Supervisory Control Using Failure Semantics and Partial Specifications

Ard Overkamp

Abstract—**A framework is presented for the supervisory control of nondeterministic discrete-event systems based on failure semantics. It guarantees deadlock-free behavior under all circumstances, it allows for powerful specifications, it forms a sound basis for modular control, and it can handle nondeterminism without extra effort. A synthesis method to generate the least restrictive supervisor is presented.**

Secondly, the control problem with partial specification is formulated, and it is shown that this control problem can be rewritten to a control problem with full specification. Special care has to be taken for traces with an unbounded internal extension (divergence). A condition, denoted bounded recurrence, is introduced to handle these traces. It is shown that the external behavior of the controlled system is not restricted by this condition.

Index Terms— **Failure semantics, nondeterministic discrete-event systems, partial specifications, supervisory control.**

I. Introduction

NONDETERMINISM reflects the lack, or the deliberate hiding, of information. Reality may be considered deterministic, but models are an abstraction of reality. Models stress some aspects of reality by hiding irrelevant details. The hiding of details causes systems to exhibit nondeterministic behavior. Therefore, we wish to use nondeterministic processes to model discrete-event systems. Of course, models can be made deterministic by also including the unimportant details and stating which details are important and which are not. However, this will lead to unnecessarily complex models.

Supervisory control of deterministic discrete-event systems was first introduced by Ramadge and Wonham [1]. In [2] the basic supervisory control problem for nondeterministic systems was introduced and solved. The approach is based on failure semantics. Failure semantics provides a theoretical foundation to reason about the behavior of nondeterministic discrete-event systems. It is introduced by Hoare [3]. In [4] the supervisory control problem for nondeterministic systems with partial specification was formulated, and it was stated, without proofs, that it can be reduced to the basic control problem. In this paper a proof of this result is presented. Partial specifications allow for implementation-independent specifications, which are highly desired if specifications are used by different manufacturers or if new implementations are expected in the future. Partial specifications are also well suited for control problems in layered architectures such as the ISO-OSI network model [5]. Protocol design problems in layered architectures can be treated as control problems by considering the lower level service as the uncontrolled system, the protocol as the supervisor, and the higher-level service as the specification. Usually the lower level uses implementation events that are not used in the higher level. This leads very naturally to a control problem with partial specification.

A discussion on the motivation for the use of failure semantics will be given in Section IV. The approach based on failure semantics will be compared with frameworks based on deterministic systems with marking. It will be shown that the supervisory control framework based on failure semantics is a flexible and elegant method. It guarantees deadlock-free behavior under all circumstances, it allows for powerful specifications, it forms a sound basis for modular control, and it can handle nondeterminism without extra effort.

The control problem discussed in this paper is to find a supervisor such that the controlled system can replace a given specification in any environment. The precise meaning of this statement and the motivation for it are given in Section IV.

The main difference between the approach presented in this paper and the approach presented by Kumar and Shayman [6] is that the latter uses the prioritized synchronization operator. This more complex synchronization operator requires a stronger semantics than failure semantics. Therefore, they use trajectory semantics for their models. Future experience will provide information whether this enhanced complexity is required, or useful, to handle supervisory control problems. Kumar and Shayman use a language as specification, whereas in this paper a process is used. A process can specify nondeterministic properties, therefore the class of legal implementations can be more accurately specified by a process than by a language.

Another control framework, which can handle nondeterministic systems, is presented by DiBenedetto *et al.* [7]. This approach is based on input/output (I/O) automata.

In [8] Inan presents the projected specification problem which is similar to the control problem with partial specification discussed in this paper. Although Inan uses a nondeterministic supervisor as a finite representation of the possibly infinite set of solutions, the approach is based on languages. He does not consider unbounded internal continuations (divergence, see Section III).

II. Processes

Let Σ denote the set of all possible events. A *trace* s is a finite sequence of events $s = \sigma_1\sigma_2\cdots\sigma_n$, with for all $1 \le i \le n, \sigma_i \in \Sigma$. The *length* of a trace is the number of

Manuscript received July 21, 1995; revised April 23, 1996 and October 8, 1996. Recommended by Associate Editor, S. Lafortune.

The research was performed while the author was with CWI, 1009 GB, Amsterdam, the Netherlands.

Publisher Item Identifier S 0018-9286(97)02805-5.

0018–9286/97$10.00 © 1997 IEEE

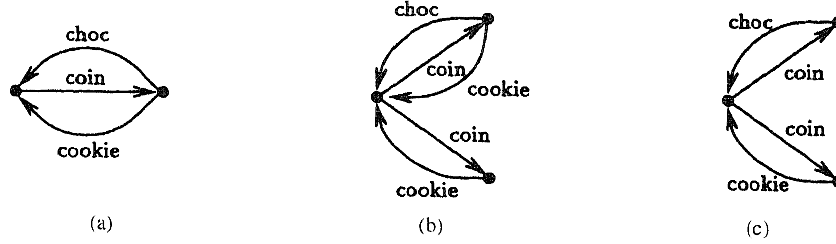


Fig. 1. Models of a vending machine.

events in the trace. Let ε be the empty trace, i.e., the sequence of events with length zero. Let Σ^n denote the set of traces with length n: $\Sigma^n = \{\sigma_1\sigma_2\cdots\sigma_n : \forall 1 \leq i \leq n, \sigma_i \in \Sigma\}$. Let Σ^* denote the set of all traces: $\Sigma^* = \cup_{n=0}^{\infty} \Sigma^n$. Let Σ^+ denote the set of all nonempty traces: $\Sigma^+ = \cup_{n=1}^{\infty} \Sigma^n = \Sigma^* - \{\varepsilon\}$. A *language* is a subset of Σ^*.

A string $v \in \Sigma^*$ is a *prefix* of a string $s \in \Sigma^*$ if $s = vt$ for some $t \in \Sigma^*$. The set of all prefixes of string s will be denoted by $\overline{s} = \{v \in \Sigma^* : \exists t \in \Sigma^* \text{ s.t. } s = vt\}$. The *prefix closure* of a language $K \subseteq \Sigma$ is the set of all its prefixes: $\overline{K} = \{v \in \Sigma^* : \exists s \in K \text{ s.t. } v \in \overline{s}\}$. A language is called *prefix closed* if it is equal to its prefix closure: $K = \overline{K}$. The next event function λ gives all events that are possible after a string: $\lambda(K, s) = \{\sigma \in \Sigma : s\sigma \in K\}$. The ρ-function is the complement of the next event function. It gives all events that cannot be executed after a string: $\rho(K, s) = \Sigma - \lambda(K, s)$.

Deterministic processes can be uniquely described by the language they generate. For nondeterministic processes this is not enough [9].

Example 1: Consider a vending machine that hands out a cookie or a chocolate bar in exchange for a coin. In Fig. 1 the representations of three vending machines are given by finite-state automata. All three machines can generate the same language but will behave differently. Therefore, it is not sufficient to describe their behavior by the language that they can generate. How the machines behave is best illustrated by letting a user operate the machines. After a client inserts a coin, the first machine will always hand out what the user requests. It will never refuse to give a cookie or a chocolate bar. If the user wants to have a chocolate bar from the second machine, he might get disappointed because the machine can reach a state in which it cannot give a chocolate bar. It will, however, never refuse to hand out a cookie. The third machine can sometimes refuse to give a cookie and sometimes refuse to give a chocolate bar, but it cannot refuse both at the same time. If a user requests either of the sweets, no matter if it is a cookie or chocolate bar, then the machine cannot refuse and it must hand out one of them. To describe the behavior of the machines it is necessary to not only describe the events that can be executed, i.e., the language, but also the event sets that can be refused. This is the basis of failure semantics [3]. A machine can refuse event set $R \subseteq \Sigma$ after string s if it can reach a state by executing string s, and it cannot execute any event of event set R in this state. Note, however, that because of nondeterminism, the machine might also be able to reach another state after string s, and in this state it might be able to execute an event of event set R. So, it is possible that an event can be executed after a trace, although it can also be refused after the same trace.

The event sets that can be refused are called *refusals.* A set of refusals is called a *refusal set.* For instance, the refusal set of the third machine after a coin is inserted is the following:

$$\{\emptyset, \{\text{coin}\}, \{\text{cookie}\}, \{\text{choc}\}, \{\text{coin, cookie}\}, \{\text{coin, choc}\}\}.$$

As explained in Example 1, the machine cannot refuse both the cookie and choc-event, so the event set {cookie, choc} is not an element of the refusal set.

In the sequel the terms "process" and "system" are used interchangeably.

Definition 1: A *process* is defined as a triple $A = (\Sigma(A), L(A), \text{ref}(A))$, where

$$\Sigma(A) \subseteq \Sigma \text{ is the set of event labels}$$
$$L(A) \subseteq \Sigma(A)^* \text{ is the language generated by } A$$
$$\text{for } s \in L(A), \text{ref}(A, s) \subseteq 2^{\Sigma(A)} \text{ is the refusal set after } s$$

and which satisfies the following five conditions [3].

1) $\varepsilon \in L(A)$.
2) $L(A) = \overline{L(A)}$.
3) $s \in L(A) \Rightarrow \emptyset \in \text{ref}(A, s)$.
4) $R \in \text{ref}(A, s)$ and $R' \subseteq R \Rightarrow R' \in \text{ref}(A, s)$.
5) $R \in \text{ref}(A, s) \Rightarrow R \cup \rho(L(A), s) \in \text{ref}(A, s)$.

These conditions state, respectively, that the language has to be nonempty and prefix closed, the refusal sets have to be nonempty and closed under the operation of taking the subset, and events that cannot be refused must be in the language.

For $s \notin L(A)$ the refusal set $\text{ref}(A, s)$ is defined to be $2^{\Sigma(A)}$. Let $\Pi(\Sigma)$ be the set of all processes A with $\Sigma(A) = \Sigma$.

The ref-function associates to each string a set of subsets of Σ. If a subset R is an element of $\text{ref}(A, s)$, then the process has the possibility after trace s to block all events in R. That is, if a user offers (via the synchronous composition defined below) to the system a set of events which is in the refusal set, then the system has the possibility of blocking all these events. No event can be executed. This is called a deadlock.

Definition 2: System A can *deadlock* after trace $s \in L(A)$ if $\Sigma(A) \in \text{ref}(A, s)$. System A is *deadlock-free* after s if $\Sigma(A) \notin \text{ref}(A, s)$.

It will be assumed that if A is deadlock-free after trace s, then eventually it will execute an event from $\lambda(L(A), s)$. So a system will continue unless it deadlocks. Note, however, that if a process *can* deadlock after a trace, then this does not mean that it actually *will* deadlock. If a process can deadlock after trace s, then it can reach a state q_1 in which it cannot execute any further event. But, it could also be, because of

nondeterminism, that it reaches another state, say q_2, in which it *can* execute an event.

Deterministic Processes

A discrete-event system A is considered deterministic if from each state there is at most one transition corresponding with each event $\sigma \in \Sigma(A)$. After observing a trace $s \in L(A)$ it is uniquely determined in which state the system is. So it is also uniquely determined which events can be executed after s and which events are refused after s. We will consider a process deterministic if and only if any event that can be executed after a trace cannot be refused after the same trace.

The class of nondeterministic processes strictly contains the class of deterministic processes.

