Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

Using Model Checking to Solve Supervisor Synthesis Problems

Andreas Morgenstern and Klaus Schneider

Abstract— ***Verification procedures, which check whether a given system satisfies a given specification, are nowadays mature for industrial usage. The more general supervisor synthesis problem asks how a system has to be restricted or which actions have to be selected such that the system satisfies a given specification. Supervisor synthesis problems are often formulated in frameworks like game structures that are more general than the Kripke structures that are traditionally used in verification. For this reason, current verification tools can not be used for supervisory control problems.***

In this paper, however, we present a reduction of alternating time μ-calculus model checking problems (on game structures) to model checking problems of the μ-calculus on Kripke structures. As a result, arbitrary model checkers can be used to solve supervisor synthesis problems. As a demonstration of the applicability of our approach, we show how the classical supervisory control problems of Ramadge and Wonham can be solved within our framework.

I. INTRODUCTION

In the past two decades, many verification procedures for the temporal behavior of reactive systems have been developed [Schneider, 2003], and this research already lead to tools that are used in industry. These tools are able to check whether a system $\mathcal{K}$ satisfies a given temporal specification φ. There are a lot of formalisms, in particular, the μ-calculus [Kozen, 1983], ω-automata [Thomas, 1990], as well as various temporal logics [Pnueli, 1977], [Emerson and Clarke, 1982] and (monadic) predicate logics [Büchi, 1960] to formulate the specification φ [Schneider, 2003]. Moreover, the increased industrial interest already lead to standardization efforts on specification logics [Beer et al., 2001], [Accellera, 2004].

Besides the verification problem, where the entire system $\mathcal{K}$ and its specification must be already available, the more general controller/supervisor synthesis problem is of interest for the development of safety-critical systems. The task is here to check whether it is possible to restrict a given system $\mathcal{K}$ such that the restriction $\mathcal{K}'$ satisfies a given specification φ. 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: well established variants are, in particular, the *supervisory control problem* [Ramadge and Wonham, 1987], *module checking* [Kupferman and Vardi, 1996], and *alternating time μ-calculus model checking* [Alur et al., 2002]. In all of the mentioned references, the general question is to check whether the system can be restricted such that a given specification is met. While in [Kupferman and Vardi, 1996] and [Alfaro et al., 2001], it was discussed whether supervisor synthesis problems can be solved by ordinary model checkers, [Alur et al., 2002] and [Ramadge and Wonham, 1987] used special specification languages and tools. We briefly consider the two latter approaches.

In supervisory control [Ramadge and Wonham, 1987], both the 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 [Ziller and Schneider, 2003], [Ziller and Schneider, 2005] to more general specifications given in the μ-calculus. This is achieved by translating supervisory control problems to model checking problems on a Kripke structure. However, the temporal logic that was used required a slight extension of the μ-calculus by a special operator κ.

Alternating time μ-calculus specifications as presented in [Alur et al., 2002] are evaluated on finite state transition systems where two players (representing the system and the environment) concurrently and independently select actions that are allowed in the considered state. Alternating time μ-calculus extends the ordinary μ-calculus with new modal operators to describe that one of the players can enforce by its choice of an action that a certain state set is reached, regardless of the choice of the other player. Using alternating time μ-calculus, it is therefore easier to formulate statements like 'the controller can enforce that certain states are visited irrespectively of how the environment behaves'. Such typical properties can not be directly formulated using the ordinary μ-calculus. Therefore, alternating time μ-calculus and the derived alternating time temporal logics ATL and ATL* have been proposed as extensions to traditional temporal logics like LTL or CTL* that are already used for verification.

In this paper, however, we will show how alternating time μ-calculus model checking problems (on concurrent game structures) are translated to ordinary μ-calculus model checking problems (on Kripke structures). The purpose of this translation is to use the efficient machinery offered by existing model checking tools to solve supervisory control problems. We emphasize that we do not argue that alternating time μ-calculus is unnecessary. Instead, we want to show that already available μ-calculus model checkers can be used to solve problems that were formulated with alternating time μ-

calculus.

To demonstrate the applicability of our translation, we describe the supervisory control problem of [Ramadge and Wonham, 1987] in alternating time μ-calculus, which allows us a further translation to an ordinary μ-calculus model checking problem. In comparison to the translation given in [Ziller and Schneider, 2003], [Ziller and Schneider, 2005], the approach presented in this paper does not require the introduction of new operators in the μ-calculus. Hence, a side product of our approach is the reduction of the classical supervisory control problem to a *pure μ-calculus model-checking problem* that can be solved with standard tools.

The outline of the paper is as follows: In the next section, we review basic definitions of Kripke structures, the propositional μ-calculus, and the alternating time μ-calculus. In Section III, we present our reduction from alternating time μ-calculus model checking to the propositional μ-calculus model checking. Section IV describes the reduction from supervisory control problems to alternating time μ-calculus model checking.

