Bridging the Gap between Supervisory Control and Reactive Synthesis: Case of Full Observation and Centralized Control ⋆

Rüdiger Ehlers * **Stéphane Lafortune** ** **Stavros Tripakis** ***
Moshe Vardi ****

* *University of Bremen*
** *University of Michigan*
*** *University of California, Berkeley and Aalto University*
**** *Rice University*

Abstract: We present a formal connection between supervisory control theory in the field of control engineering and reactive synthesis in the field of formal methods. We focus on the case of fully-observed discrete-event systems that are controlled by a single controller/supervisor in order to achieve a safety specification and a non-blocking specification. The connection is shown by a reduction of the corresponding supervisory control problem to a problem of reactive synthesis with plants and maximal permissiveness, subject to a CTL temporal logic specification. In order to establish the desired reduction, we prove two new results regarding (i) a simplified version of the standard supervisory control problem and (ii) a class of reactive synthesis problems that admit unique maximally permissive solutions. The reduction complements prior work at the boundary of supervisory control and reactive synthesis.

1. INTRODUCTION

We present a formal connection between synthesis problems that have been considered, largely separately, in the two communities of control science in engineering and formal methods in computer science. By making this connection mathematically precise, we hope to "bridge the gap" between two research areas that aim at tackling similar synthesis problems for discrete event systems, but from different angles, and by emphasizing different, and often complementary, aspects. Such a formal bridge should be a source of inspiration for new lines of investigation that will leverage the power of the synthesis techniques that have been developed in these two areas.

Supervisory Control: The control science and engineering community has been investigating feedback control of Discrete Event Systems (DES) using models from computer science, such as automata and Petri nets. The body of control theory developed in DES has been for specifications that are expressible as regular languages, in the case of DES modeled by automata, or in terms of constraints on the state (marking vector), in the case of DES modeled by Petri nets. Control-theoretic frameworks have been developed for both of these modeling formalisms; cf. Seatzu et al. [2013]. In this paper, we focus on the supervisory control theory for DES modeled by finite-state automata and subject to regular language specifications. Both the "plant" (i.e., uncontrolled system) and the specification are represented as finite-state automata over a common event set. The foundations for this framework were developed in the seminal work of Ramadge and Wonham [1987]. Since then, a whole body of theory has been developed that covers a wide variety of control architectures and information structures, with vertical and horizontal modularity. The reader is referred to Cassandras and Lafortune [2008] and Wonham [2013] for textbook expositions of this theory. The focus of this theory is on the synthesis of provably safe and non-blocking controllers for a given plant, despite limited actuation and limited sensing capabilities.

Reactive Synthesis: The design of *reactive systems*, i.e., systems that engage in an ongoing interaction with their environment, is one of the most challenging problems in computer science. In reactive systems, a correct system should satisfy the specification with respect to *all* environment behaviors. The specification is usually expressed in a temporal logic. Pnueli and Rosner [1989b], Abadi et al. [1989], and Dill [1989] argued that the right way to approach synthesis of reactive systems is to use the model of a, possibly infinite, game between the environment and the system. A correct system can be then viewed as a winning strategy in this game. It turns out that satisfiability of the specification is not sufficient to guarantee the existence of such a strategy. Abadi et al. [1989] called specifications for which a winning strategy exists *realizable*. Since then, the subject of *reactive synthesis* has been an active area of research, attracting considerable attention; see, e.g., Pnueli and Rosner [1989a], Vardi [1995] and Kupferman and Vardi [2000].

⋆ This work was supported by the University of California at Berkeley, the University of Michigan, Rice University, Aalto University, the Academy of Finland, and the US National Science Foundation, via projects *ExCAPE: Expeditions in Computer Augmented Program Engineering* and *COSMOI: Compositional System Modeling with Interfaces*. This work was also supported in part by the iCyPhy Research Center (Industrial Cyber-Physical Systems, supported by IBM and United Technologies), and the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley (supported by NSF, NRL, and the following companies: Bosch, National Instruments, and Toyota).

Related Works: This paper is not the first to explore connections between supervisory control and reactive synthesis. On the supervisory control side, several authors have considered control of DES subject to temporal logic specifications; see, e.g., Thistle and Wonham [1986], Lin [1993], and Jiang and Kumar [2006]. Supervisory control of DES with *infinite* behavior has also been considered by many researchers; see, e.g., Ramadge [1989], Kumar et al. [1992], Thistle and Wonham [1994a], and Thistle and Wonham [1994b]. On the other hand, several researchers in the formal methods community have investigated supervisory control of fully- and partially-observed DES; see, e.g., Hoffmann and Wong Toi [1992], Asarin et al. [1995], Madhusudan [2001], Kupferman et al. [2000], Arnold et al. [2003], and Riedweg and Pinchinat [2003].

