A Unified Model Checking Framework for the Supervisor Synthesis Problem

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

Abstract. The supervisor synthesis problem asks whether one can restrict the behavior of a reactive system such that it satisfies a given specification. As it is more general, this problem is harder than the verification problem. Several approaches based on different logics have been developed to tackle the supervisor synthesis problem. In this paper, we show that the most prominent logics, namely alternating time $\mu$-calculus and extensions of the Ramadge-Wonham framework, can be reduced to the model checking problem of the propositional $\mu$-calculus. As a result, our algorithms may be used as a frontend on top of existing model-checking tools like Averest.

1 Introduction

Applications in safety critical areas require the verification of the developed systems. Additionally, these systems are often reactive systems that must react sufficiently fast to the stimuli of the environment. For this reason, these systems are often real-time systems, and hence, their temporal behavior is of essential importance for their correctness.

In the past two decades, many verification methods for the temporal behavior of reactive systems have been elaborated [16], 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 $\varphi$. There are a lot of formalisms, in particular, the $\mu$-calculus [10], $\omega$-automata [17], as well as temporal [12,9,8] and predicate logics [7,6] to formulate the specification $\varphi$ [16]. Moreover, industrial interest already lead to standardization efforts on specification logics [5,4,13].

Besides the verification problem, where the entire system $\mathcal{K}$ and its specification must be already available, one can also consider the controller/supervisor synthesis problem. The task is here to check whether there is a reactive system $\mathcal{S}$ such that $\mathcal{K} \parallel \mathcal{S}$ satisfies $\varphi$. Obviously, this problem is more general than the verification problem. Efficient solutions for this problem could be used to guide design decisions in the development of reactive systems.

The controller synthesis problem is not new; already established variants are the supervisory control problem [14], module checking [11], and alternating time $\mu$-calculus model checking [1,2,3]. In all the mentioned formalisms, the general question is to check whether the behavior of the system can be restricted such that a given specification is met. While in [11] and [3], it was discussed whether supervisor synthesis problems can
be solved by using ordinary model-checkers, [1] and [14] solved the supervisory control problem using new specification languages.

In supervisory control, both the open system and the specification are usually given in form of finite automata, and the specifications are usually safety and nonblocking properties. Recently, this formalism has been extended in [18] to allow more general specifications in form of the $\mu$-calculus. This is achieved by translating supervisory control problems to certain model checking problems on a Kripke structure. The temporal logic that is used is the propositional $\mu$-calculus, where a slight extension by so-called monotone state transformers was used.

For concurrent games and model checking of alternating time $\mu$-calculus formulas, the system is modeled by an infinite game which is played by two players (representing the system and the environment). The specifications are given by an alternating time $\mu$-calculus formulas. Alternating time $\mu$-calculus (ATL) extends the ordinary $\mu$-calculus with new modal operators that differentiate between the players. Using alternating time $\mu$-calculus, it is therefore easier to formulate statements like ‘The controller can enforce the visit of certain states irrespectively of how the environment behaves.’ Such typical properties can not be easily formulated using the ordinary $\mu$-calculus. Therefore, alternating time $\mu$-calculus and its corresponding alternating time temporal logics ATL and ATL* have been proposed as better alternatives to traditional temporal logics like LTL or CTL* that are used for verification.

In this paper, however, we will prove that alternating time $\mu$-calculus problems on concurrent games can be translated to propositional $\mu$-calculus model checking problems on Kripke structures. This has the advantage that the efficient machinery of already existing model checking tools can be used to answer supervisory control problems. Therefore, alternating time $\mu$-calculus may be seen as syntactic sugar that enriches propositional $\mu$-calculus without enhancing its expressiveness. Hence, we do not argue that alternating time $\mu$-calculus is unnecessary, but want to emphasize that already available $\mu$-calculus model checkers can be used to solve problems that were formulated with alternating time $\mu$-calculus.

Furthermore, using the transformation to $\mu$-calculus model checking, we present solutions of the supervisory control problems as alternatives to [18]. The advantage of our solution is that we do not have to introduce new monotone state transformers. Hence, we present a reduction from the classical supervisory control problem to pure $\mu$-calculus model-checking. Again, this reduction enables us to use already available $\mu$-calculus model checkers to solve supervisory control problems.

The outline of the paper is as follows: We start by introducing Kripke structures, the propositional $\mu$-calculus and alternating time $\mu$-calculus. In Section 3, we present our reduction from alternating time $\mu$-calculus model checking to the propositional $\mu$-calculus model checking. Section 4 describes the reduction from supervisory control problems to propositional $\mu$-calculus model checking. Finally, we summarize the added values of the paper and sketch our ideas for future work.

---

1 For a more detailed view of what can be formulated using ATL, we refer to [1].
2 Formal Background

2.1 The Propositional \( \mu \)-Calculus

The propositional \( \mu \)-calculus is a well-known specification logic whose model checking algorithms form the heart of state-of-the-art verification tools [16]. We briefly present its syntax and semantics in this section.

Definition 1 (Syntax of \( \mu \)-Calculus). Given a set of variables \( \mathcal{V} \), the set of \( \mu \)-calculus formulas over \( \mathcal{V} \) is defined as the least set \( \mathcal{L}_\mu \) that satisfies the following rules:

\[
\begin{align*}
\mathcal{V} \cup \{0, 1\} & \subseteq \mathcal{L}_\mu, \\
\neg \varphi, \varphi \land \psi, \varphi \lor \psi & \in \mathcal{L}_\mu, \text{ provided that } \varphi, \psi \in \mathcal{L}_\mu, \\
\varphi^{\diamond}, \Box \varphi & \in \mathcal{L}_\mu, \text{ provided that } \varphi \in \mathcal{L}_\mu, \\
\mu x. \varphi, \nu x. \varphi & \in \mathcal{L}_\mu, \text{ provided that } \varphi \in \mathcal{L}_\mu \text{ and } x \in \mathcal{V}.
\end{align*}
\]

The semantics is defined on labeled transition systems which are often called Kripke structures due to the relationship to modal logic [16].

Definition 2 (Kripke Structures). A Kripke structure \( \mathcal{K} = (I, S, R, \mathcal{L}) \) for a finite set of variables \( \mathcal{V} \) is given by a finite set of states \( S \), a set of initial states \( I \subseteq S \), a transition relation \( R \subseteq S \times S \), and a label function \( \mathcal{L} : S \rightarrow 2^\mathcal{V} \) that maps each state to a set of variables.

