconnection - Basic Reference Model, 1984, IS 7498.

[32] ISO, Information Processing Systems - Open Systems Interconnection - Transport Service Definition, 1986, IS 8072.

[33] ISO, Information Processing Systems - Open Systems Interconnection - LOTOS - a formal description technique based on the temporal ordering of observational behaviour, 1989, IS 8807.

[34] ISO, Information technology - Formal description of ISO 8072 in LOTOS, 1990, ISO/IEC/TR 10023.

[35] J.-P. Katoen, *Causal behaviours and nets*, Memoranda Informatica 94-70, University of Twente, 1994.

[36] J.-P. Katoen, Causal behaviours and nets, in: G. de Michelis and M. Diaz, eds., *Application and Theory of Petri Nets 1995*, LNCS 935 (Springer-Verlag, Berlin, 1995) 258-277.

[37] J.-P. Katoen, *Quantitative and qualitative extensions of event structures*, Ph.D. Thesis, University of Twente, Enschede, The Netherlands, 1996.

[38] J.-P. Katoen, R. Langerak and D. Latella, Modelling systems by probabilistic process algebra: an event structures approach, in: [70] 253-268.

[39] J.-P. Katoen, R. Langerak, D. Latella and E. Brinksma, On specifying real-time systems in a causality-based setting, in: B. Jonsson and J. Parrow, eds., *Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 1135 (Springer-Verlag, Berlin, 1996) 385-404.

[40] K. Kuutti, The concept of activity as a basic unit of analysis for CSCW research, in: L. Bannon, M. Robinson and K. Schmidt, eds., *Proceedings of the Second European Conference on Computer-Supported Cooperative Work* (Kluwer Academic Publishers, Dordrecht, 1991) 249-264.

[41] H. Kwakernaak and R. Sivan, *A primer on Signals and Systems. Part 1*, Lecture Notes, University of Twente, Enschede, 1986.

[42] L. Lamport, On interprocess communication. Part I: Basic formalism, in: *Distributed Computing* 1 (Springer-Verlag, Berlin, 1986) 77-85.

[43] R. Langerak, *Transformations and semantics for LOTOS*, Ph.D. Thesis, University of Twente, Enschede, The Netherlands, 1992.

[44] R. Langerak, Bundle event structures: a non-interleaving semantics for LOTOS, in: [14] 331-346.

[45] R. Langerak, E. Brinksma and J.-P. Katoen, Causal ambiguity and partial orders in event structures, in: A. Marzurkiewicz and J. Winkowski, eds., *Concur '97: Concurrency Theory*, LNCS 1243 (Springer-Verlag, Berlin, 1997) 317-331.

[46] G. Leduc and L. Léonard, A timed LOTOS supporting a dense time domain and including new timed operators, in: [14] 87-102.

[47] L. Léonard and G. Leduc, An enhanced version of timed LOTOS and its application to a case study, in: [70] 483-498.

[48] A. Mazurkiewicz, Basic notions of trace theory, in: [2] 285-363.

[49] C. Miguel, A. Fernández and L. Vidaller, LOTOS extended with probabilistic behaviours, in: *Formal Aspects of Computing* 5 (1993) 253-281.

[50] C. Miguel, L. Vidaller and A. Fernández, Extending LOTOS towards performance evaluation, in: [14] 103-118.

[51] C. Miguel, L. Vidaller and A. Fernández, Extended LOTOS, in: A. Danthine, ed., *The OSI95 Transport Service with Multimedia Support* (Springer-Verlag, Berlin, 1994) 312-337.

[52] X. Nicollin and J. Sifakis, An overview and synthesis on timed process algebras, in: J.W. de Bakker, C. Huizing, W.P. de Roever and G. Rozenberg, eds., *Real-Time: Theory in Practice*, LNCS 600 (Springer-Verlag, Berlin, 1992) 526-548.

[53] X. Nicollin and J. Sifakis, The algebra of timed processes, ATP: theory and application, in: *Information and Computation* 114 (1994) 131-178.

