SYNTHESIZING DETERMINISTIC CONTROLLERS IN SUPERVISORY CONTROL

Andreas Morgenstern and Klaus Schneider
University of Kaiserslautern, Department of Computer Science
P.O. Box 3049, 67653 Kaiserslautern, Germany

Keywords: Controller Synthesis, Supervisory Control, Discrete Event Systems.

Abstract: Supervisory control theory for discrete event systems is based on finite state automata whose inputs are partitioned into controllable and uncontrollable events. Well-known algorithms used in the Ramadge-Wonham framework disable or enable controllable events such that it is finally possible to reach designated final states from every reachable state. However, as these algorithms compute the least restriction on controllable events, their result is usually a nondeterministic automaton that can not be directly implemented. For this reason, one distinguishes between supervisors (directly generated by supervisory control) and controllers that are further restrictions of supervisors to achieve determinism. Unfortunately, controllers that are generated from a supervisor may be blocking, even if the underlying discrete event system is nonblocking. In this paper, we give a modification of a supervisor synthesis algorithm that enables us to derive deterministic controllers. Moreover, we show that the algorithm is both correct and complete, i.e., that it generates a deterministic controller whenever one exists.

1 INTRODUCTION

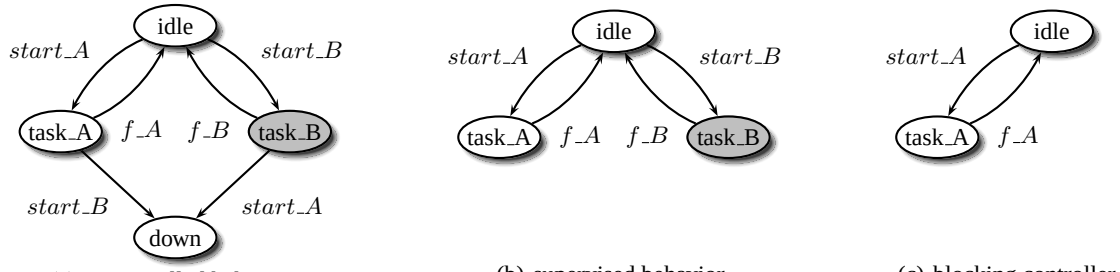
New applications in safety critical areas require the verification of the developed systems. In the past two decades, a lot of verification methods for checking the temporal behavior of a system have been developed (Schneider, 2003), and the research lead to tools that are already used in industrial design flows. These tools are able to check whether a system $\mathcal{K}$ satisfies a given temporal specification φ. There are a lot of formalisms, in particular, the μ-calculus (Kozen, 1983), ω-automata (Thomas, 1990), as well as temporal (Pnueli, 1977; Emerson and Clarke, 1982; Emerson, 1990) and predicate logics (Büchi, 1960b; Büchi, 1960a) to formulate the specification φ (Schneider, 2003). Moreover, industrial interest lead already to standardization efforts on specification logics (Accellera, 2004).

Besides the verification problem, where the entire system $\mathcal{K}$ and its specification must be already available, one can also consider the controller synthesis problem. The task is here to check whether there is a system $\mathcal{C}$ such that the coupled system $\mathcal{K} \parallel \mathcal{C}$ satisfies φ. Obviously, this problem is more general than the verification problem. Efficient solutions for this problem could be naturally used to guide the development of finite state controllers.

The controller synthesis problem is not new; several approaches exist for the so-called supervisory control problem. In particular, the supervisory control theory initiated by Ramadge and Wonham (Ramadge and Wonham, 1987) provides a framework for the control of discrete event systems. The system (also called a plant) is thereby modeled as a generator of a formal language. The control feature is represented by the fact that certain events can be disabled by a so-called supervisor. One result of supervisory control theory is that in case of formal languages, i.e., finite state machines, such a supervisor can be effectively computed.

However, if an implementation has to be derived from a supervisor, several problems have to be solved (Dietrich et al., 2002; Malik, 2003). A particular problem that we consider in this paper is the *derivation of a deterministic controller* from a supervisor that guarantees the *nonblocking property*. A system is thereby called nonblocking, if it is always possible to complete some task, i.e. to reach some designated (marked) state from every reachable state. If we consider the events as signals that can be sent to the plant, a valid controller should decide in every state what signal should be sent to the plant to ensure that the marked state is actually reached. However, even if the generated supervisor is nonblocking, a controller

Morgenstern A. and Schneider K. (2005).
SYNTHESIZING DETERMINISTIC CONTROLLERS IN SUPERVISORY CONTROL.
In *Proceedings of the Second International Conference on Informatics in Control, Automation and Robotics - Robotics and Automation*, pages 24-31
DOI: 10.5220/0001164500240031



(a) uncontrolled behavior (b) supervised behavior (c) blocking controller

Figure 1: Generation of a Blocking Controller

that is derived by simply selecting in each state one of the allowed events/signals could be blocking.

As an example, consider the automaton that is given in Figure 1(a). This automaton represents a system with two tasks *task_A* and *task_B* that can be started with events *start_A* and *start_B*, respectively. These events are controllable, i.e. they can be disabled by a supervisor. If one of the machines completes its task, the (uncontrollable) events *f_A* and *f_B* occur, respectively, leading again to the initial state *idle*. Whenever both machines work at the same time, the system breaks down, since the state *down* is reached from where on no further progress is possible. Supervisory control theory can fix the problem that state *down* is reached by disabling events *start_B* in state *task_A* and *start_A* in state *task_B* (Figure 1(b)). However, when we have to implement a *deterministic* controller that has to select one of the signals *start_A* and *start_B*, we get a serious problem: if the controller always selects *start_A*, the marked state *task_B* is never reached, and therefore the nonblocking property is violated (Figure 1(c)).