Contribution: In the present paper, we restrict attention to the classical version of centralized supervisory control for fully-observed systems modeled by languages of finite strings. Our goal is to establish a precise connection of supervisory control problems with problems of reactive synthesis, by showing how specific problem instances reduce to each other. To our knowledge, such reductions have not been published elsewhere. Our results therefore complement the existing work. We start in Section 2 by briefly presenting necessary background material from supervisory control and reactive synthesis. The main results of this paper are contained in Section 3. First, we present in Section 3.1 a simplification of the basic supervisory control problem, non-blocking version, to one where the safety specification has been absorbed into the plant model. We then show that the resulting Simple Supervisory Control Problem (SSCP) has a state-based solution. Second, for bridging reactive synthesis with supervisory control, we need two technical steps: the first step is to consider reactive synthesis with plants; the second step is to bring in the issue of maximal permissiveness into this reactive synthesis setting. These two steps are covered in Section 3.2. We then establish the formal reduction from SSCP to a reactive synthesis problem with plants and maximal permissiveness in Section 3.3. A discussion and some concluding comments follow in Sections 3.4 and 4.

Due to space limitations, we do not review temporal logics such as LTL, CTL, and CTL*. Moreover, our presentation excludes proofs and has few examples. Our focus is on presenting the necessary concepts for our main results, Theorem 4 and Corollary 2. We refer the reader to Ehlers et al. [2013] for a detailed treatment of our results.

2. BACKGROUND

Supervisory Control: In supervisory control theory, plants are typically modeled as deterministic finite-state automata. A deterministic finite-state automaton (DFA) is a 5-tuple $G = (X, x_0, X_m, E, \delta)$ where

- X is a finite set of states, $x_0 \in X$ is the initial state, and $X_m \subseteq X$ is the set of *marked* states;
- E is a finite set of events. E is (implicitly) partitioned into two disjoint subsets: $E = E_c \cup E_{uc}$ where E_c models the set of *controllable* events and E_{uc} the set of *uncontrollable* events.
- $\delta : X \times E \to X$ is the transition function, which in general will be partial.

The transition function is partial because G models the physically possible behavior of the uncontrolled plant, as a generator of events. Selection of the states to "mark," i.e., to be included in X_m, is a modeling consideration to capture strings that represent that the system has completed some task. It is customary to extend δ to strings. The DES G defines the following languages: $\mathcal{L}(G) = \{\sigma \in E^* \mid \delta(x_0, \sigma) \text{ is defined}\}$ and $\mathcal{L}_m(G) = \{\sigma \in E^* \mid \delta(x_0, \sigma) \in X_m\}$.

A *supervisor* for G is a function $S : E^* \to 2^E$. To ensure that S never disables an uncontrollable event, we require that $E_{uc} \subseteq S(\sigma)$ for all $\sigma \in E^*$. Given $G = (X, x_0, X_m, E, \delta)$ and $S : E^* \to 2^E$ for G, the *closed-loop system* S/G is formally defined as follows: $S/G = (X', x_0', X_m', E, \delta')$ where

- $X' = X \times \mathcal{L}(G)$
- $x_0' = (x_0, \varepsilon)$
- $X_m' = X_m \times \mathcal{L}(G)$
- $\delta'\big((x,\sigma), e\big) = \begin{cases} (\delta(x,e), \sigma e) & \text{if } \delta(x,e) \text{ is defined} \\ & \text{and } e \in S(\sigma) \\ \text{undefined} & \text{otherwise.} \end{cases}$

S/G is an automaton, therefore, languages $\mathcal{L}(S/G)$ and $\mathcal{L}_m(S/G)$ are well defined. It is easy to verify that $\mathcal{L}_m(S/G) = \mathcal{L}(S/G) \cap \mathcal{L}_m(G)$ since a marking in S/G is completely determined by a marking in G. S is said to be *non-blocking for* G iff $\overline{\mathcal{L}_m(S/G)} = \mathcal{L}(S/G)$ where the overline notation denotes prefix-closure.

Consider a plant G and two supervisors S_1, S_2 for G. We say that S_1 *is no more permissive than* S_2 iff $S_1(\sigma) \subseteq S_2(\sigma)$ for any σ. We say that S_2 *is strictly more permissive than* S_1 iff S_1 is no more permissive than S_2 and $S_1 \neq S_2$.