Definition 3: Process A is called *deterministic* if for all $s \in L(A)$

$$R \in \text{ref}(A, s) \Leftrightarrow R \subseteq \rho(L(A), s).$$

A prefix-closed language $K \subseteq \Sigma^*$ uniquely defines the deterministic process Det (K), with

$$\begin{aligned} \Sigma(\text{Det}(K)) &= \Sigma \\ L(\text{Det}(K)) &= K \\ \text{ref}(\text{Det}(K), s) &= 2^{\rho(K,s)}. \end{aligned}$$

It is not difficult to prove that Det (K) satisfies the conditions 1)–5) of Definition 1. It will be left without proof that all processes constructed in the rest of the paper satisfy these conditions.

The class of deterministic processes does not correspond exactly with the class of deterministic automata. Consider an automaton that can make a nondeterministic choice, but in each option it behaves exactly the same as in the other options. Thus, the nondeterministic choice cannot be detected from the behavior of the automaton. This system is considered nondeterministic from an automaton point of view because it can make a nondeterministic choice. But, it is deterministic from a process point of view because it satisfies Definition 3. A system will be called deterministic if its process representation is deterministic.

Synchronous Composition

Control will be enforced by synchronization on common events. The controlled system (i.e., the synchronous composition of the plant and the supervisor) can only execute those events that both the supervisor and the plant can execute.

Definition 4: Let $A, B \in \Pi(\Sigma)$. The *synchronous composition* of A and B is the process $A||B \in \Pi(\Sigma)$, with

$$\begin{aligned} \Sigma(A||B) &= \Sigma \\ L(A||B) &= L(A) \cap L(B) \\ \text{ref}(A||B, s) &= \{R_a \cup R_b: R_a \in \text{ref}(A, s) \quad \text{and} \\ &\qquad R_b \in \text{ref}(B, s)\}. \end{aligned}$$

Behavior State Representation

For processes there is no canonical automaton representation such as the minimal state deterministic automaton representation used for languages. In this section we will define a representation that will be used in the rest of the paper as the automaton representation of processes. It will be used to describe processes and to perform computations on processes. The representation is based on an equivalence relation similar to the Nerode equivalence relation used for languages [10].

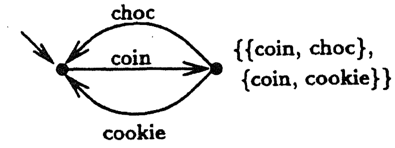


Fig. 2. Behavior state representation of a vending machine.

Let A/s be the process that behaves as process A after it has executed trace s. Two traces s and s' can be considered equivalent if $A/s = A/s'$, that is, if the traces that A can execute and the event sets that A can refuse after s and s', and the event sets that A can refuse after any continuation of s and s' are the same. One can regard A/s as the state reached after trace s. To differentiate this notion of state from the states used in regular nondeterministic automata, we will call A/s the *behavior state* reached after trace s. These behavior states can be used to make a behavior state transition structure, where the set $\{A/s: s \in L(A)\}$ is the state space and the transition function is defined by $A/s \xrightarrow{\sigma} A/s\sigma$. The initial state is $A = A/\varepsilon$. Associated with each state is a set of refusals defined by

$$\text{ref}(A/s) = \text{ref}(A/s, \varepsilon) = \text{ref}(A, s).$$

This transition structure will be denoted the *behavior state representation* of process A. Note that this representation forms a deterministic transition structure because each pair $(A/s, \sigma)$ uniquely defines the next state $A/s\sigma$. It can be seen as if the nondeterministic properties are encoded inside the refusal sets of the behavior states, instead of modeled by the transition function.

In Fig. 2 the behavior state representation of the process defined by Fig. 1(c) is given. For compactness reasons, only the maximal refusals, i.e., the refusals not strictly contained in another refusal, are shown. As refusal sets are closed under the operation of taking subsets, the whole refusal set can be derived from the maximal refusals. Also, the refusal sets of behavior states A/s with ref $(A/s) = 2^{\rho(L(A),s)}$ are not shown. These refusals can be derived from the outgoing edges of the state.

The refusal set of behavior state A/ε is

$$\begin{aligned} \text{ref}(A/\varepsilon) &= 2^{\rho(L(A),\varepsilon)} \\ &= \{\emptyset, \{\text{choc}\}, \{\text{cookie}\}, \{\text{choc}, \text{cookie}\}\}. \end{aligned}$$

The refusal set of behavior state A/s is the set of all subsets of $\{\text{coin}, \text{cookie}\}$ and $\{\text{coin}, \text{choc}\}$. It is shown after Example 1.

Converting a nondeterministic finite-state machine to a behavior state representation is basically the same as converting a nondeterministic state machine to a deterministic version. This conversion has a known complexity that is worst case exponential in the size of the state space of the original state machine. But in practice, systems have sufficient structure such that this conversion may not be a problem.

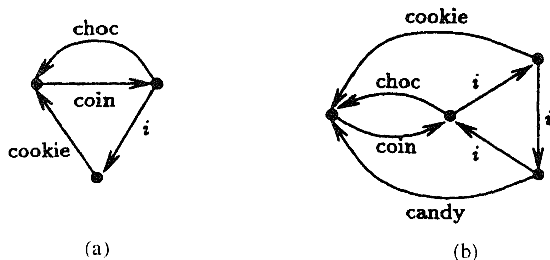


Fig. 3. Internal events and divergence.

III. PROJECTION

Very often events occur that are not visible from the outside of a system. Inside the system an error can happen, the effect of which can only be detected later. Also, low-level implementation events usually do not show up on a higher level. In order to be able to investigate the external or higher level description of a system, we need a method to project these internal events out of a description.

Definition 5: Let $\Sigma_e \subseteq \Sigma$ denote the set of *external events* and $\Sigma_i = \Sigma - \Sigma_e$ the set of *internal events*. The *projection* of a trace onto event set Σ_e is the trace from which all events not in Σ_e are removed [1]. The projection of a language is the set of all its projected traces. In the sequel a small "p" will denote projections of traces, languages, and refusal sets. A large "P" will denote projections of processes (see Definition 7). A subscript indicates the event set on which the projection is done. Consider a vending machine as shown in Fig. 3(a). After inserting a coin, a chocolate bar can be obtained, provided the machine does not execute internal event i, after which only a cookie can be obtained. The question is what the projected system can refuse after a coin is inserted. Suppose a customer insists on having a cookie. He refuses to accept a chocolate bar. The machine must now execute the internal event because that is the only possible event that is not blocked. The system ends up in a state in which it must hand out a cookie. So, it is clear that the machine cannot refuse to engage in a cookie-event. The machine can refuse to hand out a chocolate bar because the customer cannot prevent the internal event from occurring. After the internal event the vending machine cannot engage in the choc-event.

If a system refuses external event set $R_e \subseteq \Sigma_e$, but it can still execute an internal event, then it may end up in a state in which it does not refuse this external event set. On the other hand, if the system does not refuse R_e, but it can still execute an internal event, then it may end up in a state in which it can refuse R_e. It turns out that the refusals of the projected system are defined by those states in which the machine cannot execute internal events. These states correspond with refusals that contain the set of internal events. The projection of the refusal set $\text{ref}(A, s)$ on events set Σ_e is defined by

$$p_e(\text{ref}(A,s)) = \{R \subseteq \Sigma_e : R \cup \Sigma_i \in \text{ref}(A,s)\}.$$

Divergence

But there is another problem. It may happen that a machine can execute internal events forever. See for example Fig. 3(b). After a coin is inserted, the machine can always choose to execute an internal event because it cannot be blocked from the outside. To the customer it appears as if the machine refuses all external events. This phenomenon is called divergence [3].

A trace s is called divergent with respect to a system and an external event set if the system can execute an unbounded number of internal events after s. We cannot write "an infinite number of events," because only finite traces are considered.

Definition 6: The set of *divergent traces* with respect to a language $K \subseteq \Sigma^*$ and external event set Σ_e is denoted by $\text{div}(K, \Sigma_e)$ and defined by

$$\text{div}(K, \Sigma_e) = \{s \in K : \forall n \in \mathbb{N}, \exists s_i \in \Sigma_i^* \text{ s.t. } |s_i| > n \text{ and } ss_i \in K\}.$$

Let $A \in \Pi(\Sigma)$ be a process. For notational convenience we will use $\text{div}(A, \Sigma_e)$ to denote $\text{div}(L(A), \Sigma_e)$.

As $\text{div}(A, \Sigma_e)$ is a set of traces, i.e., a language, the projection $p_e(\text{div}(A, \Sigma_e))$ is well defined.

Definition 7: The *projection* of process $A \in \Pi(\Sigma)$ on alphabet Σ_e is the process $P_e(A) \in \Pi(\Sigma_e)$, where

$$\Sigma(P_e(A)) = \Sigma_e$$
$$L(P_e(A)) = p_e(L(A))$$
$$\text{ref}(P_e(A), s_e) = \begin{cases} 2^{\Sigma_e}, & \text{if } s_e \in p_e(\text{div}(A, \Sigma_e)) \\ \bigcup_{s \in p_e^{-1}(s_e)} p_e(\text{ref}(A,s)) & \text{otherwise.} \end{cases}$$

In the definition above it is assumed that if a system can diverge, then it has the ability to refuse all external events. In some situations this is a rather pessimistic assumption. Sometimes a more optimistic approach is justified. Consider, for instance, a network in which a lost message automatically causes the retransmission of the message. The internal events "message-lost" and "retransmit" together form a loop of internal events. After retransmission the retransmitted message may also get lost, which causes the next retransmission. With a pessimistic point of view one can argue that the system can execute internal events indefinitely and can therefore refuse all external events. But usually it is assumed that eventually, after sufficient retransmissions, the network will be able to deliver the message. This can be interpreted as if the system cannot refuse the external "message-received" event. It would go beyond the scope of this paper to further investigate the consequences of this more optimistic interpretation of divergence [11]. In the sequel the pessimistic approach toward divergence will be used.

IV. SPECIFICATION, IMPLEMENTATION, AND CONTROL

In general, a design problem can be defined as: given a specification, find an implementation that satisfies the specification. A design problem can be considered a supervisory control problem if the implementation consists of an already existing uncontrolled process G and a still-to-be-designed supervisor process S. In this paper, the control problem of finding a supervisor S such that $G||S$ can replace a given specification process E is analyzed. The use of failure semantics for this control problem is described and motivated in this section. The following example will illustrate in what sense an implementation must be able to replace a specification.

Example 2: A system usually does not work on its own. It is embedded in a larger system. For instance, a hard-

disk unit is used inside a computer system. The computer is usually designed at a different location than the hard-disk unit. During the design phase a standard is negotiated between the computer manufacturer and the disk manufacturer. This standard is the specification of the hard-disk. After this standard is established, the computer designer models a computer system in which it expects a hard-disk unit that behaves according to this specification. It is the hard-disk developer's task to build a hard-disk unit that satisfies this specification. Without him knowing how the computer system will look, he has to design a unit that works together with this system. He has to build an implementation of the hard disk that can replace the specification used for the design of the computer system.

Consider the following implementation relation [12], [3].

Definition 8: Let $A, B \in \Pi(\Sigma)$. A *reduces* B, denoted by $A \sqsubseteq B$, if

1) $L(A) \subseteq L(B)$;
2) $\forall s \in L(A), \mathrm{ref}\,(A, s) \subseteq \mathrm{ref}\,(B, s)$.

Here, point 1) states that system A may only do what system B allows, and point 2) states that A may only refuse what B can also refuse. We will say that process $G||S$ implements specification E if $G||S \sqsubseteq E$.

The next two results are well known in computer science [13]. The first one states that two processes are considered equal if they both implement each other. The second one states that the reduction relation forms a congruence with the synchronous composition.