In (Malik, 2003; Dietrich et al., 2002), the generation of deterministic controllers is restricted to cases where certain conditions hold. It is proved that these conditions guarantee that *every* deterministic controller derived from the supervisor is nonblocking. However, no controller can be constructed in case the discrete event system does not satisfy these conditions. In particular, a valid controller may exist, even if the conditions of (Malik, 2003; Dietrich et al., 2002) do not hold. For example, this is the case for the automaton given in Figure 1. A valid controller is obtained by selecting *start_B* in state *idle*.

In this paper, *we present a new approach to generate deterministic controllers from supervisors* that does not suffer from the above problem. To this end, we introduce a more general property than nonblocking which we call *forceable nonblocking*. A discrete event system satisfies this property if and only if there exists a deterministic controller that ensures that every run (either finite or infinite) of the controlled system visits a marked state. Obviously, this requirement is stronger than the nonblocking property. Our algorithm guarantees that a marked state will be reached, no matter how the plant behaves. In contrast, the nonblocking property only requires that the plant *has the chance* to reach a marked state. Although our property is more general than nonblocking, our algorithm is just a slight adjustment of the original supervisor synthesis algorithm which is known to have moderate complexity bounds.

The paper is organized as follows: In the next Section, we present the basics of supervisory control theory. In Section 3, we present our new algorithm to compute deterministic nonblocking controllers from supervisors whenever this is possible. Finally, the paper ends with some conclusions and directions for future work.

2 SUPERVISORY CONTROL THEORY

In this section, we will give a brief introduction to the supervisory control theory as initiated by Ramadge and Wonham (Ramadge and Wonham, 1987). For a more detailed treatment of the topic we refer to (Wonham, 2001).

Traditionally, control theory has focused on control of systems modeled by differential equations, so-called continuous variable dynamic systems. There, the feedback signal from the controller influences the behavior of the system, enforcing a given specification that would not be met by the open-loop behavior. Another important class of system models are those where states have symbolic values instead of numerical ones. These systems change their state whenever an external or internal event occurs. This class of systems, called *discrete event systems (DES)*, is the focus of supervisory control theory (Ramadge and Wonham, 1987).

The theoretical roots of supervisory control theory explain some of the terminology used. In the Ramadge Wonham (RW) framework, one speaks of a

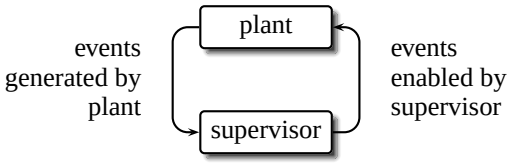


Figure 2: The Ramadge-Wonham Framework

plant, a system which generates events and encompasses the whole physically possible behavior of the system to be controlled (including unwanted situations). The *specification* is a subset of this behavior that should be matched by adding a controller. A *supervisor* is an entity that is coupled with the plant through a communication channel that allows the supervisor to influence the behavior of the plant by enabling those events that may be generated in the next state of the system (see Figure 2). Usually, in a physical system, not all of the events can be influenced by an external supervisor. This is captured by distinguishing between events that can be prevented from occurring, called *controllable events*, and those that cannot be prevented, called *uncontrollable events*. We denote the sets of uncontrollable and controllable events as Σ_u and Σ_c, respectively, and define $\Sigma = \Sigma_c \cup \Sigma_u$.

The Ramadge Wonham formulation of the supervisory control problem makes use of formal language theory and automata: A finite automaton is a 5-tuple $\mathcal{A} = \langle Q, \Sigma, \delta, q^0, M\rangle$ where Σ is a set of events, Q is a set of states, $\delta : Q \times \Sigma \rightarrow Q$ is a transition function, and $q^0 \in Q$ is the initial state. The states in the set $M \subseteq Q$ are chosen to mark the completion of tasks by the system and are therefore called *marker states*. We write $\delta(q, \sigma) \downarrow$ to signify that there exists a transition labeled with σ, leaving q. It is often necessary to refer to the set of events for which there is a transition leaving state q. We refer to these events as active events:

Definition 1 (Active Events) *Given an automaton $\mathcal{A} = \langle Q, \Sigma, \delta, q^0, M\rangle$ and a particular state $q \in Q$, the set of active events of q is:*

$$\mathsf{act}_{\mathcal{A}}(q) := \{\sigma \in \Sigma \mid \delta(q, \sigma) \downarrow\}$$

If the plant and the supervisor are represented using finite automata, the control action of the supervisor is captured by the synchronous product:

Definition 2 (Automata Product) *Given automata $\mathcal{A}_\mathcal{P} = \langle \Sigma, Q_\mathcal{P}, \delta_\mathcal{P}, q^0_\mathcal{P}\ M_\mathcal{P}\rangle$ and $\mathcal{A}_\mathcal{S} = \langle \Sigma, Q_\mathcal{S}, \delta_\mathcal{S}, q^0_\mathcal{S}, M_\mathcal{S}\rangle$, the product $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{S}$ is the automaton $\langle \Sigma, Q_\mathcal{P} \times Q_\mathcal{S}, \delta_{\mathcal{P}\times\mathcal{S}}, (q^0_\mathcal{P}, q^0_\mathcal{S}), M_\mathcal{P} \times M_\mathcal{S}\rangle$, where*

$$\delta_{\mathcal{P}\times\mathcal{S}}((p,q),\sigma) = (p',q') \text{ iff } \delta_\mathcal{P}(p,\sigma) = p' \wedge \delta_\mathcal{S}(q,\sigma) = q'$$

Note that in a state (p, q) of the synchronous product, the active events are exactly those events that are active both in the plant and the supervisor,i.e.

$$\mathsf{act}_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{S}}((p,q)) = \mathsf{act}_{\mathcal{A}_\mathcal{P}}(p) \cap \mathsf{act}_{\mathcal{A}_\mathcal{S}}(q).$$

Disabling controllable events in the states of the supervisor will therefore also disable them in the product. This is how the supervisor enforces his control function.