[54] D.L. Parnas, On the criteria to be used in decomposing systems into modules, in: *Communications of the ACM* 15, 12 (1972) 1053-1058.

[55] V. Pratt, Modelling concurrency with partial orders, in: *International Journal of Parallel Programming* 15, 1 (1986) 33-71.

[56] D.A.C. Quartel, L. Ferreira Pires, H.M. Franken and C.A. Vissers, An engineering approach towards action refinement, in: *Proceedings of the 5th IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems* (IEEE Computer Society Press, Los Alamitos, CA, 1995) 266-273.

[57] D.A.C. Quartel, L. Ferreira Pires, M.J. van Sinderen, H.M. Franken and C.A. Vissers, On the role of basic design concepts in behaviour structuring, in: *Computer Networks and ISDN Systems* 29 (1997) 413-436.

[58] J. Quemada and A. Fernandez, Introduction of quantitative relative time into LOTOS, in: H. Rudin and C.H. West, eds., *Protocol Specification, Testing and Verification VII* (North-Holland, Amsterdam, 1987) 105-121.

[59] J. Quemada, D. de Frutos and A. Azcorra, TIC: A TImed Calculus, in: *Formal Aspects of Computing* 5 (1993) 224-252.

[60] J. Quemada, C. Miguel, D. de Frutos and L. Llana, A timed LOTOS extension, in: [62] 239-263.

[61] E. Rechtin, The art of systems architecting, in: *IEEE Spectrum* (October 1992) 66-69.

[62] T. Rus and C. Rattray, eds., *Theories and experiences for real-time system development, AMAST Series in Computing* 2 (World Scientific, Singapore, 1994).

[63] J. Schot, *The role of Architectural Semantics in the formal approach of Distributed Systems Design*, Ph.D. Thesis, University of Twente, Enschede, The Netherlands,

1992.

[64] G. Scollo, *On the engineering of logics*, Ph.D. Thesis, University of Twente, Enschede, The Netherlands, 1993.

[65] H.A. Simon, The organization of complex systems, in: H.H. Pattee, ed., *Hierarchy theory. The challenge of complex systems* (George Braziller, New York, 1973) 3-27.

[66] M. van Sinderen, *On the Design of Application Protocols*, Ph.D. Thesis, University of Twente, Enschede, The Netherlands, 1995.

[67] M. van Sinderen, L. Ferreira Pires, C.A. Vissers and J.-P. Katoen, A design model for open distributed processing systems, in: *Computer Networks and ISDN Systems* 27 (1995) 1263-1285.

[68] R. Sisto, L. Ciminiera and A. Valenzano, Probabilistic characterization of algebraic protocol specifications, in: *Proceedings of the 12th International Conference on Distributed Computing Systems* (IEEE Computer Society Press, Los Alamitos, CA, 1992) 260-268.

[69] J.H.A. de Smit and E.A. van Doorn, *Kansrekening voor INF*, Lecture Notes, University of Twente, Enschede, 1986.

[70] R.L. Tenney, P.D. Amer, M.Ü. Uyar, eds., *Formal Description Techniques VI*, IFIP Transactions C-22 (North-Holland, Amsterdam, 1994).

[71] K.J. Turner, ed., *Using Formal Description Techniques. An Introduction to Estelle, LOTOS and SDL* (John Wiley & Sons, New York, 1993).

[72] C.A. Vissers, *The Design of Telematic Systems*, oral presentations, http://wwwtios.cs.utwente.nl/tios/dg/education/vakken/214005, 1995.

[73] C.A. Vissers, L. Ferreira Pires and J. van de Lagemaat, Lotosphere, an attempt towards a design culture, in: [6] 3-25.

[74] C.A. Vissers, L. Ferreira Pires and D.A. Quartel, *The Design of Telematic Systems*, Lecture Notes, University of Twente, Enschede, The Netherlands, 1995.

[75] C.A. Vissers, G. Scollo, M. van Sinderen and E. Brinksma, Specification styles in distributed systems design and verification, in: *Theoretical Computer Science* 89 (1991) 179-206.