Proposition 1: Let $A, B \in \Pi(\Sigma)$. Then $A = B \Leftrightarrow A \sqsubseteq B \wedge B \sqsubseteq A$.

Proposition 2: Let $A_1, A_2, B_1, B_2 \in \Pi(\Sigma)$ such that $A_1 \sqsubseteq A_2$ and $B_1 \sqsubseteq B_2$. Then $A_1||B_1 \sqsubseteq A_2||B_2$.

In Example 2 the implementation of the hard-disk has to [illegible] such that it can replace its specification in any computer [illegible]em. This is guaranteed by the reduction relation. Let $G||S$ [illegible]d for the implementation of the hard-disk, E for the [illegible]cification, and C for the rest of the computer system. Then [illegible]e following implication, which is a direct consequence of Proposition 2, states that $G||S$ can replace E in any computer system:

$$G||S \sqsubseteq E \Rightarrow \forall C, \quad (G||S)||C \sqsubseteq E||C. \tag{1}$$

The basic supervisory control problem is to find a supervisor S such that $G||S \sqsubseteq E$.

Implication 1 shows that the reduction relation is strong enough to be used as an implementation relation. The following result shows that it also forms a necessary condition to guarantee deadlock-free behavior. This result forms the main motivation for the use of failure semantics and the reduction relation.

Theorem 1: Let $A, E \in \Pi(\Sigma)$

$$A \sqsubseteq E$$
$$\Leftrightarrow$$
$$\forall C \quad L(A||C) \subseteq L(E||C), \quad \text{and}$$
$$E||C \text{ deadlock-free} \Rightarrow A||C \text{ deadlock-free}.$$

Proof: The $\Rightarrow$-part follows from Proposition 2. For the proof of the $\Leftarrow$-part assume that A does not reduce E. Then either $L(A) \not\subseteq L(E)$ or there exists an $s \in L(A)$ such that $\mathrm{ref}\,(A, s) \not\subseteq \mathrm{ref}\,(E, s)$. Assume there exists an $s \in L(A)$ such that $s \notin L(E)$. Let C be a process such that $s \in L(C)$. Then $s \in L(A||C)$ but $s \notin L(E||C)$, so $L(A||C) \not\subseteq L(E||C)$. For the other alternative let $s \in L(A)$ such that there exists an $R \in \mathrm{ref}\,(A, s)$ and $R \notin \mathrm{ref}\,(E, s)$. Let C be a process such that $\mathrm{ref}\,(C, s) = 2^{\Sigma - R}$. Then $\Sigma = R \cup (\Sigma - R) \in \mathrm{ref}\,(A||C, s)$, but $\Sigma \notin \mathrm{ref}\,(E||C, s)$. So $E||C$ is deadlock-free, but $A||C$ is not. ■

A Comparison of Frameworks

In the rest of this section we will compare the approach based on failure semantics and the reduction relation with other approaches. The comparison is not intended to be complete. It just illustrates some differences between the approaches.

The original framework introduced by Ramadge and Wonham [1] was intended to handle only deterministic systems. The framework presented in this thesis is also capable of handling nondeterministic systems. But even if we restrict our attention to deterministic systems, there are some important differences.

It can be shown in the framework presented by Ramadge and Wonham that the corresponding implication of (1) is not satisfied. In that framework a discrete-event system A is modeled by the triple $(\Sigma(A), L(A), L_m(A))$, where $L(A) \subseteq \Sigma(A)^*$ is the prefix-closed language that A can generate and $L_m(A) \subseteq L(A)$ is the language that A accepts or marks.

Definition 9: Let A and B be discrete-event systems. $A \sqsubseteq_m B$ if

1) $L(A) \subseteq L(B)$;
2) $L_m(A) \subseteq L_m(B)$;
3) $L(A) = \overline{L_m(A)}$.

A system is called *M-nonblocking* if it satisfies point 3).

System $G||S$ is considered an implementation of E in the Ramadge–Wonham framework if $G||S \sqsubseteq_m E$. Note that if $L(E) = \overline{L_m(E)}$, then points 2) and 3) together imply point 1). Usually the specification is not given as a process but as a language $K \subseteq L_m(G)$. In this case the specification process E can be defined by $L(E) = \overline{K}$ and $L_m(E) = K$. Sometimes a nonmarking supervisor is required, that is $L_m(S) = L(S)$. In this case it is usually assumed that $L_m(E) = L(E) \cap L_m(G)$. These differences are not important for the following discussion which mainly concerns point 3).

We wish that an implementation could replace the specification in any environment. This is, however, not guaranteed by the $\sqsubseteq_m$ relation. The next example illustrates that in general it cannot be guaranteed that the implementation is M-nonblocking in any environment, i.e.,

$$G||S \sqsubseteq_m E \wedge E||C \text{ is } M\text{-nonblocking}$$
$$\not\Rightarrow$$
$$(G||S)||C \text{ is } M\text{-nonblocking}.$$

Example 3: Let E be the specification with $L_m(E) = a(b + c)$ and $L(E) = \overline{a(b + c)}$; let $A = G||S$ be the implementation with $L_m(A) = ab$ and $L(A) = \overline{ab}$; and let C represent the rest

of the computer system with $L_m(C) = ac$ and $L(C) = \overline{ac}$. Observe that $A \sqsubseteq_m E$, but $L(A||C) = a$ and $L_m(A||C) = \varepsilon$, thus $L(A||C) \neq \overline{L_m(A||C)}$. Thus, $A||C \not\sqsubseteq_m E||C$.

It can be derived from results obtained by Wonham and Ramadge [14] on modular control that $(G||S)||C$ is only M-nonblocking if $L_m(G||S)$ and $L_m(C)$ are *nonconflicting*. That is, processes A and B are nonconflicting if common prefixes in both processes can be extended to a common marked trace

$$\overline{L_m(A) \cap L_m(B)} = \overline{L_m(A)} \cap \overline{L_m(B)}.$$

This constraint also limits the use of modular control in the Ramadge–Wonham framework. If a specification E can be decomposed as $E_1||E_2 = E$, then it has computational advantages to first synthesize both S_1 and S_2 such that $G||S_1$ implements E_1 and $G||S_2$ implements E_2. In the framework based on failure semantics it can be deduced from Proposition 2 and the fact that $G \sqsubseteq G||G$, that

$$G||S_1 \sqsubseteq E_1 \wedge G||S_2 \sqsubseteq E_2 \Rightarrow G||S_1||S_2 \sqsubseteq E_1||E_2.$$

In the Ramadge–Wonham framework, however, it is necessary that $L_m(G||S_1)$ and $L_m(G||S_2)$ are nonconflicting in order to guarantee that $G||S_1||S_2$ is M-nonblocking. This constraint is often difficult to satisfy.

The discussion above considers how well the M-nonblocking property and the deadlock freeness property behave within their own framework. It does not compare the properties directly with each other. The M-nonblocking property states that a process is always able to complete a task, whereas the deadlock freeness property states that a process is always able to continue. Note that it cannot be specified by a marked language that the implementation should be deadlock-free. Even if there are transitions leading out of each marked state in the specification, then still an implementation which deadlocks in a marked state satisfies the specification according to Definition 9. Therefore, marking cannot be used to guarantee deadlock-free behavior. It depends on the particular application which approach is more suited. An open question is whether the marking condition on states can be replaced by an event that indicates the completion of a task. With this approach a process can be considered nonblocking if it cannot refuse such a task completion event. The nonblocking property can then be adequately handled within the framework based on failure semantics.

Within the computer science area synthesis is investigated based on infinite trace theory [15]–[17]. Also within the control theory area, this approach has been followed [18]. Infinite trace automata have an acceptance condition which is similar to the marking condition for finite trace automata. Because of this acceptance condition, the corresponding implication of (1) will not be satisfied within this framework. Also, there will be extra constraints necessary for modular control synthesis.

It is logical, if one considers that the implication should be able to replace the specification and that the specification is given as a process. It can be shown that with a process a more accurate specification of all legal implementations can be given than with a formulation that uses legal languages. A specification process can be seen as the nondeterministic choice between all legal implementations.

V. Controller Synthesis

In Section VI the supervisory control problem with partial specification will be discussed. In this section it will be shown how the basic control problem with full specification can be solved.

The basic supervisory control problem is formulated as follows. Given an uncontrolled system G and a specification E, find a supervisor S such that $G||S \sqsubseteq E$.

In some applications the supervisor does not have the ability to block all events. For instance, if an alarm event is executed when some water level exceeds a threshold, then this event can be observed by the supervisor but it cannot be blocked. Usually the presence of uncontrollable events is modeled by splitting up the event set into two subsets Σ_c and Σ_u, where $\Sigma_c \subseteq \Sigma$ represents the controllable events and $\Sigma_u = \Sigma - \Sigma_c$ the uncontrollable events. The basic supervisory control problem is extended with the requirement that the supervisor has to be complete, i.e., it does not block any uncontrollable events.

Definition 10: Let $G \in \Pi(\Sigma)$ be an uncontrolled system. Supervisor $S \in \Pi(\Sigma)$ is *complete* if

$$\forall s \in L(S||G), \quad \forall R_s \in \mathrm{ref}(S, s), \quad R_s \cap \Sigma_u \subseteq \rho(L(G), s).$$

Definition 11: Let the uncontrolled system $G \in \Pi(\Sigma)$ and a specification $E \in \Pi(\Sigma)$ be given. The *basic supervisory control problem* is to find a complete supervisor $S \in \Pi(\Sigma)$ such that $G||S \sqsubseteq E$.

Ramadge and Wonham showed that for the existence of complete supervisor in a deterministic setting, the existence of a controllable language is a necessary and sufficient condition [1]. In [2] it is shown that for nondeterministic systems, the language also has to satisfy another condition which is called reducibility. This reducibility condition guarantees that the supervisor only blocks events where this is allowed according to the refusal sets of the specification.

Definition 12: Let $G, E \in \Pi(\Sigma)$. Let K be a language contained in $L(G)$ and $L(E)$. K is *reducible* (w.r.t. G, E) if

$$\forall s \in K, \quad \forall R_g \in \mathrm{ref}(G, s), \quad \rho(K, s) \cup R_g \in \mathrm{ref}(E, s).$$

K is *controllable* (w.r.t. G) if

$$K\Sigma_u \cap L(G) = K.$$

An interpretation of the reducibility condition follows from the definition of reduction and synchronous composition. Observe that

$$\mathrm{ref}(G||S, s) \subseteq \mathrm{ref}(E, s) \Leftrightarrow \forall R_g \in \mathrm{ref}(G, s), \ \forall R_s \in \mathrm{ref}(S, s), \ R_g \cup R_s \in \mathrm{ref}(E, s).$$

If S is deterministic, then $R_s \in \mathrm{ref}(S, s)$ if and only if $R_s \subseteq \rho(L(S), s)$. As $\mathrm{ref}(E, s)$ is subset closed, it follows that

$$\forall R_g \in \mathrm{ref}(G, s), \ \forall R_s \subseteq \rho(L(S), s), \ R_g \cup R_s \in \mathrm{ref}(E, s) \Leftrightarrow \forall R_g \in \mathrm{ref}(G, s), \ \rho(L(S), s) \cup R_g \in \mathrm{ref}(E, s).$$

So, if S is deterministic then $\text{ref}\,(G||S, s) \subseteq \text{ref}\,(E, s)$ for all $s \in L(G||S) \subseteq L(E)$, if and only if $L(S)$ is reducible.

