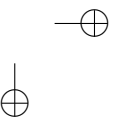
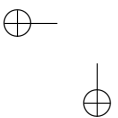


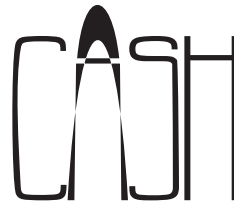
Tomas Krilavičius

Hybrid Techniques for Hybrid Systems



Graduation committee:

Prof. Dr. Ir. A.J. Mouthaan	University of Twente, The Netherlands
Prof. Dr. H. Brinksma (promotor)	University of Twente / Embedded Systems Institute, The Netherlands
Dr. Ir. R. Langerak (assistent-promotor)	University of Twente, The Netherlands
Prof. Dr.-Ing. S. Kowalewski	RWTH Aachen University, Germany
Prof. Dr. K.G. Larsen	Aalborg University, Denmark
Dr. J.W. Polderman	University of Twente, The Netherlands
Prof. Dr. A.J. van der Schaft	University of Groningen, The Netherlands
Prof. Dr. F.W. Vaandrager	Radboud University Nijmegen, The Netherlands



IPA Dissertation Series 2006-15.

CTIT Ph.D.-Thesis Series No. 06-90, ISSN 1381-3617.

The research reported in this dissertation has been carried out under the auspices of the Institute for Programming Research and Algorithmics (IPA) and within the context of the Centre for Telematics and Information Technology (CTIT). The funding of the research was provided by the NWO Grant through project number 617.023.002 (Compositional Analysis and Specification of Hybrid Systems).

Translation of summary: A. Nijmeijer and R. Langerak.

Typeset by L^AT_EX.

Cover: A. Jonkuté.

Printed: Febodruk - <http://www.febodruk.nl>.

Copyright © 2006 by T. Krilavičius, Enschede, The Netherlands.

ISBN: 90-365-2397-4







HYBRID TECHNIQUES FOR HYBRID SYSTEMS

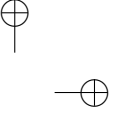
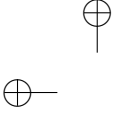
DISSERTATION

to obtain the doctor's degree
at the University of Twente, on the authority of
the rector magnificus, Prof. Dr. W.H.M. Zijm,
on account of the decision of the graduation committee
to be publicly defended
on Wednesday, September 6, 2006 at 15:00

by

Tomas Krilavičius
born on 14 February 1974
in Alytus, Lithuania





The dissertation is approved by:

Prof. Dr. H. Brinksma (promotor)



Acknowledgements

I am thankful to many people for their help and support during my work on the thesis.

First of all I would like to thank my promotor Ed Brinksma for guiding me through the not always serene waters of research. I also wish to thank my advisor Rom Langerak who has helped me to acclimatise in the Formal Methods group and the world of hybrid systems.

I would like to thank my former supervisors Valentinas Kriaučiukas and Henrikas Pranevičius for introducing me to the formal methods and academia.

I thank the members of my graduation committee Stefan Kowalewski, Kim Larsen, Jan Willem Polderman, Frits Vaandrager, and Arjan van der Schaft for assessing my manuscript and giving valuable comments. I also received useful comments from Henrik Bohnenkamp, Germanas Budnikas, Mariëlle Stoelinga, Ivan S. Zapreev and Rasa Zulytė. Moreover, I thank my master students Arend van Putten and Helen Schonenberg for asking the right questions and in that way improving my dissertation.

I would like to thank our secretaries Ellen Roberts-Tieke and Joke Lammerink. I am indebted to Joke, who helped me to survive my first days in Enschede and made my stay in the Netherlands a lot easier.

I had the pleasure to participate in the EU Ametist project, where I have met a lot of nice people. I would like to thank all of them, especially Biniam Gebremichael, Angelika Mader, Sebastian Panek and Gera Weiss.

I am member of an hybrid systems conspiracy called the miniCASH. I would like to thank Agung Julius, Rajashekar Kakumani, Rom Langerak and Stefan Strubbe for fruitful discussions during our scientific lunch meetings.

I thank “moksliukai” Erinija Pranckevičienė, Gailius Raškinis, Aušra Saudargienė and Minija Tamošiūnaitė for keeping me in touch with science in Lithuania.

Thanks to Axel Belinfante, Machiel van der Bijl, Laura Brandà Briones, Holger Hermanns, David N. Jansen, Joost-Pieter Katoen, Piotr Kordy, Marcos E. Kurban, Andre Nijmeijer, Arend Rensink, Theo Ruys, Pedro D’Argenio, Conrado Daws, Ric Klaren, Patrick Sathyanathan, Rajasekhar Kakumani, István Nagy, Joost Noppen, Yaroslav S. Usenko, Gebremichael Biniam and others for being nice colleagues and sharing office space, hallway and Fridays in Rappa with me.

I want to thank my friends, who made my life pleasant in Enschede: Ivan S. Zapreev, Mass S. Lund, Suzana Andova, Yaroslav S. Usenko, Andre Nijmeijer, Laura Brandà Briones, Liudvika Leišytė, Žygimantas Medelis and Ivan and Lesia Krechetov.

I could seldom meet my Lithuanian friends, but we have kept in touch over all these years: Aurimas Švedas, Irmantas Švedas, Airida Rekšytė, Laimonas Butkus, Linas Vaitulevičius, Edita Mockutė, Danas Kazlauskas, Redas Kazlauskas and Jonas Sergejenka.

Aš dėkoju šeimos nariams ir giminėms, kurie man padėjo studijose ir gyvenime,

ACKNOWLEDGEMENTS

ypač tėvams Antanui Krilavičiui ir Jūratei Krilavičienei, broliui Kęstui Krilavičiui, Magdutei ir Alfonsui Raškiniams, Aldonai Valaitytei, Daliai Jenčiuvienei, bei puseserei Laimai Zubrienei (Blaškevičiūtei).

Thanks to Agnė Jonkutė just for being with me.

Contents

Acknowledgements	v
Table of Contents	vii
1 Introduction	1
1.1 Hybrid systems	2
1.2 Major problems in the area of hybrid systems	3
1.2.1 Modelling of hybrid systems	3
1.2.2 Analysis of hybrid systems	3
1.2.3 Deployment of hybrid systems' models	5
1.2.4 Testing of hybrid systems models	6
1.3 Main results	6
1.4 Outline of the dissertation	7
2 Bestiarium of hybrid systems	11
2.1 Introduction	11
2.2 Examples of hybrid systems	11
2.2.1 A bouncing ball	12
2.2.2 A thermostat	13
2.2.3 A leaking gas burner	14
2.2.4 A fluid level controller	15
2.2.5 Railroad gate control	17
2.2.6 Batch plant control	18
2.2.7 Mobile vehicles	21
2.3 Conclusions	23
3 Overview of models for hybrid systems	25
3.1 Introduction	25
3.2 Classification of hybrid systems	25
3.3 Hybrid formalisms	30
3.3.1 Grouping hybrid formalisms	30
3.3.2 Piecewise affine systems	32
3.3.3 Mixed logical dynamical systems	33
3.3.4 Complementarity systems	34
3.3.5 Max-min-plus-scaling systems	35
3.3.6 Hybrid automata	36
3.3.7 Hybrid behavioural automaton	37
3.3.8 Hybrid input/output automata	38

TABLE OF CONTENTS

3.3.9	Process algebras for hybrid systems	40
3.3.10	Masaccio	46
3.3.11	Charon	47
3.3.12	Bond graphs	47
3.3.13	Modelica	49
3.4	Conclusions	50
4	Stability analysis for hybrid automata	51
4.1	Introduction	51
4.1.1	Stability	51
4.1.2	Hybrid stability	52
4.2	Notions of stability and hybrid stability	54
4.2.1	Stability of dynamical systems	54
4.2.2	Stability of hybrid automata	54
4.3	Estimating stability of hybrid automaton	55
4.3.1	Contractive cycles and stability of hybrid automaton	56
4.3.2	Gain automata and algorithm	57
4.3.3	Stability of two-dimensional linear continuous hyperplane hybrid automaton	60
4.4	Conservative estimation of gains	60
4.4.1	Gains	61
4.4.2	Calculation of gains	62
4.4.3	Optimising the Lyapunov function choice	63
4.5	Conclusions	64
5	Behavioural Hybrid Process Calculus	65
5.1	Introduction	65
5.2	Behavioural approach	67
5.3	Trajectories	68
5.4	Hybrid transition systems	75
5.4.1	Bisimulation	76
5.5	Language and operational semantics	77
5.5.1	Language	77
5.5.2	Operational semantics of BHPC	79
5.5.3	Consistent signal flow	83
5.5.4	Congruence property	84
5.6	Expansion law	84
5.7	Derived BHPC operators	85
5.7.1	Parametrisation of action prefix	85
5.7.2	Idling	85
5.7.3	Delays	86
5.7.4	Guard	86
5.8	Application of BHPC	86
5.8.1	Bouncing ball	86
5.8.2	Thermostat	87
5.8.3	Dry friction	88
5.8.4	Two tanks	89

5.9	An experimental version of calculus	90
5.10	Conclusions	91
6	BHPC in context of related frameworks	93
6.1	Introduction	93
6.2	BHPC and hybrid dynamical systems	93
6.3	BHPC and transition systems based approaches	94
6.3.1	Hybrid automata and BHPC	96
6.4	BHPC and simulation languages	98
6.5	Conclusions	98
7	Simulation of Behavioural Hybrid Process Calculus	99
7.1	Introduction	99
7.1.1	Simulation of continuous and discrete systems	100
7.1.2	Simulation of hybrid systems	101
7.2	Behavioural Hybrid Process Calculus simulation algorithm	102
7.2.1	Language	102
7.2.2	Simulation of process algebras	103
7.2.3	Abstract simulation algorithm for BHPC	104
7.2.4	Transformation to normal form	106
7.2.5	Simulating discrete events	110
7.2.6	Simulating continuous-time behaviour	111
7.3	Non-determinism	115
7.4	Visualisation of models	117
7.5	Visualisation of results	118
7.5.1	Graphs	119
7.5.2	Event traces and message sequence charts	119
7.5.3	Combined view	120
7.5.4	Visualisation of components	123
7.6	Simulation of Zeno behaviour	124
7.7	Architecture	124
7.8	Simulation modes	128
7.9	Tools overview	129
7.10	Conclusions	133
8	Concluding remarks	135
8.1	Hybrid systems	135
8.2	Modelling of hybrid systems	135
8.3	Analysis of hybrid systems	137
8.4	General remarks	138
A	Stability	141
A.1	Proofs from Section 4.3	141
A.2	Optimising the Lyapunov function choice	142

TABLE OF CONTENTS

B Proofs from Chapter 5	145
B.1 Proof of Theorem 5.5.4	145
B.2 Proof of Theorem 5.5.6	147
B.2.1 Formats based proof	150
B.3 Proofs of Theorems 5.6.2 and 5.6.3	150
B.3.1 Proof of Theorem 5.6.2	150
B.3.2 Proof of Theorem 5.6.3	153
C Functions from Chapter 7	157
D Bhave prototype	161
D.1 Functionality and input language	161
D.1.1 Simplified treatment of parameters	163
D.2 Technical implementation details	164
D.3 Examples	164
D.4 Conclusions	166
Bibliography	167
Index	183
Summary	189
Samenvatting	191

All of the true things I am about to tell you are
shameless lies.

Kurt Vonnegut, Jr.

1

Introduction

The Industrial Revolution was the major technological, socio-economical and cultural change in the late XVIIIth and the early XIXth century, when an economy based on manual labour was replaced to one dominated by industry and machine manufacture. It was one of the principal factors that shaped the Western Civilisation as we know. The Digital Revolution is reshaping our lives now. Digital devices change people's everyday life and the way businesses operate, by bringing a higher level of automation. Such changes may be convenient and beneficial, but they make us vulnerable and dependent too.

One of the products of the Digital Revolution is *embedded systems* that are a special-purpose computer systems completely encapsulated by the device it controls. We encounter such systems everywhere in our life. Our day usually starts with an electrical alarm clock. We wash in water warmed using a digitally controlled kettle, our heating is controlled by a digital thermostat. We drink coffee produced by a coffee machine that has basic digital controls. Most means of transportation that help us to reach our workplace, like cars, buses, trains are full of digital controllers. Entrance sensor-controlled doors at the office are awaiting for us, and then an elevator. The computers, normal and cellular phones, printers, scanners, faxes and again coffee machines are part of everyday office life. In the evening to get dinner we use microwaves or cookers. And then a remote control for TV, VCR, DVD or an integrated audio/video system.

We expect all these devices to function properly at any time we need them. While the malfunctioning of TVs, audio systems, refrigerators or microwaves just irritates us or leaves us hungry, the glitches in the car, air-plane, nuclear plant control systems may threaten life, and faults in a nuclear missile control facility may bring the end to western civilisation. It is easy to see, why it is crucial to have means to make such systems function properly.

Research in *systems and control theory* provides engineers and scientists with tools

to solve the technological problems brought by the Industrial Revolution. *Computer science* aims at providing techniques for the general use of computers. Embedded systems is the place where these two worlds meet. The ambition to design the means for the development of flawless and robust embedded systems brings together the control and computer science communities. *Formal analysis* is one of the methods in computer science, which helps to gain confidence in such systems.

Formal methods provide rigorous mathematically-based languages, techniques and automated tools for modelling and analysis of software and hardware systems. There is a number of examples of the effective use of such techniques [Clarke and Wing, 1996]. Successful examples of application in consumer electronics [Havelund et al., 1999, Bengtsson et al., 2002], diverse communication and control protocols [David and Yi, 2000], automotive industry [Lindahl et al., 2001, Gebremichael et al., 2004], chemical industry [Mader et al., 2001, Bohnenkamp et al., 2004], heavy industry [Fehnker, 1999] and other areas are easy to find.

In formal methods embedded systems are often modelled and analysed as *hybrid systems*, the class of dynamical systems that combine *continuous evolution* and *discrete transitions*.

1.1 Hybrid systems

Hybrid systems combine continuous real-time behaviour and discrete events. Discrete events are caused by the evolution of continuous dynamics or external stimuli, and the continuous dynamics change in response to discrete events. Often such systems arise from the combination of an analogue continuous-time process and a digital controller, common in nowadays consumer electronics and other embedded systems. Of course, it may be just physical systems as electrical circuits with diodes and transistors, mechanical systems with collisions or friction models with stick and slip phases.

Several examples of systems with hybrid phenomena manifestations:

- Consumer systems: (cellular) phones, TV sets, microwaves.
- Traffic control systems: Air Traffic Management, (Sea) Port Traffic Management, Highway Supervision.
- Production process control and robotics: chemical industry, energy (production and distribution), food industry.
- Biological systems: the cell-cycle control system.

A list of hybrid systems examples from the scientific literature is provided in Chapter 2.

Almost any system can be modelled using some hybrid formalism, therefore it does not make sense to strive for a very general formal definition. A particular definition of hybrid systems can be adopted depending on potential use of it. In Chapter 3 some of them are presented. Introductions to hybrid systems are available in van der Schaft and Schumacher [2000], De Schutter and Heemels [2004].

1.2 Major problems in the area of hybrid systems

Essentially, research in hybrid systems aims at providing means for easy and reliable design and production of hybrid systems. At a closer look, it can be roughly grouped into several topics:

Modelling of hybrid systems is the process of creating an abstract model that uses mathematical concepts to describe the behaviour of a system. In some cases it is a transformation from a vague, natural language description to a rigorous mathematical model of system. Consequently, it includes accumulation and analysis of requirements and restrictions (e.g., physical, economical or even moral), an abstraction from not so important details, a choice of formalism, etc. But equally often an existing system or design are modelled formally for further analysis.

Analysis of hybrid systems includes checking consistency of requirements and restrictions, proving or refuting properties of a model, analysis of model behaviour, performance analysis.

Deployment of hybrid systems models includes choice of the final design of the product, and the *controller generation*, *code generation*, and other production related activities.

Testing is a process used to assess the quality and correctness of system. The *system under test* is fed input data while the output data is monitored for checking its correctness.

1.2.1 Modelling of hybrid systems

Modelling of hybrid system is the process of creating an abstract model that uses mathematical language to describe the behaviour of a system. In some cases it is a transformation from a vague, natural language description to a rigorous mathematical model of a system. Frequently the requirements should be collected and refined, physical, economical and other restrictions accumulated and estimated. Depending on the modelling goals, different formalisms can be chosen and different abstractions applied. However, equally often the model is derived from an existing system or design. Moreover, several models have been created and analysed to obtain interesting properties of the system.

1.2.2 Analysis of hybrid systems

Analysis of hybrid systems includes several different activities and various analysis methods, e.g., *verification* and *simulation*. Good sources for information about analysis of hybrid systems are van der Schaft and Schumacher [2000], De Schutter and Heemels [2004], the proceedings of *International Hybrid Systems Workshops* [Grossman et al., 1993, Antsaklis et al., 1995, Alur et al., 1996b, Antsaklis et al., 1997, 1999] and the proceedings of *International Workshops on Hybrid Systems: Computation and Control* [Henzinger and Sastry, 1998, Vaandrager and van Schuppen, 1999, Lynch and Krogh, 2000, Benedetto

and Sangiovanni-Vincentelli, 2001, Tomlin and Greenstreet, 2002, Maler and Pnueli, 2003, Alur and Pappas, 2004].

Verification of hybrid systems

(*Formal*) *Verification* of hybrid systems is the act of proving or refuting the correctness of a system with respect to a certain formal specification or property. The process of verification usually consists of several parts. A formal model of the system, containing the possible behaviour of it, is constructed. Moreover, the requirements are formulated in the formal language. Then, a set of rules is applied to determine whether the formal model satisfies requirements.

Verification can be employed to analyse a wide set of various properties.

- Usually properties originating from computer science, as *deadlocks* [van der Schaft and Schumacher, 2000, p.10], *fairness* [Katoen, 1999, p.232–233], *reachability* [van der Schaft and Schumacher, 2000, p.111-118] are analysed using verification. More generally they are referred to as *safety* and *liveness* properties.
- Characteristics taking roots in the systems and control theory, e.g., *stability* (Chapter 4), *controllability*, *observability* are important as well (see Polderman and Willems [1998]).

The term *verification* is usually used with automatic or semi-automatic procedures in mind. In general, the properties can be proved or refuted with pencil and paper, often following some well-defined method or procedure. However, for bigger systems, manual methods are impractical, if not impossible.

Model checking is one of the prevalent verification methods. It strives to explore the full state space while examining satisfiability of interesting properties. It has the following benefits [Katoen, 1999, p.34–37].

- Generality of the approach. It is applicable to hardware verification, software engineering, multi-agent systems, communication protocols, embedded systems and so forth.
- Partial verification is possible, the set of properties to be verified can be arbitrarily chosen.
- Ease of use, i.e., model checkers can be used almost as easily as compilers.
- Sound mathematical foundations.

Unfortunately, model checking suffers from several drawbacks [Katoen, 1999, p.34–37].

- It is more suitable for the analysis of *control oriented* applications, and less suited to data-intensive applications, since the treatment of data usually implies an infinite state space.
- Some classes of systems have *decidability* issues, because more often than not it is hard to come up with a finite abstraction of an infinite state space.
- Only the model is verified, not the real system.

- Only stated requirements are investigated.

However, even taking all these drawbacks into account, verification of hybrid systems provides a good insight on safety and performance properties of system.

At the same time, we do not reject semi-automatic approaches, like automatic *theorem proving* [Clarke and Wing, 1996, Kaufmann et al., 2000, Owre and Shankar, 2003], which can be very effective when dealing with infinite state-space systems, but require a higher qualification of users and can easily become unmanageable for beginners.

Simulation of hybrid systems

Simulation is a prevailing technique for the analysis of hybrid systems in industry and, often, academia. There exists a plethora of definitions of simulation and mathematical simulation. We will quote a version from Cellier [1991, p.6]: “A *simulation* is an experiment performed on a model”. We discuss a certain type of simulation, a *mathematical simulation*, which is a coded description of an experiment with a reference to the model to which this experiment to be applied [Cellier, 1991, p.6].

Simulation is used to analyse the response of the system to a particular inputs according to *scenario*. It helps to detect the potential weaknesses and errors, and provides information on performance of system. There is a number of simulation tools which provide various facilities for analysis of hybrid systems. Simulation tools and techniques are discussed in depth in Chapter 7.

1.2.3 Deployment of hybrid systems’ models

Correct hybrid systems models are used for different purposes, as *code generation*, *controller generation* (in some sources *controller synthesis* or just *control*) and other production related activities, as scheduling, production planning, documentation, etc.

Code generation

Often hybrid systems consist of a digital controller and an analogue process. Therefore, an attempt to derive a software for digital controller from the system’s model seems quite natural. Such an approach is called *code generation*. For more information about code generation for timed systems see Amnell et al. [2002].

Controller generation

It is a common practice to separate a system into a *plant* and a *controller*. A *plant* denotes a system to be controlled and a *controller* is a component of a system that makes it operate within the desired limits, i.e., it receives outputs of the plant and sends signals to the plant, which should steer it towards the desired behaviour.

Controller generation (in some sources *controller synthesis* or *control*) is a process, such that from the specification of a plant and a set of desired properties of the system, a controller is derived automatically (semi-automatically). A number of different objectives are pursued in controller generation, e.g.:

1. INTRODUCTION

- In *least restrictive control for safety and liveness* it is required that all trajectories of the system satisfy safety and liveness properties and at the same time allow as many inputs as possible at each state [Lygeros et al., 1998].
- In *optimal control*, desired behaviour should be reached while minimising a given cost function.
- In *hierarchical control*, the control task is decomposed into a self-contained hierarchically organised functional layers in such a way that the resulting controller is able to achieve desired goals.
- In *distributed control*, the control tasks are distributed over different components in order to efficiently solve complex problems separating them to sub-problems that are solved by (possibly specialised) components.

For more information about controllers generation see Polderman and Willems [1998], Tomlin et al. [2000], Julius [2005].

Scheduling Some control problems can be reduced to *scheduling* problems, where scheduling is defined as the process of assigning tasks to a set of resources. Usually, scheduling problems are less difficult than controller generation, and feasible solutions are easier to design. Abstraction of controller generation to scheduling is common in modern production and chemical industries. Successful application of formal methods for scheduling of such systems was demonstrated by the AMETIST¹ project.

1.2.4 Testing of hybrid systems models

Testing is one of the techniques to assess the quality and correctness of systems. The *system under test* (SUT) is fed input data and output data is monitored to check its correctness. Models of hybrid systems contain descriptions of the system behaviour, and therefore can be used for tests generation. This is relatively new research for hybrid systems [Zhao et al., 2003]. Berkenkötter and Kirner [2005] gives an overview of available testing techniques for real-time and hybrid systems. More results on testing of timed systems is available in Larsen et al. [2003], Briones and Brinksma [2004, 2005]. A nice introduction to general model-based testing techniques is Broy et al. [2005].

1.3 Main results

In this dissertation we contribute to several areas of hybrid systems research.

Modelling of hybrid systems. In Chapter 3 we survey principal characteristics of hybrid systems and classifications based on these characteristics. We derive a classification scheme from the survey and apply it to a list of major frameworks for modelling and analysis of hybrid systems.

¹AMETIST, IST-2001-35304, <http://ametist.cs.utwente.nl>.

In Chapter 5 we introduce Behavioural Hybrid Process Calculus, a framework for modelling and analysis of hybrid systems that combines process algebraic theory and the behavioural approach to dynamical systems.

Analysis of hybrid systems. In Chapter 4 we propose a technique for stability estimation of hybrid automata.

In Chapter 7 we propose a technique for simulation of Behavioural Hybrid Process Calculus.

1.4 Outline of the dissertation

In this thesis we pursue several goals. We demarcate the hybrid systems realm by illustrating them by examples containing hybrid phenomena and formalisms designed to handle it. Hybrid systems combine two worlds. It does not come as a surprise then that formalisms designed to handle hybrid phenomena usually are derived from continuous or discrete formalisms by augmenting them to certain extent with complementary elements from these adverse worlds. We will follow this trend even further by insisting that research of hybrid systems should be carried out in cooperation by the representatives of these worlds. We illustrate it by proposing elegant and deceptively simple technique for stability estimation of hybrid automata. This experience, together with the analysis of other formalisms shows that an approach with good theoretical foundations, uniform treatment of continuous and discrete behaviours and well defined compositionality is favourable. We therefore propose the Behavioural Hybrid Process Calculus (BHPC) that conforms to these requirements. Furthermore, we survey hybrid systems' simulation techniques and propose a set of procedures for BHPC simulation.

In the remaining part of this section we summarise our motivation and contribution for each chapter of the thesis.

Hybrid phenomena emerges in diverse ways and in various systems. There is no commonly accepted formal definition of hybrid systems. We withhold from giving one too. Instead, we informally introduce hybrid systems as a fusion of discrete and continuous worlds that retain continuous and discrete characteristics while attaining new properties arising from interaction of these two worlds. We illustrate diversity of hybrid systems in Chapter 2 by revealing a panorama of systems of different size and complexity that manifest hybrid phenomena. Only a small corner of the hybrid world is touched, but even such contact reveals diversity and dynamism of the hybrid systems' universe.

In Chapter 3 we examine some of the major existing frameworks and formalisms for the specification and analysis of hybrid systems. We explore essential characteristics of models and frameworks for hybrid systems, such as *compositionality*, *incorporation of continuous and discrete constituents*, etc., and reveal shortcomings and advantages of these approaches.

As we already mentioned, hybrid systems can be seen as some sort of amalgamation that brings together computer science and control theory. Therefore we anticipate that cooperation amongst these two areas should bring fruitful results. We assess these expectation in Chapter 4 by applying techniques from computer science and control theory in stability analysis for hybrid automata. We present an elegant and relatively

simple technique that not only provides an algorithm for stability estimation for a certain class of automata, but also illustrates reuse of well known efficient techniques from both areas. Furthermore, we believe that the results from Chapter 4 inspire some analogue research, where properties of automata can be exploited.

Recent interest in embedded systems materialised in a multitude of models for the specifications and analysis of hybrid systems. However, besides multiple advantages, some of these frameworks suffer from drawbacks, or even lack essential features.

Hybrid systems combine continuous and discrete evolution, but to get adequate representation of both worlds, the fusion should be evenly balanced w.r.t. constituent elements, and should capture interaction between them. However, many formalisms concentrate on the continuous part, while the discrete behaviour is relatively neglected. That allows to utilise control theory techniques relatively easy, however, discrete behaviour remains somewhere in the outskirts. The inverse situation surfaces in the Φ -calculus, where only the environment is continuous.

Yet another immensely important quality of formalisms is compositionality. Any encounter with the design and development of large, complex systems can be used as an argument that it is at least impractical, if not futile, to engage in it without a *divide and conquer* strategy. However, formally defined compositionality is missing in many systems. The situation is better in the *hybrid automata* case, although, parallel composition is well-defined only for discrete actions. Hybrid process algebras provide good facilities for compositional modelling of systems with many other modelling and analysis mechanisms present in process algebras. Unfortunately, *strong bisimulation* is not a congruence in some of these process calculi, and it means that bisimilar components (impossible to distinguish by just observing them) can not be substituted. *Hybrid I/O automata* is a very nice and well developed model for hybrid systems, however, it lacks mechanisms that are usually supplied with process algebras and is built-upon the so-called *directed* or *input/output* communication paradigm that we find too restrictive and not in accordance with our intuition. The same (directed communication paradigm) applies for *Hybrid χ* . *Hybrid behavioural automata* nicely introduces the behavioural approach [Polderman and Willemms, 1998] into hybrid automata formalism, but it lacks tools provided by process algebras, and treats continuous and discrete behaviours completely differently. MASACCIO and CHARON provide a nice hierarchical view of systems, but they have rather inefficacious formalisations. Bond graphs and Modelica™ are strongly oriented towards simulation and are not suitable for theoretical exercises.

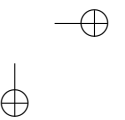
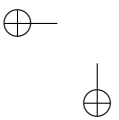
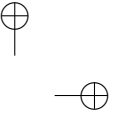
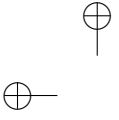
We propose the Behavioural Hybrid Process Calculus (BHPC) in Chapter 5. In this formalism we take into account lessons learned from existing formalisms. We aim at achieving balanced incorporation and uniform treatment of discrete and continuous constituents, well defined and adequate treatment of their interaction, formally defined compositionality and other issues. The resulting calculus combines classical process algebraic techniques [Milner, 1989, Hoare, 1985, Bergstra and Klop, 1984, Bolognesi and Brinksma, 1987] and behavioural approach [Polderman and Willemms, 1998]. It is a formalism with a convenient *separation of concerns* (discrete and continuous behaviours can be separated syntactically), *adequate compositionality facilities*, i.e., *generalised choice* and *parallel composition*. Moreover, bisimulation is a congruence in the calculus. In Chapter 6 we compare BHPC with other approaches.

Although the specification process itself provides already a lot of insight on static characteristics of systems, at some moment further analysis techniques are employed

1.4. OUTLINE OF THE DISSERTATION

to extract information about dynamic behaviour. *Mathematical simulation* is one of such techniques. It is widely accepted in industry and academia, as one of the main tools to analyse dynamic behaviour of diverse systems. There exists a number of simulation techniques and tools that provide different levels of services and facilities. In Chapter 7 we address different hybrid systems simulation techniques and major problems arising in such a process, and some solutions to them. Moreover, we propose a technique for simulation of Behavioural Hybrid Process Calculus (Chapter 5).

In this work we only touch a small part of hybrid systems universe. However, even this encounter allows to envision prospective research tendencies. We address potential research directions and conclude our work in Chapter 8.



Oh, a sleeping drunkard
Up in Central Park,
And a lion-hunter
In the jungle dark,
And Chinese dentist,
And a British queen—
All fit together
In the same machine.
Nice, nice, very nice;
Nice, nice, very nice;
Nice, nice, very nice—
So many different people
In the same device.

Bokonon (Kurt Vonnegut, Jr.)

2

Bestiarium of hybrid systems

2.1 Introduction

In this chapter we present and comment on a set of examples of hybrid systems with diverse properties. The aim is to illustrate the variety of hybrid phenomena and the occurrence of these in different applications. We will return to some of the examples in the subsequent chapters.

The analysis of examples gives a better intuition on hybrid phenomena and the complications it carries. Moreover, the presented examples can be used not only as an illustration of the variety of hybrid phenomena, but as a reference to a specific type of hybrid systems and the literature where it is analysed. The list is not exhaustive as only the illustrative examples were selected.

To introduce some examples we will use the *hybrid automaton* formalism. A hybrid automaton [Alur et al., 1993, Henzinger, 1996] is one of the most popular approaches to model and analyse hybrid systems. In this chapter we explain hybrid automata on the bouncing ball (Section 2.2.1), thermostat (Section 2.2.2) and railroad gate control (Section 2.2.5) examples. A formal definition of hybrid automaton is provided in Section 3.3.6.

2.2 Examples of hybrid systems

Hybrid systems occur in different sizes and complexity. They range from small and simple systems with few discrete states and simple continuous-time behaviour to large and complex systems with complex non-linear behaviour and a big number of discrete states. In our list we set off to illustrate a wide range of hybrid phenomena by presenting examples of different complexity and types. Considering the diversity of hybrid phenomena, it is not possible to illustrate all manifestations or list all interesting

examples.

We start our list with a simple, but non-trivial example of the bouncing ball (Section 2.2.1). Even such a small example warns against perils awaiting in the area of hybrid systems by exhibiting the Zeno behaviour [Johansson et al., 1999]. The thermostat example (Section 2.2.2), even by being rather simple and small, illustrates somewhat realistic heating control system. The fluid level control (Section 2.2.4) and the leaking gas burner (Section 2.2.3) exemplify characteristic behaviour of wide classes of hybrid systems in a relatively simple way. Important concepts of *modularity* and *concurrency* are illustrated in the railroad gate control example (Section 2.2.5). The batch control example (Section 2.2.6) presents an extensive group of the industry-relevant problems, and recommends potential techniques to deal with them. In the mobile vehicles example (Section 2.2.7) we subsume a wide range of large and complex hybrid systems, and discuss potential approaches to handle them.

2.2.1 A bouncing ball

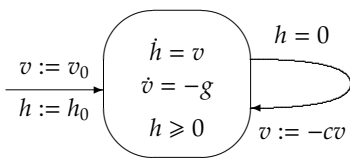


Figure 2.1: A bouncing ball

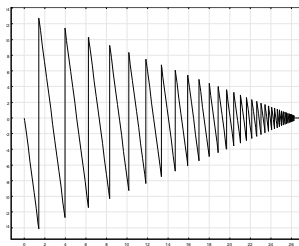


Figure 2.2: Velocity v

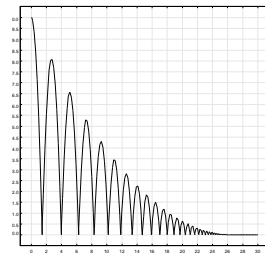


Figure 2.3: Altitude h

A bouncing ball is a simple example of hybrid systems. It is a simplified model of an elastic ball that is bouncing and losing a fraction of its energy with every bounce. The altitude of the ball is h , v is a vertical speed, and c is a coefficient for the lost energy. The ball moves according to the flow conditions

$$\dot{h} = v \quad \dot{v} = -g$$

and at the bounce time the velocity is reassigned to $-cv$.

The hybrid automaton for the bouncing ball is depicted in Figure 2.1 and an example of evolution (velocity and altitude) is presented in Figures 2.2 and 2.3, respectively.

The automaton in Figure 2.1 has only one *location* (or *discrete state*) with *continuous dynamics* (or *flow conditions*) $\dot{h} = v, \dot{v} := -g$ and *invariant* $h \geq 0$, where invariant is a predicate which should hold while the systems stays in the location. *Initial state* $v := v_0, h := h_0$ is provided by the arrow without a source location. An arrow decorated by a *guard* $h = 0$ (in some cases, additional condition $y \leq 0$ is added to prevent repetitive switching while skipping continuous evolution) and an *assignment* $v := -cv$ denotes a *switch* (or discrete change of evolution) which can occur if the guard is satisfied, and which alters the continuous state (defined by the assignment). In this case the evolution starts from the initial state and evolves according to the flow

conditions. When the height becomes 0, the transition is taken, and evolution starts with reassigned velocity, and so on, so forth.

In both figures we see an example of Zeno behaviour [Johansson et al., 1999], when switching between the modes becomes more and more frequent. In other words, we get an infinite number of switches in a finite amount of time. Fortunately, the behaviours converge, therefore a solution can be proposed (in this case 0 altitude and 0 velocity).

References The bouncing ball example is presented in Lygeros and Sastry [1999], De Schutter and Heemels [2004], van der Schaft and Schumacher [2000, p.37–38] as an example of hybrid systems. Several different versions of the bouncing ball are presented and analysed in Johansson et al. [1999]

2.2.2 A thermostat

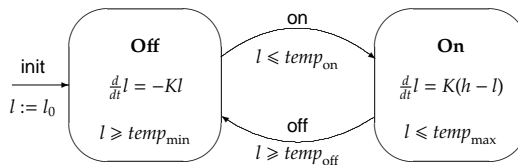


Figure 2.4: A thermostat

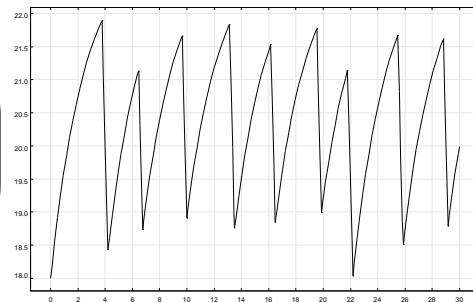


Figure 2.5: Change of the temperature

A thermostat is one of the best-known introductory examples of hybrid systems. The room temperature is controlled by a thermostat, which continuously senses the temperature and switches a heater on and off. The temperature changes are defined by the ordinary differential equations. When the heater is off, the temperature decreases according to the exponential function $l(t) = \theta e^{Kt}$, where t is time, l is the temperature in the room, θ is the initial temperature, and K is a constant determined by the room. When the heater is on, the temperature increases according to the function $l(t) = \theta e^{-Kt} + h(1 - e^{-Kt})$, where h is a constant that depends on the power of the heater. The temperature should be maintained between $temp_{\min}$ and $temp_{\max}$. Temperatures $temp_{\min}$ and $temp_{\max}$ are the minimal and maximal thresholds, when the heater can be turned on and off, respectively.

A hybrid automaton based model of the thermostat is shown in Figure 2.4. The system starts with the temperature $l_0 \in [temp_{\min}, temp_{\max}]$ in location **Off**. The hybrid automaton consists of two location, **Off** and **On**, and the switches between them (on and off). The temperature falls according equation $\frac{d}{dt}l = -Kl$, until it reaches the interval $[temp_{\min}, temp_{\max}]$. Then the guard on the switch decorated with action on becomes enabled. The system can then switch to location **On** at any point of the interval, and it must do so until the temperature l falls below $temp_{\min}$. In location **On**

the temperature increases according to equation $\frac{d}{dt}l = K(h - l)$, until it reaches interval $[temp_{off}; temp_{max}]$. When the temperature becomes higher or equal to $temp_{off}$, the guard on off is enabled and again, the transition is taken before temperature exceeds $temp_{max}$, and so forth.

An example of evolution is depicted in Figure 2.5. The thermostat starts with temperature 18 in location **Off** and immediately switches to location **On**, and then evolves as defined above.

It is quite common for hybrid systems to show the cyclic behaviour as in the thermostat example and almost cyclic behaviour as in the bouncing ball example. Supervision of repetitive tasks is the most common application area of the hybrid systems. This knowledge is not so important in modelling small examples of hybrid systems, but it becomes very useful when solving scheduling problems for larger systems, like the *batch plant control* discussed in Section 2.2.6.

References The thermostat example is widely used in the literature. A simple, classical thermostat, which is modelled by the two-states hybrid automaton is described in Alur et al. [1995], Henzinger [1996], Lygeros and Sastry [1999]. In Henzinger et al. [1997] several different versions of the thermostat are presented, and a profound analysis of different properties using HyTECH is given. In Rönkkö and Ravn [1997b], the hybrid actions approach is used to model a simple thermostat. An ACP-style process algebra is used to specify a simple thermostat in Vereijken [1995]. In Jacobs [2000] a thermostat is specified using coalgebras with monoid actions. A version of thermostat is presented in van der Schaft and Schumacher [2000, p.38–40]. A BHPC specification of the thermostat is available in Brinksma and Krilavičius [2005] and Example 5.8.2.

2.2.3 A leaking gas burner

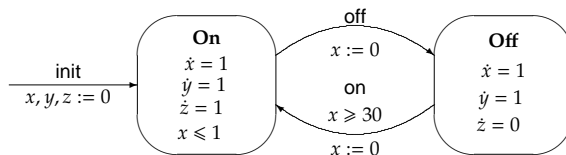


Figure 2.6: A leaking gas burner

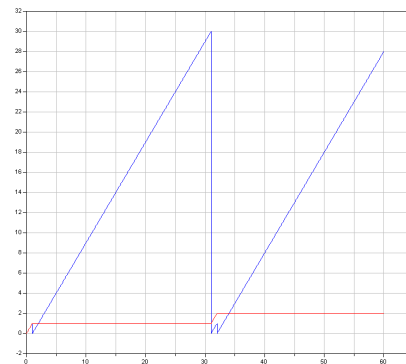


Figure 2.7: Evolution of the gas burner

A leaking gas burner is a simple example of a hybrid system. It describes a valve, which controls a gas supply to a burner. The following properties are checked.

- A continuous leaking period cannot extend beyond a specified number of time units.

- The accumulated time of leakage is at most some specified amount of time in any interval of at least 60 seconds.

The goal is to check whether over a prolonged time period the cumulative leaking time constitutes only a certain percentage of the overall evolution time.

Such system can be modelled by a two-states hybrid automaton, as depicted in Figure 2.6 from Alur et al. [1995]. In this model the leaking time is 1 time unit and the gas burner will not leak for 30 time units after a leakage has been stopped. The goal is to show that the accumulated time of leakage is at most one twentieth of the time in any interval of at least 60 time units. The clock x records the time spent in the current location, the integrator z records the cumulative leakage time and the clock y records total elapsed time. An example of such evolution is depicted in Figure 2.7, where the cumulative leaking time is represented by the slowly rising line, the fast rising line represents time flow in the states and the resets indicate switching moments.

At the first glance this system does not seem so much different from the thermostat (Section 2.2.2). But the similarity is deceptive. The system is enlarged with an important summand, which can be considered as an implicit memory. In this example, an evolution of the system depends not only on the locations, but on its previous behaviour. In contrast, in the thermostat (Section 2.2.2) example all necessary information is encoded in locations.

References The leaking gas burner is a popular example and can be encountered in many papers. In Alur et al. [1995], Henzinger and Rusu [1998] a leaking gas burner with leaking and non-leaking periods is analysed. Lamport [1993] uses a temporal logic of actions to model a gas burner.

2.2.4 A fluid level controller

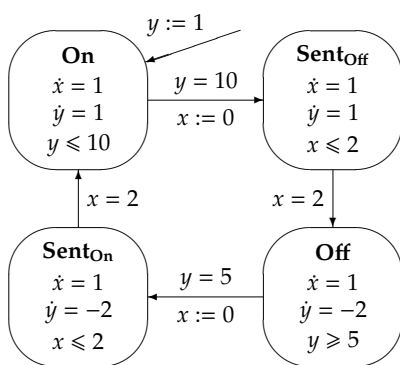


Figure 2.8: A fluid level controller

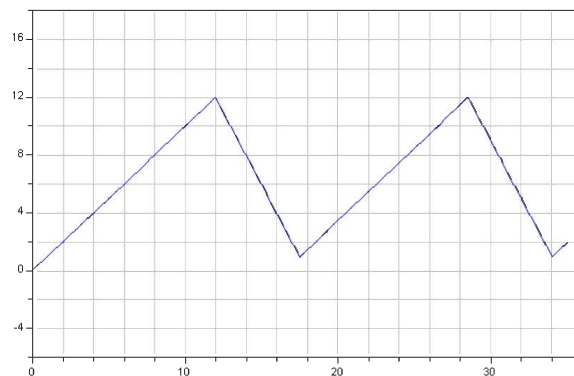


Figure 2.9: Change of the fluid level

There are several examples of hybrid systems describing different varieties of fluid level controller. The examples range from a simple fluid level control in a tank to a fluid level control in a complex network of tanks with several valves and pumps. The main goal in the examples is to maintain the required fluid level in several vessels

by changing the fluid input and draining speed, opening and closing (starting and stopping) the valves and the pumps (with potentially delayed reaction of the pumps and/or valves). One of the most popular versions of the fluid level control examples is presented in [Alur et al., 1995] and is described in the following way.

The fluid level in a tank is controlled through a monitor, which continuously senses the fluid level and turns a pump on and off. The fluid level changes as a piecewise linear function over time. When the pump is off, the fluid level, denoted by a variable y , falls by 2 units per second; when the pump is on, the fluid level rises by 1 units per second. It is required to keep the fluid level between 1 and 12 units. The pump receives a signal from a monitor delayed by 2 time units. Thus, the signals to turn the pump on and off should be sent before the threshold is reached. The corresponding hybrid automaton model is presented in Figure 2.8, and an example of fluid level fluctuation is depicted in Figure 2.9. The hybrid automaton has four locations

- **On** - the pump is on,
- **Sent_{Off}** - the pump is on, but a signal to stop the pump is sent,
- **Off** - the pump is off,
- **Sent_{On}** - the pump is off, but a signal to start the pump is sent.

The example can be thought-of as an extension of the thermostat (Section 2.2.2) with delays. In fact, adding a location to model a delay is standard practice in hybrid (and timed) systems' modelling.

References The fluid level control examples are found in many papers in several flavours: the tanks are connected in different ways, a control of volume of fluid can be accomplished differently (opening and closing valves of outlets, switching on and off the pumps), etc.

The simplest version of the fluid level control is a one-tank system, the one we described in the previous section. It is described in Alur et al. [1995], De Schutter and Heemels [2004]. An analogue version is presented in van der Schaft and Schumacher [2000, p.38–41].

A similar example is given in Henzinger et al. [1993], but fluid flows in at a constant speed, and an output valve can be closed and opened, when it is required. The valve reacts without delays.

In Rönkkö and Ravn [1997a] hybrid actions are used to model a one tank system, where leaking starts only when the maximum fluid level is reached, but restrictions for filling are introduced. A similar system, where a pump and an outlet valve can be turned on and off is described in Heymann et al. [1997].

One tank system with several fillers is specified in Rönkkö and Ravn [1997a] using hybrid actions. A more complex example with one tank is given in Cuzzola and Morari [2001], where two outlets are activated, when the fluid level reaches the preset limits and one outlet provides a constant output all the time. Input flow varies between 0 and u_{max} .

Another class of the fluid level control examples is oriented to solving scheduling problems. For several tanks only one filler is available, and the objective is to optimise its use. In Heymann et al. [1997], Lygeros and Sastry [1999], Simić et al. [2000], De

Schutter and Heemels [2004] a particular fluid level in two tanks should be maintained. In Labinaz et al. [1996] there are three tanks, input and output flows are constant.

To make it even more complicated, tanks can be interconnected. In Kowalewski et al. [1999] two interconnected tanks, which are placed at different height, are analysed. A pump, a valve between tanks and an output valve can be switched on and off. The dynamics of such system are far from trivial. The required fluid level should be maintained in the three interconnected tanks (from the first tank fluid flows to the second, from the second to the third, and from the third it flows out) in Raisch et al. [1999]. The first and the third tanks can be filled by independently turning the first and the third pumps on.

2.2.5 Railroad gate control

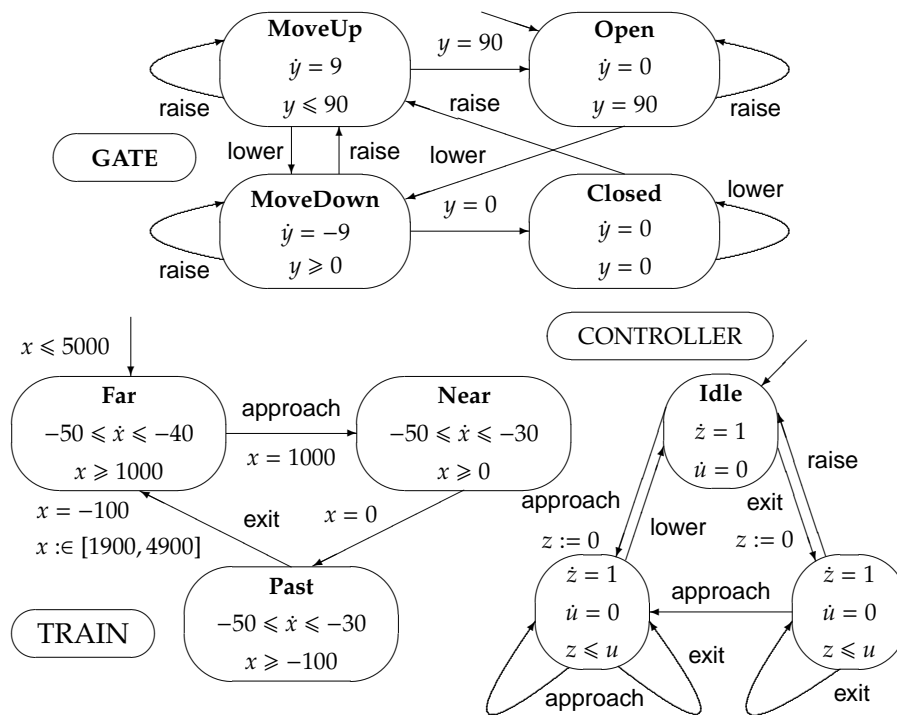


Figure 2.10: Train, gate and controller automata

A railroad gate control models a train on a circular track with a controlled gate. A controller issues open and close depending on information about the train movement. A version from Henzinger [1996] is presented below.

The initial speed of the train is between 40 and 50 metres per second. At the distance, which is represented by a variable x , of 1000 metres from the gate, the train issues an approach event and may slow down to 30 metres per second. At the distance of 100 metres past the gate it issues an exit event. The circular track is between 2 and 5 kilometres long. When an approach event is received, the controller issues a

lower event with a u seconds delay, and when an `exit` event is received, the controller issues a `raise` event within u seconds. The elapsed time is represented by a variable z . Initially the gate is open. A position of the gate, which is represented by a variable y , is measured in degrees, and initially is 90. When a `lower` event is received, the gate starts closing at the rate of 9 degrees per second, and when `raise` event is received, the gate starts opening at the same rate. The purpose of the model is to find u - the reaction delay. Hybrid automata of the train, the gate and the controller are presented in Figure 2.10. The specification is separated into three parts, which model the gate, the train and the controller. These three components communicate by sending and receiving messages, which correspond to actions. Therefore the train announces about it's arrival by issuing `approach` event, and the controller responds to it by changing it's state.

References An application of HyTECH and hybrid automaton for the rail-road gate control is presented in Henzinger [1996], Henzinger et al. [1995, 1997]. Several different approaches are used: with a simplified differential inclusion and using a timed automaton in Puri and Varaiya [1995]. One more version of the railroad gate control is presented in van der Schaft and Schumacher [2000, p.51–52].

2.2.6 Batch plant control

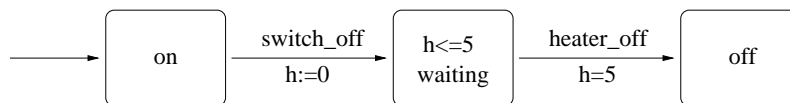


Figure 2.11: Heater

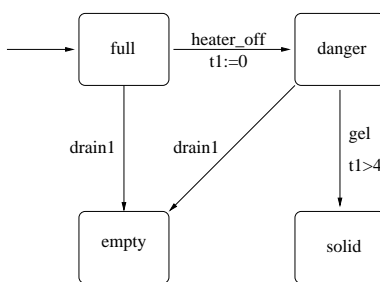


Figure 2.12: Container T1

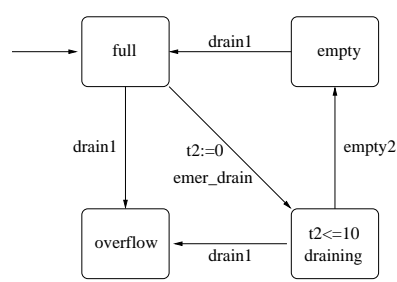


Figure 2.13: Container T2

A large class of examples for hybrid systems describe *Batch plant control* of several varieties. The general scenario is the following: there are several chemical substances, and they should be mixed in specified proportions, following a specified order. The temperature should be controlled all the time or only after mixing substances. In some versions a mixing device is used, and it can be turned on and off. If the resulting chemical substance is ready, it should be removed from the production vessel. When a system of several vessels is used, vessels should be ready to receive the product when it is ready.

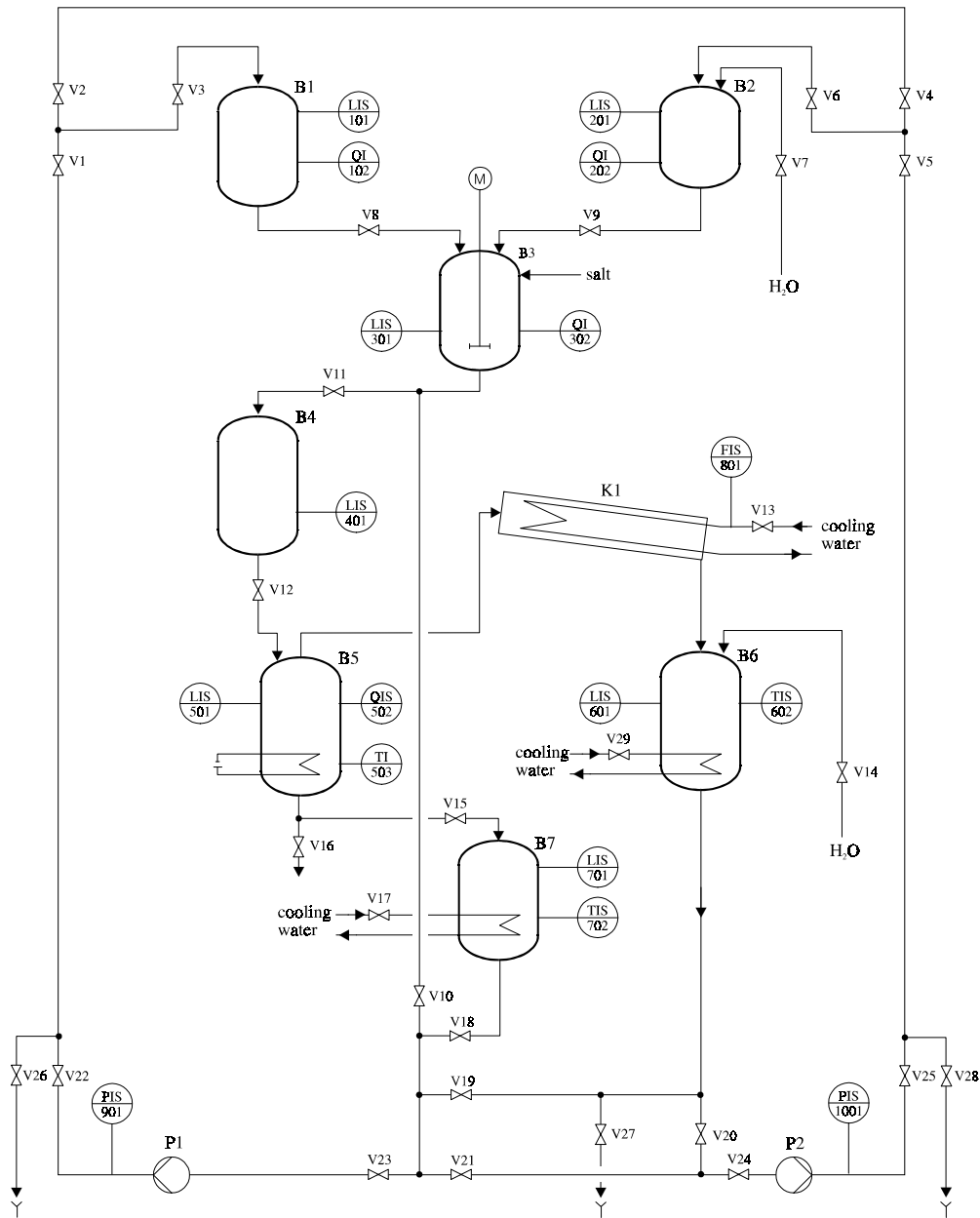


Figure 2.14: Piping and instrumentation diagram of the plant

Sensors for temperature, fluid level, concentration and other measurements are used. Delays for the actuators and the sensors can be introduced in more complicated versions.

Such systems are quite complicated, because it is necessary to pursue several

different objectives. In the literature, examples vary quite a lot, and sometimes case studies of production environments are presented.

We illustrate such a type of system by an experimental batch control plant presented in Kowalewski [1998] which was used as a case study in the VHS project¹. Different approaches to control the plant are presented in Kowalewski et al. [2001], Bemporad et al. [2001], Huuck et al. [2001], Mader et al. [2001], Niebert and Yovine [2001].

The plant is depicted by so-called P/I (piping and instrumentation diagram) in Figure 2.14. It is originally designed as an educational apparatus for student exercises. It “produces” batches of diluted salt solution from a concentrated salt solution (container B1) and water (container B2). They are mixed in container B3 to obtain the diluted solution, which is transported to container B4 and then to container B5. In B5 an evaporation process is started. The evaporated water is condensed in K1 and then goes to container B6, where it is cooled and pumped to B2. The remaining hot, concentrated salt solution from B5 is transported back to B7, cooled down and pumped back to B1.

The reader is encouraged to consult Kowalewski et al. [2001], Bemporad et al. [2001], Huuck et al. [2001], Mader et al. [2001], Niebert and Yovine [2001] for different modelling approaches. Here we illustrate one of the possible ways to model the containers and heaters. In Figures 2.11, 2.12 and 2.13 automata of the heater and two containers from Huuck et al. [2001] are presented, respectively. These automata are used to model the condenser failure. Only an evaporator component is modelled to analyse the system’s reaction to failure, and these three automata form only a part of it. The heater, after receiving `switch_off` resets clock h (a *clock* is a special type of continuous variable, which can be defined like $\dot{h} = 1$) and waits in the location **waiting** for 5 time units, then sends `heater_off` and switches to the location **off**. Container T1 (which corresponds to B5 in Figure 2.14) gets emptied immediately after receiving event `drain1`. Event `gel` indicates that the solution in the container became solid. If container T2 (B7 in Figure 2.14) is not empty, when T1 is drained, it overflows. Therefore action `emer_drain` should be enforced timely.

Even by examining a small part of bigger system it is easy to see that modular approach is indispensable. Moreover, the components can be reused in other models of the same or similar systems.

References A version of the chemical reaction is modelled and analysed in Anderson et al. [1993]. In this example a special attention is drawn to the analysis of the safety requirements.

In Jacobs [2000] chemical reactions are specified and analysed using coalgebras with monoid actions, which capture the influence of time on the state space. A simple reactor for waste water treatment is modelled and analysed in Williams and Newell [1997]. In Philippe et al. [2000] the Pontryagin’s Maximum Principle is used to minimise the overall operating time of the system, which is modelled using the hybrid automaton. An evaporator vessel, as a part of a chemical reaction is modelled in Mosterman [1999]. A complex batch evaporator is analysed in Kowalewski and Stursberg [1998].

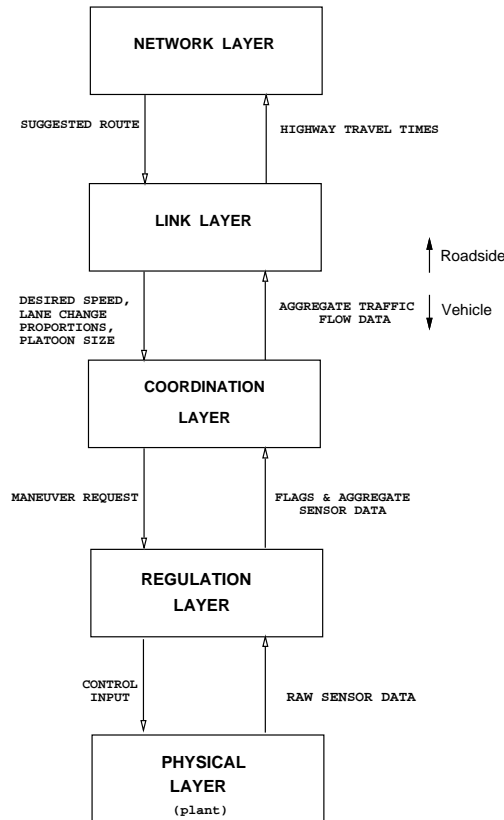


Figure 2.15: AHS control hierarchy

2.2.7 Mobile vehicles

Examples of mobile vehicles control range from simple models of automatic or semi-automatic movement to complex multi-vehicle movement coordination systems. Vehicles in examples range from personal cars to air-planes and submarines. Control objectives vary and in some cases are conflicting. Often such systems are modelled in a hierarchical way, i.e., there are several principal control centres, which provide recommendations for the agents. In addition, the agents can interact with each other without the mediation of these centres.

The following types of systems belong to the mobile vehicles class.

- Automated Highway Systems (AHS).
- Air Traffic Management Systems (ATM), Flight Vehicle Management Systems (FVMS), Autonomous Flight Vehicles.
- Sea Traffic Management Systems (STMS), Autonomous Underwater Vehicles.

¹ESPRIT-LTR Project 26270 VHS (Verification of Hybrid systems), <http://www-verimag.imag.fr/VHS/>.

- Mobile Robots' movement coordination systems.

Most of such systems have the following configuration and functionality:

- An autonomous or semi-autonomous vehicle, with certain requirements:
 - Movement of a single vehicle, according to the comfort and physical requirements, safe and comfortable manoeuvring.
 - Movement of a single vehicle, following an optimal route, avoiding obstacles.
 - Movement in a group of vehicles, avoiding collisions, joining and leaving a group.
 - Communication with other vehicles (negotiations, etc.) and control centre(s), responding to it's commands and warnings.
- A movement coordination centre tasks are:
 - Traffic control.
 - Routeing vehicles.
 - Vehicles "handing-on/off" amongst the control centres.
 - Issuing diverse warnings.

Simple models of mobile vehicles are presented in Alur et al. [2001], Lygeros et al. [1997]. Alur et al. [1999], Koo et al. [2001], Lynch [1996], Lygeros and Sastry [1999] add more complexity and elaboration to the examples. The examples from Lygeros et al. [1999], Tomlin et al. [1998], Branicky et al. [2000] subsume simpler problems and examples.

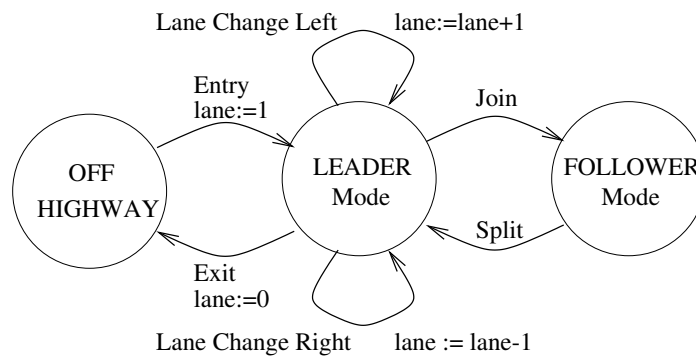


Figure 2.16: Discrete states of an AHS vehicle

We do not provide models of mobile vehicle systems here, because they are usually very complex and big. However, we illustrate the complexity of such systems by providing Figures 2.15 and 2.16 from Lygeros and Sastry [1999], which illustrate Automated Highway System hierarchy and a discrete abstraction of an AHS vehicle, respectively.

2.3 Conclusions

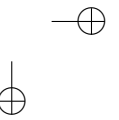
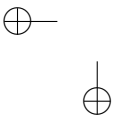
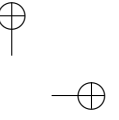
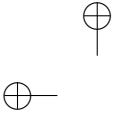
A list of hybrid systems' examples was presented in this chapter. The list is by no means complete, as only illustrative examples were included.

The presented examples demonstrate the diversity of the hybrid systems world. The systems range from small and deceptively simple to large and complex systems. However, even small examples (e.g., the bouncing ball example, Section 2.2.1) contain surprises, like Zeno phenomena.

Moreover, we used the hybrid automaton formalism (Section 3.3.6) to present some of the examples.

