

A Generalised Approach to Supervisor Synthesis

Roberto Ziller

University of Karlsruhe
Institute for Computer Design and Fault Tolerance
P.O. Box 6980, 76128 Karlsruhe, Germany
email: ziller@informatik.uni-karlsruhe.de

Klaus Schneider

University of Kaiserslautern
Department of Computer Science
P.O. Box 3049, 67653 Kaiserslautern, Germany
email: Klaus.Schneider@informatik.uni-kl.de

Abstract

We present a generalisation of the supervisory control problem proposed by Ramadge and Wonham. The objective of that problem is to synthesise a controller which constrains a system's behaviour according to a given specification, ensuring controllability and coaccessibility. By introducing a new representation of the solution using systems of μ -calculus equations we are able to handle these two conditions separately and thus to exchange the coaccessibility requirement by any μ -calculus expression. Well-known results on the complexity of μ -calculus model checking allow us to easily assess the computational complexity of any generalisation. As an example we solve the synthesis problem under consideration of fairness constraints.

1 Introduction

Many embedded systems used in safety-critical applications consist of reactive real-time controllers, whose design requires automatic tools to improve efficiency and avoid errors made by humans. Modern verification methods [4] allow designers to check a given specification for a controller, but do not support the actual specification process except for providing a simulation trace when an error has been found. Ideally, the specification itself should be generated by a tool that takes the informal requirements of the designer and either outputs a correct specification or rejects them if they cannot be implemented.

A solution that takes the latter approach is offered by the Ramadge-Wonham framework [11, 18]. The main idea is thereby to model the physically possible behaviour of a system and the specification for its desired behaviour by finite-state machines. The method can be used to check whether a controller that ensures the specification can be constructed and, if this is not the case, to compute the largest subset of the specification for which a controller exists. This can then be used in place of the original specification, as long as the resulting behaviour is still acceptable.

The above is an informal description of the *supervisory control problem* formulated and solved by in [11, 19]. Its solution is required to satisfy two conditions. The first one is *controllability*, meaning that the behaviour of the system under supervision must remain within the specification. The second one is *coaccessibility*, meaning that the system must always be able to complete at least one task. These requirements are examples of *safety* and *liveness* properties. However, the description of reactive systems often includes *fairness* properties and therefore requires extended capabilities. In this paper, we generalise the supervisory control problem so that such properties can also be considered.

Related work in this direction uses Büchi and Rabin automata to model infinite behaviour and to derive results on controllability analogous to those of the classical framework [15, 16]. However, finite behaviour is not considered. Besides, translating an informal description of a system into ω -automata is not a trivial task, and the absence of directions to support that translation leaves a gap between theoretical possibilities and application.

An approach that can handle both finite and infinite behaviour is found in [1]. The system and its specification are described using μ -calculus formulas. These are translated into alternating tree automata, and the supervisor synthesis problem is reduced to the *μ -calculus satisfiability problem* [17], which is in turn cast as a search for winning strategies in parity games [7].

In contrast, our approach reduces the supervisory control problem to the *μ -calculus model checking problem* [17]. We depart from specifications written in temporal logics, which have equivalent formulations in the modal μ -calculus defined over systems of fixpoint equations. We illustrate our point of view with several specifications taken from verification literature [13], which we extend to encompass the RW-controllability condition. The result is a generalised framework, of which the basic RW-Model and the extension in [16] are special cases. Moreover, the μ -calculus equations are well-suited for implementation with symbolic methods, which efficiently reduce the state explosion

problem [2]. The chosen approach also enables us to use well-known results on the complexity of μ -calculus model checking [5] to derive the time complexity of any generalisation. Finally, the μ -calculus description can be understood by tools originally intended for verification, which are hereby extended to also handle controller synthesis.

The paper is organised as follows: Section 2 presents the Ramadge-Wonham framework and the supervisory control problem (SCP). Section 3 brings in μ -calculus expressions to present the known solution to SCP in a new formulation. Section 4 presents our generalised supervisory control problem along with several specification examples and its solution. The conclusion summarises the work.

2 The Ramadge-Wonham Framework

The framework parallels continuous systems control theory, in which a system and its controller form a closed loop. There, the feedback signal from the controller influences the behaviour of the system, enforcing a given specification that would not be met by the open-loop behaviour. This foundation on control theory explains some of the terminology adopted within the RW-framework, like the terms *discrete event system* (to designate an event-driven, discrete-space system, in opposition to time-driven, continuous systems) and *plant* (to designate the system to be controlled). It also leads naturally to the basic assumption that the description of the plant encompasses the whole physically possible behaviour of the system to be controlled (including unwanted situations), and that a *specification* is a subset of this behaviour that corresponds to the actions wanted to remain executable under control.

The plant is viewed as a system that generates events. It is also assumed that it has a control input, through which some of the events that could happen in each state can be prevented from occurring. The controller, referred to as *supervisor*, is an external agent that has the ability to observe the events generated by the plant and to influence its behaviour through the control input, as illustrated in Figure 1.

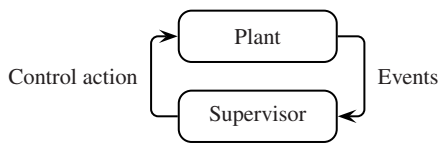


Figure 1. The basic RW-model

Control problems are formulated using language theory and finite automata. A finite automaton is a 5-tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q^0, M \rangle$, where Σ is a set of events, Q is a set of states, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, 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*. Those readers familiar with the

RW-literature will recall that δ is traditionally defined as a partial deterministic *function*. We use a *relation* instead because this simplifies notation later on. We write $\delta(q, \sigma, q')$ to signify that $(q, \sigma, q') \in \delta$. In order to ensure the same functionality, we require the relation to be deterministic, that is, $\delta(q, \sigma, q') \wedge \delta(q, \sigma, q'') \Rightarrow q' = q''$.

In the following, it is convenient to define the set of *active events* $\text{act}_{\mathcal{A}}(q)$ as the subset of events for which there is a transition leaving state q :

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

$$\text{act}_{\mathcal{A}}(q) := \{\sigma \in \Sigma \mid \exists q' \in Q. \delta(q, \sigma, q')\}.$$

When a plant and its supervisor are represented by finite automata $\mathcal{A}_{\mathcal{P}}$ and $\mathcal{A}_{\mathcal{S}}$, respectively, the control action of the latter amounts to running these automata in parallel, according to the following definition:

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

$$\delta_{\mathcal{P} \times \mathcal{S}}((p, q), \sigma, (p', q')) \Leftrightarrow \delta_{\mathcal{P}}(p, \sigma, p') \wedge \delta_{\mathcal{S}}(q, \sigma, q').$$

Note that if a given transition is present in only one of the states p or q , it will *not* be present in state (p, q) , i.e.,

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

The control action of the supervisor enables only the events in $\text{act}_{\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{S}}}$. Hence, in order to forbid the occurrence of event σ when $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{S}}$ is in state (p, q) , it suffices to omit σ in state q .

