# On the Computational Complexity of the Verification of Modular Discrete-Event Systems [1]

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

**Abstract**

This paper investigates issues related to the computational complexity of automata intersection problems. For several classes of problems, comparing the behavior of sets of interacting finite automata is found to be PSPACE-complete, even in the case of automata accepting prefix-closed languages (equivalently, even when all states are marked). This paper uses these results to investigate the computational complexity of problems related to the verification of supervisory controllers for modular discrete-event systems. Modular discrete-event systems are sets of finite automata combined by the parallel composition operation. We find that a large number of modular discrete-event system verification problems are also PSPACE-complete, even for prefix-closed cases. These results suggest that while system decomposition by parallel composition could lead to significant space savings, it may not lead to sufficient time savings that would aid in the study of "large scale" systems.

## 1 Introduction

There has been considerable interest lately in the discrete-event systems community in concepts related to modular systems. Some plants too complicated to model as monolithic discrete-event systems may be easier to model as modular plants of interacting subsystems. Manipulating plants 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 exponential in the number of components. Likewise, there may be several separate specifications for a system and therefore it would be advantageous to keep the specifications modular. If we were to try to combine diverse specifications to form a single monolitic specification, the monolithic specification may be unbearably large due to a similar state explosion problem. Several researchers in the discrete-event systems community have investigated using modular plants and specifications for discrete-event systems; see for instance [1] and [5].

[1]This research was supported in part by NSF grant CCR-0082784.

Problems in PSPACE are defined to be those problems that can be solved by a Turing machine restricted to a tape polynomially long with respect to the length of the input. Similar to a NP-complete problem, a PSPACE-complete problem is a problem such that any other problem in PSPACE can be reduced to that PSPACE-complete problem in a polynomial amount of time using a many-one reduction. It is known that NP $\subseteq$ PSPACE, but the inclusion is believed to be proper. Showing that a problem is PSPACE-complete is strong evidence that the problem is computationally expensive. These problems can be solved in polynomial time if and only if P=NP and NP=PSPACE. The NP vs. PSPACE problem is a major open problem in computer science similar to the P vs. NP problem. A well known result from complexity theory is that PSPACE=NPSPACE [3]. This means that a Turing machine using a polynomial length tape cannot solve more problems if it operates in a nondeterministic manner. Using known methods in computer science such as Savitch's algorithm [11], we can convert the nondeterministic polynomial space algorithms into deterministic polynomial space algorithms. For more background on complexity theory, please consult a standard textbook such as [3].

In this paper we investigate controller verification issues associated with modular systems. Given a (possibly decentralized) control system, a modular plant and a modular specification, can we verify that our specification is satisfied in a computationally feasible manner? In general, we find that verification problems for modular discrete-event systems are PSPACE-complete, meaning that these problems are probably intractable. Although there may not be great savings in time by verifying systems in a modular manner, there is possibly great savings in computation space. This is why the discrete-event systems community began to investigate the use of modules in the first place - to avoid the *state* explosion problem.

There has recently been a lot of research on the computational complexity of other modular problems in discrete-event systems. See for example [4], [7], [9], [10] and [12]. Undecidability properties related to the existence of nonblocking decentralized observers and controllers are explored in [7] and [12]. Rudie and Willems [10] show how

deciding co-observabilty for a bounded number of observers can be decided in polynomial time. Gohari and Wonham [4] show that several controller existence problems are NP-hard and [9] clarifies these results by demonstrating that several controller existence problems are PSPACE-complete. Verification has also been investigated extensively and is an ongoing research topic in several areas related to discrete-event systems. See for example [1].

The work in this paper was inspired by the deterministic finite automata intersection (DFA-Int) problem investigated by Kozen [6]. We generalize Kozen's work and apply our results to verification problems in discrete-event systems. Because we are drawing from the work of two separate research areas, we explicitly introduce the notation and assumptions we will be using in Section 2 where we also introduce several simplifying assumptions that do no cause a loss of generality. In Section 3, we present new results related to the computational complexity of automata intersection problems. In the fourth section of this paper we reduce several results of the third section to discrete-event system controller verification problems and present our main control related results. We close by discussing the implications of the presented results and areas of possible related future research. Due to the necessary brevity of this paper's format we refrain from demonstrating any results besides Theorem 1 because this theorem is central to many results in this paper.

