PSPACE-completeness of Modular Supervisory Control Problems *

Kurt Rohloff and Stéphane Lafortune
Department of Electrical Engineering and Computer Science
The University of Michigan
1301 Beal Ave., Ann Arbor, MI 48109-2122, USA
{krohloff,stephane}@eecs.umich.edu; www.eecs.umich.edu/umdes

January 3, 2005

Abstract

In this paper we investigate computational issues associated with the supervision of concurrent processes modeled as modular discrete-event systems. Here, modular discrete-event systems are sets of deterministic finite-state automata whose interaction is modeled by the parallel composition operation. Even with such a simple model process model, we show that in general many problems related to the supervision of these systems are PSPACE-complete. This shows that although there may be space-efficient methods for avoiding the state-explosion problem inherent to concurrent processes, there are most likely no time-efficient solutions that would aid in the study of such "large-scale" systems. We show our results using a reduction from a special class of automata intersection problem introduced here where behavior is assumed to be prefix-closed. We find that deciding if there exists a supervisor for a modular system to achieve a global specification is PSPACE-complete. We also show many verification problems for system supervision are PSPACE-complete, even for prefix-closed cases. Supervisor admissibility and online supervision operations are also discussed.

*This research was supported in part by NSF grant CCR-0082784.

1 Introduction

There has been considerable interest lately in both the discrete-event systems community and the computer science community in concepts related to modular systems. Some systems too complicated to model in a monolithic manner may be easier to model as modular interacting subsystems. Manipulating systems as separate interacting agents has the added advantage of avoiding the "state explosion" problem; when several finite-state systems are combined, the size of the state space of the composed system is potentially exponential in the number of components, so we may wish to keep system models modular whenever possible. Likewise, when supervising or verifying system behavior, there may be several separate specifications and therefore it would be advantageous to keep the specifications modular as well. If we try to combine diverse modular specifications of concurrent systems to form a single monolithic specification, that specification may have a state space too large for a reasonable computation device to handle due to a similar state explosion problem. We investigate the computational complexity of decision problems associated with the supervision of modular systems.

Although there are several ways of specifying modular systems, we make the assumption that the modular systems and specifications discussed in this paper are modeled as deterministic finite-state automata interacting via the parallel composition operation. This modeling method is generally considered to be the simplest method that is expressive enough to be used for real-world problems, and decision problems related to finite-state deterministic automata are also thought to be relatively easy in general. We explicitly define the automata models and the composition operation used for our investigations in the next section.

We show that many problems related to the supervision of these simple systems are PSPACE-complete, which implies that similar supervision issues for more general models are likewise intractable. For background information on the theory of computation and properties of class PSPACE, please consult one of the standard textbooks such as [8] or [9]. Although for supervision purposes we find there may not be great savings in time by specifying systems in a modular manner, there is possibly great savings in computation space. This is why researchers began to investigate the use of modules in the first place - to avoid the *state* explosion problem.

There has recently been a large volume of work investigating connections between theoretical computer science and supervisory control of discrete-event systems. Many of the system models

used in supervisory control are based on ideas that originated from computer science, such as finite-state automata and Petri nets. Discrete-event system theory can be used to model a large range of systems such as telecommunication networks, database systems and computer logic units. Some notable examples of crossover work between control theory and theoretical computer science include [1, 2, 4, 13, 26, 30, 32, 35]. We discuss and explore further connections between these fields.

The work in this paper was inspired by the automata intersection problem initially investigated by Kozen [17]. We extend the work of the computer science literature to a special subclass of automata intersection problems involving prefix-closed languages specified by deterministic finite-state automata and show how these problems are relevant to supervision and verification problems. The first main result of this paper is to show that several decision problems related to comparing the prefix-closed behavior of deterministic automata are PSPACE-complete.

The results on the computational difficulty of automata intersection problems are reduced to supervisor existence and verification problems using polynomial-time many-one reductions. We assume the supervision systems in this paper are "parallel" supervision systems that are realized as deterministic finite-state automata where the supervised system is synthesized by a parallel composition of the supervisor automata with the unsupervised system. See [7, 24, 25, 30, 31, 36] for a sample of major innovative works from the control community on parallel supervision discrete-event systems and the text [6] for a general introduction to discrete-event system theory. An event is disabled by a parallel supervisor at a given state if there are no output transitions for that event at that supervisor state. Supervisors can only update on the occurrence of observable events and can only disable controllable events. A supervisor with these properties is called "admissible".

Decentralized supervision problems are also investigated where local supervisors make local observations and generate local control actions that are combined globally. Decentralized supervision systems can be centralized through the use of the parallel composition operation. We assume in this work that no explicit communication occurs between the various modules in the system besides the implicit communication of shared events through the parallel composition operation used to coordinate local behavior. Our results also hold for more general supervision systems as discussed in [43].

Another main result of this paper is that supervisor existence problems for modular systems are in PSPACE when we desire the supervised system to behave exactly as the specification. We use these supervisor existence results to explore deciding supervisor admissibility.

It is also discussed how if given a (possibly decentralized) supervision system, a modular plant and a modular specification, it can be verified that the specification is satisfied in a computationally feasible manner. In general, we find that verification problems for modular discrete-event systems modeled as sets of deterministic finite-state automata are PSPACE-complete, meaning that these problems are probably intractable.

The online supervision of modular discrete-event systems is also investigated. Online supervision methods calculate control actions on the fly as behavior is observed as opposed to the traditional offline methods where control actions are precalculated. Online supervision has been used to avoid computational intractability for the supervision of monolithic discrete-event systems. See [11, 33, 43] for example. We find that online supervision does not help us with modular system problems as with monolithic systems.

There have been several papers from the computer science community that discuss topics related to ours. The difficulty of coupled automata problems is discussed in [5], but this reference does not discuss computational complexity nor the specific problems discussed here. The complexity of verification for systems using more complicated models such as temporal logic and alternating tree automata is discussed in [12, 18, 20, 39]. Although researchers in computer science have shown more complicated verification problems are PSPACE-complete, this paper shows that the supposedly simpler verification of finite-state automata modular systems are PSPACE-complete. The synthesis of distributed systems and supervisors is discussed in [19, 26, 27] under assumptions different from those made in this paper. The supervision of systems from a computer science viewpoint that is similar to the paradigm used here is discussed in [1], but this reference does not discuss modular systems.

The supervision of modular systems is currently receiving much attention from the control research community. See [15, 16, 22, 23, 28, 29, 40, 41] for example. Some of the earlier results relating to modular supervision are shown in [40, 41]. Properties of modular discrete-event systems when the modules have disjoint alphabets are investigated in [28, 29]. Various local specification and concurrent supervision problems, respectively, are investigated in [15, 16]. The supervision

of modular systems using specific architectures is discussed in [22, 23].

With the exception of [10], there has been little work investigating the computational complexity of modular supervision. NP-hardness results for modular supervision problems are shown in [10]. We improve on the results in [10] by showing PSPACE-completeness results and by looking at more problems besides just supervisor existence. Incremental system verification for modular discrete-event systems under weaker assumptions than discussed in this paper are shown in [3]. Online supervision, but for monolithic systems is discussed in [11, 33, 38]. There has been little or no discussion in the control literature related to the online supervision of modular systems.

As would be natural when drawing from the work of two separate research areas, we need to explicitly introduce the notation and assumptions we will be using in the rest of this paper. We do this in Section 2 where we also introduce several simplifying assumptions used in the rest of this paper that do no cause a loss of generality. In Section 3 we present our results related to the computational complexity of several automata intersection problems. In the fourth section we present a brief review of supervisory control of discrete-event systems. In the fifth section of this paper, we reduce several results of the third section to discrete-event systems supervisor existence problems. In the sixth section of this paper we discuss supervisor admissibility. In the seventh section of the paper we show how supervisor verification for modular systems is PSPACE-complete and in the eighth section we discuss online supervision. We close this paper by discussing the implications of the results presented in this paper and by discussing areas of possible future research related to this work.

2 Notation and Assumptions

Although the notation for automata problems used by researchers in computer science and discrete-event systems is similar, there are subtle differences. We generally use the notation of computer science theory when we present the automata intersection problems and we use the notation of supervisory control when we discuss work related to discrete-event systems. However, to aid the reader, we use this section to review the notation used in both fields. For more background information on theoretical computer science, please reference the seminal text by Hopcroft and Ullman [14]. Furthermore, a background on supervisory control and discrete-event

systems can be gained in [6].

We define the automaton G as a 5-tuple $(X^G, x_o^G, \Sigma^G, \delta^G, X_m^G)$ where X^G is the set of states, x_o^G is the initial state, Σ^G is the automaton alphabet, $\delta^G : X^G \times \Sigma^G \rightarrow X^G$ is the (possibly partial) state transition function, and X_m^G is the set of "final" or "marked" states.