However, reactive systems may contain events that can not be prevented from occurring, e.g. system failures and sensor or alarm signals. Therefore, the event set Σ is partitioned into the sets of *controllable events* Σ_c (which the supervisor can disable) and *uncontrollable events* Σ_u (whose occurrence cannot be avoided). This places a condition on the existence supervisors: a specification given by an automaton $\mathcal{A}_{\mathcal{E}}$ can be implemented by a supervisor only if, for every state (p, q) of $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$, every event in $\text{act}_{\mathcal{A}_{\mathcal{P}}}(p) \setminus \text{act}_{\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}}((p, q))$ is controllable.

Specifications that do not fulfill this requirement are termed *uncontrollable*, because they allow the plant to reach a state in which uncontrollable events can occur and, at the same time, try to forbid the occurrence of one or more of these events in that state. Formally, this means that the product $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ has one or more *bad states*, which are states (p, q) that fail to satisfy the following condition:

$$\text{act}_{\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}}((p, q)) \supseteq \text{act}_{\mathcal{A}_{\mathcal{P}}}(p) \cap \Sigma_u. \quad (1)$$

Analysing the controllability of a specification further requires some language theory: Every automaton \mathcal{A} has an associated *marked language*, denoted $L_m(\mathcal{A})$, which consists of all event sequences that end up in a marker state, hence representing the tasks the system is able to complete. When δ is extended in the usual way to process strings from Σ^* ,

$$L_m(\mathcal{A}) = \{s \in \Sigma^* : \delta(q^0, s, q) \wedge q \in M\}.$$

Given a specification automaton $\mathcal{A}_\mathcal{E}$, the language $K = L_m(\mathcal{A}_\mathcal{E})$ is *controllable* if and only if $\mathcal{A}_\mathcal{E}$ has no bad states.

The marked language of plant $\mathcal{A}_\mathcal{P}$ under control of supervisor $\mathcal{A}_\mathcal{S}$ is $L_m(\mathcal{A}_\mathcal{P}) \cap L_m(\mathcal{A}_\mathcal{S})$, and is denoted $L_m(\mathcal{A}_\mathcal{S}/\mathcal{A}_\mathcal{P})$. Ramadge and Wonham have shown that, for any plant $\mathcal{A}_\mathcal{P}$ and any specification language $K \subseteq L_m(\mathcal{A}_\mathcal{P})$, there exists the *supremal controllable sublanguage* of K , denoted $\text{supC}(K)$. This result is of practical interest: Given that the specification language K is uncontrollable, it is possible to compute $\text{supC}(K)$ and to construct a supervisor $\mathcal{A}_\mathcal{S}$ such that $L_m(\mathcal{A}_\mathcal{S}/\mathcal{A}_\mathcal{P}) = \text{supC}(K)$. This language can replace the original specification, as long as the resulting behaviour under control is still acceptable.

Another aspect to consider is whether the supervisor always allows the system to make progress towards the completion of some task. This is not the case when the system can (1) reach a state in which no task is finished and no more events can occur (deadlock) or (2) be caught forever within a subset of states, none of which corresponds to a finished task (livelock). A supervisor that avoids these situations is said to be *non-blocking*. A non-blocking automaton is coaccessible, which means that there is at least one path leading from every state to a marker state. Controllability and coaccessibility come together in the following problem:

Definition 3 (Supervisory Control Problem (SCP) [11])

Given a plant $\mathcal{A}_\mathcal{P}$, a specification language $K \subseteq L_m(\mathcal{A}_\mathcal{P})$ representing the desired behaviour of $\mathcal{A}_\mathcal{P}$ under supervision, and a minimally acceptable behaviour $A_{min} \subseteq K$, find a non-blocking supervisor $\mathcal{A}_\mathcal{S}$ such that $A_{min} \subseteq L_m(\mathcal{A}_\mathcal{S}/\mathcal{A}_\mathcal{P}) \subseteq K$.

SCP is solvable if and only if $\text{supC}(K) \supseteq A_{min}$, and $\text{supC}(K)$ is its least restrictive solution [11]. A coaccessible automaton $\mathcal{A}_\mathcal{S}$ whose marked language is equal to $\text{supC}(K)$ can be computed from the automata $\mathcal{A}_\mathcal{P}$ and $\mathcal{A}_\mathcal{E}$, with $\mathcal{A}_\mathcal{E}$ constructed so that $L_m(\mathcal{A}_\mathcal{E}) = K$. Because the resulting automaton is a supervisor, this computation is often referred to as *supervisor synthesis*.

The above is a summary of the most important concepts originally presented in [11, 19, 12]; for a comprehensive description the reader is also referred to [18, 3].

3 Classical Supervisor Synthesis

In this section we associate Kripke structures with the automata used in the RW-framework and define a μ -calculus

over them. Kripke structures are used in Subsection 3.3 to present a new description of the solution for SCP and are also needed to present our main result in Section 4. Because the μ -calculus is not usual in this context, we start with a brief review of basic concepts.

3.1 Fixpoint Calculus

Notations for extremal fixpoints of monotone operators have been introduced by different authors [9]. In particular, Tarski's work [14] has been frequently used in verification and synthesis literature [6, 15]. The following is an adaptation of the results found in these sources to suit our needs.

An operator $f : 2^X \rightarrow 2^X$ on the powerset 2^X is said to be *monotone* if, for any subsets $x_i, x_j \subseteq X$,

$$x_i \subseteq x_j \Rightarrow f(x_i) \subseteq f(x_j). \quad (2)$$

Such an operator has least and greatest fixpoints, which are the solutions of:

$$\begin{aligned} x &\stackrel{\mu}{=} f(x) \quad \text{and} \\ x &\stackrel{\nu}{=} f(x), \end{aligned}$$

where the symbols $\stackrel{\mu}{=}$ and $\stackrel{\nu}{=}$ indicate that we seek for the least and greatest values of x that satisfy these equations. The solutions are denoted $\mu x.f(x)$ and $\nu x.f(x)$, and known to satisfy:

$$\begin{aligned} \mu x.f(x) &= \bigcap \{x \subseteq X : x = f(x)\} \quad \text{and} \\ \nu x.f(x) &= \bigcup \{x \subseteq X : x = f(x)\}. \end{aligned}$$

Given that X is finite and f is monotone, the least fixpoint can be found by an iteration starting with $x_0 = \emptyset$ and computing $x_{i+1} = f(x_i)$ until, for some j , $x_j = x_{j-1}$ holds. The greatest fixpoint can be obtained by the same iteration starting with $x_0 = X$. In this paper, the fixpoint operators will be applied to the state set of a finite Kripke structure.