The semantics of \( \mathcal{L}_\mu \) is given in the next definition where formulas are mapped to sets of states of a Kripke structure:

Definition 3 (Semantics of \( \mu \)-Calculus). Given a Kripke structure \( \mathcal{K} = (I, S, R, \mathcal{L}) \) over the variables \( \mathcal{V} \), we associate with each formula \( \varphi \in \mathcal{L}_\mu \) a set of states \( [\varphi]_\mathcal{K} \subseteq S \) by the following rules:

\[
\begin{align*}
[0]_\mathcal{K} & := \{\}\ , \\
[1]_\mathcal{K} & := S \\
[x]_\mathcal{K} & := \{s \in S \mid x \in \mathcal{L}(s)\} \text{ for all variables } x \in \mathcal{V} \\
[\neg \varphi]_\mathcal{K} & := S \setminus [\varphi]_\mathcal{K} \\
[\varphi \lor \psi]_\mathcal{K} & := [\varphi]_\mathcal{K} \cup [\psi]_\mathcal{K} \\
[\varphi \land \psi]_\mathcal{K} & := [\varphi]_\mathcal{K} \cap [\psi]_\mathcal{K} \\
[\varphi^{\diamond}]_\mathcal{K} & := \text{pre}_R^\mathcal{K} ([\varphi]_\mathcal{K}) \\
[\Box \varphi]_\mathcal{K} & := \text{pre}_R^\mathcal{K} ([\varphi]_\mathcal{K}) \\
[\mu x. \varphi]_\mathcal{K} & := \bigcap \{Q \subseteq S \mid [\varphi]_\mathcal{K}^Q \subseteq Q\}, \text{ where } \mathcal{K}^Q_x \text{ is obtained by changing the label function } \mathcal{L} \text{ of } \mathcal{K} \text{ such that exactly the states in } Q \text{ are labeled with } x. \\
[\nu x. \varphi]_\mathcal{K} & := \bigcup \{Q \subseteq S \mid [\varphi]_\mathcal{K}^Q \subseteq Q\}, \text{ where } \mathcal{K}^Q_x \text{ is defined as above.}
\end{align*}
\]

The modal operators \( \Diamond \) and \( \Box \) are simply defined by existential and universal predecessor operators [16] which are in turn defined for every state set \( Q \) as follows:

\[
\begin{align*}
\text{pre}_R^\mathcal{K}(Q) & = \{s \in S \mid \exists s' \in S. \ R(s, s') \land s' \in Q\} \\
\text{pre}_R^\mathcal{K}(Q) & = \{s \in S \mid \forall s' \in S. \ R(s, s') \rightarrow s' \in Q\}
\end{align*}
\]
2.2 Alternating Time $\mu$-Calculus

A concurrent game structure models the behavior of a reactive system by two players $A$ and $B$ (the environment and the system) of a game. In every step of the game, both players choose independently of each other an action, and the resulting pair of actions uniquely determines the next state of the game. This intuitive idea is formalized in the following definition of a concurrent game structure:\(^2\):

**Definition 4 (Concurrent Game Structure).** A concurrent game structure $G = (I, S, \delta, \Gamma_A, \Gamma_B, L)$ for a set of variables $V$ and a set of actions $A$ is given by a finite set of states $S$, a set of initial states $I \subseteq S$, a partial transition relation $\delta \subseteq S \times A \times A \times S$, and label functions $\Gamma_A, \Gamma_B : S \rightarrow 2^A$, and $L : S \rightarrow 2^V$.

Intuitively, a concurrent game structure is a finite state transition system whose transitions are labeled with pairs of actions $(a, b)$ that have been chosen by the players $A$ and $B$ in the corresponding source state. Note that there is no ordering in the actions, i.e., the two players $A$ and $B$ choose their actions independently of each other.

![Diagram of a concurrent game structure](image)

**Fig. 1.** The train example

As an example for a concurrent game, consider Figure 1. This concurrent game models a protocol for a train entering a railroad station. If the train is outside of the station, the train selects action `request` to indicate its wish to enter the station. If the request is granted by the station controller, the train is allowed to enter the station. Otherwise, it enters the `train_wait` state, where it remains until a `grant` action of the station is seen, or the train drives past the station which is signaled by selecting action `abort`\(^3\).

Obviously, concurrent game structures are extensions of Kripke structures. Thus, we can directly use Definition 3 to evaluate $\mu$-calculus formulas on concurrent game

---

\(^2\) The original definition given in [1,2,3] allowed more than two players, which is however not necessary for our purpose.

\(^3\) To save space, we let * stand for a don’t care, in the above example this means that it is unimportant, which action the controller chooses when the train chooses `abort`
structures. However, the additional labels on the transitions induce further modal operators [1,2,3] that are particularly useful for reasoning about games and their reactive behavior:

**Definition 5 (Syntax of Alternating Time μ-Calculus).** Given a set of variables $\mathcal{V}$, the formulas of the alternating time $\mu$-calculus $\mathcal{L}_{AT}$ is the least set that satisfies the following points:

- $\mathcal{V} \cup \{0, 1\} \subseteq \mathcal{L}_{AT}$
- $\neg \varphi, \varphi \land \psi, \varphi \lor \psi \in \mathcal{L}_{AT}$, provided that $\varphi, \psi \in \mathcal{L}_{AT}$
- $\diamond \varphi, \square \varphi \in \mathcal{L}_{AT}$, provided that $\varphi \in \mathcal{L}_{AT}$
- $\exists x. \varphi, \forall x. \varphi \in \mathcal{L}_{AT}$, provided that $\varphi \in \mathcal{L}_{AT}$

The semantics of $\mathcal{L}_{AT}$ is defined on concurrent game structures in the same way as the semantics of $\mathcal{L}_\mu$ is defined on Kripke structures. The interesting new point is the definition of the new modal operators $\boxtimes_A$ and $\boxtimes_B$:

**Definition 6 (Semantics of $\mathcal{L}_{AT}$).** Given a concurrent game structure $\mathcal{G} = (\mathcal{I}, \mathcal{S}, \delta, \Gamma_A, \Gamma_B, \mathcal{E})$ for a set of variables $\mathcal{V}$ and a set of actions $\mathcal{A}$, we associate with each formula $\Phi \in \mathcal{L}_{AT}$ a set of states $[\Phi]_G \subseteq \mathcal{S}$ by the following rules:

- $[[0]]_G := \{\}$
- $[[1]]_G := \mathcal{S}$
- $[[x]]_G := \{s \in \mathcal{S} | x \in \mathcal{L}(s)\}$ for all variables $x \in \mathcal{V}$
- $[[\neg \varphi]]_G := \mathcal{S} \setminus [[\varphi]]_G$
- $[[\varphi \lor \psi]]_G := [[\varphi]]_G \cup [[\psi]]_G$
- $[[\varphi \land \psi]]_G := [[\varphi]]_G \cap [[\psi]]_G$
- $[[\diamond \varphi]]_G := \{s \in \mathcal{S} | \exists a \in \Gamma_A(s) \exists b \in \Gamma_B(s) \exists s' \in \mathcal{S} \delta(s, a, b, s') \land s' \in [[\varphi]]_G\}$
- $[[\square \varphi]]_G := \{s \in \mathcal{S} | \forall a \in \Gamma_A(s) \forall b \in \Gamma_B(s) \forall s' \in \mathcal{S} \delta(s, a, b, s') \rightarrow s' \in [[\varphi]]_G\}$
- $[[\exists x. \varphi]]_G := \{s \in \mathcal{S} | \exists a \in \Gamma_A(s) \forall b \in \Gamma_B(s) \forall s' \in \mathcal{S} \delta(s, a, b, s') \rightarrow s' \in [[\varphi]]_G\}$
- $[[\forall x. \varphi]]_G := \{s \in \mathcal{S} | \exists b \in \Gamma_B(s) \forall a \in \Gamma_A(s) \forall s' \in \mathcal{S} \delta(s, a, b, s') \rightarrow s' \in [[\varphi]]_G\}$

Note that the four modal operators cover all possible quantifier prefixes. As $\neg \square \varphi$ is equivalent to $\diamond \neg \varphi$, we can even eliminate either $\square$ or $\diamond$. Note, however, that there is no such duality between $\boxtimes_A$ and $\boxtimes_B$. Their duals correspond to two further operators that we do not explicitly mention.
The essential novelty of alternating time μ-calculus are the modal operators \( \Box_A \) and \( \Diamond_B \) that are particularly useful to specify games: Intuitively, a state \( s \) satisfies \( \Box_A \varphi \), if player \( A \) can choose an action in state \( s \) such that a state \( s' \) is reached where \( \varphi \) holds (no matter which action player \( B \) chooses). Analogously, \( [B \varphi]_G \) is the set of states where player \( B \) can enforce the game to a state \( s' \) where \( \varphi \) holds independent of \( A \)’s choice.

### 3 Translating \( \mathcal{L}_{AT} \) to \( \mathcal{L}_\mu \)

In this section, we show that \( \mathcal{L}_{AT} \) model checking and \( \mathcal{L}_\mu \) model checking are equivalent problems that can be easily reduced to each other. Thus, we can solve essentially the same problems with both formalisms. As \( \mathcal{L}_{AT} \) and concurrent game structures are extensions of \( \mathcal{L}_\mu \) and Kripke structures, respectively, one reduction is immediately clear. To show the other reduction, we formally describe a translation from concurrent game structures to associated Kripke structures, and from \( \mathcal{L}_{AT} \) formulas \( \varphi \) to corresponding \( \mathcal{L}_\mu \) formulas \( \mathcal{AT}_\mu (\varphi) \).

**Definition 7 (Associated Kripke Structure of a Concurrent Game Structure).** Given a concurrent game structure \( G = (I, S, \delta, \Gamma_A, \Gamma_B, \mathcal{L}) \) for a set of variables \( V \) and a set of actions \( \mathcal{A} \), we define the associated Kripke structure \( K_G = (I', S', \mathcal{R}, \mathcal{L}') \) over the variables \( V \cup \{x_A, x_B, x_N\} \) as follows:

- \( I' = I \times \{\text{nop}\} \times \{0\} \)
- \( S' = S \times (A \cup \{\text{nop}\}) \times \{0, 1\} \)
- \( \mathcal{R}((s, \text{nop}, 0), (s, \alpha, \iota)) : \Leftrightarrow \exists s' \in S. \)
  \[ \begin{align*}
  \iota &= 1 \land \alpha \in \Gamma_A(s) \land \exists b \in \Gamma_B(s). \delta(s, \alpha, b, s') \\
  \iota &= 0 \land \alpha \in \Gamma_B(s) \land \exists a \in \Gamma_A(s). \delta(s, a, \alpha, s')
  \end{align*} \]
- \( \mathcal{R}((s, \alpha, \iota), (s', \text{nop}, 0)) : \Leftrightarrow \)
  \[ \begin{align*}
  \iota &= 1 \land \alpha \in \Gamma_B(s) \land \exists b \in \Gamma_A(s). \delta(s, \alpha, b, s') \\
  \iota &= 0 \land \alpha \in \Gamma_A(s) \land \exists a \in \Gamma_B(s). \delta(s, a, \alpha, s')
  \end{align*} \]
- \( \mathcal{L}'((s, \alpha, \iota)) : = \mathcal{L}(s) \cup \)
  \[ \begin{align*}
  \{x_N\} & \text{ if } \alpha = \text{nop} \land \iota = 0 \\
  \{x_A\} & \text{ if } \alpha \in \Gamma_A(s) \land \iota = 1 \\
  \{x_B\} & \text{ if } \alpha \in \Gamma_B(s) \land \iota = 0
  \end{align*} \]

The idea behind the construction of the associated Kripke structure is an unrolling of the concurrent game structure. Each state \((s, \text{nop}, 0) \in S'\) directly corresponds to a state \( s \in S \) of the game structure. Every transition from a state \( s \in S \) with \( s' \in \delta(s, a, b) \) is split into halves: one that leads from the corresponding state \((s, \text{nop}, 0) \to (s, a, 1)\) which means that player \( A \) made the first move. Moreover, there is a transition from \((s, \text{nop}, 0) \to (s, b, 0)\) which means that player \( B \) made the first move. From the states \((s, a, 1)\) and \((s, b, 0)\) there is a further transition to the state \((s', \text{nop}, 0)\) to complete the transition of \( G \) in \( K_G \).
Hence, the construction of $K_G$ essentially introduces ‘intermediate states’ to convert the concurrent game structure into a turn based game structure, where the order in which the players make their moves is irrelevant. States of the form $(s, \text{nop}, 0)$ are called choice states. As each choice state $(s, \text{nop}, 0)$ directly corresponds to the state $s$ of the game structure, it is possible to map sets of states of the game structure to corresponding states of the Kripke structure. The converse is also possible for sets of choice states of the Kripke structure. As an example for the translation of a concurrent game to its associated Kripke structure, consider the Kripke Structure that is given in Figure 3: this is the corresponding Kripke structure to our train game.

For the following constructions, we make implicitly use of certain invariants of the above construction. These invariants are listed in the next lemma.