II. FORMAL BACKGROUND

A. The Propositional μ-Calculus

The propositional μ-calculus is a well-known specification logic whose model checking algorithms form the heart of state-of-the-art verification tools [Schneider, 2003]. We briefly present its syntax and semantics in this section.

Definition 1 (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 $\mathcal{L}_\mu$ that satisfies the following rules:

- $\mathcal{V} \cup \{0,1\} \subseteq \mathcal{L}_\mu$
- $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi \in \mathcal{L}_\mu$, provided that $\varphi, \psi \in \mathcal{L}_\mu$
- $\Diamond\varphi, \Box\varphi \in \mathcal{L}_\mu$, provided that $\varphi \in \mathcal{L}_\mu$
- $\mu x.\varphi, \nu x.\varphi \in \mathcal{L}_\mu$, provided that $\varphi \in \mathcal{L}_\mu$ and $x \in \mathcal{V}$.

The semantics is defined on labeled transition systems which are often called Kripke structures due to the relationship to modal logic [Schneider, 2003].

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

The meaning of a $\mathcal{L}_\mu$ formula is a set of states that is recursively described in the next definition. To this end, we make use of the following abbreviations of existential and universal predecessors of a state set Q:

$$\mathsf{pre}_\exists^{\mathcal{R}}(Q) = \{s \in \mathcal{S} \mid \exists s' \in \mathcal{S}.\ \mathcal{R}(s,s') \wedge s' \in Q\}$$
$$\mathsf{pre}_\forall^{\mathcal{R}}(Q) = \{s \in \mathcal{S} \mid \forall s' \in \mathcal{S}.\ \mathcal{R}(s,s') \to s' \in Q\}$$

$\mathsf{pre}_\exists^{\mathcal{R}}(Q)$ is the set of states that have at least one successor state in Q, and $\mathsf{pre}_\forall^{\mathcal{R}}(Q)$ is the set of states that have no successor states outside Q. Using these set operators, the semantics is as follows:

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

- $[\![0]\!]_{\mathcal{K}} := \{\}$
- $[\![1]\!]_{\mathcal{K}} := \mathcal{S}$
- $[\![x]\!]_{\mathcal{K}} := \{s \in \mathcal{S} \mid x \in \mathcal{L}(s)\}$ for all variables $x \in \mathcal{V}$
- $[\![\neg\varphi]\!]_{\mathcal{K}} := \mathcal{S} \setminus [\![\varphi]\!]_{\mathcal{K}}$
- $[\![\varphi \vee \psi]\!]_{\mathcal{K}} := [\![\varphi]\!]_{\mathcal{K}} \cup [\![\psi]\!]_{\mathcal{K}}$
- $[\![\varphi \wedge \psi]\!]_{\mathcal{K}} := [\![\varphi]\!]_{\mathcal{K}} \cap [\![\psi]\!]_{\mathcal{K}}$
- $[\![\Diamond\varphi]\!]_{\mathcal{K}} := \mathsf{pre}_\exists^{\mathcal{R}}([\![\varphi]\!]_{\mathcal{K}})$
- $[\![\Box\varphi]\!]_{\mathcal{K}} := \mathsf{pre}_\forall^{\mathcal{R}}([\![\varphi]\!]_{\mathcal{K}})$
- $[\![\mu x.\varphi]\!]_{\mathcal{K}} := \bigcap\{Q \subseteq \mathcal{S} \mid [\![\varphi]\!]_{\mathcal{K}_x^Q} \subseteq Q\}$, where $\mathcal{K}_x^Q$ is obtained by changing the label function $\mathcal{L}$ of $\mathcal{K}$ such that exactly the states in Q are labeled with x.
- $[\![\nu x.\varphi]\!]_{\mathcal{K}} := \bigcup\{Q \subseteq \mathcal{S} \mid [\![\varphi]\!]_{\mathcal{K}_x^Q} \subseteq Q\}$, where $\mathcal{K}_x^Q$ is defined as above.

Global model checking procedures are used to compute the set of states $[\![\Phi]\!]_{\mathcal{K}}$ for given Φ and $\mathcal{K}$. Many sophisticated algorithms have been developed in the past that even take advantage of abstractions and other techniques to handle complex problems [Schneider, 2003].

B. Alternating Time μ-Calculus

Concurrent game structures [Alur et al., 2002] extend Kripke structures as follows: For every transition, two players A and B choose independently of each other actions α and β, respectively, and the resulting pair (α, β) uniquely determines the next state of the game.