The presented examples show that even small systems corresponding to somewhat realistic systems can be successfully modelled and discussed. For larger systems a "divide and conquer" approach helps a lot.

A nice selection of various examples of hybrid systems is available in Lygeros and Sastry [1999], van der Schaft and Schumacher [2000] and De Schutter and Heemels [2004]. Good sources for examples are the proceedings of *International Hybrid Systems Workshops* [Grossman et al., 1993, Antsaklis et al., 1995, Alur et al., 1996b, Antsaklis et al., 1997, 1999] and the proceedings of *International Workshops on Hybrid Systems: Computation and Control* [Henzinger and Sastry, 1998, Vaandrager and van Schuppen, 1999, Lynch and Krogh, 2000, Benedetto and Sangiovanni-Vincentelli, 2001, Tomlin and Greenstreet, 2002, Maler and Pnueli, 2003, Alur and Pappas, 2004].



There was a red-haired man who had no eyes or ears. Neither did he have any hair, so he was called red-haired theoretically. He couldn't speak, since he didn't have a mouth. Neither did he have a nose. He didn't even have any arms or legs. He had no stomach and he had no back and he had no spine and he had no innards whatsoever. He had nothing at all! Therefore there's no knowing whom we are even talking about. In fact it's better that we don't say any more about him.

Daniil Kharms

3

Overview of models for hybrid systems

3.1 Introduction

The high interest in hybrid systems and control theory communities materialised into diverse hybrid formalisms. A variety of hybrid phenomena and differences between communities have led to multiple formalisms, which are suitable for different tasks and tastes. Considering that we are contributing to this multitude, we survey some of the major approaches. The list is not exhaustive, therefore we encourage an interested reader to consult the proceedings of *International Hybrid Systems Workshops* [Grossman et al., 1993, Antsaklis et al., 1995, Alur et al., 1996b, Antsaklis et al., 1997, 1999] and the proceedings of *International Workshops on Hybrid Systems: Computation and Control* [Henzinger and Sastry, 1998, Vaandrager and van Schuppen, 1999, Lynch and Krogh, 2000, Benedetto and Sangiovanni-Vincentelli, 2001, Tomlin and Greenstreet, 2002, Maler and Pnueli, 2003, Alur and Pappas, 2004] for more information. Furthermore, our intention is not to provide detailed analysis and description of available hybrid systems' formalisms, but rather to illustrate the main characteristics and features of different approaches.

We do not consider probabilistic and stochastic aspects of hybrid systems in this dissertation, and consequently, we do not present or analyse hybrid probabilistic and hybrid stochastic formalisms.

3.2 Classification of hybrid systems

Classifying diverse phenomena is one of the principal occupations of scientists. Therefore it does not come as a surprise that several attempts were made to classify hybrid systems. Different authors have chosen diverse aspects of hybrid systems as a base for their classification. Diverse properties were emphasised based on the potential use

3. OVERVIEW OF MODELS FOR HYBRID SYSTEMS

Category	Explanation
Autonomous switching	Flow conditions change on hitting specified region border
Autonomous jump (reset)	Continuous state changes on hitting specified region border
Controlled switching	Flow conditions change in response to a control command
Controlled jump (reset)	Continuous state changes in response to a control command

Table 3.1: Classification of hybrid systems by Branicky et al. [1994]

Category	Subcategory	Explanation
Sampling	Continuous	Value of the continuous state is assumed to be known and available all the time
	Regular	Value of the continuous state is evaluated at some predetermined, fixed sampling period
Dynamics	Linear	Dynamics are defined by the linear diff. equations
	Nonlinear	Dynamics are defined by the nonlinear diff. equations
Determinism (Continuous dynamics)	Deterministic	The evolution of systems is a uniquely determined by the current state and inputs
	Nondeterministic	The evolution of systems is not determined uniquely by the current state and inputs
Determinism (Discrete dynamics)	Deterministic	Each state is mapped to a single next state
	Nondeterministic	State is mapped to a set of states
Control action	Continuous	Continuous control functions with continuous domain and range
	Discrete	Function with a discrete range and a continuous or discrete domain
	Combined	Combination of discrete and continuous control
Specifications	Continuous	Specifications are based on the continuous time behaviour and/or variables
	Discrete	Specifications are based on the discrete event behaviour and/or variables
	Combined	Specifications are based on combination of the continuous and discrete variables

Table 3.2: Classification of hybrid systems by Labinaz et al. [1996]

of formalisms. Here we summarise classifications proposed in Branicky et al. [1994], Labinaz et al. [1996], Mosterman [1999].

Category	Type	Explanation
Events	Time	Events are generated at predetermined times
	State	Events occur because system crosses some thresholds, the time of their occurrence is not known a priori - they need to be detected
Simulation model	Dynamical blocks	Blocks of sorted and solved equation may simply appear or disappear (vehicle entering or leaving highway), and, can be dynamically added/removed
	Changeable continuous dynamics	In some cases equations can be replaced by others, changing computational causality, and the system of equations may have to be sorted again
	Constraint changes	In other cases, algebraic constraints amongst state variables may become active and the system of equations needs to be solved again (the rod making contact to the floor)
Re-initialisation	Explicit	Change explicitly specified by user by a new initial state
	Integrated	The system of equations may have to be integrated to derive physically consistent initial values for a new mode. This ensures conservation of the thermodynamic extensity holds
Event iteration	Invariant	State vector is invariant across the entire iteration
	Updated	State vector is updated after every iteration step
Chattering		Fast switching amongst several modes
Dirac pulses		Non-continuous changes for continuous variables; if their magnitudes are numerically approximated, comparison maybe affected by non-Dirac type variables

Table 3.3: Classification of hybrid simulation phenomena by Mosterman [1999]

Branicky et al. [1994] considers causality of flow conditions and switches. The proposed classification is presented in Table 3.1. Two types of stimuli are singled out: external and internal. The changes induced by the internal stimuli are called *autonomous*, and the changes generated by the external stimuli are called *controlled*. In the table flow conditions denote the continuous evolution, e.g., it may be a vector field or a set of trajectories. The proposed classification groups hybrid systems to four classes. Hybrid systems with *autonomous switching* change the flow conditions by hitting a border of the specified region, where the *region* is a part of state space, defined by, e.g., the set of inequalities. *Autonomous jump* defines the discrete change of the state, which occurs, when the specified region's border is reached. *Controlled*

3. OVERVIEW OF MODELS FOR HYBRID SYSTEMS

Formalism	Sampling	Cont. dynamics	Determinism		Switching	Resets
			Cont. dyn.	Discr. dyn.		
PWA	both	linear	det ^a	det	auto-nomous	n.a.
MLD	both	linear	det	det	both	n.a.
ECL	both	linear	det	det	both	aut
MMPS	both	linear	det	det	both	n.a.
HA	cont	non-linear	det	nondet	both	both
HBA	cont	non-linear	nondet	nondet	both	both
BHPC	cont	non-linear	nondet	nondet	both	both
HyPA	cont	non-linear	nondet	nondet	both	both
ACP _{hs} ^{srt}	cont	linear	det	nondet	both	both
Hybrid χ	cont	non-linear	det	nondet	both	both
Φ -calc.	cont	non-linear	det	nondet	both	both
HIOA	cont	non-linear	nondet	nondet	both	both
MASACCIO	cont	non-linear	nondet	nondet	both	both
CHARON	cont	non-linear	nondet	nondet	both	both
bond graphs	cont	non-linear	det	det	both	both
Modelica TM	cont	non-linear	det	det	both	both

^aThere exists an implicit non-determinism in PWA, when the border is a part of several regions. In such a case it is either resolved by the implementation or reported as an error.

Table 3.4: Properties of hybrid systems

switching and jump occur in response to the external stimuli, or control commands.

Most of the hybrid systems are likely to exhibit different combinations of the classes proposed in Table 3.1. System may manifest both autonomous switching and autonomous jump behaviour, as in the bouncing ball example (Section 2.2.1) or have a complex behaviour, where some components have controlled switching and jumps, and others jump and/or switch autonomously, as in the railroad gate control example (Section 2.2.5).

Meanwhile Labinaz et al. [1996] considers a finer view of properties of hybrid

systems. The proposed classification, with some notions adapted for consistency reasons, is presented in Table 3.2. We give supplementary comments to the notions used in the table.

Sampling defines the availability of continuous state values.

- *Regular*: measurements are taken based on some predetermined, fixed sampling period.
- *Continuous*: measurements are available (and known) at all times.
- *Both*: regular and continuous sampling modes are supported.

Continuous dynamics can be classified as *linear* or *non-linear*.

Determinism of continuous dynamics. A continuous dynamical system is *deterministic* if the system's evolution is uniquely determined by the current state and inputs. It is *non-deterministic* if there are several ways the system can evolve, i.e., the same input can potentially lead from the current state to one of several states. In the process algebras' case continuous behaviour is called *non-deterministic*, if the time passage can resolve a choice.

Determinism of discrete dynamics. Discrete dynamics are called *non-deterministic* if the same discrete action can potentially take the system from one state to one of several different states.

Control actions can be purely continuous functions or discrete actions, or combination of both.

Specifications are usually combine continuous and discrete behaviour, but in some cases they can be reduced to only continuous or discrete.

This classification is more appropriate for the ranking diverse modelling approaches, but the notions of the complexity of continuous dynamics, the sampling type, the manifestation of (non)determinism, and the control actions' types are applicable for hybrid systems too. For instance, in the thermostat example (Section 2.2.2) the sampling is continuous, the continuous dynamics are linear, the continuous evolution is deterministic, the discrete switching is deterministic w.r.t. the locations, but it is non-deterministic in time, and the control actions are discrete.

Mosterman [1999] proposes a classification (Table 3.3) of hybrid systems simulation.

In Table 3.4 we classify formalisms presented in Sections 3.3.2–3.3.13 and Chapter 5 using a combination of schemas from Branicky et al. [1994] and Labinaz et al. [1996] (Tables 3.1 and 3.2). The columns in the table correspond to the properties of hybrid systems and the rows to the models.

Several comments are required to explain Table 3.4.

- In the general definition of complementarity systems non-linear behaviour is allowed, but later restrictions are made to the linear behaviour in the linear complementarity systems, therefore in Table 3.4 it is qualified as linear.
- In some formalisms as automata, transition systems, process algebras, hierarchical models, Modelica™, bond graphs, only continuous sampling is defined, but regular sampling can be modelled in these formalisms, and most of the time in some standardised way.

- In BHPC and HyPA continuous dynamics are non-deterministic w.r.t. choice and alternative composition, respectively.
- MLD's are well posed (Section 3.3.3), if the continuous dynamics are deterministic, therefore in Table 3.4 the continuous dynamics are qualified as deterministic.
- In Modelica™ discrete control inputs should be adjusted to the modelling language.

3.3 Hybrid formalisms

3.3.1 Grouping hybrid formalisms

Since hybrid systems is a meeting place for computer science and control theory communities representatives, it is not surprising that different formalisms were devised to handle hybrid phenomena. Additional diversity was added by the competing factions in the communities itself. Respecting it we distinguish several origin-based groups.

Dynamical systems Hybrid *dynamical systems* are classical dynamical systems extended to handle hybrid phenomena. We present some of them in Sections 3.3.2–3.3.5. These formalisms suffer from several drawbacks, and have some strong points. Often control theory results and tools are applicable to these formalisms (not always, e.g., it does not work so well with Lyapunov stability theory). However, often these approaches suffer from the following drawbacks:

- Discrete behaviour is often oversimplified or even neglected. It is somehow translated to the continuous world, mostly in *ad-hoc* fashion, thus expensive and error-prone¹.
- Formalisms usually are not modular, techniques as parallel composition (interconnection) are not available, and it complicates work with big and complex systems.

Automata, process algebras and transition systems Automata (Sections 3.3.6, 3.3.7), transition systems (Section 3.3.8), process algebras (Sections 3.3.9) and hierarchical approaches (Sections 3.3.10, 3.3.11) are based on developments in computer science. Their strength lies in the following characteristics.

- Elaborated techniques to specify and analyse discrete behaviour. Many results from automata and process algebras research can be used for the analysis of discrete behaviour of these systems.
- The approaches are modular and support parallel composition.

But there are some serious drawbacks:

¹It becomes error-prone and expensive, because every time new development procedures and methods are (re)invented.

- Often tools and even algorithms for these formalisms are missing.
- Continuous behaviour is oversimplified or even neglected. Often it is considered as something what should be specified and analysed by somebody else (e.g., control theorists, engineers).

Simulation languages A specific group of formalisms are *simulation languages* (Sections 3.3.12 and 3.3.13). As the group name tells, these are mostly used for modelling and simulation of physical systems, and less for theoretical research. They have tool support, and often are used in practice.

Grouping of formalisms

Labinaz et al. [1996] separates hybrid formalisms into the following groups.

- Automata and transitions systems: hybrid input/output automata (Section 3.3.8), hybrid automata (Section 3.3.6), hybrid behavioural automata (Section 3.3.7), Behavioural Hybrid Process Calculus (Chapter 5), hybrid process algebra HyPA (Section 3.3.9), process algebra for hybrid systems (Section 3.3.9), Hybrid χ (Section 5), Φ -calculus (Section 5), MASACCIO (Section 3.3.10), CHARON (Section 3.3.11).
- Algebraic structures [Jacobs, 2000, Grossman and Larson, 1995] are not presented here.
- Dynamical systems: piecewise affine systems (Section 3.3.2), mixed logical dynamical systems (Section 3.3.3), complementarity systems (Section 3.3.4), max-min-plus-scaling systems (Section 3.3.5).
- Programming languages (also called simulation languages): bond graphs (Section 3.3.12), ModelicaTM (Section 3.3.13).

We adopt a refined version of grouping by Labinaz et al. [1996] for the formalisms listed in Sections 3.3.2–3.3.13.

- Dynamical systems:
 - Piecewise affine systems (PWA), Section 3.3.2;
 - Mixed logical dynamical (MLD) systems, Section 3.3.3;
 - Complementarity (LCS, ELCS) systems, Section 3.3.4;
 - Max-min-plus-scaling (MMPS) systems, Section 3.3.5.
- Automata:
 - Hybrid automata (HA), Section 3.3.6;
 - Hybrid behavioural automata (HBA), Section 3.3.7.
- Hybrid transition systems:
 - Hybrid I/O automata (HIOA), Section 3.3.8.
- Hybrid process algebras (Section 3.3.9)

- Behavioural Hybrid Process Calculus (BHPC), Chapter 5;
- Hybrid process algebra HyPA ;
- Process algebra for hybrid systems (ACP_{hs}^{srt}) ;
- Hybrid χ ;
- Φ -calculus.
- Hierarchical approaches:
 - MASACCIO, Section 3.3.10;
 - CHARON, Section 3.3.11.
- Simulation languages:
 - Bond graphs, Section 3.3.12;
 - Modelica™, Section 3.3.13.

3.3.2 Piecewise affine systems

Continuous time piecewise affine (PWA) systems [Sontag, 1981, Heemels et al., 2001, De Schutter and Heemels, 2004], [Cuijpers, 2004, p.89–91] are described by

$$\dot{x} = A_i x + B_i u + f_i \quad (3.1a)$$

$$y = C_i x + D_i u + g_i \quad (3.1b)$$

for $[x \ u]^T \in \Omega_i$ and $i = 1, \dots, N$ where Ω_i 's are convex polyhedra, e.g., given by a finite number of linear inequalities, in the input/state space with non-overlapping interiors, but coinciding boundaries. The polyhedra can be open or closed. Variables $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $y \in \mathbb{R}^k$ denote the state, input and output, respectively. The A_i, B_i, C_i, D_i are matrices of appropriate dimensions and f_i, g_i are constant vectors.

Discrete piecewise affine (PWA) systems [Sontag, 1981, Heemels et al., 2001, De Schutter and Heemels, 2004] are described by

$$x(t+1) = A_i x(t) + B_i u(t) + f_i \quad (3.2a)$$

$$y(t) = C_i x(t) + D_i u(t) + g_i \quad (3.2b)$$

for $[x(t) \ u(t)]^T \in \Omega_i$ and $i = 1, \dots, N$ where Ω_i are convex polyhedra (i.e., given by a finite number of linear inequalities) in the input and state space with non-overlapping interiors. Variables $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^m$, $y(t) \in \mathbb{R}^k$ denote the state, input and output, respectively, at time t .

The system is driven by inputs and switching occurs by getting into the different regions of state space, which are convex polyhedra defined by Ω_i 's. The mode of evolution is determined by the region of the state space.

Example 3.3.1. Example of an integrator with upper saturation is taken from De Schutter and Heemels [2004]. A simple integrator with upper saturation can be defined as:

$$x(t+1) = \begin{cases} x(t) + u(t) & x(t) + u(t) \leq 1 \\ 1 & x(t) + u(t) \geq 1 \end{cases}$$

$$y(t) = x(t)$$

It can be rewritten in (3.1) form

$$\begin{aligned} \Omega_1 &= \{(x(t), u(t)) \in \mathbb{R}^2 \mid x(t) + u(t) \leq 1\} \\ \Omega_2 &= \{(x(t), u(t)) \in \mathbb{R}^2 \mid x(t) + u(t) \geq 1\} \\ A_1 &= 1, \quad A_2 = 0, \quad B_1 = 1, \quad B_2 = 0 \\ f_1 &= 0, \quad f_2 = 1, \quad C_1 = 1, \quad C_2 = 1 \\ D_1 &= 0, \quad D_2 = 0, \quad g_1 = 0, \quad g_2 = 0 \end{aligned}$$

□

A PWA system is called *well-posed*, if Equation (3.1) is uniquely solvable in \dot{x} and y , once $x(0)$ and u are specified.

Remark 3.3.2. In Morari et al. [2003] a different definition of PWA is given. But this definition already includes some elements of mixed logical dynamical systems (Section 3.3.3). □

Tool support See Section 3.3.3.

3.3.3 Mixed logical dynamical systems

Discrete time *mixed logical dynamical* (DT-MLD) systems [Bemporad and Morari, 1999, Heemels et al., 2001, De Schutter and Heemels, 2004] are described through the following relations:

$$x(t+1) = A_t x(t) + B_{1t} u(t) + B_{2t} \delta(t) + B_{3t} z(t) \quad (3.5a)$$

$$y(t) = C_t x(t) + D_{1t} u(t) + D_{2t} \delta(t) + D_{3t} z(t) \quad (3.5b)$$

$$E_{2t} \delta(t) + E_{3t} z(t) \geq E_{1t} u(t) + E_{4t} x(t) + E_{5t} \quad (3.5c)$$

where $t \in \mathbb{Z}$, $A_t, B_{1t,2t,3t}, C_t, D_{1t,2t,3t}, E_{1t,2t,3t}$ are matrices of suitable dimensions

$$x = [x_c \ x_l]^T, \quad x_c \in \mathbb{R}^{n_c}, \quad x_l \in \{0, 1\}^{n_l}, \quad n = n_c + n_l$$

$$y = [y_c \ y_l]^T, \quad y_c \in \mathbb{R}^{p_c}, \quad y_l \in \{0, 1\}^{p_l}, \quad p = p_c + p_l$$

$$u = [u_c \ u_l]^T, \quad u_c \in \mathbb{R}^{m_c}, \quad u_l \in \{0, 1\}^{m_l}, \quad m = m_c + m_l$$

where x is the state of the system, whose components are distinguished between continuous x_c and $\{0, 1\}$ x_l (logical, translated to 0 – 1 form); y is the output vector; u is the command input, collecting both binary (on/off) commands (logical commands

translated to $\{0, 1\}$ format) u_i and continuous commands u_c ; $\delta \in \{0, 1\}^{n_l}$ and $z \in \mathbb{R}^{r_c}$ represent auxiliary logical and continuous variables, respectively.

Instead of using the usual logical rules to represent control rules, the logical facts involving continuous variables are translated to linear inequalities (3.5c). These inequalities are mixed with the equations defining continuous dynamics of the system.

Continuous time *mixed logical dynamical* (CT-MLD) systems can be defined by replacing $x(t + 1)$ by \dot{x} in (3.5a) [Bemporad and Morari, 1999]. Nonlinear version can be defined by changing the linear equations and inequalities in (3.5) to nonlinear functions.

$$\dot{x} = A_i x + B_i^1 u + B_i^2 \delta + B_i^3 z \quad (3.7a)$$

$$y = C_i x + D_i^1 u + D_i^2 \delta + D_i^3 z \quad (3.7b)$$

$$g_i \geq E_i^1 x + E_i^2 u + E_i^3 \delta + E_i^4 z \quad (3.7c)$$

where $x = \begin{bmatrix} x_r^T & x_b^T \end{bmatrix}^T$ with $x_r \in \mathbb{R}^{n_r}$ and $x_b \in \{0, 1\}^{n_b}$, y, u have a similar structure, and $z \in \mathbb{R}^{r_c}$ and $\delta \in \{0, 1\}^{n_l}$ are auxiliary variables. The inequalities (3.7c) have to be interpreted component-wise.

Tool support The HYSDEL (HYbrid System DEscription Language) is described in Section 7.9. It can be combined with Multi-Parametric Toolbox (MPT) [Kvasnica et al., 2004], which is a free Matlab toolbox for design, analysis and deployment of optimal controllers for discrete PWA and MLD systems.

3.3.4 Complementarity systems

Complementarity systems [van der Schaft and Schumacher, 2000, p.71–110] are described by

$$f(\dot{x}, x, y, u) = 0 \quad (3.8a)$$

$$0 \leq y \perp u \geq 0 \quad (3.8b)$$

where \perp denotes *complementarity* of y and u . Two vectors of variables of equal length are called *complementary*, if for all indices i the pair of variables (u_i, y_i) is a subject to complementarity condition ($y_i = 0 \vee u_i = 0$). In this formulation, the variables y_i, u_i play completely symmetric roles. Sometimes it is possible to choose y_i and u_i in such a way that the conditions can be written in a different manner

$$\dot{x} = f(x, u) \quad (3.9a)$$

$$y = h(x, u) \quad (3.9b)$$

$$0 \leq y \perp u \geq 0 \quad (3.9c)$$

The flow conditions in (3.8), (3.9) should be supplemented by event conditions which describe what happens when there is a switch between modes. In some applications such switching structures can be very elaborate.

Applications of linear complementarity systems include constrained mechanical systems, electrical networks with ideal diodes and other dynamical systems with piecewise affine relations, variable structure systems, constrained optimal control problems, projected dynamical systems, and so on [Heemels, 1999, p.27–39].

Linear complementarity systems

Usually complementarity systems are restricted to linear case, namely *linear complementarity systems* (LCS) [van der Schaft and Schumacher, 1998, Heemels et al., 2000, 2001, De Schutter and Heemels, 2004]. An LCS is governed by the equations

$$\dot{x}(t) = Ax(t) + B_1u(t) + B_2w(t) \quad (3.10a)$$

$$y(t) = Cx(t) + D_1u(t) + D_2w(t) \quad (3.10b)$$

$$v(t) = E_1x(t) + E_1u(t) + E_3w(t) + g \quad (3.10c)$$

$$0 \leq v(t) \perp w(t) \geq 0 \quad (3.10d)$$

where $v(t), w(t) \in \mathbb{R}^s, x(t) \in \mathbb{R}^n, u(t) \in \mathbb{R}^k, y(t) \in \mathbb{R}^l$ and $A, B_{1,2}, C, D_{1,2}, E_{1,2,3}$ are matrices of appropriate dimensions and g is a constant vector of appropriate dimensions. $w(t)$ and $v(t)$ are called *complementarity variables*.

Extended linear complementarity systems

Extended linear complementarity systems (ELCS) [De Schutter and Moor, 1995, Heemels et al., 2001, De Schutter and Heemels, 2004] are defined by equations

$$\dot{x}(t) = Ax(t) + B_1u(t) + B_2d(t) \quad (3.11a)$$

$$y(t) = Cx(t) + D_1u(t) + D_2d(t) \quad (3.11b)$$

$$g \geq E_1x(t) + E_1u(t) + E_3d(t) \quad (3.11c)$$

$$0 = \sum_{i=1}^p \prod_{j \in \phi_i} (g - E_1x(t) - E_1u(t) - E_3d(t))_j \quad (3.11d)$$

where $d(t) \in \mathbb{R}^r$ is an auxiliary variable. Condition (3.11d) is equivalent to

$$\prod_{j \in \phi_i} (g - E_1x(t) - E_1u(t) - E_3d(t))_j = 0 \quad \text{for each } i \in \{1, 2, \dots, p\} \quad (3.12)$$

due to the inequality conditions (as in inequality (3.11c)), with p groups of linear inequalities (for every index set ϕ_i) such that in every group at least one inequality should hold with equality.

3.3.5 Max-min-plus-scaling systems

Max-min-plus-scaling systems [De Schutter and van den Boom, 2001, 2002a, De Schutter and Heemels, 2004] is a class of discrete event systems that can be modelled using maximisation, minimisation, addition and scalar multiplication.

Symbols \vee and \wedge will be used to denote *maximisation* and *minimisation*, respectively. If $v, w \in \mathbb{R}$ then $v \vee w = \max(v, w)$ and $v \wedge w = \min(v, w)$.

Definition 3.3.3 (MMPS expression). A *max-min-plus-scaling* (MMPS) expression f of x_1, \dots, x_n is defined by the grammar (Backus Naur form):

$$f ::= x_i \mid \alpha \mid f_1 \vee f_2 \mid f_1 \wedge f_2 \mid f_1 + f_2 \mid \beta f$$

with $i \in \{1, \dots, n\}, \alpha, \beta \in \mathbb{R}$, and where f, f_1 and f_2 are again MMPS expressions. \square

In discrete time MMPS expressions are given by equations

$$x(t + 1) = \mathcal{M}_x(x(t), u(t)) \quad (3.13a)$$

$$y(t) = \mathcal{M}_y(x(t), u(t)) \quad (3.13b)$$

$$c \geq \mathcal{M}_c(x(t), u(t), d(t)) \quad (3.13c)$$

where \mathcal{M}_x , \mathcal{M}_y and \mathcal{M}_c are MMPS expressions in terms of the components of $x(t)$, the input $u(t)$ and the auxiliary variables $d(t)$, which are all real-valued.

Example 3.3.4. Let us consider Example 3.3.1. Then in MMPS form it can be defined as follows:

$$x(k + 1) = \min(x(k) + u(k), 1)$$

$$y(k) = x(k).$$

□

MMPS strength lies in a number of available analytical analysis methods and algorithms for model predictive controllers (MPC) generation, see, e.g., De Schutter and van den Boom [2002b].

3.3.6 Hybrid automata

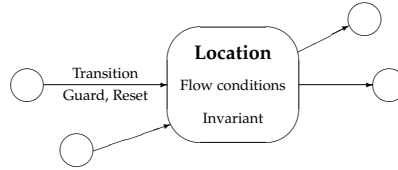


Figure 3.1: Hybrid Automaton

The hybrid automaton model [Alur et al., 1993, Henzinger, 1996] is one of the most popular approaches to model and analyse hybrid systems. Informally the hybrid automaton model was already explained on the bouncing ball (Section 2.2.1), thermostat (Section 2.2.2) and railroad gate control (Section 2.2.5) examples. Figure 3.1 depicts a conceptual view of a hybrid automaton. Essentially, a hybrid automaton combines two types of behaviour. Discrete changes are described by *transitions*, which are decorated with *action names*, *guards* and *resets*. The guards define conditions allowing to take the transition. The resets define changes of continuous state made by the transition and the action names are used as references in synchronisation. Continuous behaviour is described in *locations*. It is given by *flow conditions*, which define continuous change of the continuous state and usually are given by differential equations, and *invariants*, which restrict evolution in the location and usually are given by some kind of inequalities.

We provide a formal definition of hybrid automata to clarify the details.

Definition 3.3.5 (Hybrid automaton). A *hybrid automaton* is a collection $H = (X, L, Init, Inv, f, E, Guard, Assign, \Sigma)$ where:

- $X \subseteq \mathbb{R}^n$ is the *continuous state space* and $\mathbf{x} = (x_1, x_2, \dots, x_n)$, where $x_i \in \mathbb{R}, i = 1, 2, \dots, n$, represents the continuous dynamics.
- L is a finite set of *locations*.
- $Init \subseteq L \times X$ is a set of *initial location state pairs*.
- $Inv : L \rightarrow 2^X$ assigns to each location l an *invariant* to be satisfied by the state \mathbf{x} while in the location l .
- $f : L \rightarrow (X \rightarrow \mathbb{R}^n)$ assigns to each location l a *continuous vector field* f_l such that the state $\mathbf{x} \in X$ should satisfy $\frac{d}{dt}\mathbf{x} = f_l(\mathbf{x})$.
- $E \subseteq L \times \Sigma \times L$ is the set of *transitions*, also called *switches*, where Σ is a set of transition labels.
- $Guard : E \rightarrow 2^X$ assigns to each transition a *guard* that has to be satisfied by the state \mathbf{x} if the transition is taken.
- $Assign : E \rightarrow (X \rightarrow X)$ assigns to each transition an *assignment* that may alter the state \mathbf{x} when the transition is taken.

□

Tool support Hybrid automata are supported by several tools.

HyTech (Section 7.9) is an automatic tool for the analysis of embedded systems. HyTech computes the condition under which a linear hybrid system (each vector field is linear) satisfies a temporal requirement. Hybrid systems are specified as collections of automata with discrete and continuous components, and temporal requirements are verified by symbolic model checking. If the verification fails, then HyTech generates a diagnostic error trace.

d/dt (Section 7.9) is a tool for reachability analysis of continuous and hybrid systems with linear differential inclusions. The tool accepts as input a hybrid automaton where continuous dynamics are linear possibly with uncertain, bounded input of the form $\frac{dx}{dt} = Ax + u$ where u is input taking values in a bounded convex polyhedron U and invariants and transition guards are defined by convex polyhedra.

3.3.7 Hybrid behavioural automaton

Hybrid behavioural automaton was introduced in Julius et al. [2002], exhaustive description of HBA and some of its applications are available in Julius [2005]. It is a modification of hybrid automata based on the use of the behavioural theory [Polderman and Willems, 1998].

Here we present a definition of hybrid behavioural automaton (HBA) from Julius [2005, p.20–21].

Definition 3.3.6 (Hybrid behavioural automaton). A *hybrid behavioural automaton* is a collection $A = (L, W, \mathfrak{B}, E, T, Inv)$, where

- L is a set of locations of discrete states;

- W is the set of continuous variables taking values in \mathbb{W} ;
- \mathfrak{B} maps each location to its continuous behaviour. A behaviour is a subset of $\mathbb{W}^{\mathbb{R}_+}$;
- E is a set of events/labels;
- T is the set of transitions. Each transition is given as a 5-tuple $(l, \mathbf{a}, l', G, R)$. The triple (l, \mathbf{a}, l') is a subset of $L \times E \times L$, where l is the origin location, \mathbf{a} is the label of the transition, l' is the target location. $G := (\gamma, g)$ is the guard of the transition, where $\gamma : \mathfrak{B}(l) \times \mathbb{R}_+ \rightarrow \text{range}(v)$ and $g \subset \text{range}(\gamma)$, and $R : \mathfrak{B}(l) \times \mathbb{R}_+ \rightarrow 2^{\mathfrak{B}(l')}$ is the reset map of the transition.
- Inv is the invariant condition for the automata. It maps each location $l \in L$ to a pair $Inv(l) := (v, V)$, where $v : \mathfrak{B}(l) \times \mathbb{R}_+ \Rightarrow \text{range}(v)$ and $V \subset \text{range}(v)$.

□

A *hybrid execution* [Julius, 2005, p.22] can be thought of as a trajectory of type $(\mathbb{R}_+ \times \mathbb{Z}_+, L \times (\mathbb{W} \cup (E \times \mathbb{W}^{\mathbb{R}_+}))$) in the following sense. Each trajectory is associated with the subset of $(\mathbb{R}_+ \times \mathbb{Z}_+)$, on which it is defined. If ζ is a hybrid execution, the subset is denoted as T_ζ . It is assumed that for any hybrid execution ζ , its time axis T_ζ is structured such that

- $(0, 0) \in T_\zeta$;
- For any $t \in \mathbb{R}_+$ and $n \in \mathbb{Z}_+$ then $(t, n') \in T_\zeta$ for all nonnegative integer $n' < n$;
- For any $t \in \mathbb{R}_+$, if $(t, 0) \in T_\zeta$ then $(t', 0) \in T_\zeta$ for all nonnegative real $t' < t$.

The execution of HBA starts in a particular location, and proceeds with a continuous trajectory that satisfies the invariant condition in the location. Whenever there is a transition, whose guard is satisfied, a transition can occur. When a transition occurs, the location changes, and the continuous trajectory is reset to another one compatible with the reset map and the invariant condition of the new location.

3.3.8 Hybrid input/output automata

A *Hybrid input/output automaton* (HIOA) [Lynch et al., 2003] is a mathematical framework for modelling and analysis of hybrid systems. It has evolved from the well known *Input/Output Automata* [Lynch and Tuttle, 1989]. HIOA is a kind of nondeterministic, potentially infinite-state, state machine. The state of it is defined by state variables. It may be augmented with special *input* and *output* variables. The state changes in two ways.

- *Discrete transitions* change the state instantaneously.
- The state can change according to some *trajectory* when time passes. Trajectories are continuous or discontinuous functions that describe the evolution of the state variables along with the input and output variables over time intervals.

Let $V \subseteq \mathbb{V}$ be a set of variables. A *valuation* \mathbf{v} for V is a function that associates with each variable $v \in V$ a value in $\text{type}(v)$ (where $\text{type}(v)$ is a *static type* of v). We write $\text{val}(V)$ for the set of valuations for V . Let J be a left-closed interval of \mathbb{T} (where \mathbb{T} is a *time axis*) with left endpoint equal to 0. Then a J -trajectory for V is a function $\tau : J \rightarrow \text{val}(V)$, such that for each $v \in V, \tau \downarrow v \in \text{dtype}$ (where $\text{dtype}(v)$ is a *dynamic type* of v , i.e., a set of functions from left-closed intervals of \mathbb{T} to $\text{type}(v)$). A trajectory for V is a J -trajectory for V , for any J . We write $\text{trajs}(V)$ for the set of all trajectories for V .

Here we provide definitions of a hybrid automaton (different from the hybrid automaton defined in Section 3.3.6) and the hybrid input/output automaton from Lynch et al. [2003].

Definition 3.3.7 (Hybrid automaton). A *hybrid automaton* (HA)

$\mathcal{A} = (W, X, Q, \Theta, E, H, D, \mathcal{T})$ consists of

- A set W of *external variables* and a set X of *internal variables*, disjoint from each other ($V \triangleq W \cup X$).
- A set $Q \subseteq \text{val}(X)$ of *states*.
- A nonempty set $\Theta \subseteq Q$ of *start states*. Disjoint sets E of *external actions* and a set H of *internal actions* ($A \triangleq E \cup H$).
- A set $D \subseteq Q \times A \times Q$ of *discrete actions*. Action a is *enabled* in \mathbf{x} , iff $\exists \mathbf{x}' (\mathbf{x}, a, \mathbf{x}') \in D$.
- A set \mathcal{T} of trajectories for V such that $\tau(t) \upharpoonright X \in Q \forall \tau \in \mathcal{T}$ and $t \in \text{dom}(\tau)$.

Notation is extended, namely $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}'$ is used as a shorthand for $(\mathbf{x}, a, \mathbf{x}') \in D$, and \mathcal{A} is omitted, when it is clear from the context. Given the trajectory $\tau \in \mathcal{T}$, the first valuation of trajectory τ restricted to X ($\tau.\text{fval} \upharpoonright X$) is denoted by $\tau.\text{fstate}$. If τ is closed then $\tau.\text{lval} \upharpoonright X$ (last valuation of τ) is denoted by $\tau.\text{lstate}$. In addition, the following axioms should hold

- (Prefix closure) $\forall \tau \in \mathcal{T} \wedge \tau' \leq \tau \implies \tau' \in \mathcal{T}$.
- (Suffix closure) $\forall \tau \in \mathcal{T} \wedge t \in \text{dom}(\tau) \implies \tau \geq t \in \mathcal{T}$, where $\tau \geq t \triangleq (\tau \upharpoonright [t, \infty)) - t$.
- (Concatenation closure) Let τ_0, τ_1, \dots be a sequence of trajectories in \mathcal{T} such that, for each non-final index i , τ_i is closed and $\tau_i.\text{lstate} = \tau_{i+1}.\text{fstate}$. Then $\tau_0 \frown \tau_1 \frown \tau_2 \cdots \in \mathcal{T}$.

□

For details and explanations see Lynch et al. [2003].

Based on the definition of hybrid automata 3.3.7 a hybrid input/output automata is defined.

Definition 3.3.8 (Hybrid I/O automaton). A *hybrid I/O automaton* (HIOA) \mathcal{A} is a tuple \mathcal{H}, U, Y, I, O where

- $\mathcal{H} = (W, X, Q, \Theta, E, H, D, \mathcal{T})$ is a *hybrid automaton* (Definition (3.3.7)).
- U and Y partition W into *input* and *output* variables, respectively. Variables in $Z \triangleq Z \cup Y$ are called *locally controlled* and $V \triangleq W \cup X$.

- I and O partition E into *input* and *output* actions, respectively. Actions in $L \triangleq E \cup U$ are called *locally controlled* and $A \triangleq E \cup H$.

The following axioms should be satisfied

- (Input action enabling) $\forall \mathbf{x} \in Q \wedge \forall a \in I \exists \mathbf{x}' \in Q$ such that $\mathbf{x} \xrightarrow{a} \mathbf{x}'$.
- (Input trajectory enabling) $\forall \mathbf{x} \in Q \wedge \forall v \in \text{trajs}(U) \exists \tau \in \mathcal{T}$ such that $\tau.\text{fstate} = \mathbf{x}, \tau \downarrow U \leq v$, and either
 - $\tau \downarrow U = v$, or
 - τ is closed and some $l \in L$ is enabled in $\tau.\text{lstate}$.

□

For details and explanations see Lynch et al. [2003].

3.3.9 Process algebras for hybrid systems

Process algebra (in some sources referred to as a *process calculus*) forms a framework for modelling and analysis of broad class of systems. It has been developed in the computer science community throughout the last several decades starting from the innovative works of Hoare [1978] and Milner [1980], and many others developed further on by, see e.g., Bolognesi and Brinksma [1987], Baeten and Weijland [1990]. Process algebra provides a starting point for a structured approach and a systematic methodology for design and development of large and complex systems in a compositional and hierarchical way. It is a mathematical structure that is intended to describe processes and interaction amongst them.

A (*labelled*) *transition system* is used to describe the dynamic behaviour of the system. It consists of the states and a construct that defines changes of the states. Usually such changes are defined by transitions, which are defined as a relation over a subset of Cartesian product of the (source and target) states and labels, where labels refer to actions. The basic labelled transition system can be defined as follows.

Definition 3.3.9 (Transition system). A *labelled transition system* is a collection $TS = (S, \mathcal{A}, \rightarrow)$ where:

- S is a *set of states* denoted by s_1, s_2, \dots
- \mathcal{A} is a *set of actions* denoted by a_1, a_2, \dots
- $\rightarrow \subseteq S \times \mathcal{A} \times S$ is a *transition relation*.

We will write $s \xrightarrow{a} s$ instead of $(s, a, s) \in \rightarrow$ to simplify notation.

If an *initial state* $s_0 \in S$ is distinguished, then a structure (TS, s_0) is called a *rooted transition system*. □

A simple version of language for untimed systems can be defined as follows.

$$B ::= \mathbf{0} \mid a.B \mid B_1 + B_2 \mid B_1 \parallel_A B_2 \mid B[\sigma] \mid P$$

where $a \in \mathcal{A}$ is an action name, $A \subseteq \mathcal{A}$ is a *synchronisation set* and σ is a *renaming function*.

$\frac{}{a.B \xrightarrow{a} B}$	$\frac{B \xrightarrow{a} B'}{P \xrightarrow{a} B'} \quad P \triangleq B$	$\frac{B_1 \xrightarrow{a} B'_1, B_2 \xrightarrow{a} B'_2}{B_1 \parallel_A B_2 \xrightarrow{a} B'_1 \parallel_A B'_2} \quad a \in A$
$\frac{B \xrightarrow{a} B'}{B[\sigma] \xrightarrow{\sigma(a)} B'[\sigma]}$	$\frac{B_1 \xrightarrow{a} B'_1}{B_1 + B_2 \xrightarrow{a} B'_1}$ $\frac{B_1 \xrightarrow{a} B'_1}{B_2 + B_1 \xrightarrow{a} B'_1}$	$\frac{B_1 \xrightarrow{a} B'_1}{B_1 \parallel_A B_2 \xrightarrow{a} B'_1 \parallel_A B_2}$ $\frac{B_1 \xrightarrow{a} B'_1}{B_2 \parallel_A B_1 \xrightarrow{a} B_2 \parallel_A B'_1} \quad a \notin A$

Table 3.5: SOS rules for simple process algebra

- 0 is a *deadlock* (in some sources it is called *stop*), the process that does not show any behaviour.
- $a.B$ is an *action-prefix*. It first performs a and then engages in process B .
- $B_1 + B_2$ is a *choice* operator (in some sources referred to as an *alternative composition*). It selects one of the two processes *nondeterministically* and engages in it.
- $B_1 \parallel_A B_2$ is a *parallel composition* operator. It defines a process that executes processes B_1 and B_2 concurrently forcing the actions from A to synchronise. If the actions are not in A , they are executed in an interleaving manner, i.e., they are executed sequentially in an arbitrary order.
- $B[\sigma]$ defines an action *renaming*, where function σ takes an action name and changes it into another action name. $B[\sigma]$ behaves like B but with the actions renamed according to σ .
- $P \triangleq B$ is a *recursion*, where P is a process identifier and it behaves as B .

Typically, additional operators, e.g., disrupt [Bolognesi and Brinksma, 1987], communication merge [Baeten and Weijland, 1990], are added for convenience.

The behaviour of the processes is given by transition systems. *Structural operational semantics* (SOS) rules [Plotkin, 1981, 2003] is one of the ways to describe, how a transition system is built by the processes, stepwise. The rules have a form

$$\frac{\text{premises}}{\text{conclusions}}$$

and state that if the premises hold, then so do the conclusions. We exemplify use of the SOS rules for our example process algebra in Table 3.5.

In some cases [Milner, 1980, Baeten and Weijland, 1990] an *axiomatisation* (in some sources called *equational theory*) was taken as a starting point to get an algebra in purely mathematical sense. We are not going into the details and refer an interested reader to Milner [1980].

The presented process algebra can be used to specify untimed systems. However, the same basic ingredients are used in extended versions as timed [Baeten and Bergstra, 1991, D'Argenio and Brinksma, 1996, D'Argenio, 1999], mobile [Milner, 1999], probabilistic [D'Argenio et al., 1999, Andova, 2002], stochastic [D'Argenio et al., 1997, D'Argenio, 1999] and hybrid process algebras [Rounds and Song, 2003, Bergstra and Middelburg, 2005, van Beek et al., 2004, Cuijpers, 2004, Brinksma and Krilavičius, 2005]. We discuss hybrid process algebras in more details below and in Chapter 5.

Process algebra for hybrid systems

Bergstra and Middelburg [2005] proposes ACP_{hs}^{srt} , a process algebra for hybrid systems. It is an extension of combination of ACP^{srt} [Baeten and Middelburg, 2002] and ACP_{ps} [Baeten and Bergstra, 1997].

The ACP_{hs}^{srt} language contains the usual process algebraic operators extended to deal with hybrid behaviour and in addition is augmented with operators to express continuous-time evolution. Continuous-time components in the process algebra for hybrid systems are defined by signals, and consequently operators to define processes *emitting signals* ($\psi \wedge P, \phi \curvearrowright P$) are introduced. The first operator ($\psi \wedge P$) emits the signal ψ and then behaves as P . The second expression ($\phi \curvearrowright P$) evolves according to ϕ until P performs its first action (the action is not allowed to violate ϕ). State changing transition in ACP_{hs}^{srt} is performed by transition operator $\chi \curvearrowright P$ with χ defining transition conditions and P the process that continues afterwards.

Alternative composition ($P_1 + P_2$) defines an arbitrary choice between processes. The choice is resolved by one of them performing its first action. The choice between idling processes will be postponed until one of them can perform its first action. If both processes cannot idle any longer, postponement is not an option. If one of processes cannot idle any longer and choice has not yet been resolved, the the choice will not be resolved in its favour. While idling, the conjunction of signals emitted by processes is emitted.

Parallel composition ($P_1 \parallel P_2$) has interleaving semantics and behaves in the following way: (1) first either P_1 or P_2 performs its first action and next it proceeds in parallel with the process following that action and the processes that did not perform an action; (2) if their first actions can be performed synchronously, first P_1 and P_2 perform their first actions synchronously and next it proceeds in parallel with processes following those actions. If processes choose to idle then the following rules apply: (1) one of the processes can perform an action only before other process starts performing an action or deadlocks; (2) processes can synchronise at some time moment. The state transitions caused by performing the first action must not be precluded by the other process: (1) the signal emitted by the other process must hold in the state immediately before and after the transition; (2) if the other process is idling when the action is performed, a state evolution with discontinuities for the state variable of which the value changes by the transition must be possible. There is only one action left when actions are performed synchronously.

Unfortunately, strong bisimulation is not a congruence in ACP_{hs}^{srt} .

Example 3.3.10 (Thermostat). To illustrate ACP_{hs}^{srt} an example from Bergstra and Middelburg [2003] is presented.

Initially, the temperature is 18 °C and the heating is on. While the heating is on, the temperature T in the room goes up according to the differential equation $\dot{T} = -T + 22$. When the temperature becomes 20 °C, the heating will be turned off. While the heating is off, the temperature in the room goes down according to the differential equation $\dot{T} = -T + 17$. When the temperature becomes 18 °C, the heating will be turned on again.

The recursive specification of the thermostat consists of the following equations:

$$\begin{aligned}
Th &= (T = 18) \wedge Th^{\text{on}}, \\
Th^{\text{on}} &= (18 \leq T \leq 20 \wedge \dot{T} = -T + 22)^{\ulcorner} \\
&\quad \sigma_{\text{rel}}^* \left((T = 20) : \rightarrow \left((T^\bullet = \bullet T)^{\ulcorner} \widetilde{\text{turn-off}} \cdot Th^{\text{off}} \right) \right), \\
Th^{\text{off}} &= (18 \leq T \leq 20 \wedge \dot{T} = -T + 17)^{\ulcorner} \\
&\quad \sigma_{\text{rel}}^* \left((T = 18) : \rightarrow \left((T^\bullet = \bullet T)^{\ulcorner} \widetilde{\text{turn-on}} \cdot Th^{\text{on}} \right) \right).
\end{aligned}$$

The signal transition operator \ulcorner and the signal evolution operator \ulcorner are needed here to make precise that the temperature in the room does not change instantaneously at the points of time at which the heating is turned off or on and that the temperature in the room changes continuously as described above during the periods in between. *Conditional progress* is expressed by operator $(\phi : \rightarrow P)$. It behaves as P if conditions ϕ hold at its start. *Relative delay* operator $\sigma_{\text{rel}}^*(P)$ idles for time rel and then proceeds as P . \square

Hybrid process algebra HyPA

Hybrid process algebra HyPA [Cuijpers, 2004, Cuijpers and Reniers, 2005] is a conservative extension of ACP [Baeten and Weijland, 1990] augmented with a disrupt operator from LOTOS [Bolognesi and Brinksma, 1987]. Variants of flow and a re-initialisation-clause from van der Schaft and Schumacher [2000] are added to handle hybrid behaviour.

Strong bisimulation is not a congruence relation with respect to the parallel composition of the subsystems in HyPA. However, *robust bisimilarity* [Cuijpers, 2004, p.68–70] and *stateless bisimilarity* [Cuijpers and Reniers, 2005] are congruences [Mousavi, 2005, p.163–166].

Hybrid transition system [Cuijpers, 2004, p.60–61] defines dynamic behaviour of the HyPA.

Definition 3.3.11 (Hybrid transition system). A *hybrid transition system* is a six-tuple $\langle X, \mathcal{A}, \sigma, \mapsto, \rightsquigarrow, \checkmark \rangle$, consisting of a *state space* X , a *set of action labels* \mathcal{A} , a *set of flow labels* Σ , and *transition relations* $\mapsto \subseteq X \times \mathcal{A} \times X$ and $\rightsquigarrow \subseteq X \times \Sigma \times X$, and *termination predicate* $\checkmark \subseteq X$. \square

For the transitions following notation is used: \mapsto denotes a discrete transition; \rightsquigarrow denotes a signal transition; \rightarrow denotes any type of transition.

HyPA language includes classical process algebraic operators extended to deal with hybrid behaviour and several additional operators to handle continuous-time evolution and its interaction with discrete counterpart. *Flow clauses* ($c \in C$) model never terminating physical behaviour. Flows are described by flow predicate, and are called solutions of that predicate. Discrete changes of continuous behaviour are modelled by *re-initialisation* operator ($d \gg p$), where d is a *re-initialisation clause* defining changes of signal.

Alternative composition ($P \oplus P$) models a nondeterministic choice between processes. It is nondeterministic for both the discrete and continuous actions. The passing of time influences the valuation of the model variables and can introduce choices in the system behaviour. The choice is done before an action.

Parallel composition ($P \parallel P$) models concurrent execution of processes. Discrete actions are executed in an interleaving manner, i.e., if two actions synchronise, they are executed at the same time and are observable as a new action described by a *communication function* ($\gamma \in \mathcal{A} \times \mathcal{A} \mapsto \mathcal{A}$). If actions do not synchronise they are executed sequentially in an arbitrary order. Flow clauses, in contrast to discrete actions, have to synchronise all the time. They synchronise only if they coincide.

Example 3.3.12 (Steam boiler). The boiler process [Cuijpers and Reniers, 2003], [Cuijpers, 2004, p.66–67] consists of a volume of water V [m³], an inflow of water Q_i [$\frac{\text{m}^3}{\text{sec}}$] and a steam production Q_s [$\frac{\text{m}^3}{\text{sec}}$]. The relation amongst volume, inflow and steam production, is described by the differential equation $\dot{V} = Q_i - Q_s$. The steam production is determined by the Heater process, which limits it between the constants Q_{\min} [$\frac{\text{m}^3}{\text{sec}}$] and Q_{\max} [$\frac{\text{m}^3}{\text{sec}}$]. We do not have more information on the Heater, and can not describe the behaviour of s in more detail. The inflow is determined by a Valve process, which can be opened and closed using the actions ro and rc , respectively. If the valve is open, the inflow to the boiler has value Q_{in} [$\frac{\text{m}^3}{\text{sec}}$]. If it is closed, the inflow is 0 [$\frac{\text{m}^3}{\text{sec}}$]. Furthermore, there is a Controller that interferes with the valve by telling it to open and close using the action so and sc . The goal of this controller, is to keep the volume of water between the constants V_{\min} [m³] and V_{\max} [m³]. The controller uses a clock t [sec] to measure the sampling time T [sec] between interactions. Furthermore, it takes a margin of V_{safe} [m³] into account, to compensate for errors due to the sampling time. The total system is the parallel composition of the Water process, the Heater, the two modes of the Valve, and the Controller, over which communication is enforced through the definitions $op = ro \gamma so$, $cl = rc \gamma sc$, and $H = \{so, sc, ro, rc\}$:

$$\begin{aligned}
 \text{Water: } & (w \mid \dot{w} = v - s) \\
 \text{Heater: } & (s_{\min} \leq s \leq s_{\max}) \\
 \text{ValveOpen: } & (v = v_{\text{in}}) \blacktriangleright (rc \odot \text{ValveClose}) \\
 \text{ValveClose: } & (v = 0) \blacktriangleright (ro \odot \text{ValveOpen}) \\
 \text{Controller: } & [t \mid t^+ = 0] \gg \left(t \begin{array}{l} \dot{t} = 1 \\ t \leq T \end{array} \right) \blacktriangleright [t^- = T] \gg \\
 & \left(\begin{array}{l} [w^- \geq w_{\max} - w_{\text{safe}}] \gg sc \odot \text{Controller} \oplus \\ [w_{\min} + w_{\text{safe}} \leq w^- \leq w_{\max} - w_{\text{safe}}] \gg \text{Controller} \oplus \\ [w^- \leq w_{\max} + w_{\text{safe}}] \gg so \odot \text{Controller} \end{array} \right) \\
 \text{Boiler: } & \partial_H (\text{Water} \parallel \text{Heater} \parallel (\text{ValveOpen} \oplus \text{ValveClosed}) \parallel \text{Controller})
 \end{aligned}$$

In the example \odot denotes sequential composition, i.e., sequential executions of processes. Disrupt operator \blacktriangleright , originally introduced in LOTOS [Bolognesi and Brinksma, 1987], defines a special type of sequential composition, such that the second process

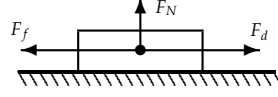


Figure 3.2: Dry friction

can take over the executions without waiting for the first one to terminate. Encapsulation $\partial_H(P)$ models that certain discrete actions (from the set $H \subseteq \mathcal{A}$) are blocked during the execution of the process P . In this example it is used to model that synchronisation between discrete actions is enforced. \square

Tool support Linearization² of HyPA is described in van den Brand et al. [2005]. Simulation tool for HyPA is presented in Schouten [2005].

Hybrid χ

Hybrid χ [Schiffelers et al., 2003, van Beek et al., 2004] is a process algebraic formalism for modelling and analysis of hybrid systems.

Continuous-time behaviour in Hybrid χ is modelled by a *signal emission* operator (denoted $u \curvearrowright P$) and a *delay predicate* ($[v]$). u is a predicate over variables that should hold initially over extended valuation, and then the process behaves as P . Otherwise it is considered to be inconsistent. Delay predicate defines change of the χ variables over time with conditions (v) usually in the ODE/DAE form.

In Schiffelers et al. [2003] version of Hybrid χ two types of choice amongst the processes are adopted: a *choice* and an *alternative composition* (denoted $P \parallel P$). The first one is nondeterministic for the discrete actions and does not make a choice for the continuous time transitions. The second is the same for the discrete actions, and uses the *weak time-determinism* principle, which means that the passage of time cannot result in making a choice, if both alternatives can perform the transition with the same trajectory and the same time step. If one of the processes can perform a time transition and the other cannot, then the alternative is lost. In van Beek et al. [2004] choice operator is abandoned and only *alternative composition* is used. It allows nondeterministic choice between actions, and continuous variables should synchronise (passage of time cannot result in making choice).

Parallel composition ($P \parallel P$) of χ adopts usual interleaved semantics for actions. The time behaviour of processes is synchronised over channels (H is a set of channels labels) by means of the *send action* $h !! \mathbf{e}_n$ and *receive action* $h ?? \mathbf{x}_n$, where h is a channel. \mathbf{e}_n and \mathbf{x}_n denote the values of expressions which are sent or received over the channel, respectively.

Example 3.3.13 (Dry friction). In Figure 3.2 a dry friction [van Beek et al., 2004] phenomenon is depicted. A driving force F_d is applied to a body on a flat surface with frictional force F_f . When the body is moving with positive velocity v , the frictional

²Informally, *linearization* is a procedure of transforming a process algebraic expression into an equivalent system of *linear process equations*, i.e., a process algebraic expression containing only basic process algebraic operators (action prefix, alternative composition) and a special form of recursion [Usenko, 2002].

force is $F_f = \mu F_n$ ($F_n = mg$). When the velocity is zero and $|F_d| < \mu_0 F_N$, the frictional force neutralises the driving force.

```

⟨cont  $x, v$ , alg  $F_d$ 
,  $x = 0, v = 0$ ,
stop  $\mapsto v = 0, -\mu_0 F_N \leq F_d \leq \mu_0 F_N \parallel [F_d \leq -\mu_0 F_N \rightarrow \text{skip}] ; \text{neg}$ 
       $\parallel [F_d \geq -\mu_0 F_N \rightarrow \text{skip}] ; \text{pos}$ 
pos  $\mapsto m\dot{v} = F_d - \mu F_N, v \geq 0 \parallel F_d \leq \mu_0 F_N \rightarrow \text{skip} ; \text{stop}$ 
       $\parallel [F_d \leq -\mu_0 F_N \rightarrow \text{skip}] ; \text{neg}$ 
neg  $\mapsto m\dot{v} = F_d + \mu F_N, v \leq 0 \parallel F_d \geq -\mu_0 F_N \rightarrow \text{skip} ; \text{stop}$ 
       $\parallel [F_d \geq \mu_0 F_N \rightarrow \text{skip}] ; \text{pos}$ 
|  $F_d = f(\text{time}), \dot{x} = v$ 
skip ; neg  $\parallel$  skip ; stop  $\parallel$  skip ; pos
⟩

```

Semicolon (;) in Hybrid χ denotes a sequential composition. \square

Tool support The hybrid χ simulator is described in Section 7.9. For more information about Chi simulator (compiler) see <http://chi-compiler.gforge.se.wtb.tue.nl>.

Φ -calculus

Φ -calculus [Rounds and Song, 2003] extends π -calculus [Milner, 1999] by adding *active environments* that flow continuously over time. This allows to model hybrid mobile systems, i.e., systems that can reconfigure themselves, certain components can appear and disappear from the system.

The system defined in Φ -calculus can evolve in several different ways.

- System may execute discrete actions that change only the process expressions.
- System may engage in flow actions that change only the environment continuously.
- Φ -specific actions that change both the environment and a process expressions discretely.

3.3.10 Masaccio

MASACCIO [Henzinger, 2000, Henzinger et al., 2001] is a hierarchical model for hybrid dynamical systems, based on Statecharts [Harel, 1987] and hybrid automata (Section 3.3.6). It is a formal model for hybrid dynamical systems which are built from *atomic discrete components* (difference equations) and *atomic continuous components* (differential equations) using *parallel* composition, *serial* composition (choice), *renaming*,

hiding of variables and *components*. Components communicate via *interfaces*, which determine the possible use of the components, and *executions* (a set of executions), which define the potential behaviours of the component.

Atomic components are specified by guarded jump actions. Atomic continuous components are specified by flows, which can be defined using differential equations.

MASACCIO requires the components dependency relation to be acyclic to guarantee the existence of input/output values for atomic discrete components and trajectories for atomic continuous components. This condition is quite restrictive and it eliminates some potential sources of nondeterminism.

3.3.11 Charon

CHARON [Alur et al., 2000, 2001] is a language for hierarchical specification of the interacting hybrid systems (hierarchical approach). It extends Statecharts [Harel, 1987] by adding continuous behaviour and formalising it for modelling and simulation needs.

The building block for describing the system architecture in CHARON is an *agent* that communicates with its environment via shared variables. Agents can be composed using a *parallel composition* to model concurrency, a *hiding* of variables to restrict sharing of information and an *instantiation* to support reuse. The flow inside an agent is described using a *mode*, where the mode is a hierarchical state machine that can have submodes and transitions connecting them. Modes communicate only via well-defined entry and exit points. State of system is changed by:

- *Discrete updates*, which are specified by *guarded actions*. Interleaving semantics is assumed for parallel composition of actions.
- *Continuous updates*, which are specified only for *analog variables* by several types of constraints, which can be defined at different levels of mode hierarchy:
 - *Differential* constraints (differential equations);
 - *Algebraic* constraints (algebraic equations);
 - *Invariants* (e.g., inequalities).

CHARON supports nondeterminism for both discrete and continuous updates. Operational and trace semantics are defined for modes and agents. For further information see [Alur et al., 2000, 2001].

Tool support Java implementation of CHARON toolkit is available (Section 7.9).

3.3.12 Bond graphs

Bond graphs [Broenink, 1999, van Amerongen and Breedveld, 2003] are a graphical description of dynamic behaviour of physical systems, based on energy and energy exchange. Bond graphs are directed graphs, where:

- Nodes are called:
 - *Elements* and represent basic physical components;

3. OVERVIEW OF MODELS FOR HYBRID SYSTEMS

- *Junctions* and represent energy conservation laws.
- Edges are called *bonds* and represent the energy exchange and the energy flow direction (*causal stroke*). Bonds can be interpreted in two ways:
 - As an interaction of energy. The connected subsystems form a load to each other by their energy exchange;
 - As a bilateral signal flow. The connection is interpreted as two signals, an effort and flow, flowing in opposite direction, thus determining the computational direction of the bond variables.

Every edge and node are assigned algebraic differential equations (DAE), which describe the behaviour of the system.

The main elements of bond graphs are following:

- *Storage elements*:
 - *Generalised displacement*, for a q -type variables (denoted C), e.g., capacitors (charge), springs (displacement), volume, angular displacement;
 - *Generalised momentum*, for a p -type variables (denoted I), e.g., inductors (flux linkage), masses (momentum), angular momentum, pressure momentum.
- *Dissipations* (denoted R), e.g., electric resistor, mechanical friction.
- *Sources* (denoted Se, Sf), e.g., electric mains (voltage), gravity (force), pump (flow).
- *Conversions*:
 - *Transformers* (denoted TF), e.g., electric transformers, toothed wheels;
 - *Gyrators* (denoted GY), e.g., electro-motor, centrifugal pump.
- *0- and 1-junctions* (denoted 0 1) for ideal connecting two or more sub-models.
- *Distributions*:
 - *Effort* (denoted E), e.g., force, torque, voltage, pressure;
 - *Flow* (denoted F), e.g., velocity, angular velocity, current, volume-flow.

Tool Support Bond graphs modelling is supported by the industrial strength 20-sim modelling and simulation package³ (Section 7.9).

³<http://www.20sim.com/>.

3.3.13 Modelica

Modelica™ [Fritzson and Engelson, 1998, Mod, 2005]⁴ is a language for hierarchical physical modelling. It is an object oriented language for modelling of physical systems for the purpose of simulation. It is

- *Non-causal*, i.e., based on differential and algebraic equations;
- *Multi-domain*, i.e., it is possible to combine different physical domains in one model;
- Has a general type system that unifies object-orientation, multiple inheritance and templates within a *class* construct.

Models are declared as classes with interfaces (*connectors*), which contain all information required to define the interaction. Models can be extended using inheritance and encapsulation mechanisms. The equations describe models *non-causally*.

Basic semantics of Modelica™ is given by the hybrid DAE (*hybrid differential algebraic equations*). A hierarchical Modelica™ model is transformed to the following form $v := [\dot{x}; x, y; t; m; \text{pre}(m); p]$

$$\begin{aligned}c &:= f_c(\text{relation}(v)) \\m &:= f_m(v, c) \\0 &= f_x(v, c)\end{aligned}$$

where

- p are Modelica™ variables declared as parameters or constants;
- t is a Modelica™ variable *time*;
- $x(t)$ are Modelica™ variables of the type *REAL*, appearing differentiated;
- $m(t_e)$ are Modelica™ variables of the following types *DISCRETE REAL*, *BOOLEAN*, *INTEGER*, which are unknown. They change value only at event instants t_e . $\text{pre}(m)$ are the values of m immediately before the event (left-limit);
- $y(t)$ are algebraic Modelica™ variables of type *REAL*;
- $x(t_e)$ are the conditions of all **if** expressions including **when** clauses;
- $\text{relation}(v)$ is a *relation* containing variables v_i (variables from inequalities).