**Lemma 1 (Invariants for the Construction of $K_G$).** Given a concurrent game structure $G = (I, S, \delta, \Gamma_A, \Gamma_B, L)$ for a set of variables $V$ and a set of actions $A$, and its associated Kripke structure $K_G$ over the variables $V \cup \{x_A, x_B, x_N\}$ as defined in Definition 7. Then, the following holds for every transition $R((s, \alpha, \iota), (s', \alpha', \iota'))$:

- $\alpha = \text{nop}$ implies $\iota = 0$
- $\alpha' = \text{nop}$ implies $\iota' = 0$
- $\alpha \in A$ holds iff $\alpha' = \text{nop}$
- $\alpha = \text{nop}$ holds iff $\alpha' \in A$
- $\alpha \in A$ and $\iota = 1$ imply that $\alpha \in \Gamma_A(s)$
- $\alpha \in A$ and $\iota = 0$ imply that $\alpha \in \Gamma_B(s)$

In particular, the above lemma implies that the transition system of $K_G$ is a bipartite graph that consists of two kinds of states: those with actions $\alpha \in A$ and those with action $\alpha = \text{nop}$. Moreover, the action $\text{nop}$ determines the boolean flag $\iota = 0$. For actions other than $\text{nop}$, the boolean flag $\iota$ stores the information which one of the players was responsible for the first half of the move.

The above definition captures the translation from game structures to Kripke structures. With the following definition, we proceed with the translation of $L_{AT}$ to $L_m$:
Definition 8 (Translating $\mathcal{L}_{AT}$ to $\mathcal{L}_{\mu}$). For every formula $\varphi \in \mathcal{L}_{AT}$ over the variables $\mathcal{V}$, we define a formula $\AT_{\mu}(\varphi) \in \mathcal{L}_{\mu}$ over the variables $\mathcal{V} \cup \{x_A, x_B, x_N\}$ inductively as follows:

- $\AT_{\mu}(0) := 0$
- $\AT_{\mu}(1) := x_N$
- $\AT_{\mu}(x) := x$ for variables $x \in \mathcal{V}$
- $\AT_{\mu}(\neg \varphi) := x_N \land \neg \AT_{\mu}(\varphi)$
- $\AT_{\mu}(\varphi \land \psi) := \AT_{\mu}(\varphi) \land \AT_{\mu}(\psi)$
- $\AT_{\mu}(\varphi \lor \psi) := \AT_{\mu}(\varphi) \lor \AT_{\mu}(\psi)$
- $\AT_{\mu}(\Box \varphi) := \Box \AT_{\mu}(\varphi)$
- $\AT_{\mu}(\Box\varphi) := \Box(\varphi \land \Box \AT_{\mu}(\varphi))$
- $\AT_{\mu}(\Box B \varphi) := \Box(\varphi \land \Box \AT_{\mu}(\varphi))$
- $\AT_{\mu}(\mu x. \varphi) := \mu x. \AT_{\mu}(\varphi)$
- $\AT_{\mu}(\nu x. \varphi) := \nu x. \AT_{\mu}(\varphi)$

Note that according to the definition of the associated Kripke structure $\mathcal{K}_G$, we have $\llbracket x_N \rrbracket_{\mathcal{K}_G} = S \times \{\text{nop}\} \times \{0\}$, i.e., the choice states. Moreover, $\llbracket x_N \rrbracket_{\mathcal{K}_G} \cap \llbracket x_A \lor x_B \rrbracket_{\mathcal{K}_G} = \emptyset$, i.e., each state where either $x_A$ or $x_B$ holds is not a choice state. For this reason, we could also perform the construction without the variable $x_N$ and use $\neg(x_A \lor x_B)$ instead of $x_N$ in the above translation.

Lemma 2. Given a concurrent game structure $G = (\mathcal{I}, S, \delta, \Gamma_A, \Gamma_B, \mathcal{L})$ for a set of variables $\mathcal{V}$ and a set of actions $\mathcal{A}$, and its associated Kripke structure $\mathcal{K}_G$ over the variables $\mathcal{V} \cup \{x_A, x_B, x_N\}$ as given in Definition 7. Then, the following holds:

- $\llbracket x \rrbracket_{\mathcal{K}_G} = \{(s, \alpha, \ell) \mid s \in \llbracket x \rrbracket_G \land a = \text{nop} \land \ell = 0\}$
- $\llbracket x_N \rrbracket_{\mathcal{K}_G} = \{(s, \alpha, \ell) \mid s \in S \land a = \text{nop} \land \ell = 0\}$
- $\llbracket x_A \rrbracket_{\mathcal{K}_G} = \{(s, \alpha, \ell) \mid s \in S \land a \in \Gamma_A(s) \land \ell = 1\}$
- $\llbracket x_B \rrbracket_{\mathcal{K}_G} = \{(s, b, \ell) \mid s \in S \land b \in \Gamma_B(s) \land \ell = 0\}$
- $\pre_{\mathcal{K}_G}(Q \times \{\text{nop}\} \times \{0\}) = \{(s, a, 1) \in S' \mid a \in \Gamma_A(s) \land \exists b \in \Gamma_B(s) \exists s' \in S. \delta(s, a, b, s') \land s' \in Q\} \cup \{(s, b, 0) \in S' \mid b \in \Gamma_B(s) \land \exists a \in \Gamma_A(s) \exists s' \in S. \delta(s, a, b, s') \land s' \in Q\}$
- $\pre_{\mathcal{K}_G}(Q \times \{\text{nop}\} \times \{0\}) = \{(s, a, 1) \in S' \mid a \in \Gamma_A(s) \land \forall b \in \Gamma_B(s) \forall s' \in S. \delta(s, a, b, s') \rightarrow s' \in Q\} \cup \{(s, b, 0) \in S' \mid b \in \Gamma_B(s) \land \forall a \in \Gamma_A(s) \forall s' \in S. \delta(s, a, b, s') \rightarrow s' \in Q\}$
- $\pre_{\mathcal{K}_G}(Q \times \mathcal{A} \times \{1\}) = \{(s, \text{nop}, 0) \in S' \mid \exists a \in \Gamma_A(s). \exists b \in \Gamma_B(s) \exists s' \in S. \delta(s, a, b, s') \land s' \in Q\}$
- $\pre_{\mathcal{K}_G}(Q \times \mathcal{A} \times \{0\}) = \{(s, \text{nop}, 0) \in S' \mid \exists a \in \Gamma_A(s). \exists b \in \Gamma_B(s) \exists s' \in S. \delta(s, a, b, s') \land s' \in Q\}$
- $\pre_{\mathcal{K}_G}(Q \times \mathcal{A} \times \{1\}) = \{(s, \text{nop}, 0) \in S' \mid \exists a \in \Gamma_A(s). \forall b \in \Gamma_B(s) \forall s' \in S. \delta(s, a, b, s') \rightarrow s' \in Q\}$
- $\pre_{\mathcal{K}_G}(Q \times \mathcal{A} \times \{0\}) = \{(s, \text{nop}, 0) \in S' \mid \exists a \in \Gamma_A(s). \forall b \in \Gamma_B(s) \forall s' \in S. \delta(s, a, b, s') \rightarrow s' \in Q\}$