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

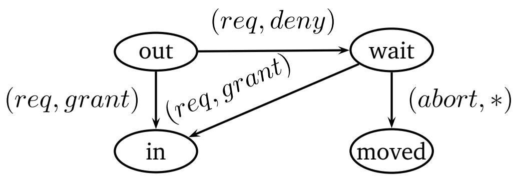

Fig. 1. Example for a Concurrent Game Structure

Figure 1 illustrates an example for a concurrent game structure that formalizes a protocol for a train entering a railroad station (we use * to represent any action).

Since concurrent game structures are generalizations of Kripke structures, we can directly use Definition 3 to evaluate μ-calculus formulas on concurrent game structures. However, the additional labels on the transitions induce further modal operators [Alur et al., 2002]: These additional modal operators express that one of the players can enforce by a suitable choice of an action that, regardless of the choice of the other player, a certain state set is reached. The addition of these modal operators to the μ-calculus yields the alternating time μ-calculus:

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

- $\mathcal{V} \cup \{0,1\} \subseteq \mathcal{L}_{AT}$
- $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi \in \mathcal{L}_{AT}$, provided that $\varphi, \psi \in \mathcal{L}_{AT}$
- $\Diamond\varphi, \Box\varphi \in \mathcal{L}_{AT}$, provided that $\varphi \in \mathcal{L}_{AT}$
- $\boxed{\Diamond}_A\varphi, \boxed{\Diamond}_B\varphi \in \mathcal{L}_{AT}$, provided that $\varphi \in \mathcal{L}_{AT}$
- $\mu x.\varphi, \nu x.\varphi \in \mathcal{L}_{AT}$, provided that $\varphi \in \mathcal{L}_{AT}$

The semantics of $\mathcal{L}_{AT}$ is defined on concurrent game structures analogously to the semantics of $\mathcal{L}_\mu$ on Kripke structures. The interesting new point is the semantics of the new modal operators $\boxed{\Diamond}_A$ and $\boxed{\Diamond}_B$:

Definition 6 (Semantics of $\mathcal{L}_{AT}$): Given a concurrent game structure $\mathcal{G} = (\mathcal{I}, \mathcal{S}, \delta, \Gamma_A, \Gamma_B, \mathcal{L})$ 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 $[\![\Phi]\!]_\mathcal{G} \subseteq \mathcal{S}$ by the following rules:

- $[\![0]\!]_\mathcal{G} := \{\}$
- $[\![1]\!]_\mathcal{G} := \mathcal{S}$
- $[\![x]\!]_\mathcal{G} := \{s \in \mathcal{S} \mid x \in \mathcal{L}(s)\}$ for all variables $x \in \mathcal{V}$
- $[\![\neg\varphi]\!]_\mathcal{G} := \mathcal{S} \setminus [\![\varphi]\!]_\mathcal{G}$
- $[\![\varphi \vee \psi]\!]_\mathcal{G} := [\![\varphi]\!]_\mathcal{G} \cup [\![\psi]\!]_\mathcal{G}$
- $[\![\varphi \wedge \psi]\!]_\mathcal{G} := [\![\varphi]\!]_\mathcal{G} \cap [\![\psi]\!]_\mathcal{G}$
- $[\![\Diamond\varphi]\!]_\mathcal{G} := \left\{ s \in \mathcal{S} \;\middle|\; \begin{array}{r} \exists a \in \Gamma_A(s).\exists b \in \Gamma_B(s). \\ \delta(s,a,b) \in [\![\varphi]\!]_\mathcal{G} \end{array} \right\}$
- $[\![\Box\varphi]\!]_\mathcal{G} := \left\{ s \in \mathcal{S} \;\middle|\; \begin{array}{r} \forall a \in \Gamma_A(s).\forall b \in \Gamma_B(s). \\ \delta(s,a,b) \in [\![\varphi]\!]_\mathcal{G} \end{array} \right\}$
- $[\![\boxed{\Diamond}_A\varphi]\!]_\mathcal{G} := \left\{ s \in \mathcal{S} \;\middle|\; \begin{array}{r} \exists a \in \Gamma_A(s).\forall b \in \Gamma_B(s). \\ \delta(s,a,b) \in [\![\varphi]\!]_\mathcal{G} \end{array} \right\}$
- $[\![\boxed{\Diamond}_B\varphi]\!]_\mathcal{G} := \left\{ s \in \mathcal{S} \;\middle|\; \begin{array}{r} \exists b \in \Gamma_B(s).\forall a \in \Gamma_A(s). \\ \delta(s,a,b) \in [\![\varphi]\!]_\mathcal{G} \end{array} \right\}$
- $[\![\mu x.\varphi]\!]_\mathcal{G} := \bigcap\{Q \subseteq \mathcal{S} \mid [\![\varphi]\!]_{\mathcal{G}_x^Q} \subseteq Q\}$, where $\mathcal{G}_x^Q$ is obtained by changing the label function $\mathcal{L}$ of $\mathcal{G}$ such that exactly the states in Q are labeled with x.
- $[\![\nu x.\varphi]\!]_\mathcal{G} := \bigcup\{Q \subseteq \mathcal{S} \mid [\![\varphi]\!]_{\mathcal{G}_x^Q} \subseteq Q\}$, where $\mathcal{G}_x^Q$ is defined as above.