[76] C.A. Vissers, M. van Sinderen and L. Ferreira Pires, What makes industries believe in formal methods, in: [13] 3-26.

[77] Y. Wand and R. Weber, On the ontological expressiveness of information system analysis and design grammars, in: *Journal of Information Systems* 3 (1993) 217-237.

[78] Y. Wang, Algebraic reasoning for real-time probabilistic processes with uncertain information, in: H. Langmaack, W.-P. de Roever, J. Vytopil, eds., *Formal techniques in real-time and fault-tolerant systems* , LNCS 863 (Springer-Verlag, Berlin, 1994) 680-693.

[79] Webster's, *Webster's Third New International Dictionary of the English Language*, Merriam-Webster, Springfield, MA, 1986.

[80] M.K. de Weger, *Structuring of business processes. An architectural approach to dis-*

tributed systems development and its application to business processes, Ph.D. Thesis, University of Twente, Enschede, The Netherlands, 1998.

[81] M.K. de Weger and C.A. Vissers, Issues in Design Methodologies for Distributed Information Systems, in: A.A. Verrijn-Stuart and T.W. Olle, *Methods and associated tools for the information systems life cycle*. *Proceedings of the IFIP WG8.1 Working Conference* (Elsevier Science Publishers, Amsterdam, 1994) 195-208.

[82] M.K. de Weger, H.M. Franken and C.A. Vissers, A development model for distributed information systems, in: Portugese Telecom, *Proceedings First International Distributed Conference*, *IDC'95*, Madeira, Portugal, 16-17 november, 1995, 13 pp.

Index

V

Index of functions and symbols

$I\text{-}Caus_a$	information causality conditions of action *a* 190
$I\text{-}Refs_a$	information reference relations of action *a* 190
$L\text{-}Caus_a$	location causality conditions of action *a* 211
$L\text{-}Refs_a$	location reference relations of action *a* 210
$T\text{-}Caus_a$	time causality conditions of action *a* 211
$T\text{-}Refs_a$	time reference relations of action *a* 210
$ITL\text{-}Caus_a$	mixed causality conditions of action *a* 209
$ITL\text{-}Refs_a$	mixed reference relations of action *a* 208
$L_{causality}$	notation of design model 102
$L'_{causality}$	pre-formal notation of design model 102
$L_{execution}$	notation of execution model 102
$\boldsymbol{A}$	domain of action names 39
$\underline{\boldsymbol{A}}$	domain of interaction names 42
$\boldsymbol{B}$	domain of causality-based behaviour definitions 56, 103
$\boldsymbol{BA}$	domain of alternative behaviour definitions 161
$\boldsymbol{BA_{basic}}$	domain of basic alternative behaviour definitions 161
$\boldsymbol{BA(a, b)}$	domain of alternative behaviour definitions of two actions *a* and *b* 135
$\boldsymbol{C}$	domain of elementary and conjunctive causality conditions 102
$\boldsymbol{CC}$	domain of (disjunctive) causality conditions 102
$\boldsymbol{CR}$	domain of causality relations 103
$\boldsymbol{E}$	domain of execution-based behaviour definitions 57
$\boldsymbol{Ent}$	domain of entity names 42
$\boldsymbol{EE}$	domain of partial execution-based behaviour definitions 71
$\boldsymbol{EP}$	domain of execution-based behaviour definitions extended with probability 93
$\boldsymbol{I}$	domain of information values 39
$\boldsymbol{L}$	domain of location values 39
$\boldsymbol{N}$	domain of natural numbers
$\boldsymbol{P}$	domain of integral probability values 221
$\boldsymbol{R}$	domain of real numbers
$\boldsymbol{T}$	domain of time values 39
$\boldsymbol{U}$	domain of uncertainty values 103
$\boldsymbol{X}$	domain of executions 56
$\boldsymbol{X'}$, $\boldsymbol{X}_\iota$, $\boldsymbol{X}_\tau$, $\boldsymbol{X}_\lambda$, $\boldsymbol{X}_{\iota\tau}$, $\boldsymbol{X}_{\iota\lambda}$, $\boldsymbol{X}_{\tau\lambda}$, $\boldsymbol{X}_{\iota\tau\lambda}$	sub-domains of $\boldsymbol{X}$ 85
$\boldsymbol{XX}$	domain of partial executions 71
$\boldsymbol{XX'}$, $\boldsymbol{XX}_\iota$, $\boldsymbol{XX}_\tau$, $\boldsymbol{XX}_\lambda$, $\boldsymbol{XX}_{\iota\tau}$, $\boldsymbol{XX}_{\iota\lambda}$, $\boldsymbol{XX}_{\tau\lambda}$, $\boldsymbol{XX}_{\iota\tau\lambda}$	sub-domains of $\boldsymbol{XX}$ 85
$\boldsymbol{Z}$	domain of discrete time values 261
χ, $\chi(Ac)$	execution (with action domain *Ac*) 56, 58, 84
γ	elementary, conjunctive or alternative causality condition 102
γ^+	(indirect) dependency closure of alternative causality condition γ 169
ι, ι_a, $\iota(a)$	information value of an action (*a*) 17, 40
φ	uncertainty association 103
λ, λ_a, $\lambda(a)$	location value of an action (*a*) 17, 40