The behavior of the plant represented by a finite automaton is closely related to two formal languages over the alphabet of events Σ, the generated language $\mathsf{L}(\mathcal{A})$ and the marked language $\mathsf{L_m}(\mathcal{A})$. The generated language $\mathsf{L}(\mathcal{A})$ represents sequences of events that the plant generates during execution while the marked language $\mathsf{L_m}(\mathcal{A})$ represents those event sequences that lead to a marker state. Formally, the two languages are defined as follows[1]:

Definition 3 (Generated and Marked Language)

$$\mathsf{L}(\mathcal{A}) = \{w \in \Sigma^* : \delta(q^0, w) \downarrow\}$$
$$\mathsf{L_m}(\mathcal{A}) = \{w \in \Sigma^* : \delta(q^0, w) \in M\}$$

Given both the plant $\mathcal{A}_\mathcal{P}$ and the supervisor $\mathcal{A}_\mathcal{S}$, the generated and marked language of the controlled system are denoted by $\mathsf{L}(\mathcal{A}_\mathcal{P}/\mathcal{A}_\mathcal{S})$ and $\mathsf{L_m}(\mathcal{A}_\mathcal{P}/\mathcal{A}_\mathcal{S})$ and defined by the generated and marked language of the product automaton:

$$\mathsf{L}(\mathcal{A}_\mathcal{P}/\mathcal{A}_\mathcal{S}) := \mathsf{L}(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{S}) = \mathsf{L}(\mathcal{A}_\mathcal{P}) \cap \mathsf{L}(\mathcal{A}_\mathcal{S})$$

$$\begin{aligned}\mathsf{L_m}(\mathcal{A}_\mathcal{P}/\mathcal{A}_\mathcal{S}) :&= \mathsf{L_m}(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{S}) \\ &= \mathsf{L_m}(\mathcal{A}_\mathcal{P}) \cap \mathsf{L_m}(\mathcal{A}_\mathcal{S})\end{aligned}$$

When we consider algorithms, it is also necessary that the specification is given as a finite automaton. The assumption that the uncontrollable events can not be prevented from occurring, places restrictions on the possible supervisors. Therefore, a specification automaton $\mathcal{A}_\mathcal{E}$ is called *controllable with respect to a plant $\mathcal{A}_\mathcal{P}$*, if and only if for every state (p, q) of $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ that is reachable by a string in $\mathsf{L}(\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E})$ and every uncontrollable event $\sigma \in \Sigma_u$, the following holds:

$$\sigma \in \mathsf{act}_{\mathcal{A}_\mathcal{P}}(p) \Rightarrow \sigma \in \mathsf{act}_{\mathcal{A}_\mathcal{E}}(q).$$

In other words, $\mathcal{A}_\mathcal{E}$ is controllable if and only if no word of $\mathsf{L}(\mathcal{A}_\mathcal{P})$ that is generated according to the specification, exits from the behavior permitted by the specification if it is followed by an uncontrollable event. Specifications that do not fulfill this requirement are called *uncontrollable*. If a specification is uncontrollable, the product automaton contains one or more reachable *bad states*, which are states (p, q) that fail to satisfy the following condition:

[1] As usual, we allow δ to process also words instead of only single events.

$$\mathsf{act}_{\mathcal{A}_\mathcal{P}\times\mathcal{A}_\mathcal{E}}((p,q)) \supseteq \mathsf{act}_{\mathcal{A}_\mathcal{P}}(p) \cap \Sigma_u$$

Given a specification automaton $\mathcal{A}_\mathcal{E}$, the language $K = \mathsf{L_m}(\mathcal{A}_\mathcal{E})$ is *controllable* if and only if $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$ has no bad states. Besides controllability, another important property of discrete event systems is the *nonblocking* property which states that it is always possible to complete some task, i.e. that from every reachable state $q \in Q$, it is possible to reach a marked state. Formally, an automaton is nonblocking, if and only if for each reachable state $q \in Q$, we have

$$\mathsf{L_m}(q) = \{w \in \Sigma^* \mid \delta(q,w) \in M\} \neq \emptyset.$$

States that have a path to a marked state are called *coreachable*. Ramadge and Wonham have shown that given a specification K which is not controllable, it is possible to construct for every plant $\mathcal{A}_\mathcal{P}$ and every specification $\mathcal{A}_\mathcal{E}$ the *supremal controllable sublanguage* of K, denoted $\mathsf{supC}(K)$. This result is of practical interest: Given that the specification language K is uncontrollable, it is possible to compute $\mathsf{supC}(K)$ and to construct a supervisor $\mathcal{A}_\mathcal{S}$ such that $\mathsf{L_m}(\mathcal{A}_\mathcal{S}/\mathcal{A}_\mathcal{P}) = \mathsf{supC}(K)$. This implies that the controlled system is nonblocking, meaning that the constructed supervisor does not prevent the plant from completing a task. This supervisor is a solution to the following problem:

Definition 4 (Supervisory Control Problem) *Given a plant $\mathcal{A}_\mathcal{P}$, a specification language $K \subseteq \mathsf{L_m}(\mathcal{A}_\mathcal{P})$ representing the desired behavior of $\mathcal{A}_\mathcal{P}$ under supervision, find a nonblocking supervisor $\mathcal{A}_\mathcal{S}$ such that $\mathsf{L_m}(\mathcal{A}_\mathcal{S}/\mathcal{A}_\mathcal{P}) \subseteq K$.*

Given a specification automaton $\mathcal{A}_\mathcal{E}$, we can construct the least restrictive solution from the product automaton $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$. The marked language of this least restrictive solution $\mathcal{A}_\mathcal{S}$ is equal to $\mathsf{supC}(K)$. If an automaton $\mathcal{A} = \langle Q, \Sigma, \delta, q_\mathcal{A}^0, M_\mathcal{A}\rangle$ is given that represents the product of the plant and the specification, algorithm 1 can be used to compute this supervisor (Ziller and Schneider, 2003).