There is also another version of the modal μ -calculus, defined on equation systems [5, 13] of the form:

$$\begin{cases} x_1 &\stackrel{\sigma_1}{=} \varphi_1 \\ \vdots & \\ x_n &\stackrel{\sigma_n}{=} \varphi_n \end{cases}$$

where $\sigma_i \in \{\mu, \nu\}$ for $i \in \{1, \dots, n\}$. The formulas φ_i may consist of the propositional operators \neg, \wedge , and \vee , and the modal operators EX, AX, EY, and AY. Here, E and A are path quantifiers that specify that the property that follows them in the expression must hold on at least one path (E) or on all paths (A) from some state in a Kripke structure. X and Y are temporal operators that limit the length of the path on which φ has to be fulfilled to the immediate successor or the immediate ancestor states, respectively (see [4, 13] for background on modal operators and temporal logics). As usual, we require that the occurrences of all x_j in every φ_i must occur under an even number of negation symbols.

Any such system of equations can be translated into a single fixpoint expression that uses the operators μ and ν , and vice versa [5, 13]. We shall use the equation system to present our results throughout the paper.

3.2 Automata and Kripke Structures

Definition 4 (Kripke Structure of an Automaton) Given an automaton $\mathcal{A} = \langle \Sigma, Q, \delta, \mathfrak{q}, M \rangle$ representing the product of a plant and a specification, we define its associated Kripke structure $\mathcal{K}_{\mathcal{A}} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ over the Boolean variables $\mathcal{V}_{\mathcal{A}} := \{x_q \mid q \in Q\} \cup \{x_b, x_m, x_u\}$ as follows:

- $\mathcal{S} := Q \times \{0, 1\}$
- $\mathcal{I} := \{(q^0, 0), (q^0, 1)\}$
- $\mathcal{R}((q, 0), (q', 0)) \Leftrightarrow \exists \sigma \in \Sigma_u. \delta(q, \sigma, q')$
- $\mathcal{R}((q, 1), (q', 1)) \Leftrightarrow \exists \sigma \in \Sigma. \delta(q, \sigma, q')$
- $\mathcal{L}((q, 0)) := \{x_q, x_u\} \cup \begin{cases} \{x_b\} & \text{if } q \text{ is bad} \\ \{\} & \text{otherwise} \end{cases}$
- $\mathcal{L}((q, 1)) := \{x_q\} \cup \begin{cases} \{x_m\} & \text{if } q \in M \\ \{\} & \text{if } q \notin M. \end{cases}$

Here, $\Sigma = \Sigma_c \cup \Sigma_u$ (see Section 2), \mathcal{S} is a set of states, \mathcal{I} is the set of initial states, and $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S}$ relates states $(q, 0)$ and $(q', 0)$ exactly when an uncontrollable event leads from q to q' and states $(q, 1)$ and $(q', 1)$ exactly when there is an event (controllable or not) leading from q to q' in \mathcal{A} . This creates a structure with two disconnected substructures, each of which has a copy of the original states in \mathcal{A} . Finally, \mathcal{L} labels each state of \mathcal{S} with a subset of $\mathcal{V}_{\mathcal{A}}$, thereby enabling us to address sets of states through Boolean expressions. Note that the Kripke structure can be constructed from the automaton in time $O(|Q||\Sigma|)$.

As an example, suppose the automaton in Figure 2 represents the product of some plant and specification. The composite numbers of the states have been replaced by singletons for simplicity. States 3 and 5 are assumed to be bad, and the event set is partitioned into $\Sigma_c = \{\alpha, \lambda\}$ and $\Sigma_u = \{\beta, \gamma\}$. The two halves of the associated Kripke

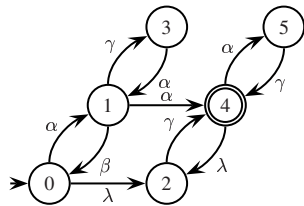


Figure 2. Example automaton

structure are shown in Figure 3. Each state (q, i) is labelled with the variable x_q . The left substructure has transitions only where the automaton has uncontrollable transitions, and its states have in common the label x_u . Additionally,

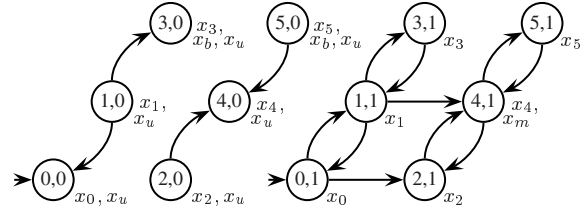


Figure 3. The associated Kripke structure

the states that correspond to bad states in the automaton have the label x_b . The right substructure reflects all transitions from the automaton, with the label x_m identifying the state that corresponds to a marker state. Note that x_u distinguishes the states from the two substructures, and that the left side does not know anything about the marker states, while the right side does not know about the bad states.

The following definitions give syntax and semantics of μ -calculus formulas:

Definition 5 (Syntax of μ -Calculus) Given a set of variables \mathcal{V} , the set of μ -calculus formulas over \mathcal{V} is defined as the least set \mathcal{L}_{μ} that satisfies the following rules:

- $\mathcal{V} \cup \{0, 1\} \subseteq \mathcal{L}_{\mu}$
- $\neg\varphi, \varphi \vee \psi, \varphi \wedge \psi \in \mathcal{L}_{\mu}$, provided that $\varphi, \psi \in \mathcal{L}_{\mu}$
- $\text{EX } \varphi, \text{EY } \varphi \in \mathcal{L}_{\mu}$
- $\kappa_{\pi}(\varphi) \in \mathcal{L}_{\mu}$, provided that $\varphi \in \mathcal{L}_{\mu}$
- $\mu x. \varphi \in \mathcal{L}_{\mu}$, provided that $\varphi \in \mathcal{L}_{\mu}$.

Definition 5 differs from those usually found in the literature in that it includes the formula $\kappa_{\pi}(\varphi)$. This allows us to use any monotone state transformer function $\pi : 2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}}$ in the computations. In particular, we will define a function π to map states of one of the above mentioned substructures to the other.