Tool Support Modelica™ language is used in several tools, which are described in Section 7.9.

⁴For more information see <http://www.modelica.org/>

3.4 Conclusions

In this chapter we surveyed a collection of hybrid formalisms. By no means, the collection is complete, as only typical representatives of different types of approaches were selected. Nevertheless, the formalisms reveal critical qualifications necessary to handle hybrid phenomena in its diverse appearances.

Thus, for the control theory based approaches, usually it is easy to adopt control theory tools and results (not always, for example, it does not work well with Lyapunov stability theory). But at the same time discrete behaviour is often oversimplified. Frequently such formalisms are not modular, i.e., techniques as parallel composition (interconnection) and choice are not available. It complicates work with big and complex systems. Without proper modularity support reuse of components is awkward and distribution of work and responsibilities is not obvious, if not intricate.

The situation in the approaches originating from computer science is almost opposite. Formalisms as automata, process calculi and hierarchical approaches are modular and support hierarchical and (or) compositional modelling. Diverse techniques to specify and analyse discrete communication are available. But serious drawbacks can be found on the continuous side. Frequently continuous behaviour is oversimplified. Often it is considered as something what should be specified and analysed by somebody else, for example, control theorists or engineers. Not seldom tools and analysis techniques are just future work.

A third group, the simulation languages, provide good tool support and are often used in practice. However, the theoretical foundations usually are not so precise. Syntax and semantics are not suited for further theoretical research, and complicates implementation of new developments in the area.

Diversity of approaches and often contradictory requirements show that the ultimate formalism is not there yet. It may be a quest for Utopia, but analysis of strong and weak points of formalisms and studies of examples contribute for gradual improvement of methods and techniques dealing with hybrid systems.

Several development paths for hybrid systems formalisms are plausible. Practitioners are likely to further specialise in the approaches that smoothly handle particular aspects of hybrid phenomena and satisfactorily solve specific problems. Another direction is development of general methods and techniques that are applicable to broader classes of hybrid systems and are suitable for solving general problems, but at the same time may lack some performance or decisiveness. We believe that both approaches contribute equally to the research field.

We think that a unifying formalism would contribute to the hybrid systems research. That is an approach where both continuous and discrete behaviours are treated on the equivalent terms. Moreover, we would like to emphasise the importance of the modular development techniques, especially compositionality. It is not just convenient, but indispensable in the development of large systems. Furthermore, it is a necessary condition for trustworthy reuse of components. Definitely, such unifying formalism should provide a natural and foundational treatment of hybrid phenomena. Of course, besides all these nice theoretical properties, it should be applicable in practice, that is, techniques for modelling and analysis eventually should result in effective tools.

Everything that's extreme is difficult. The middle parts are done more easily. The very centre requires no effort at all. The centre is equal to equilibrium. There's no fight in it.

Daniil Kharms

4

Stability analysis for hybrid automata

4.1 Introduction

We present stability analysis for hybrid automata in this chapter. Some results were published in Langerak et al. [2003a,b]. In this work we demonstrate that cooperation between computer science and control theory can yield fruitful results.

The problem of hybrid systems stability has been known to be non-trivial. By now there are many stability results, some of which we discuss in Section 4.1.2. However, most of these results are applicable only to certain classes of hybrid systems. In this chapter we propose a general technique for stability analysis of hybrid automata. We exploit structure of automata and cyclic nature of stability of hybrid automata, as well as means to estimate influence of switches and continuous behaviour in locations on stability of hybrid automata. So-called *gains* define effect of switches and locations to stability. However, the procedures to estimate gains are not always known. We describe procedures for the gains estimation of certain class of automata, namely *Linear Continuous Hyperplane Hybrid Automaton*. However, we believe that this technique can be relatively easily extended to a wider class of hybrid automata.

4.1.1 Stability

Stability analysis is a well-established research area. The notion of *stability* is very important in the systems and control theory. Intuitively, stability means that small disturbances and small initial deviations should not alter the system's behaviour significantly. We provide a formal definition of stability in the following section. A wider discussion on stability is available in Willems [1970], Polderman and Willems [1998] and in most systems and control theory textbooks.

4.1.2 Hybrid stability

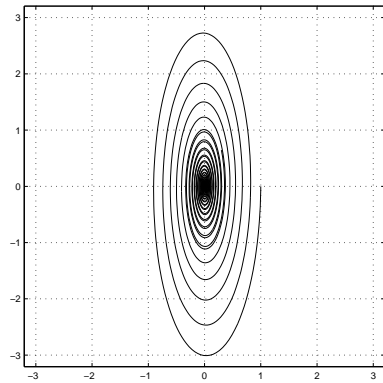


Figure 4.1: Trajectory of $\dot{x} = A_1x$

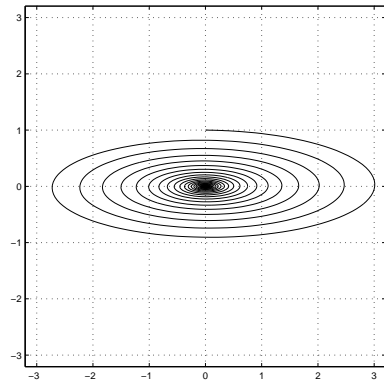


Figure 4.2: Trajectory of $\dot{x} = A_2x$

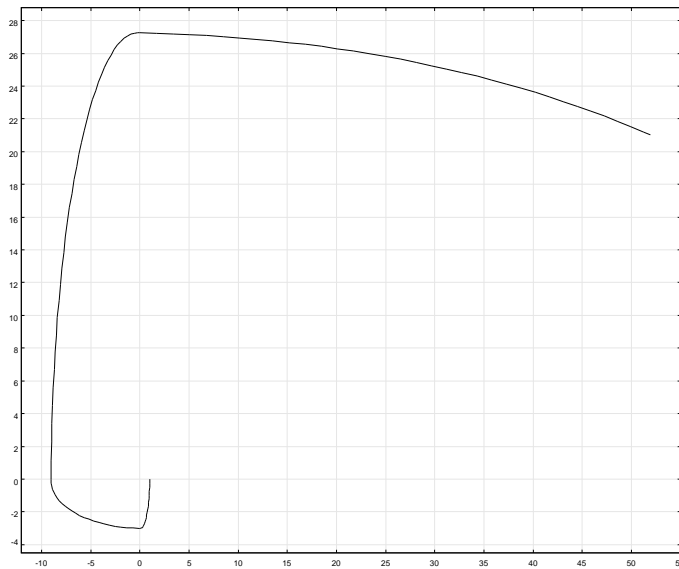


Figure 4.3: Unstable hybrid system

Hybrid systems can have very complex behaviour. Even very simple continuous dynamics can harbour some nasty behaviour (see Section 2.2.1). It is not surprising that stability analysis for hybrid systems is often very complicated and results are available only for certain, usually small, classes of hybrid systems.

We present a motivating example to show that difficulties can be encountered even in apparently simple cases.

Example 4.1.1. Consider the hybrid system taken from Branicky [1998]

$$\dot{x} = \begin{cases} A_1x, & \text{if } x_1x_2 < 0 \\ A_2x, & \text{if } x_1x_2 > 0 \end{cases}$$

$$A_1 = \begin{bmatrix} -1 & 10 \\ -100 & -1 \end{bmatrix} \quad A_2 = \begin{bmatrix} -1 & 100 \\ -10 & -1 \end{bmatrix}$$

Both dynamics $\dot{x} = A_1x$ and $\dot{x} = A_2x$ are stable, as the trajectories in Figures 4.1 and 4.2 indicate. However, the hybrid system with $\dot{x} = A_1x$ in the second and fourth quadrants, and $\dot{x} = A_2x$ in the first and third quadrants with initial conditions $x(0) = [1 \ 0]^T$ is unstable, see Figure 4.3. \square

Related Research Stability is one of the major problems in the area of hybrid dynamical systems. By now there are many results on the stability of hybrid systems, for an overview see Ye et al. [1998], D.Liberzon and A.S.Morse [1999], Michel [1999], Lygeros and Sastry [1999], DeCarlo et al. [2000], Davrazos and Koussoulas [2001].

Classical Lyapunov theory is a general approach to study stability of dynamical systems. *Lyapunov functions* (Definition 4.4.1) belong to a special class of functions that can provide an upper-bound to a system state without explicitly calculating it. Several attempts have been made to apply modified versions to stability analysis of hybrid systems. Usually, two main courses are taken: (1) *Common Lyapunov function* and (2) *Multiple Lyapunov functions* approaches.

Common Lyapunov function: This approach is based on checking the existence of a Lyapunov function common to all locations [D.Liberzon and A.S.Morse, 1999]. It is limited to rather small class of linear hybrid systems [Davrazos and Koussoulas, 2001]. One serious drawback is the difficulty to construct a common Lyapunov function: it is easy to design examples of hybrid systems that are stable but do not have a common Lyapunov function.

Multiple Lyapunov functions. This approach is based on constructing a separate Lyapunov function for each continuous subsystem and restricting the systems switching behaviour [Branicky, 1997, 1998, Žefran and Burdick, 1998, Michel, 1999, S.Pettersson and B.Lennartson, 1999, Lygeros and Sastry, 1999]. Usually, in this approach it is required that the value of the Lyapunov function does not increase every time the same subsystem is visited. In general it is not clear how to check that the sequence is non-increasing. To solve the problem Koutsoukos and Antsaklis [2002] constructs piecewise linear Lyapunov functions and Pettersson and Lennartson [1996], Johansson and Rantzer [1998], Mignone et al. [2000] construct piecewise quadratic Lyapunov functions.

Other approaches are *Modifying Theorems*, *Poincaré mappings* and *Lagrange stability* [Hassibi et al., 1999]. For more information on these approaches and other hybrid stability issues see Kourjanski and Varaiya [1996], Ye et al. [1998], D.Liberzon and A.S.Morse [1999], Michel [1999], Lygeros and Sastry [1999], DeCarlo et al. [2000], Davrazos and Koussoulas [2001], Rubensson [2003].

4.2 Notions of stability and hybrid stability

4.2.1 Stability of dynamical systems

Stability is a very important property in the design of dynamic systems. Intuitively, stability means that small disturbances should not alter the system behaviour completely, i.e., small causes produce small effects. There are several slightly different definitions of stability, based on this intuition.

We will settle for *autonomous systems* (or *closed systems*), i.e., complete systems, where all behaviour is specified and the future of every trajectory is completely determined by its past [Polderman and Willems, 1998, p.67].

Let $x = [x_1, x_2, \dots, x_n]^T$ ($x \in \mathbb{R}^n$), let

$$\dot{x} = f(x, t) \quad (4.2)$$

be a dynamical system and let x_e be an *equilibrium state* of the system such that $f(x_e, t) = 0$ for all t . We will denote the solution at t with given initial conditions x_0 at t_0 by $x(t; x_0, t_0)$.

Here we present one of the basic stability definitions from Willems [1970, p.3-12].

Definition 4.2.1 (Stability of equilibrium state). The equilibrium state x_e , or the equilibrium solution $x(t) = x_e$, is called *stable* if for any given t_0 and $\varepsilon > 0$, there exists a positive $\delta(\varepsilon, t_0)$ such that

$$\|x_0 - x_e\| < \delta \implies \|x(t; x_0, t_0) - x_e\| < \varepsilon \quad \forall t \geq t_0.$$

The equilibrium state is called *unstable*, if it is not stable. \square

A detailed study of the stability of dynamical systems is given in Willems [1970].

4.2.2 Stability of hybrid automata

The notion of stability presented in Section 4.2.1 can be lifted to hybrid systems. In the literature it is done in the several different ways, depending on the chosen hybrid system model. In most cases, some kind of *hybrid time trajectories* are defined, and a *mapping* from the hybrid time trajectories to the corresponding continuous state space is defined. Then stability is defined on the set of such mappings.

In Ye et al. [1998] *motions* are introduced as a basis for *hybrid dynamical systems* and then different types of stability are defined on the set of all motions and *invariant sets*.

For hybrid automata Lygeros et al. [2003], Simić et al. [2001] implicitly or explicitly define *hybrid time trajectories* and *hybrid traces* (or *executions*). Then the stability is defined on the set of all hybrid traces and an equilibrium or an invariant set, i.e., all hybrid traces should be stable w.r.t. an equilibrium, or in more general case, an invariant set.

Since we investigate stability of hybrid automata, we will define hybrid traces for it and base our stability definition on them.

Definition 4.2.2 (Hybrid trace). A *hybrid trace* of a hybrid automaton is an infinite or finite sequence of the form $\sigma = x_1 e_1 x_2 e_2 \dots x_{m-1} e_{m-1} x_m$, with an associated monotonically increasing time sequence $\tau_0 \tau_1 \dots \tau_m$ (where $\tau_0 = 0$ and $\tau_i \in \mathbb{R} \cup \{\infty\}$), such that

- Each e_i is a transition from l_i to l_{i+1} ;
- Each x_i is a mapping from $[\tau_{i-1}, \tau_i]$ to \mathbb{R}^n satisfying $\frac{d}{dt}x_i = f_i(x_i)$;
- Initial (*Init*) and switching (*Guard*) constraints and assignments (*Assign*) are respected, thus $(l_1, x_1(0)) \in \text{Init}$, and $\forall 1 \leq i \leq m - 1$ holds $(e_i, x_i(\tau_i)) \in \text{Guard}$ and $(e_i, x_i(\tau_i), x_{i+1}(\tau_i)) \in \text{Assign}$.

□

We will now extend the notion of stability from Section 4.2.1 to define stability of hybrid automata. We start from more general *multiple equilibria* stability with potentially different equilibria in each location.

Definition 4.2.3 (Multiple equilibria stability of hybrid automata). Let H be a hybrid automaton with n locations and $x_1^*, x_2^*, \dots, x_n^*$ be equilibria for each location of the hybrid automaton. The hybrid automaton H is called *multiple equilibria stable* iff for all hybrid traces $x_1 e_1 x_2 e_2 \dots$ with the initial state $x_1(0)$ and $l(x_i)$ a location corresponding to x_i :

$$\forall \varepsilon > 0 \exists \delta > 0 \|\mathbf{x}_1(0) - \mathbf{x}_{l(x_1)}^*\| < \delta \text{ and } \forall i \forall t \in [\tau_{i-1}, \tau_i] \text{ holds } \|x_i(t) - x_{l(x_i)}\| < \varepsilon.$$

An automaton that is not stable is called *unstable*. □

In this study we will restrict to a simpler version of stability with a unique equilibrium for all locations.

Definition 4.2.4 (Stability of hybrid automaton). A hybrid automaton is called *stable* iff $\forall \varepsilon > 0 \exists \delta > 0$ such that $\|\mathbf{x}^0\| < \delta$ for all hybrid traces $x_1 e_1 x_2 e_2 \dots$ with $x_1(0) = \mathbf{x}^0$ and $\forall i \forall t \in [\tau_{i-1}, \tau_i] : \|x_i(t)\| < \varepsilon$. An automaton that is not stable is called *unstable*. □

Definition 4.2.5 (Stable locations). We call a location of a hybrid automaton *stable*, if dynamics in the location are stable. If all locations of an automaton are stable, we say that the automaton has *stable locations*. □

Unfortunately, stable locations are not sufficient to make an automaton stable, see Example 4.1.1. In Section 4.3 we demonstrate, how HA stability can be checked using cycle detection and gains estimation.

4.3 Estimating stability of hybrid automaton

In this section we introduce a notion of *contractive cycle* and define stability criteria based on the existence of such cycles. To check for non-contractive cycles we define a *gain automaton* and modify a regular expression constructing from finite automata algorithm to detect such cycles.

4.3.1 Contractive cycles and stability of hybrid automaton

It does not come as a surprise that there are two potential instability sources in hybrid automata. One of them are locations that can be unstable by itself (see Definition 4.1.1). However, even stable locations are not sufficient to make an automaton stable, as Example 4.1.1 shows. This second source of instability is related with cycles, one of the principal sources of problems and elements of research in automata theory.

Any cycle is a potential source of instability that can be caused by switches and locations that increase the norm of the state. To estimate contribution of switches and locations to the instability we use gains.

Definition 4.3.1 (Symbolic gains). Let H be a hybrid automaton.

Let l be a location of H . Then a *gain of location* l is a ratio of the *out-bound* (a state when a location is left) and the *in-bound* (a state when a location is entered) states

$$\alpha_l^* = \frac{\|\mathbf{x}_{\text{out}}^l\|}{\|\mathbf{x}_{\text{in}}^l\|}$$

where, $\|\mathbf{x}_{\text{in}}^l\|$ and $\|\mathbf{x}_{\text{out}}^l\|$ are norms of in-bound and out-bound states, respectively. However, the norms depend on in-bound and out-bound states that are not statically defined. Therefore we define a *symbolic gain of location* α_l that is an upper bound for all reachable in-bound and out-bound states, i.e., maximal α_l^* for the location l .

Let e be a switch of H and $(e, \mathbf{x}, \mathbf{x}') \in \text{Assign}$ be a corresponding assignment. The *gain of switch* e is a ratio of the norm of the state after and before taking the switch

$$\alpha_e^* = \frac{\|\mathbf{x}'\|}{\|\mathbf{x}\|}$$

An assignment can be defined in such a way that \mathbf{x} and \mathbf{x}' are not static, therefore we define a *symbolic gain of switch* α_e that is an upper bound for all reachable \mathbf{x} and \mathbf{x}' . \square

Further on we will use only symbolic gains of locations and switches, therefore we will refer to *symbolic gains* just as *gains*, where it is clear from the context. Note that in this definition we do not provide any procedure to estimate gains. The reason is that it can be done in many different ways that best fit a particular setting. However, in Section 4.4 we do propose one of such techniques.

If the symbolic gain of each location and switch that can be visited more than once is less or equal to 1 and all locations are stable, then it is clear that hybrid automaton is stable, because in a trace the number of locations and switches that are visited only once is bounded, and the switches and locations that are visited more than once do not destabilise the trace.

However, the above presented conditions are too restrictive, because the lower gains can compensate for the higher gains, and thereby preserve stability. It is natural that criteria for stability are based on cycles, because in an automaton with a finite number of locations, locations are visited several times only if there are cycles in the automaton. Therefore, we proceed with defining a particular type of cycles.

Definition 4.3.2 (Contractive cycle). Let H be a hybrid automaton, then a *contractive cycle* of H is a sequence of transitions e_1, e_2, \dots, e_m such that e_i is a transition from a location l_i to a location l_{i+1} , and $\alpha_{e_1} \cdot \alpha_{e_1 e_2} \cdot \alpha_{e_2} \cdot \alpha_{e_2 e_3} \cdot \dots \cdot \alpha_{e_m} \cdot \alpha_{e_m e_1} \leq 1$. \square

Theorem 4.3.3. *Let H be a hybrid automaton with stable locations. Then H is unstable, if it has a non-contractive cycle.*

The proof of Theorem 4.3.3 is available in Appendix A.1.

This theorem provides a sufficient condition for stability, namely the absence of non-contractive cycles.

4.3.2 Gain automata and algorithm

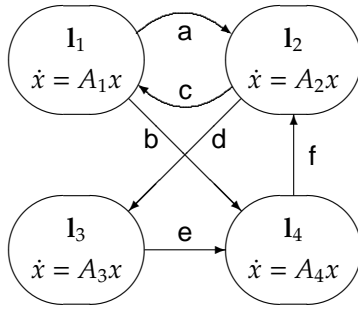


Figure 4.4: Example of hybrid automaton

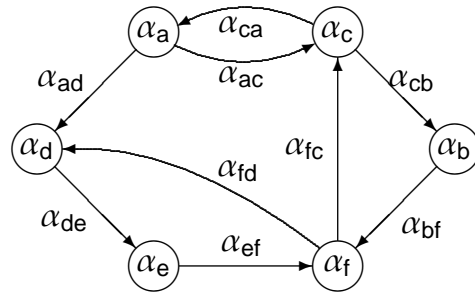


Figure 4.5: Example of gain automaton

One location can have different symbolic gains depending on incoming and outgoing transitions (guards on transitions). Therefore, for non-contractive cycle calculations we transform a hybrid automaton into another type of automaton.

Definition 4.3.4 (Gain automaton). A *gain automaton* is a collection $GA = (S, S^0, G, g)$ where

- S is the set of *vertices*.
- S^0 is the set of *initial vertices*.
- $G \subseteq S \times \mathbb{R}_+ \times S$ is the set of *edges labelled with gains*.
- $g : S \rightarrow \mathbb{R}_+$ is the labelling function that assigns gains to vertices.

□

We construct gain automata from hybrid automata in a following way.

Definition 4.3.5 (Gain automaton of hybrid automaton). Let H be a hybrid automaton. Then the gain automaton for H is defined by $GA_H = (S_H, S_H^0, G_H, g_H)$ where:

- The vertices of the gain automaton are the transitions of H , i.e., $S_H = E$.
- The initial vertices of the gain automaton are the transitions from the initial locations of H .

4. STABILITY ANALYSIS FOR HYBRID AUTOMATA

- For every location l and for each pair of transitions $e \rightarrow l \xrightarrow{e'}$ in H there is an edge $e \xrightarrow{\alpha_{ee'}} e'$ in G_H , where $\alpha_{ee'}$ is a symbolic gain of location l with in-bound transition e and out-bound transition e' .
- The labelling function g_H assigns a corresponding symbolic gain of switch α_e to every vertex of the gain automaton.

□

Example 4.3.6 (Gain automaton). Let H be a hybrid automaton from Figure 4.4. Then the corresponding gain automaton $GA_H = (S_H, S_H^0, G_H, g_H)$, constructed from H , is depicted in Figure 4.5. □

We define an algorithm for the detection of non-contractive cycles that operates on the gain automaton. If such cycles are detected, then the corresponding hybrid automaton with stable locations and continuous switching is unstable. The algorithm is inspired by the well-known algorithm for transforming a finite automaton into an equivalent regular expression [Floyd and Beigel, 1994, p.232–247], [Hopcroft et al., 2001, p.91–101], [Linz, 2001, p.71–98]. The vertices of gain automaton are deleted successively and the edges transformed. The main steps of the algorithm are the following:

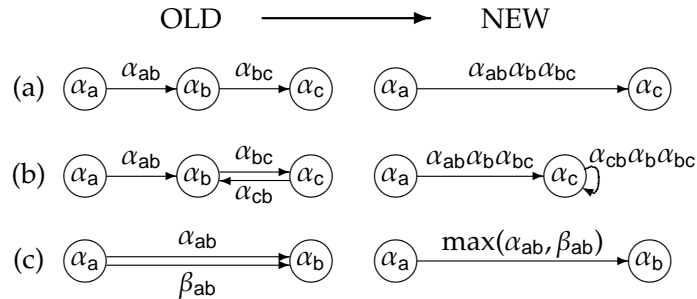


Figure 4.6: Basic Steps of the Algorithm

Vertex elimination A vertex is eliminated. Each possible pair of an incoming and outgoing edge of this vertex leads to a new edge, labelled with the product of the symbolic gains, as depicted in Figure 4.6(a).

Double edge elimination If two edges have the same initial and final vertex they are transformed into a single edge, labelled with the maximum of the symbolic gains, as depicted in Figure 4.6(c)

Loop edge analysis It is possible that deleting a vertex creates a loop edge, as illustrated in Figure 4.6(b). If the symbolic gain of such loop edge multiplied by the symbolic gain of the location is > 1 the algorithm is terminated. Otherwise, the loop edge is removed.

Based on these steps we get Algorithm 4.1.

```

algorithm DetectStability( gain automata )
begin
  while there is more than one state
    forall loop edges
      if a non-contractive loop is found
        then return non-contractive loop detected;
      end
    end
    eliminate a state;
    eliminate all resulting double edges;
  end
  forall loop edges /*for the last state*/
    if a non-contractive loop is found
      then return non-contractive loop detected;
    end
  end
end

```

Algorithm 4.1: Detection of hybrid automaton stability

Theorem 4.3.7. *Let H be a hybrid automaton with stable locations, then Algorithm 4.1 detects a non-contractive loop in GA_H iff H contains a non-contractive cycle and inverse.*

Proof. \Rightarrow : Trivial since each non-contractive loop corresponds to a non-contractive cycle.

\Leftarrow : Suppose there is a non-contractive cycle in H . Then in the gain automaton there is a cycle, where the product of the symbolic gain labels is bigger than 1. Let's call such cycle a non-contractive gain cycle.

Each removal step in the algorithm preserves the existence of a non-contractive gain cycle (see vertice elimination and double edge elimination steps). It holds for loop edge elimination, since if a non-contractive gain cycle contains a contractive loop, then the gain cycle without the contractive loop remains non-contractive.

At every step of the algorithm a non-contractive gain cycle remains present. Suppose there is a non-contractive gain cycle. If it were not detected by the algorithm, then the algorithm would produce a gain automaton with a single state and no transitions, so not containing a non-contractive gain cycle. It contradicts the fact that at every step the presence of a non-contractive gain cycle is preserved. Therefore if H has a non-contractive cycle it will be detected by the algorithm. \square

Complexity The number of vertices in GA_H is equal to the number of transitions of H . The number of edges in GA_H is at most quadratic in the number of transitions of H , but usually it is substantially lower. The complexity of Algorithm 4.1 is linear in the number of GA_H vertices. Therefore the complexity of Algorithm 4.1 in worst case is quadratic in the number of H locations.

4.3.3 Stability of two-dimensional linear continuous hyperplane hybrid automaton

The proposed stability analysis procedure consists of two steps, i.e., estimating conservative gains (Section 4.4) and detecting non-contractive cycles (Section 4.3.2). The estimation of gains will be defined for a specific type of hybrid automaton

Definition 4.3.8 (Linear continuous hyperplane HA). Let H be a hybrid automaton. Then we call it *Linear Continuous Hyperplane Hybrid Automaton* (LCHHA), if it conforms to the following requirements.

- $Init = L' \times \mathbb{R}^n$ for $L' \subseteq L$, i.e., we can start in any state of a given initial location.
- Inv maps each location to the condition `true`, i.e., there are no invariants. Therefore transitions are never forced and it is allowed to stay in a location forever. This item is not a restriction, but just a technical convenience.
- Dynamics in each location are linear, i.e., $\frac{d}{dt}\mathbf{x} = f_l(\mathbf{x}) = A_l\mathbf{x}$, $A_l \in \mathbb{R}^{n \times n}$.
- The guards are hyperplanes defined by equations of the form $\mathbf{v}_e^T \mathbf{x} = 0$ for some $\mathbf{v}_e \in \mathbb{R}^n$.
- $Assign$ does not alter the state, i.e., it is a hybrid automaton with continuous switching. This condition reduces all symbolic gains of switches to 1. As a result, they are omitted from further analysis.

□

Further on we restrict LCHHA to the two-dimensional continuous state space ($X \subseteq \mathbb{R}^2$) and get a *Two-dimensional Linear Continuous Hyperplane Hybrid Automaton* (TLCHHA). This restriction is related to the special properties of the *gains* calculation, which are discussed in Section 4.4.

Remark 4.3.9. Linearity and hyperplanes restrictions enable a certain type of conservative gains estimation procedure that is discussed in Section 4.4. □

Notice that Zeno behaviour is not possible if the guards on the incoming and outgoing switches are different. The system is required to progress from one hyperplane to the other before switching. The amount of time that this takes is invariant under scaling and is completely determined by the dynamics in the location and the angle between the hyperplanes. Since the number of locations and the number of transitions enabling guards is finite it follows that there exists a minimal dwell time.

4.4 Conservative estimation of gains

In this section we discuss conservative estimates of location's gains via Lyapunov functions.

4.4.1 Gains

Calculation of gains is not trivial. Here we show how to estimate gains of locations of LCHHA (Definition 4.3.8). We propose to use the worst case scenario, i.e., an upper-bound that depends only on the particular location and corresponding incoming and outgoing transitions.

In estimation of *conservative gains* we will take the advantage of the existence of a *Lyapunov function* in each location. *Lyapunov functions* are real-valued functions of the system's state that are monotonically non-increasing.

Definition 4.4.1 (Lyapunov function). Let's consider $\dot{x} = f(x)$ where $x \in \mathbb{R}^n$ is a state vector and $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$. Then a *Lyapunov function* is defined as a scalar function $V : \mathbb{R}^n \rightarrow \mathbb{R}$ that has the following properties.

- $V(0) = 0$;
- It is *positive definite*, i.e., $\forall \|x\| \neq 0 \Rightarrow V(x) > 0$;
- It is *continuous* and *differentiable*;
- Its derivative along every solutions of $\dot{x} = f(x)$ is *non-positive*, so, it is *non-increasing*.

□

Usually the following technique is used to construct Lyapunov functions for linear systems ($\dot{x} = Ax$, $x \in \mathbb{R}^n$). $V(x) = x^T Px$ is taken as a Lyapunov function candidate with P a *symmetric definite positive matrix* ($P > 0$ and $x^T Px > 0$ if $x \neq 0$). Then

$$\dot{V}(x) = \dot{x}^T Px + x^T P \dot{x} = \dot{x}^T A^T Px + x^T P A \dot{x} = \dot{x}^T (A^T P + P A) \dot{x}$$

and $V(x)$ is a Lyapunov function iff exists P such that $A^T P + P A \leq 0$. See Polderman and Willems [1998] for further details.

Definition 4.4.2 (Conservative gain). Let H be a TLCHHA with stable locations. With each location l we associate a symmetric positive definite matrix P_l such that $A_l^T P_l + P_l A_l \leq 0$. Let e_{in} represent a transition to l and e_{out} a transition from l and let L_{in} , given by $v_{in}^T x = 0$, and L_{out} , given by $v_{out}^T x = 0$, denote the corresponding switching hyperplanes. Define ellipsoids $E_{in} = \{x \in L_{in} \mid x^T P_l x = 1\}$ and $E_{out} = \{x \in L_{out} \mid x^T P_l x = 1\}$. The corresponding *gain* $\alpha_{in/out}$ is defined as

$$\alpha_{in/out} = \max_{\substack{x_i \in E_{in}, \\ x_o \in E_{out}}} \frac{x_o^T x_o}{x_i^T x_i}.$$

Obviously, since $V(x) = x^T P_l x$ is a Lyapunov function for $\frac{d}{dt} x = A_l x$ we have that any trajectory that enters the location through L_{in} and leaves through L_{out} has the ratio of the norms with upper bound $\sqrt{\alpha_{in/out}}$. Moreover, we use the square of the Euclidean norm, because it does not change the ratio, however it is more convenient technically. □

Now we have to sort out how to calculate a gain and how to choose P_l such that the gain is minimal.

4.4.2 Calculation of gains

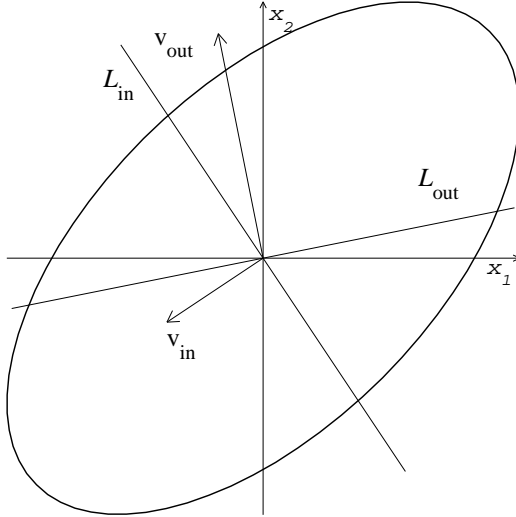


Figure 4.7: The situation in a location

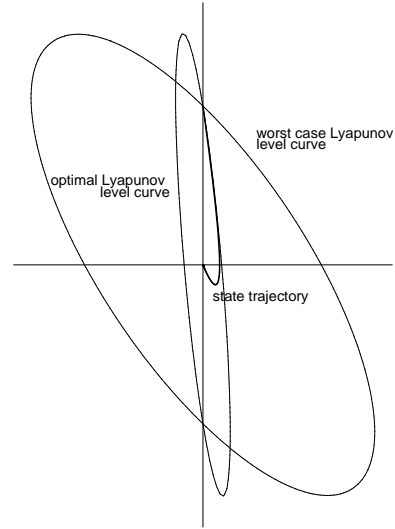


Figure 4.8: Level curves of the Lyapunov functions

When P_l is given, the calculation of gain in two dimensions is rather simple. The situation in a location is depicted in Figure 4.7. The switching lines are given by $\tilde{v}_{in}^T x = 0$ and $\tilde{v}_{out}^T x = 0$. Let \tilde{v}_{in} and \tilde{v}_{out} be orthogonal to v_{in} and v_{out} respectively. Then we get

$$\alpha_{in/out} = \frac{\tilde{v}_{out}^T \tilde{v}_{out}}{\tilde{v}_{in}^T \tilde{v}_{in}} \frac{\tilde{v}_{in}^T P_l \tilde{v}_{in}}{\tilde{v}_{out}^T P_l \tilde{v}_{out}}. \quad (4.3)$$

If \tilde{v}_{in} and \tilde{v}_{out} are on the same level curve, then Equation (4.3) reduces to

$$\alpha_{in/out} = \frac{\tilde{v}_{out}^T \tilde{v}_{out}}{\tilde{v}_{in}^T \tilde{v}_{in}}. \quad (4.4)$$

Remark 4.4.3 (Restriction to two dimensions). Restriction to two dimensions arises from the specifics of gains estimation in the higher dimensions. For example, if $n = 3$, then incoming and outgoing hyperplanes are two-dimensional planes passing through the origin and they cut the ellipsoid into ellipses, which intersect at the hyperplanes intersection line. Since maximisation of the outgoing state and minimisation of the incoming state come into play, we get that the gain is always ≥ 1 , because the incoming state is less or equal to the state of intersection and the outgoing state is more or equal to the state of intersection. Therefore the ratio of the outgoing and incoming states is always more or equal to 1. Figure 4.9 gives some insight on the situation. \square

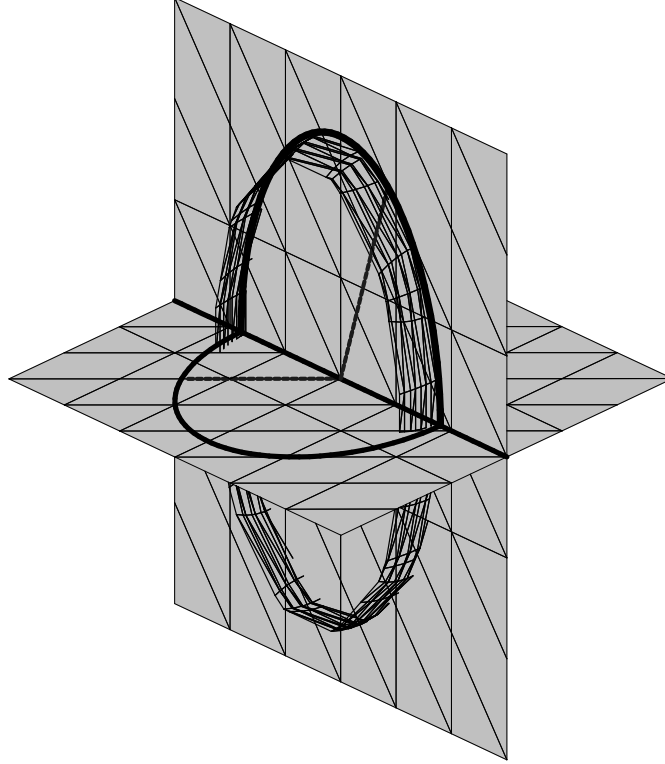


Figure 4.9: Gain estimation in three dimensions

4.4.3 Optimising the Lyapunov function choice

Stability indication provided by gains depends on the chosen Lyapunov functions in locations. These functions can fit trajectories better or worse. The better it fits the trajectory, the less conservative is the gain. Because Lyapunov functions are not unique, procedures to choose the better ones should be provided. We present a procedure for linear dynamics given by a stable matrix and quadratic Lyapunov functions.

We demonstrate the need for optimisation and the difference between loose and tight level curves in an example and Figure 4.8. Some theoretical results on the optimisation of Lyapunov functions choice are provided in Appendix A.2.

Example 4.4.4. This example is taken from Langerak et al. [2003a]. Let the dynamics in a location be given by $\frac{d}{dt}x = Ax$, where

$$A = \begin{bmatrix} 0 & 1 \\ -2 & -3 \end{bmatrix}.$$

The switching lines are given by $L_{\text{in}} = \lambda a_{\text{in}} = \lambda[0 \ 1]^T$, $L_{\text{out}} = \lambda a_{\text{out}} = \lambda[1 \ 0]^T$. Then, for a given Lyapunov function $V(x) = x^T P x$ the gain is

$$\alpha_P = \frac{a_{\text{out}}^T a_{\text{out}}}{a_{\text{in}}^T a_{\text{in}}} \frac{a_{\text{in}}^T P a_{\text{in}}}{a_{\text{out}}^T P a_{\text{out}}}$$

Let p_{22} and p_{11} are numerator and denominator of the resulting fraction, respectively. Then to find the optimal Lyapunov function we minimise α_p over the set of level curves (see Definition 4.4.2). In this example it amounts to the minimisation of $\frac{1}{p_{11}}$ or, equivalently, maximisation of p_{11} . Existence of the optimum is guaranteed by Theorem A.2.3. Extreme Lyapunov functions, minimising and maximising α_p respectively, are

$$P_{\min} = \begin{bmatrix} 12.7 & 2.59 \\ 2.59 & 1 \end{bmatrix} \quad P_{\max} = \begin{bmatrix} 0.32 & 0.41 \\ 0.41 & 1 \end{bmatrix}$$

and the corresponding minimum and maximum values of the gains are

$$\alpha_{\min} \approx 0.38 \quad \alpha_{\max} \approx 2.61.$$

Level curves and the phase portrait of $\frac{d}{dt}x = Ax$ are depicted in Figure 4.8. \square

4.5 Conclusions

In this chapter we presented a hybrid automaton stability estimation procedure that was produced as a result of cooperation between computer scientists and control theorists. We present an elegant and relatively simple technique that not only provides an algorithm for stability estimation for a certain class of automata, but also illustrates the reuse of well known efficient techniques from both areas.

The results can be generalised to include invariants, because it was introduced as a technical convenience. Potentially, the results can be extended for the higher dimensions number. However, in the meanwhile we do not know or anticipate such procedure.

Furthermore, we believe that the results can inspire some analogue research, where well known properties of automata are exploited.

*An old man set out to go into the woods, although
he didn't know what for. Then he came back and
said:
– Hey, old woman, you!
The old woman fell straight down. Since then, the
hares are white in winter.*

Daniil Kharms

5

Behavioural Hybrid Process Calculus

5.1 Introduction

The growing interest in hybrid systems both in computer science and control theory has generated a new interest in models and formalisms that can be used to specify and analyse such systems. A prominent framework for hybrid systems is provided by the family of hybrid automata models (hybrid automata (Section 3.3.6), hybrid behavioural automata (Section 3.3.7) and hybrid input/output automata (Section 3.3.8)). More recently process algebraic models have been put forward as a vehicle for the study of hybrid systems (Section 3.3.9).

Process algebra [Milner, 1989, Hoare, 1985, Bergstra and Klop, 1984, Bolognesi and Brinksma, 1987] is a theoretical framework for the modelling and analysis of the behaviour of concurrent discrete event systems that has been developed within computer science in the past quarter century. It has generated a deeper understanding of the nature of concepts such as observable behaviour in the presence of non-determinism, system composition by interconnection of concurrent system components, and notions of behavioural equivalence of such systems. It has contributed fundamental concepts such as bisimulation, and has been successfully used in a wide range of problems and practical applications in concurrent systems. We have illustrated the basic ingredients of process algebra in Section 3.3.9.

We believe that the basic tenets of process algebra are highly compatible with the behavioural approach to dynamical systems [Polderman and Willems, 1998] (Section 5.2). In this chapter we present an extension of classical process algebra that is suitable for the modelling and analysis of continuous and hybrid dynamical systems that can be seen as a generalisation of the behavioural approach in a hybrid setting. It provides a natural framework for the concurrent composition of such systems, and can deal with non-deterministic behaviour that may arise from the occurrence of internal switching events. Standard process algebraic techniques lead to the characterisation of

the observable behaviour of such systems as equivalence classes under some suitably adapted notion of bisimulation, yielding a potentially interesting mathematical interpretation of the notion of hybrid behaviour. A technical advantage of our approach is that, in contrast to Cuijpers and Reniers [2003] and Bergstra and Middelburg [2005] strong bisimulation is a congruence relation with respect to the parallel composition of subsystems¹, i.e., substitution of a subsystem by a bisimilar subsystem does not affect the behaviour of the composition.

We propose a process algebraic calculus that extends the standard repertoire of operators that combine discrete functional behaviour with features to also represent and compose continuous-time behaviour.

As mentioned above, we are inspired by the so-called behavioural approach to dynamic systems due to Polderman and Willems [1998]. In control theory, which is the relevant context in our case, the traditional presentation of dynamic behaviour, assumes a number of continuous-time input and output variables, whose evolution, respectively, influences and depends on the evolution of state variables. This evolution is typically defined in terms of differential equations.

Although in practice most dynamical systems are ultimately described in this format, Willems' behavioural approach starts from a more general point of view. System behaviour is characterised by a time-dependent relation between the observable or *manifest* variables of a system. Input and output become derived notions that depend on the constraints that the overall relation imposes on the individual variables. Thus behaviour can be simply seen as the set of all allowed real-time evolutions, or *trajectories*, of the system variables. We provide some technical details of the behavioural approach in Section 5.2.

The notion of input and output as derived concepts is also well-known in process algebras with communication based on (symmetric) instantaneous synchronisation. It suggests that communication on continuous-time variables can be achieved by non-instantaneous synchronisation on (parts of) trajectories. This leads to a calculus of actions, for discrete behaviour, and trajectories, for continuous-time behaviour. As we will see, the calculus does not depend upon any particular representation of sets of allowed trajectories: it simply defines the behaviour of composed, hierarchical systems in terms of the allowed actions and trajectories of its component systems. This leads to a natural separation of concerns in which control theory is used to determine qualitative properties of dynamical behaviour (e.g., stability, controllability, etc.), and the proposed calculus describes how these propagate under complex system compositions.

Based on the above approach, this chapter introduces the concept of hybrid transition systems and defines the related notion of strong (hybrid) bisimulation that captures a natural notion of equivalent behaviour. This leads to a branching-time interpretation of hybrid behaviour, in which behaviour is not characterised by sets of trajectories and action traces, but by tree-like structures that capture also the moments in time when a choice between alternative behaviours exists.

Subsequently, a basic language for the construction of hybrid transition systems is defined. The syntax of the language is presented and its operators are explained.

¹In Cuijpers and Reniers [2003], the robust and stateless bisimulations are however congruent.

5.2 Behavioural approach

The behavioural approach [Polderman and Willems, 1998] is a prominent paradigm in systems and control theory. In this approach the mathematical model is considered to be a subset of a universum of possibilities. It restricts the world of all possible outcomes to a convenient model of reality. The subset of outcomes is called the *behaviour* of the model. Formally, the mathematical model is defined as follows [Polderman and Willems, 1998, p.3].

Definition 5.2.1 (Mathematical model). A mathematical model is a pair $(\mathbb{U}, \mathcal{B})$ with \mathbb{U} a set, called the *universum* with elements called *outcomes* and $\mathcal{B} \subseteq \mathbb{U}$, called the *behaviour*. \square

Usually models are described by equations.

Definition 5.2.2 (Behavioural equations). Let \mathbb{U} be a universum, \mathbb{E} a set, and $f_1, f_2 : \mathbb{U} \rightarrow \mathbb{E}$. The mathematical model $(\mathbb{U}, \mathcal{B})$ with $\mathcal{B} = \{u \in \mathbb{U} \mid f_1(u) = f_2(u)\}$ is said to be described by *behavioural equations* and is denoted by $(\mathbb{U}, \mathbb{E}, f_1, f_2)$. The set \mathbb{E} is called the *equation space*. We also call $(\mathbb{U}, \mathbb{E}, f_1, f_2)$ a *behavioural representation* of $(\mathbb{U}, \mathcal{B})$. \square

In this context dynamical systems are defined as follows [Polderman and Willems, 1998, p.8].

Definition 5.2.3 (Dynamical system). A dynamical system Σ is defined as a triple $\Sigma = (\mathbb{T}, \mathbb{W}, \mathcal{B})$ with $\mathbb{T} \subseteq \mathbb{R}$, called the *time axis*, \mathbb{W} a set called the *signal space*, and $\mathcal{B} \subseteq \mathbb{T} \rightarrow \mathbb{W}$ called the *behaviour*. \square

The set \mathbb{T} specifies the set of time instances relevant to a problem, therefore it can be \mathbb{R} for continuous-time systems and \mathbb{Z} for discrete-time systems.

A behaviour \mathcal{B} is a family of time trajectories taking their values in the signal space, the collection of all possible trajectories

$$\mathcal{B} = \{w : \mathbb{T} \rightarrow \mathbb{W} \mid w \text{ is compatible with the laws that govern the system } \Sigma\}.$$

where laws are usually expressed by the behavioural expressions.

Usually, in modelling additional variables are used. In behavioural approach they are called *latent variables*. The dynamical systems model (Definition 5.2.3) is easily extended with latent variables [Polderman and Willems, 1998, p.10].

Definition 5.2.4 (Dynamical system with latent variables). A *dynamical system with latent variables* is defined as $\Sigma_L = (\mathbb{T}, \mathbb{W}, \mathbb{L}, \mathcal{B}_f)$ with $\mathbb{T} \subseteq \mathbb{R}$ the *time-axis*, \mathbb{W} the (*manifest*) *signal space*, \mathbb{L} the *latent variable space*, and $\mathcal{B}_f \subseteq (\mathbb{W} \times \mathbb{L})^{\mathbb{T}}$ the *full behaviour*. It defines a *latent variable representation* of the *manifest dynamical system* $\Sigma = (\mathbb{T}, \mathbb{W}, \mathcal{B})$ with (*manifest*) *behaviour* $\mathcal{B} = \{w : \mathbb{T} \rightarrow \mathbb{W} \mid \exists l : \mathbb{T} \rightarrow \mathbb{L} \text{ such that } (w, l) \in \mathcal{B}_f\}$. \square

The manifest variables can be thought of as *external* or *observable*. In contrast, the latent variables are *implicit*, observable only through the manifest variables.

Latent variables are frequently used in modelling, e.g., to represent internal currents and voltages in electrical circuits to express ports behaviour; as state variables in control theory to express the memory; as the basic probabilistic space in probability theory and many more.

The behavioural approach also provides techniques for building larger systems from smaller ones. In Polderman and Willems [1998] it is used to build the system from the plant and the controller, but we believe that the technique is general enough to be used for building bigger systems from the components, i.e., to build the plant and the controller itself from the smaller components.

Interconnection of two dynamical systems is defined as follows.

Definition 5.2.5 (Interconnection). Let $\Sigma_1 = (\mathbb{T}, \mathbb{W}, \mathcal{B}_1)$ and $\Sigma_2 = (\mathbb{T}, \mathbb{W}, \mathcal{B}_2)$ be two dynamical systems with the same time-axis and the same signal space. The *interconnection* of Σ_1 and Σ_2 denoted by $\Sigma_1 \wedge \Sigma_2$, is defined as

$$\Sigma_1 \wedge \Sigma_2 = (\mathbb{T}, \mathbb{W}, \mathcal{B}_1 \cap \mathcal{B}_2).$$

□

The behaviour of the interconnection consists simply of those trajectories $w : \mathbb{T} \rightarrow \mathbb{W}$ that are compatible with the laws of both Σ_1 (i.e., w belongs to \mathcal{B}_1) and of Σ_2 (i.e., w belongs also to \mathcal{B}_2). Thus in the interconnected system, the trajectories that can be generated must be acceptable to both Σ_1 and Σ_2 .

Combining it with extensions from Julius [2005] it is possible to construct complex systems from the components, or, as it is explained in Polderman and Willems [1998, p.366–368] and Julius [2005, p.71–122], to use it as a control in the behavioural context.

More results and details on the behavioural theory and its application to hybrid systems is available in Polderman and Willems [1998], Julius [2005].

5.3 Trajectories

Several different notions of trajectories are available in the literature. Lynch et al. [2003] introduces general *trajectories* and related operations (e.g., concatenation, restriction). Lygeros et al. [1999] introduces *hybrid time trajectories* and *executions* to define hybrid evolution. We define different version of trajectories and related operation that comply better with the behavioural approach [Polderman and Willems, 1998]. Moreover, we define operations (some of them just for technical convenience) for parallel composition of trajectories.

We assume that trajectories are defined over time intervals $(0, t]$ (where t can be ∞) and map to a *signal space* to define the evolution of the system. Components of the signal space correspond to the different aspects of the continuous-time behaviour, like temperature, pressure, etc. They are associated with *trajectory qualifiers* that identify them.

Definition 5.3.1 (Signal space). Let \mathcal{W} be a set of *signal domains* (typically $\subseteq \mathbb{R}$) and \mathcal{T} be a set of *trajectory qualifiers*. A *signal space* is a pair

$$\mathbb{W} = (W_1 \times \cdots \times W_n, (q_1, \dots, q_n))$$

with $W_i \in \mathcal{W}, q_i \in \mathcal{T}$, where q_i denotes the trajectory qualifier of W_i , and $q_i \neq q_j$ for $i \neq j$, i.e., all W_i have different trajectory qualifiers. □

Example 5.3.2 (Signal space). A bouncing ball is a simple example of a hybrid system (Example 2.2.1). The altitude of the ball is h , v is the vertical speed, and c is a coefficient for the energy loss.

To define dynamical behaviour of ball we can use the following signal space:

$$\mathbb{W}_{\text{BB}} = (\mathbb{R}_+ \times \mathbb{R}, (\textit{Altitude}, \textit{Velocity}))$$

where qualifiers *Altitude* and *Velocity* refer to the altitude of ball (in \mathbb{R}_+) and the vertical speed (in \mathbb{R}), respectively. \square

Usually trajectories are defined over infinite time intervals. However, hybrid systems often evolve according to some trajectory only for a certain period of time. The restriction to interval $(0, t]$ allows to define such evolutions. There are two reasons for the choice of such type of intervals. It is convenient technically. Moreover, it most accurately reflects the reality. An evolution starts at a certain time with a certain state, and the first change occurs in the left limit. Consequently, the evolution stops at a certain time moment with a certain state.

Definition 5.3.3 (Trajectory). Let $\mathbb{W} = (W_1 \times \cdots \times W_n, (q_1, \dots, q_n))$ be a signal space. Then a *trajectory* in signal space \mathbb{W} is a function

$$\varphi_{\mathbb{W}} : (0, t] \rightarrow W_1 \times \cdots \times W_n \quad (5.1)$$

where $t \in \mathbb{R}_+$ is the duration of the trajectory, also denoted as $t(\varphi)$. We will omit subscript \mathbb{W} when a signal space is clear from the context. \square

Furthermore, we will allow infinite trajectories, but with certain limitations.

We extend the definition of trajectories with an *empty trajectory*.

Definition 5.3.4 (Empty trajectory). We will use ϵ to denote an *empty* (or a *completed* trajectory). \square

We will introduce more properties of the empty trajectory later in this prefix, e.g., it is an a neutral element of concatenation (Definition 5.3.11).

Definition 5.3.5 (Set of trajectory qualifiers). A function $T : \Phi \rightarrow \mathcal{T}$, where Φ is a set of any trajectories and \mathcal{T} is a set of qualifiers, collects all trajectory qualifiers of the trajectory:

$$\begin{aligned} T(\varphi_{\mathbb{W}}) = & \\ & \{q \mid \varphi_{\mathbb{W}} : (0, u] \rightarrow W_1 \times \cdots \times W_n \\ & \wedge \mathbb{W} = (W_1 \times \cdots \times W_n, (q_1, \dots, q_n)) \wedge q \in \{q_1, \dots, q_n\}\}. \end{aligned}$$

We will define type of ϵ in such a way that $\forall \varphi \neq \epsilon \ T(\varphi) = T(\epsilon)$. \square

Remark 5.3.6 (Type of empty trajectory). We want empty trajectory to comply with every type of trajectory for the technical convenience. Therefore in the last sentence of Definition 5.3.5 we add such requirement. However, a different choice can be made, where empty trajectory has type that is different from all other types. \square

Notation 5.3.7. We will use Φ to denote a set of trajectories. Usually, we will require that all trajectories (except the empty trajectory) in the set evolve in the same signal space (have the same trajectory qualifiers, i.e., formally $\forall \varphi, \psi \neq \epsilon \in \Phi \ T(\varphi) = T(\psi)$). We will mention explicitly, if it is not the case. We will allow infinite trajectories in Φ .

For brevity reasons instead of writing $\varphi \upharpoonright (0, t]$ we will write $\varphi \upharpoonright t$, where φ is a trajectory and $\upharpoonright t$ is a restriction of a function (sometimes denoted $|_t$). \square

Definition 5.3.8 (Projection). Let $\varphi : (0, u] \rightarrow W_1 \times \dots \times W_n$ be a trajectory, such that $\mathbb{W} = (W_1 \times \dots \times W_n, (q_1, \dots, q_n))$. Then a *projection* of the trajectory w.r.t. a trajectory qualifier q_i ($i = 1, \dots, n$) is the trajectory

$$\pi^{q_i}(\varphi) : (0, u] \rightarrow W_i$$

in signal space $\mathbb{W}_i = (W_i, q_i)$. \square

Remark 5.3.9 (Extended projections). Let

$$\varphi : (0, u] \rightarrow W_1 \times \dots \times W_n$$

be a trajectory in $\mathbb{W} = (W_1 \times \dots \times W_n, (q_1, \dots, q_n))$ and let $\mathcal{T}' \subseteq T(\varphi)$. Then we extend the projection for a set of trajectory qualifiers

$$\pi^{\mathcal{T}'}(\varphi) : (0, u] \rightarrow W'_1 \times \dots \times W'_m$$

such that $\mathbb{W}_{\mathcal{T}'} = (W'_1 \times \dots \times W'_m, (q'_1, \dots, q'_m))$, $\{q'_1, \dots, q'_m\} = \mathcal{T}'$ and $\forall q_i \in \mathcal{T}' \ \pi^{q_i}(\varphi) = \pi^{q_i}(\pi^{\mathcal{T}'}(\varphi))$. \square

Example 5.3.10 (Trajectories and projections). Let $\mathbb{W}_{\text{BB}} = (\mathbb{R}_+ \times \mathbb{R}, (\textit{Altitude}, \textit{Velocity}))$ be a signal space for the bouncing ball (Example 5.3.2). Then the trajectory for the bouncing ball can be defined as a mapping

$$\varphi : (0, t] \rightarrow \mathbb{R}_+ \times \mathbb{R}$$

and, e.g., given as

$$\begin{aligned} \frac{d}{dt} \pi^{\textit{Altitude}}(\varphi) &= \pi^{\textit{Velocity}}(\varphi) \\ \frac{d}{dt} \pi^{\textit{Velocity}}(\varphi) &= -g \end{aligned}$$

with initial values $\pi^{\textit{Altitude}}(\varphi)(0) = h_0$ and $\pi^{\textit{Velocity}}(\varphi)(0) = v_0$, respectively. \square

If the signal types of two trajectories coincide, they can be concatenated to one trajectory, which is not necessarily smooth.

Definition 5.3.11 (Concatenation of trajectories). Let $\varphi : (0, t] \rightarrow W_1 \times \dots \times W_n$ (where $t \neq \infty$) and $\psi : (0, u] \rightarrow W_1 \times \dots \times W_n$ be trajectories. The *concatenation* of φ and ψ is given by the trajectory

$$\phi; \psi : (0, t + u] \rightarrow W_1 \times \dots \times W_n$$

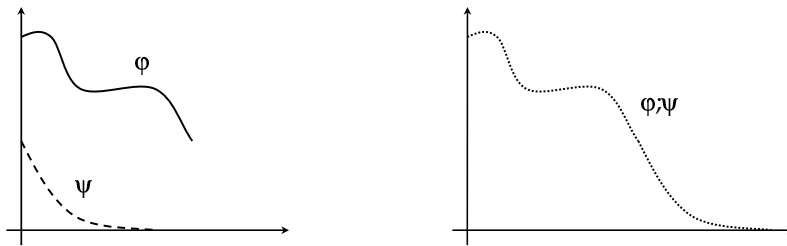


Figure 5.1: Concatenation

defined by

$$\varphi ; \psi(t') = \begin{cases} \varphi(t'), & 0 < t' \leq t \\ \psi(t' - t), & t < t' \leq t + u \end{cases}$$

Moreover, ϵ is a neutral element in the concatenation, i.e., for all $\Phi, \varphi \in \Phi \quad \epsilon ; \varphi = \varphi = \varphi ; \epsilon$. \square

Example 5.3.12 (Concatenation). Let φ and ψ be trajectories depicted by the solid and dashed lines on the left side of Figure 5.1, respectively. Then the concatenation $\varphi ; \psi$ is a trajectory depicted on the right side of Figure 5.1 by a dotted line. \square

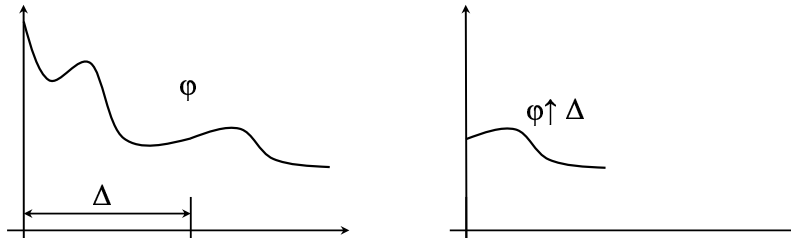


Figure 5.2: Time-shift

For the sake of convenience, a time-shift operation is defined. It displaces a trajectory to “the left” by some time. An example of the time shift by Δ time units is presented in Figure 5.2. The figure on the left side represents the original function, and the figure on the right side represents the function after the time-shift.

Definition 5.3.13 (Time-shift). Let Φ be a set of trajectories and $\varphi : (0, t] \rightarrow W_1 \times \dots \times W_n$ be a trajectory. Then the *time-shift* operator

$$\uparrow : \Phi \times \mathbb{R}_{\geq 0} \rightarrow \Phi$$

is defined for $t' < t$ as follows:

$$\varphi \uparrow t' : (0, t - t'] \rightarrow W_1 \times \dots \times W_n \text{ such that } \forall u \in (0, t - t'] \quad \varphi \uparrow t'(u) = \varphi(t' + u).$$

\square

If one trajectory coincides on the signal space with the initial part of another trajectory, it is called a *prefix* of this trajectory.

Definition 5.3.14 (Prefix of trajectory). Let $\varphi : (0, t] \rightarrow W_1 \times \dots \times W_n$ and $\psi : (0, u] \rightarrow W_1 \times \dots \times W_n$ be trajectories, such that $t \leq u$. Then φ is a *prefix* of ψ (denoted $\varphi \leq \psi$), if $\varphi = \psi \upharpoonright t$. Furthermore, if $\varphi \leq \psi$ and $t < u$, then φ is called a *strict prefix* of ψ and denoted $\varphi < \psi$. \square

We define a set of trajectories prefixes and a closure of such set.

Definition 5.3.15 (Set of trajectories prefixes). Let Φ be a set of trajectories such that $\forall \chi, \kappa \in \Phi \ T(\chi) = T(\kappa)$. Then a *set of trajectories prefixes* is defined as follows $\mathcal{P}ref^<(\Phi) = \{\varphi \mid \exists \psi \in \Phi, \varphi < \psi\} \setminus \Phi$. We will define a *set of trajectories prefixes minus empty trajectory* as $\mathcal{P}ref^+ = \mathcal{P}ref^<(\Phi) \setminus \epsilon$. \square

Definition 5.3.16 (Set of trajectories prefixes closure). Let Φ ($\forall \varphi, \psi \in \Phi \ T(\varphi) = T(\psi)$) be a set of trajectories. A set that includes all behaviours from Φ and all prefixes of the behaviours from the Φ can be defined as follows $\overline{\Phi} = \Phi \cup \mathcal{P}ref^<(\Phi)$. We will define a *set of trajectories closure minus empty trajectory and minus infinite trajectories* as follows

$$\overline{\Phi}^+ = \{\varphi \mid \exists \psi \in \Phi, \varphi < \psi, t(\varphi) \neq \infty\} \setminus \epsilon$$

\square

As a supplement to the trajectory prefix, we introduce a notion of trajectory continuation, the remainder of the taken trajectory.

Definition 5.3.17 (Trajectory continuation). Let $\varphi : (0, t] \rightarrow W_1 \times \dots \times W_n$ and $\psi : (0, u] \rightarrow W_1 \times \dots \times W_n$ be trajectories such that $\psi < \varphi$. Then we define a *trajectory continuation* of φ after taking ψ

$$\varphi \setminus \psi : (0, t - u] \rightarrow W_1 \times \dots \times W_n,$$

such that

$$\varphi \setminus \psi = \varphi \upharpoonright u.$$

\square

Trajectory continuations define the remainder of the taken trajectory. A generalised version of it, a *set of trajectory continuations*, singles out a subset of trajectory continuations, i.e., all remainders from the set of trajectories, which have the same initial part.