Essentially, this algorithm consists of two loops. The inner loop calculates the coreachable states x_C, and the outer loop computes the good states x_G, i.e. states that are not bad states. Since removing bad states could destroy the coreachability property and removing non-coreachable states could result in new bad states, the two loops have to be nested. Based on this algorithm, we will provide an algorithm that calculates a supervisor with the property that every deterministic controller generated from this supervisor is a valid controller, i.e. guarantees that a marked state is reached, irrespectively of the behavior of the plant.

Algorithm 1: Supervisor Synthesis Algorithm

$x_G^0 = Q_\mathcal{A} \setminus \{q \in Q \mid q \text{ is initial bad}\}$;
$j = 0$;
repeat
  $x_C^{(0,j)} = M \cap x_G^j$;
  $i = 0$;
  repeat

$$x_C^{(i+1,j)} = x_G^j \cap \left(x_C^i \cup \left\{ q \in Q \;\middle|\; \begin{array}{l} \exists \sigma \in \mathsf{act}_\mathcal{A}(q)\,. \\ \delta_\mathcal{A}(q,\sigma) \in x_C^{(i,j)} \end{array} \right\}\right)$$

    $i = i + 1$;
  until $x_C^i = x_C^{i-1}$;
  $x_G^{j+1} = x_G^j \cap \left\{ q \in Q \;\middle|\; \begin{array}{l} \forall \sigma \in \mathsf{act}_\mathcal{A}(q) \cap \Sigma_u. \\ \delta_\mathcal{A}(q,\sigma) \in x_C^{(i,j)} \cap x_G^j \end{array} \right\}$
  $j = j + 1$;
until $x_g^j = x_g^{j-1}$;

3 CONTROLLER SYNTHESIS

We have seen by the example given in Figure 1 that the nonblocking property is too weak to guarantee that a marked state is reached under control by a deterministic controller. This is due to the fact that a state is coreachable even if there exists an infinitely long sequence of events that never visits a marked state. We therefore sharpen the coreachability property as follows:

Definition 5 (Forceably Coreachable States) *A state is forceable coreachable, if it is coreachable and*

$$\exists n \in \mathbb{N}. \forall t \in \Sigma^*. \begin{cases} \delta(q,t)\downarrow \wedge |t| \geq n \Rightarrow \exists t' \sqsubseteq t. \delta(q,t') \in Q_m \wedge \\ \delta(q,t)\downarrow \wedge |t| < n \Rightarrow \left(\begin{array}{l} \exists t' \sqsubseteq t. \delta(q,t') \in Q_m \vee \\ \mathsf{act}_\mathcal{A}(q) \neq \emptyset \end{array}\right) \end{cases}$$

Intuitively, a state is forceable coreachable, if there exists a threshold after which a marked state is unavoidable. In contrast to the definition of coreachability that imposes a condition on the future, we demand something about the past: we demand that after a certain amount of steps (referenced by n), a marked state must have been visited. As long as this bound n is not reached, we demand that either the system does not stop or that a marked state has already been reached.

In terms of temporal logics, we demand that *on all paths a marked state must be reached*. In contrast, the nonblocking property only states that *for all states there exists a path where M is reached.* We call an automaton *forceable nonblocking*, if each reachable

state is forceable coreachable. The Controller Synthesis Problem is now given as follows:

Definition 6 (Controller Synthesis Problem) *Given a plant $\mathcal{A}_\mathcal{P}$, a specification language $K \subseteq \mathsf{L_m}(\mathcal{A}_\mathcal{P})$ representing the desired behavior of $\mathcal{A}_\mathcal{P}$ under control, find a nonblocking supervisor $\mathcal{A}_\mathcal{C}$ such that*

- $\mathsf{L_m}(\mathcal{A}_\mathcal{C}/\mathcal{A}_\mathcal{P}) \subseteq K$.
- *$\mathcal{A}_\mathcal{C} \times \mathcal{A}_\mathcal{P}$ is forceable nonblocking.*

Hence, a controller ensures that a marked state is actually reached. It is very easy to derive a deterministic controller from such a solution: in every step, we can simply select a controllable event to ensure that a marked state is actually reached. This is due to the fact, that we demand that all paths leaving a forceable coreachable state sooner or later reach a marked state. Therefore, it is irrelevant which of the active controllable events we select.

Theorem 1 *Given $\mathcal{A}_\mathcal{P} = \langle Q, \Sigma, \delta, q^0_{\mathcal{A}_\mathcal{P}}, M_{\mathcal{A}_\mathcal{P}}\rangle$ and $\mathcal{A}_\mathcal{C} = \langle Q, \Sigma, \delta, q^0_{\mathcal{A}_\mathcal{C}}, M_{\mathcal{A}_\mathcal{C}}\rangle$ such that*

$$\mathsf{L}(\mathcal{A}_\mathcal{C}) \subseteq \mathsf{L}(\mathcal{A}_\mathcal{P}) \wedge \mathsf{L_m}(\mathcal{A}_\mathcal{C}) \subseteq \mathsf{L_m}(\mathcal{A}_\mathcal{P}),$$

then, the following holds: If $\mathcal{A}_\mathcal{C}$ is forceable coreachable then $\mathcal{A}_\mathcal{C} \times \mathcal{A}_\mathcal{P}$ is forceable coreachable.

