Recent Results on Computational Issues in Supervisory Control *

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

April 25, 2003

Abstract

We present and discuss the implications of recent results by several researchers on computational issues in supervisory control of discrete event systems. The first issue discussed is the boundary between decidability and undecidability in centralized and decentralized control problems for systems modeled by finite-state automata and subject to regular language specifications. The second issue discussed is the PSPACE-completeness of a large class of verification and control problems for decentralized and modular systems modeled by sets of interacting finite-state automata coupled by parallel composition. The state space explosion inherent to modular systems and the possible time-space tradeoff that arises for computations on these systems are discussed.

1 Introduction

This paper presents a survey of recent results in applying concepts from the theory of computation to problems related to the supervisory control of discrete-event systems. The current literature in theoretical computer science is mature, well-developed and contains many results that are particularly relevant to problems in supervisory control theory. There has already been considerable interest in applying theoretical computer science results to supervisory control problems. Some notable examples of crossover work between these two fields include [3], [4], [6], [14], [17], [36], [42], [44], [47], [48], [50], [52], [58] and [64].

We assume our systems are modeled as deterministic finite-state automata with regular language specifications as introduced in [42]. We assume the control systems in this paper are "parallel" control systems that are realized as deterministic finite-state automata where the controlled system is synthesized by a parallel composition of the control automata with the uncontrolled system. See [9], [34], [35], [42], [43] and [54] for a sample of major innovative works from the control community on parallel control discrete-event systems and the text [7] for a general introduction to discrete-event system theory.

An important property inherent to computing the solutions to problems is the concept of decidability. Intuitively, a problem is decidable if there exists an algorithm to solve that problem and a problem is said to be undecidable if there are no algorithms to solve that problem. Later in this

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

paper we give more formal background information on the concept of decidability. It was recently found in [29] and [57] that several decentralized nonblocking controller existence problems are *undecidable*. We discuss these and other decidability results associated with similar yet simpler problems discussed in [20], [33], [53], [54] and [65]. We examine the boundary between decidable and undecidable controller existence problems in a supervisory control setting given range specifications.

Besides decidability results, there has been considerable interest lately in the discrete-event systems 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 similar to how decentralized control may at times be more natural than centralized control in some instances. 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 controlling or verifying system behavior, there may be several separate specifications and therefore it would be advantageous to keep the specifications modular as well.

There have been several papers from the computer science community that discuss the difficulty of several modular system problems. The complexity of verification for systems using more complicated models than automata such as temporal logic and alternating tree automata is, discussed in [16], [26], [28] and [59]. Synthesizing distributed systems and controllers, but under assumptions different from those made in this paper, is discussed in [27], [36] and [39]. Bergeron [3] discusses the control of systems from a computer science viewpoint that is similar to the paradigm used here, but does not discuss modular systems.

The control of modular systems is currently receiving much attention from the control research community. See [14], [21], [22], [31], [32], [40], [41], [60] and [61] for example. Reference [60] shows some of the earlier results to modular control. Properties of modular discrete-event systems when the modules have disjoint alphabets are investigated in [40] and [41]. Various local specification and concurrent control problems, respectively, are investigated in [21] and [22]. References [31] and [32] discuss the control of modular systems using specific architectures. With the exception of [14], there has been little work investigating the computational complexity of modular control besides the results discussed herein. NP-hardness results for modular control problems are presented in [14] and incremental system verification for modular discrete-event systems under weaker assumptions than discussed in this paper is discussed in [5].

A problem that is decidable for a monolithic system is guaranteed to be decidable for a modular system because we can convert modular systems to monolithic systems. However, associated with the state explosion problem, problems that are computationally trivial for monolithic systems may become intractable for modular systems. We discuss some results related to the computational complexity of decision problems associated with the control of modular systems. In particular, we discuss results shown in [47], [48] and [50]. Of particular relevance for proving many of the results discussed here is the class of problems called automata intersection problems which were initially discussed in [24]. Many problems related to the verification and control of modular and decentralized systems are found to be PSPACE-complete, meaning that they are probably much more difficult than NP-complete problems. To aid the reader who may be unfamiliar with concepts from computational complexity, we include a brief introduction to this topic.

As would be natural when drawing from the work of two separate research areas, we need to explicitly introduce the notation we will be using in the rest of this paper. We do this in Section 2 where we also give a brief introduction to supervisory control theory. In Section 3 we discuss de-

cidability issues in decentralized control along with an introduction to the concept of undecidable problems. Section 4 discusses the recent results on the computational complexity of decidable problems in decentralized and modular control along with the necessary theoretical background. The paper closes in Section 5 with a discussion of the results presented herein.