π, π_E	probability function (associated with execution set *E*) 93
π_a, $\pi(a)$	simple integral probability attribute of action *a* 221, 226
$\pi_a(\tau)$, $\pi(a, \tau)$	simple stochastic probability attribute of action *a* 261
π^*_a, $\pi^*(a)$	extended integral probability attribute of action *a* 242, 244
$\pi^*_a(\tau)$, $\pi^*(a, \tau)$	extended stochastic probability attribute of action *a* 266
ρ, ρ_a, $\rho(a)$	causality relation of an action (*a*) 103
τ, τ_a, $\tau(a)$	time value of an action (*a*) 17, 40
υ, υ_a, $\upsilon(a)$	uncertainty attribute of an action (*a*) 102
Γ, Γ_a, $\Gamma(a)$	causality condition of an action (*a*) 102
I, I_a, I(*a*)	information value domain of an action (*a*) 39
Λ, Λ_a, $\Lambda(a)$	location value domain of an action (*a*) 39
T, T_a, T(*a*)	time value domain of an action (*a*) 39
Π_a, $\Pi(a)$	absolute probability of action *a* 221
ς ς_a, $\varsigma(a)$	mixed value domain of an action (*a*) 39
$<$, $<_\chi$	ordering relation of an execution (χ) 58
$=$, $=_\chi$	synchronization relation of an execution (χ) 58
$\sqsubseteq$	sub-execution relation 69
$\backslash$	restriction operator 69
$\backslash\backslash$	generalized restriction operator 69
$\vert$, $\vert_{e\chi}$	independence relation of a partial execution ($e\chi$) 71
$\wp$	power set
$\times$	product (conjunction) operator 75, 85
$\otimes$	cross-product (cross-conjunction) operator 75
$\vee$	*or*-operator 103, 117, 145, 152
$\wedge$	*and*-operator 117, 141, 143
$\surd$	start condition 101, 114
$\dagger$	impossible condition 131, 133
!	*must* uncertainty value 100
?	*may* uncertainty value 100
$[\![..]\!]$	execution semantics of .. 102
❑	*or*-operator on behaviours 132, 162
■	*and*-operator on behaviours 161
$\angle_\chi$	prefix relation on executions 93
✔	execution semantics completion operator 155
$\equiv$	(indirect) dependency equivalence relation 168, 169
$\vdash$	sub-condition relation 170

Summary

This thesis presents basic design concepts, design methods and a basic design language for distributed system behaviours. This language is based on two basic concepts: the action concept and the causality relation concept. Our methods focus on behaviour refinement, which consists of replacing an abstract behaviour by a more concrete behaviour, such that the concrete behaviour conforms to the abstract behaviour.