## 2 Notation and Assumptions

In this paper we will generally use the notation of computer science theory when we discuss automata intersection problems and we will use the notation of supervisory control when we discuss work relating to discrete-event systems. For the aid of the reader we use this section to review the basic notations used in both fields.

Computer scientists generally assume that deterministic automata have complete transition functions while supervisory control researchers allow partial transition functions. A result of this difference is evidenced in the meaning of the languages *accepted*, *marked* and *generated* by automata. For an automaton $G$, in theoretical computer science, the language *accepted* by an automaton is the set of all strings that lead to final (marked) states, denoted by $L(G)$. $L(G)$ is equivalent to the language *marked* ($\mathcal{L}_m(G)$) in supervisory control. The language *generated* in supervisory control ($\mathcal{L}(G)$) is the set of strings whose state transitions are defined by the transition function of a discrete-event system $G$. Note that we use a regular $L$ for computer science notation and a script $\mathcal{L}$ for discrete-event systems notation.

We generally assume that the state transition functions of automata are partial, but this assumption does not greatly alter our results. We assume without loss of generality that the automata discussed in this paper have a common alphabet $\Sigma$. If the automata do not have a common alphabet, their alphabets can be extended in polynomial time by adding self-loops at all states for all events not previously in the alphabet that we wish to add. This modification of the automata does not alter the results of the parallel composition operation that is used in this paper to model automata interaction.

We model the interaction between individual automata with the parallel composition operation as seen in [2]. $G\|H$ represents the parallel composition of $G$ with $H$. For a set of $h$ automata $H_1, H_2 ..., H_h$, we use $\mathcal{H}$ to denote $H_1\|H_2\|...\|H_h$. We use a similar notation to denote the intersection of a set of languages. For a set languages $\{K_1, K_2 ..., K_k\}$, we use the short-hand notation $\mathcal{K}$ to denote $K_1 \cap K_2 \cap ... \cap K_k$. When we use the $\mathcal{K}$ notation, we also assume without loss of generality that all the languages in question have a common alphabet.

With these definitions and assumptions, it should be apparent that for $\mathcal{M} = M_1\|M_2\|...\|M_m$:
$\mathcal{L}(\mathcal{M}) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2) \cap ... \cap \mathcal{L}(M_m)$
$\mathcal{L}_m(\mathcal{M}) = \mathcal{L}_m(M_1) \cap \mathcal{L}_m(M_2) \cap ... \cap \mathcal{L}_m(M_m)$

The problem of showing that two nondeterministic finite automata are equivalent is PSPACE-complete [3]. With this information it is also easily shown that deciding $L(M) \subseteq L(H)$ for the nondeterministic case is also PSPACE-complete because verifying $L(M) \subseteq L(H)$ and $L(H) \subseteq L(M)$ also verifies that $L(H) = L(M)$. Because of these discouraging results for simple nondeterministic automata comparison problems, we discuss deterministic automata exclusively in the remainder of this paper. It is well known that deciding $L(H) \subseteq L(M)$ and $L(H) = L(M)$ in the deterministic case can be done in polynomial time. We now investigate some automata comparison problems for sets of interacting deterministic finite automata.

## 3 Complexity of Automata Intersection Problems

Kozen demonstrates that given a set of deterministic automata $\{M_1, M_2, ..., M_m\}$, the problem of deciding if $L(\mathcal{M}) = \emptyset$ is PSPACE-complete. In this section of the paper we examine other finite 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 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 in general, several classes of composed automata decision problems are in PSPACE.

**Proposition 1** *Given an instance of two sets of interacting deterministic finite automata $M_1, ..., M_m$ and $H_1, ..., H_h$ ac-*

*cepting languages not necessarily prefix-closed, deciding if $L(\mathcal{M}) \subseteq L(\mathcal{H})$ and $L(\mathcal{M}) = L(\mathcal{H})$ are in PSPACE.*

The proof of this theorem relies on showing there exists a polynomial space nondeterministic sample-path algorithm to solve the listed problems. Now that we have shown a class of problems are in PSPACE, we show PSPACE-completeness results using reductions from the Linear Bounded Automaton acceptance problem (LBA for short). The LBA acceptance problem is a known PSPACE-complete problem even in the deterministic case [3].