2 Notational Review

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 relating 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 [18]. Furthermore, a background on supervisory control and discrete-event systems can be gained in [7].

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, $\delta^G : X^G \times \Sigma^G \to X^G$ is the (possibly partial) state transition function, and X_m^G is the set of "final" or "marked" states. We discuss deterministic systems and specifications exclusively in this paper. We discuss our motivation for this distinction in Section 4.

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 $H = (X^H, x_o^H, \Sigma^H, \delta^H, X_m^H)$.

We now define the parallel composition of G and H denoted by $G \| H$:
$G \| H := ((X^G \times X^H), (x_o^G, x_o^H), \Sigma^G \cup \Sigma^H, \delta^{G\|H}, (X_m^G \times X_m^H))$
where

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

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.

Following the modeling system of Ramadge and Wonham [42, 63], we model systems as finite-state automata with external controllers. Control actions are enforced by selectively disabling controllable events. Controllers are also realized as finite-state automata that can observe some events and control a potentially different set of events. Controllers should not be able to disable uncontrollable events and control actions should not update on the occurrence of locally unobservable events.

Given a controller S and a system G, we denote the composed system of S controlling G as the controlled system S/G. Furthermore, because we assume we are using parallel controllers realized as finite-state automata, S/G is equivalent to $S \| G$. Controller 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 controllers (i.e., decentralized control), we assume that an event is disabled if it is disabled by at least one controller. For a set of decentralized controllers $\{S_1, ..., S_s\}$, 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 controllers $\mathcal{S}_1^s$ controlling G is equivalent to S_1^s/G. As stated before, a controller observes only locally observable events and can disable only locally controllable events, denoted by Σ_{oi} and Σ_{ci}, respectively, for controller S_i.

A very important associated result used in demonstrating controller existence for many problems is the controllability and coobservability theorem seen below.

Theorem 1 *[54] For a finite-state automaton system G and a finite-state automaton specification H such that $\mathcal{L}_m(H) \subseteq \mathcal{L}_m(G)$, sets of controllable events $\{\Sigma_{c1}, ..., \Sigma_{cs}\}$ and sets of observable events $\{\Sigma_{o1}, ..., \Sigma_{os}\}$ there exists a set of partial observation controllers $\{S_1, S_2, ...S_s\}$ such that*
$\mathcal{L}_m(S_1^s/G) = \mathcal{L}_m(H)$ and $\mathcal{L}(S_1^s/G) = \overline{\mathcal{L}_m(H)}$
if and only if the following three conditions hold:

1. *$\mathcal{L}_m(H)$ is controllable with respect to $\mathcal{L}(G)$ and Σ_{uc}.*
2. *$\mathcal{L}_m(H)$ is coobservable with respect to $\mathcal{L}(G)$, Σ_{o1},...,Σ_{os} and Σ_{c1},...,Σ_{cs}.*
3. *$\mathcal{L}_m(H)$ is $\mathcal{L}_m(G)$-closed.*

We expect the reader is familiar with the concepts of controllability, coobservability and $\mathcal{L}_m(G)$-closure which are discussed in [7]. Given two automata G and H, algorithms exist to decide controllability, coobservability and $\mathcal{L}_m(G)$-closure.

A less restrictive version of Theorem 1 also holds for the case of generated language specifications (and hence prefix-closed specifications) where the $\mathcal{L}_m(G)$-closure condition is disregarded. Also, in the case of centralized control (i.e., one controller), coobservability is called observability. In this paper we use the terminology of control theory even though it may be counter to naming conventions currently used in computer science theory. (That is, coobservability is not non-observability.)

The result of Theorem 1 is useful for deciding if given a system G and a specification language E such that $E \subseteq \mathcal{L}_m(G)$, there exists a nonblocking controller S such that $\mathcal{L}_m(S/G) = E$. This is

called the nonblocking controller existence problem with zero tolerance. A similar problem that we discuss in the next section is the controller existence problem with tolerance where given languages A and E such that $A \subseteq E \subseteq \mathcal{L}(G)$, we must decide if there exists a controller S such that $A \subseteq \mathcal{L}(S/G) \subseteq E$. This is called the "controller existence problem with tolerance", or the "range problem" for short. There are extensions of the range problem for the cases of decentralized control and/or if we desire the controlled system to exhibit nonblocking behavior.

3 Decidability Issues in Decentralized Control

Now that we have introduced the preliminaries of supervisory control theory we discuss some recent research on the decidability of several controller existence problems. In Section 2 we discussed how we can decide controller existence when the behavior of the controlled system is required to match a specification. It is well known in the literature that deciding decentralized controller existence with zero tolerance and nonblocking behavior is decidable, but it was recently shown in [29] and [57] that no algorithm exists to decide nonblocking decentralized controller existence with tolerance. In this section we discuss the boundary between decidability and undecidability for control problems with tolerance.