Intuitively, a state s satisfies $\boxed{\Diamond}_A\varphi$ if player A can choose an action in state s such that, no matter which action player B chooses, a state s' is reached where φ holds. Analogously, $[\![\boxed{\Diamond}_B\varphi]\!]_\mathcal{G}$ is the set of states where player B can enforce that a state s' where φ holds is reached, independently of A's choice. Note that the above four modal operators cover all possible quantifier prefixes. As $\neg\Box\varphi$ is equivalent to $\Diamond\varphi$, we can even eliminate either $\Box$ or $\Diamond$. Note, however, that there is no such duality between $\boxed{\Diamond}_A$ and $\boxed{\Diamond}_B$, since their duals correspond to two further operators.

III. 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 φ to corresponding $\mathcal{L}_\mu$ formulas $\mathsf{AT}_\mu(\varphi)$.

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

- $\mathcal{I}' = \mathcal{I} \times \{\mathsf{nop}\} \times \{0\}$
- $\mathcal{S}' = \mathcal{S} \times (\mathcal{A} \cup \{\mathsf{nop}\}) \times \{0,1\}$
- $\mathcal{R}((s,\mathsf{nop},0),(s,\alpha,\iota))$
$$:\Leftrightarrow \left(\begin{array}{l} \iota = 1 \wedge \alpha \in \Gamma_A(s) \wedge \\ \qquad \exists b \in \Gamma_B(s).\ \exists s' \in \mathcal{S}.\ s' = \delta(s,\alpha,b) \\ \vee \\ \iota = 0 \wedge \alpha \in \Gamma_B(s) \wedge \\ \qquad \exists a \in \Gamma_A(s).\ \exists s' \in \mathcal{S}.\ s' = \delta(s,a,\alpha) \end{array} \right)$$
- $\mathcal{R}((s,\alpha,\iota),(s',\mathsf{nop},0))$
$$:\Leftrightarrow \left(\begin{array}{l} \iota = 1 \wedge \alpha \in \Gamma_A(s) \wedge \\ \qquad \exists b \in \Gamma_B(s).\ s' = \delta(s,\alpha,b) \\ \vee \\ \iota = 0 \wedge \alpha \in \Gamma_B(s) \wedge \\ \qquad \exists a \in \Gamma_A(s).\ s' = \delta(s,a,\alpha) \end{array} \right)$$
- $\mathcal{L}'((s,\alpha,\iota))$
$$:= \mathcal{L}(s) \cup \left\{ \begin{array}{ll} \{x_N\} & \text{if } \alpha = \mathsf{nop} \wedge \iota = 0 \\ \{x_A\} & \text{if } \alpha \in \Gamma_A(s) \wedge \iota = 1 \\ \{x_B\} & \text{if } \alpha \in \Gamma_B(s) \wedge \iota = 0 \end{array} \right.$$

$$\begin{tikzcd} s \arrow[dd, "{(a,b)}"'] & \Longrightarrow & & {s,\mathsf{nop},0} \arrow[dl] \arrow[dr] & \\ & & {s,a,1} \arrow[dr] & & {s,b,0} \arrow[dl] \\ s' & & & {s',\mathsf{nop},0} & \end{tikzcd}$$

Fig. 2. From Game Structures to Kripke Structures

The idea behind the construction of $\mathcal{K}_\mathcal{G}$ is the introduction of intermediate states as shown in Figure 2 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 $(s,\mathsf{nop},0)$ directly correspond to states $s \in \mathcal{S}$ of the game structure. Every transition from a state $s \in \mathcal{S}$ with $s' = \delta(s,a,b)$ is split into four transitions with two intermediate states as shown in Figure 2: one that leads from state $(s,\mathsf{nop},0)$ to $(s,a,1)$ which means that player A made the first move. Moreover, there is a transition from $(s,\mathsf{nop},0)$ to $(s,b,0)$ which means that player B made the first move. From states $(s,a,1)$ and $(s,b,0)$ there are further transitions to $(s',\mathsf{nop},0)$ to complete the transition of $\mathcal{G}$ in $\mathcal{K}_\mathcal{G}$.

States of the form $(s,\mathsf{nop},0)$ are called *choice states*. As each choice state $(s,\mathsf{nop},0)$ directly corresponds to the state s of the game structure, it is possible to identify sets of states of the game structure to corresponding states of the Kripke structure.