Definition 5.3.18 (Set of trajectory continuations). Let Φ be a set of trajectories such that $\forall \chi, \kappa \in \Phi \ T(\chi) = T(\kappa)$ and ψ be a trajectory or trajectory prefix of some trajectory belonging to the set. Then a *set of trajectory continuations* for ψ is defined as follows

$$\Phi \setminus \psi = \{\varphi \mid \psi ; \varphi \in \Phi\}$$

\square

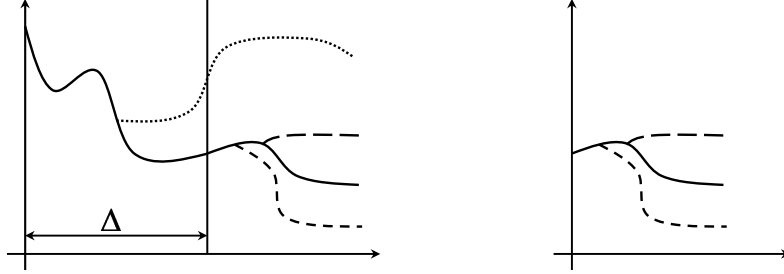


Figure 5.3: Set of continuations

Example 5.3.19 (Set of trajectory continuations). Let us have a set of trajectories depicted on the left side of Figure 5.3, consisting of 5 trajectories, that coincide until certain moment. Let us take the prefix of trajectory, depicted by the solid line, of duration Δ . Then the set of trajectory continuations for this prefix will include all continuations depicted on the right side of Figure 5.3. \square

Definition 5.3.20 (Partial prefix). Let H be a set of trajectory qualifiers, and let $\varphi : (0, t] \rightarrow W_1 \times \dots \times W_n$ in $\mathbb{W} = (W_1 \times \dots \times W_n, (q_1, \dots, q_n))$ and $\psi : (0, u] \rightarrow W'_1 \times \dots \times W'_m$ in $\mathbb{W}' = (W'_1 \times \dots \times W'_m, (q'_1, \dots, q'_m))$ be trajectories, such that $t \leq u$. Let $\mathcal{T} = \text{T}(\varphi) \cap \text{T}(\psi) \subseteq H$.

- Trajectory φ is a *partial prefix* of ψ (denoted $\varphi \leq^H \psi$), if $\pi^{\mathcal{T}}(\varphi) = \pi^{\mathcal{T}}(\psi \upharpoonright t)$.
- If $\varphi \leq^H \psi$ and $t < u$, then φ is called a *strict partial prefix* of ψ and denoted $\varphi <^H \psi$.
- In case of $t = u$ the trajectories are equal on the coinciding trajectory qualifiers and are called *partially equal* (denoted $\varphi =^H \psi$).

\square

The partial prefix relaxes requirements put by the prefix (Definition 5.3.14), i.e., only the projections over coinciding trajectory qualifiers are compared.

Example 5.3.21 (Partial prefix). Let us have two trajectories, which define the (altitude, velocity) and (altitude, temperature) pairs, respectively. Then one of these trajectories is a partial prefix of another, if the altitude evolves in the same way. It allows to define different aspects of the same object separately and then compose definitions to get a complete specification of the object. \square

Based on synchronising trajectory qualifiers, two trajectories can be composed creating a new, “wider”, trajectory, such that evolutions of coinciding trajectory qualifiers are merged and non-coinciding parts extend the state space.

Definition 5.3.22 (Composition of trajectories). Let H be a set of synchronising trajectory qualifiers, and let $\varphi : (0, t] \rightarrow W''_1 \times \dots \times W''_k$ in $\mathbb{W}'' = (W''_1 \times \dots \times W''_k, (q''_1, \dots, q''_k))$ and $\psi : (0, u] \rightarrow W'_1 \times \dots \times W'_m$ in $\mathbb{W}' = (W'_1 \times \dots \times W'_m, (q'_1, \dots, q'_m))$ be trajectories

5. BEHAVIOURAL HYBRID PROCESS CALCULUS

such that $T(\varphi) \cap T(\psi) \subseteq H$, $\pi^{T(\varphi) \cap T(\psi)}(\varphi) = \pi^{T(\varphi) \cap T(\psi)}(\psi)$ and $u \leq t$. Then a *composition of trajectories* is a trajectory

$$\varphi \times_H \psi : (0, u] \rightarrow W_1 \times \cdots \times W_n$$

in $\mathbb{W} = (W_1 \times \cdots \times W_n, (q_1, \dots, q_n))$ such that

$$\begin{aligned} T(\varphi \times_H \psi) &= T(\varphi) \cup T(\psi), \\ \pi^{T(\varphi)}(\varphi \times_H \psi) &= \varphi, \\ \pi^{T(\psi)}(\varphi \times_H \psi) &= \psi. \end{aligned}$$

□

We extend composition of trajectories to composition of sets of trajectories.

Definition 5.3.23 (Composition of sets of trajectories). Let H be a set of synchronising trajectory qualifiers, and let Φ and Ψ be sets of trajectories such that, $\forall \varphi, \psi \in \Phi$ $T(\varphi) = T(\psi)$, $\forall \varphi, \psi \in \Psi$ $T(\varphi) = T(\psi)$ and $\forall \varphi \in \Phi, \forall \psi \in \Psi$ holds $T(\varphi) \cap T(\psi) \subseteq H$. Then a *composition of sets of trajectories* is defined as follows

$$\Phi \times_H \Psi = \{\varphi \times_H \psi \mid \varphi \in \overline{\Phi} \wedge \psi \in \overline{\Psi} \wedge \varphi \stackrel{H}{=} \psi \wedge \pi^{T(\varphi) \cap T(\psi)}(\varphi) = \pi^{T(\varphi) \cap T(\psi)}(\psi)\}$$

Moreover, if $\epsilon \in \Phi \vee \epsilon \in \Psi$, then $\epsilon \in \Phi \times_H \Psi$. □

Several different ways will be used to define sets of trajectories. We will require that all the trajectories in the set have the same qualifiers, i.e., $\forall \varphi, \psi \in \Phi$ $T(\varphi) = T(\psi)$.

- By listing all trajectories belonging to the set: $\Phi = \{\varphi_1, \dots, \varphi_n\}$ such that $\forall i, j = 1, \dots, n$ $T(\varphi_i) = T(\varphi_j)$.
- By putting restrictions on the already existing set of trajectories: $\Phi \downarrow \mathcal{P}red = \{\varphi \in \Phi \mid \mathcal{P}red(\varphi)\}$, where $\mathcal{P}red$ is a predicate.
- Sometimes it is useful to define conditions on the *end-points* of trajectories or the *exit conditions*. We will use \Downarrow to denote such conditions, as the restrictions on set of trajectories: $\Phi \Downarrow \mathcal{P}red_{\text{exit}} = \{\varphi : (0, u] \rightarrow W_1 \times \cdots \times W_n \in \Phi \mid \mathcal{P}red_{\text{exit}}(\varphi(u))\}$. It is illustrated in Section 5.8, where, e.g., in Example 5.8.1 $h = 0$ specifically requires that the trajectory finishes at 0 altitude.

Moreover, an empty trajectory ϵ will denote an instantaneous exit availability.

When it is clear from the context, we will use trajectory qualifiers to access corresponding parts of trajectories, e.g., q_i will mean the same as $\pi^{q_i}(\varphi)$ ($i = 1 \dots n$) for

$$\varphi : (0, t] \rightarrow W_1 \times \cdots \times W_n \text{ with } \mathbb{W} = (W_1 \times \cdots \times W_n, (q_1, \dots, q_n)).$$

Furthermore, will use q_i instead of $\pi^{q_i}(\varphi)(u)$ with $u \in (0, t]$ as a time, when it is clear from the context, e.g., as it is used in definition of $\mathcal{P}red_{\text{exit}}$. Moreover, the combination of different conditions is allowed

$$\Phi \downarrow \mathcal{P}red \Downarrow \mathcal{P}red_{\text{exit}} = \{\varphi : (0, u] \rightarrow \mathbb{W} \in \Phi \mid \mathcal{P}red(\varphi) \wedge \mathcal{P}red_{\text{exit}}(\varphi(u))\}$$

5.4 Hybrid transition systems

Automata, state-transition diagrams and other similar models are often used to describe the dynamic behaviour of the systems. They consist of states $s \in S$ (with S as a set of states) and some construct, defining changes of the states. Most of the time changes of the states are defined by *transitions*, which are given as a relation (function) over a subset of the Cartesian product of the states ($S \times S$). Usually transitions are denoted by an arrow, e.g., $(s, s') \in \rightarrow$ or $s \rightarrow s'$.

Labelled transition systems is a class of transition systems, where transitions are labelled with some *actions* $\mathbf{a} \in \mathcal{A}$ (where \mathcal{A} is a set of actions). The transition relation is defined over subset of $S \times \mathcal{A} \times S$. For $(s, \mathbf{a}, s') \in \rightarrow$ we write $s \xrightarrow{\mathbf{a}} s'$.

A hybrid transition system is a labelled transition system with two types of transitions.

Definition 5.4.1 (HTS). A *hybrid transition system* is a tuple $HTS = \langle S, \mathcal{A}, \rightarrow, \mathbb{W}, \Phi, \rightarrow_c \rangle$, where

- S is a *state space*;
- \mathcal{A} is a *set of (discrete) action names*;
- $\rightarrow \subseteq S \times \mathcal{A} \times S$ is a *(discrete) transition relation*;
- \mathbb{W} is a *signal space*;
- Φ is a *set of trajectories* $\varphi : (0, t] \rightarrow W_1 \times \dots \times W_n$ for $t \in \mathbb{R}_+$, and $t \neq \infty, \epsilon \notin \Phi$;
- $\rightarrow_c \subseteq S \times \Phi \times S$ is a *(continuous-time) transition relation*.

We will write

$$s \xrightarrow{\mathbf{a}} s' \iff (s, \mathbf{a}, s') \in \rightarrow$$

$$s \xrightarrow{\varphi} s' \iff (s, \varphi, s') \in \rightarrow_c.$$

The set of discrete action names includes a *silent action*, denoted τ . It does not represent a potential communication and is not directly observable. Silent action may be used to specify a non-deterministic behaviour² (as *internal actions* in Milner [1989, p.37–43]). \square

Remark 5.4.2. Constraints $\epsilon \notin \Phi$ and $\forall \varphi \in \Phi, t(\varphi) \neq \infty$ just mean that we will not have infinite or empty transitions. But we will be able to construct infinite evolutions by concatenating finite trajectories. And ϵ is used to denote completed trajectories. \square

Notation 5.4.3. We will adhere to a certain notation.

- Greek alphabet symbols (like φ, ψ) will be used to denote trajectories, which are taken on a continuous-time transition.
- Latin alphabet (like \mathbf{a}, \mathbf{b}) will be used to denote actions.

\square

²Of course, it is not the only way to model nondeterminism, e.g., nondeterministic choice operator can be defined, it can be created by parallel composition.

Definition 5.4.4 (Density). We will require *density* for all trajectories

$$\forall \varphi_1, \varphi_2 : \varphi = \varphi_1 ; \varphi_2 \wedge s \xrightarrow{\varphi} s' \iff \exists s'' : s \xrightarrow{\varphi_1} s'' \wedge s'' \xrightarrow{\varphi_2} s'.$$

□

This requirement allows us to split every trajectory into arbitrarily many parts. Moreover, in such a case the set of trajectories Φ is prefix and suffix closed.

Remark 5.4.5 (Labels of continuous-time transitions). Label φ in $s \xrightarrow{\varphi} s'$ is a *semantic object*, viz. the set theoretic graph of the function φ . □

5.4.1 Bisimulation

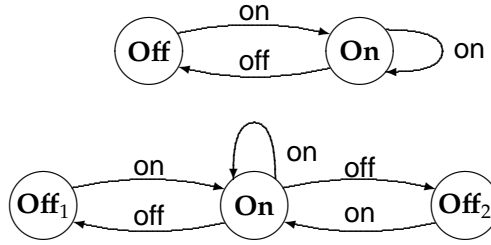


Figure 5.4: Two bisimilar automata

One of the main tools to compare systems is *strong bisimulation*. The bisimulation for continuous dynamical systems is presented in van der Schaft [2004]. The process algebraic version is nicely explained in Milner [1989]. Strong bisimulation requires both subsystems to be able to imitate each other at every step.

Example 5.4.6 (Bisimulation). We illustrate bisimulation with an example of two automatic light switches, depicted in Figure 5.4. In the figure circles represent states and arrows represent switches. Both automata represent two slightly different light switches.

The upper automaton represents a one button light switch. Let assume that the system is in the state **Off**. When the **on** is executed, the system changes to the state **On**. After a while the light goes off by executing **off**. If the light is on and the action **on** is executed, the light stays on.

The lower automaton represents a somehow complicated light switch that has two off states **Off₁** and **Off₂**, and when the light switches off, it may go to any of them. However, it exhibits the same observable behaviour, i.e., its observable reaction to actions **on** and **off** is the same.

It is easy to see that the states **On** of both automata are equivalent, and the states **Off₁** and **Off₂** of the lower automaton are equivalent to the state **Off** of the upper automata. Then we get the following bisimulation relation

$$\mathcal{R} = \{(\text{On}^u, \text{On}^l), (\text{Off}^u, \text{Off}_1^l), (\text{Off}^u, \text{Off}_2^l)\}$$

where superscripts u and l denote the upper and lower automata, respectively. □

A strong bisimulation for hybrid transition systems requires both systems to be able to execute the same trajectories and actions and to have the same branching structure.

Definition 5.4.7 (Hybrid strong bisimulation). A binary relation $\mathcal{R} \subseteq S \times S$ on the states is a *hybrid strong bisimulation*, if for all $p, q \in S$, such that $p \mathcal{R} q$, holds

$$\begin{aligned} p \xrightarrow{a} p' &\implies \exists q' \text{ such that } q \xrightarrow{a} q' \text{ and } p' \mathcal{R} q' \\ q \xrightarrow{a} q' &\implies \exists p' \text{ such that } p \xrightarrow{a} p' \text{ and } p' \mathcal{R} q' \\ p \xrightarrow{\varphi} p' &\implies \exists q' \text{ such that } q \xrightarrow{\varphi} q' \text{ and } p' \mathcal{R} q' \\ q \xrightarrow{\varphi} q' &\implies \exists p' \text{ such that } p \xrightarrow{\varphi} p' \text{ and } p' \mathcal{R} q'. \end{aligned}$$

□

The first two statements define bisimulation requirements for the discrete actions, and the last two for the continuous-time transitions.

Definition 5.4.8 (Bisimilarity). States p and q are *bisimilar* (denoted $p \sim q$), if there exists a hybrid strong bisimulation \mathcal{R} , containing the pair (p, q) . □

5.5 Language and operational semantics

5.5.1 Language

To define evolution and interaction of systems, a language and its semantics, based on hybrid transition systems are introduced. The syntax of language is presented in BNF notation (Backus-Naur form).

$$B ::= 0 \mid a.B \mid [f \mid \Phi].B \mid \sum_{i \in I} B_i \mid B \parallel_A^H B \mid \text{new } w.B \mid B[\sigma] \mid P$$

- 0 is a *deadlock*, the process that does not show any behaviour.
- $a.B$ is an *action prefix*, where $a \in \mathcal{A}$ is a discrete action name and B is a process. It first performs a and then engages in B . An action prefix denotes a discrete transition in the underlying hybrid transition system.
- $[f \mid \Phi].B(f)$ is a *trajectory prefix*, where f is a trajectory variable and Φ is a set of trajectories. It takes a trajectory or a prefix of a trajectory in Φ . If a trajectory or a part of it was taken and there exists a continuation of the trajectory, then the system can continue with a trajectory from the trajectory continuations set. If a whole trajectory was taken, then the system may continue with B , too.
- $\sum_{i \in I} B_i$ is a *choice* of processes. To generate the set we allow arbitrary index sets I . It chooses before taking an action prefix or trajectory prefix. Binary version of choice is denoted $B_1 + B_2$.

- $B \parallel_A^H B$ is a *parallel composition* of two processes with an *interconnection set* H and a *synchronisation set* A . The interconnection set $H \subseteq \mathcal{T}$ is a set of trajectory qualifiers for the synchronisation of trajectories and the synchronisation set $A \subseteq \mathcal{A}$ is the set of action names for the synchronisation of discrete transitions. Parallel composition defines a new process that executes both processes in parallel forcing trajectory prefixes and actions in A to synchronise. If actions are not in A , they are executed in the interleaving manner, i.e., sequentially in an arbitrary order. We require all shared trajectory qualifiers to be in the interconnection set H to avoid unintended synchronisation. A case when a shared qualifier is not in H , we consider to be an error.
- $\text{new } w.B$ is a *hiding* operator, where w is a set of discrete action names and trajectory qualifiers to hide.
- $B[\sigma]$ is a *renaming* operator, where σ is a renaming function. Function σ takes an action name or a trajectory qualifier and changes it. Renaming function for the actions $\sigma : \mathcal{A} \rightarrow \mathcal{A}$. For the trajectory qualifiers renaming is defined as $\sigma : \mathcal{T} \rightarrow \mathcal{T}$ and it should be injective for the trajectory qualifiers. $B[\sigma]$ behaves as B but with the actions and trajectory-qualifiers renamed according to σ .
- P is a *recursive equation*, where P is a process identifier.

We use syntactic functions $\mathcal{Q}(B)$ and $\mathcal{N}(B)$ for collecting action and trajectory qualifiers occurring in B , respectively.

We require a *consistent signal flow*, i.e., only the parallel composition is allowed to change the set of trajectory qualifiers in the process. Renaming operation only renames them, but does not change their types.

We define restrictions on processes and then show that if the conditions are satisfied then the signal flow is consistent.

Definition 5.5.1 (Trajectory qualifiers of a process). Let \mathbb{P} be a *set of processes* and \mathcal{T} be a set of trajectory qualifiers. Let ω denotes an ill-defined type. Then we define a syntactical function $\mathcal{N} : \mathbb{P} \rightarrow \mathcal{T}$ that collects trajectory qualifiers occurring in the process.

$$\begin{aligned}
 \mathcal{N}(0) &:= \emptyset \\
 \mathcal{N}(a.B) &:= \mathcal{N}(B) \\
 \mathcal{N}([f \mid \Phi].B) &:= \begin{cases} \omega, & \text{if } \mathcal{N}(B) = \omega \vee \forall \varphi \in \Phi \mathcal{N}(B) \neq \mathsf{T}(\varphi); \\ \mathsf{T}(\varphi), & \text{if } \mathcal{N}(B) = \emptyset \vee \\ & \mathcal{N}(B) \neq \emptyset \wedge \mathcal{N}(B) \neq \omega \wedge \forall \varphi \neq \emptyset \in \Phi : \mathcal{N}(B) = \mathsf{T}(\varphi); \end{cases} \\
 \mathcal{N}\left(\sum_{i \in I} B_i\right) &:= \begin{cases} \omega, & \text{if } (\exists B_i (i \in I) : B_i = \omega) \vee (\exists B_i, B_j (i, j \in I) : \\ & \mathcal{N}(B_i) \neq \emptyset \wedge \mathcal{N}(B_j) \neq \emptyset \wedge \mathcal{N}(B_i) \neq \mathcal{N}(B_j)) \\ \bigcup_{i \in I} \mathcal{N}(B_i) & \text{if } \forall i, j \in I : (\mathcal{N}(B_i) \neq \emptyset \wedge \mathcal{N}(B_j) \neq \emptyset \\ & \text{then } \mathcal{N}(B_i) = \mathcal{N}(B_j)) \end{cases}
 \end{aligned}$$

$$\begin{aligned}
 \mathcal{N}(B \parallel_A^H C) &:= \begin{cases} \omega, & \text{if } \mathcal{N}(B) = \omega \vee \mathcal{N}(C) = \omega \\ \mathcal{N}(B) \cup \mathcal{N}(C), & \text{if } \mathcal{N}(B) \neq \omega \wedge \mathcal{N}(C) \neq \omega \end{cases} \\
 \mathcal{N}(B[\sigma]) &:= \begin{cases} \omega, & \text{if } \mathcal{N}(B) = \omega \\ \sigma(\mathcal{N}(B)), & \text{if } \mathcal{N}(B) \neq \emptyset \wedge \mathcal{N}(B) \neq \omega \\ \emptyset, & \text{if } \mathcal{N}(B) = \emptyset \end{cases} \\
 \mathcal{N}(\text{new } w.B) &:= \begin{cases} \omega, & \text{if } \mathcal{N}(B) = \omega \\ \mathcal{N}(B) \setminus w, & \text{if } \mathcal{N}(B) \neq \omega \end{cases} \\
 \mathcal{N}(P) &:= T_p, \text{ where, if } P \triangleq B, \text{ then } T_p := \mathcal{N}(B)
 \end{aligned}$$

Basically, we collect all qualifiers names from process, and require that no new qualifiers appear, except in parallel composition. In hiding qualifier names are not visible to an external observer, but they are still there. In renaming only qualifiers name change, not their type. \square

Definition 5.5.2 (Consistent signal flow). We will require that all processes have a *consistent signal flow*, i.e., that for all processes $\mathcal{N}(B) \neq \omega$. \square

We formulate consistency property (Theorem 5.5.4) and give a proof after introducing SOS rules.

5.5.2 Operational semantics of BHPC

In this section we define the semantics of the BHPC operators.

Action prefix $a.B$

Process $a.B$ defines a process which executes the action a and then behaves as B .

A special *silent action*, denoted τ , is introduced. It does not represent a potential communication and is not directly observable. Silent actions may be used to specify a non-deterministic behaviour³ (as *internal actions* in Milner [1989, p.37–43]).

$$a.B \xrightarrow{a} B \quad (5.2)$$

In Section 5.7.1 we define a parametrised version of the action prefix. The use of both ordinary and parametrised action prefixes is illustrated in Section 5.8.

Trajectory prefix $[\varphi \mid \Phi].B(f)$

A *trajectory prefix* defines the behaviour that starts with a trajectory denoted by f and is followed by the trajectory continuation or behaviour specified by B .

We will use extended notation to denote parameterisation of processes, i.e., $B(f)$ will denote a process parameterised by a trajectory variable f . Furthermore, $B(\varphi; f')$ will denote substitution of such trajectory variable by $\varphi; f'$.

$$[\varphi \mid \Phi].B(f) \xrightarrow{\varphi} [\varphi' \mid \Phi \setminus \varphi].B(\varphi; f') \quad \text{for all } \varphi \in \overline{\Phi}^+ \quad (5.3)$$

³Of course, it is not the only way to model nondeterminism, e.g., nondeterministic choice operator can be defined, it can be created by parallel composition.

where Φ is a set of trajectories such that $\forall \varphi, \psi \in \Phi \ T(\varphi) = T(\psi)$. Notice that ϵ can be in Φ , because it complies with any qualifier type. f, f' are trajectory variables. Let ψ is trajectory and φ is taken from (5.3), then $\varphi ; \psi \in \Phi$ or $\varphi \in \Phi$ such that $t(\varphi) \neq \infty$ and $\varphi \neq \epsilon$. If a trajectory or a part of it was taken and there exists a continuation of the trajectory, then the system can continue with a trajectory from the trajectory continuations set (Definition 5.3.18). However, if a whole trajectory was taken, then the system may continue with the consecutive process with the substituted trajectories (see (5.4)). $(\varphi ; f')$ defines substitution of the taken trajectories in the following processes, i.e., all instances of f in B are substituted by the taken trajectory φ concatenated with its follow-up f' , or if it is finished, by the whole taken trajectory φ .

After defining concatenation (5.4) we present a derived rule that explains behaviour of the empty trajectory ϵ in trajectory prefix.

Notation 5.5.3. We will extend notation to make use of trajectory prefix more convenient

$$[q_1, \dots, q_m \mid \Phi \downarrow \text{Pred} \Downarrow \text{Pred}_{\text{exit}}]$$

where

- q_1, \dots, q_m are trajectory qualifiers, which can be used to access corresponding parts of trajectories.
- As explained in the end of Section 5.3, the set of trajectories can be defined in several different ways. We will allow such notation in the trajectory prefix definition to bring out conditions on the set of trajectories.

Furthermore, we will allow to define the set of trajectories directly in the definition of trajectory prefix, where commas will be used to separate conditions. We will use \Downarrow to separate exit conditions, when it is required. \square

Concatenation

Concatenation extends definition of trajectory prefix. It formalises behaviour after taking a complete trajectory. The process can choose to continue with another trajectory or an action prefix, depending on the successive process.

Concatenation is formalised by the following derivation rules.

$$\frac{B(\varphi) \xrightarrow{\psi} B'}{[f \mid \Phi].B(\varphi) \xrightarrow{\varphi;\psi} B'} \quad \varphi \in \Phi \quad (5.4a)$$

$$\frac{B(\epsilon) \xrightarrow{a} B'}{[f \mid \Phi].B(f) \xrightarrow{a} B'} \quad \epsilon \in \Phi \quad (5.4b)$$

In (5.4a) it is shown, how to concatenate two trajectories. While (5.4b) defines a situation, where after taking a whole trajectory process continues with an action prefix.

For a convenience we derive an equation from the concatenation and trajectory prefix rules. If $\epsilon \in \Phi$ then

$$[f \mid \Phi].B(f) \sim [f \mid \Phi \setminus \epsilon].B(f) + B(\epsilon) \quad (5.5)$$

We leave proof of these equation as an exercise for readers.

Choice $\sum\{B(v) \mid v \in I\}$

Choice is a generalised operator on sets of behaviour expressions. To generate the set we allow arbitrary index sets I . It can be thought of as a generalisation of the ordinary process algebraic choice.

$$\frac{B(w) \xrightarrow{a} B'}{\sum_{v \in I} B(v) \xrightarrow{a} B'} \quad w \in I \quad (5.6a)$$

$$\frac{B(w) \xrightarrow{\varphi} B'}{\sum_{v \in I} B(v) \xrightarrow{\varphi} B'} \quad w \in I \quad (5.6b)$$

In (5.6a) the choice for action prefixes is defined, which is the same as in usual process algebras. Rule (5.6b) tells that choice for trajectories is made before taking a trajectory.

Use of choice is illustrated in examples from Section 5.8.

Parallel composition $B_1 \parallel_A^H B_2$

Parallel composition models concurrent evolution of several processes. During the evolution they may interact with each other via synchronisation on discrete and continuous-time transitions. In BHPC synchronisation on identical names is assumed as the basic synchronisation concept. In order to avoid context-dependent interpretations of operators, the set of action names A and the set of trajectory qualifiers H that are subject to synchronisation, are made explicit in the parallel operator \parallel_A^H .

This form of synchronisation implies that parallel components jointly execute identical actions or trajectories with common signal evolutions that occur in their transitions and are subject to synchronisation.

The basic idea of synchronising trajectories is not much different than that of synchronising actions. Let B_1 and B_2 be the processes which can take trajectories

$$\varphi : (0, t] \rightarrow W'_1 \times \cdots \times W'_m \quad \text{and} \quad \psi : (0, t] \rightarrow W''_1 \times \cdots \times W''_k$$

in $\mathbb{W}' = (W'_1 \times \cdots \times W'_m, (q'_1, \dots, q'_m))$ and $\mathbb{W}'' = (W''_1 \times \cdots \times W''_k, (q''_1, \dots, q''_k))$, respectively. The static constraint is imposed that $B_1 \parallel_A^H B_2$ is only well-formed iff $\mathcal{L}(B_1) \cap \mathcal{L}(B_2) \subseteq A$ and $\mathcal{N}(B_1) \cap \mathcal{N}(B_2) \subseteq H$ (where \mathcal{L} is a syntactical function, that collects actions names from the process). Let \mathcal{W} be a set of signal domains and let

$$\mathcal{T}' = \mathcal{T}(\varphi) \cap \mathcal{T}(\psi). \quad (5.7)$$

If a set of coinciding trajectory quantifiers is a subset of the synchronisation set

$$\mathcal{T}' \subseteq H \quad (5.8a)$$

and trajectories are the same on the coinciding quantifiers

$$\pi^{\mathcal{T}'}(\varphi) = \pi^{\mathcal{T}'}(\psi), \quad (5.8b)$$

5. BEHAVIOURAL HYBRID PROCESS CALCULUS

then the resulting trajectory is a synchronised trajectory of $B_1 \parallel_A^H B_2$ that simultaneously changes the states of B_1 and B_2 , defined as

$$\chi : (0, t] \rightarrow W_1 \times \cdots \times W_n$$

in $\mathbb{W} = (W_1 \times \cdots \times W_n, (q_1, \dots, q_n))$ such that

$$\begin{aligned} \mathbb{T}(\chi) &= \mathbb{T}(\varphi) \cup \mathbb{T}(\psi), \\ \pi^{\mathcal{T}'}(\varphi) &= \pi^{\mathcal{T}'}(\psi) = \pi^{\mathcal{T}'}(\chi), \\ \varphi &= \pi^{\mathbb{T}(\varphi)}(\chi), \\ \psi &= \pi^{\mathbb{T}(\psi)}(\chi). \end{aligned}$$

It can be also defined via the composition of trajectories (Definition 5.3.22), i.e., $\chi = \varphi \times_H \psi$.

We define the following SOS rules for parallel composition

$$\frac{B_1 \xrightarrow{a} B'_1, B_2 \xrightarrow{a} B'_2}{B_1 \parallel_A^H B_2 \xrightarrow{a} B'_1 \parallel_A^H B'_2} \quad a \in A \quad (5.9a)$$

$$\frac{B_1 \xrightarrow{a} B'_1}{B_1 \parallel_A^H B_2 \xrightarrow{a} B'_1 \parallel_A^H B_2} \quad a \notin A \quad (5.9b)$$

$$\frac{B_1 \xrightarrow{\varphi} B'_1, B_2 \xrightarrow{\psi} B'_2}{B_1 \parallel_A^H B_2 \xrightarrow{\varphi \times_H \psi} B'_1 \parallel_A^H B'_2} \quad (5.7) \text{ and } (5.8) \text{ hold} \quad (5.9c)$$

Rule (5.9b) explains interleaving semantics for the discrete behaviour, when discrete actions names do not coincide. Synchronisation on actions is defined in (5.9a). Rule (5.9c) defines the parallel composition of similar trajectories. Notice that because of density (Definition 5.4.4) we do not have to give rules for the trajectories with different durations.

Parallel composition is illustrated in Examples 5.8.1 and 5.8.2.

Hiding new $w.B$

Following the conventions of the process calculus a *hiding* is introduced as a scope restriction operator. $\text{new } w.B$ restricts the use of the names w to B . Hiding for discrete actions is just an ordinary hiding. It is worth emphasising that hiding (especially in continuous-time case) should be used carefully, because two different trajectories can easily become observably equivalent, if only equivalent parts of the behaviour are

visible. Hiding may easily influence the outcome of parallel composition and choice.

$$\frac{B \xrightarrow{a} B'}{\text{new } w.B \xrightarrow{\tau} \text{new } w.B'} \quad a \in w \quad (5.10a)$$

$$\frac{B \xrightarrow{a} B'}{\text{new } w.B \xrightarrow{a} \text{new } w.B'} \quad a \notin w \quad (5.10b)$$

$$\frac{B \xrightarrow{\varphi} B'}{\text{new } w.B \xrightarrow{\pi^{\tau(\varphi)} \setminus w(\varphi)} \text{new } w.B'} \quad (5.10c)$$

The first rule states that if an action should be hidden, it is renamed to τ (silent) action. Otherwise (the second rule) nothing changes. The third rule defines hiding for the continuous-time behaviour, i.e., some qualifiers are not visible any more.

Renaming $B[\sigma]$

Renaming operator $B[\sigma]$, where σ is a *renaming function*, is defined. Renaming of both action and signal names is allowed. The renaming function σ changes only trajectory qualifiers, but not their type.

$$\frac{B \xrightarrow{a} B'}{B[\sigma] \xrightarrow{\sigma(a)} B'[\sigma]} \quad \frac{B \xrightarrow{\varphi} B'}{B[\sigma] \xrightarrow{\sigma(\varphi)} B'[\sigma]} \quad (5.11)$$

Recursion

The ordinary process algebraic recursion is extended to work with trajectory prefix. It allows to define processes in terms of each other, like in equation $P \triangleq B$, where P is a process identifier and actions and signal types of B are only allowed actions and signal types in P .

$$\frac{B \xrightarrow{a} B'}{P \xrightarrow{a} B'} \quad P \triangleq B \quad \frac{B \xrightarrow{\varphi} B'}{P \xrightarrow{\varphi} B'} \quad P \triangleq B \quad (5.12)$$

5.5.3 Consistent signal flow

Theorem 5.5.4 (Consistent signal flow). *If the constraints stated in Definition 5.5.2 are satisfied, then the signal flow is consistent, i.e., a semantic type is preserved. Then for all $B_1, B_2, B_3, \varphi, \psi$ holds.*

$$\text{if } B_1 \xrightarrow{\varphi} B_2 \xrightarrow{\psi} B_3 \text{ then } B_1 \xrightarrow{\varphi;\psi} B_3$$

The proof of the theorem is provided in Appendix B.

Corollary 5.5.5. *From the Theorem 5.5.4 it is also follows that signal type is consistent, i.e., a syntactic type is preserved:*

$$B_1 \xrightarrow{\varphi} B_2 \xrightarrow{\psi} B_3 \implies \mathcal{N}(B_1) = \mathcal{N}(B_2) = \mathcal{N}(B_3)$$

5.5.4 Congruence property

Process algebras usually employ a *congruence* as a basis for systems analysis. A *congruence* for a process algebra is an equivalence relation (i.e, reflexive, symmetric and transitive) that has the substitution property, i.e, equivalent systems can replace each other inside any larger system, without changing the behaviour of that system.

Theorem 5.5.6. *Hybrid strong bisimulation equivalence on HTSs is a congruence w.r.t. the operations of BHPC defined by the in Section 5.5.2.*

We provide a proof of the theorem in Appendix B.2. Basically, we follow the procedure from Milner [1989], with certain adaptations to our calculus (mostly, trajectories related). We analyse every syntactical context and behaviour of bisimilar processes in such a context.

5.6 Expansion law

The expansion law (Theorem 5.6.3) expresses the parallel composition as a choice of processes (where parallel composition of discrete actions is resolved in the *interleaving* manner).

It is possible to reduce any process in BHPC to a basic form. For processes that do not involve parallel composition it is trivial. Below we show how the parallel composition can be eliminated.

Remark 5.6.1 (Notation). We use an extended notation in the expansion law and its proof to explicitly show substitutions. If we substitute parameter f by $\varphi; f'$, we denote it by $(\varphi; f'/f)$ instead of just writing $(\varphi; f')$. \square

We will define a mini expansion law, and then use it to prove the complete expansion law.

Theorem 5.6.2 (Mini expansion law). *Let Φ, Ψ be sets of trajectories such that $\forall \varphi, \psi \in \Phi \ T(\varphi) = T(\psi)$, $\forall \varphi, \psi \in \Psi \ T(\varphi) = T(\psi)$. Let \mathcal{T}_Φ and \mathcal{T}_Ψ be sets of trajectory qualifiers of Φ and Ψ , respectively. If $h_\Phi = \pi^{\mathcal{T}_\Phi}(h)$ and $h_\Psi = \pi^{\mathcal{T}_\Psi}(h)$, then*

$$\begin{aligned} [f \mid \Phi] . B(f) \parallel_A^H [g \mid \Psi] . C(g) = \\ [h \mid \Phi \times_H \Psi] . \\ \left([f' \mid \Phi \setminus h_\Phi] . B(h_\Phi; f'/f) \parallel_A^H [g' \mid \Psi \setminus h_\Psi] . C(h_\Psi; g'/g) \right) \end{aligned}$$

We will assume that Φ and Ψ do not contain ϵ , because if it does, then it can always be rewritten as $[f \mid \Phi \setminus \epsilon] . B(f) + B(\epsilon)$ according to (5.5).

See Section B.3.1 for a proof. Moreover, we would like to remind that in the definition of composition of sets of trajectories (Definition 5.3.23) closures of sets of trajectories are used ($\overline{\Phi}$ and $\overline{\Psi}$).

Theorem 5.6.3 (Expansion law). *Let*

$$B = \sum_{i \in I} \mathbf{b}_i . B_i + \sum_{j \in J} [f_j \mid \Phi_j] . B_j, \quad C = \sum_{k \in K} \mathbf{c}_k . C_k + \sum_{l \in L} [g_l \mid \Psi_l] . C_l$$

for some terms B_i, B_j, C_k and C_l , actions \mathbf{b}_i and \mathbf{c}_k , trajectories $[f_j \mid \Phi_j]$ and $[g_l \mid \Psi_l]$ and the corresponding sets of qualifiers names \mathcal{T}_{Φ_j} and \mathcal{T}_{Ψ_l} , finite index sets $I \cap J = K \cap L = \emptyset$. Let $h_j = \pi^{\mathcal{T}_{\Phi_j}}(h)$ and $h_l = \pi^{\mathcal{T}_{\Psi_l}}(h)$. Then

$$B \parallel_A^H C = \tag{5.14}$$

$$\sum_{\substack{i \in I \\ \mathbf{b}_i \notin A}} \mathbf{b}_i . (B_i \parallel_A^H C) + \sum_{\substack{k \in K \\ \mathbf{c}_k \notin A}} \mathbf{c}_k . (B \parallel_A^H C_k) + \sum_{\substack{i \in I, k \in K \\ \mathbf{b}_i \in A, \\ \mathbf{b}_i = \mathbf{c}_k}} \mathbf{b}_i . (B_i \parallel_A^H C_k) + \tag{5.15}$$

$$\sum_{\substack{j \in J \\ l \in L}} [h \mid \Phi_j \times_H \Psi_l]. \tag{5.16}$$

$$([f'_j \mid \Phi_j \setminus h_j] . B_j(h_j ; f'_j / f_j) \parallel_A^H [g'_l \mid \Psi_l \setminus h_l]) . C_l(h_l ; g'_l / g_l) \tag{5.17}$$

We will assume that Φ_j and Ψ_j do not contain ϵ , because if it does, then it can always be rewritten as $[f \mid \Phi \setminus \epsilon] . B(f) + B(\epsilon)$ according to (5.5).

The proof of Theorem 5.6.3 is based on Theorem 5.6.2 and Milner [1989, p.96-97]. It is presented in Section B.3.2

5.7 Derived BHPC operators

BHPC is an assembly language for a modelling of hybrid systems. We derive several operators to increase usability of the language. We introduce parametrisation of action prefix in Section 5.7.1. In Sections 5.7.2 and 5.7.3 we introduce two useful operators, *idle* and $\Delta(\text{delay})$, defining a trajectory prefix without any observable behaviour and delay, respectively. And in Section 5.7.4 we introduce a guard operator.

5.7.1 Parametrisation of action prefix

We will use parametrisation of action prefix like in Milner [1989, p.53-58].

$$\mathbf{a}(v : V) . B(v) \triangleq \sum_{v \in V} \mathbf{a}(v) . B(v) \tag{5.18}$$

Parametrisation is frequently used for *value passing*, as it is demonstrated in the bouncing ball example (Example 5.8.1).

5.7.2 Idling

Idling in BHPC is treated as a continuous-time signal without any observable behaviour. We define it as follows

$$\text{idle} . B = [t \mid \bar{0}] . B$$

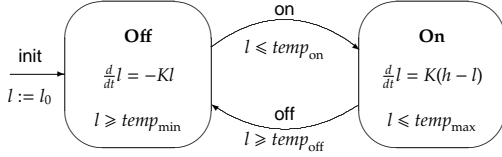


Figure 5.5: A thermostat

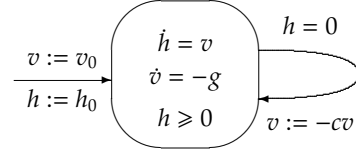


Figure 5.6: A bouncing ball

where t is a reserved variable denoting time and a set of trajectories $\bar{0} = \{(0, t] \rightarrow \mathbb{R}_+ \forall t\}$. We do not use variable t in the formal definition of calculus. However, in an implementation (see, e.g., Chapter 7) it is a part of any trajectory prefix. It does not manifest any observable behaviour, but reacts as soon as it is invoked by another process, which communicates with the process that follows the idling period.

5.7.3 Delays

In BHPC time and time related constructs, e.g., delays, are treated as a continuous-time signals with rate 1. We define delay in a following way

$$\Delta(\text{delay}).B = [t \mid t(0) = 0, t = 1 \Downarrow t = \text{delay}] . B, \quad (5.19)$$

where t is a reserved variable denoting time. We do not use variable t in the formal definition of calculus. However, in an implementation (see, e.g., Chapter 7) it is a part of any trajectory prefix. In contrast to idling (Section 5.7.2) it contains only one trajectory, and can exit only after completing it. Of course, it can take this trajectory in parts.

Such process does not manifest any observable behaviour for delay time units. After delay time units the systems progresses with the process following delay.

5.7.4 Guard

Sometimes it is useful to check some conditions explicitly, and if they are not satisfied, to stop the progress of process. *Guard* is one of such constructs.

$$\langle \text{Pred}(\bar{x}) \rangle . B(\bar{x}) = \sum_{\bar{w} \models \text{Pred}(\bar{w})} B(\bar{w}) \quad (5.20)$$

Here \bar{x} are process parameters variables. Behaviour is very simple, i.e., if a transition can be taken, then it is taken, if and only if the guard is satisfied.

5.8 Application of BHPC

To illustrate the application of BHPC we present several examples.

5.8.1 Bouncing ball

Example 5.8.1 (Bouncing ball). The bouncing ball is described in Section 2.2.1. The hybrid automaton of the bouncing ball is depicted in Figure 5.6. In BHPC it can be

defined in the following way:

$$\begin{aligned} \text{BB}(h_0, v_0) &\triangleq [h, v \mid \Phi(h_0, v_0) \Downarrow h = 0] . \text{BB}(0, -c * v) \\ \Phi(h_0, v_0) &= \{h, v : (0, t] \rightarrow \mathbb{R} \mid \\ &h(0) = h_0, v(0) = v_0, \dot{h} = v, \dot{v} = -g, h \geq 0\} \end{aligned}$$

Trajectory prefix $[h, v \mid \Phi(h_0, v_0) \Downarrow h = 0]$ defines the dynamics of the ball until the bounce, and then the process continues recursively calling itself with updated signals $\text{BB}(0, -c * v)$. Evolution of the simple bouncing ball is depicted in Figure D.2.

The given specification of the bouncing ball can be extended. We add discrete actions to sense the elasticity of the bounce and increase the ball's kinetic energy, and add a compensating controller.

$$\begin{aligned} \text{BB}(h_0, v_0) &\triangleq [h, v \mid \Phi(h_0, v_0) \Downarrow h = 0] . \text{bounce}(c : [0, 1]). \\ &[h, v \mid \Phi(0, -cv) \Downarrow v = 0] . \text{push}(v : \mathbb{R}). \text{BB}(h, v) \\ \text{Control}(v_0) &\triangleq \text{idle} . \text{bounce}(c : [0, 1]). \\ &\text{idle} . \text{push}((1 - c) v) . \text{Control}((1 - c) v) \\ \text{System}(h, v) &\triangleq \text{BB}(h_0, v_0) \parallel_{\text{push, bounce}}^{v_0} \text{Control}(v_0) \end{aligned}$$

□

5.8.2 Thermostat

Example 5.8.2 (Thermostat). The thermostat example is described in Section 2.2.2.

A hybrid automaton of the thermostat is shown in Figure 5.5. It starts with the initial temperature $l_0 \in [\text{tempMin}, \text{tempMax}]$. In locations **Off** and **On** the heater is off and on, respectively, and the temperature changes according to the flow conditions.

The corresponding model in BHPC:

$$\begin{aligned} \text{Thermostat}(l_0) &\triangleq \text{ThOff}(l_0) \\ \text{ThOff}(l_0) &\triangleq [l \mid \Phi_{\text{Off}}(l_0) \Downarrow \text{tempOn} \geq l \geq \text{tempMin}] . \text{on} . \text{ThOn}(l) \\ \text{ThOn}(l_0) &\triangleq [l \mid \Phi_{\text{On}}(l_0) \Downarrow \text{tempOff} \leq l \leq \text{tempMax}] . \text{off} . \text{ThOff}(l) \\ \Phi_{\text{Off}}(l_0) &= \{l : (0, t] \rightarrow \mathbb{R} \mid l(0) = l_0, \dot{l} = -Kl\} \\ \Phi_{\text{On}}(l_0) &= \{l : (0, t] \rightarrow \mathbb{R} \mid l(0) = l_0, \dot{l} = K(h - l)\} \end{aligned}$$

It consists of two process.

- In process **ThOff** the heater is off and the trajectory prefix defines the temperature fall. When the temperature reaches the interval $[\text{tempOn}, \text{tempMin}]$, the process can perform action **on** and switch to the process **ThOn**.
- Process **ThOn** analogously defines the period of heating.

However, it is possible to upgrade such thermostat without changing the specification itself. Let us add a controller that observes temperature and forces the thermostat to switch on and off at exactly tempOn and tempOff , correspondingly:

$$\begin{aligned} \text{Control}(l_0) &\triangleq [l \mid \text{any}(l_0) \Downarrow l = \text{tempOn}] . \text{on} . \\ &[l \mid \text{any}(l_0) \Downarrow l = \text{tempOff}] . \text{off} . \text{Control}(l) \end{aligned}$$

$$\text{UpgradedThermostat}(l_0) \triangleq \text{Thermostat}(l_0) \parallel_{\text{on,off}}^l \text{Control}(l_0)$$

where any(l) is a special function that models an observer, i.e., it accepts any behaviour for l . It works only in parallel composition. Technically it just adds exit conditions to the parallel composition of trajectory prefixes.

The results of simulation of the simple and controlled thermostat are depicted in Figure D.4. Dashed line depicts the evolution of the simple thermostat and solid line depicts the evolution of the coupled version. \square

5.8.3 Dry friction

Example 5.8.3 (Dry Friction). A dry friction is described in Example 3.3.13. Here we provide a BHPC version of it. Three processes are defined, each corresponding to a certain mode of behaviour:

- Process BodyStop defines a behaviour, when the driving force is neutralised by the friction. Corresponding dynamics are defined in Φ_{Stop} . Guards $\langle F_d \geq \mu_0 F_N \rangle$ and $\langle F_d \leq -\mu_0 F_N \rangle$ limit the choice, i.e., if the driving force is positive, then the system switches to the process BodyPos, and if it is negative, then to the process BodyNeg.
- Processes BodyPos and BodyNeg define movement of the body with positive or negative velocity, respectively. Corresponding dynamics are defined in Φ_{Pos} and Φ_{Neg} , respectively. If the driving force becomes to small and is neutralised by the friction, then the system switches to the process BodyStop.

$$\begin{aligned} \text{BodyStop}(x_0, v_0, F_d^0) \triangleq & \left[x, v, F_d \mid \Phi_{\text{Stop}}(x_0, v_0, F_d^0) \Downarrow v \neq 0, F_d \geq |\mu_0 F_N| \right] . \\ & \left(\langle F_d \geq \mu_0 F_N \rangle . \text{BodyPos}(x, v, F_d) + \right. \\ & \left. \langle F_d \leq -\mu_0 F_N \rangle . \text{BodyNeg}(x, v, F_d) \right) \end{aligned}$$

$$\begin{aligned} \text{BodyPos}(x_0, v_0, F_d^0) \triangleq & \left[x, v, F_d \mid \Phi_{\text{Pos}}(x_0, v_0, F_d^0) \Downarrow v = 0, F_d \leq \mu F_N \right] . \\ & \langle v = 0, F_d < \mu_0 F_N \rangle . \text{BodyStop}(x, v, F_d) \end{aligned}$$

$$\begin{aligned} \text{BodyNeg}(x_0, v_0, F_d^0) \triangleq & \left[x, v, F_d \mid \Phi_{\text{Neg}}(x_0, v_0, F_d^0) \Downarrow v = 0, F_d \geq -\mu F_N \right] . \\ & \langle v = 0, F_d > -\mu_0 F_N \rangle . \text{BodyStop}(x, v, F_d) \end{aligned}$$

$$\begin{aligned} \Phi_{\text{Stop}}(x_0, v_0, F_d^0) = & \{x, v, F_d, F_n : (0, t] \rightarrow \mathbb{R} \mid \\ & x(0) = x_0, v(0) = v_0, F_d(0) = F_d^0, \\ & \dot{x} = v, F_d = \sin(t)\} \end{aligned}$$

$$\begin{aligned} \Phi_{\text{Pos}}(x_0, v_0, F_d^0) = & \{x, v, F_d, F_n : (0, t] \rightarrow \mathbb{R} \mid \\ & x(0) = x_0, v(0) = v_0, F_d(0) = F_d^0, \\ & \dot{x} = v, F_d = \sin(t), m\dot{v} = F_d - \mu F_N\} \end{aligned}$$

$$\begin{aligned} \Phi_{\text{Neg}}(x_0, v_0, F_d^0) = & \{x, v, F_d, F_n : (0, t] \rightarrow \mathbb{R} \mid \\ & x(0) = x_0, v(0) = v_0, F_d(0) = F_d^0, \\ & \dot{x} = v, F_d = \sin(t), m\dot{v} = F_d + \mu F_N\} \end{aligned}$$

It is easy to see that by bringing conditions out in the trajectory prefix we can easily increase readability and clarity of specifications in some cases. \square

5.8.4 Two tanks

Example 5.8.4 (Two tanks). Consider the two tanks model [Lygeros and Sastry, 1999, De Schutter and Heemels, 2004], where both tanks have one common fluid source that provides fluid at the rate of l_{in} units per second. Through a pipe, the fluid source can be directed either to the left tank or to the right tank. Both tanks have openings at the bottom, and from the tanks water drains at the rates of d_{left} and d_{right} units per second, respectively. Initially, the tanks contain l_{left}^0 and l_{right}^0 units of fluid, respectively. The pipe can switch between the tanks instantaneously. The objective is to keep the fluid volumes in the interval $(l_{\text{min}}, l_{\text{max}})$.

Let l_{left} and l_{right} are volumes in the left and right tanks, correspondingly.

$$\begin{aligned} \text{TwoTanks}(l_{\text{left}}^0, l_{\text{right}}^0) &\triangleq \\ &\left[l_{\text{left}}, l_{\text{right}} \mid \Phi_{\text{left}}(l_{\text{left}}^0, l_{\text{right}}^0) \Downarrow l_{\text{left}} = l_{\text{max}} \vee l_{\text{right}} = l_{\text{min}} \right].\text{FillRight}. \\ &\left[l_{\text{left}}, l_{\text{right}} \mid \Phi_{\text{right}}(l_{\text{left}}, l_{\text{right}}) \Downarrow l_{\text{left}} = l_{\text{min}} \vee l_{\text{right}} = l_{\text{max}} \right].\text{FillLeft}. \\ &\text{TwoTanks}(l_{\text{left}}, l_{\text{right}}) \\ \Phi_{\text{left}}(l_{\text{left}}^0, l_{\text{right}}^0) &= \{l_{\text{left}}, l_{\text{right}} : (0, t] \rightarrow \mathbb{R} \mid \\ &l_{\text{left}}(0) = l_{\text{left}}^0, l_{\text{right}}(0) = l_{\text{right}}^0, \dot{l}_{\text{left}} = d_{\text{left}} + l_{\text{in}}, \dot{l}_{\text{right}} = d_{\text{right}}\} \\ \Phi_{\text{right}}(l_{\text{left}}^0, l_{\text{right}}^0) &= \{l_{\text{left}}, l_{\text{right}} : (0, t] \rightarrow \mathbb{R} \mid \\ &l_{\text{left}}(0) = l_{\text{left}}^0, l_{\text{right}}(0) = l_{\text{right}}^0, \dot{l}_{\text{left}} = d_{\text{left}}, \dot{l}_{\text{right}} = d_{\text{right}} + l_{\text{in}}\} \end{aligned}$$

This system is of interest, because by ignoring physical reality one can devise a controller that keeps both water levels within the required bounds: whenever l_{left} falls to l_{min} , direct the pipe to the left tank, and whenever l_{right} falls to l_{min} , direct the pipe to the right tank (or, corresponding switching at the l_{max}). However, such a controller cannot be realised physically, because it would cause the pipe to switch back and forth infinitely often within a finite amount of time, if $d_{\text{left}} + d_{\text{right}} \neq l_{\text{in}}$.

To demonstrate compositional modelling advantages we propose a slightly different version of the same system. In this specification l is water level in the tank, d_{out} is a drain rate and l_{in} is an inflow rate.

$$\begin{aligned} \text{TankIn}(l_0, d_{\text{out}}, l_{\text{in}}) &\triangleq \left[l \mid l(0) = l_0, \dot{l} = d_{\text{out}} \Downarrow \text{true} \right].\text{off}.\text{TankOut}(l, d_{\text{out}}, l_{\text{in}}) \\ \text{TankOut}(l_0, d_{\text{out}}, l_{\text{in}}) &\triangleq \left[l \mid l(0) = l_0, \dot{l} = d_{\text{out}} + l_{\text{in}} \Downarrow \text{true} \right].\text{off}.\text{TankIn}(l, d_{\text{out}}, l_{\text{in}}) \\ \text{Controller}(l_{\text{left}}^0, l_{\text{right}}^0) &\triangleq \left[l_{\text{left}}, l_{\text{right}} \mid \text{any}(t) \Downarrow l_{\text{left}} = l_{\text{max}} \vee l_{\text{right}} = l_{\text{min}} \right].\text{fill}_{\text{right}}. \\ &\left[l_{\text{left}}, l_{\text{right}} \mid \text{any}(t) \Downarrow l_{\text{left}} = l_{\text{min}} \vee l_{\text{right}} = l_{\text{max}} \right].\text{fill}_{\text{left}}. \\ &\text{Controller}(l_{\text{left}}, l_{\text{right}}) \end{aligned}$$

$$\begin{aligned} \text{System}(l_{\text{left}}^0, l_{\text{right}}^0, dl_{\text{out}}, dr_{\text{out}}, ll_{\text{in}}, lr_{\text{in}}) \triangleq & \\ & \left(\text{TankOn}(l_{\text{left}}^0, dl_{\text{out}}, ll_{\text{in}}) \left[l_{\text{left}}/l, dl_{\text{out}}/d_{\text{out}}, ll_{\text{in}}/l_{\text{in}}, \text{fill}_{\text{left}}/\text{on}, \text{fill}_{\text{right}}/\text{off} \right] \parallel \right. \\ & \left. \text{TankOn}(l_{\text{right}}^0, dr_{\text{out}}, lr_{\text{in}}) \left[l_{\text{right}}/l, dr_{\text{out}}/d_{\text{out}}, lr_{\text{in}}/l_{\text{in}}, \text{fill}_{\text{right}}/\text{on}, \text{fill}_{\text{left}}/\text{off} \right] \right) \\ & \parallel_{\text{fill}_{\text{left}}, \text{fill}_{\text{right}}}^{l_{\text{left}}^0, l_{\text{right}}^0} \text{Controller}(l_{\text{left}}, l_{\text{right}}) \end{aligned}$$

□

5.9 An experimental version of calculus

As a part of development of BHPC, we have explored several versions of the calculus and operators. The “main” calculus was presented above; however, we would like to introduce an interesting experimental alternative.

Essential difference is a changed trajectory prefix (5.3) and a special version of choice, so called *superposition*. We are not going to discuss all the differences between calculi, but only point out some interesting features.

Trajectory prefix An alternative trajectory prefix is a more elementary notion than that of an ordinary trajectory prefix.

$$[\varphi].B \xrightarrow{\varphi} B \quad (5.21a)$$

$$[\varphi].B \xrightarrow{\psi} [\varphi \setminus \psi].B \quad \text{for all } \psi < \varphi \quad (5.21b)$$

In (5.21a) a process engages in the trajectory φ , completes it and then behaves as described by B . While in (5.21b) only a part of the trajectory is taken, and then the process will continue with the remainder of the trajectory ($\varphi \setminus \psi$).

It is easy to notice that it is a more elementary version of trajectory prefix.

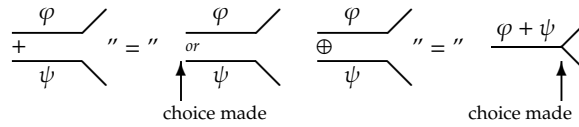


Figure 5.7: Superposition

Superposition *Superposition* is a generalised operator on sets of behaviour expressions. To generate the set we allow arbitrary index sets I . It can be thought of as a generalisation of the choice \sum in ordinary process algebra. Indeed, if all arguments are of the form $a_v.B(v)$ then the intended interpretation of

$$\oplus \{B(v) \mid v \in I\} \text{ or } \oplus_{v \in I} B(v) \text{ is } \sum_{v \in I} B(v)$$

The difference between \oplus and \sum becomes apparent in the case of trajectory prefixes: when two trajectories are superposed the choice between them is not made at

the time of superposition, but at the time when the trajectories start bifurcating, as illustrated in Figure 5.7.

$$\frac{B(w) \xrightarrow{a} B'}{\bigoplus_{v \in I} B(v) \xrightarrow{a} B'} \quad w \in I \quad (5.22a)$$

$$\frac{\{B(v) \xrightarrow{\varphi} B'(v) \mid v \in I\}, \{B(w) \xrightarrow{\varphi} B'(w) \mid w \in J\}, I \neq \emptyset}{\bigoplus_{v \in I \cup J} B(v) \xrightarrow{\varphi} \bigoplus_{v \in I} B'(v)} \quad (5.22b)$$

In (5.22a) the superposition for action prefixes is defined. Rule (5.22b) tells that we can not distinguish between two signal transitions, if they can take the same trajectory, and if some processes can evolve according to a certain trajectory, and another set can not, then the choice is resolved in the favour of the processes, which can take the trajectory.

Problems Main problems in this case raise from the negative premises in the superposition definition. It is not clear, do we get a sound transition system with a rule with negative premises.

One of the most important differences is that we were able to prove that bisimulation is a congruence in the main version of calculus, while it remains an open question for the calculus with superposition, because we were not able to find a suitable stratification mapping.

However, we believe that a further investigation of both process calculi and implications of the above mentioned differences on modelling and analysis of hybrid systems may be of interest.

5.10 Conclusions

In this chapter we have introduced the hybrid process calculus BHPC. It has been introduced in several steps. First of all we have defined notions of signal space (Definition 5.3.1) and trajectories (Definition 5.3.3), where the notion of trajectory is a basic element defining continuous-time evolution and corresponds to a notion of action in discrete systems. Based on these two notions and a classical technique to define dynamic behaviour of systems, i.e., labelled transition systems, we have introduced hybrid transition systems (Definition 5.4.1). Together with a suitable adaptation of the classical notion of hybrid strong bisimulation (introduced in Definition 5.4.7) it yields a mathematical interpretation of hybrid behaviour, viz. as equivalence classes of hybrid transition systems modulo bisimulation that can be interpreted as a generalisation of the behavioural approach to classical dynamic systems. Moreover, it can be seen as a humble step towards model-based testing techniques that use transition systems, e.g., Larsen et al. [2003], Briones and Brinksma [2004, 2005].

To specify how hybrid transition systems are built, we have defined a language for BHPC (Section 5.5.1) and have given a semantics using structural operational semantics rules [Plotkin, 1981].

5. BEHAVIOURAL HYBRID PROCESS CALCULUS

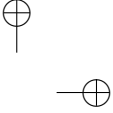
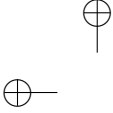
Furthermore, hybrid strong bisimulation equivalence on hybrid transition systems is a congruence w.r.t. the operators of BHPC.

We have augmented our calculus with the expansion law (Definition 5.6.3) that provides a mechanism to transform a parallel composition of two processes to a choice of processes by pushing the parallel composition one step further.

To increase the applicability of the calculus we have added several derived operators, e.g., a parameterised version of action prefix, idling, delay and guard. We have illustrated application of the calculus by a set of small examples.

In the last but one section we have discussed an alternative version of the calculus that has some advantages and drawbacks compared with the “main” version.

We believe that the calculus provides a sound and convenient framework for the investigation of hybrid phenomena and for research of techniques that could help to deal with it. BHPC combines the behavioural approach (Section 5.2) and process algebraic theory. Therefore, as future research directions we see not only a development of brand new techniques and theories, but also an adaptation of results from both areas. We believe that the development of analytical techniques for the analysis of diverse properties, e.g., stability, would help to evaluate the conceptual and practical implications of our approach as well as to provide new insight and ideas on the problems itself. Furthermore, we expect that a notion of weak bisimulation [Milner, 1989] for hybrid systems could provide a lot insight on the interchangeability of components of embedded systems and a better intuition on hybrid phenomena.



A certain old woman, out of excessive curiosity, fell out of a window, plummeted to the ground, and was smashed to pieces.

Another old woman leaned out of the window and began looking at the remains of the first one, but she also, out of excessive curiosity, fell out of the window, plummeted to the ground and was smashed to pieces.

Then a third old woman plummeted from the window, then a fourth, then a fifth.

By the time a sixth old woman had plummeted down, I was fed up watching them, and went off to Mal'tsevsky Market where, it was said, a knitted shawl had been given to a certain blind man.

Daniil Kharms

6

BHPC in context of related frameworks

6.1 Introduction

In Chapter 5 we presented *Behavioural Hybrid Process Calculus* (BHPC), a hybrid systems modelling and analysis framework based on the combination of the behavioural approach (Section 5.2) and process algebraic theory (Section 3.3.9). However, it is not the only approach that deals with hybrid phenomena. In Chapter 3 we surveyed some of the major approaches for modelling and analysis of hybrid systems. In this chapter we aim at establishing the place of BHPC in the context of the hybrid systems realm.

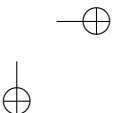
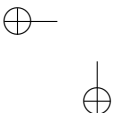
In Section 3.3.1 the formalisms are grouped to *dynamical systems*, *transition systems* based frameworks and *simulation languages*. We follow the same tactics here by starting with comparison of BHPC and dynamical systems. Then we proceed by matching it to the other transition systems based frameworks. The comparison is closed by putting BHPC alongside the simulation languages.

6.2 BHPC and hybrid dynamical systems

In Section 3.3.1 we singled out *hybrid dynamical systems*, i.e., piecewise affine systems (Section 3.3.2), mixed logical dynamical systems (Section 3.3.3), complementarity systems (Section 3.3.4) and max-min-plus-scaling systems (Section 3.3.5), as extensions of ordinary dynamical systems. Essentially, these approaches combine usual ways of defining continuous behaviour (ODE/DAE) and some basic means to for switching between the different continuous evolutions.

The underlying structure of BHPC considerably differs from the hybrid dynamical systems. However, certain aspects of approaches are comparable.

All the continuous-time behaviour that is expressible in any of the formalisms defined in Sections 3.3.2–3.3.5 can be defined in BHPC as well. The trajectory based



approach adopted in BHPC allows to use any means of continuous-time behaviour representation that are used in the above mentioned formalisms. The inverse holds only in some cases, i.e., when the trajectories can be expressed by ODE/DAE (linear or non-linear, depending on the formalism). Switching behaviour can be translated as well.