A LBA $\Psi = (Q, \Sigma, \Gamma, \delta(\cdot), q_o, B, F, p(\cdot))$ is a special type of possibly nondeterministic Turing machine. For an input string $x$ of length $n$, the tape of $\Psi$ is bounded to have $p(n)$ cells.
$Q$ is a finite set of symbols representing the set of states
$\Sigma$ is the finite set of input symbols
$\Gamma$ is the finite set of tape symbols
$\delta : Q \times \Sigma \to Q \times \Sigma \times \{L, R\}$ is the next move function
$q_o$ is the start state symbol
$B$ is a blank symbol
$F$ is the set of accepting state symbols
$p : \aleph \to \aleph$ is a polynomial function

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 even for problems dealing with automata accepting prefix-closed languages. We therefore expand the work in [6] by exploring intersection properties of automata accepting prefix-closed languages. For the sake of simplicity we use the term "prefix-closed automaton" to denote an automaton that accepts a prefix-closed language.

**Theorem 1** *Given a finite automaton $H$ and a set of interacting finite automata $M_1, ..., M_m$ all accepting prefix-closed languages, the problem of deciding if $L(\mathcal{M}) = L(H)$ is PSPACE-complete.*

**Proof:** This proof is a modified version of the proof in [6] that shows the DFA-Int problem is PSPACE-complete. Because the proof of DFA-Int is nontrivial nor are our modifications, we replicate Kozen's work and alter as necessary.

Using the definition of $\Psi$ seen above, we reduce the linear bounded automaton acceptance problem for an instance of an LBA $\Psi$ and a string $x$ to the prefix-closed case of deciding $L(\mathcal{M}) \neq L(H)$ in polynomial time. The comparison problems in Proposition 1 are more general than the problem in this theorem so we know deciding $L(\mathcal{M}) = L(H)$ 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 to the problem of deciding $L(\mathcal{M}) \neq L(H)$ for the prefix-closed case. We assume without loss of generality that $\Psi$ has a unique accepting state $q_f$ and that $\Psi$ erases its work tape and moves its read/write head to the left of the tape before accepting. We also assume that $\Psi$ 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 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 work tape with a string of normally unwritten blank symbols $B^{p(n)-(|y_1|+|y_2|)}$ 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 $y_1 q y_2 B^{p(n)-(|y_1|+|y_2|)}$.

If $x$ is the input string to $\Psi$, the initial instantaneous description ($ID_o$) would be $q_o x B^{p(n)-|x|}$. Because of our previous assumptions on how $\Psi$ accepts a string, there is a unique accepting instantaneous description $ID_f = q_f B^{p(n)}$. We use the notation $ID_j \vdash_\Psi ID_i$ to represent that according to the transition rules of $\Psi$, $ID_i$ follows in one step from $ID_j$. 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 $\Psi$ 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 $\Delta = \Gamma \cup Q \cup \{B\}$ 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 $\mathcal{M}$ that when combined using parallel composition accept the language

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

$\mathcal{M}$ always accepts all strings not containing ## and $\mathcal{M}$ 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 $\Psi$.

We can construct an automaton $H$ in time polynomial with respect to the encoding of $\Psi$ and $x$ such that $L(H) = \left(\left((\Delta \cup \{\#\})^*\right) \setminus \left((\Delta \cup \{\#\})^* \{\#\#\} (\Delta \cup \{\#\})^*\right)\right)$.