We first give the necessary background information on the theory of computation that is relevant for understanding the concept of decidability. We then discuss several results related to decidability of various problems for deciding controller existence with tolerance in [20], [29], [33], [53], [54], [57] and [65]. If the reader is interested in more background information, please consult one of the many excellent texts on the subject including [11], [18], [37] and [56].

3.1 An Introduction to Undecidability

Central to understanding the concept of undecidability is the concept of a Turing machine which is considered by many researchers in computer science to be a good theoretical model for a general computation device. Although the exact definition of a Turing machine is beyond the scope of this paper, it is similar to a finite-state automaton with unlimited memory. Given an arbitrary input string x, a Turing machine T is said to accept x if when given x as input, T performs a series of computations and halts in an accepting state. Likewise, a Turing machine T is said to reject x if T performs a series of computations when given x as input and halts in a rejecting state. Note that a Turing machine may not halt for all input strings.

A language (i.e., a set of strings) is said to be *recursively enumerable* or *Turing recognizable* if there exists a Turing machine that accepts the strings in that language and fails to accept all strings not in that language. Note that when we say that a string fails to be accepted by a Turing machine, the string could either be rejected or the Turing machine may not halt for that string. A language is said to be *recursive* or *decidable* if there exists a Turing machine that accepts those strings in the language and rejects all other strings. Note that for a decidable language, there is always a Turing machine that halts in the correct accept or reject state for a given input string, by definition. Although a decidable language is Turing recognizable, a Turing recognizable language may not be decidable and the distinction between Turing recognizable and decidable languages is nontrivial.

A decision problem is defined to be any problem such that every instance of that problem is said to be labelled either "true" or "false". Any problem instance of a decision problem can be encoded as a string x of ones and zeros that can be given as an input to a Turing machine T. A

Turing machine is said to *decide* a problem if when given a problem instance encoded as a string as input, the Turing machine performs a series of computations and eventually halts in either an accept state or a reject state. Furthermore, when the Turing machine halts, it halts in an accept state if and only if the problem instance is labelled as "true". A problem is said to be *decidable* if there is a Turing machine that decides the problem. A problem is said to be *undecidable* if it is not decidable and therefore there are no Turing machines to decide that problem. Intuitively, this follows from the common understanding in computer science that an algorithm to decide a problem must always halt by definition.

The Church-Turing hypothesis states that the Turing machine model is equivalent to any reasonable computation device in its ability to solve problems. This statement is impossible to prove, but the Church-Turing hypothesis is generally believed among computer scientists. Although Turing machines may not be as reasonable to implement as more common real-world computation devices such as RAM machines, they are very useful as a theoretical model for computations and demonstrating undecidability. Therefore, from the hypothesis, if we can show there are no Turing machines to solve a problem, then there are most likely no other computation devices that can solve the problem. Please consult one of the standard texts mentioned earlier for more background information.

The most common method to show that a problem is decidable is to use a constructive proof and present an algorithm that always halts and that solves the problem. There are no time or space constraints on algorithms to demonstrate decidability other than halting in finite time; we make a further discussion of the time and space complexity of decidable problems in Section 4

As may be expected, not all problems are decidable. This means that there are some problems that cannot be solved by any computation device and the set of undecidable problems is nontrivial. A sample of important undecidable problems includes the Turing machine string acceptance problem, the Turing machine empty input halting problem and Post's Correspondence Problem. The Turing machine string acceptance problem is formulated as follows: given a Turing machine T and a string x, decide if T halts on an accepting state if it is run with x as its input. The Turing machine empty input halting problem is formulated as follows: given a Turing machine T, decide if T halts when it is run with an empty input. Due to space limitations it is beyond the scope of this paper to describe Post's Correspondence Problem (also called PCP), but we encourage the reader to consult the standard texts on the theory of computation that we mentioned earlier for a description of this and other problems.

The most common method to show a problem B is undecidable is to perform a reduction from a known undecidable problem A to that problem (denoted $A \leq B$). Intuitively, a reduction is an algorithm that takes an instance of problem A and converts it to an instance of problem B. This means that if we can solve problem B, then we can solve problem A. Hence, by contradiction, if there is no algorithm for solving problem A, then there can be no algorithm to solve problem B if we can show a reduction between these two problems.

Now that we have introduced the preliminaries of the theory of computation, in the next subsection we discuss recent results on the decidability of the controller existence problem when the specification has tolerance.