Theorem 2 [2, Th. 13]: Let $G, E \in \Pi(\Sigma)$. There exists a complete supervisor $S \in \Pi(\Sigma)$ such that $G||S \sqsubseteq E$ if and only if there exists a nonempty, prefix-closed language K which is controllable w.r.t. $L(G)$, reducible w.r.t. $L(G)$ and $L(E)$, and contained in $L(G)$ and $L(E)$.

If K is a language satisfying the conditions in Theorem 13, then the process $\text{Det}\,(K)$ is a complete supervisor, such that $G||\text{Det}\,(K) \sqsubseteq E$. If S is a complete supervisor satisfying $G||S \sqsubseteq E$, then $L(G||S)$ satisfies the conditions in the theorem. This also implies that $\text{Det}\,(L(G||S))$ can be used as a supervisor. So if there exists a possibly nondeterministic supervisor S, then there also exists a deterministic supervisor $S_{\text{det}} = \text{Det}\,(L(G||S))$, such that $G||S_{\text{det}} \sqsubseteq E$ and $L(G||S_{\text{det}}) = L(G||S)$.

It is not difficult to prove that the set of reducible languages contained in $L(G) \cap L(E)$ is closed under arbitrary unions, so a unique supremal element exists and is contained in the set. This supremal can be efficiently computed in the case of finite-state systems [1], [2]. In [19] an algorithm is presented to compute the supremal controllable language that has linear complexity if the languages are prefix closed. This algorithm can be adapted to compute the supremal controllable and reducible sublanguage of a given language. It removes states from the process $G||E$ that violate the controllability or reducibility condition. Let $|G|, |E|$ denote the size of behavior state spaces of G and E, respectively. If G and E are given by behavior state representations, then the algorithm has complexity $|G| \times |E|$.

VI. Control of Partially Specified Systems

One aspect of a specification is that it should be implementation-independent. That is, a specification should describe *what* a system should do, not *how* it should be done. This has the advantage that two systems with a completely different implementation, but with the same specification, are interchangeable. Consider, for instance, a car. All cars have a similar specification. They have a gas-pedal on the right, a brake in the middle, and optionally a clutch on the left. It is not necessary to know the implementation-dependent aspects of the car, such as the number of pistons, or the way the fuel is injected. Just a specification given in events relevant for the user is sufficient to drive any car, from a family sedan to a high-powered sportscar.

We will divide the event set into two subsets Σ_e and Σ_i. The external events (Σ_e) are those events that are relevant for the users of the system. The specification should be stated in terms of these events. The internal or implementation events (Σ_i) are not provided to the environment. They do not appear in the specification. They are, however, observable by the supervisor because the supervisor is part of the implementation. The supervisory control problem with partial specification can be defined as follows.

Definition 13: Let $E \in \Pi(\Sigma_e)$ be the specification process, and let $G \in \Pi(\Sigma)$ be the uncontrolled system. Let $\Sigma_e \subseteq \Sigma$. The *supervisory control problem with partial specification* is to find a complete supervisor $S \in \Pi(\Sigma)$ such that $P_e(G||S) \sqsubseteq E$.

Bounded Recurrence

In a language semantics setting, the control problem with partial specification can be easily rewritten into a control problem with full specification because the following relation holds:

$$p_e(L(G||S)) \subseteq L(E) \Leftrightarrow L(G||S) \subseteq \sup_{\subseteq} p_e^{-1}(L(E))$$

where $\sup_{\subseteq}$ denotes the supremal with respect to language inclusion and $p_e^{-1}(L(E))$ is defined as

$$p_e^{-1}(L(E)) = \{K \subseteq \Sigma^* : p_e(K) = L(E)\}.$$

Note that

$$\begin{aligned}\sup_{\subseteq} p_e^{-1}(L(E)) &= \{s \in \Sigma^* : p_e(s) \in L(E)\} \\ &= \sup_{\subseteq} \{K \subseteq \Sigma^* : p_e(K) \subseteq L(E)\} \\ &= \bigcup \{K \subseteq \Sigma^* : p_e(K) \subseteq L(E)\}.\end{aligned}$$

Thus, $\sup_{\subseteq} p_e^{-1}(L(E))$ is equal to the union of all legal implementations, i.e., the union of all languages that are allowed as language of the controlled system.

Let us also try to apply this idea to nondeterministic systems. Define $P_e^{-1}(E)$ as the set of all systems that project onto E

$$P_e^{-1}(E) = \{A \in \Pi(\Sigma) : P_e(A) = E\}.$$

Let $\sup_{\sqsubseteq}$ denote the supremal with respect to the reduction relation. In [13, Th. 1] it is proven that the set of processes $\Pi(\Sigma)$, with the reduction relation as partial ordering, forms a complete upper semilattice.[1] This implies that any subset of processes has a least upper bound, i.e., a supremal. It does not imply that this supremal is an element of the subset. It only implies that this supremal exists, i.e., that it is a process. The following example shows that in general:

$$G||S \sqsubseteq \sup_{\sqsubseteq} P_e^{-1}(E) \not\Rightarrow P_e(G||S) \sqsubseteq E.$$

So, in general, it is not guaranteed that $\sup_{\sqsubseteq} P_e^{-1}(E)$ is an element of $P_e^{-1}(E)$.

Example 4: Let $E \in \Pi(\Sigma_e)$ be a process such that the refusal set after $p_e(s)$ does not contain the complete external event set Σ_e. If we compute the inverse projection of E, then this will include systems that can execute $s\Sigma_i^n$, with n some constant. Systems that allow $s\Sigma_i^*$ can diverge after s. When projected, they can refuse the whole external events set. This is not allowed by E. Therefore, systems that allow $s\Sigma_i^*$ are not an element of $P_e^{-1}(E)$. The supremal element of $\{s\Sigma_i^n : n \in \mathbb{N}\}$ is $s\Sigma_i^*$. So, the supremal element of $P_e^{-1}(E)$ will allow $s\Sigma_i^*$. We have that the supremal element of $P_e^{-1}(E)$ does not project onto E. Therefore, $G||S \sqsubseteq \sup_{\sqsubseteq} P_e^{-1}(E)$ does not imply $P_e(G||S) \sqsubseteq E$.

In the example above we saw that if the number of loops of internal events is bounded by a constant n, then the system cannot diverge. The idea is now to limit the possible solutions

[1] Note that the ordering used in [13] and [3] is the same as the ordering induced by the reduction relation, except that the processes are ordered in the opposite direction.

by fixing *a priori* a constant n and allowing only solutions that make at most n internal loops if the specification cannot refuse the whole external event set. We will prove that if $n \geq 2$, then the external behavior of the final solution is not restricted by this.

First, it will be shown how the control problem with partial specification can be reduced to the basic control problem. After that, an example will be given to illustrate the approach. It will also be shown in this example why at least two internal loops are needed to guarantee that the external language is not restricted by the followed approach.

Because the external behavior is relevant to the user of the system, we want to make it as least restrictive as possible. The internal behavior is invisible to the user. It is only relevant to the implementation. There is no reason why this behavior should be least restrictive. In fact, it is even desirable to make the internal behavior as small as possible in order to keep the implementation costs as small as possible [4]. In this paper only solutions will be considered that make at most two loops of internal events when the specification cannot refuse the whole external event set. This idea is formalized by introducing the notion of bounded recurrence.

Definition 14: It will be said that the trace $s' \in \Sigma^*$ is in the *last internal part* of trace $s \in \Sigma^*$ if $s' \in \overline{s}$ and $p_e(s') = p_e(s)$. Thus, if s' is in the last internal part of s, then there exists an $s_i \in \Sigma_i^*$ such that $s's_i = s$. So s' and s are equal up to some internal events at the end of trace s.

Let the *recurrence index* of trace $s \in L(G)$ indicate how often the behavior state G/s is visited by the last internal part of trace s

$$r_i(s) = |\{s' \in \overline{s}: p_e(s) = p_e(s') \wedge G/s = G/s'\}|.$$

Consider the behavior state representation of system G given in Fig. 4(a). The recurrence indexes of the traces a, aij, and $aijij$ are 1, 2, and 3, respectively.

Definition 15: Trace $s \in L(G)$ is called *bounded recurrent* (w.r.t. G, E, and Σ_e) if

$$\Sigma_e \notin \text{ref}\,(E, p_e(s)) \Rightarrow r_i(s) \leq 2.$$

All traces $s \in \Sigma^*$ that are not an element of $L(G)$ are defined to be bounded recurrent also. A language $K \subseteq \Sigma^*$ is called *bounded recurrent* if all traces $s \in K$ are bounded recurrent. Process $A \in \Pi(\Sigma)$ is called *bounded recurrent* if $L(A)$ is bounded recurrent.

Note that if $s \notin \text{div}\,(G, \Sigma_e)$, then for all $s_i \in \Sigma_i^+$ such that $ss_i \in L(G)$ we have that $G/s \neq G/ss_i$. Note also that if $\Sigma_e \in \text{ref}\,(E, p_e(s))$, then s is bounded recurrent.

Reduction of the Control Problem

In this section it will be shown that if we restrict the set of solutions to bounded recurrent processes, then the control problem with partial specification can be reduced to the basic supervisory control problem. It will also be shown that this restriction does not limit the external language of the controlled system.

Definition 16: $E^\uparrow = \sup_{\sqsubseteq} P_e^{-1}(E) = \sup_{\sqsubseteq} \{A \in \Pi(\Sigma): P(A) \sqsubseteq E\}$.

Proposition 3: The process $E^\uparrow$ satisfies

$$\begin{aligned}
\Sigma(E^\uparrow) &= \Sigma \\
L(E^\uparrow) &= \sup_{\subseteq} P_e^{-1}(L(E)) \\
&= \{s \in \Sigma^*: p_e(s) \in L(E)\} \\
\text{ref}\,(E^\uparrow, s) &= \{R \subseteq \Sigma: \Sigma_i \subseteq R \Rightarrow \Sigma_e \cap R \in \text{ref}\,(E, p_e(s))\}.
\end{aligned}$$

This characterization can be used to construct $E^\uparrow$.

Proof: Let E_{con} be the process defined by the expressions in the proposition. We have to prove that $E^\uparrow = E_{\text{con}}$. For all $n \in \mathbb{N}$ let A_n be the process defined by

$$\begin{aligned}
\Sigma(A_n) &= \Sigma \\
L(A_n) &= \{s \in \Sigma^*: s \in \overline{\Sigma_i^n}\sigma_1\overline{\Sigma_i^n}\sigma_2 \cdots \sigma_m\overline{\Sigma_i^n}, \\
&\qquad \sigma_1\sigma_2 \cdots \sigma_m \in L(E)\} \\
\text{ref}\,(A_n, s) &= \text{ref}\,(E_{\text{con}}, s).
\end{aligned}$$

First, it will be proven that for all $n \in \mathbb{N}, P_e(A_n) \sqsubseteq E$. The language part of the reduction relation follows from $L(P_e(A_n)) = p_e(L(A_n)) = L(E)$. Note that after any trace in $L(A_n)$ only a bounded number of internal events are possible, so $\text{div}\,(A_n, \Sigma_e) = \emptyset$. The refusal part of $P_e(A_n)$ follows from

$$\begin{aligned}
\text{ref}\,&(P_e(A_n), s_e) \\
&= \bigcup_{s \in p_e^{-1}(s_e)} p_e(\text{ref}\,(A_n, s)) \\
&= \bigcup_{s \in p_e^{-1}(s_e)} \{R \subseteq \Sigma_e: R \cup \Sigma_i \in \text{ref}\,(A_n, s)\} \\
&= \bigcup_{s \in p_e^{-1}(s_e)} \{R \subseteq \Sigma_e: (R \cup \Sigma_i) \cap \Sigma_e \in \text{ref}\,(E, p_e(s))\} \\
&= \text{ref}\,(E, s_e).
\end{aligned}$$

Hence, for all $n \in \mathbb{N}, P_e(A_n) \sqsubseteq E$ and as $E^\uparrow$ is the supremal of $\{A \in \Pi(\Sigma): P_e(A) \sqsubseteq E\}$, it follows that $A_n \sqsubseteq E^\uparrow$.