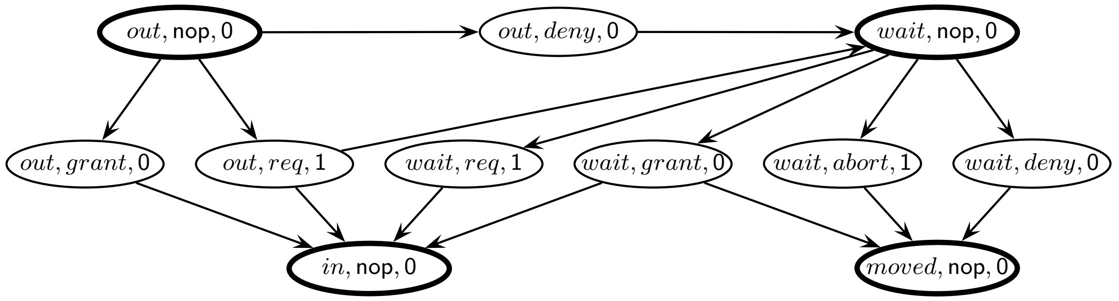

Fig. 3. Associated Kripke structure for the game structure of Figure 1

As an example for the translation, consider the associated Kripke structure shown in Figure 3 that is obtained from the game structure of Figure 1 (we omitted however the labels for $\{x_A, x_B, x_N\}$).

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

Lemma 1 (Invariants for the Construction of $\mathcal{K}_\mathcal{G}$): Given a concurrent game structure $\mathcal{G} = (\mathcal{I}, \mathcal{S}, \delta, \Gamma_A, \Gamma_B, \mathcal{L})$ and its associated Kripke structure $\mathcal{K}_\mathcal{G}$ as defined in Definition 7. Then, the following holds for every transition $\mathcal{R}((s, \alpha, \iota), (s', \alpha', \iota'))$:

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

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

We already explained the translation from game structures to Kripke structures. Next, we proceed with the translation of $\mathcal{L}_{AT}$ to $\mathcal{L}_\mu$:

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 $\mathsf{AT}_\mu(\varphi) \in \mathcal{L}_\mu$ over the variables $\mathcal{V} \cup \{x_A, x_B, x_N\}$ inductively as follows :

- $\mathsf{AT}_\mu(0) := 0$
- $\mathsf{AT}_\mu(1) := x_N$
- $\mathsf{AT}_\mu(x) := x$ for variables $x \in \mathcal{V}$
- $\mathsf{AT}_\mu(\neg\varphi) := x_N \wedge \neg\mathsf{AT}_\mu(\varphi)$
- $\mathsf{AT}_\mu(\varphi \wedge \psi) := \mathsf{AT}_\mu(\varphi) \wedge \mathsf{AT}_\mu(\psi)$
- $\mathsf{AT}_\mu(\varphi \vee \psi) := \mathsf{AT}_\mu(\varphi) \vee \mathsf{AT}_\mu(\psi)$
- $\mathsf{AT}_\mu(\Diamond\varphi) := \Diamond\Diamond\mathsf{AT}_\mu(\varphi)$
- $\mathsf{AT}_\mu(\Box\varphi) := \Box\Box\mathsf{AT}_\mu(\varphi)$
- $\mathsf{AT}_\mu(\boxtimes_A\varphi) := \Diamond(x_A \wedge \Box\mathsf{AT}_\mu(\varphi))$
- $\mathsf{AT}_\mu(\boxtimes_B\varphi) := \Diamond(x_B \wedge \Box\mathsf{AT}_\mu(\varphi))$
- $\mathsf{AT}_\mu(\mu x.\varphi) := \mu x.\mathsf{AT}_\mu(\varphi)$
- $\mathsf{AT}_\mu(\nu x.\varphi) := \nu x.\mathsf{AT}_\mu(\varphi)$

Note that according to the definition of the associated Kripke structure $\mathcal{K}_\mathcal{G}$, we have $[\![x_N]\!]_{\mathcal{K}_\mathcal{G}} = \mathcal{S} \times \{\mathsf{nop}\} \times \{0\}$, i.e., the choice states. Moreover, $[\![x_N]\!]_{\mathcal{K}_\mathcal{G}} \cap [\![x_A \vee x_B]\!]_{\mathcal{K}_\mathcal{G}} = \{\}$, 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 $\neg(x_A \vee x_B)$ instead of x_N in the above translation.

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

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

Hence, we can reduce the computation of $[\![\varphi]\!]_\mathcal{G}$ to the computation of $[\![\mathsf{AT}_\mu(\varphi)]\!]_{\mathcal{K}_\mathcal{G}}$, which can be solved by ordinary μ-calculus model checking.