For an automaton G, in theoretical computer science, the language *accepted* ($L(G)$) by the automaton G is the set of all strings that lead to a final state. $L(G)$ is equivalent to the language *marked* ($\mathcal{L}_m(G)$) in discrete-event system theory. The language *generated* in discrete-event system theory ($\mathcal{L}(G)$) is the set of strings whose state transitions are defined by the transition function $\delta^G(\cdot)$. Note that we use a script $\mathcal{L}$ for discrete-event systems notation and a regular L for computer science notation. When $\delta^G(\cdot)$ is a partial function, $\mathcal{L}(G) \subset \Sigma^*$. $\mathcal{L}(G)$ is a prefix-closed language, i.e., it contains all the prefixes of all its strings. $\mathcal{L}_m(G)$ and $L(G)$ are not prefix-closed in general. For a language K, we use $\overline{K}$ to denote the set of all the prefixes of all the strings in K. We call an automaton that accepts a prefix-closed language a *prefix-closed automaton*. We also say an automaton is *nonblocking* if the prefix-closure of its marked language is equal to its generated language, i.e., $\overline{\mathcal{L}_m(G)} = \mathcal{L}(G)$.

To review the parallel composition operation, suppose we have the automaton G defined above and another automaton $H = (X^H, x_o^H, \Sigma^H, \delta^H, X_m^H)$.

The parallel composition of G and H denoted by $G \| H$ is defined as follows:

$G \| H := ((X^G \times X^H), (x_o^G, x_o^H), \Sigma^G \cup \Sigma^H, \delta^{G\|H}, (X_m^G \times X_m^H))$

where

$$\delta^{G\|H}((x^G, x^H), \sigma) = \left\{ \begin{array}{ll} (\delta^G(x^G, \sigma), \delta^G(x^H, \sigma)) & \text{if} \quad \delta^G(x^G, \sigma)! \wedge \delta^H(x^H, \sigma)! \\ (\delta^G(x^G, \sigma), x^H) & \text{if} \quad \delta^G(x^G, \sigma)! \wedge (\sigma \notin \Sigma^H) \\ (x^G, \delta^H(x^H, \sigma)) & \text{if} \quad \delta^H(x^H, \sigma)! \wedge (\sigma \notin \Sigma^G) \\ \text{undefined otherwise} & \end{array} \right\}$$

Note that we use the unary operator ! where $f(\alpha)!$ returns true if $f(\cdot)$ is defined for input α, false otherwise. We assume without loss of generality that the automata in this paper have a common alphabet Σ because we can always add self-loops at all states for all events not initially in an automaton's alphabet.

Given a set of h modules modeled as automata $\{H_1, H_2, ..., H_h\}$, we use the script notation

$\mathcal{H}_1^h$ to denote the set of the module automata $\{H_1, H_2, ..., H_h\}$ and the regular notation H_1^h to denote the parallel composition $H_1 \| H_2 \| ... \| H_h$. H_1^h accepts (generates) a string t if and only if t is accepted (generated) by all automata in $\mathcal{H}_1^h = \{H_1, H_2 ..., H_h\}$. This implies that $\mathcal{L}_m(H_1^h) = \mathcal{L}_m(H_1) \cap ... \cap \mathcal{L}_m(H_h)$.

Similarly, for a set of k languages $\{K_1, K_2, ..., K_k\}$, we use the script notation $\mathcal{K}_1^k$ to denote the set $\{K_1, K_2, ..., K_k\}$ and the regular notation K_1^k to denote the intersection of the languages $K_1 \cap K_2 \cap ... \cap K_k$. We also use the notation $\mathcal{L}(\mathcal{H}_1^h)$ and $\mathcal{L}_m(\mathcal{H}_1^h)$ to denote the sets of languages $\{\mathcal{L}(H_1), \mathcal{L}(H_2), ..., \mathcal{L}(H_h)\}$ and $\{\mathcal{L}_m(H_1), \mathcal{L}_m(H_2), ..., \mathcal{L}_m(H_h)\}$, respectively.

It is well-known ([9]) that the problem of showing the language equivalence of two nondeterministic finite-state automata is PSPACE-complete. With this information it is also easily shown that for two automata A and B, deciding $L(A) \subseteq L(B)$ for the nondeterministic case is also PSPACE-complete because verifying $L(A) \subseteq L(B)$ and $L(B) \subseteq L(A)$ also verifies that $L(A) = L(B)$, a known PSPACE-complete problem. Because of these discouraging results for simple nondeterministic automata comparison problems, we discuss deterministic automata exclusively in this paper. It is well known that we can decide $L(A) \subseteq L(B)$ and $L(A) = L(B)$ in the deterministic case in polynomial time [14].

3 Complexity of Automata Intersection Problems

Kozen [17] demonstrates that given a set $\mathcal{A}_1^a = \{A_1, A_2, ..., A_a\}$ of deterministic automata, the problem of deciding if $L(A_1^a) = \emptyset$ is PSPACE-complete. This problem is called the finite-state automata intersection emptiness problem (also called DFA-Int.) This problem has also been discussed in [9, 21]. Kozen's result is rather disappointing because PSPACE-complete problems are known to be at least as hard as NP-complete problems. It is known that if a is fixed to be less than some value, then the DFA-Int problem can be solved in polynomial time.

In this section of the paper we examine other finite-state automata intersection problems and prove some computational complexity results. We are explicitly interested in decision problems comparing the behaviors of two sets of composed automata. We commonly use the phrases "composed automata" and "intersected automata" to denote the same concept - a set of automata interacting through parallel composition.

We find that in general, decision problems involving composed automata are PSPACE-

complete although a few problems are decidable in polynomial time. We start by demonstrating that a general class of composed automata decision problems are in PSPACE.

Proposition 1 *Given an instance of two sets of interacting deterministic finite-state automata $\mathcal{A}_1^a$ and $\mathcal{B}_1^b$ accepting languages not necessarily prefix-closed, the problem of deciding the following expressions are in PSPACE:*

1. $L(A_1^a) \subseteq L(B_1^b)$
2. $L(A_1^a) = L(B_1^b)$

Proof:

We show that the problem of deciding if $L(A_1^a) \not\subseteq L(B_1^b)$ is in NPSPACE. A well known result from complexity theory is that PSPACE=NPSPACE [37]. This means that a deterministic Turing machine bounded to a polynomial amount of space cannot solve more problems if it operates in a nondeterministic manner. Showing a problem to be in NPSPACE is sufficient to show that problem is also in PSPACE.

We present a nondeterministic algorithm using a polynomial amount of space that decides if there is a string s accepted by all $A_1, ..., A_a$ that constitute $\mathcal{A}_1^a$ but not by all $B_1, ..., B_b$ that constitute $\mathcal{B}_1^b$. We start by placing markers on all the start states of $A_1, ..., A_a, B_1, ..., B_b$ and nondeterministically generate events σ from the common alphabet Σ. As the events are generated the current states of $A_1, ..., A_a, B_1, ..., B_b$ are updated in accordance with their transition structures. If a generated event is ever undefined at the current state of an automaton, the algorithm halts. If a string of events is generated such that the current states in $A_1, ..., A_a$ are marked and a current state in one of the automata in $B_1, ..., B_b$ is not marked, we know that $L(A_1^a) \not\subseteq L(B_1^b)$.

Keeping track of the current automata states and updating them as nondeterministic events are generated requires less space than the encodings of $\mathcal{A}_1^a$ and $\mathcal{B}_1^b$ so this operation takes a polynomial amount of space with respect to the problem encoding. Therefore, deciding $L(A_1^a) \not\subseteq L(B_1^b)$ is in NPSPACE. We therefore know that deciding $L(A_1^a) \subseteq L(B_1^b)$ is in coNPSPACE. Because NPSPACE=PSPACE=coPSPACE=coNPSPACE [8] deciding if $L(A_1^a) \subseteq L(B_1^b)$ is also in PSPACE.

By a similar construction, it can easily be shown that deciding $L(A_1^a) = L(B_1^b)$ is also in PSPACE. ■

Now that we have shown a class of problems is in PSPACE, we would like to show some PSPACE-completeness results. Kozen's proof for DFA-Int depends on a reduction from the LBA acceptance problem which is given a linear bounded automaton(LBA) Ψ and a string x, to decide if Ψ accepts x. The LBA acceptance problem is a well-known PSPACE-complete problem [14].

A linear bounded automaton Ψ is a special type of Turing machine with a bounded tape defined as follows:

$\Psi = (Q, \Sigma, \Gamma, \delta, q_o, B, F, p)$

where

Q is the finite set of symbols to represent the set of states of Ψ,

Σ is the finite set of input symbols,

Γ is the finite set of tape symbols,

$\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$ is the next move function,

q_o is the start state symbol,

\$ is a blank symbol,

F is the set of accepting state symbols,

$p : \mathbb{N} \rightarrow \mathbb{N}$ is a polynomial function.