3.2 Boundary Between Undecidability and Decidability

Because of the controllability and coobservability theorem, deciding controller existence for supervisory control problems with partial observation when we wish the controlled system to exactly match a specification's behavior is well understood. We can always calculate if there exists a set

of partial observation decentralized controllers for a system that achieves a specification. However, the controller existence problem does not always have a solution when the controlled system behavior is allowed to be in a range of solutions. For this problem, we are given a system G and two prefix-closed regular languages A and E such that $A \subseteq E \subseteq \mathcal{L}(G)$. As was mentioned earlier, we are asked to decide if there exists a partial observation controller S such that $A \subseteq \mathcal{L}(S/G) \subseteq E$. We may place additional restrictions on this problem where instead of a centralized controller S, we may be required to use a decentralized control system. Furthermore, we may also have the additional requirement that S/G is nonblocking.

The decidability of the simplest form of this problem (i.e., centralized control and blocking is of no concern) is investigated by Rudie and Wonham in [53] and Lin in [33]. Given a language M, let $infO(M)$ represent the infimal prefix-closed observable superlanguage of M and let $supC(M)$ represent the supremal controllable sublanguage of M. Lin [33] shows that there exists a controller S such that $A \subseteq \mathcal{L}(S/G) \subseteq E$ if and only if $infO(A) \subseteq supC(E)$. A method for calculating $supC(E)$ is given in [62] and [53] shows a method for calculating $infO(A)$. Therefore, by construction, the centralized controller existence problem with tolerance where blocking is disregarded is decidable. Furthermore, [53] show a method for synthesizing this controller.

We now investigate a slightly more complicated controller existence problem with tolerance and nonblocking specifications. Suppose we are given a system G and a specification language E such that $E \subseteq \mathcal{L}_m(G)$, and the goal is to decide if there exists a nonblocking safe controller S such that $\mathcal{L}_m(S/G) \subseteq E$ and $\overline{\mathcal{L}_m(S/G)} = \mathcal{L}_m(S/G)$. There is a method in [20] for computing the *union* of all safe nonblocking solutions to this problem. This effectively decides if there exists a nonblocking controller S such that $\mathcal{L}_m(S/G) \subseteq E$. Yoo [65] shows a method for synthesizing such a controller S to solve this problem. Hence, for the centralized control case, both the existence and synthesis problems for nonblocking safe controllers are decidable even when we assume partial observation.

Another slightly more complicated problem than the first problem we discuss above is the *decentralized* controller existence problem when we do no care about blocking. Let $infCCCo(M)$ represent the infimal prefix-closed, controllable and coobservable superlanguage of M. It is shown in [54] that there exists a pair of decentralized controllers S_1 and S_2 such that $A \subseteq \mathcal{L}(S_1 \| S_2/G) \subseteq E$ if and only if $infCCCo(A) \subseteq E$. A method for calculating $infCCCa(A)$ is also shown in [54], so the (possibly blocking) decentralized controller existence problem with tolerance is decidable. Using the method outlined in [54], we can synthesize a controller to solve this problem.

We now look at the fourth and final problem in our hierarchy of controller existence problems with tolerance. Suppose we are given an automaton G and two regular languages A and E such that $A \subseteq E \subseteq \mathcal{L}_m(G)$. It is shown in [29] and [57] using different reductions that the problem of deciding if there exists a pair of decentralized controllers S_1 and S_2 such that $A \subseteq \mathcal{L}_m(S_1 \| S_2/G) \subseteq E$ and $\overline{\mathcal{L}_m(S_1 \| S_2/G)} = \mathcal{L}(S_1 \| S_2/G)$ is *undecidable*.

In [29], the authors use a reduction from the Turing machine empty input halting problem to a modified version of the decentralized controller existence problem that uses a Rabin automaton model, but the reduction also holds for our system model. In [57], the author uses a reduction from Post's Correspondence Problem to another slightly modified decentralized controller existence problem where it is assumed that $A = \emptyset$. This restriction is not problematic because the reduction still holds for proving the decentralized nonblocking controller existence problem when we make no assumption on A.

Note that these nonblocking decentralized control problems with tolerance are undecidable for two-controller systems and as may be expected, remain undecidable if we assume a more general n-controller system. The decidability of the decentralized controller existence problem

where blocking is not a concern is not altered if we use an n-controller system, but in general the computational complexity of these problems is greatly increased in this case. We discuss these results in the next section.

We would also like to draw attention to the fact that the decidability of control problems is not altered if we assume modular systems or specifications. We discuss this more in the following section, but an algorithm for converting modular systems to monolithic ones is known. However, using known methods, the conversion of modular systems to monolithic systems may take exponential time and space in the worst case. We show in the following section that modular control and verification problems are in general much more computationally expensive than monolithic control and verification problems.