We do not provide transformation methods for all above mentioned hybrid dynamical systems. However, we demonstrate, how *piecewise affine systems* (PWA) (Section 3.3.2) and *mixed logical dynamical systems* (MLD) (Section 3.3.3) can be transformed to BHPC, in (6.1a) and (6.1b), respectively, with N as a number of modes. Notice that to simulate the specification in Behavioural Hybrid Process Calculus input $u(t)$ should be specified.

$$\begin{aligned} \text{Process}_i(x^0) &\triangleq [x, y, u \mid \Phi_i(x^0) \downarrow [x \ u]^T \in \Omega_i] \cdot \sum_{j=1, \dots, N} \left(\left\langle \begin{bmatrix} x \\ u \end{bmatrix} \in \Omega_j \right\rangle \cdot \text{Process}_j(x) \right) \\ \Phi_i(x^0) &= \{x : (0, t] \rightarrow \mathbb{R}^n, y : (0, t] \rightarrow \mathbb{R}^p, u : (0, t] \rightarrow \mathbb{R}^m \mid \\ &\quad x(0) = x^0, \dot{x} = A_i x + B_i u + f_i, y = C_i x + D_i u + g_i\} \end{aligned} \quad (6.1a)$$

$$\begin{aligned} \text{Process}_i(x^0) &\triangleq [x, y, u \mid \Phi_i(x^0)]. \\ &\quad \sum_{j=1, \dots, N} \left(\left\langle E_{2t}^j \delta(t) + E_{3t}^j z(t) \geq E_{1t}^j u(t) + E_{4t}^j x(t) + E_{5t}^j \right\rangle \cdot \text{Process}_j(x) \right) \\ \Phi_i(x^0) &= \{x : (0, t] \rightarrow \mathbb{R}^{n_c} \times \{0, 1\}^{n_i}, y : (0, t] \rightarrow \mathbb{R}^{m_c} \times \{0, 1\}^{m_i}, \\ &\quad u : (0, t] \rightarrow \mathbb{R}^{k_c} \times \{0, 1\}^{k_i} \mid \\ &\quad x(t+1) = A_t^i x(t) + B_{1t}^i u(t) + B_{2t}^i \delta(t) + B_{3t}^i z(t) \\ &\quad y(t) = C_t^i x(t) + D_{1t}^i u(t) + D_{2t}^i \delta(t) + D_{3t}^i z(t) \\ &\quad E_{2t}^i \delta(t) + E_{3t}^i z(t) \geq E_{1t}^i u(t) + E_{4t}^i x(t) + E_{5t}^i \} \end{aligned} \quad (6.1b)$$

Moreover, De Schutter and Heemels [2004] shows that piecewise affine systems (Section 3.3.2), mixed logical dynamical systems (Section 3.3.3), complementarity systems (Section 3.3.4) and max-min-plus-scaling systems (Section 3.3.5) are equivalent (some under certain assumptions) and how transformation amongst some of them can be carried out. Thereby, the transformation to BHPC can be achieved in several steps.

In contrast, the inverse transformations (from BHPC to one of hybrid dynamical formalisms) are possible only for a restricted classes of BHPC. Furthermore, it is not clear, how to express parallel composition, and modularity in general, in dynamical systems.

6.3 BHPC and transition systems based approaches

Frequently formalisms originating from computer science are implicitly or explicitly built on the transitions systems. As was explained in Sections 3.3.9 and 5.4, transition systems consist of states and transitions. Moreover, a language as in process algebras or a visual representation as automata are used to describe the way, the transition systems are built.

In this section we compare BHPC to such formalisms by studying the corresponding operators and other relevant properties.

Continuous-time behaviour. In BHPC continuous-time behaviour is represented by the trajectories. Similar approaches are chosen in hybrid I/O automata (HIOA, Section 3.3.8) and hybrid behavioural automata (HBA, Section 3.3.7). However, in HIOA the trajectories are defined in a different way, for example HIOA has *point* trajectories and trajectories with left and right closed or open domains in contrast to BHPC where only trajectories with time-domain $(0, t]$ are allowed. Other approaches are more restrictive than BHPC, i.e., continuous-time behaviour should be represented by the differential equations.

Hybrid transition systems. Hybrid transition systems are not so different in BHPC, ACP_{hs}^{srt} (Section 3.3.9), HyPA (Section 3.3.9) and Hybrid χ (Section 5). However, in HyPA and Hybrid χ a successful termination is defined in contrast to BHPC.

Choice. Choice is non-deterministic for actions, but different ways to resolve choice for continuous-time behaviour are chosen. In BHPC and HyPA it is done at the beginning of transition. In Hybrid χ time progress does not resolve it, however, if the trajectories separate, it leads to deadlock. Only actions can resolve choice in ACP_{hs}^{srt} , time progress does not resolve it. Moreover, if signals bifurcate while time passes, it deadlocks.

Parallel composition. In BHPC parallel composition of actions is resolved in the interleaving fashion and signals synchronise on common qualifiers. In parallel composition of continuous-time behaviour of HyPA the participating processes synchronise on the common part of the global signal space. In Hybrid χ a different communication paradigm is chosen, i.e., it is carried out via *directed* channels, in contrast to BHPC, where direction is not important. A similar choice was made in HIOA, where inputs and outputs are specified. In MASACCIO (Section 3.3.10) and CHARON (Section 3.3.11) variables are separated to inputs and outputs.

Bisimulation. In BHPC the hybrid strong bisimulation is a congruence relation with respect to the defined operators. The same holds in Hybrid χ . In contrast, it is not a congruence for the parallel composition of subsystems in Bergstra and Middelburg [2005]. In HyPA only special types of bisimulation, i.e., robust and stateless bisimulations are congruences, and strong bisimulation is not.

Mobility and Φ -calculus. In Φ -calculus (Section 5), instead of mixing a continuous (signal) flow and discrete actions, systems are discrete entities surrounded by the continuous world. The discrete process can interact with the environment, where evolution is typically described by the autonomous differential equations and typically over \mathbb{R} . In addition, the processes can reconfigure themselves.

BHPC is more flexible w.r.t. defining continuous behaviour, and its evolution, while in Φ -calculus it is possible to change configuration of the system.

```

algorithm HAtoBHPC (ha:HYBRID AUTOMATON)
begin
  forall  $l \in L$  do
     $exit\_cond = \emptyset$ ; // exit conditions
     $guardActProc_l = \emptyset$ ; // corresponding guard, action and recursive call
    forall  $(l, \sigma_j, l_i) \in E$  do // for all outgoing transitions
      forall  $x_i \in \mathbf{x}$  do // calculate resets
        if  $x_i \in \text{dom}(\text{Assign}(l, \sigma_j, l_i))$ 
          then  $x'_i = \text{Assign}(l, \sigma_j, l_i)(x_i)$ ; //  $x_i$  value is changed by the transition
          else  $x'_i = x_i$ ; //  $x_i$  value does not change
        end
      end
       $guardActProc_l = guardActProc_l + \langle \text{Guard}(l, \sigma_j, l_i) \rangle, \sigma_j \cdot \text{process}_{l_i}(\mathbf{x}')$ ;
       $exit\_cond_l = exit\_cond_l \vee \text{Guard}(l, \sigma_j, l_i)$ ;
    end
     $\text{process}_l(\mathbf{x}_0) \triangleq \text{substitute}(x, x_0, guardActProc_l) +$  // substitute  $x$  by the initial values  $x_0$ 
     $[x \mid \mathbf{x}(0) = \mathbf{x}_0, \dot{\mathbf{x}} = f_l \downarrow \text{Inv}(l) \downarrow \text{inv}_l].guardActProc_l$ ;
  end
  forall  $(l, \mathbf{x}) \in \text{Init}$  do
     $\text{initProcess}_{l,\mathbf{x}} \triangleq \text{process}_l(\mathbf{x})$ ;
  end
  return process and initProcess;
end

```

Algorithm 6.1: Transformation of hybrid automaton to BHPC

6.3.1 Hybrid automata and BHPC

We give special treatment for hybrid automata (HA, Section 3.3.6) and show how it can be translated to BHPC.

Hybrid automata can be easily transformed to the BHPC. A simple Algorithm 6.1 for this procedure is presented.

- Every initial state is transformed to an invocation of the corresponding process with the corresponding initial values.
- All locations are transformed to the corresponding processes.
- All outgoing transitions of the particular locations are collected to the corresponding sets of the exit conditions, the action prefixes and the resets, and then added to a generalised choice.

Remark 6.3.1. There are no trajectory prefixes with zero duration (however there exists an empty trajectory that was introduced for the technical reasons) in the BHPC, while in the hybrid automata a system can leave a location immediately. Therefore for the straightforward translation we add a choice with all exit transitions with the dynamics. \square

All processes are transformed according to Schema 6.2. In the schema l is a current location, f_l are flow conditions in the current location, and Inv is an invariant of the

current location. In the choice all outgoing transitions are listed (with the guards *Guard* and the action prefixes (labels) σ). *Assign* is extended, i.e., if a variable is not in the reset map, then it is considered to be a part of the map, but with an unchanged value. Every location is transformed in the same manner.

$$\begin{aligned} \text{process}_i(\mathbf{x}_0) \triangleq & \sum_{\forall(l, \sigma_j, l_i) \in E} \left(\langle \text{Guard}(l, \sigma_j, l_i) \rangle . \sigma_j . \text{process}_{l_i}(\text{Assign}(l, \sigma_j, l_i)) \right) + \\ & \left[\mathbf{x} \mid \mathbf{x}(0) = \mathbf{x}_0, \dot{\mathbf{x}} = f_i(\mathbf{x}) \downarrow \text{Inv}(l) \parallel \bigvee_{\forall(l, \sigma_j, l_i) \in E} \text{Guard}(l, \sigma_j, l_i) \right]. \quad (6.2) \\ & \sum_{\forall(l, \sigma_j, l_i) \in E} \left(\langle \text{Guard}(l, \sigma_j, l_i) \rangle . \sigma_j . \text{process}_{l_i}(\text{Assign}(l, \sigma_j, l_i)) \right) \end{aligned}$$

Remark 6.3.2 (Initial state). In the BHPC we do not define an initial state at the time 0 in contrast with hybrid automaton. If in the hybrid automaton an initial state is unique, then in the BHPC the initial state is a left limit of the initial trajectory. We overcome this problem by adding a choice amongst direct exits with trajectory prefix succeeded by the exits. \square

Remark 6.3.3 (Different semantics of HA). In some definitions of hybrid automata it is allowed to take a discrete transition iff the invariant of the target location is satisfied after it. In such a case an additional condition is conjoined with corresponding trajectory prefix exit conditions and process exit guards. It estimates invariant condition of the target location with re-assigned values and can be defined in the following way $\text{Inv}(l_i)(\text{Assign}(l, \sigma_j, l_i))$ with the resulting expression $\text{Guard}(l, \sigma_j, l_i) \wedge \text{Inv}(l_i)(\text{Assign}(l, \sigma_j, l_i))$. \square

Remark 6.3.4. Hybrid behavioural automata (Section 3.3.7) is a behavioural [Polderman and Willems, 1998] extension of hybrid automaton (Section 3.3.6). BHPC is a mix of behavioural and process algebraic approaches. Therefore, a translation from HBA to BHPC is even more natural than a translation of hybrid automaton to BHPC (Section 6.3.1). By applying according modifications to the Algorithm 6.1 we get such transformation. \square

We illustrate the application of the algorithm in the Example 6.3.5.

Example 6.3.5 (Thermostat). The thermostat example is described in Section 2.2.2. Hybrid automaton of the thermostat is depicted in Figure 5.5. Algorithm 6.1 transforms it to the following BHPC model.

$$\begin{aligned} \text{process}_{\text{Off}}(l_0) \triangleq & \langle l_0 \leq \text{tempOn} \rangle . \text{on} . \text{process}_{\text{On}}(l_0) + \\ & \left[l \mid l(0) = l, \frac{d}{dt}l = -k * l \downarrow l \geq \text{tempMin} \parallel l \leq \text{tempOn} \right]. \\ & \langle l \leq \text{tempOn} \rangle . \text{on} . \text{process}_{\text{On}}(l); \\ \text{process}_{\text{On}}(l_0) \triangleq & \langle l_0 \geq \text{tempOff} \rangle . \text{off} . \text{process}_{\text{Off}}(l_0) + \\ & \left[l \mid l(0) = l, \frac{d}{dt}l = k(h - l) \downarrow l \leq \text{tempMax} \parallel l \geq \text{tempOff} \right]. \\ & \langle l \geq \text{tempOff} \rangle . \text{off} . \text{process}_{\text{Off}}(l); \end{aligned}$$

$$\text{initProcess}_{\text{Off},l_0} \triangleq \text{process}_{\text{Off}}(l_0);$$

□

6.4 BHPC and simulation languages

Simulation languages are specialised languages designed for simulation, while Behavioural Hybrid Process Calculus is a general framework for theoretical analysis of hybrid phenomena itself, as well as modelling and analysis of hybrid systems. Therefore, a comparison is not altogether appropriate. Nevertheless we look at the principal similarities and differences

Bond graphs and BHPC

Bond graphs (Section 3.3.12) represent the energy-based interaction structure of some system. It implies that bond graphs are restricted to the systems which can be naturally represented using energy as the basic concept. In contrast, BHPC is a general framework for modelling and analysis of the dynamical systems, based on such fundamental concepts, as trajectories and actions. An effort to connect the bond graphs and hybrid process algebras was made in Cuijpers [2004, p.96–140], Cuijpers et al. [2004] by constructing *constitutive hybrid processes*. This example shows that to some extent interaction of several different formalisms gives some positive results in modelling of the certain classes of hybrid systems. One of the future research directions for BHPC can be establishing connection with bond graphs.

Modelica and BHPC

Modelica™ (Section 3.3.13) represents a pragmatic approach to simulation of physical systems. It can be used to model certain classes of hybrid systems. In Modelica™ in case of specified events, a simulation of physical process can be interrupted and modified according to the specification. Non-determinism is not supported, in contrast to BHPC, where a nondeterministic choice for both signals and actions is available. Summarising, BHPC and Modelica™ are more complementary, than competing, approaches.

Currently research is carried on in using Modelica™ and Dymola as simulation platform for a subset of BHPC [van Putten, 2006].

6.5 Conclusions

In this chapter Behavioural Hybrid Process Calculus was compared to several important frameworks for modelling and analysis of hybrid systems. It was shown that BHPC has most of the features supported by other approaches. While it may miss some strengths of specialised approaches as the simulation languages or Φ -calculus (mobility), it has some strong points too. The behavioural world-view allows to handle any type of continuous-time behaviour while some approaches are restricted to ODE/-DAE. Furthermore, hybrid strong bisimulation is a congruence, and it is indispensable for modular design of large systems.

First it marked out a race-course, in a sort of circle, ('the exact shape doesn't matter,' it said,) and then all the party were placed along the course, here and there. There was no 'One, two, three, and away,' but they began running when they liked, and left off when they liked, so that it was not easy to know when the race was over. However, when they had been running half an hour or so, and were quite dry again, the Dodo suddenly called out 'The race is over!'...

Lewis Carroll

7

Simulation of Behavioural Hybrid Process Calculus

7.1 Introduction

Simulation is a well-established technique for development and analysis of dynamical systems. It is widely used in industry and academia. Moreover, development of complex embedded systems that contain a multitude of diverse components is hardly possible without employment of simulation tools. Frequently simulation is used in building models of designs or existing systems (especially in biology, chemistry, physics, social sciences, etc.) that adequately model reality.

Often such systems or adequate abstractions of them exhibit hybrid phenomena. That is where simulation of hybrid systems comes in play. And while it is not so widely spread as simulation of continuous-time systems, it is getting common rapidly, especially in the embedded systems area.

We will distinguish two important simulation application areas.

- Simulation in *model development*.
- Simulation in *system analysis*.

These two exercises emphasise slightly different aspects of simulation, but the generic features are common.

Simulation in model development Simulation is regularly used as a model development tool. Usually it is employed to carry on several procedures.

- While building a model, simulation is used to ascertain that the components of the model and the model itself behave as expected.

- If an error-trace is provided, a simulation tool can help to examine the path that leads to an error and analyse the reasons of the error.

Moreover, sometimes simulation is employed in a testing manner. The model can be validated against the real system (and inverse) by feeding the same inputs to both of them, and then comparing the outputs.

Simulation in system analysis Simulation is frequently employed as a system analysis tool. Given a model, a set of experiments is designed to probe the system.

- Chosen properties can be tested. Such tests do not guarantee that the properties hold always, but nevertheless they provide a beneficial insight on the systems' behaviour.
- Experiments for performance analysis can be designed and carried out. The results provide a good intuition on the efficiency of model, assist in detecting bottlenecks and inefficient components.

7.1.1 Simulation of continuous and discrete systems

Simulation of hybrid systems, as hybrid systems itself, has two principal precursors, i.e., simulation of continuous and of discrete systems. Simulation of continuous systems is a well-established industrial practice and area of research. However, simulation of discrete systems is often substituted by testing in software industry and verification (e.g., model checking) in computer science.

Simulation of continuous systems Simulation of continuous systems is an established area in science and is regularly used in industry. There is a plentitude of industrial and academic tools for simulation of continuous systems. Some of them support only simulation of ordinary differential equations¹ (ODE), but tools supporting differential algebraic-equations² (DAE) are getting commonplace too. In simulation of hybrid systems simulation of continuous systems can be seen as a simulation of the continuous part of hybrid systems with some additional requirements (e.g., an accurate event detection) and therefore results of research in this area can be reused in simulation of hybrid systems. For more information about simulation of continuous systems see Cellier [1991], Zeigler et al. [2000].

Simulation of discrete systems Simulation of discrete systems is an important area in computer science as well as in control theory. Usually it is simulation of various types of automata (e.g., Bengtsson et al. [1998], Kaynar et al. [2002], Behrmann et al. [2004]), process algebras (e.g., Bolognesi and Brinksma [1987], van Eijk [1988], Eertink [1994]) in computer science or simulation of discrete event systems [Zeigler et al., 2000] in control theory.

¹An ordinary differential equation is a relation that contains functions of only one independent variable, and one or more of its derivatives with respect to that variable. It's general form is $F(x, y, \dot{y}, \ddot{y}, \dots, y^{(n)}) = 0$.

²A differential-algebraic equation is a relation in which the derivatives are not (in general) expressed explicitly, and typically derivatives of some of the dependent variables may not appear in the equations at all. General form of such equations is $F(\dot{x}, x, y, t) = 0$, where $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$ are differential and algebraic variables, respectively.

7.1.2 Simulation of hybrid systems

Simulation of hybrid systems combines continuous-time and discrete behaviour simulation. Unfortunately, it is not enough just to put together these two types of simulators, because additional means are needed to deal with interaction of the two worlds. Therefore, new procedures and techniques are designed to deal with hybrid phenomena as well as continuous-time and discrete behaviours.

There exists a number of techniques for simulation of hybrid systems. The major ones [van der Schaft and Schumacher, 2000, p.25–30] are the following.

The smoothing method. The hybrid model is transformed (or attempt is taken to transform) to a *smooth model* that approximates selected aspects of the hybrid model. Such an approach gives an impression of being superfluous, i.e., a system is modelled as a hybrid, and then again transformed to a smooth system. Potential benefit is that it might be easier to model a system as hybrid. For more information see van der Schaft and Schumacher [2000, p.25–26].

The event tracking method. It is the prevalent way of simulating hybrid systems. The simulation sequence is the following (slightly modified version from van der Schaft and Schumacher [2000, p.26–28])

1. Event handling:
 - (Re-)Initialisation of a (new) continuous state;
 - Determination of a (new) *mode*.
2. Simulation of the continuous-time dynamics within a given mode.
3. Event detection.

First of all, the initial conditions (the initial mode/process and the initial signal values) are set. Then the continuous-time behaviour is simulated until an event is detected. If an event occurs, the exact (or the closest conceivable) event time and the corresponding continuous state (signals) values are computed. Then the new values and the mode are determined and the simulation cycle repeats.

The time-stepping method. In this approach events are not tracked. Some discretisation scheme is chosen, and then the hybrid system is approximated by the discretised system. The approach may work quite well for the specific classes of hybrid systems with carefully chosen discretisation schemes. The broader discussion and an example are available in van der Schaft and Schumacher [2000, p.28–30].

Event tracking algorithms The *event tracking method* is the *de facto* standard of hybrid systems simulation. However, it comes in several different flavours. Andersson [1994, p.116] discusses the application of this technique to object-oriented simulation of hybrid systems. Use of the event tracking in 20-sim tool (Section 7.9) is discussed in Broenink and Weustink [1996]. Mosterman and Biswas [2002] discusses the application of the event tracking method for simulation of physical dynamical systems. Simulation of hybrid systems in Ptolemy II (Section 7.9) is described in Liu and Lee

[2002]. Barton and Lee [2002, p.266] uses a classical approach with small variations for hybrid dynamical systems modelling and simulation framework.

Meanwhile Fabian et al. [1998] adopt a simulation algorithm for Hybrid χ (Section 5) by adding a treatment for processes. The algorithm proceeds in the following order:

1. The system is initialised (the state and the processes).
2. While there are active processes and simulation time is not finished.
 - (a) While there are active processes:
 - i. Non-blocking statements of the active processes are executed;
 - ii. Initial states are calculated;
 - iii. The conditions are checked and the processes are sorted to active and inactive.
 - (b) If there are time-outs, which will be activated sooner than a minimal solver step, the processes which were blocked by time, are activated and the cycle in step (2) is repeated.
 - (c) If the minimal solver step is shorter than the time to the nearest time-out, DAE are solved and the cycle (2) is repeated.

7.2 Behavioural Hybrid Process Calculus simulation algorithm

In this section we propose a technique for simulation of Behavioural Hybrid Process Calculus based on the modified event tracking algorithm presented in Section 7.1.2. Part of the algorithms presented in this section were tested in a prototype implementation (Appendix D) and Krilavičius and Schonenberg [2005], Schonenberg [2006], van Putten [2006].

7.2.1 Language

Different operators are used in process algebraic languages. Sometimes certain operators are used for theoretical language development and have corresponding versions (sometimes slightly different) adapted for a practical use. We choose the following operators for simulation of BHPC:

$$B ::= 0 \mid a(v).B \mid [f \mid \Phi].B \mid \langle Pred \rangle .B \mid \sum_{i \in I} B_i \mid B \parallel_A^H B \mid \text{new } w.B \mid B[\sigma] \mid P$$

Notice that several operators from the original language are restricted, and a guard operator is added.

- An original action prefix (5.2) is replaced by the parameterised action prefix (Section 5.7.1). It allows to change values of trajectory qualifiers instantaneously (by relating them with an action prefix). Moreover, if there are no parameters, it behaves in the same way as ordinary action prefix.

- Additional requirements are added to a trajectory prefix (5.3). We require that all trajectories in a trajectory prefix do not bifurcate, but only have different durations, i.e., $\forall \varphi, \psi \in \Phi$ holds $\varphi < \psi$ or $\psi < \varphi$. Such restrictions allow to treat all trajectory prefixes as defined by ODE/DAE with initial conditions and different simulation time.
- A guard operator ($\langle Pred \rangle . B$) is added.
- Hiding (new $w.B$) is restricted to action prefixes (for simplicity reasons, see below for the explanations).
- Renaming ($B[\sigma]$) is restricted to action prefixes.

It diminishes flexibility, but does not limit expressiveness of calculus excessively. In some cases instead of using processes as templates one will have to define several processes with different trajectory qualifiers.

A possible alternative solution is to use a syntactic renaming, like macro commands that substitute parameterisable parts of processes before simulation. Besides, our main intention behind renaming of qualifiers is the same, i.e., to give opportunity to use processes definitions as templates.

The original BHPC language is rich with diverse mathematical symbols. A set of characters and signs from Greek and Latin alphabets augmented with mathematical symbols is used to define processes. Therefore, tools dealing with the calculus should provide necessary means to handle such collections of symbols. A special editor is necessary to support it. However, a traditional approach in such a case is to design an alternative ASCII³ based notation and provide tools that can translate it to the original language using, e.g., \LaTeX ⁴.

There are at least two common practices in designing a new (an alternative) language.

1. An alternative version of the original language can be designed. It is just an ASCII notation for the original language, and usually is more useful as a language for a tool that is built as a *proof of concept*, an educational tool or an experimental tool, used for teaching or the further formalism development, respectively.
2. An extended version of the language with additional constructs and structures can be devised. It is the language for a tool that is aimed at the bigger case studies or use in industry.

Because our objective is to design the basic algorithms for BHPC simulation, we choose the first option. See van Putten [2006] for a detailed description of the ASCII version of BHPC.

7.2.2 Simulation of process algebras

Essentially, simulation of process algebras is based on the correct application of *structural operational semantics* (SOS) rules [Plotkin, 1981]. Operational semantics provide

³For more information about ASCII consult <http://en.wikipedia.org/wiki/ASCII>.

⁴See <http://www.latex-project.org/>.

information necessary for a stepwise execution of process algebraic expressions. For every process algebraic operator rules define all allowed executions. However, not all rules can be applied straightforwardly.

Parallel composition operator requires special treatment. Usually an *expansion law* (see Section 5.6 for the BHPC version) or *linearization* [Usenko, 2002] are exploited to deal with it.

Informally, *linearization* is a procedure of transforming a process algebraic expression into an equivalent system of *linear process equations*, i.e., a process algebraic expression containing only basic process algebraic operators (action prefix, alternative composition) and a special form of recursion [Usenko, 2002].

The expansion law expresses the parallel composition of the choice of processes as a choice of the processes by pushing the parallel composition one step further.

In BHPC case application of the rules is almost always straightforward, with exception of parallel composition. It is transformed according to the modified expansion law, and then the choice rule is applied.

However, certain complications emerge from the specifics of the expansion law. To separate an initial part of trajectory prefix an information about evolution of the trajectories is needed, and usually it is not available. Thereby we adapt the expansion law in such a way that trajectory prefixes part is not resolved, i.e., all possible combinations (or parallel compositions) of processes starting with trajectory prefixes are created. Then, a chosen combination is simulated until evolution conditions are satisfied or another choice (stop simulation of the chosen combination) presents itself.

Some of techniques for process algebras simulation are described in van Eijk [1988], Eertink [1994].

7.2.3 Abstract simulation algorithm for BHPC

The BHPC simulation algorithm is based on the event tracking approach (discussed in Section 7.1.2). It is tailored to specifics of process algebraic simulation, in particular Behavioural Hybrid Process Calculus (Chapter 5).

Remark 7.2.1. Some functions used in this chapter are used to abstract from data types and solvers. Consequently, pseudo code is provided only for the functions that define main concepts of BHPC simulation. However, all used functions are listed and described in Appendix C. \square

In the algorithm we abstract from the implementation issues, because different approaches can be advantageous in different development frameworks. An abstract simulation procedure (Algorithm 7.1) follows these steps.

1. The system is initialised.
2. While a current simulation time is less than the maximal simulation time and no other stop conditions have occurred, a transition is taken and the state is updated. When the simulation time has finished or other stop condition has occurred, it is halted.

Moreover, we assume that complete specification is globally accessible. The essential part of simulation is hidden in `TakeTransition` (Algorithm 7.2).

```

types
  record STATE
    t : TIME;
    lQ: LIST OF SETS OF QUALIFIERS;
    p : PROCESS;
    \* a process expression and pointers to the corresponding qualifiers sets *\
  end
  record STATUS
    continue: BOOLEAN;
    msg      : TEXT;
  end
end

algorithm BHPC_Simulation (SimTime:TIME; Q0:QUALIFIERS; p0:PROCESS);
var
  status: STATUS;
  state : STATE;
begin
  (state, status) := Initialise(p0,Q0);
  while (state.t < SimTime and status.continue) do
    (state,status) := TakeTransition(state, status, SimTime);
  end
  print(status.msg);
end

```

Algorithm 7.1: Simulation of BHPC

- The process expression is transformed to so-called *normal form* (see Section 7.2.4 for an explanation and a transformation algorithm).
- A non-deterministic choice is made (see Section 7.3 for different ways to treat non-determinism) to proceed with discrete or continuous transitions.
 - If the discrete course is chosen, a discrete transition is selected (Algorithm TakeDiscreteTransition).
 - If the continuous course is chosen, a continuous transition is selected (Algorithm TakeContinuousTransition).

The initialisation step is defined in Algorithm 7.3. It is a simple procedure, i.e., initial values and process are assigned to the corresponding state values and the simulation time is initialised.

Remark 7.2.2. It may be necessary to store values of qualifiers derivatives depending on the chosen type of continuous behaviour and solver. In such a case sets of qualifiers should be extended accordingly to store all important information. Moreover, initialisation procedure should take it into account. \square

```

algorithm TakeTransition(state:STATE ; status:STATUS ; T:TIME);
begin
  /* state : t – time, Q – values of qualifiers, p – process */
  (state, status) := transToNF( state, status);
  if ( status.cont and !isEmpty(state.p))
    choice := NondetChoose(state.p);
    case typeOfChoice(choice)
      continuous:
        (state, status) := TakeContinuousTransition(state, status, choice, T);
      discrete:
        (state, status) := TakeDiscreteTransition(state, status, choice);
    end
  end
  return ( state, status);
end

```

Algorithm 7.2: TakeTransition

```

algorithm Initialise(p0:PROCESS; Q0:QUALIFIERS);
var
  status : STATUS;
  state : STATE;
begin
  /* state : t – time, Q – values of qualifiers, p – process */
  status.continue := true;
  status.msg := "Successful termination";
  state.t := 0;
  state.p := p0;
  InitialiseQualifiers( state.lQ, Q0);
  return( state, status);
end

```

Algorithm 7.3: Initialise

7.2.4 Transformation to normal form

In Algorithm 7.2 before taking a transition a process algebraic expression is translated to BHPC *normal form*. Such an approach makes simulation of parallel composition easier.

Definition 7.2.3 (BHPC normal form). We will call a BHPC *normal form* a process algebraic expression of the following form:

$$\sum_{i \in I} b_i \cdot B_i + \sum_{\substack{j_1 \in J_1 \\ \dots \\ j_m \in J_m}} \left(\dots \left([f_{j_1} \mid \Phi_1] \cdot B_{j_1} \parallel_{A_1}^{H_1} [f_{j_2} \mid \Phi_2] \cdot B_{j_2} \right) \parallel_{A_2}^{H_2} \dots \parallel_{A_{m-1}}^{H_{m-1}} [f_{j_m} \mid \Phi_m] \cdot B_{j_m} \right)$$

```

algorithm transfToNF (state :STATE, status :STATUS)
var state', state'': STATE;
begin
  if (!isNormForm(state))
  case (state.p)
    0: state := setDeadlock(state);
    B' [σ]:
      state' := setState(B', state);
      if !isNormForm(state') then (state', status) := transfToNF(state', status); end
      state := rename(state', σ);
  new w.B':
    state' := setState(B', state);
    if !isNormForm(state') then (state', status) := transfToNF(state', status); end
    state := hide(state', status);
  ⟨Pred⟩.B':
    if !Evaluate(Pred)
    then state.p := setDeadlock(state);
    else
      if !isNormForm(state)
      then (state', status) := transfToNF(state, status); end
      state := setProcess(B', state, state');
    end
  ∑i∈I Bi:
    forall i ∈ I do
      state' := setState(Bi, state);
      if (!isNormForm(state'))
      then (state', status) := transfToNF(state', status); end
      state := replaceChoiceComp(state, state', i);
    end
  B1 ||AH B2:
    state' := setState(B1, state); state'' := setState(B2, state);
    if !isNormForm(state') then (state', status) := transfToNF(state', status); end
    if !isNormForm(state'') then (state'', status) := transfToNF(state'', status); end
    (status, state) := applyExpLaw(state', state'', A, H, status);
  P:
    (status, state) := resolveRecursion(state, status);
    if !isNormForm(state) then (state, status) := transfToNF(state, status); end
  end
  return (state, status);
end

```

Algorithm 7.4: Transformation to normal form

7. SIMULATION OF BEHAVIOURAL HYBRID PROCESS CALCULUS

Moreover, parallel compositions of trajectory prefixes can be enclosed by renaming, i.e., can have a form

$$\sum_{\substack{j_1 \in J_1 \\ \dots \\ j_m \in J_m}} \left(\dots \left(\left([f_{j_1} \mid \Phi_1] . B_{j_1} \parallel_{A_1}^{H_1} [f_{j_2} \mid \Phi_2] . B_{j_2} \right) [\sigma] \parallel_{A_2}^{H_2} \dots \parallel_{A_{m-1}}^{H_{m-1}} [f_{j_m} \mid \Phi_m] . B_{j_m} \right) [\sigma'] \dots \right) [\sigma'']$$

□

Remark 7.2.4. Notice that usually normal form refers to an expression of form $\sum_{i \in I} \mathbf{b}_i . B_i + \sum_{j \in J} [f_j \mid \Phi_j] . B_j$. However, in the BHPC case to transform any process expression to such a form *a priori* information about the trajectories is necessary (see Section 7.2.2). Another approach is simulation of separate trajectory prefixes, but that is not feasible. Therefore we choose the above defined form as easiest way to simulate BHPC. □

We use normal form for two main reasons.

- It provides separated discrete and continuous behaviours in the form convenient for simulation.
- The modified expansion law and other procedures allow to transform almost any process algebraic expression to such form in a finite number of steps. We provide an explanation of the pathological cases and the modified expansion law below.

To transform a parallel composition to a normal form an expansion law is used. Unfortunately, as was mentioned in Section 7.2.2, application of the expansion law (Theorem 5.6.3) is not trivial because it needs an oracle. To solve this problem we use the normal form and present a modified version of expansion law. Basically, it is just an intermediate form from the proof the expansion law (Theorem 5.6.3).

Theorem 7.2.5 (Modified Expansion Law). *Let*

$$B = \sum_{i \in I} \mathbf{b}_i . B_i + \sum_{j \in J} [f_j \mid \Phi_j] . B_j, \quad C = \sum_{k \in K} \mathbf{c}_k . C_k + \sum_{l \in L} [g_l \mid \Psi_l] . C_l$$

for some process B_i, B_j, C_k and C_l , actions \mathbf{b}_i and \mathbf{c}_k , trajectories-prefixes φ_j and ψ_l , index sets $I \cap J = K \cap L = \emptyset$. Then

$$B \parallel_A^H C = \sum_{\substack{i \in I \\ \mathbf{b}_i \notin A}} \mathbf{b}_i . (B_i \parallel_A^H C) + \sum_{\substack{k \in K \\ \mathbf{c}_k \notin A}} \mathbf{c}_k . (B \parallel_A^H C_k) + \sum_{\mathbf{a} = \mathbf{b}_i = \mathbf{c}_k \in A} \mathbf{a} . (B_i \parallel_A^H C_k) + \quad (7.1a)$$

$$\sum_{\substack{j \in J \\ l \in L}} \left([f_j \mid \Phi_j] . B_j \parallel_A^H [g_l \mid \Psi_l] . C_l \right) \quad (7.1b)$$

Notice, that this version of expansion law does not resolve the parallelism of the continuous part. It resolves discrete behaviour and collects all trajectory prefixes together while preserving the order of the parallel composition.

Proof. The proof of Theorem 7.2.5 follows directly from the expansion law (Theorem 5.6.3) and its proof.

Expression (7.1a) defines the expansion law for the discrete components. A nice explanation and detailed proof of the expansion law for discrete actions are given in Milner [1989, p.96–97].

Expansion of continuous components is defined by (7.1b). How to achieve such syntactical form is shown in the proof of the original expansion law (Theorem 5.6.3) (moving choice outside of parallel composition). \square

We present Algorithm 7.4 that transforms almost any process algebraic expression to a normal form, and afterwards elaborate on the potential problems and cases, when transformation is not possible. Unfortunately, the termination of `transfToNF` is not guaranteed in general. The problem stems from the specifics of recursion.

Recursion In general, recursion just tells that simulation has to proceed with another process expression. The switching from one expression to another includes initialisation of the new process expression.

However, processes like $B \triangleq a.B+B$ can be encountered. Such processes (equations) are called *not guarded* [Milner, 1989, p.65] and there is no nice theoretical escape from such situation. Of course, such processes can be disallowed, i.e., a compiler can perform a static check and ask to modify not guarded processes.

The task can be handed over to a simulation engine that, after repeatedly trying to resolve recursion for n times (with n chosen by a user), can ask the user for a help, e.g., to choose another branch of execution.

Invocation of this algorithm can be augmented with a recursion counter that can be used to carry information about the depth of recursion. If recursion is not resolved until a certain depth, then the normalisation procedure can be stopped and a status variable used to carry information about the failure.

In Algorithm 7.4 we do not explicitly use the above defined technique to detect not guarded recursion.

Renaming Recall that we use only renaming of actions. If a process is in a form $a.B$, then SOS rules for renaming are applied straightforwardly. Otherwise the process is normalised and then renaming is applied to a normal form. However, because we do not resolve parallel composition of trajectory prefixes, renaming becomes somehow more complicated.

We define the following rule for renaming of normal form. Let

$$B = \sum_{i \in I} b_i \cdot B_i + \sum_{j \in J} [f_j \mid \Phi_j] \cdot B_j, \quad C = \sum_{k \in K} c_k \cdot C_k + \sum_{l \in L} [g_l \mid \Psi_l] \cdot C_l$$

Then

$$\begin{aligned} (B \parallel_A^H C)[\sigma] = & \\ & \left(\sum_{\substack{j \in I \\ b_j \neq A}} b_j \cdot (B_j \parallel_A^H C) \right) + \sum_{\substack{k \in K \\ c_k \neq A}} c_k \cdot (B \parallel_A^H C_k) + \sum_{a=b_i=c_k \in A} a \cdot (B_i \parallel_A^H C_k) + \end{aligned}$$

$$\begin{aligned}
& \sum_{\substack{j \in J \\ l \in L}} \left([f_j \mid \Phi_j] . B_j \parallel_A^H [g_l \mid \Psi_l] . C_l \right) [\sigma] = \\
& \sum_{\substack{i \in I \\ b_i \notin A}} \sigma(b_i) . (B_i \parallel_A^H C) + \sum_{\substack{k \in K \\ c_k \notin A}} \sigma(c_k) . (B \parallel_A^H C_k) [\sigma] + \sum_{a=b_i=c_k \in A} \sigma(a) . (B_i \parallel_A^H C_k) [\sigma] + \\
& \sum_{\substack{j \in J \\ l \in L}} \left([f_j \mid \Phi_j] . B_j \parallel_A^H [g_l \mid \Psi_l] . C_l \right) [\sigma]
\end{aligned}$$

Hiding Almost the same as renaming, only selected actions are renamed to τ and it is not applied for trajectory prefixes (trajectory qualifiers).

Guard If a guard is true, then we get the succeeding process, otherwise we get $\mathbf{0}$.

Choice Every component of choice is transformed to a normal form and transitions are grouped.

Parallel composition Every component of parallel composition is normalised and then the modified expansion law is applied.

In `BHAVE` prototype we use a simplified treatment of processes parametrisation. We use parameters as shortcuts to initialise qualifiers. Moreover, we access the last simulation value using the corresponding qualifier identifier. See D.1.1 for the details.

7.2.5 Simulating discrete events

`TakeDiscreteTransition` takes care of discrete transitions in simulation of BHPC. It performs the chosen action prefix, calculates new process expression, updates qualifiers values if necessary and returns state.

In general, *simulation of discrete events* (sometimes called *event iteration*⁵) includes several steps and vary for different approaches. The general procedure is the following.

1. A set of actions becomes available for execution. An action is chosen and executed. Depending on the formalism the system may choose to proceed with continuous behaviour or
 - If it is one process, an action is chosen and executed;
 - In approaches, that do not use expansion law to flatten parallel composition, the action is selected according to the semantics of formalism.
2. Executing the action (defined in the previous step) may include:
 - Recalculation of state;
 - Activation of other events.
3. The system returns to the first step.

⁵Not to be mixed with event-tracking.

Eertink [1994], van Eijk [1988] discuss techniques for synchronisation of ordinary and parametrised action prefixes.

Synchronisation of actions for Omola is discussed in Andersson [1994], for Hybrid χ in Fabian [1999].

Remark 7.2.6 (Simulating discrete events and discrete simulation). Here we use a notion of *simulating discrete events* to refer to a part of the hybrid systems simulation process dealing with discrete events. However, resembling notions of *discrete simulation* or *discrete event simulation* have several different meanings. These notions may refer to the use of discrete time in the simulation process or an abstraction from continuous-time behaviour and simulation of discrete behaviour only. \square

7.2.6 Simulating continuous-time behaviour

Roughly, simulation of continuous-time behaviour in hybrid systems is simulation of continuous systems (Section 7.1.1) with events detection. It consists of one or more steps, i.e., the behaviour to be simulated is chosen (sometimes non-deterministically) and then simulated for a specified time or until a specified event(s) occurs. Different ODE/DAE solvers can be used, the integration step size may be adjusted.

In BHPC simulation Algorithm 7.5 takes care of the simulation of continuous-time behaviour. The modified expansion law provides us with a set of parallel compositions. Therefore, we adopt the following simulation procedure.

- All possible combinations of parallel trajectory prefixes are created.
- Combination is chosen and simulated while monitoring the progression of simulation. In case of predefined events corresponding actions are taken.

Combining parallel trajectory prefixes The normal form (Definition 7.2.3) provides the following process algebraic expression

$$\sum_{\substack{j_1 \in J_1 \\ \dots \\ j_m \in J_m}} \left(\dots \left([f_{j_1} \mid \Phi_{j_1}] . B_{j_1} \parallel_{A_1}^{H_1} [f_{j_2} \mid \Phi_{j_2}] . B_{j_2} \right) \parallel_{A_2}^{H_2} \dots \parallel_{A_{m-1}}^{H_{m-1}} [f_{j_m} \mid \Phi_{j_m}] . B_{j_m} \right)$$

possibly with renaming operators.

Then we can collect all possible trajectory prefixes combinations while preserving order of parallel compositions and renaming operators.

$$\begin{aligned} & \left([f_{j_{1,1}} \mid \Phi_{j_{1,1}}] , \quad [f_{j_{2,1}} \mid \Phi_{j_{2,1}}] , \quad \dots , \quad [f_{j_{m,1}} \mid \Phi_{j_{m,1}}] \right) \\ & \left([f_{j_{1,2}} \mid \Phi_{j_{1,2}}] , \quad [f_{j_{2,2}} \mid \Phi_{j_{2,2}}] , \quad \dots , \quad [f_{j_{m,2}} \mid \Phi_{j_{m,2}}] \right) \\ & \dots \quad \dots \quad \dots \quad \dots \\ & \left([f_{j_{1,|j_1|}} \mid \Phi_{j_{1,|j_1|}}] , \quad [f_{j_{2,|j_2|}} \mid \Phi_{j_{2,|j_2|}}] , \quad \dots , \quad [f_{j_{m,|j_m|}} \mid \Phi_{j_{m,|j_m|}}] \right) \end{aligned}$$

We get $K = |J_1| \cdot |J_2| \cdot \dots \cdot |J_m|$ combinations. Trajectory prefixes in every combination are composed preserving the order and renaming. Notice, that renaming is not important for trajectory prefixes, but should be preserved for the succeeding processes. Therefore,

some data structures must be used to store this information. The combination is chosen non-deterministically and simulation is started. The simulation is monitored for events, and the following actions are taken.

- In the case of deadlock the simulation is stopped, the status and state variables are updated accordingly.
- In the case, when the maximal simulation time is expired, the simulation is stopped, and the status and state variables are updated.
- If the evolution conditions are to be violated and it is not possible to exit, then the simulation is stopped and the corresponding variables are updated. If it is possible to exit, then the violation event is changed to the must-exit event (inside Algorithm `Solve`).
- If the can-exit event is generated, then the continuous simulation is stopped, the state and status are updated.

The abstract algorithm is presented in Algorithm 7.5. Different versions of the algorithm may be defined depending on how non-deterministic choice to exit or continue is implemented. E.g., in this version for simplicity reasons we do not postpone an exit, i.e., if it is possible to exit, we stop continuous simulation, update the state and allow to choose how to continue. Nevertheless, we believe, that the presented version conceives the essence of the proposed method. Moreover, our experience from the implementation of `BHAVE` prototype shows that a structure of algorithm often depends on the chosen data structures, programming language and other design/implementation choices.

We concisely describe some of the main functions used by the algorithm.

- All combinations of trajectory prefixes are generated by Algorithm `CreateCombinations`.
- Algorithm `NondetChooseCombination` nondeterministically chooses a combination from a set of combinations.
- The set of events is generated by Algorithm `DefineEvents`. It includes the events necessary to properly treat non-deterministic exits of trajectories. In trajectory prefix a set of trajectories can be represented by an ODE (or DAE) together with the corresponding exit conditions. It induces a continuous choice between continuations and exiting. In Section 7.3 we discuss possible ways to make such choice.
- Combination is simulated in Algorithm `Solve`. It takes a set of trajectory prefixes and halting conditions, extracts equations, initial conditions and halting conditions, and supplies it to a numerical solver. The algorithm is data types and numerical solver dependent, i.e., different solvers may have different syntactic and semantic limitations. In some cases, the procedure can be complicated technically, e.g., when a solver works as a library, then it may be necessary to compile it together with equations.

```

algorithm TakeContinuousTransition(state:STATE,
    status:STATUS, choice:CHOICE, T:TIME)
var cs : COMBINATION OF TRPREFIXES;
    state ' : STATE;
    comb : COMBINATION;
begin
    cs := CreateCombinations( state, choice);
    comb := NondetChooseCombination(cs);
    events := DefineEvents( state, comb);
    (event, state ', status, cs) := Solve(T, state, cs, events);
    ( state, status) = UpdateSimState(event, state ', status, comb);
    return ( state, status);
end

```

Algorithm 7.5: TakeContinuousTransition

- Algorithm UpdateSimState updates process expression in the state variable by replacing it with the new process that is generated from the combination and event that has stopped continuous simulation.
 - If an event that completely stops simulation is generated then the process expression is not changed.
 - If the event is must-exit event then the corresponding expression(s) of the form $[f \mid \Phi].B$ is (are) replaced by an expression(s) B .
 - If the event is can-exit event then the corresponding expression(s) of the form $[f \mid \Phi].B$ is (are) replaced by an expression(s) $[f \mid \Phi].B + B$.

Some functions appearing in Algorithm 7.5 are not explained here. These and other functions used in this chapter are listed and described in Appendix C, Bhave prot and <http://www.cs.utwente.nl/tools/bhave>.

Remark 7.2.7. In BHAVE prototype we use a simplified algorithm, where only certain types of events and continuous behaviour are supported. Explanation of the approach used in BHAVE prototype is provided in Appendix D. \square

Simulating trajectory prefixes

In BHPC continuous behaviour is specified by the sets of trajectories. The sets of trajectories can be represented in different ways, which conceptually are equivalent, but during simulation should be treated differently.

- A trajectory is represented in such a way that its value at any time in the interval can be easily computed (e.g., explicit functions).
- Trajectories are represented by implicit functions (ODE/DAE) that should be solved in some special way, usually using numerical algorithm, to get the values at a certain time moment. In fact, simulation is one of the ways to solve them.

Simulation of trajectories represented by explicit functions is almost trivial. Values at any time can be easily calculated, event time can be estimated exactly (up to software/hardware supported error bounds).

However, usually trajectories are represented by implicit functions (ODE/DAE). Moreover, ODE/DAE are standard way to represent continuous behaviour in the control theory. In such case ODE/DAE solvers are employed. In `BHAVE` prototype we use Open Maple, that is a suite of functions that allows to access `MAPLE` algorithms and data structures in compiled C or C++ program. It imposes certain limitations, e.g., solver with stop conditions works only for ODE. At the same time, it provides enough functionality for a prototype.

We provide some information about other ODE/DAE solvers below.

Consistent state (re-)initialisation The problem of *consistent state (re)initialisation* in the simulation of Behavioural Hybrid Process Calculus emerges, when the processes in parallel composition attempt to modify qualifiers values (e.g., trajectory prefixes). Moreover, it is aggravated by the numerical and real number representation in computer errors. In some cases, e.g., when the equivalent mathematical operations are carried out in a different order, resulting values may slightly differ leading to an inconsistency. That is an important issue, and more robust simulation techniques seem to be in demand.

Remark 7.2.8 (Complex evolutions). In some cases, e.g., for cyclic evolutions like the sine function, just initial values may be insufficient. In such a case values of derivatives at initial time can be used. \square

In general, the problem of consistent state (re)initialisation is usually related with event detection and discrete event simulation (Section 7.2.5) problems, because most of the time the exact location of an event and recalculation of the (continuous) state influence the initial values for the next simulation step. A nice reference for some of such problems is Brenan et al. [1991].

ODE/DAE solvers ODE/DAE solvers are the main engine in the simulation of continuous systems. There is a plentitude of solvers that use different algorithms, support various types of equations (ODE/DAE), etc.

Typically such solvers use specialised numerical algorithms that give good approximations of the equations being solved. Advanced solvers often use dynamical step adjustment.

As was mentioned above, in `BHAVE` prototype we use `MAPLE` to solve ODE. It imposes some functionality and performance limitations, but is sufficient for a prototype. However, for industrial strength tool it may be beneficial to explore other ODE/DAE solvers. Here we provide references to some of them.

A theoretical background and an implementation (DASSL) are presented in Brenan et al. [1991]. In Lee and Zheng [2005] *linear multi-step* methods (LMS), *Runge-Kutta* methods and the first order RK method (also called *forward Euler*) are presented and an application of RK2-3 ODE solver in HyVisual (Section 7.9) is explained. An introductory explanation of simulation of continuous and hybrid systems, and application of various solvers are given in Taylor [1999]. Concise explanation of use of solvers in

Ptolemy (Section 7.9) is presented in Liu et al. [1999]. Otter and Cellier [1995] discusses problems related with higher index models.

Fabian [1999, p.111–122] discusses application of DASSL [Brenan et al., 1991] package in Hybrid χ (Section 7.9) simulator.

The development of the methods and tools to solve ODE/DAE is a very interesting topic. However, in the context of simulation of hybrid systems the choice of the right algorithms is more important than the development of such algorithms. Therefore it is important to incorporate different solvers into a simulator and in such a way to provide an opportunity to experiment with them.

7.3 Non-determinism

Non-determinism in discrete systems In computer science *non-determinism* was explicitly introduced by Rabin and Scott [1959], where a *non-deterministic automaton* is defined as a machine with many choices in its moves. Therefore, by exhibiting the same observable behaviour the system has a liberty to choose a target state from one of several new states, whereas in deterministic system the target state is uniquely defined by the source and the action.

Non-determinism in continuous systems A continuous system is called deterministic if an input and a current state uniquely define an output and a new state. Often it is related to the so-called *well-posedness* property (e.g., De Schutter and Heemels [2004, p.35]). Usually the property is required to hold.

Non-determinism in hybrid systems Usually a system is called *deterministic* if its evolution is single valued w.r.t. the states and the inputs, i.e., the states and inputs uniquely determine the target state and the output. The system is called *non-deterministic*, if the same input potentially leads to one of the several new states. In the hybrid systems case such a definition is too vague, i.e., non-determinism can occur in different parts of the systems, and it depends on the formalism. It occurs in the continuous and discrete parts of system, as well as in interaction between these two types of behaviours. Non-determinism arises from various causes, e.g., high abstraction of the model, incomplete knowledge about the system, or is introduced by parallel composition.

- *Non-determinism of actions* is a phenomenon that occurs when a target state is not defined uniquely by the source state and the action. An example of such behaviour is a coffee machine, such that just by throwing in a coin you may get coffee or tea non-deterministically. Such type of non-determinism in computer science is often related with so-called *silent actions* [Milner, 1989, p.37–43] that can be used to hide actions from the observers.
- Non-determinism of continuous systems may take different form in hybrid systems. In the BHPC's trajectory prefix a set of trajectories can be represented by differential equations together with the corresponding exit conditions. It induces a continuous choice between continuations and exiting.

One of the possible approaches that would work with simple dynamics, is to choose non-deterministically a solution in this interval as an exit point (or the point, where, again, the choice is made non-deterministically to continue with the trajectory prefix, or to exit and to continue with the following process). We employ this technique in BHAVE prototype because it is a prototype tool and should handle only simple dynamics.

Yet another solution for this problem is to detect the moment when exit conditions are enabled and then simulate in small steps choosing non-deterministically to continue or to exit at every step, while the conditions are enabled. However such a technique is very time consuming and is not suitable for performance sensitive simulations.

- *Non-determinism of switching time* is a phenomenon usually present in hybrid automata (Section 3.3.6). Hybrid automata may take a discrete transition whenever the guard expression evaluates to true. Moreover, in hybrid automata a guard often evaluates to true in an interval, and consequently, the transition can be taken non-deterministically in that interval. In the case of BHPC it corresponds to non-determinism of continuous-time behaviour, because it is a choice between different trajectories.

In the different formalisms the manifestation of non-determinism may vary, and diverse solutions are used to deal with it. We list some of them.

- In some cases non-determinism is treated as an under-specification of the system (e.g., in Modelica™ [Fritzson and Engelson, 1998, Mod, 2005]). In piecewise affine systems (Section 3.3.2) and mixed logical dynamical systems (Section 3.3.3) a well-posedness requirement is assumed [De Schutter and Heemels, 2004, p.35]. It requires deterministic solutions of ODE/DAE. Therefore, non-determinism is treated as an error and is not allowed.
- Non-determinism is resolved by *the tool (translator, compiler, etc.)*. In other words, it is left to be resolved by the implementation of a particular tool. It may appear as a nice solution for the non-determinism, e.g., if the system consists of two devices loaded non-deterministically, only one device can be loaded all the time and in such a way it may expose the potential problem. At the same time it causes some problems, e.g., if the same execution is taken all the time, all other executions are skipped. Moreover, the choice of the execution usually depends on the physical layout of the specification, and then the choices a or b ($a + b$) and b or a ($b + a$) can be resolved in a different way causing hard to detect problems. This technique is adopted in the Hybrid χ simulator [Fabian, 1999, Hofkamp, 2001] (Section 7.9).
- *Statistical methods* can be used to resolve the choice, i.e., the choice is associated with some distribution and is resolved respectively. Usually a uniform distribution is used, but other distributions can be employed too. Such an approach allows to explore different executions, but it causes some problems, illustrated by an example, where the load is distributed between two parties, i.e., uniform distribution divides the load evenly, and it is not necessary the case in the real system. Besides, adding distributions on choices changes the semantics of the

model, because non-determinism is not the uniform or any other distribution, and in such a case *hybrid-stochastic* approaches (like Strubbe et al. [2003]) seem more appropriate.

- Nondeterminism can be resolved by the user. Such an approach is one of the most straightforward ways to implement non-determinism. However it would require constant user interaction, and usually that is not convenient. A generalised version of such an approach is to use a *scheduler* [Kaynar et al., 2002]. The scheduler collects information about the choices and chooses according to the collected information and the scheduling algorithm. Unfortunately, it generates significant overhead for the simulator, and if some particular scheduling is known, then it rather should be a part of the specification, than the simulator. However, such an approach provides a maximal control over the simulation process (w.r.t. choices) and allows to explore different simulation scenarios.

All listed approaches have their strengths and weaknesses. A suitable solution for this problem could be a simulation tool that allows a modeller to choose the appropriate method to resolve the choice from the following list.

- Non-determinism is not allowed.
- The choice is resolved by the tool.
- The choice is resolved using some distribution (usually uniform).
- The choice is resolved by the scheduler (individual schedulers can be assigned to the resolution points).

In *BHAVE* prototype to resolve non-determinism we adopt several of the discussed techniques that, provide the best flexibility and are not too complex for a prototype implementation. Therefore, the choice amongst action prefixes and trajectory prefixes can be made either by user, or user may ask the tool to choose. In the second case the tool makes choice based on the internal ordering. Moreover, the choice based on internal ordering can be easily extended to use a random generator. And the choice to exit or continue simulation of trajectory prefix is made by the tool.

7.4 Visualisation of models

Visualisation of technical problems dates back to da Vinci's technical drawings, Egyptian, Babylonian and Greek geometry, if not cave paintings. Nowadays visualisation of technical problems is a standard practice in academia and industry. Modelling of hybrid systems is no exception. Visual representation as well as textual, are frequently used to represent such systems. Moreover, some formalisms *per se* are visual, and are augmented by the corresponding textual languages for convenience.

In this stage of development we do not propose any visualisation approach for Behavioural Hybrid Process Calculus models. However, we believe, that trees, directed graphs or automata like structures can be used to represent processes. Moreover, we believe that in future *object diagrams* (accordingly modified) or *block diagrams* can be used to represent BHPC processes.

We discuss some of the most popular hybrid systems models visualisation techniques to illustrate potential choices for BHPC.

Top and Akkermans [1994] separates several representation layers.

- In the *technical (functional)* layer the technical design representing main parts of the system is defined. *Object diagrams* (explained below) are suitable for this level.
- The *physical* layer is used to define the actual physical processes of a system. *Bond graphs* (Section 3.3.12) can be used in this layer.
- In the *mathematical* layer some mathematical formalism is used to describe a mathematical structure of system. ODE/DAE are appropriate at this level.

A number of visualisation techniques for hybrid systems are known. Some of them are discussed in Otter and Cellier [1995]. We will survey some of these techniques.

- *Block diagrams* [Hamon and Rushby, 2004, Stateflow] is one of the prevalent visualisation methods. A system is divided into blocks that receive an input, transform it according to some rules (equations) and output it. Blocks can have memory, be constituents of other blocks and be composed of interconnected blocks. SIMULINK (Section 7.9) is based on the block diagrams modelling paradigm.
- *Object diagrams* represent an object oriented approach to modelling of dynamic systems. In contrast to block diagrams, physical objects are represented by mnemonically shaped icons. Objects can be interconnected, consist or be integral parts of other objects. Considerable advantages of this approach are *encapsulation* and *inheritance*. Informally, encapsulation provides a means to hide internal object details from the outside world. Inheritance allows the specific objects to inherit properties of generic ones. Object diagrams are gaining a lot of popularity lately, and are used in tools, e.g., Modelica™ (Section 18).

Jovanovic et al. [2004] proposes a *tree view* representation for CSP [Hoare, 1985] based hybrid formalism. It is convenient for browsing a model, but too cumbersome for development [Jovanovic et al., 2004]. Thus, block diagrams are used in the development process.

Samarin [2002] presents an example of purely visual simulation environment for physical processes.

7.5 Visualisation of results

Simulation results usually represent the evolution of the system in time. At the ground level it is represented by a mapping of state values and events to the time-line.

However, often only state values are mapped to the time line. In such a case *graphs (plots)* are used to visualise simulation results. It corresponds to a combination of the mathematical and physical layers in the Top and Akkermans [1994] terminology.

At the functional layer behaviour of the objects constituting the system can be displayed as a 2D or 3D animation.

Event traces or message sequence charts [Rudolph et al., 1996, ITU-T, 2000] adequately represent behaviour of discrete systems. Graphs are adequate for the ordinary continuous system. However, for hybrid systems a combined view is crucial. Several approaches are proposed in Samarin [2002], Lee and Zheng [2005], Hedlund [1999, p.74], but none of them are satisfactory.

We will investigate principal visualisation techniques of dynamical systems and propose a combined view in the forthcoming sections.

7.5.1 Graphs

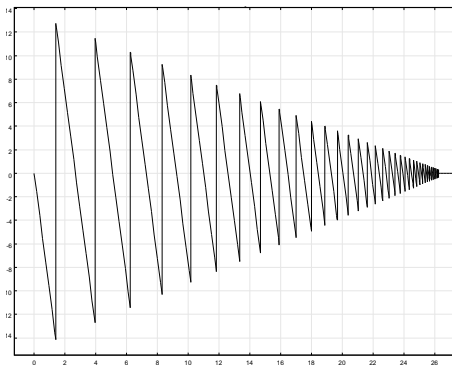


Figure 7.1: Velocity v

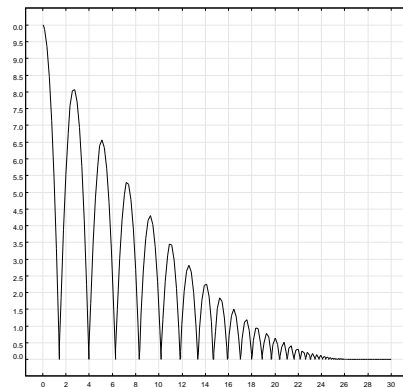


Figure 7.2: Altitude h

Ordinary *graphs (plots)* are the simplest way to visualise evolution of the system. Velocity and altitude in Figures 7.1 and 7.2 (from the example in Section 2.2.1), respectively, exemplify the use of graphs to depict evolution of system.

7.5.2 Event traces and message sequence charts

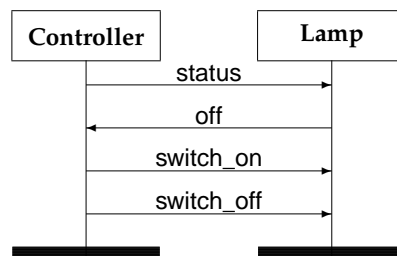


Figure 7.3: Example of Lamp and Controller simulation

Usually, simulations of discrete systems in computer science are presented by *event traces*, i.e., ordered sequences of events identifiers or *event-state traces*, i.e., ordered alternating sequences of events and states identifiers (or state valuations). If in the

discrete version of the example from Section 2.2.1 bouncing is denoted by `bounce`, then

- An example of an event trace is $\langle \text{bounce}, \text{bounce}, \text{bounce}, \dots \rangle$;
- An example of an event-state trace is

$$(h = 10, v = 0), \text{bounce}, (h = 0, v = -14.1), \text{bounce}, (h = 0, v = -12.8), \dots$$

Traces can be decorated with some additional information, if it is defined so in the formalism.

Remark 7.5.1 (Event traces and event-state traces). Often the notion of *event traces* is used to refer to *event-state traces*, because in some formalisms events are sufficient to uniquely determine states. \square

A more informative visualisation technique for discrete systems simulation is *message sequence charts* (MSC) [Rudolph et al., 1996, ITU-T, 2000]. In message sequence charts processes are represented by vertical lines, and horizontal lines (vectors) connecting the processes represent communication amongst them.

Example 7.5.2. Let us have a system consisting of a lamp and a controller that can request a state of the lamp (`status`), get an answer (`on`, `off`), and switch the lamp on (`switch_on`) and off (`switch_off`). An example of simulation of such system is presented in Figure 7.3. The controller starts by requesting the state of the lamp, receives an answer `off`, then switches it on and off.

Notice that in the example only message sequence charts like notation is used, because concrete design may depend on the formalism that employs it. \square

Remark 7.5.3 (MSC). A formal definition of the message sequence charts (MSC) is available in Rudolph et al. [1996], ITU-T [2000]. However, we just propose to adopt a similar notation style to visualise the discrete behaviour (according to the formalism, as in UPPAAL (Section 7.9)). \square

7.5.3 Combined view

Graphs and event traces or message sequence charts adequately represent continuous-time and discrete behaviour, respectively. However, hybrid systems combine both types of behaviour, therefore some combination of several representations is needed.