Definition 6 (Semantics of μ -Calculus) Given a Kripke structure $\mathcal{K} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ over the variables \mathcal{V} , we associate with each formula $\Phi \in \mathcal{L}_{\mu}$ a set of states $\llbracket \Phi \rrbracket_{\mathcal{K}} \subseteq \mathcal{S}$ by the following rules:

- $\llbracket x \rrbracket_{\mathcal{K}} := \{s \in \mathcal{S} \mid x \in \mathcal{L}(s)\}$ for all variables $x \in \mathcal{V}$
- $\llbracket \neg\varphi \rrbracket_{\mathcal{K}} := \mathcal{S} \setminus \llbracket \varphi \rrbracket_{\mathcal{K}}$
- $\llbracket \varphi \wedge \psi \rrbracket_{\mathcal{K}} := \llbracket \varphi \rrbracket_{\mathcal{K}} \cap \llbracket \psi \rrbracket_{\mathcal{K}}$
- $\llbracket \varphi \vee \psi \rrbracket_{\mathcal{K}} := \llbracket \varphi \rrbracket_{\mathcal{K}} \cup \llbracket \psi \rrbracket_{\mathcal{K}}$
- $\llbracket \kappa_{\pi}(\varphi) \rrbracket_{\mathcal{K}} := \pi(\llbracket \varphi \rrbracket_{\mathcal{K}})$ for monotone $\pi : 2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}}$
- $\llbracket \text{EX } \varphi \rrbracket_{\mathcal{K}} := \{s \in \mathcal{S} \mid \exists s' \in \mathcal{S}. \mathcal{R}(s, s') \wedge s' \in \llbracket \varphi \rrbracket_{\mathcal{K}}\}$
- $\llbracket \text{EY } \varphi \rrbracket_{\mathcal{K}} := \{s' \in \mathcal{S} \mid \exists s \in \mathcal{S}. \mathcal{R}(s, s') \wedge s \in \llbracket \varphi \rrbracket_{\mathcal{K}}\}$
- $\llbracket \mu x. \varphi \rrbracket_{\mathcal{K}} := \bigcap \{Q \subseteq \mathcal{S} \mid \llbracket \varphi \rrbracket_{\mathcal{K}_x^Q} \subseteq Q\}$.

The last expression gives the least set of states $Q \subseteq \mathcal{S}$ such that $Q = \llbracket \varphi \rrbracket_{\mathcal{K}_x^Q}$ holds, where \mathcal{K}_x^Q is the Kripke structure where exactly the states Q are labelled with the variable x .

If φ is a monotonic function of x , then $\mu x.\varphi$ is its least fixpoint [14]. We can also define some further macro operators like $AX \varphi := \neg EX \neg\varphi$, $AY \varphi := \neg EY \neg\varphi$, $EG \varphi := \nu y.\varphi \wedge EX y$, $EF \varphi := \mu y.\varphi \vee EX y$, $AG \varphi := \nu y.\varphi \wedge AX y$, $AF \varphi := \mu y.\varphi \vee AX y$, and $\nu x.\varphi(x) := \neg \mu x.\neg\varphi(\neg x)$. The latter can be shown to be the greatest fixpoint of φ .

In order to apply the above definitions to Kripke structures stemming from automata according to Definition 4, we define $\pi(Q) := \{(q, \neg i) \mid (q, i) \in Q\}$, which is easily verified to be monotone (see condition 2). The function κ_π then toggles the variable x_u that identifies the substructure to which a given state pertains, thereby enabling us to switch from one substructure to the other. For simplicity, we write just κ for κ_π from this point on. Further, we have:

- $\llbracket x_b \rrbracket_{\mathcal{K}_A} := \{(q, 0)\}$ for all q that are initially bad
- $\llbracket x_m \rrbracket_{\mathcal{K}_A} := \{(q, 1)\}$ for all $q \in M$
- $\llbracket x_q \rrbracket_{\mathcal{K}_A} := \{q\} \times \{0, 1\}$ for all $q \in Q$
- $\llbracket x_u \rrbracket_{\mathcal{K}_A} := \{(q, 0)\}$ for all $q \in Q$
- $\llbracket \neg x_u \rrbracket_{\mathcal{K}_A} := \{(q, 1)\}$ for all $q \in Q$

In Section 4 we will need to refer to the Kripke structure formed by the states $\llbracket \neg x_u \rrbracket_{\mathcal{K}_A}$ only. The following definition establishes a notation for this purpose:

Definition 7 (Restriction of a Kripke Structure) *Given a Kripke structure $\mathcal{K} = \langle S, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ over the variables \mathcal{V} and a Boolean expression φ over \mathcal{V} , the restriction of \mathcal{K} to the state set $\llbracket \varphi \rrbracket_{\mathcal{K}}$ is $\mathcal{K}|_\varphi := \langle S|_\varphi, \mathcal{I}|_\varphi, \mathcal{R}|_\varphi, \mathcal{L}|_\varphi \rangle$, where:*

- $S|_\varphi := \llbracket \varphi \rrbracket_{\mathcal{K}}$,
- $\mathcal{I}|_\varphi := \mathcal{I} \cap \llbracket \varphi \rrbracket_{\mathcal{K}}$,
- $\mathcal{R}|_\varphi := S|_\varphi \times S|_\varphi \cap \mathcal{R}$,
- $\mathcal{L}|_\varphi(s) := \begin{cases} \mathcal{L}(s) & \text{if } s \in S|_\varphi \\ \text{undefined} & \text{otherwise} \end{cases}$

For example, $\mathcal{K}_A|_{\neg x_u}$ means the structure obtained by discarding the left substructure of \mathcal{K}_A in Figure 3.