S is needed in order to enforce the *safety specification* imposed on G. In supervisory control, the safety specification is modeled by a prefix-closed regular language, denoted by L_a, over the event set E of G. L_a is prefix-closed since for a string to be safe, all of its prefixes should also be safe. In this paper, we define the *admissible marked language* L_{am} for plant G as $L_{am} := L_a \cap \mathcal{L}_m(G)$. S is said to be *safe for* G *with respect to* L_{am} if $\mathcal{L}(S/G) \subseteq \overline{L_{am}}$. A supervisor S which is non-blocking for G and safe w.r.t. L_{am} is said to be *maximally-permissive with respect to* G *and* L_{am} if there is no supervisor S' which is non-blocking for G, safe w.r.t. L_{am}, and strictly more permissive than S. The theorem below shows that, for non-blockingness and safety, a *unique* maximally-permissive supervisor exists, provided that a supervisor exists at all. This well-known result from Ramadge and Wonham [1987] motivates the definition of BSCP-NB that follows.

Theorem 1. Consider G and L_{am} as defined above. If there exists a supervisor which is non-blocking for G and safe w.r.t. L_{am}, then there exists a unique maximally-permissive supervisor S_{mpnb} which is non-blocking for G and safe w.r.t. L_{am}.

Definition 1. (BSCP-NB). Given G and L_{am} as defined above, find if it exists, or state that there does not exist, a supervisor for G which is non-blocking for G, safe w.r.t. L_{am}, and maximally-permissive.

Reactive Synthesis: In reactive synthesis, we build correct-by-construction "controllers" (or system imple-

mentations) from declarative specifications. Controllers are *open dynamical systems*. A controller is open in the sense that it has inputs and outputs, and its dynamics depend on the inputs that it receives. These inputs come from the controller's *environment*. A specification is declarative in the sense that it states how a controller must behave, but it is not concerned with its internal structure. Rather, the specification only describes the desired behavior of the controller on the interface level, i.e., using its sets of inputs and outputs. To perform synthesis from this specification, we need to formalize it. In reactive synthesis, this is typically done by describing the specification in a *logic*. The logic CTL* [Emerson and Halpern, 1986] is well-suited for this purpose and extends standard Boolean logic by *temporal operators* and *path quantifiers* that intuitively allow us to connect the system's signal valuations in one step with the actions in other, future time steps. In the context of logic, we also call the signals *atomic propositions*. LTL and CTL are proper subsets of CTL*. In this paper, we will consider a single specific CTL formula to capture non-blockingness.

Informally, the basic Reactive Synthesis Problem (or RSP) is to synthesize a controller that provably satisfies a given temporal logic specification (in some temporal logic), for all realizations of the environment variables, which come as inputs to the controller. (We formally define Reactive Synthesis Control Problem *with plants* (RSCP) in Section 3.2.) To get an overview about the possible behaviors of a system, for the scope of synthesis, we typically view a system implementation as a *computation tree* denoted by the tuple $\langle T, \tau \rangle$. Let AP_I be the set of input signals (events) and AP_O be the set of output signals (events). Formally, for some interface $(\mathsf{AP}_I, \mathsf{AP}_O)$ of a reactive system, a computation tree is a tuple $\langle T, \tau \rangle$, where $T = (2^{\mathsf{AP}_I})^*$ and $\tau : T \to 2^{\mathsf{AP}_I \cup \mathsf{AP}_O}$. The tree describes all the possible traces by having τ map every input sequence to an output signal valuation that the controller produces after having read the input sequence. Without loss of generality, we assume that every node in the computation tree is also labeled by the last input.

3. BRIDGING THE GAP

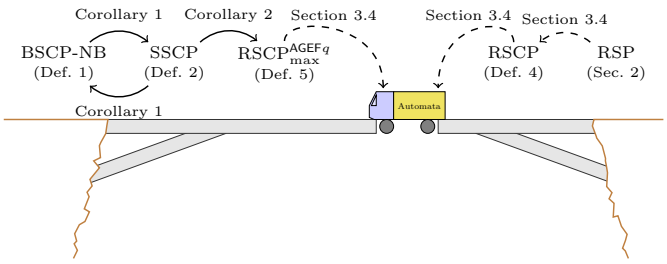

Fig. 1. Relations between different synthesis and supervisory control problems.