Second, it will be proven that $L(E_{\text{con}}) = L(E^\uparrow)$. As all processes that reduce E have no trace not contained in $L(E)$, it follows that $L(E^\uparrow) \subseteq L(E_{\text{con}})$. As $\cup_{n \in \mathbb{N}} \Sigma_i^n = \Sigma_i^*$ and for all $n \in \mathbb{N} L(A_n) \subseteq L(E^\uparrow)$, it follows that $L(E_{\text{con}}) = \cup_{n \in \mathbb{N}} L(A_n) \subseteq L(E^\uparrow)$. Hence $L(E_{\text{con}}) = L(E^\uparrow)$.

Next it will be proven that for all $s \in L(E_{\text{con}}) = L(E^\uparrow), \text{ref}\,(E_{\text{con}}, s) \subseteq \text{ref}\,(E^\uparrow, s)$. For all $s \in L(E^\uparrow)$ there exists an $n \in \mathbb{N}$ such that $s \in A_n$. As $\text{ref}\,(A_n, s) = \text{ref}\,(E_{\text{con}}, s)$ and $A_n \sqsubseteq E^\uparrow$, it follows that $\text{ref}\,(E_{\text{con}}, s) \subseteq \text{ref}\,(E^\uparrow, s)$.

Finally, it will be proven that $\text{ref}\,(E^\uparrow, s) \subseteq \text{ref}\,(E_{\text{con}}, s)$. Suppose the inclusion does not hold. Then there must exist a process B and a refusal $R \in \text{ref}\,(B, s)$ such that $P_e(B) \sqsubseteq E$ and $R \notin \text{ref}\,(E_{\text{con}}, s)$. That is, $\Sigma_i \subseteq R$ and $R \cap \Sigma_e \notin \text{ref}\,(E, p_e(s))$. But then it follows from the definition of projection that $P_e(B) \not\sqsubseteq E$, which contradicts our assumption. Hence $\text{ref}\,(E^\uparrow, s) \subseteq \text{ref}\,(E_{\text{con}}, s)$.

We have proven that $L(E_{\text{con}}) = L(E^\uparrow)$ and for all $s \in L(E^\uparrow), \text{ref}\,(E_{\text{con}}, s) = \text{ref}\,(E^\uparrow, s)$, so $E_{\text{con}} = E^\uparrow$. ■

Definition 17: $E_{\text{br}}^\uparrow = \sup_{\sqsubseteq} \{A \in \Pi(\Sigma): P_e(A) \sqsubseteq E, A$ is bounded recurrent$\}$. By analogy with [3], the process $E_{\text{br}}^\uparrow$ can be seen as the nondeterministic choice between all legal bounded recurrent implementations.

Proposition 4: Process $E_{\rm br}^{\uparrow}$ satisfies

$$\Sigma(E_{\rm br}^{\uparrow}) = \Sigma$$
$$L(E_{\rm br}^{\uparrow}) = \{s \in \Sigma^*: p_e(s) \in L(E) \text{ and } s \text{ is bounded recurrent}\}$$
$$\text{ref}\,(E_{\rm br}^{\uparrow}, s) = \{R \subseteq \Sigma: \Sigma_i \subseteq R \cup \rho(L(E_{\rm br}^{\uparrow}), s) \Rightarrow \Sigma_e \cap R \in \text{ref}\,(E, p_e(s))\}.$$

This characterization can be used to construct $E_{\rm br}^{\uparrow}$.

Proof: Let $E_{\rm con}$ be the process defined by the expressions in the proposition. We have to prove that $E_{\rm br}^{\uparrow} = E_{\rm con}$. First it will be proven that $P_e(E_{\rm con}) \sqsubseteq E$. The language part follows from $L(P_e(E_{\rm con})) = p_e(L(E_{\rm con})) = L(E)$. For the refusal part note that $\Sigma_e \notin \text{ref}\,(E, s_e)$ implies $s_e \notin p_e(\text{div}\,(E_{\rm con}, \Sigma_e))$. So, if $s_e \in p_e(\text{div}\,(E_{\rm con}, \Sigma_e))$, then $\text{ref}\,(P_e(E_{\rm con}), s_e) = 2^{\Sigma_e} = \text{ref}\,(E, s_e)$. If $s_e \notin p_e(\text{div}\,(E_{\rm con}, \Sigma_e))$ then, by the same line of reasoning as in the proof of Proposition 3, it follows that

$$\begin{aligned}
&\text{ref}\,(P_e(E_{\rm con}), s_e)\\
&= \bigcup_{s \in p_e^{-1}(s_e)} p_e(\text{ref}\,(E_{\rm con}, s))\\
&= \bigcup_{s \in p_e^{-1}(s_e)} \{R \subseteq \Sigma_e: R \cup \Sigma_i \in \text{ref}\,(E_{\rm con}, s)\}\\
&= \bigcup_{s \in p_e^{-1}(s_e)} \{R \subseteq \Sigma_e: (R \cup \Sigma_i) \cap \Sigma_e \in \text{ref}\,(E, p_e(s))\}\\
&= \text{ref}\,(E, s_e).
\end{aligned}$$

So, $P_e(E_{\rm con}) \sqsubseteq E$. The term $\rho(L(E_{\rm br}^{\uparrow}), s)$ is needed in the construction of $\text{ref}\,(E_{\rm con}, s)$ to guarantee that $E_{\rm con}$ is a process. The formal proof that it is a process is left to the reader. As $E_{\rm con}$ is a bounded recurrent process and $P_e(E_{\rm con}) \sqsubseteq E$, it follows that $E_{\rm con} \sqsubseteq E_{\rm br}^{\uparrow}$.

It remains to prove that $E_{\rm br}^{\uparrow} \sqsubseteq E_{\rm con}$. Suppose the relation does not hold. Then there must exist a process A such that $P_e(A) \sqsubseteq E, A$ is bounded recurrent, but A does not reduce $E_{\rm con}$. As $p_e(L(A)) \subseteq L(E)$ and A is bounded recurrent, it follows that $\forall s \in L(A), p_e(s) \in L(E)$ and s is bounded recurrent. Therefore, $L(A) \subseteq L(E_{\rm con})$. As $A \not\sqsubseteq E_{\rm con}$ there must exist an $s \in L(A)$ and a $R \in \text{ref}\,(A, s)$ such that $R \notin \text{ref}\,(E_{\rm con}, s)$, i.e., $\Sigma_i \subseteq R$ and $R \cap \Sigma_e \notin \text{ref}\,(E, p_e(s))$. But then $P_e(A) \not\sqsubseteq E$, which contradicts our assumptions. Hence $E_{\rm br}^{\uparrow} = E_{\rm con}$. ∎

Theorem 3: Let $G, S \in \Pi(\Sigma), E \in \Pi(\Sigma_e)$, and $E_{\rm br}^{\uparrow}$ be constructed as above

$$G||S \sqsubseteq E_{\rm br}^{\uparrow}$$
$$\Leftrightarrow$$
$$P_e(G||S) \sqsubseteq E \text{ and } S \text{ is bounded recurrent.}$$

Proof ($G||S \sqsubseteq E_{\rm br}^{\uparrow} \Rightarrow S$ is Bounded Recurrent): As $E_{\rm br}^{\uparrow}$ is bounded recurrent and $L(G||S) \subseteq L(E_{\rm br}^{\uparrow})$, it follows that $G||S$ is bounded recurrent. Let $s \in L(S) \cap L(G) = L(G||S)$, then s is bounded recurrent. If $s \in L(S) - L(G)$, then s is bounded recurrent by definition. Hence S is bounded recurrent.

($G||S \sqsubseteq E_{\rm br}^{\uparrow} \Rightarrow P_e(G||S) \sqsubseteq E$): The language part of the reduction relation follows from $L(P_e(G||S)) = p_e(L(G||S)) \subseteq p_e(L(E_{\rm br}^{\uparrow})) = L(E)$. For the refusal part note that $\text{div}\,(G||S, \Sigma_e) \subseteq \text{div}\,(E_{\rm br}^{\uparrow}, \Sigma_e)$ because $L(G||S) \subseteq L(E_{\rm br}^{\uparrow})$. Let $s_e \in p_e(L(G||S))$. If $s_e \in p_e(\text{div}\,(G||S, \Sigma_e))$, then $s_e \in p_e(\text{div}\,(E_{\rm br}^{\uparrow}, \Sigma_e))$, so

$$\begin{aligned}
\text{ref}\,(P_e(G||S), s_e) &= 2^{\Sigma_e}\\
&= \text{ref}\,(P_e(E_{\rm br}^{\uparrow}), s_e)\\
&= \text{ref}\,(E, s_e).
\end{aligned}$$

If $s_e \notin p_e(\text{div}\,(G||S, \Sigma_e))$, then

$$\begin{aligned}
\text{ref}\,(P_e(G||S), s_e) &= \bigcup_{s \in p_e^{-1}(s_e)} p_e(\text{ref}\,(G||S, s))\\
&\subseteq \bigcup_{s \in p_e^{-1}(s_e)} p_e(\text{ref}\,(E_{\rm br}^{\uparrow}, s))\\
&= \text{ref}\,(E, s_e).
\end{aligned}$$

The last step follows from the same line of reasoning as is used in the proofs of Propositions 3 and 4. The language part and the refusal part together prove that $P_e(G||S) \sqsubseteq E$.

($P_e(G||S) \sqsubseteq E$ and S is bounded recurrent $\Rightarrow G||S \sqsubseteq E_{\rm br}^{\uparrow}$.) If S is bounded recurrent then so is $G||S$ because $L(G||S) \subseteq L(S)$. As $E_{\rm br}^{\uparrow}$ is the supremal element of the set $\{A \in \Pi(\Sigma): P_e(A) \sqsubseteq E, A$ is bounded recurrent, it must hold that $G||S \sqsubseteq E_{\rm br}^{\uparrow}$. ∎

The following two results are proven in the Appendix. Theorem 4 states that the control problem with partial specification can be converted to a basic control problem with full specification as defined and solved in Section V.

Theorem 4: Let $G \in \Pi(\Sigma)$ and $E \in \Pi(\Sigma_e)$. There exists a complete supervisor $S \in \Pi(\Sigma)$, such that $P_e(G||S) \sqsubseteq E$ if and only if there exists a complete supervisor $S_{\rm br} \in \Pi(\Sigma)$ such that $G||S_{\rm br} \sqsubseteq E_{\rm br}^{\uparrow}$.

A condition for the existence of a complete supervisor $S_{\rm br}$ that solves $G||S_{\rm br} \sqsubseteq E_{\rm br}^{\uparrow}$ is given in Theorem 12. Corollary 1 states that the external language of the implementation is not restricted by the use of bounded recurrent supervisors.