Any formula Φ over the variables \mathcal{V}_A also describes a subset of the states of \mathcal{A} according to the following projection, which maps states from the Kripke structure back to the originating automaton:

Definition 8 (Kripke Structure State Projection) *Given an automaton \mathcal{A} , its associated Kripke structure \mathcal{K}_A and a μ -calculus formula Φ over the variables \mathcal{V}_A , we define the projection of $\llbracket \Phi \rrbracket_{\mathcal{K}_A}$ onto the state set Q of \mathcal{A} as:*

$$\llbracket \Phi \rrbracket_{\mathcal{A}} = \{q \in Q \mid (q, 0) \in \llbracket \Phi \rrbracket_{\mathcal{K}_A} \vee (q, 1) \in \llbracket \Phi \rrbracket_{\mathcal{K}_A}\}.$$

With this projection, we can construct an automaton from Φ by restricting the state set of the original automaton \mathcal{A} to $\llbracket \Phi \rrbracket_{\mathcal{A}}$. This completes the toolset to convert an automaton into a Kripke structure, compute a subset of states, and translate the result back to an automaton. For example, the

set of states of $\tau \subseteq \delta$ that are accessible only through states laying in τ is given by:

$$x_{Ac} \stackrel{\mu}{=} \Phi_\tau \wedge (EY x_{Ac} \vee x_{q^0}), \quad (3)$$

where Φ_τ represents the states of τ . Similarly, the set of states of $\tau \subseteq \delta$ that are coaccessible only through states laying in τ is given by $\llbracket x_{Co} \rrbracket_{\mathcal{A}}$, with:

$$x_{Co} \stackrel{\nu}{=} \Phi_\tau \wedge (EX x_{Co} \vee x_m). \quad (4)$$

Of special interest is the *alternation depth* of a fixpoint expression [6, 10] or an equation system. Roughly speaking, this is the nesting depth of alternating μ and ν -operators whose computation depends on each other. Expressions with a single operator have alternation depth 1 and are also called *alternation-free*. The alternation depth of an equation system is the largest number of blocks of formulas seeking for a least or greatest fixpoint in the equation system that depend on each other to compute. The following result [13] will be useful to assess the computational complexity of our solution¹:

Theorem 1 (Complexity of μ -Calculus Model Checking)

For every equation system E of alternation depth l , whose length $|E|$ is given by the sum of the lengths of the right sides of the equations in the system, and every Kripke structure $\mathcal{K} = \langle S, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$, there is an algorithm to compute its solution in time

$$O\left(\left(\frac{|E||S|}{l}\right)^{l-1} |\mathcal{R}||E|\right).$$

Corollary 1 *A system of equations of constant length and alternation depth l written for a Kripke structure associated to an automaton can be solved in time*

$$O(|Q|^l |\Sigma|).$$

Proof. By construction, the Kripke structure from Definition 4 has $|S| = 2|Q|$ and $|\mathcal{R}| \leq 2|Q||\Sigma|$. For constant $|E|$, the result follows immediately.

3.3 Solving SCP

In this subsection, we present the solution for SCP in a new form. Our approach differs from the existing ones in that we replace the textual description of the algorithm by a system of μ -calculus equations. For that purpose, we use the Kripke structure associated to $\mathcal{A}_P \times \mathcal{A}_E$, with \mathcal{A}_E constructed so that $L_m(\mathcal{A}_E) = K$. The solution obtained is amenable to further mathematical manipulation, leading naturally to the generalisation we propose.

¹This holds when the function κ_π can be computed in time $O(\mathcal{R})$, which we assume.

There are two different approaches in the literature to solve SCP, namely the original algorithm [19, 3, 18] and the one given in [8]. The first approach compares the automata $\mathcal{A}_{\mathcal{P}}$ and $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ to find all states that are initially bad and then removes them from $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ along with their transitions. Next, the resulting automaton is made trim, i.e., accessible and coaccessible. Because removing bad states can destroy coaccessibility and removing non-coaccessible states can expose new bad states, the algorithm is restarted with the trimmed automaton replacing the initial automaton $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$, until a fixpoint is reached. Therefore, the search for initial bad states has to be repeated at each iteration. Since this requires information from $\mathcal{A}_{\mathcal{P}}$ which is not present in our Kripke structure, this approach is not well suited as a base for our new formulation.

On the other hand, the algorithm from [8] does not eliminate bad or non-coaccessible states from $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ at each iteration, but collects them and delays elimination until a fixpoint is reached. A state is considered bad if it has an uncontrollable transition leading to a state already classified as bad or non-coaccessible. The trimming operation is substituted by collecting non-coaccessible states and by taking the accessible component of the automaton obtained after the fixpoint has been reached. The initial bad states have to be computed only once at the beginning of the solution process. This corresponds to the computation of the set $\llbracket x_b \rrbracket_{\mathcal{K}_{\mathcal{A}}}$, which is part of the construction of the Kripke structure associated to $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$, and hence we base our solution on the latter approach.

We have derived μ -calculus expressions for the bad and for the non-coaccessible states in [20]. However, the generalisations we aim at now are best described in terms of the states to be preserved, instead of those to be eliminated. We shall therefore collect the coaccessible states (instead of the non-coaccessible ones) into a set denoted x_C and the complement of the bad states, which we call *good states*, into the set x_G .

When collecting states, it is important to choose the appropriate half of the Kripke structure according to the transitions that matter in each case. For the good states, the computation has to be carried out on the substructure identified by x_u , since only the uncontrollable transitions matter. For the coaccessible states, all transitions are relevant, and hence this computation has to be done on the substructure identified by $\neg x_u$. When it comes to consider the good states in the computation of the coaccessible states and vice-versa, we switch from one substructure to the other using the function κ . Hence the expression for x_C can be derived by setting $\Phi_{\tau} = \kappa(x_G)$ in equation 4 to keep only coaccessible states that are good:

$$x_C \stackrel{\mu}{=} \kappa(x_G) \wedge (\text{EX } x_C \vee x_m). \quad (5)$$

An expression for x_G is difficult to obtain directly, so we start with an expression for the bad states and complement it later. We collect the bad states into a set denoted x_B , adding a new state to this set when it has an uncontrollable transition leading into a state that was already found to be bad or non-coaccessible. The initial value for x_B is x_b , and the non-coaccessible states are given by $\neg \kappa(x_C)$, which maps them on the corresponding states on the substructure identified by x_u . The expression for x_B is thus:

$$x_B \stackrel{\mu}{=} \text{EX } (x_B \vee \neg \kappa(x_C)) \vee x_b. \quad (6)$$

The expression for x_G is obtained by complementing equation 6 and substituting x_G for $\neg x_B$. Because the complement brings in unwanted states identified by $\neg x_u$, we explicitly restrict the result to x_u in equation 8. Note also that the complement makes x_G a *greatest* fixpoint. We then have the following system of equations:

$$\begin{cases} x_C \stackrel{\mu}{=} \kappa(x_G) \wedge (\text{EX } x_C \vee x_m) & (7) \\ x_G \stackrel{\nu}{=} x_u \wedge \text{AX } (x_G \wedge \kappa(x_C)) \wedge \neg x_b. & (8) \end{cases}$$