A linear bounded automaton operates in a manner similar to a regular Turing machine except that the single input and work tape is constrained to use at most $p(n)$ cells where $n = |x|$ is the length of the input x.

Kozen reduces the LBA acceptance problem to the DFA-Int problem by generating a set of automata $\mathcal{B}_{\Psi,x}$ that simulate the set of computations performed by Ψ for a given input x. The string x is accepted by Ψ if and only if the automaton $B_{\Psi,x}$ equivalent to the parallel composition of the automata in $\mathcal{B}_{\Psi,x}$ accepts a non-empty language.

Many topics related to modular plants and specifications in discrete-event systems deal with the prefix-closure of languages so it would be advantageous for us to investigate whether automata intersection problems are PSPACE-complete when we restrict our attention to problems dealing with automata accepting prefix-closed languages. We therefore expand the work in [17] by exploring parallel composition properties of automata generating prefix-closed languages.

Theorem 1 *Given a finite-state automaton A and a set of interacting finite-state automata $B_1, ..., B_b$ all accepting prefix-closed languages, the problem of deciding if $L(B_1^b) = L(A)$ is PSPACE-complete.*

Proof: This proof is a modified version of the proof in [17] that shows the DFA-Int problem is PSPACE-complete. Because neither the proof of DFA-Int nor our modifications are trivial, we replicate Kozen's work and alter as necessary.

Using the definition of Ψ seen above, we reduce in polynomial time the linear bounded automaton acceptance problem for an instance of an LBA Ψ and a string x to the prefix-closed case of deciding $L(B_1^b) \neq L(A)$ for a given set of automata $\{B_1, ..., B_b\}$ and an automaton A. The comparison problems in Proposition 1 are more general than the problem in this theorem so we know deciding $L(B_1^b) = L(A)$ for the prefix-closed case is in PSPACE. It is therefore sufficient for us to demonstrate that the deterministic LBA acceptance problem can be reduced in polynomial-time using a many-one mapping to the problem of deciding $L(B_1^b) \neq L(A)$ for the prefix-closed case.

We assume without loss of generality that Ψ has a unique accepting state q_f and that Ψ erases its work tape and moves its read/write head to the left of the tape before accepting. We also assume that Ψ takes an even number of steps before accepting. These assumptions can be made without loss of generality because the size of the Turing machine finite control will at most double.

The instantaneous description (ID) of a Turing machine represents as a finite string the current state of the Turing machine, the current contents of the work tape and the location of the read/write head. Suppose $y = y_1 y_2$ where y is the contents of a Turing machine tape and y_1 represents the content of the tape to the left of the read/write head. Let the first letter of y_2 represent the tape cell being read by the read/write head and let the rest of y_2 be the content of the tape to the right of the read/write head. If q represents the current state, an effective representation of the ID would be the string $y_1 q y_2$.

In this proof, we pad the ID representation with a string of normally unwritten blank symbols $\$^{p(n)-(|y_1|+|y_2|)}$ to make explicit in the representation of the ID the fact that the LBA has a work tape of size $p(n)$ where n is the length of the input string. Therefore with $y = y_1 y_2$, an ID of the LBA would be $y_1 q y_2 \$^{p(n)-(|y_1|+|y_2|)}$.

If x is the input string to Ψ, the initial instantaneous description (ID_o) would be $q_o x\$^{p(n)-|x|}$. Because of our previous assumptions on how Ψ accepts a string, there is a unique accepting instantaneous description $ID_f = q_f\$^{p(n)}$. We use the notation $ID_j \vdash_\Psi ID_i$ to represent that according to the transition rules of Ψ, instantaneous description ID_i follows in one step from instantaneous description ID_j. It should therefore be readily apparent that $[x \in L(\Psi)]$ if an only if $[\exists(ID_o, ID_1, ..., ID_f)$ such that $\forall i \in [1, ..., f](ID_{i-1} \vdash_\Psi ID_i)]$. This means that a string x is accepted by Ψ if and only if there is a sequence of instantaneous descriptions $ID_o \vdash_\Psi ID_1 \vdash_\Psi ... \vdash_\Psi ID_f$ starting with the initial instantaneous description ID_o and finishing with the accepting instantaneous description ID_f.

Let $\Delta = \Gamma \cup Q \cup \{\$\}$ and let $\#$ be a previously unused symbol. To perform our reduction from an instance of the LBA acceptance problem to an instance of the prefix-closed automata intersection problem, we generate a set of prefix-closed interacting automata $\mathcal{B}_{\Psi,x}$ such that the automaton $B_{\Psi,x}$ equivalent to the automaton formed by the parallel composition of the automata in $\mathcal{B}_{\Psi,x}$ accepts the language

$$[[(\Delta \cup \{\#\})^*] \setminus [(\Delta \cup \{\#\})^* \{\#\#\} (\Delta \cup \{\#\})^*]]$$
$$\bigcup \{\#ID_o\#ID_1\#...\#ID_f\#\#\} \subset (\Delta \cup \{\#\})^*$$
$$\text{if } [\forall i \in \{1, ..., f\}(ID_{i-1} \vdash_\Psi ID_i)],$$
$$[[(\Delta \cup \{\#\})^*] \setminus [(\Delta \cup \{\#\})^* \{\#\#\} (\Delta \cup \{\#\})^*]]$$
$$\text{otherwise.}$$

$B_{\Psi,x}$ always accepts all strings not containing $\#\#$ and $B_{\Psi,x}$ accepts a string ending with $\#\#$ if and only if there is a sequence of instantaneous descriptions from ID_o to ID_f that represent a set of valid computations for Ψ.

We can construct an automaton $A_{\Psi,x}$ in time polynomial with respect to the encoding of Ψ and x such that

$L(A_{\Psi,x}) = [[(\Delta \cup \{\#\})^*] \setminus [(\Delta \cup \{\#\})^* \{\#\#\} (\Delta \cup \{\#\})^*]]$.

Note that $L(B_{\Psi,x})$ and $L(A_{\Psi,x})$ are both prefix-closed by construction. For this reduction $L(B_{\Psi,x}) \neq L(A_{\Psi,x})$ if and only if $x \in L(\Psi)$ where $\mathcal{B}_{\Psi,x}$ and A are constructed in polynomial time from x and Ψ. Therefore deciding $L(B_{\Psi,x}) = L(A_{\Psi,x})$ is PSPACE-complete because PSPACE = coPSPACE.

When the read/write head moves, the state updates, a symbol is written on the current tape

cell and the tape head should move exactly one cell to the left or the right. Therefore, to verify that ($ID_i \vdash_\Psi ID_j$), we need to verify that the tape contents in ID_i and ID_j are identical except for where the read/write head wrote to the tape during the transition and that the read/write head moved exactly one tape square to the left or right according to the next move function δ. With this in mind, given a three element string $\alpha_1\alpha_2\alpha_3$ from ID_i and a three element string $\beta_1\beta_2\beta_3$ from ID_j both at the same relative locations in the instantaneous descriptions, we can verify in polynomial time that $\beta_1\beta_2\beta_3$ can follow from $\alpha_1\alpha_2\alpha_3$. If we verify this for all pairs of three element strings at the same locations in ID_i and ID_j, we can verify in polynomial time that ($ID_i \vdash_\Psi ID_j$).

We now construct two sets of interacting automata, $\mathcal{B}^{even}$ and $\mathcal{B}^{odd}$. The automata in $\mathcal{B}^{even}$ verify for a sequence of instantaneous descriptions $ID_i, ..., ID_j$ that the instantaneous descriptions at odd numbered locations follow from the even instantaneous descriptions. Similarly, the automata in $\mathcal{B}^{odd}$ verify for a sequence of instantaneous descriptions $ID_i, ..., ID_j$ that the even instantaneous descriptions follow from the odd instantaneous descriptions.

With this in mind, we construct B_i^{even} to accept the language

$$\left(\{\#\Delta^{i-1}\alpha_1\alpha_2\alpha_3\Delta^{p(n)-i-2}\#\Delta^{i-1}\beta_1\beta_2\beta_3\Delta^{p(n)-i-2}\}^*\{\#\#\}\right)$$
$$\bigcup[(\Delta\cup\{\#\})^*\setminus[(\Delta\cup\{\#\})^*\{\#\#\}(\Delta\cup\{\#\})^*]]$$

where $\alpha_1\alpha_2\alpha_3, \beta_1\beta_2\beta_3 \in \Delta^3$. A string containing ## (i.e., $\#ID_0\#ID_1\#...\#ID_f\#\#$) is accepted by B_i^{even} only if the i^{th}, $(i+1)^{\text{st}}$ and $(i+2)^{\text{nd}}$ symbols in the odd instantaneous descriptions follow from the i^{th}, $(i+1)^{\text{st}}$ and $(i+2)^{\text{nd}}$ symbols in the even instantaneous descriptions.