8
Proof. We just show one case of the proof:

\[
\begin{align*}
\text{pre}^\mathcal{T}_\mathcal{G} (Q \times \{\text{nop}\} \times \{0\}) \\
= \{(s, a, t) \in S' | \exists (s', a', t') \in S'. \\
R((s, a, t), (s', a', t')) \wedge (s', a', t') \in Q \times \{\text{nop}\} \times \{0\}\}
\end{align*}
\]

\[
= \{(s, a, t) \in S' | \exists s' \in Q. R((s, a, t), (s', \text{nop}, 0))\}
\]

\[
\begin{cases}
(s, a, t) \in S', \\
\begin{cases}
\iota = 1 \wedge a \in \Gamma_A(s) \wedge \exists b \in \Gamma_B(s). \delta(s, a, b, s') \vee \\
\iota = 0 \wedge a \in \Gamma_B(s) \wedge \exists b \in \Gamma_A(s). \delta(s, a, b, s')
\end{cases}
\end{cases}
\]

\[
= \{(s, a, 1) \in S' | a \in \Gamma_A(s) \wedge \exists b \in \Gamma_B(s) \exists s' \in S. \delta(s, a, b, s') \wedge s' \in Q\} \cup \\
\{(s, 0, 0) \in S' | b \in \Gamma_B(s) \wedge \exists a \in \Gamma_A(s) \exists s' \in S. \delta(s, a, b, s') \wedge s' \in Q\}
\]

\[
\square
\]

Theorem 1. Given a concurrent game structure \(\mathcal{G} = (I, S, \delta, \Gamma_A, \Gamma_B, \mathcal{L})\) for a set of variables \(\mathcal{V}\) and a set of actions \(A\), and an arbitrary formula \(\varphi \in \mathcal{L}_{AT}\). Then, the following holds:

\[
[\text{AT}_\mu (\varphi)]_{\mathcal{K}_\mathcal{G}} = [\varphi]_\mathcal{G} \times \{\text{nop}\} \times \{0\}
\]

Proof. We assume that the associated Kripke structure \(\mathcal{K}_\mathcal{G}\) of the game structure \(\mathcal{G}\) is \(\mathcal{K}_\mathcal{G} = (I', S', \mathcal{R}, \mathcal{L}')\). We first prove the result for formulas \(\varphi \in \mathcal{L}_{AT}\) without fixpoint operators. This proof is done by an induction on the formula \(\varphi \in \mathcal{L}_{AT}\):

\(\varphi = 0\): For the Boolean constant 0, we have

\[
[\text{AT}_\mu (0)]_{\mathcal{K}_\mathcal{G}} = [0]_{\mathcal{K}_\mathcal{G}} = \emptyset = \emptyset \times \{\text{nop}\} \times \{0\} = [0]_\mathcal{G} \times \{\text{nop}\} \times \{0\}
\]

\(\varphi = 1\): For the Boolean constant 1, we have

\[
[\text{AT}_\mu (1)]_{\mathcal{K}_\mathcal{G}} = [xN]_{\mathcal{K}_\mathcal{G}} = S \times \{\text{nop}\} \times \{0\} = [1]_\mathcal{G} \times \{\text{nop}\} \times \{0\}
\]

\(\varphi \in \mathcal{V}\): For a variable \(x \in \mathcal{V}\), we have

\[
[\text{AT}_\mu (x)]_{\mathcal{K}_\mathcal{G}} = [x]_{\mathcal{K}_\mathcal{G}} = [x]_\mathcal{G} \times \{\text{nop}\} \times \{0\}
\]

\(\lnot \varphi\): For negations, we have (at equation (1), note that for arbitrary sets \(A, B, \) and \(C\), with \(A \subseteq B\), we have \(A \cap (B \setminus C) = A \setminus C\), and at equation (2), note that \((A \times C) \setminus (B \times C) = (A \setminus B) \times C\):

\[
[\text{AT}_\mu (\lnot \varphi)]_{\mathcal{K}_\mathcal{G}} = [xN \land \lnot \text{AT}_\mu (\varphi)]_{\mathcal{K}_\mathcal{G}}
\]

\[
= [xN]_{\mathcal{K}_\mathcal{G}} \cap \lnot [\text{AT}_\mu (\varphi)]_{\mathcal{K}_\mathcal{G}}
\]

\[
= [xN]_{\mathcal{K}_\mathcal{G}} \cap S' \setminus [\text{AT}_\mu (\varphi)]_{\mathcal{K}_\mathcal{G}}
\]

\[
= [xN]_{\mathcal{K}_\mathcal{G}} \cap S' \setminus ([\varphi]_\mathcal{G} \times \{\text{nop}\} \times \{0\})
\]

\[
= (S \times \{\text{nop}\} \times \{0\}) \cap S' \setminus ([\varphi]_\mathcal{G} \times \{\text{nop}\} \times \{0\})
\]

\[\text{(1)}\]

\[
= (S \setminus [\varphi]_\mathcal{G}) \times \{\text{nop}\} \times \{0\}
\]

\[\text{(2)}\]

\[
= [-\varphi]_\mathcal{G} \times \{\text{nop}\} \times \{0\}
\]

\[
\text{with } \mathcal{L}_{AT} = \mathcal{L}_{Z\Gamma} \cap \text{AT}
\]

\[
[\text{AT}_\mu (\text{AT}_\mu (\varphi))]_{\mathcal{K}_\mathcal{G}} = [\text{AT}_\mu (\varphi)]_{\mathcal{K}_\mathcal{G}}
\]

\[
\square
\( \varphi \land \psi \): For conjunctions, we have (at equation (3), note that for arbitrary sets \( A, B, \) and \( C \), we have: \( (A \times C) \cap (B \times C) = (A \cap B) \times C \):

\[
\begin{align*}
\llbracket AT_\mu (\varphi \land \psi) \rrbracket_{K_G} &= \llbracket AT_\mu (\varphi) \land AT_\mu (\psi) \rrbracket_{K_G} \\
&= \llbracket AT_\mu (\varphi) \rrbracket_{K_G} \cap \llbracket AT_\mu (\psi) \rrbracket_{K_G} \\
&= (\llbracket \varphi \rrbracket_G \times \{\text{nop} \} \times \{0\}) \cap (\llbracket \psi \rrbracket_G \times \{\text{nop} \} \times \{0\}) \\
&= (\llbracket \varphi \land \psi \rrbracket_G) \times \{\text{nop} \} \times \{0\}
\end{align*}
\]