Note that $\llbracket x_C \rrbracket_{\mathcal{K}_{\mathcal{A}}} \subseteq \llbracket x_G \rrbracket_{\mathcal{K}_{\mathcal{A}}}$. Therefore, x_C contains the states that are coaccessible *and* controllable. Hence the solution for SCP is the automaton derived from the accessible states of x_C . The set of accessible states can be computed by setting $\Phi_{\tau} = x_C$ in equation 3 and restricting the result to the states $\neg x_u$:

$$x_{Ac} \stackrel{\mu}{=} \neg x_u \wedge x_C \wedge (\text{EY } x_{Ac} \vee x_{q^0}). \quad (9)$$

The above discussion is a proof of the following:

Proposition 1 (Solution of SCP) *The solution of SCP is given by restricting the automaton $\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}$ to the states $\llbracket x_{Ac} \rrbracket_{\mathcal{A}_{\mathcal{P}} \times \mathcal{A}_{\mathcal{E}}}$, with x_{Ac} given by equation 9.*

The complexity of the overall computation is given by Corollary 1. Since the system of equations 7 and 8 has $l = 2$, we get $O(|Q|^2 |\Sigma|)$, as expected [12, 3]. An example that illustrates the computation of the fixpoints is given in [20].

We can now proceed to our main result, which allows using other equations to describe the states to be collected, thereby generalising the class of problems that can be solved.

4 Generalising Supervisor Synthesis

This section discusses limitations of SCP and introduces the generalised version of the problem, GSCP. Several generalisation possibilities are presented in Subsection 4.2, and the solution for GSCP is presented in Subsection 4.3. Again, Corollary 1 enables us to assess the computational complexity of any solution derived through generalisation.

4.1 Generalised Supervisory Control Problem

The solution for SCP presented in Subsection 3.3 generates an automaton \mathcal{A}_S such that $\mathcal{A}_P \times \mathcal{A}_S$ is controllable and non-blocking, i.e., coaccessible. While controllability is likely to be required in any synthesis result, it is not difficult to imagine problems in which coaccessibility is not adequate to specify a desired behaviour. For example, suppose the system to be controlled is a manufacturing cell designed to produce a number of different parts, and that we want to restrict its behaviour through some specification. Suppose further that we want the resulting supervisor to allow the system to always be able to produce any of those parts. The last condition is a fairness constraint that cannot be expressed within SCP: If we model the production of each part as a finished task using marker states and apply the standard synthesis algorithm, then every non-empty supervisor will allow the system to reach at least one marker state. However, there is no guarantee that all marker states can be reached, and even if they can, it is still another problem to find out if they continue to be reachable all the time. In theory, the minimally acceptable behaviour A_{min} in the formulation of SCP in Definition 3 could be used to reject an incomplete solution, but then the problem is transferred to finding out what A_{min} should be. If this were easy to determine, it could already be regarded as a solution, which we don't have by hypothesis.

This leads naturally to the question whether specifications like the above can be included in the synthesis process by modifying the requirement for coaccessibility, while controllability continues to be computed as before. The states fulfilling the new requirement would be computed on the same substructure used to compute the coaccessible states, and the new requirement must be fulfilled in every state of the right side of the Kripke structure $\mathcal{K}_{\mathcal{A}_P \times \mathcal{A}_S}$, much like the requirement in SCP that all those states are coaccessible.

In the sequel, we bring in a series of fixpoint specifications used in formal verification. Each of them is an expression in temporal logics that can be interpreted as a requirement on the states of $\mathcal{K}_{\mathcal{A}_P \times \mathcal{A}_S}$. We therefore generalise the supervisory control problem by replacing the requirement for a non-blocking supervisor by a temporal logics condition. Our generalised problem is formulated as follows²:

Definition 9 (Generalised Supervisory Control Problem)

(GSCP) Given a plant \mathcal{A}_P and a specification represented by both a language $K \subseteq L_m(\mathcal{A}_P)$ and a temporal logics condition Ω , find a supervisor \mathcal{A}_S for \mathcal{A}_P such that $L_m(\mathcal{A}_S/\mathcal{A}_P) \subseteq K$ and $\mathcal{K}_{\mathcal{A}_P \times \mathcal{A}_S} \upharpoonright_{\neg x_u} \models A \Omega$.

Before presenting the solution for GSCP, we give some examples of what the temporal logics condition Ω could be.

²The symbol \models is read 'satisfies'

4.2 Examples of Fixpoint Specifications

This subsection lists temporal logics expressions used in the design of reactive systems. Such expressions represent states of a Kripke structure that satisfy some temporal property. For example, the expression $EF\varphi$ stands for the states from which there is an infinite path that eventually reaches a state whose labelling satisfies the Boolean expression φ . The states satisfying any temporal logics expression can be computed translating the expression into a system of μ -calculus equations (cf. [13]) whose solution is the desired set of states. Temporal logics expressions are thus used in formal verification to solve the model checking problem, which consists of solving the corresponding equation system and to *check* whether its solution contains the initial states of the Kripke structure.

In contrast, we shall use the solution of the equation system to *restrict* the Kripke structure $\mathcal{K}_{\mathcal{A}_P \times \mathcal{A}_S}$ to the states satisfying the desired property, thereby synthesising a supervisor as formulated in GSCP. The problems that can be solved are not limited to the expressions presented below, but open to any expression written according to a specific need. The following ones are just examples to illustrate the application of the generalised approach.

- $EG\varphi$ holds in every state of a structure that has an infinite path that runs through states that satisfy the property φ . This property can be expressed in the μ -calculus as:

$$\left\{ \begin{array}{l} x \stackrel{\nu}{=} \varphi \wedge EX x \end{array} \right.$$

- $EF\psi$ holds in every state of a structure that has an infinite path that reaches a state that satisfies the property ψ . Its μ -calculus definition is:

$$\left\{ \begin{array}{l} x \stackrel{\nu}{=} EX x \\ z \stackrel{\mu}{=} \psi \wedge x \vee EX z \end{array} \right.$$

x represents the set of states that have an infinite path, and z will therefore satisfy those states that can reach a state where $\varphi \wedge x$ holds.