Similarly, let us also construct B_i^{odd} to accept the language

$$(\{\#\Delta^{p(n)+1}\}\{\#\Delta^{i-1}\eta_1\eta_2\eta_3\Delta^{p(n)-i-2}\#\Delta^{i-1}\theta_1\theta_2\theta_3\Delta^{p(n)-i-2}\}^*\{\#\Delta^{p(n)+1}\#\#\})$$
$$\bigcup[(\Delta\cup\{\#\})^*\setminus[(\Delta\cup\{\#\})^*\{\#\#\}(\Delta\cup\{\#\})^*]]$$

where $\eta_1\eta_2\eta_3, \theta_1\theta_2\theta_3 \in \Delta^3$. A string containing ## (i.e., $\#ID_0\#ID_1\#...\#ID_f\#\#$) is accepted by B_i^{odd} only if the i^{th}, $(i+1)^{\text{st}}$ and $(i+2)^{\text{nd}}$ symbols in the even instantaneous descriptions follow from the i^{th}, $(i+1)^{\text{st}}$ and $(i+2)^{\text{nd}}$ symbols in the odd instantaneous descriptions. Remember that we assume without loss of generality that f is odd.

Let us construct B_i^{even}'s and B_i^{odd}'s for i ranging from 0 to $(p(n)-1)$. This should take less than $6|\Delta|^3 p(n)$ states each, so this construction can be performed in polynomial time with respect to the encodings of Ψ and x. Note that by their constructions, the languages accepted

by the B_i^{even}'s and B_i^{odd}'s are prefix-closed.

Define $\mathcal{B}^{even} = \{B_0^{even}, ..., B_{p(n)-1}^{even}\}$ and $\mathcal{B}^{odd} = \{B_0^{odd}, ..., B_{p(n)-1}^{odd}\}$. B^{even} and B^{odd} are respectively the automata equivalent to $(B_0^{even} \| ... \| B_{p(n)-1}^{even})$ and $(B_0^{odd} \| ... \| B_{p(n)-1}^{odd})$. B^{even} accepts a string containing ## (notably $\#ID_i\#ID_{i+1}\#...\#ID_j\#\#$) only if the odd instantaneous descriptions follow from the even instantaneous descriptions. Likewise, B^{odd} accepts a string containing ## (notably $\#ID_i\#ID_{i+1}\#...\#ID_j\#\#$) only if the even instantaneous descriptions follow from the odd instantaneous descriptions.

We construct a final automaton B^{final} that accepts the prefix closure of the following set of strings:

$$[(\Delta \cup \{\#\})^* \setminus [(\Delta \cup \{\#\})^* \{\#\#\} (\Delta \cup \{\#\})^*]] \bigcup \left(\{\#ID_o\}\{\#\Delta^{p(n)+1}\}^*\{\#ID_f\#\#\}\right)$$

B^{final} accepts a string of instantaneous descriptions ending with ## only if the first instantaneous description is ID_o and the final instantaneous description is ID_f. Note that B^{final} also accepts a prefix-closed language. Constructing B^{final} takes less than $6|\Delta|^3 p(n)$ states, so this construction can be performed in polynomial time.

Let $\mathcal{B}_{\Psi,x} = \{B^{final}\} \cup \mathcal{B}^{even} \cup \mathcal{B}^{odd}$. If there is a valid accepting computation for Ψ with input x then $ID_0, ID_1, ..., ID_f$ is a sequence of valid accepting computations for Ψ and $\#ID_0\#ID_1\#...\#ID_f\#\#$ is accepted by $B_{\Psi,x}$. Likewise, if a string containing ## is accepted by $B_{\Psi,x}$, it must be the string $\#ID_0\#ID_1\#...\#ID_f\#\#$ representing a valid computation on Ψ for accepting input x. A string containing ## is accepted by all the automata in $\mathcal{B}_{\Psi,x}$ if and only if there is a valid computation for Ψ that accepts x. We therefore know $[x \in L(\Psi)] \iff [L(B_{\Psi,x}) \neq L(A_{\Psi,x})]$. This completes our many-one mapping.

$A_{\Psi,x}$ and the components of $\mathcal{B}_{\Psi,x}$ can be constructed in polynomial time with respect to the size of the encoding of x and Ψ. Therefore, the problem of deciding $L(\mathcal{B}_1^b) = L(A)$ for the prefix-closed case is PSPACE-complete. ■

The primary alterations in the proof of Theorem 1 from the proof of DFA-Int is that the automata in the Theorem 1 proof accept all prefix-closed strings not containing the substring ## and they accept a string containing ## if and only if their parts of the LBA computation are valid. Therefore a string containing ## is accepted by the construction in Theorem 1 if and only if a string x is accepted by Ψ.

The result of Theorem 1 is particularly discouraging. PSPACE-complete problems are thought to be rather difficult; they are known to be at least as difficult as NP-complete problems. However, it is well known that the problems of deciding if $L(A) \subseteq L(B)$ and $L(A) = L(B)$ are in P for monolithic automata. This observation prompts the following proposition.

Proposition 2 *Given an instance of a deterministic finite-state automaton A not necessarily accepting a prefix-closed language and a finite set of interacting deterministic finite-state automata $\mathcal{B}_1^b = \{B_1, ..., B_b\}$ also not necessarily accepting prefix-closed languages, the problem of deciding if $L(A) \subseteq L(B_1^b)$ is in P.*

Proof: We demonstrate this proposition by presenting a polynomial time procedure to solve this problem.

Because the automata all have common alphabets

$[L(A) \subseteq L(B_1^b)] \iff [\forall\ i \in (1, ..., b)\ [L(A) \subseteq L(B_i)]]$

$L(A) \subseteq L(B_i)$ can be verified in polynomial time with respect to the length of the encoding of A and B_i, so $[\forall\ i \in (1, ..., b)\ [L(A) \subseteq L(B_i)]]$ can be verified in polynomial time with respect to the length of the encoding of A and $\mathcal{B}_1^b$. Therefore the problem of deciding $L(A) \subseteq L(B_1^b)$ is in class P. ■

However, even with the positive results of Proposition 2, the converse problem is very difficult. For the prefix closed case, given a finite set of interacting deterministic finite-state automata $\mathcal{B}_1^b = \{B_1, ..., B_b\}$ deciding if $L(B_1^b) \subseteq L(A)$ is PSPACE-complete.

Theorem 2 *Given a finite-state automaton A and a set of interacting finite-state automata $\mathcal{B}_1^b = \{B_1, ..., B_b\}$ all generating prefix-closed languages, the problem of deciding if $L(B_1^b) \subseteq L(A)$ is PSPACE-complete.*

Proof: We already know this problem is in PSPACE due to Proposition 1 above. The construction used in this proof is identical to the proof used in Theorem 1 above. We do not repeat the construction for the sake of brevity. We can show using $\mathcal{B}_{\Psi,x}$ and $A_{\Psi,x}$ from the construction in the proof of Theorem 1 that:

$[x \notin L(\Psi)] \iff [L(B_{\Psi,x}) \not\subseteq L(A_{\Psi,x})]$

Therefore, the problem of deciding $L(B_1^b) \not\subseteq L(A_{\Psi,x})$ for the prefix-closed case given $\{B_1, ..., B_b\}$ and A is PSPACE-complete. We then know that the problem of deciding $L(B_1^b) \subseteq L(A_{\Psi,x})$ for the prefix-closed case is PSPACE-complete. ■

We can now extend our results to the non-prefix-closed cases of the problems discussed above.

Corollary 1 *The problems of deciding if $L(B_1^b) = L(A)$ and $L(B_1^b) \subseteq L(A)$ given $\mathcal{B}_1^b$ and A for the non-prefix-closed case is PSPACE-complete.*

Proof: It is easy to see that these problems are a special case of the problem in Proposition 1 above, so these problems are in PSPACE. It should also be apparent that these problems are more general than the decision problems in Theorem 1 and Theorem 2 above so these problems are in PSPACE and are PSPACE-hard; consequently these problems are PSPACE-complete. ■

After Corollary 1 this should be readily apparent, but for the sake of entirety we mention that the problems discussed in Proposition 1 are PSPACE-complete.

4 Control of Discrete-Event Systems

Discrete-event systems are systems modeled as having discrete states and discrete events that cause transitions between those states. Following the modeling system of Ramadge and Wonham [30, 42], we model systems as finite-state automata with external supervisors. Control actions are enforced by selectively disabling controllable events. Supervisors are also modeled as finite-state automata that can observe some events and control a potentially different set of events. Supervisors should not be able to disable uncontrollable events and control actions should not update on the occurrence of locally unobservable events.