Usually only graphs are used to visualise continuous behaviour, and discrete events are visible as a change of behaviour. Results in BHPC can be visualised in an ordinary way, employing graphs and MSC. However, these techniques are not always sufficient to visualise hybrid phenomena.

We propose to use MSCplots or MSP (*message sequence plots*) [Schonenberg, 2006]. Message sequence plots is a combination of MSC [Rudolph et al., 1996, ITU-T, 2000] and plots that captures both discrete and continuous-time behaviour, and their interaction. MSP has two main compounds: message-sequence charts rotated 90° combined with plots. We do not provide a formal definition of message sequence plots, because we do not want to introduce a new formalism, but rather to propose an abstract technique for visualisation of hybrid behaviour. Therefore, we just illustrate MSP in Figure 7.4.

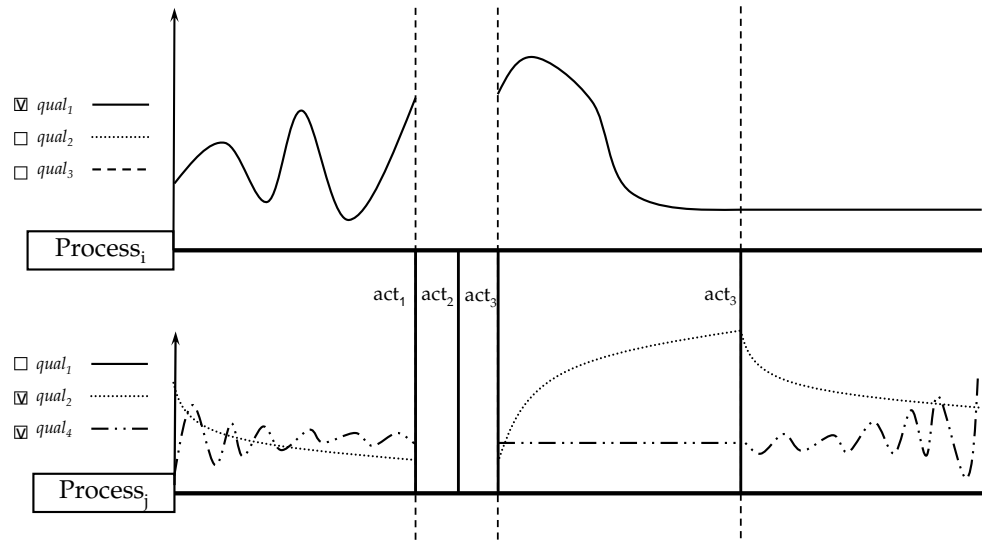


Figure 7.4: Visualisation of hybrid simulation

- Horizontal lines connected to the corresponding boxes with process identifiers represent processes and *time-line* (or *life-line* in MSC terminology). Time is assumed to flow to the right along each time-line at the same speed.

Figure 7.4 consists of Process_i and Process_j that are represented by horizontal lines connected to boxes with processes identifiers.

- Labelled vertical lines going across time-lines represent communication, i.e., action prefixes in BHPG case. Notice that we use simple lines instead of arrows, because communication in BHPG is not directed.

In Figure 7.4 communication between Process_i and Process_j consists of actions act_1 , act_2 and act_3 .

Continuous-time evolution is depicted by plots over time-lines. A legend is used to depict relation between plots and qualifiers. Qualifiers that interest user can be selected. If several processes evolve concurrently, the synchronising qualifiers appear for both processes.

In Figure 7.4 qualifiers $qual_1$, $qual_2$, $qual_3$ and $qual_4$ are depicted. Process_i is related with qualifiers $qual_1$, $qual_2$ and $qual_3$, and only qualifier $qual_1$ is selected to be visible. Process_j is related with qualifiers $qual_1$, $qual_2$ and $qual_4$, and qualifiers $qual_2$ and $qual_4$ are selected to be visible. At some moment actions act_1 , act_2 and act_3 interrupt continuous evolution. After communication continuous-time evolution goes on.

Folding and unfolding can be introduced to control visibility of components of parallel composition. In such a case communication between these processes should be depicted in some other way, e.g., as lines with action names that perpendicularly cross the process line.

7. SIMULATION OF BEHAVIOURAL HYBRID PROCESS CALCULUS

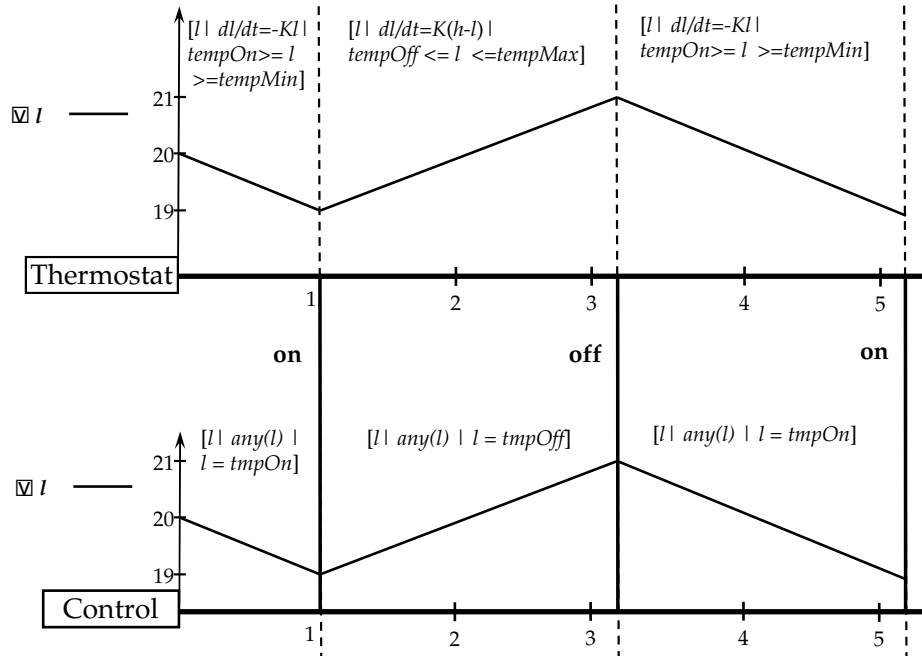


Figure 7.5: Upgraded Thermostat evolution in MSP

Moreover, a figure can be decorated with process expression, e.g., action- and trajectory prefixes at the corresponding parts of communication lines and graphs, respectively.

Recursive calls can be depicted as boxes with a new process identifier on the time-line.

We exemplify a potential use of such technique using an upgraded thermostat from Example 5.8.2. In an ordinary way the evolution of system would be depicted as in Figure D.4. While in Figure 7.5 we give an example of evolution depicted using MSP. It is easy to see that all information that is available in an ordinary plot (a plot of temperature changes) is available in MSP. Furthermore, in the message sequence plot both processes and communication between them are visualised. Therefore, causes of changes are visible too. Correspondingly, all information that is visible in message sequence charts is visible in MSP. Moreover, it is a lot easier to observe causes of communication, and vice versa.

The same evolution with explicit recursive calls is depicted in Figure 7.6. By comparing Figures D.4, 7.5 and 7.6 it is clear that the developer and user will have to choose between the amount of represented information and clarity. Therefore, the user should be allowed to choose what he wants to see, and be able to hide (fold) or expose (unfold) parts of MSP.

The proposed technique can be easily adopted to other hybrid systems modelling frameworks with minimal changes, e.g., to depict a directed communication arrows can be used.

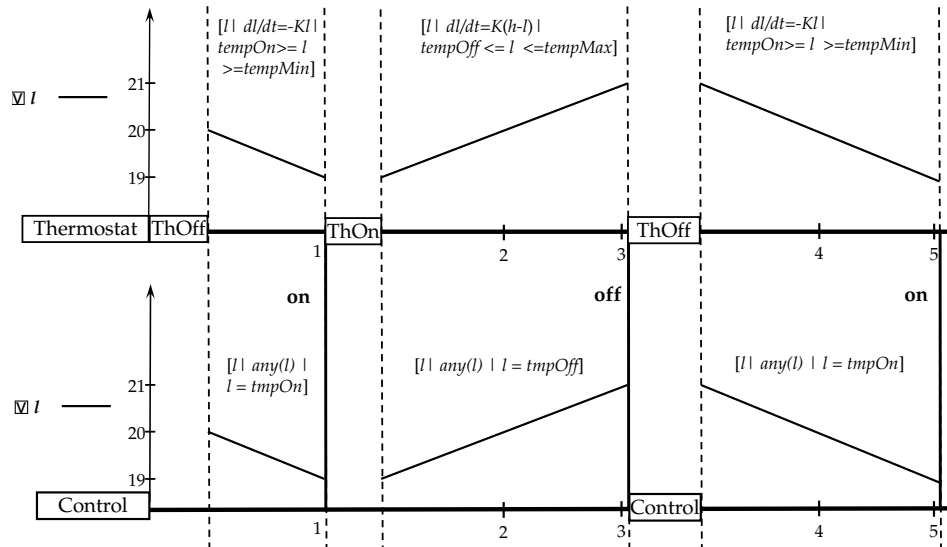


Figure 7.6: Upgraded Thermostat evolution in MSP with recursion

Other techniques to visualise hybrid simulation output In some tools discrete changes are indicated by vertical lines or thicker dots (stars, squares, etc.) at the switching moment (e.g., in Fabian [1999], Lee and Zheng [2005]). Such an approach has a critical drawback, i.e., if a switching structure is complex and the system switches several times at the same time point, then only the values are visible in the graph and the switching order is not.

Hedlund [1999, p.74] uses “additional dimension” in the three-dimensional plots to display switching planes/points.

Levine [2003] visualises the growth of explored branches. Such an approach may be useful to display results of several simulation runs.

One more solution is to use two-dimensional plots (planes) to represent continuous-time evolution phases and insets of directed graphs (or just vectors representing actions) between them. Such visualisation methods are regularly used in the literature to illustrate the nature of hybrid phenomenon. It can be one plot, like in Andersson [1994, p.106] or several plots [Mosterman and Biswas, 2002, p.5], [Branicky and Mattsson, 1997].

7.5.4 Visualisation of components

In some cases it is preferable to visualise a system components as physical objects. It may uncover some properties of a system that are indiscernible in mathematical representation (graphs, equations). Moreover, it helps to present dynamical behaviour of system to an audience not familiar with graphs or other types of charts. Packages, like Dymola (Section 18) and SIMULINK (Section 7.9) provide means for visual modelling.

Samarin [2002] proposes a visual approach for modelling and simulation of physical systems⁶.

3D visualisation The most impressive manner of visualising simulations is a 3D (three-dimensional) visualisation, especially when an object is placed into the realistic environment. Mueller-Wittig et al. [2002] exemplifies such an approach. Levine [2003] uses it for visualisation of path-finding. Tools, like Dymola (Section 18) and SIMULINK (Section 7.9) provide means for 3D visualisation.

Different, but very impressive examples of 3D visualisation are available on TV (e.g., Discovery) every day.

7.6 Simulation of Zeno behaviour

There is a number of problems in the hybrid systems area. The *Zeno phenomenon* is one of them. It manifests itself when the system tries to take an infinite number of discrete transitions in a finite amount of time. An example of Zeno behaviour is distinctly visible in Figures 2.2 and 2.3 (Section 2.2.1), when the steps are getting repeatedly smaller. Such behaviour forces the simulator to make continually smaller steps and at some moment, if the formalism and/or tool are not Zeno-proof, it yields imprecise results caused by numerical computation problems. The Zeno phenomenon is studied only to a small extent, however, some interesting results are reported in Johansson et al. [1999].

Efficient techniques to deal with Zeno behaviour are still to be designed. As a precaution, a tool can try to detect situations when the step-size is repeatedly decreasing and inform user about it.

In BHAVE prototype we do not use any technique to detect Zeno behaviour. However, usually in definitions of trajectory prefixes conditions are used to restrict allowed evolutions. Consequently, numerical errors leading to the restricted values are detected, and instead of producing misleading simulation results, simulation process is halted. Moreover, simulations of the bouncing ball (Example 5.8.1) and the two tanks (Example 5.8.4) with BHAVE prototype show that the prototype progresses in a very small steps and stabilises on certain values. While in naïve experiments with some tools simulation continues in the restricted state space due numerical errors.

We anticipate, that in the future, the techniques based on states sequence analysis can be designed to detect situations when the same process expression in succeeding states is observed, and the steps (time intervals) between these states are decreasing.

7.7 Architecture

There is a plentitude of simulation tools (Section 7.9) for numerous hybrid systems modelling and analysis frameworks. Different architectures are embraced depending on the origins, purpose of the tool, the hybrid formalisms it supports.

In Figure 7.7 we separate the generic components of a hybrid systems simulator. Such an architecture is suitable for a BHPC simulation tool. In BHAVE toolset

⁶A demo is available at <http://www-sop.inria.fr/mimoso/rp/SimulationInPhysics/index.html>.

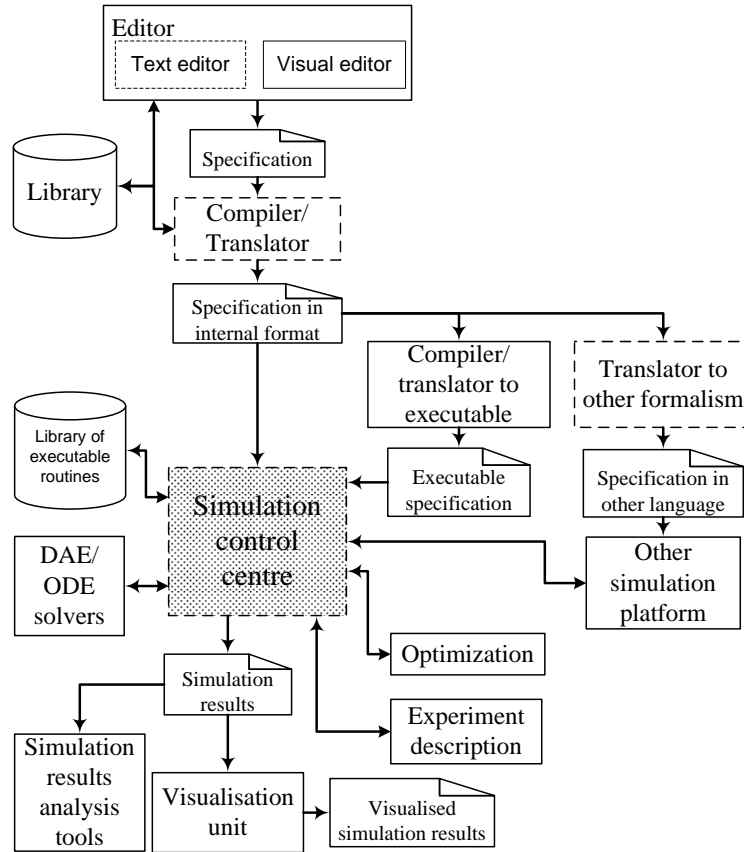


Figure 7.7: General architecture of simulation package

(Appendix D) we implemented only a part of the presented architecture to evaluate adequacy of the proposed simulation algorithms and architecture. These parts are singled out by the dashed border lines.

Editor: provides facilities for editing specifications. If ASCII⁷ based language is used, an ordinary text editor is sufficient for the task. However, features as syntax highlighting, context sensitive help, code folding/unfolding are very handy too. An integrated development environment is an even a more effective solution, if it provides the above mentioned facilities, and has features like special symbol support, etc. Moreover, visual editors are required for visual formalisms, and most of the time they are specific to the formalism. Nevertheless block diagrams and object diagrams editors are general enough to be used with different tool-sets.

BHPCC [van Putten, 2006] provides text editor and some visual editor functionality. An ordinary text editor can be used to edit specifications in BHPCC ASCII

⁷See <http://en.wikipedia.org/wiki/ASCII>.

language [van Putten, 2006, Bhave prot] as well.

Compiler/Translator: transforms a specification to an internal format. Detailed design of the component is subject to the chosen formalism and chosen simulation technique.

In `BHAVE` toolset this functionality is provided by `BHPCC`. It translates a specification in `BHPC` ASCII language to an internal XML based format [van Putten, 2006, Bhave prot].

Library: is usually used to store a collection of code snippets, predefined elements.

Compiler/Translator to executable: transforms a specification from the internal format to the executable format. In some cases it is done in one step, e.g, if the internal representation is used directly by the simulation engine.

Simulation control centre: is a simulation engine. It manages simulation process, provides facilities for interactive simulation control, determines how to proceed, glues together all simulation systems components: ODE/DAE solvers, executable routines libraries, other simulation tools. Moreover, it generates simulation results in requested form.

Several components of `BHAVE` toolset provide such functionality. `BHAVE` prototype is a tool for a hybrid simulation of `BHPC` specification provided in the internal format (Appendix D). Besides, `DISCRETE BHAVE` can be used for discrete simulation [Krilavičius and Schonenberg, 2005, Schonenberg, 2006].

Translator to other formalism: translates the specification from the internal representation form to other formalisms (to be analysed using another tool).

`BHPC2Mod` component (provided as a part of `BHPCC`) translates sequential `BHPC` specification to `Modelica™` [van Putten, 2006] that can be simulated using, e.g., `Dymola` (Section 18).

Experiment description: unit provides facilities for complex experiments planning. In particular, an elaborate experiment planning unit is profitable, if the batch simulation mode (Section 7.8) is used regularly.

Library of executable routines: contains pre-compiled code snippets, etc.

Optimisation: component provides simulation optimisation facilities.

ODE/DAE solvers simulate (solve) ODE/DAE.

In `BHAVE` prototype `MAPLE` is used to solve ODE (Appendix D).

Other simulation platform: can be used for co-simulation.

`Modelica™` language (in particular, `Dymola`) can be used to simulate sequential `BHPC` specifications translated by `BHPC2Mod`.

Visualisation unit: provides visualisation facilities for simulator.

Current version of `BHAVE` toolset does not include any visualisation tools. However, Microsoft (R) Excel 2003 XY(Scatter) routine of Chart Wizard was successfully used to generate simple plots.

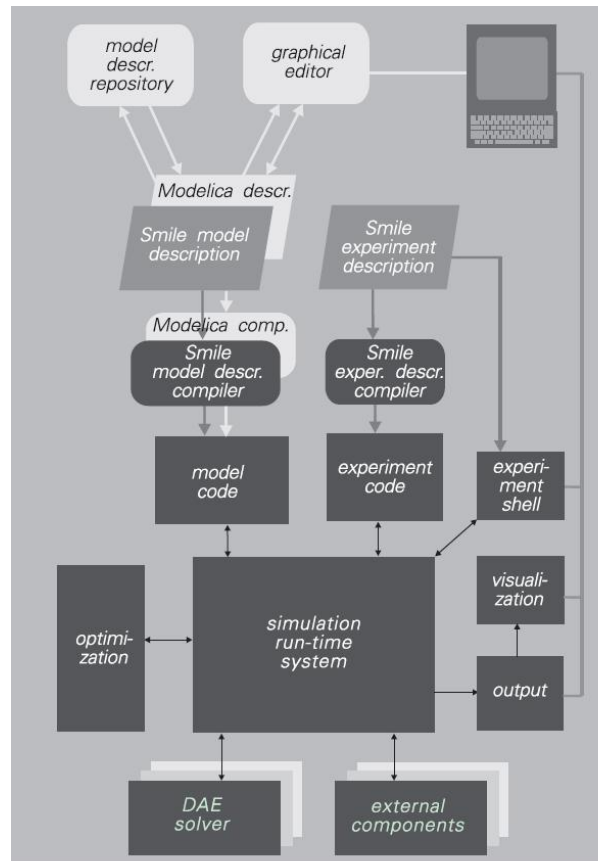
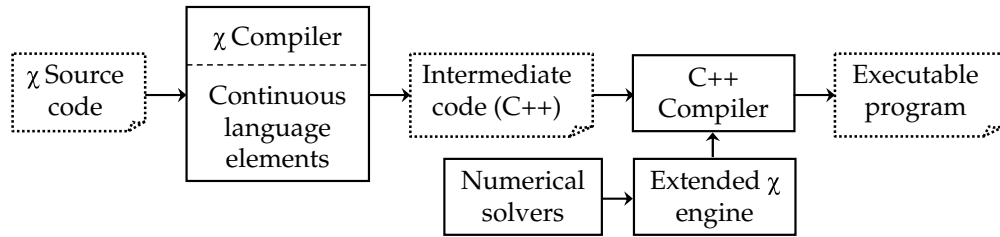


Figure 7.8: Smile/M architecture

Simulation results analysis tools. Specialised tools and packages can be employed for simulation results analysis.

Other architectures Different authors propose slightly different views on the architectures or the architectures itself, but the essence remains the same.

- In Ernst et al. [1997] the Smile/M architecture is described, see Figure 7.8. Smile/M is the Smile-Modelica™ simulation environment. The architecture resembles our version augmented with an interaction between the Smile system and Modelica™. Furthermore, more attention is concentrated on the experiment planning and interaction between Modelica™ and Smile.
- Broenink and Weustink [1996] give an orthogonal view of the simulation process. The process is divided into three constituents.
 1. The *simulation model* is the object structure of the model to be simulated. It corresponds to the executable specification (or the specification in internal

Figure 7.9: Hybrid *chi* simulator

format) in our architecture. Therefore, the model to simulate or its internal representation is thought about more as a part of the system, not just the data.

2. The *simulation experiment* provides additional simulation information. It corresponds to the experiment description and simulation control centre interface in our architecture.
 3. The *simulation program* provides a simulation kernel and its relation with the model and experiment. It directly corresponds to the simulation control centre in our architecture.
- Hofkamp [2001] discusses the steps of transforming the Hybrid χ (Section 7.9) specification to the executable code. A resembling simulator structure is presented in Fabian [1999, p.85], see Figure 7.9. The hybrid χ architecture implements only a part of our architecture. That is, the χ Compiler corresponds to the coupled compiler to internal code and compiler to executable. Then the C++ compiler transforms it to executable with added simulation control centre (or extended χ engine in the hybrid *chi* terminology) and numerical solvers. Such an approach allows to gain a better performance, but may put certain restrictions on the flexibility of the control centre. Furthermore, it would increase complexity of software and dependency on supplementary software, e.g., C++ compiler.

7.8 Simulation modes

There are several different ways to engage in simulation. In some cases it is convenient to scrutinise every step of simulation before choosing the next step, i.e., to run simulation stepwise. Running simulation for while and only then analysing the results is appropriate choice for some problems. In some cases it is handy to define a set of experiments, and then run it as a batch. These techniques are called *simulation modes*.

Interactive simulation is a kind of simulation that includes an active human participation, i.e., at the certain moments of simulation an operator should interactively choose how to proceed. Interactive simulation is usually employed while debugging specification, or analysing a small part of it.

Automatic simulation is a kind of simulation that does not require human interaction. Mostly it is used for long simulation runs. However, it can be combined with the interactive simulation, i.e., an human operator can stop the process at a certain moment, adjust the settings/values and then let it proceed.

Batch simulation is a generalised version of automatic simulation. Series of simulation runs are specified and executed, mostly without any human intervention. It can be a simple list of simulation runs with different parameters, or some batch language can be employed to define the order and types of experiments based on the simulation results. Usually batch simulation is used to perform long and complex experiments. A special tools can be used to analyse the results of such experiments.

The ultimate goal for BHP simulation is to provide all above mentioned simulation modes. However, in the *proof-of-concept* version of tool the interactive mode have been implemented first. The automatic and batch simulation régimes can be added later.

7.9 Tools overview

There are many different tools and languages for modelling, simulation and analysis of hybrid systems. An exhaustive overview of tools and languages is given in Carloni et al. [2004].

Here we provide concise descriptions of some of the most popular tools. Moreover, we compare their features in Table 7.1 and attempt to establish the B_{HAVE} toolset and B_{HAVE} prototype whereabouts in the listed tools context.

The table requires some explanations.

- By *partial semantics* we mean that only a part of behaviour is defined in a formal way.
- We distinguish three classes of maturity. We use notion *industrial* to denote tools that are widely used in industry, have good support and documentation, are stable and reliable. *Academic* tools define a wide range of tools that are usually developed in research institutions, and require more expertise from users. However, some of the tools in this class are almost industrial strength, and some just in a bit better state than prototypes. By *prototypes* we mean tools that were developed to illustrate and evaluate theoretical results. Usually such tools are unstable, unreliable and lack functionality.

20-sim 20-sim is a bond graphs (Section 3.3.12) based industrial level tool for modelling and simulation of dynamical systems. It fully supports graphical modelling and allows to design and analyse dynamical systems in an intuitive and user friendly way. Web page: <http://www.20sim.com/>.

AnyLogic AnyLogic [Borshchev et al., 2000] is a virtual prototyping environment, based on UML-RT, Java and algebraic-differential equations. The tool is used to model

7. SIMULATION OF BEHAVIOURAL HYBRID PROCESS CALCULUS

Tool	Formal semantics	Formalism	Maturity	Features
20-Sim	yes	bond graphs	industrial	modelling, simulation
AnyLogic	no	UML-RT and DAE	industrial	modelling, simulation
CHARON	yes	CHARON	academic	modelling, simulation
CheckMate	yes	threshold event driven hybrid systems	academic	modelling, verification
d/dt	yes	hybrid automata	prototype	verification
Hybrid χ	yes	Hybrid <i>chi</i> calculus	academic	modelling, simulation
HYSDEL	partial	PWA, MLD	academic	modelling, simulation, controller generation
HyTech	yes	hybrid automata	academic	modelling, verification
HyVisual	yes	HyVisual	academic	visual modelling, simulation
Dymola	partial	Modelica™	industrial	modelling, simulation
SCICOS	no	SCICOS	industrial	modelling, simulation, analysis
SHIFT	yes	dynamic networks of hybrid automata	academic	modelling, simulation
STATEFLOW / SIMULINK	partial	finite state machines, flow diagrams, statecharts	industrial	modelling, simulation, analysis
UPPAAL	yes	networks of timed automata	academic	modelling, verification
BHAVE toolset	yes	Behavioural Hybrid Process Calculus	prototype	modelling, simulation

Table 7.1: Hybrid systems modelling and analysis tools

the broad spectrum of systems, but the simulation and modelling results should be evaluated taking in consideration that the underlying semantics of the tool are not formal. Web page: <http://www.xjtek.com>.

Charon The CHARON toolkit is based on the CHARON language (Section 3.3.11). The toolkit is written in Java. It has a graphical user interface, a visual input language similar to STATEFLOW (Section 7.9), a type checker and a simulator. For visualisation of output a Ptolemy (Section 7.9) plotter is used. Web page: <http://www.cis.upenn.edu/mobies/charon>.

CheckMate CHECKMATE [Krogh and Chutinan, 1999, Silva et al., 2000] is a hybrid systems verification toolbox for SIMULINK (Section 7.9). It is based on the *threshold event driven hybrid systems* (TEDHS) [Krogh and Chutinan, 1999, Silva et al., 2000]. Discrete changes in TEDHS are generated only by hitting specified thresholds that are defined by hyperplanes. Non-linear continuous dynamics are supported. The tool is constructed using SIMULINK GUI and user defined MATLAB *m*-files. Web page: <http://www.ece.cmu.edu/~webk/checkmate>.

d/dt d/dt [Asarin et al., 2001] is a tool for reachability analysis of continuous and hybrid systems with linear differential inclusions. The tool accepts as an input a hybrid automaton (Section 3.3.6) with linear continuous dynamics, potentially with an uncertain, bounded input of the form $\frac{dx}{dt} = Ax + u$ where u is an input taking values in a bounded convex polyhedron U and the invariants and transition guards are defined by convex polyhedra. Web page <http://www-verimag.imag.fr/~tdang/ddt.html>.

Hybrid χ The Hybrid χ (Section 5) simulator is described in Fabian [1999], van Beek and Rooda [2000]. It has been successfully applied to a number of case studies. Web page (Chi compiler): <http://chi-compiler.gforge.se.wtb.tue.nl>.

HYSDEL HYSDEL [Torrise et al., 2002, Potočník et al., 2003] is a control-oriented language for describing hybrid systems. Systems are modelled as *discrete hybrid automata* (DHA) [Bemporad, 2003] or mixed logical dynamical systems (Section 3.3.3) and transformed to the corresponding piecewise-affine systems (Section 3.3.2). HYSDEL is limited to affine hybrid systems and discrete dynamics. It can be used together with Multi-Parametric Toolbox (MPT) [Kvasnica et al., 2004], which is a free MATLAB toolbox for design, analysis and deployment of optimal controllers for PWA systems. Web page: <http://www.tik.ee.ethz.ch/~samarjit/HYSDEL.html>.

HyTech HyTech is an automatic tool for the analysis of embedded systems. HyTech computes the condition under which a linear hybrid system satisfies a temporal requirement. Hybrid systems are specified as collections of hybrid automata (Section 3.3.6) with discrete and continuous components, and temporal requirements are verified by symbolic model checking. If verification fails, then HyTech generates a diagnostic error trace. The standard reference to the HyTech algorithm is Alur et al. [1996a], and the standard reference to the HyTech tool is Henzinger et al. [1997]. Web page: <http://www-cad.eecs.berkeley.edu/~tah/HyTech>.

HyVisual and Ptolemy HyVISUAL [Lee and Zheng, 2005, Brooks et al., 2004] is a visual modeller and simulator for continuous-time dynamical and hybrid systems. It is a part of the Ptolemy project⁸ and is based on Ptolemy II [Lee, 2004]. Ptolemy II is a toolkit written in Java for modelling and design of heterogeneous, concurrent systems.

HyVISUAL provides means for modelling and simulation of continuous-time dynamical and hybrid systems. The models are built using block diagrams based

⁸See <http://ptolemy.eecs.berkeley.edu/>.

language. Semantics for the language are given in Lee and Zheng [2005]. HyVISUAL web page: <http://ptolemy.eecs.berkeley.edu/hyvisual>. Ptolemy II web page: <http://ptolemy.eecs.berkeley.edu/ptolemyII>.

Modelica™ Modelica™ (Section 3.3.13) is a language for hierarchical physical modelling. It is an object-oriented language for modelling physical systems for the purpose of simulation. Modelica™ is non-causal and multi-domain, with a fast growing collection of libraries. It is used in several commercial tools, like Dymola (<http://www.Dynasim.se>) and MathModelica (<http://www.mathcore.com>), and in an open source tool Open Modelica [Fritzson et al., 2002]⁹. For more information about Modelica™ developments see <http://www.modelica.org>.

Scicos SCICOS [Nikoukhah and Steer, 1997] is a SCILAB¹⁰ based package for the modelling and simulation of dynamical systems. SCILAB can be considered as a free version of MATLAB and SCICOS as a SIMULINK. Web page: <http://www.scicos.org>.

Shift SHIFT [Deshpande et al., 1997] is a language for modelling and analysis of dynamic networks of hybrid automata. Hybrid systems in SHIFT have a dynamically changing structure. Components of the hybrid system can be created, interconnected and destroyed as the system evolves. For more information and developments of SHIFT and successor λ -SHIFT see <http://www.path.berkeley.edu/shift> and <http://www.gigascale.org/shift>.

Stateflow and Simulink STATEFLOW/SIMULINK [Hamon and Rushby, 2004, Stateflow] pair is a toolset for the modelling and design of dynamical systems. It is based on the combination of Statecharts [Harel, 1987], finite state machines and flow diagrams. The tool is a commercial product of Mathworks. Together with MATLAB it is a *de facto* standard tool for academia and industry dealing with continuous and discrete dynamics. Web page: <http://www.mathworks.com/products/stateflow>.

UPPAAL UPPAAL [Behrmann et al., 2004] is an integrated tool environment for modelling, validation and verification of real-time systems based on networks of timed automata [Alur and Dill, 1992], extended with data types. It is restricted to very simple dynamics, namely clocks. Web page: <http://www.uppaal.com>.

BHAVE prototype in the context of hybrid systems modelling and analysis tools
The BHAVE toolset (Appendix D) is a collection of software for modelling and analysis of hybrid systems specified using Behavioural Hybrid Process Calculus. The BHAVE prototype (Appendix D) is a proof-of-concept tool for simulation of a subset of Behavioural Hybrid Process Calculus.

BHAVE prototype is rather an immature tool in contrast to industrial strength tools as Dymola, STATEFLOW/SIMULINK or 20-sim. However, it is easy to see from Table 7.1 that it falls into a category of tools that have formal semantics and can be used for

⁹For more information see <http://www.ida.liu.se/labs/pelab/modelica/t.php3?page=open.php3>.

¹⁰See <http://scilabsoft.inria.fr>.

modelling and simulation of hybrid systems. We would like to note that most of industrial strength tools are not based on formalisms with formal semantics in contrast to the tools from academia. But the increasing maturity and number of tools with formal semantics shows a growing interest in this type of tools. We find such a tendency encouraging for the `BHAVE` toolset, because it is based on a sound hybrid formalism. Moreover, the experiments with small examples (Appendix D) show that even a prototype can be used to simulate systems with comparatively complicated behaviour, e.g., Zeno phenomena.

It is hard to estimate the exact position of the `BHAVE` toolset in such a wide context, because a prototype implementation does not reveal the full power of `BHPC` (see Chapter 6 for comparison of `BHPC` with diverse formalisms).

7.10 Conclusions

In this chapter we presented a technique for simulation of Behavioural Hybrid Process Calculus. Simulation procedures for all main `BHPC` operators or their generalised versions were defined. The only exceptions are hiding and renaming of trajectory qualifiers. However, for renaming a preprocessor directives (macro commands) like `approach` can be used.

Besides, the major problems occurring in simulation of hybrid systems were surveyed and potential solutions discussed. Occurrence of the presented problems in `BHPC` was examined and solutions or directions for future investigation sketched.

We presented an abstract Behavioural Hybrid Process Calculus simulation technique that provides conceptual solutions. Technical details and implementation dependent design decisions were left out, because it can be done variously in different development environments. Moreover, we anticipate that even such abstract technique will have to be adapted to a particular implementation following requirements of the modern software engineering and hardware limitations. To facilitate such changes the simulation algorithm was partitioned into small procedures. Consequently, only small parts may have to be adapted.

For some problems future research directions rather than solutions were proposed. For example, it is not completely clear how to deal with or even always detect Zeno behaviour. Efficiency of techniques for non-deterministic choice in a dense interval can be improved as well as the adequateness of making a nondeterministic choice.

We did not propose any technique for the general renaming (and hiding) of trajectory qualifiers. However we believe that it can be done. The problem is essentially related to representation of processes and qualifiers. To support renaming of qualifiers it is necessary to design data structures that allow separate internal and external representations of qualifiers w.r.t. renaming. Moreover, a technique to dynamically map these qualifiers is necessary.

Only the simplified version of continuous-time simulation algorithm was evaluated. We anticipate the presented algorithm may have to be adapted for a concrete implementation. Moreover, some restrictions can be imposed by the chosen ODE/DAE solvers and other symbolic manipulation tools that, e.g., are used to handle conditions and exit conditions in trajectory prefixes, guards.

The proposed simulation algorithms do not include techniques that deal with

7. SIMULATION OF BEHAVIOURAL HYBRID PROCESS CALCULUS

numerical and real numbers representation in computer errors. However such issues are important in complex real-life systems simulation. Robust simulation techniques that can deal with such problems are necessary for industrial complexity tools.

We believe that the proposed techniques can be successfully applied to the simulation of Behavioural Hybrid Process Calculus, and are a good starting point to development of more robust and applicable techniques. As a proof of it, certain parts of the theory proposed in this chapter were applied in `BHAVE` toolset and especially in `BHAVE` prototype (Appendix D) and in Krilavičius and Schonenberg [2005], Schonenberg [2006] and van Putten [2006].

On one occasion a man went off to work and on the way he met another man who, having bought a loaf of Polish bread, was going his way home. And that's just about all there is to it.

Daniil Kharms

8

Concluding remarks

In this chapter we summarise and evaluate the results that were presented in the dissertation. Individual conclusions are available in the last sections of each chapter. In addition, we present several directions and ideas for future research.

8.1 Hybrid systems

In Chapter 1 we introduced *embedded systems* and linked them with hybrid systems. Examples of omnipresence of hybrid systems were provided as a motivation for research. Moreover, we surveyed major topics in the area of hybrid systems (Section 1.2). Two best studied topics of hybrid systems are modelling and analysis of hybrid systems. However, it does not mean that all modelling and analysis problems are already solved. On the contrary, it is just getting mature, terminology is getting unified, and problems more defined, but not really solved. Moreover, we surveyed two emerging research topics: deployment and especially testing of hybrid systems. Of course, these areas already exist for quite a while, but only now they are becoming an object of thorough and systematic investigation¹.

8.2 Modelling of hybrid systems

In Chapter 2 we exemplified hybrid systems by a set of examples. The examples were selected to represent diverse properties of hybrid systems, its variety and occurrence in different applications. Moreover, we think that a good collection of examples can be used to compare and evaluate different formalisms, or rather the suitability of these

¹Control generation for continuous systems is old and mature subject, but it's recent extensions to hybrid systems accommodate only small classes of such systems.

techniques for modelling (and analysis). Such a collection of examples differs from benchmarks, because benchmarks are more fitted to tools comparison. Consequently, such a list of examples would help to identify some characteristics of formalisms in the early stages of development. We believe that the examples provided in Chapter 2 together with the examples from van der Schaft and Schumacher [1998] and De Schutter and Heemels [2004] can be a good starting point for such a collection.

The survey (Chapter 3) of major formalisms for modelling and analysis of hybrid systems and their classification provides a substantial amount of information about principal characteristics of hybrid phenomena. Moreover, as we had anticipated, the analysis shows that hybrid formalism usually retains most of the originating theory characteristics (control theory or computer science) and includes a bigger or smaller selection of features from the complementing theory. In some cases these additional features are hardly noticeable, and sometimes such an amalgamation is not so far from the nice equilibrium between the two world views. However, we did not find a formalism that had both satisfactory integration of concepts from control theory and computer science, fundamental treatment of discrete and continuous behaviours, and well founded definition of compositionality. All these properties are important for an adequate treatment of hybrid systems. Of course, it does not mean that some approaches are worse than others, usually they just reflect different intuitions and goals, and emphasise different characteristics that may be more beneficial in some cases (e.g., mobility).

One of our goals was to unify these three trends and propose a formalism that has a good theoretical foundation, sound definition of compositionality, and is applicable in practice. We think, that a good theoretical foundation and sound definition of compositionality were achieved by defining Behavioural Hybrid Process Calculus (BHPC). Moreover, we tackled issue of practical applicability by defining theory for simulation of BHPC and validating it in prototype tools. However, further work is needed to adapt the formalism to industry needs.

An attempt to merge computer science and control theory in hybrid systems research was taken in Chapter 5. BHPC is based on two fundamental notions of actions and trajectories that describe discrete and continuous evolution of dynamical systems, respectively. At the higher abstraction level these two types of behaviour are treated uniformly, i.e., as normal elements of process algebra. Their behaviour is defined using structural operational semantics (SOS) rules [Plotkin, 1981, 2003]. The rules respect the differences between trajectory prefixes and action prefixes, based on our intuition how such processes should behave. For example, in parallel composition trajectory prefixes are always required to synchronise, while for action prefixes interleaving semantics is adopted. A formal definition of concurrently evolving components allows to reason about them in a compositional manner, i.e., to derive behaviour of the resulting system from the behaviours of the components.

We defined a hybrid strong bisimulation relation for BHPC and proved that it is a congruence. It is one of the most important properties to attain well defined compositionality. Such a property allows to interchange bisimilar processes in any process algebraic expression. In other words, it allows to refine process, change their internal representation, and interchange them without any losses as long as they manifest the same behaviour.

Strong bisimulation (Section 5.4.1) treats silent actions in the same way as usual

actions. However, often it is interesting to analyse systems w.r.t. weaker equivalences. Weak bisimulation [Milner, 1989] is one of the most popular of such equivalences. Hermanns [1998] gives a nice introduction on bisimulation of stochastic models. Bisimulations for hybrid stochastic systems are analysed in Strubbe [2005]. Therefore, we believe that defining a hybrid weak bisimulation or other weaker equivalences for BHPC and analysing its properties may be worthy research topic. However, it is not a trivial topic. For example, it is not completely clear, how to treat hiding of trajectories or trajectory qualifiers. Therefore, developing a theory of weak bisimulation would take a lot more efforts than a hybrid strong bisimulation for BHPC.

We discussed an experimental version of calculus (Section 5.9). It has some very nice properties, as very intuitive version of choice, i.e., superposition that can be seen as a generalisation of ordinary choice, and a different version of trajectory prefix. It is still an experimental version, but we believe that further work on it could yield very interesting results.

In Chapter 6 we compared BHPC with formalisms presented in Chapter 3. Hybrid systems characteristics, extracted in Chapter 3 were used as guidelines for the systematic comparison.

8.3 Analysis of hybrid systems

In Chapter 4 we proposed a technique for stability estimation for a certain class of hybrid automata. In addition, this chapter demonstrates advantages of interdisciplinary research as well. It combines the ideas from computer science and control theory and based on cycles detection and conservative gains estimation. We borrowed the well-known algorithm for transforming finite automaton into an equivalent regular expression for cycles detection from computer science. From control theory we picked-up the idea of conservative gains and used them to estimate stability of cycles.

Moreover, the result shows that at a higher abstraction level it is beneficial to treat both, continuous and discrete behaviours uniformly. But when it is necessary, it should be easy to go to the detailed level and apply actions specific to continuous or discrete behaviour.

The current gains estimation technique is restricted to the two dimensional case. This restriction was imposed by the used gains calculation techniques. The main algorithm abstracts from these techniques. Therefore, potentially the results could be generalised for higher dimensions.

Yet another important issue is the practical applicability of BHPC. In other words, the question was (and is) to what extent it permits tools. We have chosen simulation, because it is one of the *de facto* standards for the analysis of hybrid systems, widely accepted in industry and academia. The results of this exercise were reported in the last technical chapter (Chapter 7).

We devised a simulation algorithm for a subset of BHPC operators and tested some of the proposed techniques in BHAVE prototype (Appendix D). The proposed simulation algorithm defines one of the possible ways to simulate Behavioural Hybrid Process Calculus. However, different software development techniques, programming languages and hardware imposed limitations may require to change certain parts of the algorithm or order of operations. Moreover, parts like the process transfor-

mation to normal form (Section 7.2.4) can be changed to a first transition function [van Eijk, 1988, Eertink, 1994, Schonenberg, 2006] without a loss of generality, because the idea of the procedure is the same. Only a simplified version of continuous simulation (Section 7.2.6) was validated in *BHAVE* and therefore, it will require further extensions.

Chapter 7 surveyed the major problems in simulation of hybrid systems in the light of BHPc and in a more general layout. For some of the issues solutions were proposed, and for the remaining ones the potential courses to tackle the problems were discussed. One of such issues is detection and simulation of Zeno behaviour (Section 7.6). Undetected Zeno behaviour can cause significant decrease in performance and errors in numerical simulation. We proposed a potential solution, that is, to analyse traces and detect a situation, when the same process expression appears within decreasing time intervals. Such analysis may help to detect some occurrences of Zeno behaviour, but at the same time, it would decrease simulation speed, and that may be a problem in real-time simulation.

Yet another interesting problem that is important in a wider context is an effective simulation of non-determinism. Section 7.3 discusses some of techniques and approaches to this problem. Often, non-determinism is supposed to imitate a user. Usually, only the interactive simulation and, in some cases, use of scheduler fulfill such intentions. However, the use of a scheduler decreases performance, and an interactive simulation often is not practical. Therefore, practical issues of different approaches should be analysed.

An interesting continuation of BHPc simulation is an extension of the given algorithm with simulation of hiding and renaming of qualifiers. It would provide more flexibility to use process definitions as templates. However, renaming and hiding are very powerful tools, and should be used with care. For example, hiding may influence the outcome of parallel composition or choice.

Numerical errors and specifics of real numbers representation in computers is one of the major problems in simulation of complex numerically intensive systems. Simulation techniques that take into account these problems are of great interest. We do not really solve this issue, but more formal and careful specification may be helpful to detect an unexpected behaviour and identify its cause.

8.4 General remarks

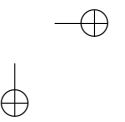
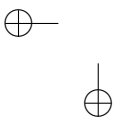
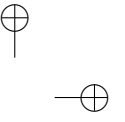
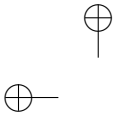
Our aim was to better understand what constitutes the area of hybrid systems, analyse existing formalisms while paying special attention to compositionality, and balance of discrete and continuous behaviours representation. It resulted in stability analysis of a certain class of hybrid automata and Behavioural Hybrid Process Calculus. Furthermore, we proposed a technique for simulation of BHPc. We believe that the proposed calculus provides a good framework for compositional modelling of hybrid systems.

We would like conclude our work with several remarks of a more general nature.

Developments in formal methods research show that research of hybrid systems testing is a very promising area. However it is interesting not only as a separate activity, but as a part of complete life-cycle of mixed hardware and software systems. Methodologies for better incorporation of formal techniques in the whole life-cycle are unjustly neglected. Issues like formal specifications development techniques [Ruys,

2001] and evolution of formal specifications require more attention than they are given now. We believe that it can be interesting research area.

We think, that the mentioned problems and their theoretical solutions, as well as problems discussed in introduction, should be investigated not only on paper, but also, by building and applying tools (based on the developed theory) to sets of benchmarks and practical problems. Such thorough validation of results could be very helpful in gaining more insight on the research area and identifying other relevant issues. We believe that at least partially our research conforms to these requirement, because we not only proposed a technique to solve certain problems (compositional modelling of hybrid systems), but validated it on prototype tool. However, as it is mentioned above, it would be interesting to extend the tool, incorporate it to the systems development life-cycle and experiment with it on the real life case studies. We believe that such an investigation could provide a lot more insight how to improve BHPC, and what future research directions are promising.



A

Stability

A.1 Proofs from Section 4.3

Theorem A.1.1. *Let H be a hybrid automaton with stable locations. Then H is unstable, if it has a non-contractive cycle.*

Proof. Seeking a contradiction, assume that all cycles are contractive. Choose $\varepsilon > 0$ and let $\delta_k > 0$ be such that $\lim_{k \rightarrow \infty} \delta_k = 0$. Assume that for every k there exists a trace $\sigma_k = x_{1,k}e_{1,k} \cdots e_{M-k-1,k}x_{m_k,k}$, with

$$\|x_{1,k}(0)\| < \delta_k \quad \|x_{m_k,k}(t_k)\| \geq \varepsilon$$

Remove from each trace σ_k all cycles to obtain $\tilde{\sigma}_k$. Denote by $\tilde{\eta}_k$ the event sequences corresponding to $\tilde{\sigma}_k$. Since the number of events is finite and since $\tilde{\sigma}_k$ does not contain cycles, it follows that there exists an infinite number of $\tilde{\sigma}_{k_i}$ s and an event sequence $\tilde{\eta}$ such that for all i :

$$\tilde{\eta}_{k_i} = \tilde{\eta} = e_1 e_2 \cdots e_{M-1}.$$

Hence every $\tilde{\sigma}_{k_i}$ is of the form

$$\tilde{\sigma}_{k_i} = x_1^{k_i} e_1 x_2^{k_i} e_2 \cdots x_{M-1}^{k_i} e_{M-1} x_M^{k_i}$$

with

$$x_j^{k_i} : [\tau_{k_i,j}, \tau'_{k_i,j}] \rightarrow \mathbb{R}^n$$

trajectories in location l_j . If for some j $\tau'_{k_i,j} < \tau_{k_i,j+1}$, then a cycle has been removed in between. Since all cycles are contractive in the sense of Definition 4.3.2 we conclude that

$$\|x(\tau_{k_i,j+1})\| \leq \|x(\tau'_{k_i,j})\|. \quad (\text{A.1})$$

A. STABILITY

Notice that since $\|x_{m_k,k}(t_k)\| \geq \varepsilon$ and $\|x_k(0)\| < \delta_k$ we also have that

$$\|x_M^{k_i}(\tau'_{k_i,M})\| \geq \varepsilon \quad \|x_1^{k_i}(\tau_{k_i,1})\| < \delta_k. \quad (\text{A.2})$$

Now, choose $\tilde{\delta}_M > 0$ such that for any trajectory x in location l_M we have

$$\|x(0)\| < \tilde{\delta}_M \Rightarrow \|x(t)\| < \varepsilon.$$

Assume that $\tilde{\delta}_j$ has been defined. Choose $\tilde{\delta}_{j-1}$ such that for every trajectory x in location l_j we have

$$\|x(0)\| < \tilde{\delta}_{j-1} \Rightarrow \|x(t)\| < \tilde{\delta}_j.$$

Take i such $\delta_{k_i} < \tilde{\delta}_1$. It follows that

$$\|x_1^{k_i}(\tau'_{k_i,1})\| < \tilde{\delta}_2$$

from which we conclude

$$\|x_2^{k_i}(\tau'_{k_i,2})\| < \tilde{\delta}_3.$$

Repeating this argument we finally get:

$$\|x_M^{k_i}(\tau'_{k_i,M})\| < \varepsilon.$$

This contradicts the first inequality in Equation (A.2) and therefore not all cycles of H can be contractive. This concludes the proof. \square

A.2 Optimising the Lyapunov function choice

Stability indication provided by gains depends on the chosen Lyapunov functions in locations. These functions can fit trajectories better or worse. The better it fits the trajectory, the less conservative is the gain. Because Lyapunov functions are not unique, procedures to choose the better ones can be provided. We present a procedure for linear dynamics given by a stable matrix and quadratic Lyapunov functions. The need for optimisation and the difference between loose and tight level curves are easily seen in Figure 4.8.

Some of the results were reported in Langerak et al. [2003a,b].

Let $A \in \mathbb{R}^n$ be a stable matrix. Let choose a non-zero $x_0 \in \mathbb{R}^n$ and define the set of level curves corresponding to quadratic Lyapunov functions

$$\Omega_{x_0} = \{P \in \mathbb{R}^{n \times n} \mid A^T P + P A \leq 0, x_0^T P x_0 = 1\}.$$

Ω_{x_0} is a parametrisation of the level curves corresponding to quadratic Lyapunov functions and level unity.

Lemma A.2.1. *Let $A \in \mathbb{R}^{n \times n}$ and let $x_0 \in \mathbb{R}^n$ be a non-zero vector that does not belong to an A -invariant subspace of dimension at most $n - 1$. Let U be an open neighbourhood of $0 \in \mathbb{R}^n$. Then $\text{span}_{t \in U}(\exp(At))x_0 = \mathbb{R}^n$.*

Proof. Assume the contrary. Then for all $t \in U$ there exists non-zero $z \in \mathbb{R}$ such that $z^T \exp(At)x_0 = 0$. Repeat differentiation and substituting $t = 0$ then yields $z^T A^k x_0 = 0 \ k \geq 0$. Define $\mathcal{V} = \text{span}_{k \geq 0} \{A^k x_0\}$. Obviously, \mathcal{V} is an A -invariant subspace. Moreover $z^T \mathcal{V} = 0$ and therefore $\dim \mathcal{V} \leq n - 1$. This is a contradiction and the statement follows. \square

Lemma A.2.2. Let v_1, v_2, \dots, v_n be a basis of \mathbb{R}^n and let $c \in \mathbb{R}$ be a positive constant. Define $\Omega = \{P = P^T \geq 0 \mid v_i^T P v_i \leq c, i = 1, \dots, n\}$. Then Ω is bounded.

Proof. It suffices to prove that there exists a constant $M > 0$ such that for all $x \in \mathbb{R}^n$ with $x = \sum_{i=1}^n \lambda_i v_i$ and $\sum_{j=1}^n \lambda_j^2 = 1$ we have that $x^T P x \leq M$ that is the quadratic forms $x^T P x$ are uniformly (w.r.t. Ω) bounded on the unit sphere. Choose any such x . Then

$$\begin{aligned} x^T P x &= \left(\sum_{i=1}^n \lambda_i v_i \right)^T P \left(\sum_{i=1}^n \lambda_i v_i \right) \\ &= \sum_{i=1}^n \lambda_i^2 v_i^T P v_i + \sum_{i \neq j} \lambda_i \lambda_j v_i^T P v_j \\ &\leq c \sum_{i=1}^n \lambda_i^2 + \frac{1}{2} \sum_{i \neq j} \lambda_i \lambda_j (v_i^T P v_i + v_j^T P v_j) \\ &\leq c + c \sum_{i \neq j} \lambda_i \lambda_j \\ &\leq c + \frac{1}{2} c \sum_{i \neq j} (\lambda_i^2 + \lambda_j^2) \\ &= c + c(n-1) = cn \end{aligned}$$

Where we used that for any two vectors $v, w : v^T P w + w^T P v \leq v^T P v + w^T P w$. \square

Theorem A.2.3. Let $A \in \mathbb{R}^{n \times n}$ be a stable matrix and let $x_0 \in \mathbb{R}^n$ be a non-zero vector. Define $\Omega = \{P \in \mathbb{R}^{n \times n} \mid A^T P + P A \leq 0, x_0^T P x_0 = 1\}$.

- If x_0 does not belong to a proper A -invariant subspace then Ω is compact.
- Every $P \in \Omega$ is positive semi-definite.
- Ω is convex.

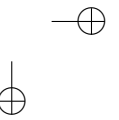
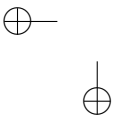
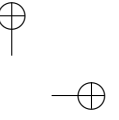
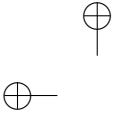
Proof. Notice that since A is stable the set Ω is non-empty. Choose any $P \in \Omega$. Due to the condition on x_0 the trajectory $x(t) = \exp(At)x_0$ spans \mathbb{R}^n . Since x satisfies $\frac{d}{dt}x = Ax$ it follows that $x(t)^T P x(t) \leq x_0^T P x_0 = 1 \ \forall t \geq 0$. Choose time instants t_1, t_2, \dots, t_n such that $\text{span}(x(t_1), \dots, x(t_n)) = \mathbb{R}^n$. It follows from Lemma A.2.2 (with $c = 1$) that Ω is bounded.

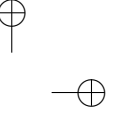
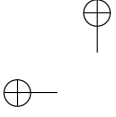
To see that Ω is closed, choose a sequence $P_k \in \Omega$ such that $\lim_{k \rightarrow \infty} P_k = P$. Then

$$A^T P + P A = \lim_{k \rightarrow \infty} A^T P_k + P_k A =: -Q_k.$$

By assumption $Q_k \geq 0$ and therefore $\lim_{k \rightarrow \infty} Q_k =: Q \geq 0$. We conclude that $P \in \Omega$. This proves the first statement.

The second and the third parts are obvious. \square





B

Proofs from Chapter 5

B.1 Proof of Theorem 5.5.4

In this appendix we present a proof of Theorem 5.5.4. We repeat the lemma here for the readers convenience.

Theorem B.1.1 (Consistent signal flow). *If the constraints stated in Definition 5.5.2 are satisfied, then the signal flow is consistent, i.e., a semantic type is preserved. Then for all $B_1, B_2, B_3, \varphi, \psi$ holds.*

$$\text{if } B_1 \xrightarrow{\varphi} B_2 \xrightarrow{\psi} B_3 \text{ then } B_1 \xrightarrow{\varphi;\psi} B_3$$

Proof. By induction on the length of the derivation of $B_1 \xrightarrow{\varphi} B_2$ and analysis of the rules that apply to the syntactical format of the resulting B_2 .

$l = 1$. This means that we must have applied the trajectory prefix rule with $\varphi \in \overline{\Phi}$ and for some name f'

$$B_1 = [f \mid \Phi].B(f) \xrightarrow{\varphi} [f' \mid \Phi \setminus \varphi].B(\varphi; f') = B_2$$

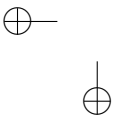
Then we get the following cases.

1. **Case 1.** $B_2 \xrightarrow{\psi} B_3$ is result of applying trajectory prefix rule (5.3), i.e., $\psi \in \overline{\Phi \setminus \varphi}$ and

$$B_2 = [f' \mid \Phi \setminus \varphi].B(\varphi; f') \xrightarrow{\psi} [f'' \mid \Phi \setminus \varphi \setminus \psi].B(\varphi; \psi; f'') = B_3$$

but then we have $\varphi; \psi \in \overline{\Phi}$ and $\Phi \setminus \varphi \setminus \psi = \Phi \setminus (\varphi; \psi)$ so that

$$B_1 = [f \mid \Phi].B(f) \xrightarrow{\varphi;\psi} [f'' \mid \Phi \setminus (\varphi; \psi)].B(\varphi; \psi; f'') = B_3$$



2. **Case 2.** $B_2 \xrightarrow{\psi} B_3$ is result of applying concatenation rule (5.4b), i.e., $\psi = \psi_1 ; \psi_2$ with $\psi \in \Phi \setminus \varphi$ and

$$\frac{B(\varphi ; \psi_1) \xrightarrow{\psi_2} B_3}{[f' \mid \Phi \setminus \varphi].B(\varphi ; f') \xrightarrow{\psi_1 ; \psi_2} B_3} \quad \psi_1 \in \Phi \setminus \varphi$$

So the premise $B(\varphi ; \psi_1) \xrightarrow{\psi_2} B_3$ must hold. But as $\varphi ; \psi_1 \in \Phi$, we can apply another instance of concatenation, i.e.,

$$\frac{B(\varphi ; \psi_1) \xrightarrow{\psi_2} B_3}{[f \mid \Phi].B(\varphi ; f') \xrightarrow{\varphi ; \psi_1 ; \psi_2} B_3} \quad \varphi ; \psi_1 \in \Phi$$

and done.

- $l > 1$.
1. $B_1 \xrightarrow{\varphi} B_2$ is result of concatenation, i.e., $\varphi = \varphi_1 ; \varphi_2$. Then $B_1 = [f \mid \Phi].B$ with $\varphi_1 \in \Phi$ and we get $B(\varphi_1) \xrightarrow{\varphi_2} B_2$, but by induction hypothesis $B(\varphi) \xrightarrow{\varphi_2 ; \psi} B_3$ and then by concatenation $B_1 \xrightarrow{\varphi_1 ; \varphi_2 ; \psi} B_3$.
 2. $B_1 \xrightarrow{\varphi} B_2$ is result of Σ -rule (5.6b), i.e., $B_1 = \sum_{i \in I} C_i$ and for $i \in I$ $C_i \xrightarrow{\varphi} B_2$. Then by induction hypothesis $C_i \xrightarrow{\varphi ; \psi} B_3$ and by Σ -rule $B_1 \xrightarrow{\varphi ; \psi} B_3$.
 3. $B_1 \xrightarrow{\varphi} B_2$ is result of \parallel_A^H -rule (5.9c), i.e., $B_1 = B_{11} \parallel_A^H B_{12}$ and $B_{11} \xrightarrow{\varphi_l} B'_{11}$, $B_{12} \xrightarrow{\varphi_r} B'_{12}$, $\varphi = \varphi_l \times_H \varphi_r$, and $B_2 = B'_{11} \parallel_{B_{12}}$. This implies $\psi = \psi_l \times_H \psi_r$ and $B'_{11} \xrightarrow{\psi_l} B''_{11}$, $B'_{12} \xrightarrow{\psi_r} B''_{12}$, and consequently $B_3 = B''_{11} \parallel_A^H B''_{12}$. Then by the induction hypothesis

$$B_{11} \xrightarrow{\varphi_l ; \psi_l} B'_{11} \quad B_{12} \xrightarrow{\varphi_r ; \psi_r} B'_{12}$$

and by construction

$$B_1 \xrightarrow{\varphi ; \psi_l \times_H \psi_r} B'_3$$

i.e.,

$$B_1 \xrightarrow{\varphi_l \times_H \varphi_r ; \psi_l \times_H \psi_r} B'_3$$

Therefore

$$B_1 \xrightarrow{\varphi ; \psi} B'_3$$

4. $B_1 \xrightarrow{\varphi} B_2$ is result of hiding (5.10) or renaming (5.11) rules. Similar to choice or parallel composition, but simpler, therefore we omit them.
5. Let $B_1 \xrightarrow{\varphi} B_2$ be a result of recursion rule, i.e., $B_1 = X$ with $X = B$ and $B_1 \xrightarrow{\varphi} B_2$. By induction hypothesis we have $B \xrightarrow{\varphi ; \psi} B_3$. And by the recursion rule we get $X \xrightarrow{\varphi ; \psi} B_3$.

□

B.2 Proof of Theorem 5.5.6

Here we provide a proof of Theorem 5.5.6. We repeat theorem formulation for the readers convenience.

Theorem B.2.1. *Hybrid strong bisimulation equivalence on HTSs is a congruence w.r.t. the operations of BHPC defined by the in Section 5.5.2.*

Proof. Basically, we have to show that bisimulation is preserved under application of all BHPC operators. We base our proof on Milner [1989, p.99–101].

We will prove it by investigating all cases. Notice, that symmetric arguments have to be applied to complete the proof.

Action prefix case. The proof for action prefix is standard.

Trajectory prefix case. The case with $[f \mid \Phi].B_1$ and $[f \mid \Phi].B_2$ is a bit more complicated.

If $B_1(\varphi) \sim B_2(\varphi)$ for all $\varphi \in \Phi$ then $[f \mid \Phi].B_1(f) \sim [f \mid \Phi].B_2(f)$.

For each $\varphi \in \Phi$ exists bisimulation relation such that $(B_1(\varphi), B_2(\varphi)) \in \mathcal{R}_\varphi$. But then $\{([g \mid \Psi].B_1, [g \mid \Psi].B_2) \mid \text{for all } \Psi\} \cup \bigcup_\varphi \mathcal{R}_\varphi$ is the bisimulation relation we need. Then we get two subcases:

1. Action a.

$$\begin{aligned} [f \mid \Phi].B_1(f) &\stackrel{a}{\rightarrow} B'_1 && \implies \text{by concatenation rule (5.4)} \\ \exists \epsilon \in \Phi &&& \implies \text{by concatenation rule (5.4)} \\ B_1(\epsilon) &\stackrel{a}{\rightarrow} B'_1 && \implies \text{by bisimulation assumption} \\ \exists B'_2 \sim B'_1, B_2(\epsilon) &\stackrel{a}{\rightarrow} B'_2 && \implies \text{by concatenation rule (5.4)} \\ [f \mid \Phi].B_2(f) &\stackrel{a}{\rightarrow} B'_2 \end{aligned}$$

2. Trajectory φ . In this case we will use an induction on the length of the derivation.

(a) Let $h = 1$. Then in one step only (5.3) can be applied. We get:

$$\begin{aligned} [f \mid \Phi].B_1(f) &\xrightarrow{\varphi} [f' \mid \Phi \setminus \varphi].B_1(\varphi; f') && \implies \text{by trajectory prefix (5.3)} \\ \exists \varphi \in \overline{\Phi} &&& \implies \text{by trajectory prefix (5.3)} \\ [f \mid \Phi].B_2(f) &\xrightarrow{\varphi} [f' \mid \Phi \setminus \varphi].B_2(\varphi; f') \end{aligned}$$

(b) Let $h > 1$. Let $\varphi = \varphi_1; \varphi_2$ be the relevant splitting of φ , i.e., such that $\varphi_1 \in \Phi$. Then

$$\begin{aligned} [f \mid \Phi].B_1(f) &\xrightarrow{\varphi} B'_1 && \implies \text{by concatenation (5.4) and } \varphi = \varphi_1; \varphi_2 \\ B_1(\varphi_1) &\xrightarrow{\varphi_2} B'_1 && \implies \text{by bisimulation assumption} \\ \exists B'_2 \sim B'_1, B_2(\varphi_1) &\xrightarrow{\varphi_2} B'_2 && \implies \text{by concatenation rule (5.4)} \\ [f \mid \Phi].B_2(f) &\xrightarrow{\varphi} B'_2 \end{aligned}$$

Choice case. The choice case is standard, therefore we omit it.