Figure 1 describes our process for bridging the gap between supervisory control theory and reactive synthesis. We introduce problems that conceptually link BSCP-NB and RSP. These problems always differ in one aspect from their neighbors, and we can perform reductions between them. However, our bridge does not exactly meet in the middle. The reason is that the aims of supervisory control and reactive synthesis slightly differ. In supervisory control, we always want our supervisor to be maximally permissive, as it should only block unwanted actions. In reactive synthesis, on the other hand, where maximal permissiveness in unachievable in general, we want our controller to actively enforce certain properties, possibly at the expense of preventing certain overall system behavior that is unproblematic. This mismatch, and the lack of study of the general reactive synthesis problem with maximal permissiveness, prevent us from performing a sequence of reductions that map the problems completely onto each other. The problems that are closest to the missing piece of our bridge can be solved algorithmically using automata; this is captured conceptually in Figure 1 by the "vehicular" connection between the two sides of the bridge.

3.1 Simplifying the Supervisory Control Problem

We show that BSCP-NB is equivalent to a simpler supervisory control problem in which only non-blockingness is required, called SSCP. We then show that SSCP admits state-based solutions in terms of the plant model. The consideration of SSCP will simplify the reduction to the reactive synthesis setting.

Theorem 2. Consider G and L_{am} as defined previously. Let A be a *complete* DFA such that $\mathcal{L}(A) = E^*$ and $\mathcal{L}_m(A) = L_{am}$. Let S be a supervisor for G, and therefore also for $G \times A$. Then, the following two statements are equivalent:
(i) S solves BSCP-NB for plant G with respect to L_{am}.
(ii) S solves BSCP-NB for plant $G \times A$ with respect to $\mathcal{L}_m(G \times A)$.

Definition 2. (SSCP). Given G, find (if it exists, or state that none exists) a maximally-permissive non-blocking supervisor for G.

Corollary 1. BSCP-NB and SSCP are equivalent problems: each one can be reduced to the other with a polynomial-time reduction.

Definition 3. S is said to be *state-based* if $\forall \sigma_1, \sigma_2 \in E^* : \delta(x_0, \sigma_1) = \delta(x_0, \sigma_2) \Rightarrow S(\sigma_1) = S(\sigma_2)$.

Theorem 3. The solution to SSCP, if it exists, is a state-based supervisor.

3.2 Reactive Synthesis With Plants

Most classical reactive synthesis frameworks do not have a notion of a plant. An exception to the above is the work by Madhusudan [2001], where the *control problem for non-reactive environments* is defined as the problem of synthesizing a controller for a given plant modeled as a finite-state Kripke structure, so that the closed-loop system satisfies a specification in CTL or CTL*. As done by Madhusudan [2001], a plant can be captured as a *transition system*, specifically a form of *Kripke structure*: $P = (W, w_0, R, AP, L)$ where

- AP is a set of atomic propositions.
- W is a set of states with w_0 the initial state. W is (implicitly) partitioned into two disjoint subsets $W = W_s \cup W_e$: W_s models the system states (where the system must choose a move) and W_e models the environment states (where the environment must choose a move).
- $R \subseteq W \times W$ is the transition relation.

- $L : W \to 2^{AP}$ is a total labeling function mapping every state w to a set of propositions true in this state.

We assume that R is total. We define $succ_P(w) = \{w' \mid (w, w') \in R\}$. A Kripke structure plant is called finite when its set of states is finite.

The logic CTL* mentioned earlier is useful for specifying control objectives in plants. In this context, we evaluate the CTL* formula on the tree that is *induced* by the Kripke structure P. We say that P induces a computation tree $\langle T, \tau \rangle$ if the following conditions hold:

- $T \subseteq W^*$, $\epsilon \in T$
- $\{t \in T : |t| = 1\} = \{w \in W : (w_0, w) \in R\}$
- $\tau(\epsilon) = L(w_0)$
- For all $t = t_0 t_1 \dots t_n \in T$, the set of $t's$ children is precisely $\{t_0 t_1 \dots t_n t_{n+1} \mid t_{n+1} \in W, (t_n, t_{n+1}) \in R\}$
- For all $t = t_0 t_1 \dots t_n \in T$, we have $\tau(t) = L(t_n)$.

In a nutshell, the computation tree that is induced by a Kripke structure represents all possible paths in the Kripke structure at the same time. A path of P is an infinite sequence $\pi = w_0 w_1 \cdots$, such that $w_i \in W$ and $(w_i, w_{i+1}) \in R$, for all $i \geq 0$. Given some CTL* state formula ϕ, we say that some state $w \in W$ satisfies ϕ if the computation tree for the Kripke structure $P_w = (W, w, R, AP, L)$ that only differs from P by its initial state, satisfies ϕ. We say that a plant satisfies a CTL* state formula ϕ, written formally as $P \models \phi$, if the tree induced by P satisfies ϕ.