Given a supervisor S and a system G, we denote the composed system of S supervising G as the supervised system S/G. Furthermore, because we assume we are using parallel supervisors realized as finite-state automata, S/G is equivalent to $S\|G$. Supervisor S is said to be nonblocking for system G if $S\|G$ is nonblocking, i.e., if $\overline{\mathcal{L}_m(S\|G)} = \mathcal{L}(S\|G)$. For the case of multiple supervisors (i.e., decentralized supervision), we assume that an event is disabled if it is disabled by at least one supervisor. For a set of decentralized supervisors $\{S_1, ..., S_s\}$, we adopt a similar notation for $\mathcal{S}_1^s = \{S_1, ..., S_s\}$ and $S_1^s = S_1\|...\|S_s$ as seen above for $\mathcal{H}_1^h$ and H_1^h. Hence, a set of

supervisors $\mathcal{S}_1^s$ controlling G is equivalent to S_1^s/G. As stated before, a supervisor observes only locally observable events and can disable only locally controllable events, denoted by Σ_{oi} and Σ_{ci}, respectively, for supervisor S_i.

We now adapt the supervisory control theory concepts of controllability, M-closure, and coobservability from [30, 36] to handle the cases where the systems and specifications are modular. These adaptations of the definitions are intended to highlight the modular nature of the system and specifications in these properties.

Let $\mathcal{K}_1^k$ and $\mathcal{M}_1^m$ be sets of languages. Let Σ_{ci} and Σ_{oi} be the locally controllable and observable event sets respectively for $i \in \{1, ..., s\}$. Let $P_i : \Sigma^* \rightarrow \Sigma_{oi}^*$ be the natural projection that erases events in $\Sigma \setminus \Sigma_{oi}$. Furthermore let $\Sigma_c = \cup_{i=1}^s \Sigma_{ci}$ and $\Sigma_{uc} = \Sigma \setminus \Sigma_c$

Definition 1 *Consider the sets of languages $\mathcal{K}_1^k$ and $\mathcal{M}_1^m$ such that $M_1 = \overline{M_1}, M_2 = \overline{M_2}, ..., M_m = \overline{M_m}$ and the set of uncontrollable events Σ_{uc}. The set of languages $\mathcal{K}_1^k$ is* modular controllable *with respect to $\mathcal{M}_1^m$ and Σ_{uc} if $\overline{K_1^k}\Sigma_{uc} \cap M_1^m \subseteq \overline{K_1^k}$.*

Definition 2 *Consider the sets of languages $\mathcal{K}_1^k$ and $\mathcal{M}_1^m$. The set of languages $\mathcal{K}_1^k$ is* modular $\mathcal{M}_1^m$-closed *if $K_1^k = \overline{K_1^k} \cap M_1^m$.*

Definition 3 *Consider the sets of languages $\mathcal{K}_1^k$ and $\mathcal{M}_1^m$ such that $M_1 = \overline{M_1}, M_2 = \overline{M_2}, ..., M_m = \overline{M_m}$ and the sets of locally controllable, Σ_{ci}, and observable Σ_{oi} events such that $i \in \{1, ..., s\}$. The set of languages $\mathcal{K}_1^k$ is* modular coobservable *with respect to $\mathcal{M}_1^m$, P_i and Σ_{ci}, $i \in \{1, ..., s\}$ if for all $t \in \overline{K_1^k}$ and for all $\sigma \in \Sigma_c$,*

$\left(t\sigma \notin \overline{K_1^k}\right)$ *and* $(t\sigma \in M_1^m) \Rightarrow$

$\exists i \in \{1, ..., s\}$ *such that* $P_i^{-1}\left[P_i(t)\right]\sigma \cap \overline{K_1^k} = \emptyset$ *and* $\sigma \in \Sigma_{ci}$.

When there is only one observer/supervisor, coobservability is called observability for historical reasons. Note that coobservability is different from non-observability, which is counter to the usual naming conventions used in theoretical computer science. Using these definitions we can demonstrate the following theorem for the existence of supervisors for modular systems.

Theorem 3 *For a given set of finite-state automata system modules $\mathcal{G}_1^g$ and a set of finite-state automata specification modules $\mathcal{H}_1^h$ such that H_1^h is nonblocking, there exists a set of partial observation supervisors $\{S_1, S_2, ...S_s\}$ such that*

$\mathcal{L}_m(S_1^s/G_1^g) = \mathcal{L}_m(H_1^h)$ *and* $\mathcal{L}(S_1^s/G_1^g) = \mathcal{L}(H_1^h)$
if and only if the following three conditions hold:

1. $\mathcal{L}_m(\mathcal{H}_1^h)$ *is modular controllable with respect to* $\mathcal{L}(\mathcal{G}_1^g)$ *and* Σ_{uc}.
2. $\mathcal{L}_m(\mathcal{H}_1^h)$ *is modular coobservable with respect to* $\mathcal{L}(\mathcal{G}_1^g)$, P_1,...,P_s *and* Σ_{c1},...,Σ_{cs}.
3. $\mathcal{L}_m(\mathcal{H}_1^h)$ *is modular* $\mathcal{L}_m(\mathcal{G}_1^g)$*-closed.*

The proof of this theorem is constructive and is a generalization of the proof of the Controllability and Coobservability Theorem discussed in [6]; it depends on a sample-path argument that we do not show here. This result says that a set of nonblocking supervisors $S_1, S_2, ...S_s$ that achieves a set of modular specifications $\mathcal{H}_1^h$ for a modular system $\mathcal{G}_1^g$ (i.e. $\mathcal{L}_m(S_1^s/G_1^g) = \mathcal{L}_m(H_1^h)$ and $\mathcal{L}(S_1^s/G_1^g) = \mathcal{L}(H_1^h)$) exists if and only if the system is modular controllable, modular coobservable and modular $\mathcal{L}_m(\mathcal{G}_1^g)$-closed. These properties completely characterize necessary and sufficient existence conditions for supervisors of modular systems. In turn, these properties can play a role in safe supervisor synthesis when existence conditions are not satisfied for a supervised system to match a specification. Safe supervisor synthesis for monolithic systems is discussed in [6].

Given $\mathcal{H}_1^h$, deciding if H_1^h is nonblocking is a PSPACE-complete problem. This can be shown using a simple reduction from the automata intersection problem presented in [17]. However, we may have enough foreknowledge to decide this property holds in a computationally feasible manner. We assume that the modular specifications are given such that H_1^h is nonblocking. If the specification is blocking, no nonblocking supervisors that achieves the specification can exist.

Similarly, the astute reader will note that $\mathcal{L}_m(H_1^h) \subseteq \mathcal{L}_m(G_1^g)$ is a necessary condition for both $\mathcal{L}_m(S_1^s/G_1^g) = \mathcal{L}_m(H_1^h)$ and $\mathcal{L}_m(\mathcal{H}_1^h)$ to be modular $\mathcal{L}_m(\mathcal{G}_1^g)$-closed. If $\mathcal{L}_m(H_1^h) \not\subseteq \mathcal{L}_m(G_1^g)$ we can replace $\mathcal{H}_1^h$ with $\mathcal{H}_1^h \cup \mathcal{G}_1^g$ so that the specification behavior is strictly smaller than the specification behavior. $H_1\|...\|H_h\|G_1\|...\|G_g$ is the automaton equivalent of the new specification behavior. This substitution will not alter the computational complexity of the problems we discuss later in this paper.

It is also easy to show a more general theorem concerned only with prefix-closed behavior when we are not concerned with blocking.

Theorem 4 *For a given set of prefix-closed finite-state automata system modules $\mathcal{G}_1^g$ and a set of prefix-closed finite-state automata specification modules $\mathcal{H}_1^h$ such that $\mathcal{L}(H_1^h) \neq \emptyset$, there exists a set of partial observation supervisors $\{S_1, S_2, ...S_s\}$ such that $\mathcal{L}(S_1^s/G_1^g) = \mathcal{L}(H_1^h)$ if and only if the following three conditions hold:*

1. *$\mathcal{L}(\mathcal{H}_1^h)$ is modular controllable with respect to $\mathcal{L}(\mathcal{G}_1^g)$ and Σ_{uc}.*
2. *$\mathcal{L}(\mathcal{H}_1^h)$ is modular coobservable with respect to $\mathcal{L}(\mathcal{G}_1^g)$, P_1,...,P_s and Σ_{c1},...,Σ_{cs}.*
3. *$\mathcal{L}(H_1^h) \subseteq \mathcal{L}(G_1^g)$*

As with Theorem 3, if $\mathcal{L}_m(H_1^h) \not\subseteq \mathcal{L}_m(G_1^g)$, we can replace $\mathcal{H}_1^h$ with $\mathcal{H}_1^h \cup \mathcal{G}_1^g$. This substitution will not alter the computational complexity of the problems we discuss later in this paper related to this theorem.