Note that $L(\mathcal{M})$ and $L(H)$ are both prefix-closed by construction. For this reduction we show $L(\mathcal{M}) \neq L(H)$ if and only if $x \in L(\Psi)$ where $\mathcal{M}$ and $H$ are constructed in polynomial time from $x$ and $\Psi$. Therefore deciding $L(\mathcal{M}) = L(H)$ 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 $\delta$. With this in mind, given a three element string $a_1a_2a_3$ from $ID_i$ and a three element string $b_1b_2b_3$ from $ID_j$ both at the same relative locations in the instantaneous descriptions, we can verify in polynomial time that $b_1b_2b_3$ can follow from $a_1a_2a_3$. 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, $\mathcal{M}^{even}$ and $\mathcal{M}^{odd}$. The automata in $\mathcal{M}^{even}$ 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 $\mathcal{M}^{odd}$ 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 $M_i^{even}$ to accept the language
$\Big(\{\#\Delta^{i-1}a_1a_2a_3\Delta^{p(n)-i-2}\#\Delta^{i-1}b_1b_2b_3\Delta^{p(n)-i-2}\}^*\{\#\#\}\Big)$
$\bigcup \left((\Delta \cup \{\#\})^* \setminus \left((\Delta \cup \{\#\})^* \{\#\#\} (\Delta \cup \{\#\})^*\right)\right)$
where $a_1a_2a_3, b_1b_2b_3 \in \Delta^3$. A string representing a valid execution of $\Psi$ (i.e., $\#ID_0\#ID_1\#...\#ID_f\#\#$) is accepted by $M_i^{even}$ only if the $i^{\text{th}}$, $(i+1)^{\text{st}}$ and $(i+2)^{\text{nd}}$ symbols in the odd instantaneous descriptions follow from the $i^{\text{th}}$, $(i+1)^{\text{st}}$ and $(i+2)^{\text{nd}}$ symbols in the even instantaneous descriptions.

Similarly, let us also construct $M_i^{odd}$ to accept the language
$(\{\#\Delta^{p(n)+1}\}\{\#\Delta^{i-1}c_1c_2c_3\Delta^{p(n)-i-2}\#\Delta^{i-1}d_1d_2d_3\Delta^{p(n)-i-2}\}^*$
$\{\#\Delta^{p(n)+1}\#\#\})$
$\bigcup \left((\Delta \cup \{\#\})^* \setminus \left((\Delta \cup \{\#\})^* \{\#\#\} (\Delta \cup \{\#\})^*\right)\right)$
where $c_1c_2c_3, d_1d_2d_3 \in \Delta^3$. A string representing a valid execution of $\Psi$ (i.e., $\#ID_0\#ID_1\#...\#ID_f\#\#$) is accepted by $M_i^{odd}$ only if the $i^{\text{th}}$, $(i+1)^{\text{st}}$ and $(i+2)^{\text{nd}}$ symbols in the even instantaneous descriptions follow from the $i^{\text{th}}$, $(i+1)^{\text{st}}$ 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 $M_i^{even}$'s and $M_i^{odd}$'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 $\Psi$ and $x$. Note that by their constructions, the languages accepted by the $M_i^{even}$'s and $M_i^{odd}$'s are prefix-closed.

Let us define $\mathcal{M}^{even} = M_0^{even} \| ... \| M_{p(n)-1}^{even}$ and $\mathcal{M}^{odd} = M_0^{odd} \| ... \| M_{p(n)-1}^{odd}$. $\mathcal{M}^{even}$ 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, $\mathcal{M}^{odd}$ 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 $M^{final}$ that accepts the prefix closure of the following set of strings:
$\left((\Delta \cup \{\#\})^* \setminus \left((\Delta \cup \{\#\})^* \{\#\#\} (\Delta \cup \{\#\})^*\right)\right)$
$\bigcup \left(\{\#ID_o\}\{\#\Delta^{p(n)+1}\}^*\{\#ID_f\#\#\}\right)$
$M^{final}$ 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 $M^{final}$ also accepts a prefix-closed language. Constructing $M^{final}$ takes less than $6|\Delta|^3 p(n)$ states, so this construction can be performed in polynomial time.

Let $\mathcal{M} = M^{final} \| \mathcal{M}^{even} \| \mathcal{M}^{odd}$. If there is a valid accepting computation for $\Psi$ with input $x$ then $ID_0, ID_1, ..., ID_f$ is a sequence of valid accepting computations for $\Psi$ and $\#ID_0\#ID_1\#...\#ID_f\#\#$ is accepted by $\mathcal{M}$. Likewise, if a string containing ## is accepted by $\mathcal{M}$, it must be the string $\#ID_0\#ID_1\#...\#ID_f\#\#$ representing a valid computation on $\Psi$ for accepting input $x$. A string containing ## is accepted by all the automata in $\mathcal{M}$ if and only if there is a valid computation for $\Psi$ that accepts $x$. We therefore know $[x \in L(\Psi)] \iff [L(\mathcal{M}) \neq L(H)]$.