A plant P may not generally satisfy a CTL* specification ϕ. A strategy aims to restrict P so that it satisfies ϕ. Formally, a *strategy* for P is a (total) function $f : W^* \times W_s \to 2^W$ such that for all $u \in W^*, w \in W_s$, $f(u, w)$ is a non-empty subset of $succ(w)$. The intuition is that f observes the history of all states visited previously, $u \in W^*$, as well as the current system state $w \in W_s$, and chooses to allow moves to only a subset (but a non-empty subset) of the successors of w. A strategy f is *state-based* if for all $u_1, u_2 \in W^*$, and for all $w \in W_s$, we have $f(u_1, w) = f(u_2, w)$. This means that f only depends on the current state w and not on the previous history u.

A strategy f defines a new (infinite-state) Kripke structure P^f: $P^f = (W^f, w_0^f, R^f, AP, L^f)$ where

- $W^f = W^* \times W$
- $w_0^f = (\epsilon, w_0)$
- $R^f = \{\big((u, w), (u \cdot w, w')\big) \mid \big(w \in W_e \wedge (w, w') \in R\big) \vee \big(w \in W_s \wedge w' \in f(u, w)\big)\}$
- $L^f(u, w) = L(w)$ for all $u \in W^*$, $w \in W$.

Note that R^f is guaranteed to be total. This is because R is assumed to be total, and f is required to be such that $f(u, w) \neq \emptyset$.

Given Kripke structure plant P and CTL* formula ϕ, we say that a strategy f *enforces* ϕ *on* P if it is the case that $P^f \models \phi$. The *Reactive Synthesis Control Problem with plants* (RSCP) is the following:

Definition 4. (RSCP). Given finite Kripke structure plant P and CTL* formula ϕ, find (if it exists, or state that there does not exist) a strategy that enforces ϕ on P.

RSCP-CTL denotes RSCP where ϕ is required to be a CTL formula. (And similarly for other temporal logics.)

The definition of RSCP does not require that the strategy f be maximally-permissive in any way. The reason is that unique maximally-permissive strategies do not always exist. Let f_1, f_2 be two strategies for P. f_1 is said to be *no more permissive than* f_2 iff for all $u \in W^*$, $w \in W_s$ such that uw is a sequence of states that can be a prefix of a run in P^{f_2}, $f_1(u, w) \subseteq f_2(u, w)$. f_2 is said to be *strictly more permissive than* f_1 if f_1 is no more permissive than f_2 and $f_1(u, w) \neq f_2(u, w)$ for some $u \in W^*$, $w \in W_s$ such that uw is a sequence of states that can be a prefix of a run in P^{f_2}. f_1 is said to be *maximally permissive* with respect to specification ϕ if f_1 enforces ϕ and there is no strategy f_2 which enforces ϕ and is strictly more permissive than f_1. The next result characterizes one class of specifications that admit unique maximally-permissive strategies, provided one strategy exists. To the best of our knowledge, no such characterizations have been established before in the reactive synthesis literature.

Proposition 1. Let P be a Kripke structure and q be a CTL state formula without temporal operators. If there exists a strategy enforcing $\mathsf{AGEF}q$ on P, then there exists a unique, maximally-permissive, state-based strategy enforcing $\mathsf{AGEF}q$ on P.

Exploiting Proposition 1, we define the new problem $\text{RSCP}_{\max}^{\mathsf{AGEF}q}$, which is a variant of RSCP-CTL. In $\text{RSCP}_{\max}^{\mathsf{AGEF}q}$, the specification is a CTL formula of the form $\mathsf{AGEF}q$ where q is a CTL formula without temporal operators.

Definition 5. ($\text{RSCP}_{\max}^{\mathsf{AGEF}q}$). Given finite Kripke structure plant P and CTL formula $\mathsf{AGEF}q$ where q is a CTL formula without temporal operators, find (if it exists, or state that there does not exist) the unique maximally-permissive state-based strategy that enforces ϕ on P.