4 The Computational Complexity of Decidable Problems

Even though many problems in supervisory control theory are decidable, this does not automatically imply that these problems can be decided in a computationally feasible manner. Some decidable problems may be computationally difficult or even infeasible to solve in reasonable manner. The necessary computations may be so unbearably difficult that they may take too much time, or worse, use too much computation space. Researchers from theoretical science have developed a theory of computational complexity to categorize the computational "difficulty" of decidable problems and we apply this theory to decentralized and modular control problems in supervisory control theory.

We initiate this section with a review of computational complexity to acquaint the reader with this theory. We then introduce a general class of problems called automata intersection problems that are particularly relevant to deciding the computational complexity of decentralized and modular supervisory control problems. We discuss computational issues in decentralized control and then computational issues in modular control. We present online control methods which have been proposed to avoid the computational difficulty of many of these problems. The time-space trade-off inherent to computations involving many large state-space problems are discussed.

4.1 Review of Computational Complexity

We present in this subsection a brief review of needed concepts from the theory of computation. For a more thorough exposition of these topics, the reader is encouraged to consult one of the standard texts in the field such as [11], [12], [37].

Problems are said to be in class P if they can be solved in polynomial time by a deterministic computation device. The exact type of computation device does not matter as long as it is a "reasonable" computation device such as a deterministic Turing machine or a deterministic RAM machine. Similarly, problems are said to be in class NP if they can be solved in polynomial time by a nondeterministic computation device. The class PSPACE includes all problems that can be solved in a polynomial amount of space by a deterministic computation device and the class NPSPACE includes all problems that can be solved in a polynomial amount of space by a nondeterministic computation device.

By Savitch's theorem [55] we know that PSPACE=NPSPACE, but a similar result is not known for time-bounded computation. It is known that P $\subseteq$ NP $\subseteq$ PSPACE, but both of these inclusions are thought to be proper. Proving or disproving P$\neq$NP and NP $\neq$ PSPACE are major open problems in computer science.

We use a special type of reduction called a "polynomial-time many-one reduction" to denote that one problem is computationally "more difficult" than another. For two problems $C \subseteq \Sigma_c^*$ and $D \subseteq \Sigma_d^*$, we say that there is a polynomial-time many-one reduction from C to D (denoted $C \leq_m^p D$) if there exists a polynomial-time computable function $f : \Sigma_c^* \to \Sigma_d^*$ such that for each $x \in \Sigma_c^*$, $x \in C$ if and only if $f(x) \in D$ [11]. Intuitively, it can be thought that if a polynomial-time many-one reduction exists as described above, problem D can be thought to be at least as difficult to solve as problem C.

A different kind of polynomial time reduction based on Oracle Turing Machines (OTM's) is the polynomial time Turing reduction (denoted $\leq_T^p$) which is similar to the polynomial time many-one reduction described in the preceding paragraph. Although the distinction between many-one reductions and Turing reductions is beyond the scope of this paper, it suffices for us to state that a many-one reduction implies a Turing reduction (i.e., $C \leq_m^p D \Rightarrow C \leq_T^p D$). Readers interested in more information concerning details of Turing reductions should reference the texts mentioned in the beginning of this section.

Problem C is called PSPACE-complete if it is in PSPACE and *all* problems in PSPACE can be reduced to C using polynomial-time many-one reductions. Similarly, a problem D is called NP-complete if it is in NP and if all problems in NP can be reduced to D using polynomial-time many-one reductions. PSPACE-complete problems are problems that are considered to be the "most difficult" of the problems in PSPACE and are at least as hard as all NP-complete problems. Showing a problem to be NP-complete or PSPACE-complete is generally considered good evidence that the problem is intractable. This may initially seem non-intuitive, but we use polynomial-time reductions instead of polynomial-space reductions to define the PSPACE-complete class because if we were to use polynomial-space reductions, all problems in PSPACE would be PSPACE-complete. Given a PSPACE-complete problem C, if we can show for another problem D that $C \leq_T^p D$, then D is known to be PSPACE-hard by definition. (A similar definition holds for NP-hard problems.) If we know D is in PSPACE and $C \leq_m^p D$, then we know D is PSPACE-complete.

It has been shown in [12] 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 [18].

4.1.1 Automata Intersection

An important class of problems that is particularly relevant to our discussion of decentralized control and modular systems are automata intersection problems. These problems involve comparing the behavior of a set of interacting finite-state automata with the behavior of another automaton. Many automata intersection problems are PSPACE-complete, although there are some automata intersection problems in P.