Furthermore, $H$ and the components of $\mathcal{M}$ can be constructed in polynomial time with respect to the size of the encoding of $x$ and $\Psi$. Therefore, the problem of deciding $L(\mathcal{M}) = L(H)$ for the prefix-closed case is PSPACE-complete. ■

The primary alterations in the proof of Theorem 1 from the proof of DFA-Int complexity is that the automata in this proof accept all 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 if and only if a string $x$ is accepted by $\Psi$. This proof construction can also be used to show a similar language inclusion problem is PSPACE-complete.

**Theorem 2** *Given a finite automaton $H$ and a set of interacting finite automata $M_1, ..., M_m$ all accepting prefix-closed languages, the problem of deciding if $L(\mathcal{M}) \subseteq L(H)$*

*is PSPACE-complete.*

Theorems 1 and 2 are particularly discouraging because PSPACE-complete problems are thought to be rather difficult. They are known to be at least as difficult as NP-complete problems. Our PSPACE-completenes results from Theorem 1 and Theorem 2 also hold for more general problems where $H$ is replaced by $\mathcal{H}$ such as in Proposition 1 or where the automata are not restricted to be prefix-closed. Despite these negative results some similar automata intersection problems decision problems are in P.

**Proposition 2** *Given a deterministic finite automaton $H$ and a finite set of interacting deterministic finite automata $M_1, ..., M_m$, the problem of deciding if $L(H) \subseteq L(\mathcal{M})$ is in P.*

## 4 Complexity of Modular Controller Verification Problems

We now apply our results regarding the computational complexity of automata intersection problems to verification problems of supervisory controllers for discrete-event systems. Supervisory control of discrete-event systems is a control paradigm discussed in [8]. Controllers observe events occurring in a plant $G$ and issue control actions by disabling controllable events. The plants (i.e. the systems to control), specifications and controllers are represented as finite automata with partial transition functions. We use the standard notation that a controller $S$ controlling a plant $G$ is denoted by $S/G$. For discrete-event systems, all controller states are marked. This convention is used to prevent controllers from affecting the marking of the plant states. Also, $S/G$ is modelled as $S\|G$.

Controllers have also been modeled as decentralized systems where the individual controllers observe and control different sets of events. The controllers take local actions and their behavior is combined globally through the parallel composition operation. In this paper, we denote a decentralized control system $S_1, ..., S_s$ by $\mathcal{S} = S_1\|...\|S_s$. Research on decentralized controllers has been ongoing in the discrete-event systems community; see for example [10] and [13].

A common problem that arises when dealing with large plants or specifications is the *state explosion* problem [2]. When several plants $(G_1, G_2, ..., G_g)$ interact, the best known algorithm for calculating their composed behavior takes time exponential in the number of plants. This prompted the discrete-event systems community to research modular plants and specifications. The thought was that calculating a composed automaton to perform control operations from various sub-plants may be computationally too expensive for large monolithic systems, but we may be able to achieve computational savings if we were to consider the plants to be several composed subsystems. Similar to the notation used in the rest of this paper, we denote modular plants as $\mathcal{G}$ and modular specifications as $\mathcal{K}$. In this section of the paper we investigate the computational complexity of deciding if given a modular plant, specification and controller, whether the specification is satisfied.

We show that a large class of verification problems are PSPACE-complete. We start by showing a simple extension of the results from Proposition 1.

**Proposition 3** *Verifying that $\mathcal{L}_m(\mathcal{S}/\mathcal{G}) = \mathcal{L}_m(\mathcal{K})$, $\mathcal{L}_m(\mathcal{S}/\mathcal{G}) \subseteq \mathcal{L}_m(\mathcal{K})$ and $\mathcal{L}_m(\mathcal{K}) \subseteq \mathcal{L}_m(\mathcal{S}/\mathcal{G})$ are all problems in PSPACE.*

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

**Theorem 3** *Deciding the validity of the following expressions is PSPACE-complete:*

1. $\mathcal{L}(\mathcal{S}/G) = \mathcal{L}(K)$
2. $\mathcal{L}(S/\mathcal{G}) = \mathcal{L}(K)$
3. $\mathcal{L}(S/G) = \mathcal{L}(\mathcal{K})$
4. $\mathcal{L}(\mathcal{S}/G) \subseteq \mathcal{L}(K)$
5. $\mathcal{L}(S/\mathcal{G}) \subseteq \mathcal{L}(K)$
6. $\mathcal{L}(\mathcal{K}) \subseteq \mathcal{L}(S/G)$