An important idea underlying this thesis is that an effective design methodology should be based on a properly chosen and precisely defined set of basic design concepts. Properly chosen design concepts represent essential system conceptions (mental images) that are derived from the real world and allow a designer to conceive and structure the essential characteristics of a system. The set of basic design concepts and their combination rules is called a *basic design model*. We explain how a design methodology supported by design notations and automated tools depends on the basic design model.

We introduce and motivate a limited set of basic design concepts that are necessary to design distributed systems. These concepts are structured into two related conceptual domains: the entity domain and the behaviour domain. This thesis focuses on the behaviour domain, which consists of the action concept, the interaction concept and the concept of causality relation. Therefore, we elaborate the action and interaction concepts in more detail and give a formal definition of these concepts. The elaboration of the causality relation concept comprises the main part of this thesis. In order to enable a systematic and modular development of the causality relation concept, we identify the important characteristics of relations between actions and structure these characteristics in an abstraction hierarchy.

An *action* models the essential characteristics of a unit of activity that is performed by a single entity. We consider the following characteristics of an activity as essential: the result that is established by the activity, the moment at which the activity is finished and makes its total result available, and the location at which this result is made available. These characteristics are modelled by means of the information, time and location attributes of an action, respectively. We consider an interaction as a refinement of an action, which models how an activity is performed through the cooperation of multiple entities.

A *causality relation* defines one or more alternative conditions for the occurrence of an action in terms of how this action depends on the occurrences or non-occurrences of other actions. An action occurrence is caused by (or depends on) only one of its alternative conditions, although multiple of these conditions can be satisfied at the same time. We consider the uncertainty or probability that an action occurs when one (or more) of its alternative conditions are satisfied as an important concept in the design of relations between activities. This concept is represented by the probability attribute, which defines, for each alternative

condition of an action, the probability that the action occurs when this condition is satisfied. We distinguish three types of probability attributes: (i) the uncertainty attribute supports two uncertainty values: *must* and *may*, (ii) the integral probability attribute quantifies these uncertainty values, such that the *must* value corresponds to probability value 1, and the *may* value corresponds to a probability value in the range (0..1), and (iii) the stochastic probability attribute uses the time attribute of an action as a stochastic variable, such that a probability distribution function defines for the time period in which the action is allowed to occur, the probability that the action actually occurs.

We start with an initial definition of the causality relation concept that supports the design of temporal ordering relations between actions, including the uncertainty attribute. Four elementary causality conditions are defined: the start condition, the enabling condition, the disabling condition and the synchronization condition. These elementary conditions can be composed into more complex causality conditions using the conjunction (*and*-) and disjunction (*or*-) operators. The disjunction operator is used to define multiple alternative causality conditions for an action. The uncertainty attribute defines, for each of these alternative conditions, whether the action must or may occur when this condition is satisfied.

The initial definition of the causality relation concept is extended with the information, location and time attribute. This extension supports the design of the following type of constraints for each of these attributes: (i) the range of possible values that can be established in an action, (ii) how the value of an action depends on the values established in other actions, and (iii) how the occurrence of an action depends on the values established in other actions. Constraints involving different attribute types are also allowed, e.g., the time and location value established in an action may be referred to as information values by another action.

The integral and stochastic probability attribute can be used instead of the uncertainty attribute to quantify the uncertainty of action occurrences. Two interpretations of these probability attributes are distinguished: (i) the simple interpretation defines for each alternative condition of an action the probability that the action occurs when this condition is satisfied, and (ii) the extended interpretation defines for each alternative condition of an action the probability that the occurrence of the action is caused by this condition once this condition enables the action. The extended interpretation allows one to model the probability of individual actions in, e.g., choice, disabling and interleaving relations.

In order to define the formal semantics of causality relations, a so called execution model is introduced. In this model, a behaviour is defined by enumerating all possible executions of this behaviour. An execution represents the outcome of a possible run of a system that performs a specified behaviour. This outcome comprises the actions that have occurred, the information, time and location values that have been established in these actions, and how action occurrences are related in the particular execution. An execution also gets one or more probability values, which represent the probability that this execution is the outcome of a system run. In this respect, a behaviour is considered an experiment and an execution is considered a possible outcome of this experiment. The sum of the probability of all possible executions of a behaviour is equal to 1.