We can also show the following proposition:

Proposition 3 *Deciding modular controllability, modular coobservability and modular $\mathcal{M}_1^m$-closure for sets of languages specified by sets of finite-state automata is in PSPACE.*

Proof: Proving this proposition relies on a "token" argument similar to that employed by Kozen in [17] for proving automata intersection emptiness is in PSPACE. Given a set of events Σ_{uc} and two sets of automata $\mathcal{H}_1^h$ and $\mathcal{G}_1^g$, we show that the problem of deciding modular controllability of $\mathcal{L}_m(\mathcal{H}_1^h)$ with respect to $\mathcal{L}(\mathcal{G}_1^g)$ and Σ_{uc} is in PSPACE. Similar proofs exist to show deciding modular coobservability and modular $\mathcal{M}_1^m$-closure are in PSPACE but are not shown here due to space considerations. Regarding controllability, it is sufficient to show the complementary problem of deciding non-controllability is in NPSPACE.

A nondeterministic string of events t is generated one event at a time and used to model the state transitions in the finite-state automata in $\mathcal{G}_1^g$ and $\mathcal{H}_1^h$ starting from their respective start-states. The current states of the the automata in $\mathcal{G}_1^g$ and $\mathcal{H}_1^h$ need to be saved and updated as new events are generated. As each new event is generated and added to t, we test if $\exists \sigma \in \Sigma_{uc}$ such that

$$[\forall j \in \{1, ..., g\} | (t\sigma \in \mathcal{L}(G_j))] \wedge [\forall i \in \{1, ..., h\}](t \in \mathcal{L}(H_i))|]$$
$$\wedge [\exists l \in \{1, ..., h\} | (t\sigma \notin \mathcal{L}_m(H_i))].$$

If this property ever holds then modular controllability does not hold. All of these operations take a polynomial amount of memory with respect to the encodings of $\mathcal{G}_1^g$ and $\mathcal{H}_1^h$. Because this problem is in NPSPACE, it is also in PSPACE [37]. ■

It should be noted that if we bound the number of supervisors, plants and specifications to be less than some constant k, then we can decide modular controlabillity, modular coobservability and modular $\mathcal{M}$-closure in polynomial time if the modular systems and specifications are given as deterministic finite-state automata. This is why the concepts of controllability, coobservability and M-closure were extended to the modular cases; when the systems and specifications are give in modular instead of monolithic form, it is potentially computationally much more difficult to test these properties if modular to monolithic conversions are to be avoided.

5 Existence Problems for the Supervision of Modular Systems

In this section we explore the computational complexity of deciding supervisor existence for modular systems under various assumptions. We investigate problems where we assume the specification is nonblocking and that the uncontrolled system allows at least as much behavior as the specification (i.e., for the system automata $\mathcal{G}_1^g$ and the specification automata $\mathcal{H}_1^h$ $\mathcal{L}_m(H_1^h) \subseteq \mathcal{L}_m(G_1^g)$ and $\mathcal{L}(H_1^h) \subseteq \mathcal{L}(G_1^g)$ respectively). This section shows the main result of this paper: a large class of supervisor existence problems for modular discrete-event systems are PSPACE-complete. From Theorem 3 and Proposition 3 demonstrated above, it is easy to see that deciding if a decentralized supervisor exists for a modular specification and modular system is in PSPACE.

Corollary 2 *Given a set of finite-state automata system modules $\mathcal{G}_1^g$, a set of finite-state automata specification modules $\mathcal{H}_1^h$, sets of observable events $\Sigma_{o1}, ..., \Sigma_{os}$ and sets of controllable events $\Sigma_{c1}, ..., \Sigma_{cs}$, the problem of deciding if there is a set of decentralized supervisors $\mathcal{S}_1^s$ such that $\mathcal{L}_m(S_1^s/G_1^g) = \mathcal{L}_m(H_1^h)$ and $\mathcal{L}(S_1^s/G_1^g) = \mathcal{L}(H_1^h)$ is in PSPACE.*

Similar problems for prefix-closure specifications as seen in Theorem 4 and centralized supervisors as seen in [6] can also be decided in PSPACE. We now restrict our attention to what should be a relatively simple problem: given a modular system and monolithic specification marking prefix-closed languages, is there a single full-observation supervisor such that the system satisfies the specification? We show that even this restricted subclass of problems is PSPACE-complete.

Theorem 5 *The problem of deciding if there is a full-observation supervisor S with controllable event set Σ_c for a set of prefix-closed finite-state automata system modules $\mathcal{G}_1^g$ and a prefix-closed finite-state specification automaton H such that $\mathcal{L}(S/G_1^g) = \mathcal{L}(H)$ is PSPACE-complete even if we know $\mathcal{L}(H) \subseteq \mathcal{L}(G_1^g)$.*

Proof: We have already shown in Corollary 2 that this problem is in PSPACE. We reduce the problem of deciding whether $L(B_1^b) = L(A)$ to this problem where A and $\mathcal{B}_1^b$ are given prefix-closed finite-state automata. Let $\Sigma_c = \emptyset$. If there exists a supervisor S such that $\mathcal{L}(S/B_1^b) = \mathcal{L}(A)$ then $L(B_1^b) = L(A)$. If there does not exist a supervisor such that $\mathcal{L}(S/B_1^b) = \mathcal{L}(A)$ then $L(B_1^b) \neq L(A)$ because no event can be disabled. Since deciding if $L(B_1^b) = L(A)$ is PSPACE-complete even if we know $L(A) \subseteq L(B_1^b)$ from Theorem 1, the supervisor existence problem is also PSPACE-complete. ■

These results are particularly disappointing because they show that a relatively large and simple class of supervisor existence problems involving modular system automata is PSPACE-complete. Due to Theorem 5 it should also be apparent that deciding modular controllability for languages specified by finite-state automata is PSPACE-complete because modular observability and modular $\mathcal{L}_m(\mathcal{G}_1^g)$-closure are implied by full observation and prefix-closure, respectively.

For the case of full control (namely, $\Sigma_c = \Sigma$) and partial observation, we can show using similar proof methods that the supervisor existence problem for a modular system and a monolithic specifications is likewise PSPACE-complete. This implies that deciding both modular observability and modular coobservability for languages specified by deterministic finite-state automata is also PSPACE-complete. Incidentally, it is shown in [34] that the problem of deciding coobservability for monolithic systems specified by deterministic finite-state automata is PSPACE-complete.

If we do not know that $\mathcal{L}(H_1^h) \neq \emptyset$ or that $\mathcal{L}(H_1^h) \subseteq \mathcal{L}(G_1^g)$, supervisor existence problems remain PSPACE-complete. Likewise, a large class of nonblocking supervisor existence problems for modular systems specified by finite-state automata are also PSPACE-complete due to Theorem 3 because the nonblocking supervisor problems are known to be at least as difficult as prefix-closed specification problems.

6 Admissible Supervisors

As stated before, a supervisor is *admissible* if it updates control actions on locally observable events and disables only locally controllable events. Note that a supervisor's admissibility when operating on a system is not related to that system's specification. For a monolithic discrete-event system, deciding if a supervisor automaton S is admissible for a given system automaton G can be decided in polynomial time. This is because testing admissibility of a parallel supervisor S is equivalent to testing that all transitions occur on observable events and that $\mathcal{L}(S)$ is controllable with respect to $\mathcal{L}(G)$ and Σ_{uc} [6]. This method of testing admissibility also holds for modular systems. To test if a parallel supervisor S is admissible for a modular system $\mathcal{G}_1^g$ we first test that state transitions only occur on the occurrence of observable events and then it is necessary and sufficient to verify that $\mathcal{L}(S)$ is modular controllable with respect to $\mathcal{L}(\mathcal{G}_1^g)$ and Σ_{uc}. Testing that all state transitions in the controller occur on observable events takes polynomial time, but, as was stated earlier in the paper, testing modular controllability of a monolithic specification with respect to a modular finite-state automata system is PSPACE-complete. This prompts the following theorem whose proof was outlined above.

Theorem 6 *Verifying the admissibility of a single supervisor with respect to a modular finite-state automata system is also PSPACE-complete.*

When testing the admissibility of a decentralized supervision system $\{S_1, ..., S_s\}$, with respect to a modular system $\{G_1, ..., G_g\}$ we need to verify first that all state transitions in $\{S_1, ..., S_s\}$ occur on locally observable events. We then need to verify that each local controller disables only locally controllable events when the other controllers are in operation. More formally:

$\forall i \in \{1, ..., s\},\ \mathcal{L}(S_i)$ is modular controllable with respect to $\mathcal{L}(\mathcal{S}_1^s \cup \mathcal{G}_1^g \setminus S_i)$ and Σ_{uci}.

Besides having state transitions only on locally observable events, all local supervisors S_i need to be controllable with respect the system it is modifying, i.e., $\mathcal{S}_1^s \cup \mathcal{G}_1^g \setminus S_i$. The following corollary should now be evident:

Corollary 3 *The problem of testing the admissibility of decentralized control systems specified by finite-state automata is PSPACE-complete.*

7 Complexity of Modular Supervisor Verification Problems

Suppose we are given a control system known to be admissible and we want to verify if it behaves properly with respect to an unsupervised modular system and a set of specifications. This is called the verification problem. We apply our results regarding the computational complexity of automata intersection problems to verification problems for supervised discrete-event systems. We start by showing a simple extension of the results from Proposition 1.