A well known investigation into the complexity of automata intersection was performed by Kozen [24]. Suppose we are given a set of deterministic finite-state automata $\{A_1, A_2, ..., A_n\}$ with a common alphabet Σ^A such that for $i \in \{1, ..., n\}$, $A_i = (X^{A_i}, x_o^{A_i}, \Sigma^A, \delta^{A_i}, X_m^{A_i})$. Suppose also that $A_1 \| ... \| A_n$ represents the parallel composition of the automata in $\{A_1, A_2, ..., A_n\}$ and that $\mathcal{L}_m(A_i)$ represents the language marked by the automaton A_i. Furthermore, $\mathcal{A}_1^n$ represents the set

$\{A_1,...,A_n\}$ and A_1^n represents the parallel composition $A_1\|...\|A_n$. Kozen [24] demonstrates the following theorem.

Theorem 2 *[24] Given a set of automata $\{A_1,A_2,...,A_n\}$, the problem of deciding if $\mathcal{L}_m(A_1\|A_2\|...\|A_n) = \emptyset$ is PSPACE-complete.*

Kozen demonstrates this problem is in PSPACE using a nondeterministic token path argument and shows it is PSPACE-complete using a reduction from the linear bounded automata acceptance problem. This problem is called the deterministic finite-state automata intersection problem and is referred to as "DFA-Int" in the computer science literature (e.g., [11] and [12]).

The reader familiar with automata algorithms will note that if we restrict n such that for some constant k, $n \leq k$, the DFA-Int problem takes polynomial time to solve. However, known algorithms for solving DFA-Int are exponential in n, so as k grows, so does the worst-case complexity of the DFA-Int problem. A natural question to ask is whether this problem becomes any easier if we restrict our attention to the prefix-closed case. As we saw in the previous section, a restriction to prefix-closed languages for problems in supervisory control theory can make the problems decidable, so maybe a restriction to prefix-closure might make a decidable problem computationally tractable. Unfortunately, this is not so as is demonstrated in the following theorem.

Theorem 3 *[48] Given a set of deterministic finite-state automata $\{A_1,A_2,...,A_n,B\}$, the problems of deciding if $\mathcal{L}(A_1\|A_2\|...\|A_n) = \mathcal{L}(B)$ and if $\mathcal{L}(A_1\|A_2\|...\|A_n) \subseteq \mathcal{L}(B)$ are PSPACE-complete.*

The problems in this theorem are shown to be in PSPACE using a nondeterministic path argument and demonstrated to be PSPACE-complete using reductions from the linear bounded automata acceptance problem. Many results discussed in the remainder of this paper make extensive use of reductions from the problems in Theorem 3. These results mean we most likely cannot use divide-and-conquer techniques for solving many modular and decentralized control problems we discuss in the rest of this section. Related DFA-Int properties are discussed in [1], [12], and [30].

However, despite the presented negative results, not all automata intersection problems are PSPACE-complete. In [48], the following theorem is also shown.

Theorem 4 *[48] Given a set of deterministic finite-state automata $\{A_1,A_2,...,A_n,B\}$, the problem of deciding if $\mathcal{L}_m(B) \subseteq \mathcal{L}_m(A_1\|A_2\|...\|A_n)$ is in P.*

The problem in Theorem 4 is computationally easier because we can split this problem into n simpler monolithic problems. Due to the nature of the parallel composition operation, we can test $\mathcal{L}_m(B) \subseteq \mathcal{L}_m(A_1\|A_2\|...\|A_n)$ by verifying that for all $i \in \{1,...,n\}$, $\mathcal{L}_m(B) \subseteq \mathcal{L}_m(A_i)$. This result is used to show several modular and decentralized control problems are in P.

4.2 Decentralized Control Problems

The verification of decentralized control problems for monolithic systems and specifications in the framework of [54] is discussed in [48] where several decentralized control verification problems are shown to be computationally difficult. These results are demonstrated using reductions from the prefix-closed DFA-Int problems presented in the previous subsection.

Theorem 5 *[48] Given a set of controllers $\{S_1,...,S_s\}$, a system G and a specification K, deciding the validity of the following expressions is PSPACE-complete: $\mathcal{L}(S_1^s/G) = \mathcal{L}(K)$, $\mathcal{L}(S_1^s/G) \subseteq \mathcal{L}(K)$.*

The verification problems are shown to be in PSPACE using a nondeterministic token argument. However, as with the automata intersection problems (cf., Theorem 4 above), there are some important verification problems that are in P.