Proof: Let $(q,p) \in Q_{\mathcal{A}_\mathcal{C}} \times Q_{\mathcal{A}_\mathcal{P}}$ be reachable, such that $\delta_{\mathcal{A}_\mathcal{C}\times\mathcal{A}_\mathcal{P}}((q_0^{\mathcal{A}_\mathcal{C}}, q_0^{\mathcal{A}_\mathcal{P}}), s) = (q,p)$. Then, also $q \in Q_{\mathcal{A}_\mathcal{C}}$ must be reachable in $\mathcal{A}_\mathcal{C}$. Therefore, q is forceable coreachable with a constant n. Now, choose a $t \in \Sigma^*$ such that $\delta_{\mathcal{A}_\mathcal{C}\times\mathcal{A}_\mathcal{P}}((q,p),t)\downarrow$. We distinguish between two cases: First, we assume $|t| \geq n$. Then, there exists a $t' \sqsubseteq t$ such that $\delta(q,t') \in M_{\mathcal{A}_\mathcal{C}}$. Therefore, $st' \in \mathsf{L_m}(\mathcal{A}_\mathcal{C}) \subseteq \mathsf{L_m}((\mathcal{A}_\mathcal{P}))$ holds. Since all automata are deterministic, it follows that $\delta((q,p),t') \in (M_{\mathcal{A}_\mathcal{C}} \times M_{\mathcal{A}_\mathcal{P}})$ holds. In the remaining case, we have $|t| < n$. Then, either there exists a $t' \sqsubseteq t$ that visits a marked state as in the first case or $\mathsf{act}_{\mathcal{A}_\mathcal{C}}(q) \neq \emptyset$. Again, since the language inclusion holds, we have $\mathsf{act}_{\mathcal{A}_\mathcal{C}\times\mathcal{A}_\mathcal{P}}(\delta((q,p),t)) \neq \emptyset$. ■

4 CONTROLLER SYNTHESIS ALGORITHM

In this section, we develop a controller synthesis algorithm based on the supervisor synthesis algorithm of Section 1. In order to guarantee the forceable nonblocking property, we have to adopt the calculation of the coreachable states. In contrast to the coreachability property, which only demands that a marked state is reachable, i.e. that it is possible to directly reach a marked state or to reach a state which is known to be coreachable, a state is forceable coreachable if it is coreachable and all events lead to forceable coreachable states. State and event pairs that guarantee this property are collected in the set $moves$. This implies that all destination states of uncontrollable transitions leaving a state q must be identified as forceable coreachable before we can add any transition from q to moves. Otherwise, q is bad, which is identified in the x_G-loop. This ensures that the controllability property is not violated. To prevent the plant from looping, we forbid adding new moves, if we had already found a move that lead to a marked state. This is done due to the fact that those newly found moves will need a longer path to reach a marked state than the already introduced moves and may therefore introduce loops. We collect the forceable coreachable states in the set x_C by adding those states that have a path to a marked state where this can be guaranteed. Altogether, we thus have developed algorithm 2.

Algorithm 2: Controller Synthesis Algorithm

$j = 0$;
$x_G^0 = Q_\mathcal{A} \setminus \{q \in Q_\mathcal{A} \mid q \text{ is initial bad}\}$;
repeat
  $x_C^{(0,j)} = M \cap x_G^j$;
  $i = 0$;
  $move^{(0,j)} = \{\}$;
  repeat
    $$move^{(i+1,j)} = move^{(i,j)} \cup \left\{ (q,\sigma) \;\middle|\; \begin{array}{c} \delta_\mathcal{A}(q,\sigma) \in x_C^{(i,j)} \\ \wedge \\ \left(\begin{array}{l} \forall \sigma \in \mathsf{act}_\mathcal{A}(q) \cap \Sigma_u. \\ \delta_\mathcal{A}(q,\sigma) \in x_C^{(i,j)} \end{array} \right) \\ \wedge \\ \forall \sigma \in \Sigma.\ (q,\sigma) \notin move^{(i,j)} \end{array} \right\}$$
    $$x_C^{(i+1,j)} = x_G^j \cap \left(x_C^{(i,j)} \cup \left\{ q \;\middle|\; \begin{array}{l} \exists (q,\sigma) \in move^{(i+1,j)} \\ \delta_\mathcal{A}(q,\sigma) \in x_C^{(i,j)} \end{array} \right\} \right)$$
    $i = i + 1$;
  until $x_C^i = x_C^{i-1}$;
  $$x_G^{j+1} = x_G^j \cap \left\{ q \in Q \;\middle|\; \begin{array}{l} \forall \sigma \in \mathsf{act}_\mathcal{A}(q) \cap \Sigma_u. \\ \delta_\mathcal{A}(q,\sigma) \in x_C^{(i,j)} \cap x_G^j \end{array} \right\}$$
  $j = j + 1$
until $x_g^j = x_g^{j-1}$;

The above algorithm may only loop for a finite number of iterations, since there are only finitely many states: In x_C, only finitely many states may be added and from x_G only finitely many states may be removed. Therefore, there exists a k such that $x_G^k = x_G^{k+1}$ finally holds. Additionally, for every i there exists a l such that $x_C^{(i,l)} = x_C^{(i,l+1)}$ and $moves^{(i,l)} = moves^{(i,l+1)}$. For this reason, we use the following notation:$x_G^k = x_G^\infty$ and also

$x_C^{(i,\infty)} := x_C^{(i,k)}$ as well as $move^{(i,\infty)} := move^{(i,k)}$ for every i, and finally $x_C^{(\infty,\infty)} := x_C^{(l,k)}$ and $moves^{(\infty,\infty)} := moves^{(l,k)}$ for the last iteration step.
Note that according to the definition of x_C, it holds that $x_C^{(i,j)} \subseteq x_G^j$ for every i, j and thus also $x_C^{(\infty,\infty)} \subseteq x_G^\infty$ holds. Since $move$ does only contain transitions leading to forceable coreachable states, it thus contains only transitions to good states.