Parallel composition case. If $B_1 \sim B_2$ then there exists bisimulation relation \mathcal{R} such that $(B_1, B_2) \in \mathcal{R}$. Then we will extend bisimulation relation with

$$\{(B_1 \parallel_A^H C, B_2 \parallel_A^H C) \mid (B_1, B_2) \in \mathcal{R}, C \text{ is a process}\}$$

and

$$\{(C \parallel_A^H B_1, C \parallel_A^H B_2) \mid (B_1, B_2) \in \mathcal{R}, C \text{ is a process}\}.$$

Then we get two cases

1. Action a . Then we get three sub-cases, i.e., $a \in A$ or $a \notin A$ and B_1 or C can take it.

(a) Case $a \in A$.

$$B_1 \parallel_A^H C \xrightarrow{a} B'_1 \parallel_A^H C' \quad \Longrightarrow \quad \text{by parallel composition (5.9a)}$$

$$B_1 \xrightarrow{a} B'_1, C \xrightarrow{a} C' \quad \Longrightarrow \quad \text{by bisimulation assumption}$$

$$\exists B'_2 \sim B'_1, B_2 \xrightarrow{a} B'_2 \quad \Longrightarrow \quad \text{by parallel composition (5.9a)}$$

$$B_2 \parallel_A^H C \xrightarrow{a} B'_2 \parallel_A^H C'$$

(b) Case $a \notin A$ and B_1 can take a .

$$B_1 \parallel_A^H C \xrightarrow{a} B'_1 \parallel_A^H C \quad \Longrightarrow \quad \text{by parallel composition (5.9b)}$$

$$B_1 \xrightarrow{a} B'_1 \quad \Longrightarrow \quad \text{by bisimulation assumption}$$

$$\exists B'_2 \sim B'_1, B_2 \xrightarrow{a} B'_2 \quad \Longrightarrow \quad \text{by parallel composition (5.9b)}$$

$$B_2 \parallel_A^H C \xrightarrow{a} B'_2 \parallel_A^H C$$

(c) Case $a \notin A$ and C can take a . Similar to the above case.

2. Trajectory φ . Let $\varphi = \varphi_l \times_H \varphi_r$.

$$B_1 \parallel_A^H C \xrightarrow{\varphi} B'_1 \parallel_A^H C' \quad \Longrightarrow \quad \text{by parallel composition (5.9c)}$$

$$B_1 \xrightarrow{\varphi_l} B'_1, C \xrightarrow{\varphi_r} C' \quad \Longrightarrow \quad \text{by bisimulation assumption}$$

$$\exists B'_2 \sim B'_1, B_2 \xrightarrow{\varphi_l} B'_2 \quad \Longrightarrow \quad \text{by parallel composition (5.9c)}$$

$$B_2 \parallel_A^H C \xrightarrow{\varphi} B'_2 \parallel_A^H C'$$

Hiding and renaming cases. We omit the proofs for hiding and renaming, because they are standard, and moreover, similar to the above proofs.

Recursion case. The procedures following the lines of Milner [1989, p.99–101]. The induction on the depth of derivation is used, and then all above cases are analysed in the similar manner.

Let \widetilde{B} is the process expressions with at most \widetilde{X} process variables (i.e., recursive calls) and \widetilde{P} be a set of process identifiers. Then we will use an expression $\widetilde{B}(\widetilde{P})$ to denote a process expressions \widetilde{B} with \widetilde{X} substituted by \widetilde{P} .

Let \widetilde{B}_1 and \widetilde{B}_2 contain variables \widetilde{X} at most. Let $\widetilde{P}_1 \triangleq \widetilde{B}_1(\widetilde{P}_1)$ and $\widetilde{P}_2 \triangleq \widetilde{B}_2(\widetilde{P}_2)$, correspondingly. So we need to show that if $\widetilde{B}_1 \sim \widetilde{B}_2$, then $\widetilde{P}_1 \sim \widetilde{P}_2$.

For notational simplicity we will show it for a simpler version only. Let B_1 and B_2 contain variable X at most. Let $P_1 \triangleq B_1(P_1)$ and $P_2 \triangleq B_2(P_2)$, correspondingly. We show that if $B_1 \sim B_2$, then $P_1 \sim P_2$.

Let $C(P)$ be a process with only one process variable X . Then let $C(P_1)$ and $C(P_2)$ be C with process variable substituted by P_1 and P_2 , correspondingly.

We will extend bisimulation relation with

$$\{(C(P_1), C(P_2)) \mid C \text{ has at most process variable } X\}$$

Let \equiv denote the syntactical equivalence of process expressions.

To show it is enough to prove $C(P_1) \xrightarrow{a/\varphi} B' \Rightarrow C(P_2) \xrightarrow{a/\varphi} B'' \wedge B' \sim B''$. We will prove it by induction on the depth of the derivation tree by which $C(P_1) \xrightarrow{a/\varphi} B'$ is derived. We will analyse different cases of C form.

1. Let $C \equiv X$. Then we get two cases, with action prefix and trajectory prefix. We will analyse trajectory prefix case to exemplify the proof. Action prefix case is similar. We have that $C(P_1) \equiv P_1$, then

$$\begin{aligned} P_1 &\xrightarrow{\varphi} E && \implies \text{by a shorter derivation (5.12)} \\ B_1(P_1) &\xrightarrow{\varphi} E && \implies \text{by the ind. hypoth., with } B_1 \equiv C \text{ in bisimulation} \\ B_1(P_2) &\xrightarrow{\varphi} E', E' \sim E && \implies B_1(X) \sim B_2(X) \\ \exists B_2 &\xrightarrow{\varphi} E'', E'' \sim E' && \implies \text{recursion (5.12)} \\ \exists P_2 &\sim E', E \sim E'' \end{aligned}$$

2. Let $C \equiv [f \mid \Phi].C'$. Assume $[f \mid \Phi].C'(P_1) \xrightarrow{\varphi} B'_1$. Then either the prefix rule (5.3) was applied, i.e., $\varphi \in \Phi$ and

$$[f \mid \Phi].C'(P_1)(f) \xrightarrow{\varphi} [f' \mid \Phi \setminus \varphi].C'(P_1)(f/\varphi; f')$$

and thus also

$$[f \mid \Phi].C'(P_2)(f) \xrightarrow{\varphi} [f' \mid \Phi \setminus \varphi].C'(P_2)(\varphi; f'/f)$$

Or the concatenation rule (5.4) can be applied, i.e., $\varphi = \varphi_1; \varphi_2$ with $\varphi_1 \in \Phi$ and

$$\begin{aligned} C'(P_1)(\varphi_1/f) &\xrightarrow{\varphi_2} B'_1 && \implies \text{by a shorter derivation} \\ C'(P_2)(\varphi_1/f) &\xrightarrow{\varphi_2} B'_2, B'_1 \sim B'_2 && \implies \text{using the previous case} \\ C(P_2)(\varphi_1/f) &\xrightarrow{\varphi_2} B'_2, B'_2 \sim B'_2 \end{aligned}$$

where $C'(P_1)(\varphi_1/f) \equiv C'(\varphi_1/f)(P_1)$, i.e., trajectory is substituted in C' .

3. For all other cases (formats of C) the proof is standard, see, e.g., Milner [1989, p.99–101] or Brinksma [1988, p.68–69].

We have shown that for all operators strong bisimulation is a congruence. \square

B.2.1 Formats based proof

Also a formats based proof of a congruence can be provided. According to Middelburg [2001] it is enough to show that all rules are in *panth* format [Aceto et al., 2001] and *variables dependency graph* [Middelburg, 2001] is well founded.

It is not difficult to see that all SOS rules are in the *panth* format [Aceto et al., 2001, Middelburg, 2001, Mousavi, 2005]. Indeed, in each rule, all transitions in the premises end with distinct variables that do not occur in the left-hand-sides of the conclusions. We show it explicitly for several rules, and leave others as an exercise.

So, for example, a parallel composition (5.9) rules are in *panth* format. Indeed, the right side of each premise in (5.9a), (5.9b) and (5.9c) is a single variable, i.e., B'_1 and B'_2 in (5.9a), B'_1 in (5.9b), and B'_1 and B'_2 in (5.9c), respectively. Moreover, the source of conclusion of each rule contains only one functional symbol (parallel composition \parallel_A^H) and all transitions in the premises end with distinct variables (B'_1 and B'_2 in (5.9a), B'_1 in (5.9b), and B'_1 and B'_2 in (5.9c)) that do not occur in the left-hand-sides of the conclusions.

B.3 Proofs of Theorems 5.6.2 and 5.6.3

B.3.1 Proof of Theorem 5.6.2

In this appendix we prove the mini expansion law. We repeat formulation of Theorem 5.6.2 for the readers convenience.

Theorem B.3.1 (Mini expansion law). *Let Φ, Ψ be sets of trajectories such that $\forall \varphi, \psi \in \Phi \ T(\varphi) = T(\psi)$, $\forall \varphi, \psi \in \Psi \ T(\varphi) = T(\psi)$. Let \mathcal{T}_Φ and \mathcal{T}_Ψ be sets of trajectory qualifiers of Φ and Ψ , respectively. If $h_\Phi = \pi^{\mathcal{T}_\Phi}(h)$ and $h_\Psi = \pi^{\mathcal{T}_\Psi}(h)$, then*

$$[f \mid \Phi] . B(f) \parallel_A^H [g \mid \Psi] . C(g) = \tag{B.4a}$$

$$[h \mid \Phi \times_H \Psi]. \tag{B.4b}$$

$$\left([f' \mid \Phi \setminus h_\Phi] . B(h_\Phi ; f' / f) \right) \parallel_A^H [g' \mid \Psi \setminus h_\Psi] . C(h_\Psi ; g' / g) \tag{B.4c}$$

We will assume that Φ and Ψ do not contain ϵ , because if it does, then it can always be rewritten as $[f \mid \Phi \setminus \epsilon] . B(f) + B(\epsilon)$ according to (5.5).

Proof. The bisimulation relation which proves that the left hand and the right hand

sides of Equation (B.4) simulate each other, can be defined as follows

$$\begin{aligned} \mathcal{R} = & \left\{ \left([f \mid \Phi] . B(f) \parallel_A^H [g \mid \Psi] . C(g), \right. \right. \\ & [h \mid \Phi \times_H \Psi] . \\ & \left. \left([f' \mid \Phi \parallel h_\Phi] . B(h_\Phi ; f' / f) \parallel_A^H [g' \mid \Psi \parallel h_\Psi] . C(h_\Psi ; g' / g) \right) \right\} \\ & \mid \forall B \text{ is a process, } \forall \Phi \text{ is a set of named trajectories} \} \cup \text{Id}_\alpha \end{aligned} \quad (\text{B.5})$$

where Id_α is an identity function up to α -conversion.

We prove it by transforming the subexpressions and showing that both sides of the expression simulate each other.

We will call expression (B.4a) a left hand side (or an lhs) and (B.4b) and (B.4c) a right hand side (or an rhs).

The left hand side simulates the right hand side.

We will use an induction to prove that the left hand side simulates the right hand side.

Let

$$\bar{B}' = [f' \mid \Phi \parallel h_\Phi] . B(h_\Phi ; f' / f) \quad (\text{B.6a})$$

$$\bar{C}' = [g' \mid \Psi \parallel h_\Psi] . C(h_\Psi ; g' / g) \quad (\text{B.6b})$$

1. By construction $\Phi \times_H \Psi$ does not contain ϵ . Therefore, in one step ($h = 1$) only the trajectory prefix can be applied. Let $\chi \in \Phi \times_H \Psi$, then by construction of $\Phi \times_H \Psi$ there $\exists \varphi \in \bar{\Phi}, \exists \psi \in \bar{\Psi}$ such that $\chi = \varphi \times_H \psi$. Moreover, let $\Phi' = \Phi \parallel \varphi$ and $\Psi' = \Psi \parallel \psi$. By construction of composition of sets of trajectories and closure of sets of trajectories we get $(\Phi \times_H \Psi) \parallel \chi = ((\Phi \parallel \varphi) \times_H (\Psi \parallel \psi)) \cup \{\epsilon\} = (\Phi' \times_H \Psi') \cup \{\epsilon\}$.

Then

$$[h \mid \Phi \times_H \Psi] . (\bar{B}' \parallel_A^H \bar{C}') \xrightarrow{\chi} \quad (\text{B.7a})$$

$$[h' \mid (\Phi \times_H \Psi) \parallel \chi] . (\bar{B}' \parallel_A^H \bar{C}') (\chi ; h' / h) =$$

$$[h' \mid (\Phi \times_H \Psi) \parallel \chi] .$$

$$\begin{aligned} & \left([f' \mid \Phi \parallel \varphi ; h_\Phi] . B(\varphi ; h_\Phi ; f' / f) \parallel_A^H [g' \mid \Psi \parallel \psi ; h_\Psi] . C(\psi ; h_\Psi ; g' / g) \right) = \\ & [h' \mid (\Phi' \times_H \Psi') \cup \{\epsilon\}] . \end{aligned} \quad (\text{B.7b})$$

$$\left([f' \mid \Phi' \parallel h_\Phi] . B(\varphi ; h_\Phi ; f' / f) \parallel_A^H [g' \mid \Psi' \parallel h_\Psi] . C(\psi ; h_\Psi ; g' / g) \right) \quad (\text{B.7c})$$

If the rhs engages in χ , then correspondingly the lhs can engage in φ and ψ , respectively. Then we get

$$[f \mid \Phi] . B(f) \parallel_A^H [g \mid \Psi] . C(g) \xrightarrow{\chi} \quad (\text{B.8a})$$

$$[f' \mid \Phi \parallel \varphi] . B(\varphi ; f' / f) \parallel_A^H [g \mid \Psi \parallel \psi] . C(\psi ; g' / g) =$$

$$[f' \mid \Phi'] . B(\varphi ; f' / f) \parallel_A^H [g \mid \Psi'] . C(\psi ; g' / g) \quad (\text{B.8b})$$

We can distinguish two cases according to the concatenation rules (5.4), i.e., we can continue with (B.7b) or directly with (B.7c).

- It is easy to see that (B.7b) (with (B.7c)) and (B.8b) simulate each other, because if they take one step, we get the same structure up to α -conversion again. Then we can add both of them to bisimulation.
- If we choose to continue with (B.7c) part, then the resulting expression of (B.7) can be rewritten as follows:

$$[f' \mid \Phi'] . B(\varphi ; f' / f) \parallel_A^H [g' \mid \Psi'] . C(\psi ; g' / g) \quad (\text{B.9a})$$

It is easy to see that (B.9) and (B.8) are identical up to α -conversion.

Therefore, it was shown that in one step the left hand side simulates the right hand side.

2. Let the property holds for ($h = k > 1$). Then the derivation of $h = k + 1$ steps can be constructed only by applying the concatenation rule. Then the last step in the derivation tree is

$$\frac{[h \mid \Phi \times_H \Psi] . (\bar{B}' \parallel_A^H \bar{C}') \xrightarrow{\chi'} D'', D'' \xrightarrow{\chi''} D'}{[h \mid \Phi \times_H \bar{\Psi}] . (\bar{B}' \parallel_A^H \bar{C}') \xrightarrow{\chi} D'}$$

such that $\chi = \chi' ; \chi''$. By the induction hypothesis we know that for the lhs derivation tree

$$[f' \mid \bar{\Phi}] . \bar{B} \parallel_A^H [g' \mid \bar{\Psi}] . \bar{C} \xrightarrow{\chi'} E''$$

such that $(D'', E'') \in \mathcal{R}$. If $(D'', E'') \in \text{Id}_\alpha$ then it is proven. Otherwise we get the same form with the trajectory prefixes, and the same form of the proof can be applied. Furthermore, equation $D'' \xrightarrow{\chi''} D'$ has a similar but a shorter derivation tree.

The right hand side simulates the left hand side

The use a similar technique to show that the right hand side simulates the left hand side. For the case of ϵ the same explanation, as in previous case, holds.

We use an induction to prove that the right hand side simulates the left hand side.

1. In one step ($h = 1$) only the trajectory prefix can be applied. The lhs can take $\chi = \varphi \times_H \psi$ (see above for the details), then we get

$$[f \mid \Phi] . B(f) \parallel_A^H [g \mid \Psi] . C(g) \xrightarrow{\chi} \quad (\text{B.10a})$$

$$\begin{aligned} & [f' \mid \Phi \setminus \varphi] . B(\varphi ; f' / f) \parallel_A^H [g \mid \Psi \setminus \psi] . C(\psi ; g' / g) = \\ & [f' \mid \Phi'] . B(\varphi ; f' / f) \parallel_A^H [g \mid \Psi'] . C(\psi ; g' / g) \quad (\text{B.10b}) \end{aligned}$$

See explanations in the beginning of the part *The left hand side simulates the right hand side* for the details.

But then the rhs can take χ resulting in (B.9). It is easy to see that (B.9) and (B.10b) are identical up to α -conversion.

Therefore, it was shown that in one step the left hand side simulates the right hand side.

2. Let the property holds for ($h = k > 1$). Then the derivation of $h = k + 1$ steps can be constructed only by applying the concatenation rule. Then the last step in the derivation tree is

$$\frac{[f' \mid \bar{\Phi}] . \bar{B} \parallel_A^H [g' \mid \bar{\Psi}] . \bar{C} \xrightarrow{\chi'} E'', E'' \xrightarrow{\chi''} E'}{[f' \mid \bar{\Phi}] . \bar{B} \parallel_A^H [g' \mid \bar{\Psi}] . \bar{C} \xrightarrow{\chi} E'}$$

such that $\chi = \chi' ; \chi''$. By the induction hypothesis we know that for the rhs derivation tree

$$[h \mid \Phi \times_H \Psi] . (\bar{B}' \parallel_A^H \bar{C}') \xrightarrow{\chi} D'$$

such that $(D'', E'') \in \mathcal{R}$. If $(D'', E'') \in \text{Id}_\alpha$ then it is proven. Otherwise we get the same form with the trajectory prefixes, and the same form of the proof can be applied. Furthermore, equation $E'' \xrightarrow{\chi''} E'$ has a similar but a shorter derivation tree.

Bisimulation

Both sides simulate each other, therefore they are bisimilar. \square

B.3.2 Proof of Theorem 5.6.3

Theorem B.3.2 (Expansion law). *Let*

$$B = \sum_{i \in I} \mathbf{b}_i . B_i + \sum_{j \in J} [f_j \mid \Phi_j] . B_j, \quad C = \sum_{k \in K} \mathbf{c}_k . C_k + \sum_{l \in L} [g_l \mid \Psi_l] . C_l$$

for some terms B_i, B_j, C_k and C_l , actions \mathbf{b}_i and \mathbf{c}_k , trajectories $[f_j \mid \Phi_j]$ and $[g_l \mid \Psi_l]$ and the corresponding sets of qualifiers names \mathcal{T}_{Φ_j} and \mathcal{T}_{Ψ_l} , finite index sets $I \cap J = K \cap L = \emptyset$. Let $h_j = \pi^{\mathcal{T}_{\Phi_j}}(h)$ and $h_l = \pi^{\mathcal{T}_{\Psi_l}}(h)$. Then

$$B \parallel_A^H C = \tag{B.11}$$

$$\sum_{\substack{i \in I \\ \mathbf{b}_i \notin A}} \mathbf{b}_i . (B_i \parallel_A^H C) + \sum_{\substack{k \in K \\ \mathbf{c}_k \notin A}} \mathbf{c}_k . (B \parallel_A^H C_k) + \sum_{\substack{i \in I, k \in K \\ \mathbf{b}_i \in A, \\ \mathbf{b}_i = \mathbf{c}_k}} \mathbf{b}_i . (B_i \parallel_A^H C_k) + \tag{B.12}$$

$$\sum_{\substack{j \in J \\ l \in L}} [h \mid \Phi_j \times_H \Psi_l] . \tag{B.13}$$

$$\left([f_j \mid \Phi_j \setminus h_j] . B_j (h_j ; f_j' / f_j) \parallel_A^H [g_l \mid \Psi_l \setminus h_l] . C_l (h_l ; g_l' / g_l) \right) \tag{B.14}$$

We will assume that Φ_j and Ψ_j do not contain ϵ , because if it does, then it can always be rewritten as $[f \mid \Phi \setminus \epsilon] . B(f) + B(\epsilon)$ according to (5.5).

The proof of Theorem 5.6.3 is based on Theorem 5.6.2 and Milner [1989, p.96-97].

Proof. Continuous-time and discrete behaviours only influence each other, but do not interact with each other, i.e., they act at a different time. Therefore, in the parallel composition, continuous behaviours compose with each other, and, respectively, discrete

behaviours interact with each other. Consequently, the expansion law can be split in to two parts: a discrete and a continuous.

$$\text{Let } B = \sum_{i \in I} \mathbf{b}_i . B_i + \sum_{j \in J} [f_j \mid \Phi_j] . B_j \text{ and } C = \sum_{k \in K} \mathbf{c}_k . C_k + \sum_{l \in L} [g_l \mid \Psi_l] . C_l.$$

Discrete behaviour

The discrete part corresponds to the expansion law in the ordinary process algebras. The proof is similar to one in Milner [1989, p.96–97].

$$B \parallel_A^H C = \sum_{\substack{i \in I \\ \mathbf{b}_i \notin A}} \mathbf{b}_i . (B_i \parallel_A^H C) + \sum_{\substack{k \in K \\ \mathbf{c}_k \notin A}} \mathbf{c}_k . (B \parallel_A^H C_k) + \sum_{\substack{i \in I, k \in K \\ \mathbf{c}_k = \mathbf{b}_i \in A}} \mathbf{b}_i . (B_i \parallel_A^H C_k)$$

Continuous-time behaviour

The proof for the continuous-time behaviour follows from Theorem 5.6.2 and specifics of the interaction of continuous and discrete behaviour.

We will show that the components can be moved inside the sum.

$$\sum_{j \in J} [f_j \mid \Phi_j] . B_j \parallel_A^H C = \sum_{j \in J} \left([f_j \mid \Phi_j] . B_j \parallel_A^H C \right) \quad (\text{B.15})$$

There are two possible cases.

In the first case $C = \mathbf{a} . C'$, then there are only two possible evolutions on the both sides

$$\begin{aligned} \sum_{j \in J} [f_j \mid \Phi_j] . B_j \parallel_A^H \mathbf{a} . C' &\xrightarrow{\mathbf{a}} \sum_{j \in J} [f_j \mid \Phi_j] . B_j \parallel_A^H C' && \mathbf{a} \notin A \\ \sum_{j \in J} \left([f_j \mid \Phi_j] . B_j \parallel_A^H \mathbf{a} . C' \right) &\xrightarrow{\mathbf{a}} \sum_{j \in J} \left([f_j \mid \Phi_j] . B_j \parallel_A^H C' \right) && \mathbf{a} \notin A \end{aligned}$$

or deadlock, if $\mathbf{a} \in A$. Therefore both sides can simulate each other and are bisimilar.

If $C = [g \mid \Psi] . C'$, then

$$\sum_{j \in J} [f_j \mid \Phi_j] . B_j \parallel_A^H [g \mid \Psi] . C' = \sum_{j \in J} \left([f_j \mid \Phi_j] . B_j \parallel_A^H [g \mid \Psi] . C' \right)$$

Analogous induction as in Theorem 5.6.2 can be applied. We just show how it works for the base step.

When C starts with trajectory prefix, the same trajectories can be selected on the both sides.

- In the $\sum_{j \in J} [f_j \mid \Phi_j] . B_j \parallel_A^H [g \mid \Psi] . C'$ the system can choose any trajectory, where g synchronises with f_j .
- From the $\sum_{j \in J} \left([f_j \mid \Phi_j] . B_j \parallel_A^H C' \right)$ the system evolution is again defined as a choice over j of all trajectories, where f_j synchronises with g .

Let $\pi^{\text{T}(\varphi)}(\chi) = \varphi$, $\pi^{\text{T}(\psi)}(\chi) = \psi$ and $J' \subseteq J$ is a subset of trajectories that includes χ . Then, by taking χ we get

$$\begin{aligned} \sum_{j \in J} [f_j \mid \Phi_j] . B_j \parallel_A^H [g \mid \Psi] . C' &\xrightarrow{\chi} \\ [f' \mid \Phi_{j'} \setminus \varphi] . B_{j'} (\varphi ; f' / f_{j'}) \parallel_A^H [g' \mid \Psi \setminus \psi] . C' (\psi ; g' / g) & \quad j' \in J' \quad (\text{B.17a}) \end{aligned}$$

and

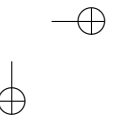
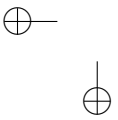
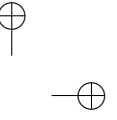
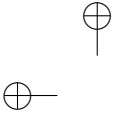
$$\sum_{j \in J} \left([f_j \mid \Phi_j] . B_j \parallel_A^H [g \mid \Psi] . C' \right) \xrightarrow{\alpha} [f' \mid \Phi_{j'} \setminus \varphi] . B_{j'} (\varphi ; f' / f_{j'}) \parallel_A^H [g' \mid \Psi \setminus \psi] . C' (\psi ; g' / g) \quad j' \in J' \quad (\text{B.18a})$$

It is easy to see that both resulting expressions are identical up to α -conversion.

Then we can repeat the proof of Theorem 5.6.2 showing that we get the same structure up to α -conversion, and it is already explained, why it happens.

Then the same technique can be applied for the construction with choice operator on both sides to show that both sides simulate each other.

Therefore, on both sides the system evolves in the same way. \square



C

Functions from Chapter 7

In this appendix we provide Table C.1 of all functions mentioned in Chapter 7 and give short explanation for functions for which pseudo code is not provided. Most of late are used to abstract from the chosen data types.

Table C.1: List of all functions from Chapter 7

Name	Ref	Description
BHPC_simulation	7.1	Main simulation function.
Initialise	7.3	Initialisation.
TakeTransition	7.2	Implements step of taking a transition.
NondetChoice	n.a.	Non-deterministically choose an element from the set of elements. In a simple case may provides <i>menu</i> for user to choose.
TakeContinuousTransition	7.5	Implements procedures necessary to take a continuous transition.
TakeDiscreteTransition	n.a.	Implements procedures necessary to take a discrete transition.
transfToNF	7.4	Transforms a process to a normal form.
isEmpty	n.a.	Syntactical function, checks is process expression empty.
InitialiseQualifiers	n.a.	Assigns initial values to corresponding qualifiers (and other variables, e.g., derivatives of qualifiers), implementation depends on the chosen data types.
isNormForm	n.a.	Syntactically checks, is a process expression is in the normal form.

Continued on next page

C. FUNCTIONS FROM CHAPTER 7

Table C.1 – continued from previous page		
Name	Ref	Description
setDeadlock	n.a.	Assigns to a process expression deadlock. Data types dependent.
setState	n.a.	Assigns new state from the old one and status changes. Data types dependent.
setProcess	n.a.	Assigns new state from the old one and process. Data types dependent.
rename	n.a.	Performs syntactical renaming on the actions to be taken. Data types dependent.
hide	n.a.	Performs syntactical hiding operation on the actions to be taken. Data types dependent.
replaceChoiceComp	n.a.	Replaces <i>i</i> 'th component of the choice by a new one. Data types dependent.
applyExpLaw	C.1	Straightforward application of modified expansion law (just two nested cycles and case). Data types dependent.
resolveRecursion	n.a.	Almost the same, as <code>initialise</code> , i.e., new process expression is assigned to the current state and the corresponding qualifiers initialised.
DefineEvents	n.a.	Takes a set of trajectory prefixes, extracts conditions and exit-conditions and generates halting conditions for numerical solvers. Data types and numerical solver dependent. Different halting conditions are supported by different solvers. E.g., in <code>BHAVE</code> prototype <code>MAPLE</code> is used as a numerical solver, and it accepts conditions only in special format (http://www.maplesoft.com). A simplified version of <code>DefineEvents</code> is used in Appendix D. All exit-conditions are collected, solved using <code>MAPLE</code> . If a solution for a qualifiers is an interval, then a halting value is nondeterministically chosen from the interval. If a solution is a single value, then this value is chosen.

Continued on next page

Table C.1 – continued from previous page		
Name	Ref	Description
CreateCombinations	n.a.	Takes a process expression in normal form and creates relevant combinations. The procedure is described in Section 7.2.6 A considerably simplified version of CreateCombinations is implemented in BHAVE prototype (Appendix D). It creates only one combination from a parallel composition of trajectory prefixes.
Solve	n.a.	Takes a set of trajectory prefixes and halting conditions, extracts equations, initial conditions and halting conditions and supplies it to a numerical solver. Data types and numerical solver dependent, i.e., different solvers may have different syntactical and semantic limitations. In some cases, the procedure can be complicated technically, e.g., when a solver works as a library, then it may be necessary to compile it together with equations. A version that uses MAPLE is implemented in BHAVE prototype. It defines instantiation of qualifiers, differential equations and halting conditions. Then MAPLE dsolve is used to solve them.
NondetChooseCombination	n.a.	Implements non-deterministic choice form a set of combinations created by CreateCombinations.
UpdateState	n.a.	Updates processes by replacing the corresponding components of choice.
UpdateSimState	n.a.	Updates state and status depending on the event.

To illustrate a potential complexity of the above mentioned functions we provide applyExpLaw Algorithm C.1. Functions isActionPrefix and isTrajPrefix return true if the parameter is an action prefix and a trajectory prefix or a parallel composition of trajectory prefixes, correspondingly. Function mergeQualifiers validates consistency of the qualifiers values and applies corresponding procedures to merge.

Most of functions appearing in Algorithm 7.4 were implemented in BHAVE prototype. The only exceptions are rename and hide. Moreover, discrete versions of all these functions (or corresponding techniques) are defined in Schonenberg [2006].

```

algorithm applyExpLaw (state', state'': STATE, A, H: SET, status: STATUS)
var state : STATE;
begin
  /* state'.p =  $\sum_{i \in I} elem.B_i$  where elem is an action-prefix or */
  /* a parallel composition of trajectory-prefixes or one trajectory prefix */
  state.t = state'.t;
  state.p =  $\sum \{ \}$ ;
  foreach lelem.lBi in state'.P do
    foreach relem.rBi in state''.P do
      if isTrajPref(lelem) and trajPref(relem)
      then
        state.P = state.P + (lelem.lBi  $\parallel_A^H$  relem.rBi);
      else
        if isAction(lelem) and isAction(relem)
        then
          case lelem, relem do
            lelem  $\in$  A, relem  $\in$  A:
              if lelem = relem
              then state.P = state.P + lelem.(lBi  $\parallel_A^H$  rBi);
            end
            lelem  $\notin$  A, relem  $\in$  A:
              state.P = state.P + lelem.(lBi  $\parallel_A^H$  relem.rBi);
            lelem  $\in$  A, relem  $\notin$  A:
              state.P = state.P + relem.(lelem.lBi  $\parallel_A^H$  rBi);
            lelem  $\notin$  A, relem  $\notin$  A:
              state.P = state.P + lelem.(lBi  $\parallel_A^H$  relem.rBi);
              state.P = state.P + relem.(lelem.lBi  $\parallel_A^H$  rBi);
          end
        else
          if isAction(lelem) and lelem  $\notin$  A
          then state.P = state.P + lelem.(lBi  $\parallel_A^H$  rBi);
        end
          if isAction(relem) and relem  $\notin$  A
          then state.P = state.P + relem.(lBi  $\parallel_A^H$  rBi);
        end
      end
    (state.Q, status) = mergeQualifiers(state'.Q, state''.Q, status);
  end
end
return (status, state);
end

```

Algorithm C.1: Transform to normal form

D

Bhave prototype

In this chapter we provide a concise description of a prototype of Behavioural Hybrid Process simulator (BHAVE prototype) that is a part of BHAVE toolset ¹. BHAVE toolset consists of several tools:

- BHPC Translator (BHPCC) [van Putten, 2006] is a parser of Behavioural Hybrid Process Calculus (BHPC) that translates BHPC specification to an internal format, and BHPC2Mod translator of sequential version of calculus to Modelica,
- DISCRETE BHAVE [Krilavičius and Schonenberg, 2005, Schonenberg, 2006] is a tool for discrete simulation of Behavioural Hybrid Process Calculus,
- BHAVE prototype is a prototype of Behavioural Hybrid Process Calculus simulation tool.

For more information about BHAVE toolset see Bhave prot, Krilavičius and Schonenberg [2005], Schonenberg [2006], van Putten [2006] and <http://fmt.cs.utwente.nl/tools/bhave>.

Theoretical background for BHAVE prototype is provided in Chapter 7. The tool was developed to validate the proposed techniques and to investigate what kind of technical problems may occur in the implementation of simulator. Therefore, the tool has rather limited functionality and performance. However, we present several examples demonstrating that even a prototype can be used to investigate interesting hybrid systems.

D.1 Functionality and input language

The tool accepts a subset of Behavioural Hybrid Process Calculus operators:

¹See <http://fmt.cs.utwente.nl/tools/bhave>.

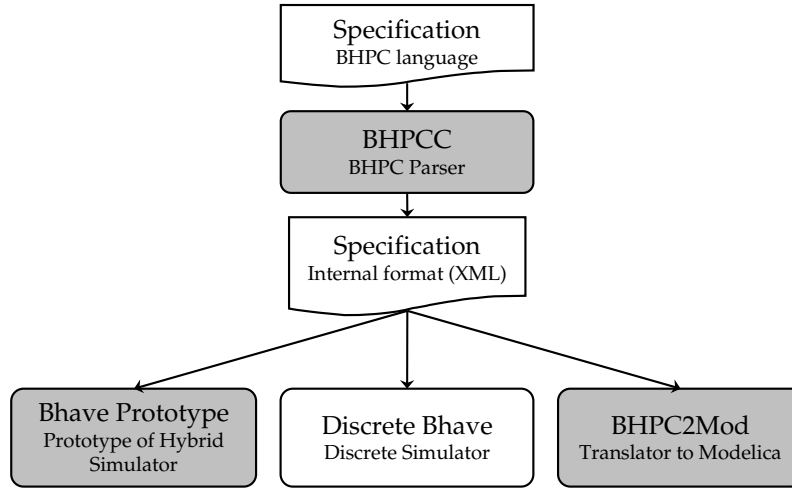


Figure D.1: BHAVE toolset

- Stop process (0).
- Action prefix (a.B).
- Trajectory prefix ($[q_1, \dots, q_m \mid \Phi \downarrow \mathcal{P}red \Downarrow \mathcal{P}red_{exit}] . B$). However, it is restricted to a certain class of continuous behaviours and conditions (limitations are mostly imposed by the specifics of MAPLE 9.5² that is used as an ODE solver)
 - Trajectories (Φ) are represented by ordinary differential equations (ODE) that are solvable by MAPLE.
 - Conditions ($\mathcal{P}red$) are always true.
 - Exit conditions ($\mathcal{P}red_{exit}$) are represented by the simple inequalities of type $x \ll = \gg e$, where x is a qualifier and e is an expression solvable by MAPLE. Moreover, more complex exit conditions can be given in a form $\bigvee_i \text{cond}(x_i)$ where x_i is a particular qualifier. In other words, exit conditions can be given as a disjunction over the exit conditions on the different variables.
- Hiding (new $w.B$) is ignored. However, user is informed about it, if it is encountered.
- Renaming ($B[\sigma]$) is ignored. However, user is informed about it, if it is encountered.
- Guards ($\langle \mathcal{P}red \rangle . B$) are given only by the simple inequalities solvable by MAPLE.
- A generalised choide $\sum_{i \in I} B_i$.

²<http://www.maplesoft.com/>

- Parallel composition ($B \parallel_A^H B$). Parallel simulation of trajectory prefixes is performed in a following way with consequent limitations:
 - All differential equations are collected from the participating processes.
 - All exit conditions are collected, grouped by qualifiers and simplified. E.g., if we have trajectory prefixes with exit conditions $\{h = 0\} \vee \{v > -10, v < 5\}$ and $\{h = 5\} \vee \{v > -5, v < 10\}$, correspondingly, then we get $\{v > -5, v < 5\}$.
 - Initial values assignments are collected.
 - Constructed system of differential equations with initial values and stop conditions is passed to MAPLE and simulated until any of exit conditions is satisfied or maximal simulation time is reached.
 - All trajectory prefixes stopped and simulation continues with succeeding processes, i.e., concatenation is not supported.
- Recursion (P), where re-initialisation parameters are given by simple expressions solvable by MAPLE.

All these restrictions mean that only part of the theory presented in Chapter 7 is tested. However, we present several examples, where we show that even the BHAVE prototype may be used to investigate quite interesting systems.

D.1.1 Simplified treatment of parameters

In BHAVE prototype we use a simplified treatment of processes parametrisation. We use parameters as shortcuts to initialise qualifiers. Moreover, we access the last simulation value using the corresponding qualifier identifier.

Components of parallel composition may have different qualifiers values (Section 7.2.2). Therefore, we propose a decision procedure to resolve it and check for the inconsistencies. Another choice would be to treat such behaviour as inconsistent.

Let Q_l, Q_r and Q_p be sets of qualifiers values of the left, right and parallel composition (parent) processes, correspondingly. Let Q is a resulting values set. Then the following cases are possible:

1. $Q_l = Q_r \implies Q = Q_l = Q_r$.
2. $Q_l \neq Q_r = Q_p \implies Q = Q_l$ and if some qualifiers are defined in Q_r and undefined in Q_l (it may happen, if), then their values are taken from Q_r .
3. $Q_r \neq Q_l = Q_p \implies Q = Q_r$ and if some qualifiers are defined in Q_l and undefined in Q_r , then their values are taken from Q_l .
4. $Q_l \neq Q_r \neq Q_p$ then for all corresponding $q_l \in Q_l, q_r \in Q_r, q_p \in Q_p$ and $q \in Q$:
 - (a) $q_l = q_r \implies q = q_l = q_r$,
 - (b) if q_r is undefined or $q_l \neq q_r = q_p \implies q = q_l$,
 - (c) if q_l is undefined or $q_r \neq q_l = q_p \implies q = q_r$,
 - (d) $q_r \neq q_l \neq q_p$ inconsistency is detected.

In other words, we take the new values and the case, when both subprocesses try to assign to the same qualifiers different values we treat as inconsistency. Moreover, if the new values is not consistent with the succeeding trajectory prefix, it should be detected when simulation of the trajectory prefix starts.

Example D.1.1. Consider process expression

$$[k, l, m \mid k' = 1, l' = 0, m' = -1 \Downarrow k \geq 3] \parallel^{k,l,m} \text{Controller}(2, l, m)$$

with $Q_p = \{k = 0, l = 1, m = 10\}$ and the process definition $\text{Controller}(k, l, m)$. Then by applying Algorithm 7.4 we get

$$Q_l = \{k = 0, l = 1, m = 10\}$$

$$Q_r = \{k = 2, l = 1, m = 10\}$$

There are several ways to resolve such situation. It can be treated as inconsistency or some procedure can be applied to resolve it. We choose the second option and apply the above defined procedure and get $Q_r = \{k = 2, l = 1, m = 10\}$. \square

An extended procedure can be applied. E.g., users can be warned about the occurrence of (2), (3), (4b), (4c) and (4d), and be allowed to choose how to proceed.

D.2 Technical implementation details

We give some technical implementation details.

- Language: C++ (standard). Compiler: Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.xx.xxxx. Additional libraries: Xerces-C++ 2.7.0 (a validating XML parser written in a portable subset of C++); maple.h (Open MAPLE). Code statistics: \sim 9klines (\sim 5klines of pure code). Documentation: Doxygen 1.4.5. Operating systems: Windows XP (however C++ code and Xerces-C should be portable to other operating systems, Open Maple portability was not investigated).
- Differential equations, expressions, inequalities solver: MAPLE 9.5.
- Input format: Behavioural Hybrid Process Calculus in the internal format [van Putten, 2006] provided by BHPCC (BHPC compiler) [van Putten, 2006].
- Output format: text file, with tabulation symbol separated columns containing time, values and actions.
- Visualisation: presently plots can be produced using Microsoft (R) Excel 2003 XY(Scatter) routine of Chart Wizard.

D.3 Examples

In this section we present several examples to show BHAVE prototype capabilities.

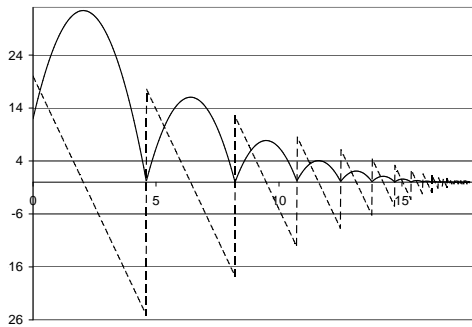


Figure D.2: Altitude and velocity of the ball

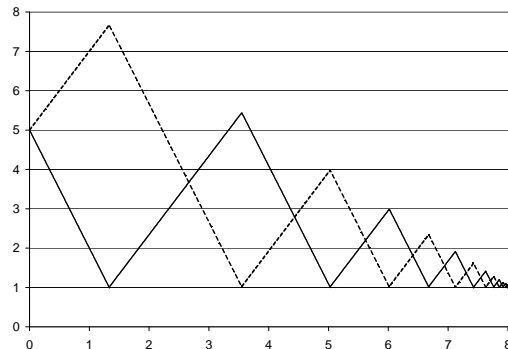


Figure D.3: Fluid level changes of two tanks

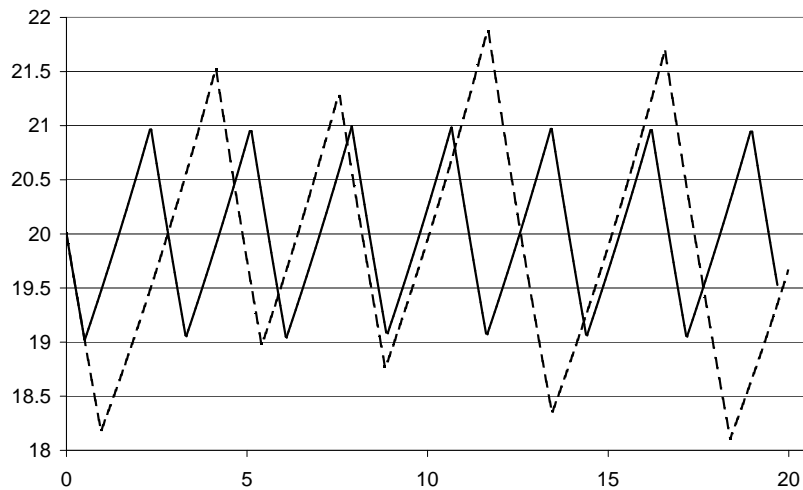


Figure D.4: Temperature changes for simple and upgraded thermostat

Example D.3.1 (Bouncing ball). The classical bouncing ball Example 5.8.1. The simulation results are depicted in Figure D.2. Velocity is depicted by the dashed line and altitude is depicted by the solid line. \square

Example D.3.2 (Simple and controlled thermostat). We reuse an example of the simple thermostat (Example 5.8.2).

Results of simulation of the simple and controlled thermostat are depicted in Figure D.4. The dashed line depicts evolution of the simple thermostat and the solid line depicts evolution of the coupled version. \square

Example D.3.3 (Two tanks). Consider two tanks model from Example 5.8.4. Then the simulation results for $d_{\text{left}} + d_{\text{right}} \geq l_{\text{in}}$ are visualised in Figure D.3.

We would like to note that both modular and simple versions were simulated, and the results were the same. Unfortunately, renaming and parts of parametrisation were carried-out manually in the modular version, because the tool does not support it

yet. □

D.4 Conclusions

In this appendix we concisely described BHAVE prototype and demonstrated how it simulates small examples of systems that exhibit hybrid behaviour. The tool is not intended for case studies or industrial applications, but more for experiments with different simulation algorithms and language constructs.

For more information about BHAVE prototype and BHAVE toolset see Bhave prot, Krilavičius and Schonenberg [2005], Schonenberg [2006], van Putten [2006] and <http://fmt.cs.utwente.nl/tools/bhave>.

Bibliography

- L. Aceto, W. J. Fokkink, and C. Verhoef. Structural operational semantics. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 3, pages 197–292. Elsevier Science Publishers, Ltd., 2001.
- R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theor. Comput. Sci.*, 138(1):3–34, 1995. ISSN 0304-3975.
- R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Grossman et al. [1993], pages 209–229. ISBN 3-540-57318-6.
- R. Alur, T. Dang, J. M. Esposito, R. B. Fierro, Y. Hur, F. Ivančić, V. Kumar, I. Lee, P. Mishra, G. J. Pappas, and O. Sokolsky. Hierarchical hybrid modeling of embedded systems. In *Proc. of the First International Workshop on Embedded Software(EMSOFT'01)*, pages 14–31. Springer, 2001. ISBN 3-540-42673-6. URL <http://citeseer.nj.nec.com/451283.html>.
- R. Alur and D. Dill. The theory of timed automata. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Proceedings of Real-Time: Theory in Practice*, volume 600 of LNCS, pages 45–73. Springer, June 1992. ISBN 3-540-55564-1.
- R. Alur, J. M. Esposito, M. Kim, V. Kumar, and I. Lee. Formal modeling and analysis of hybrid systems: A case study in multi-robot coordination. In *Proc. of the World Congress on Formal Methods in the Development of Computing Systems (FM'99)*, pages 212–232. Springer, 1999. ISBN 3-540-66587-0. URL citeseer.nj.nec.com/alur99formal.html.
- R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in CHARON. In Lynch and Krogh [2000], pages 6–19. ISBN 3-540-67259-1. URL <http://citeseer.nj.nec.com/article/alur00modular.html>.
- R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Trans. Soft. Eng.*, 22(3):181–201, 1996a. ISSN 0098-5589.
- R. Alur, T.A. Henzinger, and E.D. Sontag, editors. *Proceedings of the DIMACS/SYCON workshop on Hybrid systems III: verification and control*, volume 1066 of LNCS, Secaucus, NJ, USA, 1996b. Springer. ISBN 3-540-61155-X.
- R. Alur and G. J. Pappas, editors. *Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Proc.*, volume 2993 of LNCS, 2004. Springer. ISBN 3-540-21259-0.

BIBLIOGRAPHY

- T. Amnell, E. Fersman, P. Pettersson, W. Yi, and H. Sun. Code synthesis for timed automata. *Nordic J. of Computing*, 9(4):269–300, 2002. ISSN 1236-6064.
- T. Anderson, R. de Lemos, J. S. Fitzgerald, and A. Saeed. On formal support for industrial-scale requirements. In Grossman et al. [1993], pages 426–451. ISBN 3-540-57318-6.
- M. Andersson. *Object-Oriented Modeling and Simulation of Hybrid Systems*. PhD thesis, Department of Automatic Control, Lund Inst. of Technology, Sweden, December 1994.
- S. Andova. *Probabilistic Process Algebra*. PhD thesis, Technical University of Eindhoven (TU/e), 2002.
- P. J. Antsaklis, W. Kohn, M. D. Lemmon, A. Nerode, and S. Sastry, editors. *Hybrid Systems V*, volume 1567 of LNCS, 1999. Springer. ISBN 3-540-65643-X.
- P. J. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors. *Hybrid Systems II*, volume 999 of LNCS, 1995. Springer. ISBN 3-540-60472-3.
- P. J. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors. *Hybrid Systems IV*, volume 1273 of LNCS, 1997. Springer. ISBN 3-540-63358-8.
- E. Asarin, T. Dang, and O. Maler. d/dt a tool for reachability analysis of continuous and hybrid systems, 2001. URL http://www-verimag.imag.fr/~tdang/reach_nolcos.ps.gz.
- J. C. M. Baeten and W. P. Weijland. *Process algebra*. Cambridge University Press, New York, NY, USA, 1990. ISBN 0-521-40043-0.
- J.C.M. Baeten and J.A. Bergstra. Real Time Process Algebra. *Formal Aspects of Computing*, 3:142–188, 1991.
- J.C.M. Baeten and J.A. Bergstra. Process algebra with propositional signals. *Theor. Comput. Sci.*, 177(2):381–405, 1997.
- J.C.M. Baeten and C.A. Middelburg. *Process Algebra with Timing*. EATCS Monographs Series. Springer, 2002.
- P.I. Barton and C.K. Lee. Modeling, simulation, sensitivity analysis, and optimization of hybrid systems. *ACM Trans. Model. Comput. Simul.*, 12(4):256–289, 2002. ISSN 1049-3301.
- G. Behrmann, A. David, and K. G. Larsen. A tutorial on UPPAAL. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer, September 2004. URL <http://www.cs.auc.dk/~adavid/publications/21-tutorial.pdf>.
- A. Bemporad. *Modeling, control, and reachability analysis of discrete-time hybrid systems*. DISC, September 2003. URL http://www.dii.unisi.it/cgi-bin/ab_download.cgi?getpaper&paper=Bem03b. Lecture Notes of the DISC Course.

- A. Bemporad and M. Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, March 1999.
- A. Bemporad, F.D. Torrisi, and M. Morari. Discrete-time hybrid modeling and verification of the batch evaporator process benchmark. *European Journal of Control: Verification of Hybrid Systems*, 7(4):382–399, 2001.
- M.D. Di Benedetto and A.L. Sangiovanni-Vincentelli, editors. *Proc. of 4th International Workshop on Hybrid Systems: Computation and Control*, volume 2034 of LNCS, March 2001. Springer. ISBN 3-540-41866-0.
- J. Bengtsson, W.O.D. Griffioen, K.J. Kristoffersen, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Automated analysis of an audio control protocol using UPPAAL. *Journal of Logic and Algebraic Programming*, 52–53:163–181, July 2002.
- J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, Y. Wang, and C. Weise. New generation of UPPAAL. In *Int. Workshop on Software Tools for Technology Transfer*, June 1998.
- J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60(1/3):109–137, 1984.
- J.A. Bergstra and C.A. Middelburg. Process algebra for hybrid systems. Technical report, Dept. of Math. and Comp. Science, Technical University of Eindhoven (TU/e), P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands, 2003.
- J.A. Bergstra and C.A. Middelburg. Process algebra for hybrid systems. *Theor. Comput. Sci.*, 335(2/3):215–280, 2005.
- K. Berkenkötter and R. Kirner. Model-based testing of reactive systems. In Broy et al. [2005], pages 355–387. ISBN 3-540-26278-4.
- H.C. Bohnenkamp, H. Hermanns, R. Klaren, A. Mader, and Y.S. Usenko. Stochastic assessment of schedules in a lacquer production plant. In *Proceedings of the 1st International Conference on the Quantitative Evaluation of Systems (QEST'04)*, pages 28–37, September 2004.
- T. Bolognesi and H. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks*, 14:25–59, 1987.
- A.V. Borshchev, Y.B. Kolesov, and Y.B. Senichenkov. Java engine for UML based hybrid state machines. In *2000 Winter Simulation Conference (WSC'00)*, Orlando, Florida, USA, December 2000. URL <http://www.xjtek.com/files/papers/javaengine2000.pdf>.
- M. Branicky. Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Trans. Autom. Control*, 43:475–482, April 1998.
- M. S. Branicky, T.A. Johansen, I. Petersen, and E. Frazzoli. On-line techniques for behavioral programming. In *Proc. of 39th IEEE Conf. on Decision and Control*, pages 1840–1845, Sydney, Australia, December 2000. IEEE Computer Society Press. URL <http://citeseer.nj.nec.com/420838.html>.

BIBLIOGRAPHY

- M.S. Branicky. Stability of hybrid systems: State of the art. In *Proc. of 36th IEEE Conf. on Decision and Control*, pages 120–125, San Diego, CA, December 1997. IEEE Computer Society Press.
- M.S. Branicky, V.S. Borkar, and S.K. Mitter. A unified framework for hybrid control: Background, model, and theory. In *Proc. of 33th IEEE Conf. on Decision and Control*, Lake Buena Vista, FL, December 1994. IEEE Computer Society Press.
- M.S. Branicky and S.E. Mattsson. Simulation of hybrid systems. In Antsaklis et al. [1997], pages 31–56. ISBN 3-540-63358-8.
- K.E. Brenan, S.L. Campbell, and L.R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Springer, 1991. ISBN 0-387-97502-0.
- E. Brinksma. *On the Design of Extended Lotos*. PhD thesis, University of Twente, 1988.
- E. Brinksma and T. Krilavičius. Behavioural hybrid process calculus. Technical Report TR-CTIT-05.45, CTIT, University of Twente, 2005. URL http://www.cs.utwente.nl/~krilaviciust/publications/BHPC_techrep.pdf.
- L. Brandán Briones and E. Brinksma. A test generation framework for quiescent real-time systems. In *Formal Approaches to Software Testing: 4th International Workshop, FATES*, volume 3395/2005, 2004. ISBN 3-540-25109-X. URL <http://fmt.cs.utwente.nl/research/testing/files/BBB04.ps.gz>.
- L. Brandán Briones and E. Brinksma. Testing real-time multi input-output systems, November 2005. URL <http://fmt.cs.utwente.nl/research/testing/files/BBB05.ps.gz>.
- J. F. Broenink. Introduction to physical systems modelling with bond graphs, 1999. DRAFT (version 2, 20-5-99), to be published in the SiE Whitebook on Simulation methodologies, probably 1999.
- J.F. Broenink and P.B.T. Weustink. A combined system simulator for mechatronic systems. In *Proc. of Modeling and Simulation (ESM'96)*, pages 225–229, Budapest, Hungary, 1996. Publishing SCS Europe, Ghent. ISBN 1-56555-097-8.
- C. Brooks, A. Cataldo, E.A. Lee, J. Liu, X. Liu, S. Neuendorffer, and H. Zheng. HyVISUAL: A hybrid system visual modeler. Technical Report Technical Memorandum UCB/ERL M04/18, University of California, Berkeley, CA 94720, June 2004. URL <http://ptolemy.eecs.berkeley.edu/publications/papers/04/hyvisual>.
- M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors. *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of LNCS, 2005. Springer. ISBN 3-540-26278-4.
- L. Carloni, M.D. Di Benedetto, R. Passerone, A. Pinto, and A. Sangiovanni-Vincentelli. Modeling techniques, programming languages and design toolsets for hybrid systems. Technical Report Deliverable DHS4-5-6, Project IST-2001-38314 COLUMBUS, 2004. URL http://www.columbus.gr/documents/public/WPHS/Columbus_DHS3_0_2_Cover.pdf.

- F.E. Cellier. *Continuous System Modeling*. Springer, 1991. ISBN 0-387-97502-0.
- E.M. Clarke and J.M. Wing. Formal methods: state of the art and future directions. *ACM Comput. Surv.*, 28(4):626–643, 1996. ISSN 0360-0300.
- P.J.L. Cuijpers. *Hybrid Process Algebra*. PhD thesis, Technical University of Eindhoven (TU/e), 2004.
- P.J.L. Cuijpers, J.F. Broenink, and P.J. Mosterman. Constitutive hybrid processes. In *In proc. of Conference on Conceptual Modeling and Simulation (CSM 2004)*, Genoa, Italy, 2004.
- P.J.L. Cuijpers and M.A. Reniers. Hybrid process algebra. Technical report, Dept. of Math. and Comp. Science, Technical University of Eindhoven (TU/e), P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands, 2003.
- P.J.L. Cuijpers and M.A. Reniers. Hybrid process algebra. *JLAP*, 62(2):191–245, 2005.
- F. A. Cuzzola and M. Morari. A generalized approach for analysis and control of discrete-time piecewise affine and hybrid systems. In Benedetto and Sangiovanni-Vincentelli [2001], pages 189–203. ISBN 3-540-41866-0.
- P.R. D’Argenio. *Algebras and Automata for Timed and Stochastic Systems*. PhD thesis, Department of Computer Science, University of Twente, November 1999.
- P.R. D’Argenio and E. Brinksma. A calculus for timed automata (Extended abstract). In B. Jonsson and J. Parrow, editors, *Proceedings of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems*, Uppsala, Sweden, volume 1135 of LNCS, pages 110–129. Springer, 1996.
- P.R. D’Argenio, H. Hermanns, and J.-P. Katoen. On generative parallel composition. *Electronic Notes in Theoretical Computer Science*, 22:25, 1999.
- P.R. D’Argenio, J.-P. Katoen, and E. Brinksma. A stochastic automata model and its algebraic approach. In E. Brinksma and A. Nymeyer, editors, *Proc. of 5th International Workshop on Process Algebras and Performance Modeling, PAPM’97*, Enschede, The Netherlands, number 97-14 in Technical Report CTIT, pages 1–16. University of Twente, 1997.
- A. David and W. Yi. Modelling and analysis of a commercial field bus protocol. In *Proceedings of the 12th Euromicro Conference on Real Time Systems*, pages 165–172. IEEE Computer Society Press, 2000. ISBN 0-7695-0734-4.
- G.N. Davrazos and N.T. Koussoulas. A review of stability results for switched and hybrid systems. In *Proc. of 9th Mediterranean Conf. on Control and Automation*, Dubrovnik, Croatia, June 2001.
- B. De Schutter and W.P.M.H. Heemels. *Modeling and Control of Hybrid Systems*. DISC, September 2004. Lecture Notes of the DISC Course.
- B. De Schutter and B. De Moor. The extended linear complementarity problem. *Math. Program.*, 71(3):289–325, 1995. ISSN 0025-5610.

BIBLIOGRAPHY

- B. De Schutter and T. J. J. van den Boom. Model predictive control for max-min-plus-scaling systems – efficient implementation. In *Proc. of the Sixth Int. Workshop on Discrete Event Systems (WODES'02)*, page 343. IEEE Computer Society Press, 2002a. ISBN 0-7695-1683-1.
- B. De Schutter and T.J.J. van den Boom. Model predictive control for max-min-plus-scaling systems. In *Proc. of the 2001 American Control Conference (ACC'2001)*, pages 319–324, Arlington, Virginia, June 2001.
- B. De Schutter and T.J.J. van den Boom. Model predictive control for max-min-plus-scaling systems — Efficient implementation. In M. Silva, A. Giua, and J.M. Colom, editors, *Proceedings of the 6th International Workshop on Discrete Event Systems (WODES'02)*, pages 343–348, Zaragoza, Spain, October 2002b.
- R. DeCarlo, M. Branicky, S. Pettersson, and B. Lennartson. Perspectives and results on the stability and stabilizability of hybrid systems. *Proc. of IEEE, Special Issue on Hybrid Systems*, July 2000. URL citeseer.ist.psu.edu/decarlo00perspectives.html.
- A. Deshpande, A. Göllü, and L. Semenzato. The SHIFT programming language and run-time system for dynamic networks of hybrid automata. Technical Report UCB-ITS-PRR-97-7, Dept. of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA94720, 1997.
- D.Liberzon and A.S.Morse. Basic problems in stability and design of switched systems. *IEEE Control Systems Mag.*, 19(5):59–70, October 1999.
- H. Eertink. *Simulation Techniques for the Validation of LOTOS Specifications*. PhD thesis, University of Twente, 1994.
- T. Ernst, S. Jähnichen, and M. Klose. The architecture of the Smile/M simulation environment. In A. Sydow, editor, *In proc. of 15th IMACS world congress on Scientific Computation, Modelling and Applied Mathematics*, volume 6, Berlin, 1997. Wissenschaft und Technik Verlag.
- G. Fabian. *A Language and Simulator for Hybrid Systems*. PhD thesis, Technical University of Eindhoven (TU/e), 1999. URL <http://se.wtb.tue.nl/~vanbeek/pub/phdfabia.pdf>.
- G. Fabian, D. van Beck, and J. Rooda. Integration of the discrete and the continuous behavior in the hybrid chi simulator. In *Proc. 1998 Eur. Simulation Multiconf.*, pages 252–257, 1998.
- A. Fehnker. Scheduling a steel plant with timed automata. In *RTCSA'99*. IEEE Computer Society Press, 1999.
- R.W. Floyd and R. Beigel. *The Language of Machines, An Introduction to Computability and Formal Languages*. Computer Science Press, 1994.
- P. Fritzson, P. Aronsson, P. Bunus, V. Engelson, L. Saldamli, H. Johanson, and A. Karström. The open source Modelica project. In *Proc. of 2nd International Modelica Conference*, pages 297–306, 2002.