Theorem 6 *[48] Given a set of controllers $\{S_1, ..., S_s\}$, a system G and a specification K, deciding the validity of the following proposition is in P: $\mathcal{L}(K) \subseteq \mathcal{L}(S_1^s/G)$.*

The results of Theorems 5 and 6 can be easily extended to the case of marked language specifications. Also, as may be expected, all of the verification problems are in P if we bound the number of controllers.

Now that we know that the verification of decentralized control systems can be difficult, what if we are given a system G and an automaton specification H, and we are asked to decide if there exists a set of decentralized controllers $\{S_1, ..., S_s\}$ such that the controlled system $\mathcal{S}_1^s/G$ achieves the specification? This problem is decidable although it has been shown to be computationally difficult.

If H and G are deterministic finite-state automata, it is found in [51] that the problem of deciding coobservability is PSPACE-complete.

Theorem 7 *[51] The problem of deciding if $\mathcal{L}(H)$ is coobservable with respect to $\mathcal{L}(G)$, $\Sigma_{c1}, ..., \Sigma_{cn}$, $\Sigma_{o1}, ..., \Sigma_{on}$ is PSPACE-complete when H and G are finite-state automata.*

Deciding coobservability for the n-controller case is shown to be in PSPACE using an $\mathcal{M}$-machine construction as demonstrated in [52] with a nondeterministic token path argument. A reduction from the DFA-Int problem in Theorem 2 is used to demonstrate PSPACE-completeness. Controllability and $\mathcal{L}_m(G)$-closure can easily be decided in polynomial time for deterministic finite-state automata systems and specifications using standard automata algorithms. Therefore, using Theorem 1, we know that deciding controller existence is PSPACE-complete when we wish to match a specification. Furthermore, the PSPACE-completeness of decentralized controller existence shows us that controller synthesis is similarly computationally difficult when we are required to match a specification. Naturally, all decentralized controller existence problems for exactly matching specifications become polynomial time decidable when the number of controllers is bounded.

Although these PSPACE-completeness results show that several decision problems related to decentralized control are computationally difficult, these problems *are* decidable. Also, standard methods for deciding coobservability generally involve enumerating over a large set of system states which is usually a memory-intensive operation, which implies that these methods do not take advantage of the minimum space requirements for these problems in order to solve the problems as fast as possible. Modern computers can often exhaust their memory resources in a short amount of time, so it might be advantageous to trade some extra computation time for more efficient memory usage and therefore solve a larger class of problems. We discuss further implications of the time-space tradeoff inherent to many PSPACE-complete problems in Subsection 4.5.

As was mentioned in the previous section, many range problems for controller existence and synthesis become more difficult or even undecidable when decentralized control is used instead of centralized control. Similar results hold when comparing the computational difficulty of deciding controller existence for modular and monolithic systems. Although there is no shift from decidability to undecidability when using modular systems instead of monolithic systems, modular controller existence problems are frequently intractable even though they may be polynomial-time decidable when monolithic systems are used. We discuss computational issues related to the control of modular systems in the next section.

4.3 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 which was introduced in Section 2. This modeling method is generally considered to be the simplest method that is expressive enough to be used for real-world problems. We start by investigating several verification problems for modular systems and then we discuss controller existence problems for modular systems.

4.3.1 Modular Control System Verification

Control system verification is a very important practical problem in many industrial settings where we may wish to use a pre-designed, off-the-shelf control system with a modular system and we want to ensure that the control objectives are attained. It is generally believed that control system verification problems tend to be computationally simpler than controller existence and synthesis problems, but besides from the practical relevance of control system verification, many methods developed for and the experienced gained from verification can be used for control system existence and synthesis problems. Although in the monolithic case verification can be performed in polynomial time with respect to the size of the automata, we find that several important verification problems for modular systems and specifications are PSPACE-complete.

Theorem 8 *[47] Consider controller automata S and $\mathcal{S}_1^s$, uncontrolled 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. $\mathcal{L}(S/G_1^g) = \mathcal{L}(H)$
2. $\mathcal{L}(S/G) = \mathcal{L}(H_1^h)$
3. $\mathcal{L}(S/G_1^g) \subseteq \mathcal{L}(H)$
4. $\mathcal{L}(H_1^h) \subseteq \mathcal{L}(S/G)$.

Because we are assuming a relatively simple modeling method, our results show that the verification of more complicated systems is likewise PSPACE-complete.

Despite the seemingly large number of PSPACE-complete verification problems, there are several important modular system 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 Theorem 4 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 1 *[47] Given a controller 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.*

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

Proposition 2 *[47] Given a set of controllers $\mathcal{S}_1^s$, a set of finite-state automata system modules $\mathcal{G}_1^g$ 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.*

4.3.2 Controller Existence for Modular Systems

We now extend the supervisory control theory concepts of controllability, M-closure, and coobservability from [42] and [54] to handle the cases where the systems and specifications are modular.