Concerning the complexity, we note that every alternating time μ-calculus formula is translated to an equivalent μ-calculus formula of the same alternation depth and a size which is linear in the size of the original formula. The number of transitions of $\mathcal{K}_\mathcal{G}$ is at most multiplied by four, and the number of states of $\mathcal{K}_\mathcal{G}$ is $O(|\mathcal{S}||\mathcal{A}|)$. It is well-known that model checking of μ-calculus formulas φ of alternation depth ℓ and *length* $|\varphi|$ can be done on every Kripke structure $\mathcal{K} = (\mathcal{I}, \mathcal{S}, \mathcal{R}, \mathcal{L})$ in time $O\left(\left(\frac{|\varphi||\mathcal{S}|}{\ell}\right)^{\ell-1} |\mathcal{R}|\,|\varphi|\right)$ (see [Schneider, 2003]). We immediately obtain the following result:

Theorem 2: For every $\mathcal{L}_{AT}$-formula φ of alternation depth ℓ and length $|\varphi|$, and every concurrent game $\mathcal{G} = (\mathcal{I}, \mathcal{S}, \delta, \Gamma_A, \Gamma_B, \mathcal{L})$ there is an algorithm to compute the satisfying states $[\![\varphi]\!]_\mathcal{G}$ in time

$$O\left(\left(\frac{|\varphi|\,|\mathcal{S}|\,|\mathcal{A}|}{\ell}\right)^{\ell-1} |\delta|\,|\varphi|\right).$$

In [Alur et al., 2002] an algorithm is presented that modified an existing μ-calculus model-checking algorithm to be suited for alternating time μ-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 $O\left((|\varphi|\,|\delta|)^{\ell+1}\right)$ given in [Alur et al., 2002] is due to our use of improved algorithms for μ-calculus model checking. Nevertheless, it shows that our reduction is efficient, i.e., there is no loss of efficiency due to the reduction.

IV. SUPERVISORY CONTROL

In this section, we show how to solve the supervisory control problem stated in [Ramadge and Wonham, 1987] with the so far developed translations. In supervisory control, a finite state automaton is considered whose input set Σ is partitioned into a set of controllable inputs Σ_c that can be prevented from occurring by a supervisor and the uncontrollable inputs Σ_u that can not be influenced by the supervisor. The supervisory control problem of [Ramadge and Wonham, 1987] considers two additional state sets: the *bad states* Q_b and the *marked states* Q_m. It is the task of the supervisor to find the least restriction of controllable inputs for every state such that (1) in every remaining reachable state, *it is possible to reach a marked state in* Q_m[1], and (2) that no bad state of Q_b is reachable.

To model the problem with a game structure, we assume that we are given a finite state machine $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, Q_m, Q_b\rangle$ with states Q, inputs $\Sigma = \Sigma_c \cup \Sigma_u$, a partial transition function $\delta : Q \times \Sigma \to Q$, an initial state $q_0 \in Q$, and bad and marked state sets $Q_b, Q_m \subseteq Q$. In [Ziller and Schneider, 2003], [Ziller and Schneider, 2005], this supervisory control problem is solved by constructing the following associated Kripke structure of an automaton:

Definition 9 (Kripke Structure of an Automaton): Given an automaton $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, Q_m, Q_b\rangle$, we define its associated Kripke structure $\mathcal{K}_\mathcal{A} = (\mathcal{I}, \mathcal{S}, \mathcal{R}, \mathcal{L})$ over the Boolean variables $\mathcal{V}_\mathcal{A} := \{x_q \mid q \in Q\} \cup \{x_b, x_m, x_u\}$ as follows:

- $\mathcal{S} := Q \times \{0, 1\}$
- $\mathcal{I} := \{(q_0, 0), (q_0, 1)\}$
- $\mathcal{R}((q, 0), (q', 0)) :\Leftrightarrow \exists\sigma \in \Sigma_u.q' = \delta(q, \sigma)$
- $\mathcal{R}((q, 1), (q', 1)) :\Leftrightarrow \exists\sigma \in \Sigma.q' = \delta(q, \sigma)$
- $\mathcal{L}((q, 0)) := \{x_q, x_u\} \cup \left\{ \begin{array}{ll} \{x_b\} & \text{if } q \in Q_b \\ \{\} & \text{if } q \notin Q_b \end{array} \right.$
- $\mathcal{L}((q, 1)) := \{x_q\} \cup \left\{ \begin{array}{ll} \{x_m\} & \text{if } q \in Q_m \\ \{\} & \text{if } q \notin Q_m. \end{array} \right.$

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

We will mimic the behavior of this Kripke structure by a concurrent game structure as follows: the concurrent game $\mathcal{G}_\mathcal{M} = (\mathcal{I}, \mathcal{S}, \delta_\mathcal{G}, \Gamma_C, \Gamma_U, \mathcal{L}_\mathcal{G})$ is defined over the set of actions[2] $\mathcal{A} = \{\mathsf{accept}, \mathsf{reject}\} \cup \Sigma_c \cup \Sigma_u$ and is played by two players C and E (the controller and the environment). The game structure $\mathcal{G}_\mathcal{M}$ inherits its states and transitions directly from the automaton $\mathcal{M}$, i.e., we have $\mathcal{S} = Q$ and state pairs (s, s') are connected by a transition in $\mathcal{G}_\mathcal{M}$ iff these states are connected by a transition in $\mathcal{M}$.