- P. Fritzson and V. Engelson. Modelica - a unified object-oriented language for system modelling and simulation. In *Proc. of the 12th European Conference on Object-Oriented Programming (ECCOP'98)*, pages 67–90. Springer, 1998. ISBN 3-540-64737-6.
- B. Gebremichael, T. Krilavičius, and Y.S. Usenko. A formal model of a car periphery supervision system in UPPAAL. In *Proceedings of Workshop on Discrete Event Systems (WODES'04)*, pages 433–438, Reims, France, September 2004.
- R. L. Grossman and R. G. Larson. An algebraic approach to hybrid systems. *Theor. Comput. Sci.*, 138(1):101–112, 1995. ISSN 0304-3975.
- R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors. *Hybrid Systems*, volume 736 of LNCS, 1993. Springer. ISBN 3-540-57318-6.
- G. Hamon and J. Rushby. An operational semantics for STATEFLOW. In M. Wermelinger and T. Margaria-Steffen, editors, *FASE 2004*, LNCS, pages 229–243, February 2004.
- D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987. URL <http://citeseer.nj.nec.com/article/harel187statecharts.html>.
- A. Hassibi, S.P. Boyd, and J.P. How. A class of Lyapunov functionals for analyzing hybrid dynamical systems. In *Proc. of American Control Conference*, volume 4, pages 2455–2460, San Diego, CA, June 1999.
- K. Havelund, K.G. Larsen, and A. Skou. Formal verification of a power controller using the real-time model checker UPPAAL. In *5th International AMAST Workshop on Real-Time and Probabilistic Systems*, 1999.
- S. Hedlund. Computational methods for hybrid systems. Licentiate thesis ISRN LUTFD2/TFRT--3225--SE, Department of Automatic Control, Lund Institute of Technology, Sweden, September 1999.
- W.P. Maurice H. Heemels, B. De Schutter, and A. Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37(7):1085–1091, 2001.
- W.P.M.H. Heemels. *Linear Complementarity Systems: A Study in Hybrid Dynamics*. PhD thesis, Technical University of Eindhoven (TU/e), 1999.
- W.P.M.H. Heemels, J.M. Schumacher, and S. Weiland. Linear complementarity systems. *SIAM J. Appl. Math.*, 60(4):1234–1269, 2000. ISSN 0036-1399.
- T.A. Henzinger. The theory of hybrid automata. In *LICS 1996*, pages 278–292. IEEE Computer Society Press, July 1996.
- T.A. Henzinger. MASACCIO: A formal model for embedded components. In J. van Leeuwen, O. Watanabe, M. Hagiya, P. D. Mosses, and T. Ito, editors, *IFIP TCS*, volume 1872 of LNCS, pages 549–563, Sendai, Japan, August 2000. Springer. ISBN 3-540-67823-9.
- T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: the next generation. In *Proc. of 16th IEEE Real-Time Systems Symp.*, pages 56–65. IEEE Computer Society Press, December 1995. ISBN 0-8186-7337-0.

BIBLIOGRAPHY

- T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTECH: a model checker for hybrid systems. In O. Grumberg, editor, *CAV'97*, volume 1254 of *LNCS*, pages 460–463. Springer, June 1997. ISBN 3-540-63166-6.
- T.A. Henzinger, Z. Manna, and A. Pnueli. Towards refining temporal specifications into hybrid systems. In Grossman et al. [1993], pages 60–67. ISBN 3-540-57318-6.
- T.A. Henzinger, M. Minea, and V. Prabhu. Assume-guarantee reasoning for hierarchical hybrid systems. In Benedetto and Sangiovanni-Vincentelli [2001], pages 275–290. ISBN 3-540-41866-0.
- T.A. Henzinger and V. Rusu. Reachability verification for hybrid automata. In Henzinger and Sastry [1998], pages 190–204. ISBN 3-540-64358-3. URL citeseer.nj.nec.com/henzinger98reachability.html.
- T.A. Henzinger and S. Sastry, editors. *Hybrid Systems: Computation and Control, First International Workshop, HSCC'98, Berkeley, California, USA, April 13-15, 1998, Proc.*, volume 1386 of *LNCS*, 1998. Springer. ISBN 3-540-64358-3.
- H. Hermanns. *Interactive Markov Chains And the Quest for Quantified Quality*. Springer, 1998.
- M. Heymann, F. Lin, and G. Meyer. Synthesis of minimally restrictive legal controllers for a class of hybrid systems. In Antsaklis et al. [1997], pages 134–159. ISBN 3-540-63358-8.
- C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978. ISSN 0001-0782.
- C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., 1985.
- A.T. Hofkamp. *Reactive machine control, a simulation approach using χ* . PhD thesis, Technical University of Eindhoven (TU/e), 2001.
- J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Pub.Comp., Inc., 2001.
- R. Huuck, B. Lukoschus, and Y. Lakhnech. Verifying untimed and timed aspects of the experimental batch plant. *European Journal of Control: Verification of Hybrid Systems*, 7(4):400–415, 2001.
- ITU-T. Recommendation Z.120. Message Sequence Charts. Technical Report Z-120, International Telecommunication Union – Standardization Sector, Genève, 2000.
- B. Jacobs. Object-oriented hybrid systems of coalgebras plus monoid actions. *Texts in TCS. An EATCS Series*, 239(1):41–95, 2000.
- K.H. Johansson, J. Lygeros, S. Sastry, and M. Egerstedt. Simulation of Zeno hybrid automata. In *Proc. of 38th IEEE Conf. on Decision and Control*, pages 3538–3543, Phoenix, AZ, December 1999. IEEE Computer Society Press. ISBN 0-7803-5250-5.

- M. Johansson and A. Rantzer. Computation of piecewise quadratic Lyapunov functions for hybrid systems. *IEEE Trans. Autom. Control*, 43(4):555–559, 1998. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=664157.
- D.S. Jovanovic, B. Orlic, G.K. Liet, and J.F. Broenink. gCSP: A graphical tool for designing CSP systems. In I.R. East, D. Duce, M. Green, J.M.R. Martin, and P.H. Welch, editors, *Communicating Process Architectures 2004*, pages 233–252, 2004. ISBN 1-58603-458-8.
- A.A. Julius. *On Interconnection and Equivalence of Continuous and Discrete Systems: A Behavioral Perspective*. PhD thesis, Systems Signals and Control Group, University of Twente, 2005.
- A.A. Julius, S.N. Strubbe, and A.J. van der Schaft. Compositional modeling of hybrid systems with hybrid behavioral automata. *Submitted to the HSCC 2003.*, 2002.
- J.-P. Katoen. *Concepts, Algorithms and Tools for Model Checking*. Universität Erlangen-Nürnberg, 1999.
- M. Kaufmann, P. Manolios, and J. S. Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, June 2000. ISBN 0-7923-7744-3.
- D.K. Kaynar, A. Chefter, L. Dean, S. Garland, N. Lynch, T.N. Win, and A. Ramirez-Robredo. The IOA simulator. Technical Report MIT-LCS-TR-843, MIT Laboratory for Computer Science, Cambridge, MA, July 2002.
- T.-J. Koo, G. J. Pappas, and S. Sastry. Mode switching synthesis for reachability specifications. In Benedetto and Sangiovanni-Vincentelli [2001]. ISBN 3-540-41866-0. URL citeseer.nj.nec.com/koo01mode.html.
- M. Kourjanski and P. Varaiya. Stability of hybrid systems. In Alur et al. [1996b], pages 413–423. ISBN 3-540-61155-X. URL citeseer.nj.nec.com/kourjanski95stability.html.
- X.D. Koutsoukos and P.J. Antsaklis. Design of stabilizing switching control laws for discrete- and continuous-time linear systems using piecewise-linear Lyapunov functions. *Int. Journal Control*, 75(12):932–945, 2002.
- S. Kowalewski. Description of VHS case study 1 "Experimental Batch Plant". Draft. University of Dortmund, Germany, July 1998.
- S. Kowalewski and O. Stursberg. The batch evaporator: A benchmark example for safety analysis of processing systems under logic control. In *Proc. 4th Workshop on Discrete Event Systems (WODES)*, pages 302–307. IEE, London, 1998. URL <http://www-verimag.imag.fr/VHS/year1/cs11d.ps>.
- S. Kowalewski, O. Stursberg, and N. Bauer. An experimental batch plant as a test case for the verification of hybrid systems. *European Journal of Control: Verification of Hybrid Systems*, 7(4):366–381, 2001.

BIBLIOGRAPHY

- S. Kowalewski, O. Stursberg, M. Fritz, H. Graf, I. Hoffmann, J. Preußig, S. Simon, H. Treseler, and M. Remelhe. A case study in tool-aided analysis of discretely controlled continuous systems: the two tanks problem. In Antsaklis et al. [1999], pages 163–187. ISBN 3-540-65643-X. URL citeseer.nj.nec.com/stursberg97case.html.
- T. Krilavičius and H. Schonenberg. Discrete simulation of behavioural hybrid process calculus. In P. M. E. Bra and J. J. van Wijk, editors, *IFM2005 Doctoral Symposium on Integrated Formal Methods*, pages 33–38, Eindhoven, Netherlands, November 2005. Dept. of Math. and Comp. Science, Technical University of Eindhoven (TU/e).
- B.H. Krogh and A. Chutinan. Hybrid systems: Modeling and supervisory control. In P.M. Frank, editor, *Advances in Control*, LNCS. Springer, 1999. URL <http://www.contrib.andrew.cmu.edu/~ac4c/papers/ecc99.ps>.
- M. Kvasnica, P. Grieder, and M. Baotić. Multi-Parametric Toolbox (MPT), 2004. URL <http://control.ee.ethz.ch/~mpt/>.
- G. Labinaz, M.M. Bayoumi, and K. Rudie. Modeling and control of hybrid systems: A survey. In *IFAC 13th Triennial World Congress*, San Francisco, CA, USA, 1996. URL citeseer.nj.nec.com/labinaz96modeling.html.
- L. Lamport. Hybrid Systems in TLA+. In Grossman et al. [1993], pages 77–102. ISBN 3-540-57318-6. URL citeseer.nj.nec.com/lamport93hybrid.html.
- R. Langerak, J.W. Polderman, and T. Krilavičius. Stability analysis for hybrid automata using conservative gains. In S. Engell, H. Guéguen, and J. Zayton, editors, *Proceedings of Conference on Analysis and Design of Hybrid Systems (ADHS'03)*, pages 377–382, 2003a.
- R. Langerak, J.W. Polderman, and T. Krilavičius. Stability analysis for hybrid automata using optimal Lyapunov functions. In *Proc. of Int. Conf. on Dynamical System Modelling and Stability Investigation*, page 420, Kyiv, Ukraine, May 2003b.
- K. Larsen, M. Mikučionis, and B. Nielsen. Real-time system testing on-the-fly. In K. Sere, M. Walden, and A. Karlsson, editors, *The 15th Nordic Workshop on Programming Theory (NWPT)*, Åbo Akademi University, Turku, Finland, October 2003. Extended abstract.
- E.A. Lee. Overview of the Ptolemy project. Technical Report Technical Memorandum No. UCB/ERL M03/25, University of California, Berkeley, CA 94720, June 2004. URL <http://ptolemy.eecs.berkeley.edu/publications/papers/03/overview/>.
- E.A. Lee and H. Zheng. Operational semantics of hybrid systems. In *Hybrid Systems: Computation and Control: 8th International Workshop, HSCC*, LNCS, pages 25–53, February 2005.
- J.A. Levine. Sampling-based planning for hybrid systems. Technical report, Dept. of Electrical Engineering and Computer Science, Case Western Reserve University, September 2003. M.Sc. thesis.

- M. Lindahl, P. Pettersson, and W. Yi. Formal Design and Analysis of a Gearbox Controller. *International Journal on Software Tools for Technology Transfer*, 3(3):353–368, 2001.
- P. Linz. *An Introduction to Formal Languages and Automata*. Jones and Bartlett pub., third edition, 2001.
- J. Liu and E.A. Lee. A component-based approach to modeling and simulating mixed-signal and hybrid systems. *ACM Trans. Model. Comput. Simul.*, 12(4):343–368, 2002. ISSN 1049-3301.
- J. Liu, X. Liu, T. J. Koo, B. Sinopolia, S. Sastry, and E.A. Lee. Hierarchical hybrid system simulation. In *Proc. of 38th IEEE Conf. on Decision and Control*, Phoenix, AZ, December 1999. IEEE Computer Society Press.
- J. Lygeros, D. Godbole, and S. Sastry. A design framework for hierarchical, hybrid control. Technical report, Intelligent Machines and Robotic Laboratory, University of California, Berkeley, 1997.
- J. Lygeros, K. H. Johansson, S.N. Simić, J. Zhang, and S. S. Sastry. Dynamical properties of hybrid automata. *IEEE Trans. Autom. Control*, 48(1):2–17, 2003.
- J. Lygeros and S. Sastry. Hybrid systems: Modeling, analysis & control, ee291, 1999. URL <http://paleale.eecs.berkeley.edu/~lygeros/Teaching/ee291E.html>.
- J. Lygeros, C. Tomlin, and S. Sastry. On controller synthesis for nonlinear hybrid systems. In *Proc. of 37th IEEE Conf. on Decision and Control*, pages 2101–2106, Tampa, FL, December 1998. IEEE Computer Society Press.
- J. Lygeros, C. Tomlin, and S. Sastry. Controllers for reachability specifications for hybrid systems. *Automatica*, 35(3), March 1999. URL citeseer.nj.nec.com/lygeros99controller.html.
- N.A. Lynch. Modelling and verification of automated transit systems, using timed automata, invariants and simulations. In Alur et al. [1996b], pages 449–463. ISBN 3-540-61155-X.
- N.A. Lynch and B.H. Krogh, editors. *Proc. of Third Int. Workshop on Hybrid Systems: Computation and Control*, volume 1790 of LNCS, March 2000. Springer. ISBN 3-540-67259-1.
- N.A. Lynch, R. Segala, and F.W. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, 2003. URL <http://www.cs.kun.nl/ita/publications/papers/fvaan/HIOA.html>.
- N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.
- A. Mader, E. Brinksma, H. Wupper, and N. Bauer. Design of a PLC control program for a batch plant - VHS case study 1. *European Journal of Control: Verification of Hybrid Systems*, 7(4):416–439, 2001.

BIBLIOGRAPHY

- O. Maler and A. Pnueli, editors. *Hybrid Systems: Computation and Control, 6th International Workshop, HSCC 2003 Prague, Czech Republic, April 3-5, 2003, Proc.*, volume 2623 of LNCS, 2003. Springer. ISBN 3-540-00913-2.
- A.N. Michel. Recent trends in the stability analysis of hybrid dynamical systems. *IEEE Trans. on Circuits and Systems - I. Fundamental theory and Applications*, 46(1):120–134, 1999. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=739260.
- C.A. Middelburg. Variable binding operators in transition system specifications. *JLAP*, 47(1):15–45, 2001.
- D. Mignone, G. Ferrari-Trecate, and M. Morari. Stability and stabilization of piecewise affine and hybrid systems: an LMI approach. In *Proc. of 39th IEEE Conf. on Decision and Control*, pages 504–509, Sydney, Australia, December 2000. IEEE Computer Society Press. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=912814.
- R. Milner. *A Calculus of Communicating Systems*, volume 92 of LNCS. Springer, 1980.
- R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., 1989. ISBN 0-13-115007-3.
- R. Milner. *Communicating and Mobile Systems: The π -calculus*. Cambridge University Press, 1999. ISBN 0-521-65869-1.
- Modelica - A Unified Object Oriented Language for Physical Systems Modeling: Language Specification*. Modelica Association, February 2005. URL <http://www.modelica.org/documents/ModelicaSpec22.pdf>.
- M. Morari, M. Baotic, and F. Borrelli. Hybrid systems modeling and control. *European Journal of Control*, 9(2–3):177–189, 2003.
- P. Mosterman. An overview of hybrid simulation phenomena and their support by simulation packages. In Vaandrager and van Schuppen [1999], pages 165–177. ISBN 3-540-65734-7.
- P.J. Mosterman and G. Biswas. A Hybrid Modeling and Simulation Methodology for Dynamic Physical Systems. *SIMULATION*, 78(1):5–17, 2002. URL <http://sim.sagepub.com/cgi/content/abstract/78/1/5>.
- M. Mousavi. *Structuring Structural Operational Semantics*. PhD thesis, Technical University of Eindhoven (TU/e), 2005.
- W. Mueller-Wittig, R. Jegathesea, S. Meehae, J. Quick, W. Haibinand, and Z. Yongmin. Virtual factory - highly interactive visualisation for manufacturing. In *In proc. of the Winter Simulation Conference*, volume 2, pages 1061–1064, December 2002. ISBN 0-7803-7614-5.
- P. Niebert and S. Yovine. Computing efficient operation schemes for chemical plants in multi-batch mode. *European Journal of Control: Verification of Hybrid Systems*, 7(4): 440–453, 2001.

- R. Nikoukhah and S. Steer. *Scicos - A Dynamic System Builder and Simulator User's Guide*. INRIA, 1997.
- M. Otter and F.E. Cellier. *Software for Modeling and Simulating Control Systems*, pages 415–428. CRC Press, Boca Raton, FL, 1995.
- S. Owre and N. Shankar. Writing PVS proof strategies. In M. Archer, B. Di Vito, and C. Muñoz, editors, *Design and Application of Strategies/Tactics in Higher Order Logics (STRATA 2003)*, number CP-2003-212448 in NASA Conference Publication, pages 1–15, Hampton, VA, September 2003. NASA Langley Research Center. The complete proceedings are available at <http://research.nianet.org/fm-at-nia/STRATA2003/>.
- S. Pettersson and B. Lennartson. Stability and robustness for hybrid systems. In *Proc. of 35th IEEE Conf. on Decision and Control*, pages 1202–1207, Kobe, Japan, December 1996. IEEE Computer Society Press. URL citeseer.ist.psu.edu/pettersson96stability.html.
- M. Philippe, V.-R. Claire, and G. Gérard. Optimal control of hybrid dynamical systems with the maximum principle: Application to a non linear chemical process. In *Proc. of 39th IEEE Conf. on Decision and Control*, Sydney, Australia, December 2000. IEEE Computer Society Press.
- G.D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981. URL citeseer.ist.psu.edu/plotkin81structural.html.
- G.D. Plotkin. The origins of structural operational semantics, 2003. URL citeseer.ist.psu.edu/plotkin03origins.html. *Journal of Functional and Logic Programming*, 2003. forthcoming.
- J.W. Polderman and J. C. Willems. *Introduction to Mathematical Systems Theory: a behavioral approach*. Springer, 1998.
- B. Potočnik, A. Bemporad, F.D. Torrisi, G. Mušič, and B. Zupančič. Hysdel modeling and simulation of hybrid dynamical systems. *Proc. of 4rd IMACS Symp. on Mathematical Modelling (MATHMOD)*, 2003.
- A. Puri and P. Varaiya. Verification of hybrid systems using abstractions. In Antsaklis et al. [1995], pages 359–369. ISBN 3-540-60472-3. URL citeseer.nj.nec.com/puri94verification.html.
- M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res. Develop.*, 3:115–125, 1959. URL <http://www.research.ibm.com/journal/rd/032/ibmrd0302C.pdf>.
- J. Raisch, E. Klein, S. O'Young, C. Meder, and A. Itigin. Approximating automata and discrete control for continuous systems - two examples from process control. In Antsaklis et al. [1999], pages 279–303. ISBN 3-540-65643-X.

BIBLIOGRAPHY

- M. Rönkkö and A. P. Ravn. Hybrid action systems. Technical Report TUCS TR-110, Turku Centre for Computer Science, May 1997a. URL citeseer.nj.nec.com/123739.html.
- M. Rönkkö and A. P. Ravn. Switches and jumps in hybrid action systems. Technical Report TUCS-TR-152, Turku Centre for Computer Science, 1997b. URL citeseer.nj.nec.com/151567.html.
- W. C. Rounds and H. Song. The Phi-calculus: A language for distributed control of reconfigurable embedded systems. In Maler and Pnueli [2003], pages 435–449. ISBN 3-540-00913-2.
- M. Rubensson. *Stability Properties of Switched Dynamical Systems: A Linear Matrix Inequality Approach*. PhD thesis, Control and Automation Lab., Dept. of Signals and Systems, Chalmers University of Technology, 2003.
- E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on message sequence charts. *Comput. Netw. ISDN Syst.*, 28(12):1629–1641, 1996. ISSN 0169-7552.
- T.C. Ruys. *Towards Effective Model Checking*. PhD thesis, University of Twente, 2001.
- A. Samarin. Application de la programmation réactive à la modélisation en physique, 2002.
- R.R.H. Schiffelers, D.A. van Beek, K.L. Man, M.A. Reniers, and J.E. Rooda. Formal semantics of hybrid Chi. In K. G. Larsen and P. Niebert, editors, *FORMATS*, volume 2791 of *LNCS*. Springer, 2003. ISBN 3-540-21671-5. URL <http://se.wpa.wtb.tue.nl/~vanbeek/pub/formats03.pdf>.
- M. H. Schonenberg. Discrete simulation of behavioural hybrid process algebra. Technical report, University of Twente, 2006. Draft of master thesis.
- R. Schouten. Simulation of hybrid processes. Technical report, Technical University of Eindhoven (TU/e), August 2005. Master thesis, to be defended.
- B.I. Silva, K. Richeson, B.H. Krogh, and A. Chutinan. Modeling and verification of hybrid dynamical system using CHECKMATE. In *ADPM 2000*, September 2000.
- S.N. Simić, K. H. Johansson, S. Sastry, and J. Lygeros. Towards a geometric theory of hybrid systems. In Lynch and Krogh [2000], pages 421–436. ISBN 3-540-67259-1. URL citeseer.nj.nec.com/simic00towards.html.
- S.N. Simić, K.H.Johansson, J.Lygeros, and S. Sastry. Structural stability of hybrid systems. In *Proc. of European Control Conf. '01*, 2001.
- E.D. Sontag. Nonlinear regulation: The piecewise linear approach. *IEEE Trans. Autom. Control*, 26(2):346–358, 1981.
- S.Pettersson and B.Lennartson. Exponential stability of hybrid systems using piecewise quadratic Lyapunov functions resulting in LMIS. In *Proc. of 4th IFAC World Congress*, volume J, pages 103–108, Beijing, China, July 1999. URL citeseer.nj.nec.com/122773.html.

- Stateflow. Stateflow 6 data sheets, 2004. URL https://tagteambdserver.mathworks.com/ttserverroot/Download/20637_9397v05_SF.pdf.
- S. Strubbe. *Compositional Modelling of Stochastic Hybrid Systems*. PhD thesis, University of Twente, 2005.
- S.N. Strubbe, A.A. Julius, and A.J. van der Schaft. Communicating piecewise deterministic Markov processes. In S. Engell, H. Guéguen, and J. Zaytoon, editors, *Proc. IFAC Conf. Analysis and Design of Hybrid Systems (ADHS)*, St. Malo, France, 2003. URL <http://www.supelec-rennes.fr/adhs03/>. Preprints.
- J.H. Taylor. Tools for modeling and simulation of hybrid systems - a tutorial guide, 1999.
- C. Tomlin and M. R. Greenstreet, editors. *Hybrid Systems: Computation and Control, 5th International Workshop, HSCC 2002, Stanford, CA, USA, March 25-27, 2002, Proceedings*, volume 2289 of LNCS, 2002. Springer. ISBN 3-540-43321-X.
- C. Tomlin, J. Lygeros, and S. Sastry. A game theoretic approach to controller design for hybrid systems. *Proc. of IEEE*, 88(7), July 2000.
- C.J. Tomlin, G.J. Pappas, and S. Sastry. Conflict resolution for air traffic management: A study in multi-agent hybrid systems. *IEEE Trans. Autom. Control*, 43(4):509–521, April 1998. URL citeseer.nj.nec.com/article/tomlin98conflict.html.
- J. Top and H. Akkermans. Tasks and ontologies in engineering modelling. *Int. J. Hum.-Comput. Stud.*, 41(4):585–617, 1994. ISSN 1071-5819.
- F.D. Torrisi, A. Bemporad, G. Bertini, P. Herach, D. Jost, and D. Mignone. *HYSDEL 2.0.5 - User Manual*. Zurich, September 2002.
- Bhave prot. *BHAVE prototype*. University of Twente, 2006. URL <http://fmt.cs.utwente.nl/tools/bhave>.
- Y.S. Usenko. *Linearization in μ CRL*. PhD thesis, Technical University of Eindhoven (TU/e), December 2002.
- F. W. Vaandrager and J. H. van Schuppen, editors. *Hybrid Systems: Computation and Control, Second International Workshop, HSCC'99, Berg en Dal, The Netherlands, March 29-31, 1999, Proc.*, volume 1569 of LNCS, 1999. Springer. ISBN 3-540-65734-7.
- J. van Amerongen and P. Breedveld. Modelling of physical systems for the design and control of mechatronic systems. *Annual Reviews in Control*, 27(1):87–117, 2003.
- D. A. van Beek and J. E. Rooda. Languages and applications in hybrid modelling and simulation: positioning of Chi. *Control Engineering Practice*, 8(1):81–91, 2000.
- D.A. van Beek, K.L. Man, M.A. Reniers, J.E. Rooda, and R.R.H. Schiffelers. Syntax and consistent equation semantics of hybrid chi. Report CS-Report 04-37, Technical University of Eindhoven (TU/e), Eindhoven, November 2004.

BIBLIOGRAPHY

- P.C.W. van den Brand, M.A. Reniers, and P.J.L. Cuijpers. Linearization of hybrid processes. *JLAP, Special issue on Process Theory for Hybrid Systems*, 2005. Accepted for publication.
- A.J. van der Schaft. Bisimulation of dynamical systems. In Alur and Pappas [2004], pages 555–569. ISBN 3-540-21259-0.
- A.J. van der Schaft and J. Schumacher. Complementarity modeling of hybrid systems. *IEEE Trans. Autom. Control*, 43(4):483–490, 1998.
- A.J. van der Schaft and J.M. Schumacher. *An Introduction to Hybrid Dynamical Systems*, volume 251 of *LNCIS*. Springer, London, 2000.
- P. van Eijk. *Software tools for the specification language LOTOS*. PhD thesis, University of Twente, 1988.
- A.E. van Putten. Behavioural hybrid process calculus parser and translator to Modelica. Technical report, University of Twente, 2006. Draft of master thesis.
- J.J. Vereijken. A process algebra for hybrid systems. In *The Second European Workshop on Real-Time and Hybrid Systems*, Grenoble, France, May 1995.
- M. Žefran and J.W. Burdick. Stabilization of systems with changing dynamics. In *IEEE Trans. Autom. Control*, pages 400–415, August 1998. URL citeseer.nj.nec.com/zefran98stabilization.html.
- J.L. Willems. *Stability Theory of Dynamical Systems*. Thomas Nelson and Sons LTD., 1970.
- R. Williams and R. Newell. Hybrid analysis as a batch process controller design tool. In *Proc. of Control'97*, Sydney, Australia, October 1997.
- H. Ye, A.N. Michel, and L. Hou. Stability theory for hybrid dynamical systems. *IEEE Trans. Autom. Control*, 43(4):461–474, April 1998.
- B.P. Zeigler, H. Praenhofer, and T.G. Kim. *Theory of Modelling and Simulation*. Academic Press, second edition, 2000. ISBN 0-12-778455-1.
- Q. Zhao, B.H. Krogh, and P. Hubbard. Generating test inputs for embedded control systems. *IEEE Control Systems Magazine*, 23(4):49–57, August 2003.

Index

Symbols

Σ *see* choice
 χ simulator 46, 131
 $\mathbf{0}$ *see* deadlock
 \oplus *see* superposition
 τ action *see* silent action
20-sim 48, 129

A

ACP 43
 ACP_{hs}^{srt} ... *see* process algebra for hybrid systems
action 38, 43, 75, 78
 discrete 39
 enabled 39
 external 39
 input 40
 internal 39, *see* silent action
 output 40
 receive 45
 send 45
action prefix 77, 79
 parametrisation 85
action-prefix 41
active environment 46
AHS *see* mobile vehicles
Air Traffic Management *see* mobile vehicles
 vehicles
algebraic constraints 47
alternative composition *see* choice
AnyLogic 129
architecture 124
ASCII 103, 125
ATM *see* mobile vehicles
Automated Highway System *see* mobile vehicles
 vehicles
Autonomous Flight Vehicles *see* mobile vehicles
 vehicles

autonomous system 54
Autonomous Underwater Vehicles
 see mobile vehicles
axiomatisation 41

B

batch plant control 18
behaviour 38, 67
behavioural approach 67
 behavioural equations 67
 dynamical system 67
 dynamical system with latent variables 67
 mathematical model 67
Behavioural Hybrid Process Calculus
 28, 65–92
 discrete simulator 126, 161
 parser 125, 126
BHAVE 161, 163, 164, 166
BHAVE prototype 161
bisimilarity 77
 see also bisimulation
bisimulation 42, 43, 84, 95, 147
 hybrid strong 77, 84, 95, 147
 robust 43, 95
 stateless 43, 95
block diagrams 118, 131
bond graphs 28, 47–48, 98, 118
 bond 48
 conversion 48
 dissipation 48
 distribution 48
 effort 48
 element 47
 flow 48
 generalised displacement 48
 generalised momentum 48
 gyrator 48
 junction 48

bouncing ball

flow clause

source 48
storage element 48
transformer 48
bouncing ball 12, 86

controller 5
controller generation 5
controller synthesis *see* controller
generation

C

causal stroke 48
channel 45
CHARON 28, 47, 95, 130
 toolkit 47
chattering 27
CheckMate 130
choice ... 41, 42, 44–46, 77, 81, 84, 90, 95
class 49
closed system .. *see* autonomous system
code generation 5
code snippet 126
Common Lyapunov function 53
communication function 44
complementarity 34
complementarity systems ... 28, 34–35
 extended linear 35
 linear 35
complementarity variables 35
complementary vectors
 see complementarity
component
 atomic continuous 46
 atomic discrete 46
concatenation 70, 80–81
concatenation closure 39
congruence 43, 84, 95, 147
connector 49
conservative gain *see* gain
consistent signal flow ... 78, 79, 83, 145
 see also signal
constitutive hybrid process 98
continuous dynamics 29
 linear 26, 29, 60
 non-linear 26, 29
continuous switching 141
continuous update 47
continuous variable 38
continuous-time dynamics 101
contractive cycle 56, 59, 141
control actions 26

D

d/dt 37, 131
DASSL 114
deadlock 41, 77
delay 45, 86
density 76, 82
determinism *see* nondeterminism,
 see non-determinism
DHA ... *see* discrete hybrid automaton
differential constraints 47
discrete hybrid automaton 131
discrete state 37
discrete update 47
dry friction 45, 88
Dymola 126, 132
dynamical system 67

E

ELCS *see* complementarity systems
equational theory *see* axiomatisation
equilibrium state 54
error-trace 100
event 27, *see* action
event detection 101
event handling 101
event identifier 119
event iteration 27
event trace 119, 120
event-state trace 119, 120
execution 47
exit conditions 74
expansion law 84–85, 108, 153

F

finite automaton 58
Flight Vehicles Management .. *see* mobile
 vehicles
flow clause *see* flow conditions

flow conditions 43
 fluid level 15
 forward Euler 114
 FVMS *see* mobile vehicles

G

gain 61, 63, 142
 calculation 62
 gain automaton 57
 construction of 57
 gas burner 14
 graph 118–120
 guard 37, 55, 60, 86

H

HBA *see* hybrid behavioural automaton
 hiding 47, 78, 82–83
 see also encapsulation
 of components 47
 of variables 47
 hierarchical model 46, 47
 hierarchical physical modelling 49
 hierarchical specification *see* hierarchical model
 HIOA *see* hybrid input/output automaton
 Hybrid χ 28, 45–46, 95, 128, 131
 simulator *see* χ simulator
 hybrid automata ... 11, 28, 36–37, 39, 96, 131
 linear continuous hyperplane .. 60, 61
 hybrid behavioural automaton 28, 37–38, 97
 hybrid execution 38
 Hybrid I/O automaton 39
 hybrid input/output automaton 28, 38–40, 95
 hybrid systems 2
 classification of 25–30
 grouping of 30–32
 hybrid dynamical systems 30, 93
 modelling 3
 simulation *see* simulation
 stability *see* stability

testing 6
 hybrid trace 54
 hybrid transition system 43, 75
 HyPA 28, 43–45, 95
 linearization 45
 simulation 45
 hyperplane 60–63, 131
 HYSDEL 34, 131
 HyTech 37, 131
 HyVisual 131

I

idle 85
 initial conditions 101
 initial mode *see* initial conditions
 initial process *see* initial conditions
 initial signal values 101
 initial state 40
 initialisation 101
 interconnection set 78
 interface 47
 interleaving 41, 42, 82, 84, 95
 invariant 37, 47, 60

J

jump
 autonomous *see* reset
 controlled *see* reset

L

label *see* action, 40
 transition 37
 latent variables 67
 LCS *see* complementarity systems
 linear multi-step 114
 linearization 104
 locally controlled 39
 location 37
 LOTOS 43, 44
 Lyapunov function 61, 63
 optimising choice of 63, 142

M

MAPLE 114, 126, 158, 159, 162–164
 MASACCIO 28, 46–47, 95
 MathModelica 132
 MATLAB 34, 131
 max-min-plus-scaling systems 28, 35–36
 message sequence charts 119, 120
 message sequence plots 120
 mixed logical dynamical systems 28,
 33–34, 94, 116
 continuous time 34
 discrete time 33
 MLD *see* mixed logical dynamical
 systems
 MMPS *see* max-min-plus-scaling
 systems
 mobile robot *see* mobile vehicles
 mobile vehicles 21–22
 model layers
 functional 118
 mathematical 118
 physical 118
 technical 118
 Modelica™ 28, 49, 98, 126, 127, 132
 MSC *see* message sequence charts
 MSCplots *see* message sequence plots
 MSP *see* message sequence plots
 Multi-Parametric Toolbox 34, 131
 Multiple Lyapunov functions 53

N

non-blocking statement 102
 non-causal 49
 non-contractive cycle *see* contractive
 cycle
 non-determinism 29, 115–117
 continuous systems 115
 discrete systems 115
 hybrid systems 115
 statistical methods 116
 trajectory prefix 115
 under-specification 116
 nondeterminism 26, 38, 41, 47
 scheduler 117
 weak time-determinism 45

normal form 106

O

object diagrams 118
 Open Modelica 132
 outcomes 67

P

panth 150
 parallel composition 41, 42, 44–47, 78,
 81–82, 84, 95
 Φ -calculus 28, 46, 95
 piecewise affine systems 28, 32–33, 94,
 116
 continuous time 32
 discrete 32
 plant 5
 plot *see also* graph
 prefix 72
 partial 73
 strict 72
 strict partial 73
 prefix closure 39
 process 40, 41
 identifier 83
 process algebra 40
 process algebra for hybrid systems 28,
 42–43, 95
 process calculus *see* process algebra
 projection 70
 extended 70
 Ptolemy 131
 PWA *see* piecewise affine systems

R

railroad gate control 17
 re-initialisation 43, 101
 consistent 114
 re-initialisation clause
 see re-initialisation
 recursion 41, 78, 83
 regular expression 58
 renaming 41, 46, 78, 83

- renaming function 40, 83
- reset 27, 37, 55, 60
- autonomous 26
 - controlled 26
- Runge-Kutta 114
- S**
- sampling 29
- continuous 26, 29
 - regular 26, 29
- SCICOS 132
- SCILAB 132
- scope restriction *see* hiding
- Sea Traffic Management *see* mobile vehicles
- serial composition *see* choice
- set of actions 40
- set of states 40
- set of trajectories
- extended 70
- set of trajectories prefixes 72
- closure 72
- Shift 132
- signal
- domain 68
 - emission 45
 - space 68
- signal space 75
- see also* state space
 - manifest 67
- silent action 75, 79, 83
- simulation 5, 103
- analysis tool 100
 - BHPC 102
 - continuous systems 100
 - continuous-time behaviour 111
 - discrete events 110
 - discrete systems 100
 - event tracking 101
 - event tracking algorithms 101
 - hybrid systems 101
 - mode 128
 - model development 99
 - process algebras 103
 - results analysis 127
 - smoothing method 101
 - systems analysis 99
 - time-stepping 101
 - Zeno behaviour *see also* Zeno
- simulation experiment 128
- simulation languages 31, 98
- simulation mode
- automatic 128
 - batch 129
 - interactive 128
- simulation model 127
- simulation program 128
- simulator
- compiler 126
 - compiler to executable 126
 - control centre 126
 - editor 125
 - experiment description 126
 - library 126
 - library of executable routines 126
 - optimisation 126
 - solver 126
 - translator 126
 - translator to executable 126
 - visualisation 126
- SIMULINK 123, 124, 131, 132
- Smile/M 127
- smooth model *see* system solver
- ODE/DAE 114
- SOS *see* structural operational semantics
- specification 26
- stability 54
- of dynamical systems 54
 - of equilibrium state 54
 - of hybrid automaton 55, 141
 - of hybrid automaton, multiple equilibria 55
 - of hybrid systems 54–55
- stable *see* stability
- stable location 55, 61, 141
- start state *see* state
- state 39, 40, 75
- continuous 101
 - initial 37, 39, 55, 60
- state space 37, 43, 75
- continuous 37
- Statecharts 46, 47

- STATEFLOW 123, 130, 132
- steam boiler 44
- stimulus
- external 27
 - internal 27
- STMS *see* mobile vehicles
- stop *see* deadlock
- structural operational semantics 41, 103
- suffix closure 39
- summation operator *see* choice,
see choice
- superposition 90–91
- SUT *see* system under test
- switch
- autonomous 26
 - controlled 26
- switching *see* switch
- symbolic gain 56
- synchronisation set 40, 78
- system
- smooth 101
- system under test 6
- T**
- TEDHS *see* threshold event driven
hybrid systems
- termination 43
- theorem proving 5
- thermostat 13, 42, 87, 97
- threshold event driven hybrid systems
131
- time-shift 71
- timed automaton 132
- trajectory 38, 39, 68–75
- composition of 73
 - composition of sets of 74
 - concatenation *see* concatenation
 - continuation 72
 - duration of 69
 - empty trajectory 69
 - partially equal 73
 - qualifier 68, 69, 78
 - set of 74, 75
 - set of continuations 72
 - set of qualifier 69
 - set of qualifiers 69
- trajectory prefix 77, 79–80, 90
- trajectory variable 77, 80
- transition 37, 38, 43, 55, 75
- discrete 38, 43
 - signal 43
- transition relation 40
- continuous-time 75
 - discrete 75
- transition system 40, 41
- labelled 40, 75
- transition systems
- rooted 40
- U**
- universum 67
- unstable *see* stability
- UPPAAL 120, 132
- V**
- value passing 85
- variable space
- latent 67
- variables
- external 39
 - input 38, 39
 - internal 39
 - output 38, 39
- verification 4–5
- visualisation 120
- 3D 118, 124
 - animation 118
 - component 123
 - models 117
 - results 118
- W**
- water level *see* fluid level
- well-posed 30, 33
- well-posedness 116
- Z**
- Zeno 12, 13, 60, 124

Summary

Computer controlled systems are almost omnipresent nowadays. We expect such systems to function properly at any time we need them. The malfunctioning of home electronics just irritates us, but glitches in a car, power plant or medical support system may threaten life, and faults in nuclear missile control facility may bring the end to civilisation. Such ubiquity of computer based control puts very high reliability requirements on such systems.

Hybrid systems combine continuous real-time behaviour and discrete events. Research in hybrid systems aims at providing means for reliable design and production of hybrid systems. In this thesis we explore the world of hybrid systems. We acknowledge relevance and complexity of hybrid systems research, and emphasise several different research topics: modelling, analysis, testing and deployment of hybrid systems.

We illustrate hybrid systems by presenting a collection of examples that reflect hybrid phenomena, its variety and occurrence in different applications. Of course, the list is not exhaustive, because only illustrative examples were chosen. However, it represents the diversity of hybrid systems sufficiently well to reveal the size and complexity of problems, and the broadness of the application area.

We survey several major formalisms for modelling and analysis of hybrid systems. We overview an existing classification of hybrid systems, and propose a generalised classification scheme for diverse frameworks. Moreover, we classify the surveyed formalisms according to the proposed scheme.

We propose a technique for stability estimation for a certain class of hybrid automata. It combines ideas from computer science and control theory, and is based on cycles detection and conservative gains estimation. We borrow the well-known algorithm for transforming a finite automaton into an equivalent regular expression for cycles detection from computer science. From control theory we take the idea of conservative gains and use them to estimate stability of cycles.

We introduce Behavioural Hybrid Process Calculus (BHPC), a formalism for modelling and analysis of hybrid systems. It combines process algebraic techniques and the behavioural approach [Polderman and Willem, 1998] to dynamical systems. We take an attempt to advance fusion of computer science and control theory in hybrid systems research. BHPC is based on two fundamental notions of actions and trajectories that describe discrete and continuous evolution of dynamical systems, respectively. At a higher abstraction level these two types of behaviour are treated uniformly, i.e., as normal elements of process algebra. Their behaviour is defined using structural operational semantics (SOS) rules [Plotkin, 1981, 2003]. Moreover, the rules already respect the differences between trajectory prefixes and action prefixes, based on our intuition on how such processes should behave. For example, in parallel composition



SUMMARY

trajectory prefixes are always required to synchronise, while for action prefixes interleaving semantics is adopted. Furthermore, we define a hybrid strong bisimulation relation for BHPC and prove that it is a congruence. It is one of the most important properties to attain well defined compositionality. Such a property allows to interchange bisimilar processes in any process algebraic expression. In other words, it allows to refine process, change their internal representation, and interchange them without any losses as long as they manifest the same behaviour.

We propose a technique for simulation of BHPC. We devise a simulation algorithm for a subset of BHPC operators and test some of the proposed techniques in the `BHAVE` prototype. The proposed simulation algorithm defines one of the possible ways to simulate Behavioural Hybrid Process Calculus. Moreover, we survey the major problems in simulation of hybrid systems in the light of BHPC and in a more general layout. For some of the issues we propose solutions, for the remaining we discuss potential ways to tackle the problems.

Samenvatting

Computergestuurde systemen zijn tegenwoordig bijna overal aanwezig. We verwachten dat zulke systemen op ieder gewenst moment correct functioneren. Het disfunctioneren van electronica thuis ergert ons alleen maar, maar storingen in een auto, kerncentrale of medisch systeem kunnen levensbedreigend zijn. En als er iets misgaat in een controle centrum voor kernwapens, kan dat een eind maken aan de hele beschaving. Het alomtegenwoordig zijn van computergestuurde systemen stelt hoge eisen aan de betrouwbaarheid van zulke systemen.

Hybride systemen combineren real-time gedrag met discrete gebeurtenissen. Onderzoek naar hybride systemen tracht middelen te vinden voor het betrouwbaar ontwerpen en produceren van hybride systemen. In dit proefschrift onderzoeken we de wereld van hybride systemen. We erkennen de relevantie en complexiteit van onderzoek naar hybride systemen en benadrukken enkele verschillende onderwerpen van onderzoek: modelleren, analyse, testen en toepassen van hybride systemen.

We illustreren hybride systemen aan de hand van een verzameling voorbeelden, die hybride fenomenen zichtbaar maken, alsook zijn variëteit en aanwezigheid in verschillende applicaties. Natuurlijk is deze lijst niet volledig, omdat slechts illustratieve voorbeelden werden gekozen. De lijst representeert echter genoeg van de diversiteit van hybride systemen, om de omvang en complexiteit van de problemen en de uitgebreidheid van het toepassingsgebied te laten zien.

We onderzoeken enkele belangrijke formele formalismen voor de modellering en analyse van hybride systemen. We geven een overzicht van een bestaande classificatie van hybride systemen en stellen een algemeen classificatieschema voor verschillende frameworks voor. Verder classificeren we de onderzochte formalismen volgens het voorgestelde schema.

We introduceren een techniek voor het schatten van de stabiliteit voor een bepaalde klasse van hybride automaten. Deze techniek combineert ideeën van de informatica en de besturingstheorie en is gebaseerd op het detecteren van cyclen en een conservatieve schatting van de toestandsvergroting. We lenen het bekende algoritme om een eindige automaat in een equivalente reguliere expressie voor cyclusdetectie om te zetten van de informatica. Van de besturingstheorie nemen we het idee over van conservatieve toestandsvergroting en we gebruiken dit om de stabiliteit van cyclen te schatten.

We introduceren de Behavioural Hybrid Process Calculus (BHPC), een formalisme voor modellering en analyse van hybride systemen. Dit combineert technieken uit de procesalgebra en de behavioural approach [Polderman and Willems, 1998] van dynamische systemen. We doen een poging om het combineren van de informatica en de besturingstheorie in onderzoek naar hybride systemen te verbeteren. BHPC is gebaseerd op twee fundamentele begrippen van actions en trajectories die respectievelijk discrete en continue evolutie van dynamische systemen beschrijven. Op hoger

abstractieniveau worden deze twee soorten gedrag uniform behandeld, als normale elementen van de procesalgebra. Hun gedrag wordt bepaald door gebruik te maken van structural operational semantics (SOS) regels [Plotkin, 1981, 2003]. Verder respecteren de regels al de verschillen tussen trajectory prefixes en action prefixes, gebaseerd op onze intuïtie over hoe dergelijke processen zich zouden moeten gedragen. Bijvoorbeeld, in parallelle compositie moeten de trajectory prefixes altijd synchroniseren, terwijl voor action prefixes interleaving semantics wordt gebruikt. Verder definiëren we een hybride sterke bisimulatielatie voor BHPC en bewijzen dat het een congruentie is. Het is een van de belangrijkste eigenschappen om een goed bepaalde compositionaliteit te bereiken. Zo'n eigenschap staat het toe om bisimilaire processen om te wisselen binnen elke expressie in de procesalgebra. Met andere woorden, het staat toe om processen te verfijnen, hun interne representatie te veranderen, en hen om te wisselen zonder enig verlies, zolang ze hetzelfde gedrag vertonen.

We introduceren een techniek voor simulatie van BHPC. We construeren een simulatiealgoritme voor een deelverzameling van BHPC-operatoren en testen enkele voorgestelde technieken in het BHAVE prototype. Het voorgestelde simulatiealgoritme bepaalt een van de mogelijke manieren om de Behavioural Hybrid Process Calculus te simuleren. Verder onderzoeken we de belangrijkste problemen in simulatie van hybride systemen in het licht van BHPC en in een meer algemene opzet. Voor enkele kwesties stellen we oplossingen voor, voor de overgebleven vraagstukken bespreken we mogelijke manieren om de problemen aan te pakken.

Titles in the IPA Dissertation Series

- J.O. Blanco.** *The State Operator in Process Algebra.* Faculty of Mathematics and Computing Science, TUE. 1996-01
- A.M. Geerling.** *Transformational Development of Data-Parallel Algorithms.* Faculty of Mathematics and Computer Science, KUN. 1996-02
- P.M. Achten.** *Interactive Functional Programs: Models, Methods, and Implementation.* Faculty of Mathematics and Computer Science, KUN. 1996-03
- M.G.A. Verhoeven.** *Parallel Local Search.* Faculty of Mathematics and Computing Science, TUE. 1996-04
- M.H.G.K. Kessler.** *The Implementation of Functional Languages on Parallel Machines with Distributed Memory.* Faculty of Mathematics and Computer Science, KUN. 1996-05
- D. Alstein.** *Distributed Algorithms for Hard Real-Time Systems.* Faculty of Mathematics and Computing Science, TUE. 1996-06
- J.H. Hoepman.** *Communication, Synchronization, and Fault-Tolerance.* Faculty of Mathematics and Computer Science, UvA. 1996-07
- H. Doornbos.** *Reductivity Arguments and Program Construction.* Faculty of Mathematics and Computing Science, TUE. 1996-08
- D. Turi.** *Functorial Operational Semantics and its Denotational Dual.* Faculty of Mathematics and Computer Science, VUA. 1996-09
- A.M.G. Peeters.** *Single-Rail Handshake Circuits.* Faculty of Mathematics and Computing Science, TUE. 1996-10
- N.W.A. Arends.** *A Systems Engineering Specification Formalism.* Faculty of Mechanical Engineering, TUE. 1996-11
- P. Severi de Santiago.** *Normalisation in Lambda Calculus and its Relation to Type Inference.* Faculty of Mathematics and Computing Science, TUE. 1996-12
- D.R. Dams.** *Abstract Interpretation and Partition Refinement for Model Checking.* Faculty of Mathematics and Computing Science, TUE. 1996-13
- M.M. Bonsangue.** *Topological Dualities in Semantics.* Faculty of Mathematics and Computer Science, VUA. 1996-14
- B.L.E. de Fluiter.** *Algorithms for Graphs of Small Treewidth.* Faculty of Mathematics and Computer Science, UU. 1997-01
- W.T.M. Kars.** *Process-algebraic Transformations in Context.* Faculty of Computer Science, UT. 1997-02
- P.F. Hoogendijk.** *A Generic Theory of Data Types.* Faculty of Mathematics and Computing Science, TUE. 1997-03
- T.D.L. Laan.** *The Evolution of Type Theory in Logic and Mathematics.* Faculty of Mathematics and Computing Science, TUE. 1997-04
- C.J. Bloo.** *Preservation of Termination for Explicit Substitution.* Faculty of Mathematics and Computing Science, TUE. 1997-05
- J.J. Vereijken.** *Discrete-Time Process Algebra.* Faculty of Mathematics and Computing Science, TUE. 1997-06
- F.A.M. van den Beuken.** *A Functional Approach to Syntax and Typing.* Faculty of Mathematics and Informatics, KUN. 1997-07
- A.W. Heerink.** *Ins and Outs in Refusal Testing.* Faculty of Computer Science, UT. 1998-01
- G. Naumoski and W. Alberts.** *A Discrete-Event Simulator for Systems Engineering.* Faculty of Mechanical Engineering, TUE. 1998-02
- J. Verriet.** *Scheduling with Communication for Multiprocessor Computation.* Faculty of Mathematics and Computer Science, UU. 1998-03
- J.S.H. van Gageldonk.** *An Asynchronous Low-Power 80C51 Microcontroller.* Faculty of Mathematics and Computing Science, TUE. 1998-04
- A.A. Basten.** *In Terms of Nets: System Design with Petri Nets and Process Algebra.* Faculty of Mathematics and Computing Science, TUE. 1998-05
- E. Voermans.** *Inductive Datatypes with Laws and Subtyping – A Relational Model.* Faculty of Mathematics and Computing Science, TUE. 1999-01
- H. ter Doest.** *Towards Probabilistic Unification-based Parsing.* Faculty of Computer Science, UT. 1999-02
- J.P.L. Segers.** *Algorithms for the Simulation of Surface Processes.* Faculty of Mathematics and Computing Science, TUE. 1999-03
- C.H.M. van Kemenade.** *Recombinative Evolutionary Search.* Faculty of Mathematics and Natural Sciences, UL. 1999-04
- E.I. Barakova.** *Learning Reliability: a Study on Indecisiveness in Sample Selection.* Faculty of Mathematics and Natural Sciences, RUG. 1999-05
- M.P. Bodlaender.** *Scheduler Optimization in Real-Time Distributed Databases.* Faculty of Mathematics and Computing Science, TUE. 1999-06

- M.A. Reniers.** *Message Sequence Chart: Syntax and Semantics.* Faculty of Mathematics and Computing Science, TUE. 1999-07
- J.P. Warners.** *Nonlinear approaches to satisfiability problems.* Faculty of Mathematics and Computing Science, TUE. 1999-08
- J.M.T. Romijn.** *Analysing Industrial Protocols with Formal Methods.* Faculty of Computer Science, UT. 1999-09
- P.R. D'Argenio.** *Algebras and Automata for Timed and Stochastic Systems.* Faculty of Computer Science, UT. 1999-10
- G. Fábán.** *A Language and Simulator for Hybrid Systems.* Faculty of Mechanical Engineering, TUE. 1999-11
- J. Zwanenburg.** *Object-Oriented Concepts and Proof Rules.* Faculty of Mathematics and Computing Science, TUE. 1999-12
- R.S. Venema.** *Aspects of an Integrated Neural Prediction System.* Faculty of Mathematics and Natural Sciences, RUG. 1999-13
- J. Saraiva.** *A Purely Functional Implementation of Attribute Grammars.* Faculty of Mathematics and Computer Science, UU. 1999-14
- R. Schiefer.** *Viper, A Visualisation Tool for Parallel Program Construction.* Faculty of Mathematics and Computing Science, TUE. 1999-15
- K.M.M. de Leeuw.** *Cryptology and Statecraft in the Dutch Republic.* Faculty of Mathematics and Computer Science, UvA. 2000-01
- T.E.J. Vos.** *UNITY in Diversity. A stratified approach to the verification of distributed algorithms.* Faculty of Mathematics and Computer Science, UU. 2000-02
- W. Mallon.** *Theories and Tools for the Design of Delay-Insensitive Communicating Processes.* Faculty of Mathematics and Natural Sciences, RUG. 2000-03
- W.O.D. Griffioen.** *Studies in Computer Aided Verification of Protocols.* Faculty of Science, KUN. 2000-04
- P.H.F.M. Verhoeven.** *The Design of the MathSpad Editor.* Faculty of Mathematics and Computing Science, TUE. 2000-05
- J. Fey.** *Design of a Fruit Juice Blending and Packaging Plant.* Faculty of Mechanical Engineering, TUE. 2000-06
- M. Franssen.** *Cocktail: A Tool for Deriving Correct Programs.* Faculty of Mathematics and Computing Science, TUE. 2000-07
- P.A. Olivier.** *A Framework for Debugging Heterogeneous Applications.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08
- E. Saaman.** *Another Formal Specification Language.* Faculty of Mathematics and Natural Sciences, RUG. 2000-10
- M. Jelasity.** *The Shape of Evolutionary Search Discovering and Representing Search Space Structure.* Faculty of Mathematics and Natural Sciences, UL. 2001-01
- R. Ahn.** *Agents, Objects and Events a computational approach to knowledge, observation and communication.* Faculty of Mathematics and Computing Science, TU/e. 2001-02
- M. Huisman.** *Reasoning about Java programs in higher order logic using PVS and Isabelle.* Faculty of Science, KUN. 2001-03
- I.M.M.J. Reymen.** *Improving Design Processes through Structured Reflection.* Faculty of Mathematics and Computing Science, TU/e. 2001-04
- S.C.C. Blom.** *Term Graph Rewriting: syntax and semantics.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05
- R. van Liere.** *Studies in Interactive Visualization.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06
- A.G. Engels.** *Languages for Analysis and Testing of Event Sequences.* Faculty of Mathematics and Computing Science, TU/e. 2001-07
- J. Hage.** *Structural Aspects of Switching Classes.* Faculty of Mathematics and Natural Sciences, UL. 2001-08
- M.H. Lamers.** *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes.* Faculty of Mathematics and Natural Sciences, UL. 2001-09
- T.C. Ruys.** *Towards Effective Model Checking.* Faculty of Computer Science, UT. 2001-10
- D. Chkhaev.** *Mechanical verification of concurrency control and recovery protocols.* Faculty of Mathematics and Computing Science, TU/e. 2001-11
- M.D. Oostdijk.** *Generation and presentation of formal mathematical documents.* Faculty of Mathematics and Computing Science, TU/e. 2001-12
- A.T. Hofkamp.** *Reactive machine control: A simulation approach using χ .* Faculty of Mechanical Engineering, TU/e. 2001-13
- D. Bošnački.** *Enhancing state space reduction techniques for model checking.* Faculty of Mathematics and Computing Science, TU/e. 2001-14
- M.C. van Wezel.** *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects.* Faculty of Mathematics and Natural Sciences, UL. 2002-01

- V. Bos and J.J.T. Kleijn.** *Formal Specification and Analysis of Industrial Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02
- T. Kuipers.** *Techniques for Understanding Legacy Software Systems.* Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03
- S.P. Luttkik.** *Choice Quantification in Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04
- R.J. Willemen.** *School Timetable Construction: Algorithms and Complexity.* Faculty of Mathematics and Computer Science, TU/e. 2002-05
- M.I.A. Stoelinga.** *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-06
- N. van Vugt.** *Models of Molecular Computing.* Faculty of Mathematics and Natural Sciences, UL. 2002-07
- A. Fehnker.** *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems.* Faculty of Science, Mathematics and Computer Science, KUN. 2002-08
- R. van Stee.** *On-line Scheduling and Bin Packing.* Faculty of Mathematics and Natural Sciences, UL. 2002-09
- D. Tauritz.** *Adaptive Information Filtering: Concepts and Algorithms.* Faculty of Mathematics and Natural Sciences, UL. 2002-10
- M.B. van der Zwaag.** *Models and Logics for Process Algebra.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11
- J.I. den Hartog.** *Probabilistic Extensions of Semantical Models.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12
- L. Moonen.** *Exploring Software Systems.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13
- J.I. van Hemert.** *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining.* Faculty of Mathematics and Natural Sciences, UL. 2002-14
- S. Andova.** *Probabilistic Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2002-15
- Y.S. Usenko.** *Linearization in μ CRL.* Faculty of Mathematics and Computer Science, TU/e. 2002-16
- J.J.D. Aerts.** *Random Redundant Storage for Video on Demand.* Faculty of Mathematics and Computer Science, TU/e. 2003-01
- M. de Jonge.** *To Reuse or To Be Reused: Techniques for component composition and construction.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02
- J.M.W. Visser.** *Generic Traversal over Typed Source Code Representations.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03
- S.M. Bohte.** *Spiking Neural Networks.* Faculty of Mathematics and Natural Sciences, UL. 2003-04
- T.A.C. Willemse.** *Semantics and Verification in Process Algebras with Data and Timing.* Faculty of Mathematics and Computer Science, TU/e. 2003-05
- S.V. Nedeia.** *Analysis and Simulations of Catalytic Reactions.* Faculty of Mathematics and Computer Science, TU/e. 2003-06
- M.E.M. Lijding.** *Real-time Scheduling of Tertiary Storage.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07
- H.P. Benz.** *Casual Multimedia Process Annotation – CoMPAs.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08
- D. Distefano.** *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09
- M.H. ter Beek.** *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components.* Faculty of Mathematics and Natural Sciences, UL. 2003-10
- D.J.P. Leijen.** *The λ Abroad – A Functional Approach to Software Components.* Faculty of Mathematics and Computer Science, UU. 2003-11
- W.P.A.J. Michiels.** *Performance Ratios for the Differencing Method.* Faculty of Mathematics and Computer Science, TU/e. 2004-01
- G.I. Jojgov.** *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving.* Faculty of Mathematics and Computer Science, TU/e. 2004-02
- P. Frisco.** *Theory of Molecular Computing – Splicing and Membrane systems.* Faculty of Mathematics and Natural Sciences, UL. 2004-03
- S. Maneth.** *Models of Tree Translation.* Faculty of Mathematics and Natural Sciences, UL. 2004-04
- Y. Qian.** *Data Synchronization and Browsing for Home Environments.* Faculty of Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05
- F. Bartels.** *On Generalised Coinduction and Probabilistic Specification Formats.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

- L. Cruz-Filipe.** *Constructive Real Analysis: a Type-Theoretical Formalization and Applications.* Faculty of Science, Mathematics and Computer Science, KUN. 2004-07
- E.H. Gerding.** *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications.* Faculty of Technology Management, TU/e. 2004-08
- N. Goga.** *Control and Selection Techniques for the Automated Testing of Reactive Systems.* Faculty of Mathematics and Computer Science, TU/e. 2004-09
- M. Niqui.** *Formalising Exact Arithmetic: Representations, Algorithms and Proofs.* Faculty of Science, Mathematics and Computer Science, RU. 2004-10
- A. Löh.** *Exploring Generic Haskell.* Faculty of Mathematics and Computer Science, UU. 2004-11
- I.C.M. Flinsenberg.** *Route Planning Algorithms for Car Navigation.* Faculty of Mathematics and Computer Science, TU/e. 2004-12
- R.J. Bril.** *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets.* Faculty of Mathematics and Computer Science, TU/e. 2004-13
- J. Pang.** *Formal Verification of Distributed Systems.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14
- F. Alkemade.** *Evolutionary Agent-Based Economics.* Faculty of Technology Management, TU/e. 2004-15
- E.O. Dijk.** *Indoor Ultrasonic Position Estimation Using a Single Base Station.* Faculty of Mathematics and Computer Science, TU/e. 2004-16
- S.M. Orzan.** *On Distributed Verification and Verified Distribution.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17
- M.M. Schrage.** *Proxima - A Presentation-oriented Editor for Structured Documents.* Faculty of Mathematics and Computer Science, UU. 2004-18
- E. Eskenazi and A. Fyukov.** *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures.* Faculty of Mathematics and Computer Science, TU/e. 2004-19
- P.J.L. Cuijpers.** *Hybrid Process Algebra.* Faculty of Mathematics and Computer Science, TU/e. 2004-20
- N.J.M. van den Nieuwelaar.** *Supervisory Machine Control by Predictive-Reactive Scheduling.* Faculty of Mechanical Engineering, TU/e. 2004-21
- E. Ábrahám.** *An Assertional Proof System for Multithreaded Java -Theory and Tool Support- .* Faculty of Mathematics and Natural Sciences, UL. 2005-01
- R. Ruimerman.** *Modeling and Remodeling in Bone Tissue.* Faculty of Biomedical Engineering, TU/e. 2005-02
- C.N. Chong.** *Experiments in Rights Control - Expression and Enforcement.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03
- H. Gao.** *Design and Verification of Lock-free Parallel Algorithms.* Faculty of Mathematics and Computing Sciences, RUG. 2005-04
- H.M.A. van Beek.** *Specification and Analysis of Internet Applications.* Faculty of Mathematics and Computer Science, TU/e. 2005-05
- M.T. Ionita.** *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures.* Faculty of Mathematics and Computing Sciences, TU/e. 2005-06
- G. Lenzini.** *Integration of Analysis Techniques in Security and Fault-Tolerance.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07
- I. Kurtev.** *Adaptability of Model Transformations.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08
- T. Wolle.** *Computational Aspects of Treewidth - Lower Bounds and Network Reliability.* Faculty of Science, UU. 2005-09
- O. Tveretina.** *Decision Procedures for Equality Logic with Uninterpreted Functions.* Faculty of Mathematics and Computer Science, TU/e. 2005-10
- A.M.L. Liekens.** *Evolution of Finite Populations in Dynamic Environments.* Faculty of Biomedical Engineering, TU/e. 2005-11
- J. Eggermont.** *Data Mining using Genetic Programming: Classification and Symbolic Regression.* Faculty of Mathematics and Natural Sciences, UL. 2005-12
- B.J. Heeren.** *Top Quality Type Error Messages.* Faculty of Science, UU. 2005-13
- G.F. Frehse.** *Compositional Verification of Hybrid Systems using Simulation Relations.* Faculty of Science, Mathematics and Computer Science, RU. 2005-14
- M.R. Mousavi.** *Structuring Structural Operational Semantics.* Faculty of Mathematics and Computer Science, TU/e. 2005-15
- A. Sokolova.** *Coalgebraic Analysis of Probabilistic Systems.* Faculty of Mathematics and Computer Science, TU/e. 2005-16
- T. Gelsema.** *Effective Models for the Structure of pi-Calculus Processes with Replication.* Faculty of Mathematics and Natural Sciences, UL. 2005-17
- P. Zoetewij.** *Composing Constraint Solvers.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting.* Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell.* Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model.* Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems.* Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems.* Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality.* Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications.* Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting.* Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs.* Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations.* Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data.* Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space.* Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices.* Faculty of Science, Mathematics and Computer Science, RU. 2006-12

B. Badban. *Verification techniques for Extensions of Equality Logic.* Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2006-13

A.J. Mooij. *Constructive formal methods and protocol standardization.* Faculty of Mathematics and Computer Science, TU/e. 2006-14

T. Krilavičius. *Hybrid Techniques for Hybrid Systems.* Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-15