Based on the basic design language, we present an integrated set of methods to perform behaviour refinement. These methods support two basic types of behaviour refinement:

causality refinement, in which causality relations between abstract actions are replaced by causality relations involving their corresponding concrete actions and some inserted actions, and *action refinement*, in which an abstract action is replaced by an activity involving multiple concrete actions and their causality relations. The methods are based on the assessment of the conformance relation between the abstract behaviour and the concrete behaviour that is obtained from the abstract behaviour by means of causality refinement or action refinement. This assessment involves the determination of the abstraction of the concrete behaviour and the comparison of this abstraction with the original abstract behaviour. Rules to perform the abstraction and comparison operations have been developed.

In this thesis we extend the basic design language with the causality-oriented structuring technique defined in [16]. This technique allows one to structure a complex behaviour in terms of simpler sub-behaviours and their relationships. In order to model (infinitely) repetitive behaviours, this technique is extended with the means to (dynamically) create multiple instances of a single sub-behaviour (type) definition, including the means to refer unambiguously to each individual behaviour instance.

The ideas presented in this thesis are applied to two case studies. We apply our behaviour refinement method to the design of a system that supports a client-server interaction. At the highest abstraction level we assume that direct interactions between the client application and the server application are possible. At a lower abstraction level we implement these interactions using a federation of remote traders, which communicate via a common communication infrastructure. We also apply our basic design language to the modelling of the behaviour of the OSI Connection-oriented Transport Service. This case study also includes the modelling of timing and probability characteristics imposed by the QoS parameters of the transport service.

Samenvatting

Dit proefschrift presenteert basisontwerpconcepten, ontwerpmethoden en een basisontwerptaal voor gedragingen van gedistribueerde systemen. De basisontwerptaal is gebaseerd op twee basisconcepten: het actie-concept en het causaliteitsrelatie-concept. Onze methoden richten zich op gedragsverfijning, waarin een abstract gedrag wordt vervangen door een meer concreet gedrag, zodanig dat het concrete gedrag conformeert aan het abstracte gedrag.

Een belangrijk idee dat ten grondslag ligt aan dit proefschrift, is dat een effectieve ontwerpmethodologie moet worden gebaseerd op een verzameling geschikte en precies gedefinieerde basisontwerpconcepten. Geschikte ontwerpconcepten representeren essentiële systeembegrippen (mentale beelden) die zijn afgeleid van de reële wereld en die een ontwerper in staat stellen om essentiële systeemkenmerken te concipiëren en te structureren. Deze verzameling van basisontwerpconcepten en hun compositieregels worden een *basisontwerpmodel* genoemd. We leggen uit hoe een ontwerpmethodologie, ondersteund door ontwerpnotaties en geautomatiseerde gereedschappen, afhangt van het basisontwerpmodel.

We introduceren en motiveren een beperkte verzameling van basisontwerpconcepten die nodig is voor het ontwerpen van gedistribueerde systemen. Deze concepten zijn gestructureerd in twee gerelateerde conceptuele domeinen: het entiteitendomein en het gedragsdomein. Dit proefschrift richt zich op het gedragsdomein, dat bestaat uit het actie-concept, het interactie-concept en het causaliteitsrelatie-concept. Dientengevolge worden het actie- en interactie-concept in meer detail uitgewerkt en formeel gedefinieerd. De ontwikkeling van het causaliteitsrelatie-concept omvat het hoofddeel van dit proefschrift. Om een systematische en modulaire uitwerking van het causaliteitsrelatie-concept mogelijk te maken, identificeren we eerst de belangrijke kenmerken van relaties tussen acties en structureren deze kenmerken in een abstractiehierarchie.