\( \Diamond \varphi \): For modal formulas \( \Diamond \varphi \), we have

\[
\begin{align*}
\llbracket AT_\mu (\Diamond \varphi) \rrbracket_{K_G} &= \llbracket \Diamond \Diamond AT_\mu (\varphi) \rrbracket_{K_G} \\
&= \text{pre}_3^R (\llbracket \Diamond AT_\mu (\varphi) \rrbracket_{K_G}) \\
&= \text{pre}_3^R (\text{pre}_3^R (\llbracket AT_\mu (\varphi) \rrbracket_{K_G})) \\
&= \text{pre}_3^R (\text{pre}_3^R (\llbracket \varphi \rrbracket_G \times \{\text{nop} \} \times \{0\})) \\
&= \text{pre}_3^R \left( \begin{cases} 
(s, a, 1) \in S' | a \in \Gamma_A(s) \land \exists b \in \Gamma_B(s). \delta(s, a, b) \in \llbracket \varphi \rrbracket_G \\
(s, b, 0) \in S' | b \in \Gamma_B(s) \land \exists a \in \Gamma_A(s). \delta(s, a, b) \in \llbracket \varphi \rrbracket_G 
\end{cases} \right) \\
&= \text{pre}_3^R \left( \begin{cases} 
(s, a, 1) \in S' | a \in \Gamma_A(s) \land \exists b \in \Gamma_B(s) \exists s' \in S. \\
(s, b, 0) \in S' | b \in \Gamma_B(s) \land \exists a \in \Gamma_A(s) \exists s' \in S. \\
\delta(s, a, b, s') \land s' \in \llbracket \varphi \rrbracket_G 
\end{cases} \right) \\
&= \{ (s, \text{nop}, 0) \in S' | \exists a \in \Gamma_A(s). \exists b \in \Gamma_B(s) \exists s' \in S. \delta(s, a, b, s') \land s' \in \llbracket \varphi \rrbracket_G \} \\
&= [\Diamond \Diamond \varphi]_G \times \{\text{nop} \} \times \{0\}
\end{align*}
\]

\( \Box_A \varphi \): For player \( A \) modal formulas \( \Box_A \varphi \), we have

\[
\begin{align*}
\llbracket AT_\mu (\Box_A \varphi) \rrbracket_{K_G} &= \llbracket (x_A \land \Box AT_\mu (\varphi)) \rrbracket_{K_G} \\
&= \text{pre}_3^R (\llbracket (x_A \land \Box AT_\mu (\varphi)) \rrbracket_{K_G}) \\
&= \text{pre}_3^R (\llbracket x_A \rrbracket_{K_G} \cap \llbracket \Box AT_\mu (\varphi) \rrbracket_{K_G}) \\
&= \text{pre}_3^R (\llbracket x_A \rrbracket_{K_G} \cap \text{pre}_3^R (\llbracket AT_\mu (\varphi) \rrbracket_{K_G})) \\
&= \text{pre}_3^R (\llbracket x_A \rrbracket_{K_G} \cap \text{pre}_3^R (\llbracket \varphi \rrbracket_G \times \{\text{nop} \} \times \{0\})) \\
&= \text{pre}_3^R \left( \begin{cases} 
(s, a, 1) \in S' | a \in \Gamma_A(s) \land \forall b \in \Gamma_B(s) \forall s' \in S. \\
\delta(s, a, b, s') \rightarrow s' \in Q 
\end{cases} \right) \\
&= \text{pre}_3^R \left( \begin{cases} 
(s, a, 1) \in S' | a \in \Gamma_A(s) \land \forall b \in \Gamma_B(s) \forall s' \in S. \\
\delta(s, a, b, s') \rightarrow s' \in Q 
\end{cases} \right) \\
&= \{ (s, \text{nop}, 0) | \forall a \in \Gamma_A(s). \forall b \in \Gamma_B(s) \forall s' \in S. \delta(s, a, b, s') \rightarrow s' \in \llbracket \varphi \rrbracket_G \} \\
&= [\Box_A \varphi]_G \times \{\text{nop} \} \times \{0\}
\end{align*}
\]
The remaining cases are trivial: The proof for disjunctions $\varphi \lor \psi$ is analogous to the that of conjunctions. Formulas $\square \varphi$ are handled by the equivalence $\square \varphi = \neg \diamond \neg \varphi$, and the already proved facts for negation and $\diamond$. Finally, the remaining case for player $B$ modal formulas $\Box_B \varphi$ is analogous to the proof of player $A$ modal formulas $\Box_A \varphi$.

Hence, we have seen that the proposition holds for formulas without fixpoint operators. The generalization to arbitrary formulas is seen as follows: Due to the finiteness of the structures, every fixpoint formula $\sigma x. \varphi$ is equivalent to one of its approximants. As these are fixpoint-free formulas, the result also holds for fixpoint formulas. This can be proved formally by an induction on the nesting depth of fixpoint operators.

For the complexity, we note that in the structure, we get an increase from $|S|$ states in the game to $2|S|(|A| + 1)$ states in the associated Kripke structure. Since $|A|$ is a constant which is small compared to $|S|$, we get a constant blow-up. Also the size of the transition relation is doubled, resulting by splitting the transitions into two different transitions to simulates transitions of the game.

Concerning the formula, we note that every alternating time $\mu$-calculus formula is translated to an equivalent $\mu$-calculus formula with a size which is linear in the size of the original formula. Keeping in mind, that the $\mu$-calculus model checking of alternation depth $l$, whose length is $|\Phi|$ can be done on every Kripke structure $K = \langle S, I, R, \mathcal{L} \rangle$ in time $O \left( \frac{|\Phi| |S|}{l} \left( l - 1 \right) |R| |\Phi| \right)$ (see [16]), we immediately obtain the following theorem:

**Theorem 2.** For every $\mathcal{L}_{AT}$-formula $\Phi$ of alternation depth $l$ and length $|\Phi|$, and every concurrent game $G = (I, S, \delta, \Gamma_A, \Gamma_B, \mathcal{L})$ there is an algorithm to compute its solution in time

$$O \left( \frac{|\Phi| |S|}{l} \left( l - 1 \right) |\Phi| \right).$$

In [1] an algorithm is presented that modified an existing $\mu$-calculus model-checking algorithm to be suited for alternating time $\mu$-calculus model-checking. This algorithm involves the calculation of some turn-based games in each iteration step. Instead, our algorithm performs this work in one simple step at the beginning of the algorithm. The above improvement of the complexity compared to the complexity $O \left( (|\Phi| |\delta|)^{l+1} \right)$ given in [1] is due to improvements on $\mu$-calculus model checking that have been found since then. Nevertheless, it shows that our reduction is efficient, i.e, there is no loss of efficiency due to the reduction.