If $q_\mathcal{A}^0 \in x_C^{(\infty,\infty)}$ holds, we define a controller as follows: $\mathcal{A}_\mathcal{C} = \langle Q_\mathcal{A}, \Sigma, \delta_{\mathcal{A}_\mathcal{C}}, q_\mathcal{A}^0, M_\mathcal{A} \rangle$ with

$$\delta_{\mathcal{A}_\mathcal{C}}(q,\sigma) = \begin{cases} \delta_\mathcal{A}(q,\sigma) & \text{, if } (q,\sigma) \in move^{(\infty,\infty)} \\ \uparrow & \text{, else} \end{cases}$$

The following lemma shows that we decrease the distance to a marked state whenever we use an event enabled by the controller:

Lemma 1

$$\forall i > 0 \forall q \in \left(x_C^{(i,\infty)} \setminus x_C^{(i-1,\infty)} \right) \forall \sigma \in \mathsf{act}_{\mathcal{A}_\mathcal{C}}(q) . \\ \delta_{\mathcal{A}_\mathcal{C}}(q,\sigma) \in x_C^{(i-1,\infty)}$$

Proof: Let $q \in Q_\mathcal{A}$ such that $q \in x_C^{(i+1,\infty)} \setminus x_C^{(i,\infty)}$. This implies that there must exist a move $(q,\sigma) \in \left(move^{(i+1,\infty)} \setminus move^{(i,\infty)} \right)$ such that $\delta_\mathcal{A}(q,\sigma) \in x_C^{(i,\infty)}$. But this directly implies that $\delta_\mathcal{A}(q,\sigma) \in x_C^{(i,\infty)}$ for every $(q,\sigma) \in move^{(i+1,\infty)}$. We thus have the statement for those moves added in the $i+1$-th iteration step. Additionally, if follows from the definition of $move$ that there can be no move $(q,\sigma') \in move^{(\infty,\infty)} \setminus move^{(i+1,\infty)}$. Therefore $\delta_{\mathcal{A}_\mathcal{C}}(q,\sigma) \in x_C^{(i,\infty)}$ for every $\sigma \in \mathsf{act}_{\mathcal{A}_\mathcal{C}}(q)$. ■

The above lemma does not apply to marked states (those are contained in $x_C^{(0,\infty)}$). And indeed, without the additional set x_G, this would not be true. The next lemma fixes this deficiency.

Lemma 2

$$\forall q \in \left(M_\mathcal{A} \cap x_C^{(\infty,\infty)} \right) \forall \sigma \in \mathsf{act}_{\mathcal{A}_\mathcal{C}}(q) . \\ \delta_{\mathcal{A}_\mathcal{C}}(q,\sigma) \in x_C^{(\infty,\infty)}$$

Proof: Choose an arbitrary state $q \in M_{\mathcal{A}_\mathcal{C}} \cap x_C^{(\infty,\infty)} \subseteq x_G^\infty$. The proof follows directly for uncontrollable events because of the definition of x_G^∞. Thus, consider a controllable event. According to the definition of $\delta_{\mathcal{A}_\mathcal{C}}$, $\sigma \in \mathsf{act}_{\mathcal{A}_\mathcal{C}}(q) \cap \Sigma_c$ implies that $(q,\sigma) \in move^{(\infty,\infty)}$. According to the definition of $move$, we must have $(q,\sigma) \in move^{(i,\infty)}$ for a suitable i. Therefore, we have $\delta_\mathcal{A}(q,\sigma) \in x_C^{(i-1,\infty)}$, and thus $\delta_{\mathcal{A}_\mathcal{C}}(q,\sigma) \in x_C^{(i-1,\infty)}$. ■

Since the $x_C^{(i,\infty)}, i \in \mathbb{N}$ are monotone in i, the following Lemma follows inductively:

Lemma 3

$$\forall q \in x_C^{(\infty,\infty)} \forall t \in \Sigma^* . \\ \delta_{\mathcal{A}_\mathcal{C}}(q,t) \downarrow \Rightarrow \delta_{\mathcal{A}_\mathcal{C}}(q,t) \in x_C^{(\infty,\infty)}$$

While the above lemma only guarantees that the forceable coreachable states are never left, the next lemma shows that the plant may not stop until a marked state is reached:

Lemma 4

$$\forall i > 0 \forall q \in x_C^{(i,\infty)} \exists \sigma \in \mathsf{act}_{\mathcal{A}_\mathcal{C}}(q) . \\ \delta_{\mathcal{A}_\mathcal{C}}(q,\sigma) \in x_C^{(i-1,\infty)}$$

Proof: According to the definition and the monotony of x_C,

$$q \in x_C^{(i,\infty)} \Leftrightarrow q \in x_G^\infty \wedge \\ \left(\begin{array}{l} q \in M_\mathcal{A} \vee \exists (q,\sigma) \in move^{(i,\infty)}(q). \\ \delta_{\mathcal{A}_\mathcal{C}}(q,\sigma) \in x_C^{(i-1,\infty)} \end{array} \right)$$

If $q \in M_\mathcal{A}$ we are done, otherwise the lemma follows from the definition of $\delta_{\mathcal{A}_\mathcal{C}}$ and the monotony of $move^{(i,\infty)}$ with respect to i. ■

Finally, we now have the following theorem:

Theorem 2 *All $q \in x_C^{(\infty,\infty)}$ are coreachable in $\mathcal{A}_\mathcal{C}$.*

Proof: Take some $q \in x_C^{(\infty,\infty)}$. Then, $q \in x_C^{(i,\infty)}$ for some k. If q is marked, we are done. Otherwise, we can iteratively apply Lemma 4 to generate a string $t \in \Sigma^*$ that reaches a marked state. This is due to the fact that if we apply Lemma 4, then the i-index of the destination state decreases in each step. Therefore, after at most k iteration steps, we have constructed a word t such that $\delta(q,t) \in x_C^{(0,\infty)} = M_\mathcal{A} = M_{\mathcal{A}_\mathcal{C}}$. ■

We will show in the next lemma that also the stronger property of forceable coreachability holds:

Theorem 3 *All $q \in x_C^{(\infty,\infty)}$, are forceable coreachable*

Proof: The coreachability property follows from the last theorem. We will now show the rest of the forceable coreachability property for any $q \in x_C^{(\infty,\infty)}$: Since $q \in x_C^{(\infty,\infty)}$, there exists an $i \in \mathbb{N}$ such that $q \in x_C^{(i,\infty)} \setminus x_C^{(i-1,\infty)}$. If $i = 0$, we are done, because then x_c is marked. Otherwise we show that this i is the threshold that is required for forceable coreachability. Let $t \in \Sigma^*$ be such that $\delta_{\mathcal{A}_\mathcal{C}}(q,t) \downarrow$ holds. Applying Lemma 3 shows that $\delta_{\mathcal{A}_\mathcal{C}}(q,t) \in x_C^{(\infty,\infty)}$ holds.