It can be easily seen that the problems listed in Theorem 3 above are special cases of several other problems in PSPACE, notably problems where the marking properties of supervisory controllers of modular discrete-event systems are verified. These problems are too numerous to conveniently list, but their computational complexity easily can easily be found as a consequence of Proposition 3 and Theorem 3. Although our listed completeness results deal only with problems where either the controller, plant or specification are modular, these results can be easily extended to cases where two or three of the controller, plant and specification are modular. *It should be noted that if we put a bound on the number of controllers, plants and specifications, we can decide all of the verification problems listed here in polynomial time.*

Despite the seemingly overwhelming number of PSPACE-complete problems related to the verification of modular control problems, there are several important problems that can be verified in polynomial time even when there is no restriction on the number of modules. We have already seen in Proposition 2 that verifying $L(H) \subseteq L(\mathcal{M})$ is in P. This result can be used to prove the following propositions.

**Proposition 4** *Given a controller S, plant G and a set of specifications $K_1,...,K_k$, the problem of verifying $\mathcal{L}_m(S/G) \subseteq \mathcal{L}_m(\mathcal{K})$ is in P.*

**Proposition 5** *Given a set of controllers $S_1,...,S_s$, a set of plant modules $G_1,...,G_g$ and a specification K, the problem of verifying $\mathcal{L}_m(K) \subseteq \mathcal{L}_m(\mathcal{S}/\mathcal{G})$ is in P.*

Propositions 4 and 5 can also be used to prove that several other verification problems are in P.

## 5 Conclusion

We have shown that many controller verification problems for supervisory control of discrete-event systems are PSPACE-complete. It is believed that NP-complete problems cannot be solved in polynomial time and PSPACE-complete problems are believed to be strictly more difficult than NP-complete problems.

The current algorithms developed to solve the PSPACE-complete supervisory control verification problems take time exponential in the number of modules, so adding more modules makes the problems listed more difficult. Note that if the number of deterministic modules is bounded, all problems discussed in this paper can be solved in polynomial time. The PSPACE-completeness result arises from the fact that the number of states that potentially need to be verified is exponential in the number of modules. Our results tell us that verification of large discrete-event systems modeled as interacting sets of finite automata will not lead to computationally tractable results unless we make more assumptions about the models themselves.

A possible simplifying assumption that could be investigated in future research would be to assume that for a set of interacting automata $\mathcal{M}$ there exists an automaton $M$ representing a set of "most general behavior" such that for all $M_i$ that comprise $\mathcal{M}$, $M_i$ is a subautomaton of $M$. This assumption would make calculations of $\mathcal{M}$ much simpler and would aid in the "modularization" of the plant.

Even though many modular verification problems are often PSPACE-complete, what about controller existence problems? Controller existence problems are generally more difficult than verification problems, but this is an open area of research. As mentioned in the introduction, some recent results are shown in [4] and [9].

It may appear at first that modular verification problems are more time-expensive than monolithic problems, but we must realize that a modular problem can be padded out to a monolithic problem by performing the parallel composition of the modules. The monolithic computation may take polynomial time in the size of the monolithic system, but the size of the monolithic system is potentially exponential in the size of its modules. Therefore, there may be no consistent *time* disadvantage in performing modular computations versus monolithic computations. Modular computations appear to take longer because a different metric is used. There is potentially an exponential contraction in information storage space that occurs when a system can be converted efficiently from a monolithic system to a modular system. The possibly great savings in computation space by using modular systems is the reason the discrete-event systems community began to investigate the use of modules in the first place - to avoid the *state* explosion problem.

## References


[1] 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.

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

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

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

[5] 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.

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

[7] H. Lamouchi and J.G. Thistle. Effective control synthesis for DES under partial observations. In *Proc. 39th IEEE Conf. on Decision and Control*, pages 22–28, 2000.

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

[9] K. Rohloff and S. Lafortune. On the controller existence problem for modular discrete-event systems. Preprint.

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

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

[12] S. Tripakis. Undecidable problems of decentralized observation and control. In *Proc. 40th IEEE Conf. on Decision and Control*, pages 4104–4109, 2001.

[13] 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.