Corollary 1: Let $S \in \Pi(\Sigma)$ be a complete supervisor such that $P_e(G||S) \sqsubseteq E$. Then there exists a complete and bounded recurrent supervisor $S_{\rm br} \in \Pi(\Sigma)$ such that $P_e(G||S_{\rm br}) \sqsubseteq E$ and $L(P_e(G||S)) = L(P_e(G||S_{\rm br}))$.

The next example will illustrate the followed approach.

Example 5: Let G and E be defined by the behavior state representations given in Fig. 4(a) and 4(b), respectively. Let $\Sigma = \{a, b, c, i, j\}, \Sigma_e = \{a, b, c\}, \Sigma_i = \{i, j\}, \Sigma_c = \Sigma$, and $\Sigma_u = \emptyset$. As E can refuse the whole external event set after the empty trace and after traces that end with a b-event or c-event, it is not necessary to bound the number of internal recurrences after these traces. After traces that end on an a-event the process E cannot refuse Σ_e, so the number of internal recurrences needs to be bounded. In Fig. 4(c) the behavior state representation of $E_{\rm br}^{\uparrow}$ is given. After the traces a, ai, aij, and $aiji$, the event set $\{b, i, j\}$ cannot be refused. Suppose this event set had been included in the refusal set of $E_{\rm br}^{\uparrow}$. Then $\{b\}$ would have been in the refusal

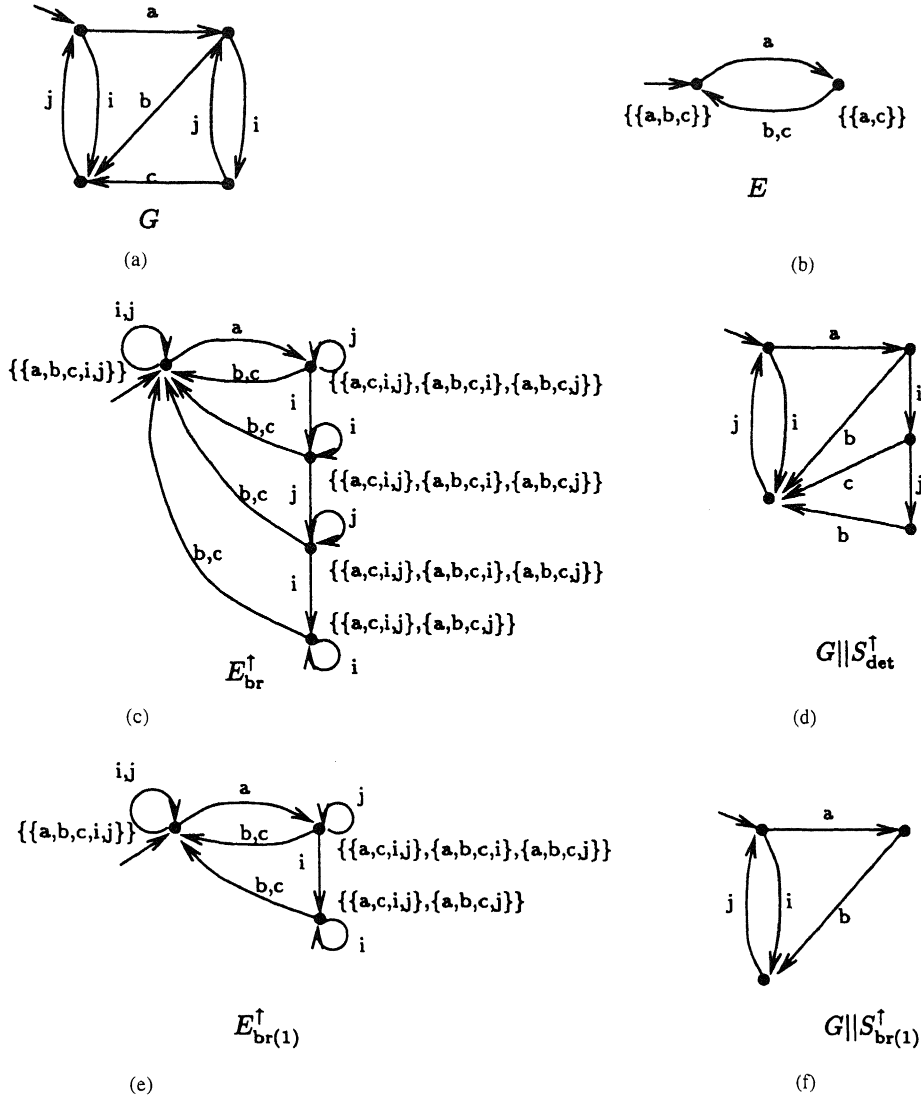


Fig. 4. Supervisor synthesis for the control problem with partial specification.

set ref $(P_e(E^{\uparrow}_{\rm br}), a)$ and process $P_e(E^{\uparrow}_{\rm br})$ would not have reduced E.

If event set $\{a, b, c, i\}$ had been an element of the refusal set ref $(E^{\uparrow}_{\rm br}, aiji)$, then by condition v of Definition 1 also $\{a, b, c, i, j\} = \{a, b, c, i\} \cup \rho(L(E^{\uparrow}_{\rm br}), aiji)$ would have been an element of ref $(E^{\uparrow}_{\rm br}, aiji)$. But then $P_e(E^{\uparrow}_{\rm br})$ could have refused $\{a, b, c\}$ which is not allowed by E. See also Proposition 4.

Event j in the states reached after traces a and aij, and event i in the states reached after traces ai and $aiji$, make self loops-because the traces $aj^*, aii^*, aijj^*$, and $aijii^*$ are not contained in $L(G)$ and are therefore, by Definition 15, bounded recurrent. Trace $aijij$ is not an element of $L(E^{\uparrow}_{\rm br})$ because $r_i(aijij) = 3$.

The synthesis procedure results in the least restrictive deterministic supervisor $S^{\uparrow}_{\rm det}$. In Fig. 4(d) only those traces of $S^{\uparrow}_{\rm det}$ are shown that are an element of $L(G)$. Observe that event i is disabled after trace aij. The reason for this can be explained as follows. Event set $\{b\}$ is an element of ref $(G, aiji)$ and therefore, by Definition 4, also of ref $(G||S, aiji)$ for all $S \in \Pi(\Sigma)$. As $E^{\uparrow}_{\rm br}$ does not allow any internal events to be executed after $aiji$, it follows that $\{b, i, j\} \in$ ref $(G||S, aiji)$ for all S such that $L(G||S) \subseteq L(E^{\uparrow}_{\rm br})$. However, this refusal is not allowed by $E^{\uparrow}_{\rm br}$. Therefore, event i needs to be disabled after trace aij.

Note that trace ac is an element of $L(P_e(G||S^{\uparrow}_{\rm det}))$.

In Fig. 4(e) the system $E^{\uparrow}_{\rm br(1)}$ is given, which contains only traces that make at most one loop of internal events in G. In Fig. 4(f) the least restrictive solution corresponding to the specification $E^{\uparrow}_{\rm br(1)}$ is given. Note that in this case $ac \notin L(P_e(G||S^{\uparrow}_{\rm br(1)}))$. So one loop is not sufficient to guarantee that all external behavior is obtainable.

Complexity

If E and G are given by behavior state representations, then computing $E^{\uparrow}_{\rm br}$ requires that each behavior state of E is replaced by either a behavior state with Σ_i self-loops if $\Sigma_e \in$ ref (E, s), or by a tree of behavior states, where on each path from root to leaf each behavior state of G occurs at most two times. The computation requires administration of the set of

states that are visited once and the set of states that are visited twice. In the worst case, all possible combinations of states can occur. This will result in complexity that is exponential in the size of the behavior state space of G and linear in the size of the behavior state space of E. Combining this with the supervisor synthesis algorithm from Section V will result in an algorithm which is exponential in $|G|$ and linear in $|E|$. When unwinding a loop of internal events in G, the algorithm has to administrate only those states that are reachable by internal events. So the algorithm will be exponential in the sizes of the sets of states reachable by internal events. In most practical systems these sets will be much smaller than the whole state space. Therefore, it is expected that the algorithm will behave better on practical systems than can be expected from the worst case analysis. Further research is needed to test the complexity of the algorithm in real life situations.

VII. Concluding Remarks

This paper extends the results of [2] as part of our investigation to set up a supervisory theory for nondeterministic systems.

The control problem that is discussed in this paper is to find a supervisor such that the controlled system can replace the specification. In particular, it is required that the controlled system cannot deadlock in situations in which the specification cannot deadlock either. It is shown that the reduction relation provides a necessary and sufficient condition to satisfy this requirement. This result indicates that a framework based on failures semantics is the most suitable for the given control problem.

It is imaginable that for some control problems a stronger relation between implementation and specification is required. For these control problems a stronger semantics, such as bisimulation semantics [20], will be necessary.

Unlike methods based on languages, failure semantics has the ability to deal with phenomena, such as divergence, that occur when processes are partially observed. Correct handling of divergence requires that solutions are restricted to bounded recurrent supervisors. Note that the restriction to bounded recurrent processes is not due to the use of failure semantics, but due to the nondeterministic properties of projected systems.

Appendix
Proof of Theorem 4 and Corollary 1

According to Theorem 3, a supervisor S_{br} solves the control problem $G||S \sqsubseteq E^{\uparrow}_{\text{br}}$ if and only if S_{br} is bounded recurrent and $P_e(G||S_{\text{br}}) \sqsubseteq E$. So, in order to prove Theorem 4 it is necessary and sufficient to show that there exists a solution to the control problem with partial specification if and only if there exists a bounded recurrent solution. The if-part is trivial because a bounded recurrent solution is a solution by itself. For the only-if part it will be shown that if there exists a solution then the language of the controlled system, denoted K, satisfies a number of conditions. From this language another language, denoted K_{br}, will be constructed that is bounded recurrent. It will be shown that K_{br} defines a bounded recurrent supervisor that solves the control problem. The core of the proof is formed by showing that K_{br} satisfies all the necessary conditions. First, it will be defined how the bounded recurrent language K_{br} can be constructed from a given language K.

Definition 18: Let $K \in \Sigma^*$ be a language, let $G \in \Pi(\Sigma)$ and $E \in \Pi(\Sigma_e)$ be processes. The Δ_i function gives all behavior states of process G that can be reached by internal events after trace s. All extensions have to be an element of the language K

$$\Delta_i(K,s) = \{G/ss_i \in \Pi(\Sigma) : \exists s_i \in \Sigma_i^+ \text{ s.t. } ss_i \in K\}.$$

The function $Q^{2\text{nd}}$ gives all states of process G that have been visited at least twice by the last internal part of trace s

$$\begin{aligned} Q^{2\text{nd}}(s) = \{A \in \Pi(\Sigma) : \exists s', s'' \in \overline{s}, s' \neq s'' \text{ s.t.} \\ p_e(s'') = p_e(s') = p_e(s) \wedge A = G/s' \\ = G/s''\}. \end{aligned}$$

The language K_{br} can be constructed inductively. Informally K_{br} behaves after one internal loop as K does in the last loop of internal events. Let $\varepsilon \in K_{\text{br}}$. Let $s_{\text{br}} \in K_{\text{br}}$. Then $s_{\text{br}}\sigma \in K_{\text{br}}$ if

$$\begin{aligned} \exists s \in K \text{ s.t } s\sigma \in K, \;\; G/s_{\text{br}} = G/s, \;\; p_e(s_{\text{br}}) = p_e(s), \text{ and} \\ \Sigma_e \notin \text{ref}\,(E, p_e(s_{\text{br}})) \Rightarrow \Delta_i(K,s) \cap Q^{2\text{nd}}(s_{\text{br}}) = \emptyset. \end{aligned}$$

The condition $\Delta_i(K,s) \cap Q^{2\text{nd}}(s_{\text{br}}) = \emptyset$ ensures that if K_{br} contains only internal continuations after s_{br} that are also internal continuations after s in K, then no state that has been visited twice by s_{br} will be visited again by these continuations. So no state will be visited more than two times by any internal part of any trace in K_{br}. Hence K_{br} will be bounded recurrent.