In every state s, player E (the environment) selects one of the possible inputs $\sigma \in \Sigma_c \cup \Sigma_u$ that the automaton can accept in this state. If the input $\sigma \in \Sigma_c$ is controllable and leads to state s' in the automaton, then we add a corresponding transition from s to s' labeled with $(\mathsf{accept}, \sigma)$ to $\mathcal{G}_\mathcal{M}$. As player C has the choice to select either action accept or reject, it can disable this input, so that the transition will not take place. On the other hand, if the input $\sigma \in \Sigma_u$ is not controllable and leads to state s' in the automaton, then we add two corresponding transitions from s to s', one labeled with $(\mathsf{accept}, \sigma)$ and the other one labeled with $(\mathsf{reject}, \sigma)$. Hence, no matter whether player C chooses action accept or reject, it can not disable such a transition. The formal definition of the game structure $\mathcal{G}_\mathcal{M} = (\mathcal{I}, \mathcal{S}, \delta_\mathcal{G}, \Gamma_C, \Gamma_U, \mathcal{L}_\mathcal{G})$ is therefore as follows:

- $\mathcal{I} = \{q_0\}$
- $\mathcal{S} = Q$
- $\Gamma_C(s) = \{\mathsf{accept}, \mathsf{reject}\}$
- $\Gamma_U(s) = \{\sigma \in \Sigma_c \cup \Sigma_u \mid \exists s'.\ s' = \delta(s, \sigma)\}$
- $s' = \delta_\mathcal{G}(s, \alpha, \sigma)$
  $$:\Leftrightarrow \left(\begin{array}{l} \alpha \in \{\mathsf{accept}, \mathsf{reject}\} \wedge \sigma \in \Sigma_u \wedge s' = \delta(s, \sigma) \\ \vee \\ \alpha \in \{\mathsf{accept}\} \wedge \sigma \in \Sigma_c \wedge s' = \delta(s, \sigma) \end{array} \right)$$
- $\mathcal{L}(s) = \begin{cases} \{x_b, x_m\} & \text{if } s \in Q_m \cap Q_b \\ \{x_b\} & \text{if } s \in Q_b \\ \{x_m\} & \text{if } s \in Q_m \\ \{\} & \text{otherwise} \end{cases}$

Hence, for every uncontrollable transition $s \xrightarrow{\sigma} s'$ with $\sigma \in \Sigma_u$, there are two transitions $s \xrightarrow{(\mathsf{accept}, \sigma)} s'$ and $s \xrightarrow{(\mathsf{reject}, \sigma)} s'$ in the game structure $\mathcal{G}_\mathcal{M}$, and for every controllable transition $s \xrightarrow{\sigma} s'$ with $\sigma \in \Sigma_c$, there is only one transition $s \xrightarrow{(\mathsf{accept}, \sigma)} s'$. The latter means that the controller C has no possibility to either disable or enable the transition, while in the former case, the transition can be taken for input σ, regardless of the choice of the controller. Thus, the definition of the transition relation $\delta_\mathcal{G}$ ensures that the controllability property is captured in our game structure: an uncontrollable input may not be prevented by player C, while every controllable input may be prevented (disabled) by player C.

To finally solve the supervisory control problem, we can proceed similarly to [Ziller and Schneider, 2003]: A state does not violate the specification, if it is

- Co-reachable, i.e., a marked state can be reached and
- good, i.e., no bad states can be reached via an uncontrollable input.

[1]Note, that it is not required that every path reaches a marked state. Instead, every state must have at least one path to reach a marked state.

[2]In contrast to the construction given in [Ziller and Schneider, 2003], [Ziller and Schneider, 2005], we retain the inputs $\sigma \in \Sigma$ as labels on the transitions in order to not only check whether there is a solution, but also to derive a suitable solution.