Een *actie* modelleert de essentiële kenmerken van een eenheid van activiteit die wordt uitgevoerd door één enkele entiteit. We beschouwen de volgende kenmerken van een activiteit als essentieel: het resultaat dat wordt opgeleverd door de activiteit, het tijdsmoment waarop de activiteit gereed is en het totale resultaat beschikbaar stelt, en de locatie waar het resultaat beschikbaar is. Deze kenmerken worden respectievelijk gemodelleerd met behulp van het informatie-, tijds- en locatieattribuut. We beschouwen een interactie als een verfijning van een actie, die modelleert hoe een activiteit gezamenlijk wordt uitgevoerd door meerdere entiteiten.

Een *causaliteitsrelatie* definieert een of meerdere alternatieve condities voor het optreden van een actie in termen van hoe deze actie afhangt van het wel of niet optreden van andere acties. Het optreden van een actie wordt veroorzaakt door (is afhankelijk van) slechts één van zijn alternatieve condities, ofschoon aan meerdere van deze condities tegelijkertijd kan

worden voldaan. We beschouwen de onzekerheid of waarschijnlijkheid waarmee een actie optreedt wanneer aan een (of meerdere) van zijn alternatieve condities is voldaan als een belangrijk concept voor het ontwerpen van relaties tussen activiteiten. Dit concept wordt gerepresenteerd door het waarschijnlijkheidsattribuut, dat voor elke alternatieve conditie van een actie de waarschijnlijkheid definieert waarmee de actie optreedt wanneer aan de conditie is voldaan. We onderscheiden drie soorten waarschijnlijkheidsattributen: (i) het onzekerheidsattribuut kent twee onzekerheidswaarden: '*must*' en '*may*', (ii) het integrale waarschijnlijkheidsattribuut kwantificeert deze onzekerheidswaarden, zodanig dat de waarde *must* correspondeert met de waarschijnlijkheidswaarde 1, en de waarde *may* correspondeert met een waarschijnlijkheidswaarde in het interval (0..1), en (iii) het stochastische waarschijnlijkheidsattribuut gebruikt het tijdsattribuut als een stochastische variabele, zodanig dat een verdelingsfunctie voor de tijdsperiode waarin de actie kan optreden de waarschijnlijkheid definieert dat de actie ook daadwerkelijk optreedt.

We beginnen met een initiële definitie van het causaliteitsrelatie-concept welke het ontwerpen van temporele orderingsrelaties ondersteunt, inclusief het onzekerheidsattribuut. Vier elementaire causaliteitscondities worden gedefinieerd: de start conditie, de 'enabling' conditie, de 'disabling' conditie en de 'synchronization' conditie. Deze elementaire condities kunnen worden samengesteld tot meer complexe causaliteitscondities met behulp van de conjunctie (en-) en disjunctie (or-) operatoren. De disjunctie operator wordt gebruikt om meerdere alternatieve condities voor een actie te definiëren. Het onzekerheidsattribuut definieert, voor elk van deze alternatieve condities, of een actie moet of mag optreden wanneer aan deze conditie is voldaan.

De initiële definitie van het causaliteitsrelatie-concept wordt uitgebreid met het informatie-, locatie- en tijdsattribuut. Deze uitbreiding ondersteunt het ontwerpen van de volgende soorten van condities voor elk van de attributen: (i) de verzameling van mogelijke waarden die kunnen worden vastgesteld in een actie, (ii) hoe de waarde van een actie afhangt van de waarden die worden vastgesteld in andere acties, en (iii) hoe het optreden van een actie afhangt van de waarden die worden vastgesteld in andere acties. Condities die betrekking hebben op meerdere soorten attributen zijn ook toegestaan; bijvoorbeeld, de tijds- en locatiewaarde die worden vastgesteld in een actie kunnen worden beschouwd als informatiewaarden wanneer een andere actie hiernaar refereert.