Proposition 4 *Given set of finite-state automata $\mathcal{S}_1^s$, $\mathcal{G}_1^g$ and $\mathcal{H}_1^h$, verifying $\mathcal{L}_m(S_1^s/G_1^g) = \mathcal{L}_m(H_1^h)$, $\mathcal{L}_m(S_1^s/G_1^g) \subseteq \mathcal{L}_m(H_1^h)$ and $\mathcal{L}_m(H_1^h) \subseteq \mathcal{L}_m(S_1^s/G_1^g)$ are all problems in PSPACE.*

Proof: This proof is similar to the proof of Proposition 1 above. Using the previously discussed definitions regarding supervisor operation for the supervisory control set-up, the proof of this proposition should be apparent and is not included for the sake of brevity. ■

Using the results of Section 3 and Proposition 4 we can demonstrate the computational difficulty of verifying a large class properties of modular supervisory control systems.

Theorem 7 *Given supervisor automata S and $\mathcal{S}_1^s$, unsupervised system automata G and $\mathcal{G}_1^g$ and specification automata H and $\mathcal{H}_1^h$. Deciding the validity of each of the following expressions is PSPACE-complete:*

1. $\mathcal{L}(S_1^s/G) = \mathcal{L}(H)$
2. $\mathcal{L}(S/G_1^g) = \mathcal{L}(H)$
3. $\mathcal{L}(S/G) = \mathcal{L}(H_1^h)$
4. $\mathcal{L}(S_1^s/G) \subseteq \mathcal{L}(H)$
5. $\mathcal{L}(S/G_1^g) \subseteq \mathcal{L}(H)$
6. $\mathcal{L}(H_1^h) \subseteq \mathcal{L}(S/G)$

Proof: The listed problems in this theorem are all special cases of the problems in Proposition 4. Therefore these problems are in PSPACE.

The problem in Theorem 1 can be reduced to Problems 1, 2 and 3 in this theorem. This reduction is not show for the sake of brevity, but should be readily apparent. Therefore, Problems 1, 2 and 3 are PSPACE-complete.

The problem in Theorem 2 can be reduced to Problems 4, 5 and 6 in this theorem. This reduction is also not show for the sake of brevity, but should be readily apparent. Therefore, Problems 4, 5 and 6 are like-wise PSPACE-complete. ■

It can be easily seen that the problems listed in Theorem 7 above are special cases of several other problems in PSPACE, notably problems where the marking properties of supervised modular discrete-event systems are verified. These problems are too numerous to conveniently list, but their computational complexity can easily be found as a consequence of Proposition 4 and Theorem 7. Although our listed completeness results deal only with problems where either the supervisor, plant or specification are modular, these results can be easily extended to cases where two or three of the supervisor, plant and specification are modular. It should be noted that if we bound the number of supervisors, plants and specifications to be less than some constant k,then we can decide all of the verification problems listed here in polynomial time.

Despite the seemingly overwhelming number of PSPACE-complete verification problems, there are several important verification problems that can be decided in polynomial time even when there is no restriction on the number of modules. We have already seen in Proposition 2 that given the finite-state automata $\mathcal{B}_1^b$ and A, verifying $L(A) \subseteq L(B_1^b)$ is in P. This result can be used to prove the following propositions.

Proposition 5 *Given a supervisor automaton S, system automaton G and a set of specification automata $\mathcal{H}_1^h$, the problem of verifying $\mathcal{L}_m(S/G) \subseteq \mathcal{L}_m(H_1^h)$ is in P.*

Proof: Because we assume without loss of generality that S, G and $\mathcal{H}_1^h$ all have common alphabets, we know

$\mathcal{L}_m(S/G) \subseteq \mathcal{L}_m(H_1^h) \iff [\forall\ i \in (1, ..., h)\ [\mathcal{L}_m(S/G) \subseteq \mathcal{L}_m(H_i)]]$

$\mathcal{L}_m(S/G) \subseteq \mathcal{L}_m(H_i)$ can be verified in polynomial time with respect to the encodings of of S, G and H_i, so verifying $\mathcal{L}_m(S/G) \subseteq \mathcal{L}_m(H_1^h)$ is also in P. ■

By similar reasoning, we can also show the following proposition:

Proposition 6 *Given a set of supervisors $S_1, ..., S_s$, a set of finite-state automata system modules $G_1, ..., G_g$ and a finite-state automata specification H, the problem of verifying $\mathcal{L}_m(H) \subseteq \mathcal{L}_m(S_1^s/G_1^g)$ is in P.*

Proof: Because we assume without loss of generality that $\mathcal{S}_1^s$, $\mathcal{G}_1^g$ and H all have common alphabets, we know

$\mathcal{L}_m(H) \subseteq \mathcal{L}_m(S_1^s/G_1^g) \iff$

$[\forall\ i \in (1, ..., s)\ [\mathcal{L}_m(H) \subseteq \mathcal{L}_m(S_i)]] \wedge [\forall\ j \in (1, ..., g)\ [\mathcal{L}_m(H) \subseteq \mathcal{L}_m(G_j)]]$

$\mathcal{L}_m(H) \subseteq \mathcal{L}_m(S_i)$ and $\mathcal{L}_m(H) \subseteq \mathcal{L}_m(G_j)$ can be verified in polynomial time with respect to the encodings of of S_i, G_j and H, so verifying $\mathcal{L}_m(H) \subseteq \mathcal{L}_m(S_1^s/G_1^g)$ is in P. ■

8 Online Control Actions

Previously there have been several attempts in the discrete-event systems community to use online supervision methods to synthesize supervisors that are difficult to synthesize in an offline manner. See for instance [33, 43] where online methods for safe decentralized supervision are discussed. One might think that similar online approaches might be used to synthesize safe supervisors for modular systems that restrict behavior in a non-trivial manner, i.e., enable at least one event, but in general this is not possible to do in an efficient manner. A further discouraging result of the work presented earlier is that for a modular system $\mathcal{G}_1^g$ and a modular specification $\mathcal{H}_1^h$, calculating if a single controllable event is safe to enable from the initial state is PSPACE-complete. We call this problem the single event problem.

Theorem 8 *Given a set of modular finite-state automata $\mathcal{G}_1^g$, a modular finite-state automata specification $\mathcal{H}_1^h$, a set of controllable events Σ_c and a set of observable events Σ_o, the problem of deciding if a controllable event σ is safe to be enabled by itself from the initial state is PSPACE-complete.*

Proof: We first show that this decision problem is in PSPACE. Let S be a supervisor that enables only σ at the initial state and disables all on the occurrence of any more events. We have already shown that verifying $\mathcal{L}(S/G_1^g) \subseteq \mathcal{L}(H_1^h)$ is in PSPACE, so we can use the verification problem in Proposition 4 to solve the problem in this proposition.

We now show that this problem is PSPACE-complete. Suppose we have two arbitrary sets of automata $\mathcal{B}_1^b$ and $\mathcal{A}_1^a$. We reduce the PSPACE-complete problem of deciding if $L(B_1^b) \subseteq L(A_1^a)$ to the single event problem using a polynomial-time many-one reduction. This will show that deciding if a single event is valid to be enabled is PSPACE-complete.

Let Σ be the alphabets for $\mathcal{B}_1^b$ and $\mathcal{A}_1^a$ and let α be an event not in Σ. Suppose for every $B_i, i \in \{1, ..., b\}$ we create an automaton $\breve{B}_i$ from B_i such that $L(\breve{B}_i) = \{\alpha\}L(B_i)$ in the following manner. Suppose x_{0i} is the initial state of B_i. $\breve{B}_i$ is a copy B_i except we create a new start state $\breve{x}_{0i}$ and the only transition from $\breve{x}_{0i}$ is on the occurrence of α and leads to x_{0i}. We repeat this procedure to create $\breve{A}_j$ from A_j for $j \in \{1, ..., a\}$. Let $\breve{\mathcal{B}}_1^b$ be the system and let $\breve{\mathcal{A}}_1^a$ be the specification. Let $\Sigma_c = \{\alpha\}$ and let Σ_o be empty. It should be apparent that this construction can be completed in polynomial time with respect to the encodings of $\mathcal{B}_1^b$ and $\mathcal{A}_1^a$.

If $L(B_1^b) \subseteq L(A_1^a)$ then we know we can enable α at the initial state and have a safe system because enabling α will not allow illegal behavior to occur. Similarly, if $L(B_1^b) \not\subseteq L(A_1^a)$ then we know we cannot enable α at the initial state and have a safe system because enabling α will lead to further illegal behavior because $L(\breve{B}_1^b) \not\subseteq L(\breve{A}_1^a)$. So, $L(B_1^b) \subseteq L(A_1^a)$ if and only if we can enable α at the initial state and have a safe system with respect to $\breve{\mathcal{B}}_1^b$, $\breve{\mathcal{A}}_1^a$, Σ_c and Σ_o This completes our polynomial-time many-one reduction. ■

Theorem 8 can also be extended to show PSPACE-hardness or PSPACE-completeness results for many common online supervision problems where multiple online control actions for safety or maximality are computed.