3.3 From Supervisory Control to Reactive Synthesis with Plants

We are now ready to show how to reduce SSCP to $\text{RSCP}_{\max}^{\mathsf{AGEF}q}$. Given a DES plant G in the form of a DFA, we first construct a plant P_G in the form of a Kripke structure. A system state of P_G is a state x of G. An environment state of P_G is either of the form (x, c), where $c \in E_c$, or $(x, \perp)$. All successors of system states are environment states, and vice versa. From a system state x, P_G has at most $|E_c|+1$ possible successors, one successor of the form (x, c) for each controllable event c which is enabled at state x in G, plus an extra successor $(x, \perp)$. Intuitively, choosing a subset of the successors of x amounts to allowing a subset of the controllable events enabled at x. If only $(x, \perp)$ is chosen, then all controllable events are disabled and only uncontrollable events (if any) are allowed to occur at x.

From environment state (x, c), P_G has an outgoing transition to a system state x' if either G has an uncontrollable transition from x to x', or G has a transition labeled c from x to x'. That is, the only transitions enabled from (x, c) are uncontrollable transitions or the controllable transition labeled c (there can only be one such transition since G is deterministic). If x has no controllable transition labeled c, then (x, c) is not a successor of x by construction. Therefore, an outgoing transition is guaranteed to exist

from every reachable environment state of the form (x, c) with $c \in E_c$. Finally, from environment state $(x, \bot)$, P_G has an outgoing transition to a system state x' if G has an uncontrollable transition from x to x'. That is, only uncontrollable transitions are allowed from $(x, \bot)$. If x has no outgoing uncontrollable transitions then a transition back to x is added to $(x, \bot)$. These "back-transitions" achieve two goals. First, they prevent deadlocks in P_G. Second, we can prove that non-blocking strategies can always be extended to allow successors of the form $(x, \bot)$; cf. Lemma 5 in Ehlers et al. [2013].

We also need to define the set of atomic propositions and the labeling function of P_G. P_G has a single atomic proposition, *acc*. The states of P_G labeled with *acc* are system states $x \in X_m$ in G, and environment states (x, c) or $(x, \bot)$ where $x \in X_m$ in G.

To express the requirements of SSCP as a temporal logic formula, we use the CTL formula $\phi_{nb} := \mathsf{AG}\,\mathsf{EF}\,acc$. ϕ_{nb} states that it is always possible to reach a marked state from any reachable state. This formula characterizes non-blockingness. Note that non-blockingness cannot be expressed in LTL.

Let $G = (X, x_0, X_m, E, \delta)$ be a DES plant with $E = E_c \cup E_{uc}$. It is convenient to define the functions $\mathsf{En} : X \to 2^E$ with $\mathsf{En}(x) = \{e \mid \delta(x, e) \text{ is defined}\}$, $\mathsf{En}_c : X \to 2^{E_c}$ with $\mathsf{En}_c(x) = \mathsf{En}(x) \cap E_c$, and $\mathsf{En}_u : X \to 2^{E_{uc}}$ with $\mathsf{En}_u(x) = \mathsf{En}(x) \cap E_{uc}$, which return, respectively, the set of all events, controllable events, and uncontrollable events, enabled at state x.

The Kripke structure plant P_G is defined to be $P_G = (W, w_0, R, AP, L)$ such that

- $W = W_s \cup W_e$, with $W_s = X$ and $W_e = X \times (E_c \cup \{\bot\})$.
- $w_0 = x_0$. Therefore, w_0 is a system state.
- $R = R_s \cup R_e$, with

$$
\begin{aligned}
R_s = \quad & \{\big(x, (x, c)\big) \mid x \in X, c \in \mathsf{En}_c(x)\} \\
& \cup \{\big(x, (x, \bot)\big) \mid x \in X\} \\
R_e = \quad & \{\big((x, c), x'\big) \mid x, x' \in X, \exists e \in E_{uc} \cup \{c\} : \\
& \delta(x, e) = x'\} \\
& \cup \{\big((x, c), x\big) \mid x \in X, c \in E_c, c \notin \mathsf{En}_c(x)\} \\
& \cup \{\big((x, \bot), x'\big) \mid x, x' \in X, \exists u \in E_{uc} : \delta(x, u) = x'\} \\
& \cup \{\big((x, \bot), x\big) \mid x \in X, \mathsf{En}_u(x) = \emptyset\}
\end{aligned}
$$

- $AP = \{acc\}$.
- $L(s) = \begin{cases} \{acc\} & \text{if } s = x \text{ or } s = (x, \bot) \\ & \text{for some } x \in X_m \\ \{\} & \text{otherwise.} \end{cases}$

The following lemma guarantees that P_G does not have deadlocks, therefore, it is a valid Kripke structure plant.