We distinguish two cases: If $|t| < i$ holds, then either $\delta_{\mathcal{A}_\mathcal{C}}(q,t) \in M_\mathcal{A}$ or $\exists \sigma \in \mathsf{act}_{\mathcal{A}_\mathcal{C}}(q).\delta_{\mathcal{A}_\mathcal{C}}(q,\sigma) \in x_C^{(i-1,\infty)}$ according to Lemma 4. Both cases satisfy the condition of forceable coreachability for the case $|t| < i$.

Now consider any string t with length i and assume that $\delta_{\mathcal{A}_\mathcal{C}}(q,t') \notin M_\mathcal{A}$ for every $t' \sqsubseteq t$. Then, we can iteratively apply Lemma 1 i-times to conclude that $\delta_{\mathcal{A}_\mathcal{C}}(q,t) \in x_C^{(0,\infty)} \subseteq M_\mathcal{A}$ holds. The forceable coreachability property therefore holds for every string with length i and thus also for every string of length greater i. ■

The following theorem gives us the correctness of our algorithm:

Theorem 4 (Correctness of the Algorithm) *If $q_\mathcal{A}^0 \in x_C^{(\infty,\infty)}$, then $\mathcal{A}_\mathcal{C}$ is forceable nonblocking and the generated supervisor $\mathcal{A}_\mathcal{C}$ is a valid solution to the controller synthesis Problem.*

Proof: Since $q_\mathcal{A}^0 \in x_C^{(\infty,\infty)}$ holds, we can conclude from Lemma 3 that every reachable state is contained in $x_C^{(\infty,\infty)}$. The first part of the statement now follows from theorem 3. For the second part, we note that the generated language is necessarily contained in the specification, because of the construction of $\mathcal{A} = \mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E}$. The forceable nonblocking property follows now from theorem 1. The controllability property can be seen as follows: Similar to the original supervisor synthesis algorithm, we can be sure that no initial bad state is reached, because we removed those states from the good states and only good states may be visited. On the other hand, we never remove single uncontrollable transitions due to the definition of $move$. Rather, we remove all states that have an uncontrollable transition to a non-good or non-forceable coreachable state in the condition for the good states. Since $x_C^\infty \subseteq x_G^\infty$ and $q_\mathcal{A} \in x_C^\infty$, we can be sure that only good states are visited. ■

We have shown that the above algorithm is correct. To show also its completeness, i.e. that the algorithm generates a controller, whenever a controller exists, we need the next definition and some additional lemmata. According to the definition of forceable coreachability, for every forceable coreachable state, there exists a constant n after which a marked state is unavoidable. Thus, we can define an ordering on the states by taking the minimal constant n for which the forceable coreachable property holds. Thus, we define for every automaton $\mathcal{A}$:

$$F_\mathcal{A}^n = \left\{ q \in Q_\mathcal{A} \;\middle|\; \begin{array}{l} q \text{ is forceable coreachable} \\ \text{with a minimal constant } n \end{array} \right\}$$

Lemma 5 *For every forceable coreachable automaton $\mathcal{A}$ the following holds:*

$$\forall i > 0 \forall q \in F_\mathcal{A}^i. \left(\begin{array}{l} \forall \sigma \in \mathsf{act}_\mathcal{A}(q).\delta_\mathcal{A}(q,\sigma) \in \bigcup_{j<i} F_\mathcal{A}^j \wedge \\ \exists \sigma \in \mathsf{act}_\mathcal{A}(q).\delta_\mathcal{A}(q,\sigma) \in \bigcup_{j<i} F_\mathcal{A}^j \end{array} \right)$$

Proof: Let q in $F_\mathcal{A}^i$. For the first part, assume that there exists $\sigma \in \mathsf{act}_\mathcal{A}(q)$ such that $\delta_\mathcal{A}(q,\sigma) \notin \bigcup_{j<i} F_\mathcal{A}^j$ holds. Then, we can distinguish two cases: if $q' = \delta_\mathcal{A}(q,\sigma)$ is not forceable coreachable, then there exists an infinite string t with $\delta(q',t) \downarrow$ that avoids all marked states. Accordingly, q can not be forceable coreachable, because otherwise all marked states are avoidable by the infinite string σt. On the other hand, if $\delta_\mathcal{A}(q,\sigma)$ is forceable coreachable, but with a constant greater or equal i, then q can not have a minimal constant i. To prove the second part of the lemma, we first note that q can not be marked, because otherwise $q \in F_\mathcal{A}^0$. Therefore, there exists a successor state, because otherwise q can not be coreachable. However, this successor state must be contained in $\bigcup_{j<i} F_\mathcal{A}^j$ according to the proof of the first part. ■

Lemma 6 *If there exists an automaton $\mathcal{A}_\mathcal{C}$ such that $\mathcal{A} \times \mathcal{A}_\mathcal{C}$ is forceable nonblocking and $\mathcal{A}_\mathcal{C}$ respects the controllability property with respect to $\mathcal{A}$, then $q_\mathcal{A}^0 \in x_C^{(\infty,\infty)}$.*

Proof:

Since $\mathcal{A} \times \mathcal{A}_\mathcal{C}$ is forceable nonblocking, every reachable state is forceable coreachable, therefore contained in some $F_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}^i$. We will show by induction on i:

$$\text{if } (p,q) \in F_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}^i \text{ for some } i, \text{ then } q \in x_C^{(i,\infty)}$$

The above lemma follows then from the fact that the initial state $(q_\mathcal{A}^0, q_{\mathcal{A}_\mathcal{C}}^0)$ must be forceable nonblocking and therefore contained in some $F_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}^i$.

Inductive Base: $i = 0$. Then (p,q) is marked and we are done.