- If $EF\psi$ is not to be restricted to infinite paths, we write

$$EF^{\text{fin}}\psi, \quad (10)$$

This means that x can be dropped in the equation system above, which is reduced to:

$$\left\{ \begin{array}{l} z \stackrel{\mu}{=} \psi \vee EX z \end{array} \right.$$

- $E[\varphi \cup \psi]$ holds in every state of a structure that has an infinite path that reaches a state that satisfies the property ψ , and up to (but not necessarily including)

this state, will only run through states that satisfy the property φ . Its μ -calculus definition is:

$$\begin{cases} x \stackrel{\nu}{=} \text{EX } x \\ z \stackrel{\mu}{=} \psi \wedge x \vee \varphi \wedge \text{EX } z \end{cases}$$

x represents the set of states that have an infinite path, and z will therefore satisfy those states that can reach a state where $\psi \wedge x$ holds, while only states satisfying φ may be traversed.

- $E[\varphi \text{U}^{\text{fin}} \psi]$ is the version of $E[\varphi \text{U} \psi]$ that also considers finite paths. Dropping the equation for x reduces the above equation system to:

$$\begin{cases} z \stackrel{\mu}{=} \psi \vee \varphi \wedge \text{EX } z \end{cases}$$

- $EFG\varphi$ holds in states that have at least one infinite path where after some point of time φ always holds. This translates to:

$$\begin{cases} x \stackrel{\nu}{=} \varphi \wedge \text{EX } x \\ z \stackrel{\mu}{=} x \vee \text{EX } z \end{cases}$$

As in our first example, x computes the set of states that have an infinite path where φ always holds. z computes the set of states that can reach the set x via a possibly finite path.

- If we want to compute the set of states that can reach a state with some property ψ at least twice, while only states satisfying φ may be traversed, then the following equation system is appropriate:

$$\begin{cases} x \stackrel{\nu}{=} \text{EX } x \\ z_1 \stackrel{\mu}{=} \psi \wedge x \vee \varphi \wedge \text{EX } z_1 \\ y \stackrel{\mu}{=} \psi \wedge \varphi \wedge \text{EX } z_1 \\ z_2 \stackrel{\mu}{=} y \vee \varphi \wedge \text{EX } z_2 \end{cases}$$

- Again, finite paths can be considered by dropping x :

$$\begin{cases} z_1 \stackrel{\mu}{=} \psi \vee \varphi \wedge \text{EX } z_1 \\ y \stackrel{\mu}{=} \psi \wedge \varphi \wedge \text{EX } z_1 \\ z_2 \stackrel{\mu}{=} y \vee \varphi \wedge \text{EX } z_2 \end{cases}$$

- $EGF\varphi$ holds in states that have at least one path where states satisfying φ are traversed infinitely often. This is computed as follows:

$$\begin{cases} x \stackrel{\mu}{=} z \wedge \varphi \vee \text{EX } x \\ z \stackrel{\nu}{=} \text{EX } x \end{cases}$$

- We will now extend the previous condition in that the path should additionally only run through states satisfying β . Hence, we want to compute the states that satisfy the property $E[G\beta \wedge GF\varphi]$:

$$\begin{cases} x \stackrel{\mu}{=} z \wedge \varphi \vee \beta \wedge \text{EX } x \\ z \stackrel{\nu}{=} \beta \wedge \text{EX } x \end{cases}$$

- We extend the previous condition once more considering different sets of states φ_i that should be reached infinitely often. The property to be computed is

$$E \left[G\beta \wedge \bigwedge_{i=1}^n GF\varphi_i \right], \quad (11)$$

which can be expressed in the μ -calculus as follows:

$$\begin{cases} x_1 \stackrel{\mu}{=} z \wedge \varphi_1 \vee \beta \wedge \text{EX } x_1 \\ \vdots \\ x_n \stackrel{\mu}{=} z \wedge \varphi_n \vee \beta \wedge \text{EX } x_n \\ z \stackrel{\nu}{=} \beta \wedge \bigwedge_{i=1}^n \text{EX } x_i \end{cases}$$

- Finally, we consider the property $E \bigwedge_{j=0}^n GF\varphi_j \vee F G\psi_j$,

which is known as the acceptance condition of Rabin automata used in [15, 16].

$$\begin{cases} x_1 \stackrel{\mu}{=} y \wedge \varphi_1 \vee y \wedge \text{EX } x_1 \\ \vdots \\ x_n \stackrel{\mu}{=} y \wedge \varphi_n \vee y \wedge \text{EX } x_n \\ y \stackrel{\nu}{=} \bigwedge_{j=0}^n \text{EX } x_j \vee \psi_j \wedge \text{EX } y \\ z \stackrel{\mu}{=} y \vee \text{EX } z \end{cases}$$

4.3 Solving GSCP

We can now present our main result, which consists of combining the solution for SCP presented in Subsection 3.3 with the conditions represented by equation systems like those in Subsection 4.2. Our generalisation is governed by the following principle, which we assume to have an axiomatic character: *while coaccessibility can be exchanged by any other condition, controllability must always be respected.*

Formally, we mean that equation 7 can be replaced by any set of equations needed to specify some desired property, while equation 8 has to be modified so that the states having the new property take the place of x_C . The systems of equations presented in Subsection 4.2 have the general form:

$$\begin{cases} x_1 \stackrel{\sigma_1}{=} \Phi_1 \\ \vdots \\ x_n \stackrel{\sigma_n}{=} \Phi_n \\ z \stackrel{\sigma_{n+1}}{=} \Psi \end{cases}$$

Here, z is the set of states satisfying condition Ω . According to our argumentation in Subsection 4.1, this condition should be applied only to the substructure of $\mathcal{K}_{\mathcal{A}_P \times \mathcal{A}_E}$ originally used to compute the coaccessible states. This can be achieved by restricting the expressions for Φ_1, \dots, Φ_n and Ψ to $\neg x_u$. Further, looking for a supervisor requires that all paths in the solution contain only good states, which implies restricting the above state sets to $\kappa(x_G)$. Hence the generalised equation system has the following pattern:

$$\begin{cases} x_1 & \stackrel{\sigma_1}{=} & \kappa(x_G) \wedge \neg x_u \wedge \Phi_1 \\ \vdots & & \\ x_n & \stackrel{\sigma_n}{=} & \kappa(x_G) \wedge \neg x_u \wedge \Phi_n \\ z & \stackrel{\sigma_{n+1}}{=} & \kappa(x_G) \wedge \neg x_u \wedge \Psi \\ x_G & \stackrel{\nu}{=} & x_u \wedge \text{AX} (x_G \wedge \kappa(z)) \wedge \neg x_b. \end{cases}$$

As in Subsection 3.3, the accessible states from z can be computed by making $\Phi_\tau = z$ in equation 3 and restricting the result to the states $\neg x_u$:

$$x_{Ac} \stackrel{\mu}{=} \neg x_u \wedge z \wedge (\text{EY } x_{Ac} \vee x_{q^0}). \quad (12)$$