Lemma 1. The above-defined R of P_G is total.

As an illustrative example, consider plant G_1 of Figure 2, where $E_c = \{c_1, c_2\}$ and $E_{uc} = \{u\}$. Its Kripke structure P_{G_1} is shown in Figure 2. States drawn as circles are system states; states drawn as rectangles are environment states. States with double lines are those labeled with *acc*.

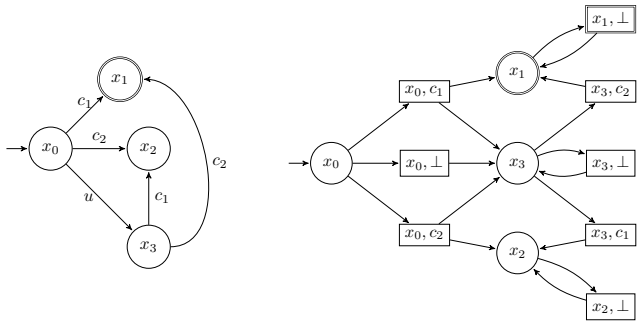

Fig. 2. Plant G_1 (left) and its Kripke structure P_{G_1}

Whenever we are concerned with a state-based strategy f for some set of states W, we simply write $f(w)$ for some $w \in W_s$ to mean the value of $f(\overline{w}, w)$ for any $\overline{w} \in W^*$. Since for state-based strategies the value of $\overline{w}$ does not make a difference, $f(w)$ is uniquely defined. We can now state our main reduction results.

Theorem 4. Let $G = (X, x_0, X_m, E, \delta)$ be a DES plant and $P_G = (W, w_0, R, AP, L)$ be a Kripke structure built from G by the above construction.

(1) Given a non-blocking maximally-permissive state-based supervisor S for G, we can compute a maximally-permissive state-based strategy f_S enforcing $\mathsf{AGEF}\,acc$ on P_G as follows: For all $w \in W_s$,
$$f_S(w) = \{(w, c) \mid c \in S(w) \cap E_c\} \cup \{(w, \bot)\}.$$
(2) Given a maximally-permissive state-based strategy f enforcing $\mathsf{AGEF}\,acc$ on P_G, we can compute a non-blocking state-based maximally-permissive supervisor S_f for G as follows: For all $x \in X$,
$$S_f(x) = E_{uc} \cup \{e \in E_c \mid (x, e) \in f(x)\}.$$

Corollary 2. SSCP can be reduced to $\text{RSCP}^{\mathsf{AGEF}q}_{\max}$ with a polynomial-time reduction.

3.4 Discussion

The solution of RSP-LTL has been well studied in the reactive synthesis literature since the publication of the seminal paper [Pnueli and Rosner, 1989b]. On the other hand, techniques for solving RSP-CTL and RSP-CTL* are provided in a number of works, for instance, Kupferman and Vardi [1999] and Madhusudan [2001]. Madhusudan's thesis [Madhusudan, 2001] also provides a method for solving RSCP-CTL* (and thus also RSCP-LTL and RSCP-CTL as special cases) by reducing it to the *module-checking problem* [Kupferman and Vardi, 1996]. In view of Theorem 4, $\text{RSCP}^{\mathsf{AGEF}q}_{\max}$ can be solved using the standard algorithm for supervisory control problem BSCP-NB.

Generally speaking, RSP can be seen as a special case of RSCP, where the plant offers some possible input at every step. (However, some technical details would need to be resolved, as RSP is formulated in terms of inputs and outputs whereas RSCP is formulated in terms of system and environment states.) Conversely, RSP may appear at first sight more restrictive than it really is, as there is no notion of a plant that encodes the possible environment behavior. Yet, we can *encode* the possible plant behavior into the specification. Starting from a specification ϕ, we can modify it to some specification ϕ' such that for satisfying ϕ', the controller computed from

an RSP algorithm has to satisfy ϕ for precisely those input streams that correspond to paths in a given plant. In this way, a strategy for controlling a plant can be obtained by chopping away the irrelevant parts of a computation tree that satisfies ϕ'. However, this approach is not interesting from a practical perspective, especially when a plant is already available in the form of an automaton. In this case, encoding the plant to a temporal logic specification does not make much sense, due to computational complexity reasons. Indeed, for most temporal logics, the reactive synthesis problems for the logic is at least exponential in the length of the formula, so keeping the size of the formula small is essential. This complexity often arises because of the need to translate the formula into some form of automaton during the synthesis algorithm. As plants are naturally described as automata, it is not wise to translate a plant automaton into a plant formula, and then back into an automaton.