The following lemma states that if K satisfies certain conditions, then K_{br} satisfies the same conditions and is bounded recurrent.

Lemma 1: Let $K \subseteq \Sigma^*$ be a language which satisfies:

1) K is not empty;
2) K is prefix closed;
3) $K \subseteq L(G)$;
4) $p_e(K) \subseteq L(E)$;
5) K is controllable;
6) K is reducible w.r.t. $E^{\uparrow}$;
7) $s \in \text{div}\,(K, \Sigma_e) \Rightarrow \Sigma_e \in \text{ref}\,(E, p_e(s))$.

Then the language K_{br}, constructed as in Definition 18, satisfies the conditions 1)—7) and also the following conditions:

8) K_{br} is bounded recurrent;
9) $p_e(K) = p_e(K_{\text{br}})$.

The proof, which is given below, uses the following three lemmas. The first lemma redefines the controllability condition in failure semantics terminology

Lemma 2: Let K be a prefix-closed language contained in $L(G)$. Then K is controllable if and only if

$$\forall s \in K, \quad \rho(K,s) \cap \Sigma_u \subseteq \rho(L(G), s).$$

Proof:

$$
\begin{aligned}
&K\Sigma_u \cap L(G) \subseteq K && \Leftrightarrow\\
&(s \in K \wedge \sigma \in \Sigma_u \wedge s\sigma \in L(G)) \Rightarrow s\sigma \in K && \Leftrightarrow\\
&\neg(s \in K \wedge \sigma \in \Sigma_u \wedge s\sigma \in L(G)) \vee s\sigma \in K && \Leftrightarrow\\
&\neg s \in K \vee \neg\sigma \in \Sigma_u \vee \neg s\sigma \in L(G) \vee s\sigma \in K && \Leftrightarrow\\
&\neg s \in K \vee \neg\sigma \in \Sigma_u \vee \neg s\sigma \notin K \vee s\sigma \notin L(G) && \Leftrightarrow\\
&(s \in K \wedge \sigma \in \Sigma_u \wedge s\sigma \notin K) \Rightarrow s\sigma \notin L(G) && \Leftrightarrow\\
&\forall s \in K, \quad \rho(K,s) \cap \Sigma_u \subseteq \rho(L(G),s).
\end{aligned}
$$

■

The following lemma states that $K_{\rm br}$ does not block more events after trace $s_{\rm br}$ than K blocks after a corresponding trace s.

Lemma 3: Assume language $K \subseteq \Sigma^*$ satisfies conditions 1)—7) of Lemma 1, and $K_{\rm br}$ is constructed according to Definition 18. Let $s_{\rm br} \in K_{\rm br}$. There exists an $s \in K$ such that $G/s = G/s_{\rm br}, p_e(s) = p_e(s_{\rm br})$, and $\rho(K_{\rm br}, s_{\rm br}) \subseteq \rho(K,s)$.

Proof: Let $s_{\rm br} \in K_{\rm br}$. If $s_{\rm br} = \varepsilon$, then let $s = \varepsilon$. It follows that $s \in K, G/s_{\rm br} = G/\varepsilon = G/s$, and $p_e(s_{\rm br}) = p_e(\varepsilon) = p_e(s)$. As $Q^{\rm 2nd}(s_{\rm br}) = \emptyset$, it follows that $s\sigma \in K$ implies that $s_{\rm br}\sigma \in K_{\rm br}$. So $\lambda(K,s) \subseteq \lambda(K_{\rm br}, s_{\rm br})$ and $\rho(K_{\rm br}, s_{\rm br}) \subseteq \rho(K,s)$

If $s_{\rm br}$ is not the empty trace, then $s_{\rm br}$ can be written as $v_{\rm br}\sigma$, with $v_{\rm br} \in K_{\rm br}$ and $\sigma \in \Sigma$. According to the definition of $K_{\rm br}$, and because $s_{\rm br} \in K_{\rm br}$, there must exist an $v \in K$ such that $v\sigma \in K, G/v = G/v_{\rm br}$, and $p_e(v) = p_e(v_{\rm br})$. Then also $G/s_{\rm br} = G/v_{\rm br}\sigma = G/v\sigma$ and $p_e(s_{\rm br}) = p_e(v\sigma) = p_e(v_{\rm br}\sigma)$. Hence for all $s_{\rm br} \in K_{\rm br}$ there exists an $s \in K$ such that $G/s = G/s_{\rm br}$ and $p_e(s) = p_e(s_{\rm br})$.

If $\Sigma_e \in {\rm ref}\,(E, p_e(s_{\rm br}))$ or $\Delta_i(K,s) \cap Q^{\rm 2nd}(s_{\rm br}) = \emptyset$, then $s\sigma \in K$ implies $s_{\rm br}\sigma \in K_{\rm br}$. So $\lambda(K,s) \subseteq \lambda(K_{\rm br}, s_{\rm br})$ and hence $\rho(K_{\rm br}, s_{\rm br}) \subseteq \rho(K,s)$.

We will prove by contradiction that there always exists an $s \in K$ such that $G/s = G/s_{\rm br}, p_e(s) = p_e(s_{\rm br})$ and either $\Sigma_e \in {\rm ref}\,(E, p_e(s_{\rm br}))$ or $\Delta_i(K,s) \cap Q^{\rm 2nd}(s_{\rm br}) = \emptyset$.

Assume (a) $\Sigma_e \notin {\rm ref}\,(E, p_e(s_{\rm br}))$ and (b) for all $s \in K$ such that $G/s = G/s_{\rm br}$ and $p_e(s) = p_e(s_{\rm br})$; it holds that $\Delta_i(K,s) \cap Q^{\rm 2nd}(s_{\rm br}) \neq \emptyset$. Let $v_{\rm br}$ be such that $v_{\rm br}\sigma_i = s_{\rm br}$. Note that $\sigma_i \in \Sigma_i$ because from $Q^{\rm 2nd}(s_{\rm br}) \neq \emptyset$ it follows that the last internal part of $s_{\rm br}$ is not empty. From the definition of $K_{\rm br}$ it follows that there exists a $v \in K$ such that $v\sigma_i \in K, G/v = G/v_{\rm br}, p_e(v) = p_e(v_{\rm br})$, and $\Delta_i(K,v) \cap Q^{\rm 2nd}(v_{\rm br}) = \emptyset$. Let $s = v\sigma_i$. Then $G/s = G/v\sigma_i = G/v_{\rm br}\sigma_i = G/s_{\rm br}$ and $p_e(s) = p_e(v\sigma_i) = p_e(v_{\rm br}\sigma_i) = p_e(s_{\rm br})$, so by assumption (b) $\Delta_i(K,s) \cap Q^{\rm 2nd}(s_{\rm br}) \neq \emptyset$. As $\Delta_i(K,s) \subseteq \Delta_i(K,v)$ and $Q^{\rm 2nd}(s_{\rm br}) \subseteq Q^{\rm 2nd}(v_{\rm br}) \cup \{G/s_{\rm br}\}$, it follows that $\Delta_i(K,s) \cap Q^{\rm 2nd}(s_{\rm br}) = \{G/s_{\rm br}\}$. Next it will be shown that $s \in {\rm div}\,(K, \Sigma_e)$

$$
\begin{aligned}
&\Delta_i(K,s) \cap Q^{\rm 2nd}(s_{\rm br}) = \{G/s_{\rm br}\} && \Rightarrow\\
&G/s_{\rm br} \in \Delta_i(K,s) && \Rightarrow\\
&\exists s_i \in \Sigma_i^+ \text{ s.t. } ss_i \in K \text{ and } G/ss_i = G/s_{\rm br} && \Rightarrow\\
&\qquad\qquad \text{(By assumption } (b))\\
&\Delta_i(K, ss_i) \cap Q^{\rm 2nd}(s_{\rm br}) \neq \emptyset && \Rightarrow\\
&\qquad\qquad \text{(Note: } \Delta_i(K,ss_i) \subseteq \Delta_i(K,s))\\
&G/s_{\rm br} \in \Delta_i(K, ss_i) && \Rightarrow\\
&\exists s_i' \in \Sigma_i^+ \text{ s.t. } ss_is_i' \in K \text{ and } G/ss_is_i' = G/s_{\rm br} && \Rightarrow\\
&\ldots.
\end{aligned}
$$

It follows that an unbounded number of internal events can be executed. So $s \in {\rm div}\,(K, \Sigma_e)$, which, by point 7) of Lemma 1, contradicts assumption (a) that $\Sigma_e \notin {\rm ref}\,(E, p_e(s_{\rm br}))$. Hence there always exists an $s \in K$ such that either $\Sigma_e \in {\rm ref}\,(E, p_e(s_{\rm br}))$ or $\Delta_i(K,s) \cap Q^{\rm 2nd}(s_{\rm br}) = \emptyset$. So $\lambda(K,s) \subseteq \lambda(K_{\rm br}, s_{\rm br})$ and $\rho(K_{\rm br}, s_{\rm br}) \subseteq \rho(K,s)$. ■

Lemma 4: Let language $K \subseteq \Sigma^*$ satisfy conditions 1)–7) of Lemma 1, and let $K_{\rm br}$ be constructed according to Definition 18. Then $p_e(K) \subseteq p_e(K_{\rm br})$.

Proof: We will prove by induction on the length of the traces that for all $s \in K$ there exists an $s_{\rm br} \in K_{\rm br}$ such that $p_e(s) = p_e(s_{\rm br}), G/s = G/s_{\rm br}$, and $Q^{\rm 2nd}(s_{\rm br}) = \emptyset$. The initial step is trivial because $\varepsilon \in K, \varepsilon \in K_{\rm br}$, and $Q^{\rm 2nd}(\varepsilon) = \emptyset$.

For the inductive step assume $s \in K$ and $s_{\rm br} \in K_{\rm br}$ such that $p_e(s) = p_e(s_{\rm br}), G/s = G/s_{\rm br}$, and $Q^{\rm 2nd}(s_{\rm br}) = \emptyset$. Let $s\sigma \in K$. We have to prove that there exists a $v_{\rm br} \in K_{\rm br}$ such that $p_e(s\sigma) = p_e(v_{\rm br}), G/s\sigma = G/v_{\rm br}$, and $Q^{\rm 2nd}(v_{\rm br}) = \emptyset$. As $Q^{\rm 2nd}(s_{\rm br}) = \emptyset$, it follows that $\Delta_i(K,s) \cap Q^{\rm 2nd}(s_{\rm br}) = \emptyset$, so $s\sigma \in K$ implies $s_{\rm br}\sigma \in K_{\rm br}$. From the definition of $Q^{\rm 2nd}$ it follows that $Q^{\rm 2nd}(s_{\rm br}\sigma) \subseteq Q^{\rm 2nd}(s_{\rm br}) \cup \{G/s_{\rm br}\sigma\} = \{G/s_{\rm br}\sigma\}$. So $Q^{\rm 2nd}(s_{\rm br}\sigma)$ contains at most one element. If $Q^{\rm 2nd}(s_{\rm br}\sigma) = \emptyset$, then $v_{\rm br} = s_{\rm br}\sigma$ satisfies the necessary conditions for the inductive step because $p_e(v_{\rm br}) = p_e(s_{\rm br}\sigma) = p_e(s\sigma), G/v_{\rm br} = G/s_{\rm br}\sigma = G/s\sigma$, and $Q^{\rm 2nd}(v_{\rm br}) = Q^{\rm 2nd}(s_{\rm br}\sigma) = \emptyset$.