9 Discussion

We have shown that a large class of automata intersection problems are PSPACE-complete, even for supposedly "simpler" prefix-closed cases. This was used in this paper to show many supervisor existence decision problems for the supervisory control of discrete-event systems are likewise PSPACE-complete. Deciding supervisor admissibility is also PSPACE-complete for modular systems. Calculating safe control actions for modular systems were shown to be PSPACE-hard. In [10] it is shown that several modular supervisor existence problems with range specifications are NP-hard. The results of this paper extend those in [10] using different methods.

This paper shows that deciding many supervision problems for modular systems modeled as interacting sets of finite-state automata do not have time-efficient solutions if P $\neq$ PSPACE. Supervisor synthesis is known to be at least as hard as deciding supervisor existence, so supervisor synthesis is therefore known to be similarly computationally difficult. There are polynomial time algorithms to decide the monolithic versions of the problems discussed in this paper, but the intuitive generalizations of these algorithms to modular systems take time and space exponential in the number of automata modules. Note that if the number of automata specifying the systems or specifications is bounded, all problems discussed in this paper can be solved in polynomial time.

Despite the negative results regarding the time-complexity of the problems discussed in this paper, it was shown that the problems are in PSPACE. Therefore, there are *always* space-efficient solutions to the problems discussed here for deterministic finite-state automata modules and other more general systems where we can verify modular controllability, modular coobservability and modular M-closure efficiently in space. These results are in a sense positive in that we can avoid, as far as computation space is concerned, the *state* explosion problem inherent to modular systems. In the worst case the size of the state space of a composed system is exponential in the number of modules, but we only have to store at most a small fraction of those modules in memory to decide supervisor existence.

The results presented herein are also disappointing because (as was mentioned previously), it is generally believed that deterministic finite-state automata problems are fairly simple and the results together with their proofs indicate that many modular problems using more general system and specification models are also intractable. Many of the PSPACE-completeness results in this paper can be extended to other more complex bounded memory system models such as large classes of bounded Petri nets, temporal logic reactive modules and RAM machines where verification of concurrent behavior can be decided using a polynomial amount of space.

We should focus our attention on special cases of interest if we wish to make further progress on developing time-efficient methods for deciding supervisor existence and synthesizing supervisors for modular systems. For instance, it might be helpful to look at specific network architectures or at problems involving systems amenable to divide-and-conquer approaches. A possible simplifying assumption that could be investigated in future research would be to assume that for

a set of interacting automata $\mathcal{G}_1^g$ there exists an automaton G representing a set of "most general behavior" such that every G_i is a copy of G with some transitions removed. This assumption would make calculations of $\mathcal{L}_m(\mathcal{G}_1^g)$ much simpler and would aid in the "modularization" of the system. It might also be helpful to restrict our attention to special cases of interest where we can make assumptions on the structure of the systems that will lead to reducing the computational complexity of deciding supervision and verification problems.

References

[1] A. Bergeron. Sharing out control in distributed processes. *Theoretical Computer Science*, 139:163–186, 1995.

[2] V. D. Blondel and J. N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.

[3] B.A. Brandin, R. Malik, and P. Dietrich. Incremental system verification and synthesis of minimally restrictive behaviors. In *Proc. of 2000 American Control Conference*, pages 4056–4061, 2000.

[4] H.-D. Burkhard. Fairness and control in multi-agent systems. *Theoretical Computer Science*, 189:109–127, 1997.

[5] S. Buss, C. Papadimitriou, and J. Tsitsiklis. On the predictability of coupled automata: An allegory about chaos. *Complex Systems*, 5:525–539, 1991.

[6] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, Boston, MA, 1999.

[7] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observations. *IEEE Trans. Auto. Contr.*, 33(3):249–260, March 1988.

[8] D.Z. Du and K.I. Ko. *Theory of Computational Complexity.* John Wiley and Sons, Inc., 2000.

[9] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[10] P. Gohari and W.M. Wonham. On the complexity of supervisory control design in the RW framework. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 30(5):643–652, 2000.

[11] N. Ben Hadj-Alouane, S. Lafortune, and F. Lin. Centralized and distributed algorithms for on-line synthesis of maximal control policies under partial observation. *Journal of Discrete Event Dynamical Systems: Theory and Applications*, 6:379–427, 1996.

[12] D. Harel, O. Kupferman, and M.Y. Vardi. On the complexity of verifying concurrent transition systems. *Information and Computation*, 173:143–161, 2002.

[13] T.A. Henzinger and P.W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221:369–392, 1999.

[14] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, Reading, MA, USA, 1979.

[15] S. Jiang, V. Chandra, and R. Kumar. Decentralized control of discrete event systems with multiple local specilizations. In *Proc. of 2001 American Control Conference*, pages 959–964, 2001.

[16] S. Jiang and R. Kumar. Decentralized control of discrete event systems with specializations to local control and concurrent systems. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 30(5):653–660, 2000.

[17] D. Kozen. Lower bounds for natural proof systems. In *Proc. 18th Symp. on the Foundations of Computer Science*, pages 254–266, 1977.

[18] O. Kupferman and M. Vardi. Verification of fair transition systems. *Chicago Journal of Theoretical Computer Science*, 1998(2):1–37, 1998.

[19] O. Kupferman and M. Vardi. Synthesizing distributed systems. In *Proc. 16th IEEE Symp. on Logic in Computer Science*, pages 81–92, 2001.

[20] O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[21] K.-J. Lange and P. Rossmanith. The emptiness problem for intersections of regular languages. In I. Havel, editor, *Proc. of the 17th Conf. on Mathematical Foundations of Computer Science, number 629 in LNCS*, pages 346–354. Springer-Verlag, 1992.

[22] R.J. Leduc, B. Brandin, M. Lawford, and W.M. Wonham. Hierarchical interface-based supervisory control: Serial case. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 4116–4121, 2001.

[23] R.J. Leduc, M. Lawford, and W.M. Wonham. Hierarchical interface-based supervisory control: AIP example. In *39th Allerton Conf. on Comm., Contr., and Comp.*, 2001.

[24] F. Lin and W. M. Wonham. Decentralized supervisory control of discrete-event systems. *Information Sciences*, 44:199–224, 1988.

[25] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44:173–198, 1988.

[26] P. Madhusudan and P.S. Thiagarajan. Distributed controller synthesis for local specifications. In *Proc. 28th Int. Colloquium Automata, Languages and Programming*, pages 396–407, 2001.

[27] A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. 31st Symp. on the Foundations of Computer Science*, pages 746–757, 1990.

[28] M. H. Queiroz and J. E. R. Cury. Modular control of composed systems. In *Proc. of 2000 American Control Conference*, 2000.

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

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

[31] P.J. Ramadge and W.M. Wonham. The control of discrete-event systems. *Proc. IEEE*, 77(1):81–98, 1989.

[32] S. L. Ricker and K. Rudie. Incorporating knowledge into discrete-event control systems. *IEEE Trans. Auto. Contr.*, 45(9):1656–1668, 2000.

[33] K. Rohloff and S. Lafortune. On the synthesis of safe control policies in decentralized control of discrete event systems. *IEEE Trans. Auto. Contr.*, 48(6):1064–1068, 2003.

[34] K. Rohloff, T.-S. Yoo, and S. Lafortune. Deciding co-observability is PSPACE-complete. *IEEE Trans. Auto. Contr.*, 48(11):1995–1999, 2003.

[35] K. Rudie and J.C. Willems. The computational complexity of decentralized discrete-event control problems. *IEEE Trans. Auto. Contr.*, 40(7):1313–1318, 1995.

[36] K. Rudie and W.M. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. Auto. Contr.*, 37(11):1692–1708, November 1992.

[37] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal Comput. Sys. Sci.*, 4(2):177–192, 1970.

[38] S. Takai and T. Ushio. On-line decentralized supervisory control of discrete event systems. In *Proc. 39th IEEE Conf. on Decision and Control*, pages 7–8, December 2000.

[39] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.

[40] Y. Willner and M. Heyman. Supervisory control of concurrent discrete event systems. *International Journal of Control*, 54(5):1143–1169, 1991.

[41] K.C. Wong and W.M. Wonham. Modular control and coordination of discrete-event systems. *Journal of Discrete Event Dynamical Systems: Theory and Applications*, 8:247–297, 1998.

[42] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete event systems. *Maths. of Control, Signals and Systems*, 1(1):13–30, 1988.

[43] T.-S. Yoo and S. Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. *Journal of Discrete Event Dynamical Systems: Theory and Applications*, 13(3):335–377, 2002.