4. CONCLUSIONS & PERSPECTIVES

This work is a first step toward bridging the gap between two research fields, and their corresponding communities, that developed over the last three decades independently for a large part, although both target the general problem of controller synthesis. Some of the results presented here may be unsurprising, or even known to some researchers in the field. Still, to our knowledge, no similar written account exists.

A missing aspect from the present work is modeling and evaluation. It would be worthwhile to develop case studies that would allow a detailed comparison of these two frameworks in terms of plant and specification modeling, computational complexity of synthesis, and implementation of derived supervisor/controller. A number of interesting topics are left as part of future work, including extending the bridge to partially-observed systems, to languages other than safety properties, and to distributed and decentralized control settings.

REFERENCES

M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proc. 25th ICALP*, volume 372 of *LNCS*, pages 1–17. Springer, 1989.

A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 303(1):7 – 34, 2003.

E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II*, 1995.

C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, second edition, 2008.

D.L. Dill. *Trace theory for automatic hierarchical verification of speed independent circuits*. MIT Press, 1989.

R. Ehlers, S. Lafortune, S. Tripakis, and M. Vardi. Reactive synthesis vs. supervisory control synthesis: Bridging the gap. Technical report, University of California at Berkeley, October 2013. `http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-162.pdf`.

E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not Never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986.

G. Hoffmann and H. Wong Toi. Symbolic synthesis of supervisory controllers. In *Proc. American Control Conf.*, 1992.

S. Jiang and R. Kumar. Supervisory Control of Discrete Event Systems with CTL* Temporal Logic Specifications. *SIAM J. Contrl. Opt.*, 44(6):2079–2103, 2006.

R. Kumar, V. Garg, and S.I. Marcus. On supervisory control of sequential behaviors. *IEEE Transactions on Automatic Control*, 37(12):1978–1985, 1992.

O. Kupferman and M. Vardi. Module checking. In R. Alur and T. Henzinger, editors, *Computer Aided Verification*, volume 1102 of *LNCS*, pages 75–86. Springer, 1996.

O. Kupferman and M. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2), June 1999.

O. Kupferman and M. Vardi. Synthesis with incomplete information. In *Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, 2000.

O. Kupferman, P. Madhusudan, P. Thiagarajan, and M. Vardi. Open systems in reactive environments: Control and synthesis. In *11th Intl. Conf. on Concurrency Theory*, pages 92–107. Springer, 2000.

F. Lin. Analysis and synthesis of discrete event systems using temporal logic. *Control Theory and Advanced Technologies*, 9:341–350, 1993.

P. Madhusudan. *Control and Synthesis of Open Reactive Systems*. PhD thesis, University of Madras, 2001.

A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *Proc. 25th Int. Colloq. on Automata, Languages, and Programming*, volume 372 of *LNCS*, pages 652–671. Springer, 1989a.

A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *ACM Symp. POPL*, 1989b.

P. Ramadge. Some tractable supervisory control problems for discrete-event systems modeled by Büchi automata. *IEEE Trans. Aut. Contr.*, 34(1):10–19, 1989.

P. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control & Optimization*, 25(1), 1987.

S. Riedweg and S. Pinchinat. Quantified mu-calculus for control synthesis. In *Mathematical Foundations of Computer Science 2003*, volume 2747 of *LNCS*, pages 642–651. Springer, 2003.

C. Seatzu, M. Silva, and J. H. Van Schuppen. *Control of Discrete-Event Systems. Automata and Petri net Perspecitves*, volume 433. Springer London, 2013. doi: 10.1007/978-1-4471-4276-8.

J. Thistle and W.M. Wonham. Control problems in a temporal logic framework. *International Journal of Control*, 44(4):943–976, April 1986.

J. Thistle and W.M. Wonham. Control of infinite behavior of finite automata. *SIAM Journal on Control and Optimization*, 32(4):1075–1097, 1994a.

J. Thistle and W.M. Wonham. Supervision of infinite behavior of discrete-event systems. *SIAM Journal on Control and Optimization*, 32(4):1098–1113, 1994b.

M. Vardi. An automata-theoretic approach to fair realizability and synthesis. In *Proc. 7th Int. Conf. on Computer Aided Verification*, volume 939 of *LNCS*, pages 267–292. Springer, 1995.

W.M. Wonham. Supervisory Control of Discrete-Event Systems. `http://www.control.toronto.edu/people/profs/wonham/wonham.html`, 2013.