Het integrale en stochastische waarschijnlijkheidsattribuut kunnen worden gebruikt in plaats van het onzekerheidsattribuut om de onzekerheid waarmee acties optreden te kwantificeren. Twee interpretaties van deze waarschijnlijksattributen worden onderscheiden: (i) de simpele interpretatie definieert voor elke alternatieve conditie van een actie de waarschijnlijkheid dat de actie optreedt wanneer aan deze conditie wordt voldaan, en (ii) de uitgebreide interpretatie definieert voor elke alternatieve conditie van een actie de waarschijnlijkheid dat het optreden van de actie wordt veroorzaakt door deze conditie wanneer deze conditie het optreden van de actie mogelijk maakt. De uitgebreide interpretatie maakt het mogelijk om de waarschijnlijkheid van individuele acties in, bijvoorbeeld, keuze, uitsluitings- en 'interleaving' relaties te modelleren.

Om de formele semantiek van causaliteitsrelaties te definiëren wordt een zogenaamd executiemodel geïntroduceerd. In dit model wordt een gedrag gedefinieerd door alle mogelijke

executies van dit gedrag op te sommen. Een executie representeert het resultaat van een mogelijke uitvoering van het gespecifieerde gedrag door een systeem. Dit resultaat omvat de acties die optreden, de informatie-, tijds- en locatiewaarden die worden vastgesteld in deze acties, en hoe de optredens van de acties aan elkaar zijn gerelateerd. Een executie krijgt ook een of meerdere waarschijnlijkheidswaarden, die de waarschijnlijkheid representeren waarmee deze mogelijke executie van het gedrag wordt uitgevoerd door het systeem. Een gedrag kan dan ook worden beschouwd als een experiment en een executie kan worden beschouwd als een mogelijke uitkomst van dit experiment. De som van de waarschijnlijkheidswaarden van alle mogelijke executies van een gedrag is gelijk aan 1.

Gebaseerd op de basisontwerptaal, presenteren we een geïntegreerde verzameling van methoden voor gedragsverfijning. Deze methoden ondersteunen twee basisvormen van gedragsverfijning: *causaliteitsverfijning*, waarin causaliteitsrelaties tussen abstracte acties worden vervangen door causaliteitsrelaties van de corresponderende concrete acties en een of meer ingevoegde acties, en *actieverfijning*, waarin een abstracte actie wordt vervangen door een activiteit bestaande uit meerdere concrete acties en hun causaliteitsrelaties. De methoden zijn gebaseerd op het vaststellen van de ‘conformance’ relatie tussen het abstracte gedrag en het concrete gedrag dat is verkregen vanuit het abstracte gedrag door middel van causaliteitsverfijning of actieverfijning. Hiervoor is het nodig om de abstractie van het concrete gedrag te bepalen en deze te vergelijken met het originele abstracte gedrag. Regels voor het bepalen van de abstractie en het maken van de vergelijking worden ontwikkeld.

In dit proefschrift breiden we de basisontwerptaal uit met de causaliteitsgeoriënteerde structureringstechniek zoals deze is gedefinieerd in [16]. Deze techniek maakt het mogelijk om een complex gedrag te structureren in termen van simpelere sub-gedragingen en hun onderlinge relaties. Om het modelleren van zich (oneindig) herhalende gedragingen mogelijk te maken, wordt deze techniek uitgebreid met de middelen om (dynamisch) meerdere instanties van één enkele gedrags(type)definitie te creëren, inclusief de middelen om op ondubbelzinnige wijze te refereren naar elke individuele gedragsinstantie.

De ideeën gepresenteerd in dit proefschrift worden toegepast op twee casussen. Onze methode voor gedragsverfijning wordt toegepast op het ontwerpen van een systeem dat een ‘client-server’ interactie ondersteunt. Op het hoogste abstractieniveau veronderstellen we dat directe interacties tussen de client- en serverapplicatie mogelijk zijn. Op een lager abstractieniveau implementeren we deze interacties gebruikmakend van een federatie van gedistribueerde ‘traders’, die communiceren via een gemeenschappelijke communicatie-infrastructuur. Onze basisontwerptaal passen we toe op het modelleren van het gedrag van de OSI ‘Connection-oriented Transport Service’. In deze casus modelleren we onder meer de tijds- en waarschijnlijkheidskenmerken die worden opgelegd door de ‘Quality of Service’ parameters van de transport service.