### 4 Supervisory Control

In this section, we show how to solve the supervisory control problem as introduced by Ramadge and Wonham [14] with the so far developed tools. In supervisory control, a finite state automaton representing a system (called a plant) and a specification (also a finite automaton), are given. The set of events $\Sigma$ that are generated during execution of the plant are partitioned into a set of controllable events $\Sigma_c$ that can be prevented from occurring by a controller and the uncontrollable events $\Sigma_u$ that are not influenced
by the controller. It is the task of the controller to guarantee that in every state of the plant under control, it is possible to reach a so-called marked state\(^4\). In other words, every active process may eventually terminate. While trying to achieve this goal, the controller has to ensure that the specification is not violated, i.e., the language generated by the plant under control must be a subset of the generated language of the automaton representing the specification.

As usual in supervisory control, we assume that we are given a finite state machine \(M = \langle Q, \Sigma, \delta, q_0, Q_m \rangle\) representing the cross product of the specification and the plant. Using the cross product, it is guaranteed that the specification is not violated by controllable events. However, this is not the case for the uncontrollable events. Therefore, the initial bad states \(Q_b\) are introduced. An initial bad state is a state pair from the cross product where in the state of the plant an uncontrollable event leads to a transition that leaves the specification. Clearly, a supervisor has to assure that such states are not reached because they violate the specification. On the other side, it is also sufficient to avoid to reach those states (see [18] for explanations).

In [18], the supervisory control problem is solved by introducing the associated Kripke structure of an automaton:

**Definition 9 (Kripke Structure of an Automaton).** Given an automaton \(M = \langle Q, \Sigma, \delta, q_0, Q_m \rangle\) representing the product of a plant and a specification, we define its associated Kripke structure \(K_A = \langle S, I, R, L \rangle\) over the Boolean variables \(V_A := \{x_q \mid q \in Q\} \cup \{x_b, x_m, x_u\}\) as follows:

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

This Kripke structure can be divided in two parts: One part that corresponds to the uncontrollable events, and the other that corresponds to all events.

We will mimic the behavior of this Kripke structure by a concurrent game as follows: the concurrent game \(G_M\) is defined over the set of actions \(A = \{\text{move}, \text{nothing}\}\) and is played by two player \(C\) and \(U\) (to give a reference to the controllable and uncontrollable events) as follows\(^5\):

- \(S = Q\)

\(^4\) Note, that it is not required that the state is actually reached.

\(^5\) One might wonder, why we did not use the event set as part of the action set. Clearly, this is possible, but leads to an unnecessary blowup in the associated Kripke structure. For solving the supervisory control problem, we only have to distinguish between controllable and uncontrollable events without knowing the particular events.
\[ \Gamma_C(s) = \begin{cases} \{ \text{nothing, move} \} & \text{if } \text{act}_M(s) \cap \Sigma_c \neq \emptyset \\ \{ \text{nothing} \} & \text{else} \end{cases} \]

\[ \Gamma_U(s) = \begin{cases} \{ \text{nothing, move} \} & \text{if } \text{act}_M(s) \cap \Sigma_u \neq \emptyset \\ \{ \text{nothing} \} & \text{else} \end{cases} \]

\[ \delta'(s, \alpha, \beta, s') \iff \\
\alpha = \text{move} \land \beta = \text{nothing} \land \exists \sigma \in \Sigma_c. \delta(q, \sigma) = s' \]

\[ \beta = \text{move} \land \exists \sigma \in \Sigma_u. \delta(q, \sigma) = s' \]

\[ \mathcal{L}(s) = \begin{cases} \{ x_b, x_m \} & \text{if } s \in Q_m \cap Q_b \\ \{ x_b \} & \text{if } s \in Q_b \\ \{ x_m \} & \text{if } s \in Q_m \\ \{ \} & \text{else} \end{cases} \]

where \( \text{act}_K(s) = \{ \sigma \in \Sigma \mid \exists s'. \delta(s, \sigma, s') \} \)

It is the task of player \( U \) to move the uncontrollable events by selecting \text{move}. To denote that no uncontrollable event arises, player \( U \) may choose action \text{nothing}. In this case, it is the task of player \( C \) to choose the next event, or player \( C \) may choose that no event occurs by selecting \text{nothing}. The definition of the transition relation ensures that the controllability property is captured in our game structure: an uncontrollable event may not be prevented by player \( C \), while every controllable event may be prevented by player \( C \).

**Lemma 3.** Let \( \phi \) be an alternating time formula. Then the following holds:

\[ \Box_U \phi \models_{\Sigma, M} = \{ q \in Q \mid \exists \sigma \in \Sigma_u. \delta(q, \sigma) \in [\phi]_{\Sigma, M} \} \]

**Proof.** The proposition directly follows from the definition of \( \Box_U \): If player \( U \) chooses \text{nothing}, then player \( C \) may also choose \text{nothing}, and the game would stop. So, if player \( U \) could navigate the game into a state where \( \Phi \) does hold, this must be an action that corresponds to an uncontrollable event.

We need a similar result on the corresponding Kripke structure \( K_M \):

**Lemma 4.** Let \( \phi \) be a \( \mu \)-calculus formula. Then the following holds:

\[ \Box \phi \land x_u \models_{K, M} = \{ (q, 0) \mid \exists \sigma \in \Sigma_u. (\delta(q, \sigma), 0) \in [\phi]_{K, M} \} \]

\[ [\Box \phi \land x_u]_{K, M} = \{ (q, 0) \mid \forall \sigma \in \Sigma_u. (\delta(q, \sigma), 0) \in [\phi]_{K, M} \} \]

The proof follows directly from the definition of the associated Kripke structure \( K_M \).

To solve the supervisory control problem, we can proceed similar to [18]: A state of the cross product does not violate the specification, if it is

- Co-reachable, i.e. a marked state may be reached and
- good, i.e. no bad states may be reachable via an uncontrollable event.
Therefore, in [18], an equation system is given that works on the above defined Kripke structure:

\[
\begin{align*}
    \nu x_{Co} &\triangleq \kappa(x_G) \land (\Diamond x_{Co} \lor x_m) \\
    \nu x_G &\triangleq x_u \land (\Box (x_G \land \kappa(x_{Co})) \land \neg x_b).
\end{align*}
\]

This is an equation system of the ordinary $\mu$-calculus with the difference that an operator $\kappa()$ is used to switch from one part of the Kripke structure to the other according to the following definition: $\kappa(Q) := \{(q, \neg i) \mid (q, i) \in Q\}$.