Inductive Step: Let $(p,q) \in F_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}^{i+1}$. Then according to lemma 5 the following holds:

$$\forall \sigma \in \mathsf{act}_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}((p,q)).\delta_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}((p,q),\sigma) \in \bigcup_{j<i} F_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}^j$$

Since the controllability property holds, we have that every uncontrollable event in q is also active in (p,q). Therefore

$$\forall \sigma \in \mathsf{act}_\mathcal{A}(q) \cap \Sigma_u.\delta_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}((p,q),\sigma) \in \bigcup_{j<i} F_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}^j$$

It now follows from the inductive hypothesis and the determinacy of $\mathcal{A}$, that

$$\forall \sigma \in \mathsf{act}_{\mathcal{A}}(q) \cap \Sigma_u . \delta_{\mathcal{A}}(q,\sigma) \in \bigcup_{j<i} x_C^{(i,\infty)} = x_C^{(i,\infty)}$$

Again considering Lemma 5, we obtain:

$$\exists \sigma \in \mathsf{act}_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}((p,q)) . \delta_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}((p,q),\sigma) \in F^i_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}$$

Therefore, using the inductive hypothesis, we obtain

$$\exists \sigma \in \mathsf{act}_{\mathcal{A}\times\mathcal{A}_\mathcal{C}}((p,q)) \subseteq \mathsf{act}_{\mathcal{A}}(q) . \\ \delta_{\mathcal{A}}(q,\sigma) \in \bigcup_{j<i} x_C^{(j,\infty)} = x_C^{(i,\infty)}$$

Now either (q,σ) is added to $move^{i+1}$, or there exists another move (q,σ') that has been already added to $move$. In both cases, we have $q \in x_C^{(i+1,\infty)}$. ■

We are now ready to show completeness of the algorithm:

Theorem 5 (Completeness of the Algorithm) *Given a plant $\mathcal{A}_\mathcal{P}$ and a specification $\mathcal{A}_\mathcal{E}$ where the controller synthesis problem is solvable. Then, $x^0_\mathcal{A} \in x_C^{(\infty,\infty)}$, i.e. the presented algorithm generates a valid controller.*

Proof: Let $\mathcal{A}_\mathcal{C}$ be an automaton that solves the controller synthesis problem. Then, necessarily $\mathsf{L}(\mathcal{A}_\mathcal{C} \times \mathcal{A}_\mathcal{P}) \subseteq \mathsf{L}(\mathcal{A}_\mathcal{E})$ holds as well as $\mathsf{L_m}(\mathcal{A}_\mathcal{C} \times \mathcal{A}_\mathcal{P}) \subseteq \mathsf{L_m}(\mathcal{A}_\mathcal{E})$. $\mathcal{A}_\mathcal{C} \times \mathcal{A}_\mathcal{P}$ is forceable nonblocking. Therefore, $\mathcal{A}_\mathcal{C} \times \mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E} = \mathcal{A}_\mathcal{C} \times \mathcal{A}$ is forceable nonblocking. According to the definition of controller synthesis problem, $\mathcal{A}_\mathcal{C}$ needs to be controllable with respect to $\mathcal{A}_\mathcal{E}$. Therefore, $\mathcal{A}_\mathcal{C}$ must be also controllable with respect to $\mathcal{A}_\mathcal{P} \times \mathcal{A}_\mathcal{E} = \mathcal{A}$. The statement follows now from Lemma 6. ■

5 CONCLUSION

In this paper, we have developed an algorithm for the generation of valid controllers from a supervisory control model as used in the Ramadge-Wonham framework. To this end, we have strengthened the coreachability property in order to guarantee that a marked state is eventually reached, irrespective of the plant's behavior. We have proved the correctness and the completeness of our algorithm. In the future, we plan to implement our Algorithm on top of our toolset Averest (Averest, 2005) to evaluate the runtime behaviour of the algorithm.

REFERENCES

Accellera (2004). PSL/Sugar. http://www.haifa.il.ibm.com/projects/verification/sugar.

Averest (2005). www.averest.org.

Büchi, J. (1960a). On a decision method in restricted second order arithmetic. In Nagel, E., editor, *International Congress on Logic, Methodology and Philosophy of Science*, pages 1–12, Stanford, CA. Stanford University Press.

Büchi, J. (1960b). Weak second order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92.

Dietrich, P., Malik, R., Wonham, W., and Brandin, B. (2002). Implementation considerations in supervisory control. In *B. Caillaud, P. Darondeau, L. Lavagno, and X. Xie, editors, Synthesis and control of discrete event systems*, pages 185–201. Kluwer Academic Publishers.

Emerson, E. (1990). Temporal and modal logic. In *Handbook of Theoretical Computer Science*, volume B, chapter Temporal and Modal Logics, pages 996–1072. Elsevier.

Emerson, E. and Clarke, E. (1982). Using branching-time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266.

Kozen, D. (1983). Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354.

Malik, P. (2003). *From Supervisory Control to Nonblocking Controllers for Discrete Event Systems*. PhD thesis, University of Kaiserslautern, Kaiserslautern, Germany.

Pnueli, A. (1977). The temporal logic of programs. In *Symposium on Foundations of Computer Science (FOCS)*, volume 18, pages 46–57, New York. IEEE Computer Society.

Ramadge, P. and Wonham, W. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230.

Schneider, K. (2003). *Verification of Reactive Systems – Formal Methods and Algorithms*. Texts in Theoretical Computer Science (EATCS Series). Springer.

Thomas, W. (1990). Automata on infinite objects. In *Handbook of Theoretical Computer Science*, volume B, chapter Automata on Infinite Objects, pages 133–191. Elsevier.

Wonham, W. (2001). Notes on control of discrete-event systems. Technical Report ECE 1636F/1637S 2001-02, Department of Electrical and Computer Engineering, University of Toronto.

Ziller, R. and Schneider, K. (2003). A generalized approach to supervisor synthesis. In *Formal Methods and Models for Codesign (MEMOCODE)*, pages 217–226, Mont Saint-Michel, France. IEEE Computer Society.