The above discussion is a proof of the following:

Proposition 2 (Solution of GSCP) *The solution of GSCP is given by restricting the automaton $\mathcal{A}_P \times \mathcal{A}_E$ to the states $\llbracket x_{Ac} \rrbracket_{\mathcal{A}_P \times \mathcal{A}_E}$, with x_{Ac} given by equation 12.*

4.4 Generalisation Examples

As a first example, let us derive the solution for SCP from the generalised problem. SCP requires that every state has a path (no matter whether finite or infinite) leading to a marker state. This can be expressed by the temporal logics condition $\text{EF}^{\text{fin}}\varphi$. Substituting $\varphi = x_m$ in expression 10 and $z = x_C$ in the generalised solution pattern for GSCP we get:

$$\begin{cases} x_C & \stackrel{\mu}{=} & \kappa(x_G) \wedge \neg x_u \wedge (x_m \vee \text{EX } x_C) \\ x_G & \stackrel{\nu}{=} & x_u \wedge \text{AX} (x_G \wedge \kappa(x_C)) \wedge \neg x_b. \end{cases}$$

The first equation can be simplified if we note that the computation of x_C starts with the empty set (because it is a least fixpoint) and that the states x_m are all on the substructure identified by $\neg x_u$. Therefore, the computation of the fixpoint never yields states of x_u that would have to be cut out by the conjunction with $\neg x_u$. Hence the latter can be dropped, which transforms the first of the equations above in equation 7. The second equation is already equation 8.

As a second example, we solve the fairness problem described in Subsection 4.1. Let the states that should remain reachable infinitely often be $\varphi_1, \dots, \varphi_n$. Then the temporal logics expression that formalises the problem is

$\text{E}[\bigwedge_{i=1}^n \text{GF}\varphi_i]$, which is expression 11 with $\beta = 1$. According to Subsection 4.3, the generalised equation system is:

$$\begin{cases} x_1 & \stackrel{\mu}{=} & \kappa(x_G) \wedge \neg x_u \wedge (z \wedge \varphi_1 \vee \text{EX } x_1) \\ \vdots & & \\ x_n & \stackrel{\mu}{=} & \kappa(x_G) \wedge \neg x_u \wedge (z \wedge \varphi_n \vee \text{EX } x_n) \\ z & \stackrel{\nu}{=} & \kappa(x_G) \wedge \neg x_u \wedge \bigwedge_{i=1}^n \text{EX } x_i \\ x_G & \stackrel{\nu}{=} & x_u \wedge \text{AX} (x_G \wedge \kappa(z)) \wedge \neg x_b. \end{cases}$$

The supervisor can again be constructed by restricting the automaton $\mathcal{A}_P \times \mathcal{A}_E$ to the set of states $\llbracket z \rrbracket_{\mathcal{A}_P \times \mathcal{A}_E}$. The computational complexity of the solution can also be easily assessed: Since the system of equations above has alternation depth 2, this problem has the same computational complexity as SCP, namely $O(|Q|^2 |\Sigma|)$.

5 Conclusion

The paper presents the Ramadge-Wonham supervisory control problem in a new formulation using a system of μ -calculus equations. In addition to providing a formal description of its solution, this approach naturally separates the representation of the two requirements of the problem, namely controllability and coaccessibility. This allows us to exchange the latter condition by any temporal logics expression, and thereby to extend the advantages of supervisor synthesis to the whole class of μ -calculus model checking problems. The computational complexity of each generalisation can be assessed easily from the alternation depth of the system of equations representing the solution.

References

- [1] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. <http://www.labri.fr/Person/vincent/Research/publications.html>, 2002.
- [2] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic Model-Checking: 10²⁰ States and Beyond. In *Proc. LICS*, 1990.
- [3] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Boston, U.S.A., 1999. ISBN 0-7923-8609-4.
- [4] E. M. Clarke, Jr, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, London, U.K., 1999. ISBN 0-262-03270-8.
- [5] R. Cleaveland, M. Klein, and B. Steffen. Faster Model Checking for the Modal μ -Calculus. In D. P. G.v. Bochmann, editor, *Computer Aided Verification (CAV'92)*, volume 663 of *LNCS*, pages 410–422, Heidelberg, Germany, 1992. Springer-Verlag.
- [6] E. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional μ -calculus. In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 267–278, Washington, D.C., 1986. IEEE Computer Society Press.

- [7] E. A. Emerson, C. S. Jutla, and P. Sistla. On model-checking for fragments of μ -calculus. In C. Courcoubetis, editor, *Computer Aided Verification*, volume 697 of *LNCS*, pages 385–396, Elounda, Greece, 1993. Springer Verlag.
- [8] R. Kumar and V. Garg. *Modeling and Control of Logical Discrete Event Systems*. Kluwer Academic Publishers, 1995. ISBN 0-7923-9538-7.
- [9] J.-L. Lassez, V. Nguyen, and E. Sonenberg. Fixed point theorems and semantics: a folk tale. *Information Processing Letters*, 14(3):112–116, May 1982.
- [10] D. Niwinski. On fixed point clones. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 464–473. L. Kott, Ed., vol 226 of *LNCS*, Springer-Verlag, 1986.
- [11] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. of Control and Optimization*, 25(1):206–230, 1987.
- [12] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.
- [13] K. Schneider. *Verification of Reactive Systems – Algorithms and Formal Methods*. EATCS Texts. Springer, 2003.
- [14] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific J. Math.*, 5(2):285–309, 1955.
- [15] J. G. Thistle and W. M. Wonham. Control of infinite behavior of finite automata. *SIAM J. of Control and Optimization*, 32(4):1075–1097, 1994.
- [16] J. G. Thistle and W. M. Wonham. Supervision of infinite behavior of discrete–event systems. *SIAM J. of Control and Optimization*, 32(4):1098–1113, 1994.
- [17] Th. Wilke. Alternating Tree Automata, Parity Games, and Modal μ -Calculus. *Bull. Soc. Math. Belg.*, 8(2), May 2001.
- [18] W. M. Wonham. Notes on control of discrete-event systems. Technical report, Dept. of Electrical and Computer Engineering, University of Toronto, Jul. 2002. Available at <http://www.control.utoronto.ca/DES>.
- [19] W. M. Wonham and P. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM J. Control and Optimization*, 25(3):637–659, May 1987.
- [20] R. M. Ziller and K. Schneider. A μ -Calculus Approach to Supervisor Synthesis. In *GIITG/GMM–Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 132–143, 2003.