A detailed discussion on the correctness of the above equation system is found in [18].

In the following, we concentrate on the corresponding equation system on the associated game. The coreachable states can be calculated by the following formula:

\[
x'_C \triangleq x'_G \land (\Diamond x'_{Co} \lor x_m),
\]

where $x'_{Co}$ denotes the coreachable states and $x'_G$ the good states. For calculating the bad states, we note that a state is bad, if an uncontrollable event leads to a state which is known to be bad or non-coreachable. Therefore a state is bad, if player $B$ can enforce a visit to an already known bad state, or to a state which is known to be non-coreachable. This is calculated by the following formula:

\[
x'_B \triangleq \Box (x'_B \lor \neg x'_{Co}) \lor x_b.
\]

By negation of the above equation, we get the good states, i.e., those states that should never be left during a game:

\[
x_G \triangleq \neg \Box (\neg x'_G \lor \neg x'_{Co}) \land \neg x_b.
\]

We therefore get the following equation system:

\[
\begin{align*}
    x'_{Co} &\triangleq x_G \land (\Diamond x'_{Co} \lor x_m) \\
    x'_G &\triangleq \neg \Box (\neg x'_G \lor \neg x'_{Co}) \land \neg x_b.
\end{align*}
\]

This equation system is very similar to the one that has been presented in [18], and indeed, when we take into account, that $\Box \phi = \neg \Diamond \neg \phi$ holds, we see that the $\kappa()$-operator has been replaced to respect the nonblocking property. Since $[x'_{Co}]_{K,G} \subseteq [x'_G]_{K_C,G}$, $x_{Co}$ does contain those states that are good and coreachable. A solution to the supervisory control problem may be given by evaluating the above equation system where the reachable states are restricted to the coreachable states. To show the correctness of the presented algorithm, we will give a reduction to the equation system formulated in [18]:

**Theorem 3.** Given an automaton $\mathcal{M} = (Q, \Sigma, \delta, q_0, Q_m, Q_b)$ representing the product of a plant and a specification, its associated Kripke structure $K_\mathcal{M}$, and its associated game $G_\mathcal{M}$, the following holds:

\[
[x_{Co}]_{K_\mathcal{M}} = [x'_{Co}]_{G_\mathcal{M}} \times \{1\}
\]
Proof. It is sufficient to show that for every iteration step \( i \) of the fixpoint evaluation the following holds: \( \llbracket x_{C_0} \rrbracket_{K_M}^i = \llbracket x_{C_0} \rrbracket_G^i \times \{1\} \) and \( \llbracket x_G \rrbracket_{K_M}^i = \llbracket x_G \rrbracket_G^i \times \{0\} \). We will show this by simultaneous induction on \( i \).

The base case \( i = 0 \) is obvious and left to the reader. Now consider the inductive step:

\[
\llbracket x_G' \rrbracket_{G_M}^{i+1} = \llbracket \neg x_b \rrbracket_{G_M}^{i+1} \cap \llbracket \neg \Xi U (\neg x_G' \lor \neg x_{C_0}') \rrbracket_{G_M}^{i+1}
\]
\[
= \llbracket \neg x_b \rrbracket_{G_M}^{i+1} \cap Q \setminus \llbracket \Xi U (\neg x_G' \lor \neg x_{C_0}') \rrbracket_{G_M}^{i+1}
\]

**Lemma 3**: \( \llbracket \neg x_b \rrbracket_{G_M} \cap Q \setminus \{ q \in Q \mid \exists \sigma \in \Sigma_u. \delta(q, \sigma) \in \llbracket \neg x_G' \lor \neg x_{C_0}' \rrbracket_{G_M}^{i} \} \)

Therefore, it holds, that: (note that we use lemma 4 at equation (1))
Finally, consider the fixpoint iteration of $x'_{C_0}$:

$$
\begin{align*}
[x_{C_0}]_{K,M}^{i+1} & = \left[\kappa(x_G)\right]_{K,M}^{i+1} \cap \left[\diamond \left( x_{C_0} \lor x_m \right) \right]_{K,M}^{i+1} \\
& = \{(q, 1) \in Q \times \{1\} \mid (q, 0) \in \left[x_G\right]_{K,M}^{i+1} \cap \left[\diamond \left( x_{C_0} \lor x_m \right) \right]_{K,M}^{i+1} \\
& = [x_G]_{\sigma,M}^{i+1} \times \{1\} \cap \left[\diamond \left( x_{C_0} \lor x_m \right) \right]_{K,M}^{i+1} \\
& = \left[\diamond \left( x_{C_0} \lor x_m \right) \right]_{\sigma,M}^{i+1} \times \{1\} \cap \left\{ (q, 1) \in Q \times \{1\} \mid \exists \sigma \in \Sigma. \ (\delta(q, \sigma), 1) \in \left[x_c \lor x_m\right]_{K,M}^{i} \times \{1\} \\
& = [x_c]_{\sigma,M}^{i+1} \times \{1\} \cap \left\{ (q, 1) \in Q \times \{1\} \mid \exists \sigma \in \Sigma. \ (\delta(q, \sigma), 1) \in \left[x_c \lor x_m\right]_{\sigma,M}^{i} \times \{1\} \\
& = \left[\diamond \left( x_{C_0} \lor x_m \right) \right]_{\sigma,M}^{i+1} \times \{1\} \cap \left\{ (q, 1) \in Q \times \{1\} \mid (q, 1) \in \left[\diamond x_c \lor x_m\right]_{\sigma,M}^{i} \times \{1\} \\
& = \left[\diamond \left( x_{C_0} \lor x_m \right) \right]_{\sigma,M}^{i+1} \times \{1\}
\end{align*}
$$

\[\square\]

5 Conclusion

In this paper, we described a translation from alternating time $\mu$-calculus model-checking problems (on concurrent games) to equivalent propositional $\mu$-calculus model-checking problems (on Kripke structures). This has the advantage that already existing model checking tools can be used to solve problems of game structures that are formalized by the alternating time $\mu$-calculus. Moreover, we presented a reduction of supervisory control theory to concurrent games, that allows us to use $\mu$-calculus model checking algorithms to solve the classical supervisory control problem as introduced by Ramadge and Wonham.

Therefore, our framework may be seen as a fronted for both supervisor synthesis and verification on top of existing model checkers like our toolset Averest [15]. The algorithms are currently implemented and we expect to present first experimental results at the workshop.

References