If $Q^{\rm 2nd}(s_{\rm br}\sigma) = \{G/s_{\rm br}\sigma\}$, then the behavior state $G/s_{\rm br}\sigma$ has been visited at least twice by the last internal part of $s_{\rm br}\sigma$. Hence $G/s_{\rm br}\sigma$ has been visited at least once by the last internal part of $s_{\rm br}$. So, there exists a $v_{\rm br} \in \overline{s_{\rm br}}$ (note: $\overline{s_{\rm br}}$, *not* $\overline{s_{\rm br}\sigma}$), such that $G/v_{\rm br} = G/s_{\rm br}\sigma = G/s\sigma$ and $p_e(v_{\rm br} = p_e(s_{\rm br}\sigma) = p_e(s\sigma)$. As $v_{\rm br}$ is a prefix of $s_{\rm br}$, and $Q^{\rm 2nd}(s_{\rm br}) = \emptyset$, it has to hold that $Q^{\rm 2nd}(v) \subseteq Q^{\rm 2nd}(s_{\rm br}) = \emptyset$. Hence $v_{\rm br}$ satisfies the necessary conditions for the inductive step.

We have proven that for all $s \in K$ there exists a trace in $s_{\rm br} \in K_{\rm br}$ such that $p_e(s) = p_e(s_{\rm br}), G/s = G/s_{\rm br}$, and $Q^{\rm 2nd}(s_{\rm br}) = \emptyset$. The first equality implies that $p_e(K) \subseteq p_e(K_{\rm br})$. ■

Proof (Lemma 1):

1), 2): These points follow directly from the definition of $K_{\rm br}$.

3): We will prove this point by induction. The initial step is satisfied because $\varepsilon \in L(G)$. For the inductive step let $s_{\rm br} \in K_{\rm br} \cap L(G)$. If $s_{\rm br}\sigma \in K_{\rm br}$, then there exists an $s \in K$ such that $s\sigma \in K$ and $G/s = G/s_{\rm br}$. As $s\sigma \in K \subseteq L(G)$ it follows that $G/s_{\rm br}\sigma = G/s\sigma$ is well defined, so $s_{\rm br}\sigma \in L(G)$.

4): From Lemma 3 it follows that for all $s_{\rm br} \in K_{\rm br}$ there exists an $s \in K$ such that $p_e(s_{\rm br}) = p_e(s)$. So $p_e(K_{\rm br}) \subseteq p_e(K) \subseteq L(E)$.

5): Let $s_{\rm br} \in K_{\rm br}$. By Lemma 3 there exists an $s \in K$ such that $G/s_{\rm br} = G/s$ and $\rho(K_{\rm br}, s_{\rm br}) \subseteq \rho(K,s)$. So $\rho(K_{\rm br}, s_{\rm br}) \cap \Sigma_u \subseteq \rho(K,s) \cap \Sigma_u \subseteq \rho(L(G),s) = \rho(L(G), s_{\rm br})$.

6): Let $s_{\rm br} \in K_{\rm br}$. By Lemma 3 there exists an $s \in K$ such that $G/s_{\rm br} = G/s, p_e(s_{\rm br}) = p_e(s)$ and $\rho(K_{\rm br}, s_{\rm br}) \subseteq \rho(K,s)$. As $p_e(s_{\rm br}) =$

$p_e(s)$, it follows that $E/p_e(s_{\rm br}) = E/p_e(s)$, so ref$(E^\uparrow, s_{\rm br}) =$ ref$(E^\uparrow, s)$. Let $R_g \in$ ref$(G, s_{\rm br})$. It follows from $G/s_{\rm br} = G/s$ that ref$(G, s_{\rm br}) =$ ref(G, s), so $R_g \in$ ref(G, s). By reducibility of $K, \rho(K, s) \cup R_g \in$ ref$(E^\uparrow, s)$. So, because refusal sets are closed under the operation of taking subsets and $\rho(K_{\rm br}, s_{\rm br}) \subseteq \rho(K, s)$, it follows that $\rho(K_{\rm br}, s_{\rm br}) \cup R_g \in$ ref$(E^\uparrow, s) =$ ref$(E^\uparrow, s_{\rm br})$.

7): This follows directly from the fact that $K_{\rm br}$ is bounded recurrent.

8): Suppose $K_{\rm br}$ is not bounded recurrent. Then there exists an $s_{\rm br} \in K_{\rm br}$ such that $\Sigma_e \notin$ ref$(E, p_e(s)$ and $r_i(s_{\rm br}) \geq 3$. It follows that $s_{\rm br}$ can be written as $v_{\rm br}\sigma_i$, where $v_{\rm br} \in K_{\rm br}$ and $\sigma_i \in \Sigma_i$. The behavior state $G/s_{\rm br}$ is visited at least three times by the last internal part of $s_{\rm br}$. One visit is by the trace $s_{\rm br}$ itself, so that the state $G/s_{\rm br}$ is visited at least twice by the last internal part of trace $v_{\rm br}$. That is

$$\begin{aligned}|\{s' \in \overline{v}_{\rm br}: G/s' = G/s_{\rm br}, \quad p_e(s') = p_e(v_{\rm br})\}| \\ = r_i(s_{\rm br}) - 1 \\ \geq 2.\end{aligned}$$

It follows that $G/s_{\rm br} \in Q^{\rm 2nd}(v_{\rm br})$. According to Lemma 3 there exists a $v \in K$ such that $G/v = G/v_{\rm br}$ and $p_e(v) = p_e(v_{\rm br})$. If $v\sigma_i \in K$ then $G/v\sigma_i \in \Delta_i(K, v)$. So in addition, $G/s_{\rm br} = G/v_{\rm br}\sigma_i = G/v\sigma_i \in \Delta_i(K, v)$. Hence for all $v \in K$ such that $v\sigma_i \in K$, we have that $G/s_{\rm br} \in \Delta_i(K, v) \cap Q^{\rm 2nd}(v_{\rm br}) \neq \emptyset$. So we have to conclude that $s_{\rm br} \notin K_{\rm br}$, which contradicts our assumptions.

9): The inclusion $p_e(K_{\rm br}) \subseteq p_e(K)$ follows from Lemma 3. The inclusion $p_e(K) \subseteq p_e(K_{\rm br})$ follows from Lemma 4. ■

Proof (Theorem 4 and Corollary 1):

(Only-If Part): First we will show that $L(G||S)$ satisfies properties 1)–7) of Lemma 1. Properties 1)–5) are easy to see. It follows from Definition 16 that $G||S \sqsubseteq E^\uparrow$, so $L(G||S)$ is reducible w.r.t. $E^\uparrow$ [point 6)]. For point 7) let $s \in$ div$(G||S, s)$. Then $\Sigma_e \in$ ref$(P_e(G||S), p_e(s)) \subseteq$ ref$(E, p_e(s))$.

As $L(G||S)$ satisfies points 1)–7), there must exist a language $K_{\rm br}$ that satisfies points 1)–9). Let $S_{\rm br} =$ Det$(K_{\rm br})$. Then it follows from point 6) that $G||S_{\rm br} \sqsubseteq E^\uparrow$. Combining this with point 8) implies that $G||S_{\rm br} \sqsubseteq E^\uparrow_{\rm br}$. Corollary 1 now follows from point 9).

(If-Part): This follows directly from Theorem 3.

REFERENCES

[1] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proc. IEEE*, vol. 77, pp. 81–98, 1989.

[2] A. Overkamp, "Supervisory control for nondeterministic systems," in [21], pp. 59–65.

[3] C. A. R. Hoare, *Communicating Sequential Processes.* Englewood Cliffs, NJ: Prentice-Hall, 1985.

[4] A. Overkamp, "Control of nondeterministic discrete event systems using failure semantics," in *Proc. 3rd European Contr. Conf.*, Rome, Italy, 1995, pp. 2778–2783.

[5] A. Tanenbaum, *Computer Networks*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[6] M. A. Shayman and R. Kumar, "Supervisory control of nondeterministic systems with driven events via prioritized synchronization and trajectory models," *SIAM J. Contr. Optimiz.*, vol. 33, no. 2, pp. 469–497, 1995.

[7] M. D. DiBenedetto, A. Saldanha, and A. Sangiovanni-Vincentelli, "Model matching for finite state machines," in *Proc. 33th Conf. Decision Contr.*, Orlando, FL, 1994, pp. 3117–3124.

[8] K. Inan, "Nondeterministic supervision under partial observation," in [21], pp. 39–48.

[9] M. Heymann, "Concurrency and discrete event control," *IEEE Contr. Syst. Mag.*, vol. 10, no. 4, pp. 103–112, 1990.

[10] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* New York: Addison-Wesley, 1979.

[11] E. Haghverdi and K. Inan, "Verification by consecutive projections," in *FORTE '92, Proc. IFIP,* Perros-Guirec, M. Diaz, and R. Groz, Eds., pp. 465–478.

[12] R. De Nicola and M. Hennessy, "Testing equivalences for processes," *Theoretical Computer Sci.*, vol. 34, pp. 83–133, 1984.

[13] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe, "A theory of communicating sequential processes," *J. ACM*, vol. 31, no. 3, pp. 560–599, 1984.

[14] W. M. Wonham and P. J. Ramadge, "Modular supervisory control of discrete-event systems," *Math. Contr., Signals Syst.*, vol. 1, no. 1, pp. 13–30, 1988.

[15] E. A. Emerson and E. M. Clarke, "Using branching time temporal logic to synthesize synchronization skeletons," *Sci. Computer Programming*, vol. 2, pp. 241–266, 1982.

[16] Z. Manna and P. Wolper, "Synthesis of communicating processes from temporal logic specifications," *ACM Trans. Programming Languages Syst.*, vol. 6, no. 1, pp. 68–93, 1984.

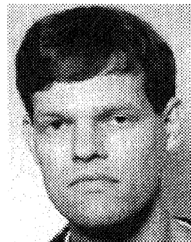
[17] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proc. 16th ACM Symp. Principles Programming Languages*, 1989, pp. 179–190.

[18] J. G. Thistle, "Control of infinite behavior of discrete-event systems," Ph.D. dissertation, Univ. Toronto, 1991; also available as Systems Control Group Rep. 9012.

[19] R. Kumar, V. Garg, and S. I. Marcus, "On controllability and normality of discrete event dynamical systems," *Syst. Contr. Lett.*, vol. 17, no. 3, pp. 157–168, 1991.

[20] R. Milner, *A Calculus of Communicating Systems.* New York: Springer, 1980.

[21] G. Cohen and J.-P. Quadrat, Eds., *Proc. 11th Int. Conf. Analysis Optimization Syst., Discrete Event Syst.,* Sophia-Antipolis, 1994, Lecture Notes in Control and Information Sciences 199. New York: Springer, 1994.



Ard Overkamp was at CWI, Amsterdam, The Netherlands as a Ph.D. Student from 1992–1996. During this period he worked on supervisory control theory motivated by design problems for layered network architectures. He is now working for a consultancy company.