Therefore, in [Ziller and Schneider, 2003], [Ziller and Schneider, 2005], an equation system[3] is given that works on the above defined Kripke structure:

$$\begin{cases} x_{Co} & \overset{\mu}{=} & \kappa(x_G) \wedge (\Diamond x_{Co} \vee x_m) \\ x_G & \overset{\nu}{=} & x_u \wedge \Box(x_G \wedge \kappa(x_{Co})) \wedge \neg x_b. \end{cases}$$

This is an equation system of the ordinary μ-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 [Ziller and Schneider, 2003], [Ziller and Schneider, 2005].

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

$$x'_{Co} \overset{\mu}{=} x'_G \wedge (\Diamond x'_{Co} \vee x_m),$$

where x'_{Co} denotes the coreachable states and x'_G denotes the good states. For calculating the bad states, we note that a state is bad, if an uncontrollable input leads to a state which is known to be bad or not coreachable. Therefore a state is bad, if player E can enforce a transition to an already known bad state, or to a state which is known to be not coreachable. This is calculated by the following formula:

$$x'_B \overset{\mu}{=} \boxed{\Diamond}_E(x'_B \vee \neg x'_{Co}) \vee x_b.$$

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

$$x'_G \overset{\nu}{=} \neg\boxed{\Diamond}_E(\neg x'_G \vee \neg x'_{Co}) \wedge \neg x_b.$$

We therefore get the following equation system:

$$E' = \begin{cases} x'_{Co} \overset{\mu}{=} x'_G \wedge (\Diamond x'_{Co} \vee x_m) \\ x'_G \overset{\nu}{=} \neg\boxed{\Diamond}_E(\neg x'_G \vee \neg x'_{Co}) \wedge \neg x_b. \end{cases}$$

The above equation system is very similar to the one that has been presented in [Ziller and Schneider, 2003], [Ziller and Schneider, 2005]. A solution to the supervisory control problem may be given by evaluating the above equation system which is a standard problem, once we translate it further to a μ-calculus model checking problem. To show the correctness of the presented algorithm, we will give a reduction to the equation system formulated in [Ziller and Schneider, 2003], [Ziller and Schneider, 2005]:

Theorem 3: Given an automaton $\mathcal{M} = \langle Q, \Sigma, \delta, q_0, Q_m, Q_b \rangle$, its associated Kripke structure $\mathcal{K}_\mathcal{M}$, and its associated game $\mathcal{G}_\mathcal{M}$, the following holds:

$$[\![x_{Co}]\!]_{\mathcal{K}_\mathcal{M}} = [\![x'_{Co}]\!]_{\mathcal{G}_\mathcal{M}} \times \{1\}$$

[3]Equation systems with minimality and maximality constraints are an equivalent formulation of the μ-calculus [Schneider, 2003].

V. Conclusions

In this paper, we described a reduction from alternating time μ-calculus model-checking problems (on concurrent game structures) to equivalent propositional μ-calculus model-checking problems (on Kripke structures). The purpose of this reduction is that already existing model checking tools can be used to solve control problems that are formalized by the alternating time μ-calculus. In addition, we presented a reduction of the supervisory control problem of [Ramadge and Wonham, 1987] to an equivalent alternating time μ-calculus model-checking problem. Using the results given in this paper, we can further translate the problem to an ordinary μ-calculus model-checking problem. This allows us to use μ-calculus model checking algorithms to solve the supervisory control problems like the classical one introduced by Ramadge and Wonham.

References

[Accellera, 2004] Accellera (2004). Property specification language reference manual, version 1.1. http://www.eda.org.

[Alfaro et al., 2001] Alfaro, L., Henzinger, T., and Majumdar, R. (2001). From verification to control: Dynamic programs for omega-regular objectives. In *Symposium on Logic in Computer Science (LICS)*, pages 279–290. IEEE Computer Society.

[Alur et al., 2002] Alur, R., Henzinger, T., and Kupferman, O. (2002). Alternating-time temporal logic. *Journal of the ACM*, 49(5):672–713.

[Beer et al., 2001] Beer, I., Ben-David, S., Eisner, C., Fisman, D., Gringauze, A., and Rodeh, Y. (2001). The temporal logic Sugar. In *Conference on Computer Aided Verification (CAV)*, volume 2102 of *LNCS*, pages 363–367, Paris, France. Springer.

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

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

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

[Kupferman and Vardi, 1996] Kupferman, O. and Vardi, M. (1996). Module checking. In Alur, R. and Henzinger, T., editors, *Conference on Computer Aided Verification (CAV)*, volume 1102 of *LNCS*, pages 75–86, New Brunswick, NJ, USA. Springer.

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

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

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

[Thomas, 1990] Thomas, W. (1990). *Automata on Infinite Objects*, volume B, chapter Automata on Infinite Objects, pages 133–191. Elsevier.

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

[Ziller and Schneider, 2005] Ziller, R. and Schneider, K. (2005). Combining supervisor synthesis and model checking. *Transactions on Embedded Computing Systems*, 4(2):331–362.