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 $P_i : \Sigma^* \to \Sigma_{oi}^*$ 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 *[47] 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 *[47] 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 *[47] 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* $P_i^{-1}[P_i(t)]\,\sigma \cap \overline{K_1^k} = \emptyset$ *and* $\sigma \in \Sigma_{ci}$.

Definitions 1, 2 and 3 are equivalent to the monolithic definitions of controllability, $\mathcal{M}$-closure and coobservability if we convert the modular systems and specifications to their monolithic equivalents. We introduce these definitions to demonstrate the computational complexity of various properties related to the control of modular systems which we discuss later in this section. It is shown in [47] using a nondeterministic token path argument that verifying modular controllability, modular $\mathcal{M}_1^m$-closure and modular coobservability for languages specified by deterministic finite-state automata is in PSPACE.

We now show a theorem relating necessary and sufficient conditions for the existence of a decentralized controller system such that a modular system achieves a modular specification.

Theorem 9 *[47] 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 controllers $\{S_1, S_2, ...S_s\}$ 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 [7] and depends on a sample-path argument which we do not show here. This result says that a set of nonblocking controllers $S_1, S_2, ...S_s$ that

achieve 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 controllers of modular systems. In turn, these properties can play a role in safe controller synthesis when existence conditions are not satisfied for a controlled system to match a specification. Safe controller synthesis for monolithic systems is discussed in [7]. We can now show the computational complexity of deciding a relatively simple controller existence problem.

Theorem 10 *[47] The problem of deciding if there is a full-observation controller 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 $\mathcal{L}(H) \subseteq \mathcal{L}(G_1^g)$.*

The controller existence problem in Theorem 10 is shown to be in PSPACE using the results of Theorem 9 and the fact that deciding modular controllability, modular $\mathcal{M}_1^m$-closure and modular controllability is PSPACE. The problem is shown to be PSPACE-complete using a reduction from the problem in Theorem 3.

The result in Theorem 10 is particularly disappointing because is shows that a relatively large and simple class of controller existence problems involving modular system automata is PSPACE-complete. Due to Theorem 10 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.

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 [24]. 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 controllers 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 system 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 discussed in this paper.

If we do not know that $\mathcal{L}(H_1^h) \neq \emptyset$ or that $\mathcal{L}(H_1^h) \subseteq \mathcal{L}(G_1^g)$, controller existence problems remain PSPACE-complete. Likewise, a large class of nonblocking controller existence problems for modular systems specified by finite-state automata are also PSPACE-complete due to Theorem 9 because the nonblocking controller problems are known to be at least as difficult as prefix-closed specification problems.

For the case of full control (namely, $\Sigma_c = \Sigma$) and partial observation, we can show using similar proof methods that the controller existence problem for a modular system and a monolithic specification 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.

4.4 Modular Problems and State Explosion

It may appear at first that modular system verification and control decision problems are more time-expensive than their equivalent 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 system 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 (if the system can be effectively decomposed into modules). Therefore, there is no consistent *time* disadvantage in performing modular computations versus monolithic computations. Modular computations appear to take longer because a different metric is used. Namely, we measure computation time with respect to the *size* of the input and the size of the modular problem input is potentially very small. Therefore, 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. We must remember that 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.

4.5 The Time-Space Tradeoff

We know PSPACE-complete decision problems (such as deciding coobservability), can always be solved in a space efficient manner, but we do not believe these problems can be solved in a time efficient manner in the worst case. The traditional methods for solving many modular control and verification problems involve searching over all the reachable system states with breadth-first and depth-first searching methods to test for the reachability of "special" states that indicate the satisfiability of system properties such as controllability or generated language inclusion. These methods would in the worst case store all reachable system states in memory, which is computationally very expensive in both time and space in the worst case. Despite these negative attributes, this method for testing system properties is most likely the *fastest* (i.e., least time expensive) method for testing these properties.

However, when using these methods for performing modular system operations, limitations in system memory generally cause a larger restriction on the problems that can be effectively solved rather than the amount of time needed for computation. This is because we may need to store all reachable system states in memory and in modern computation devices, memory can be filled very fast. Therefore the real restriction on computations involving modular systems using current methods is not computation time, but computation space. It would make sense to trade some computation time for computation space in order to solve more problems. However, we cannot trade too much time for decreased computation space because we need to allow our calculations to complete in a reasonable amount of time.

We have already shown how many decision problems for decentralized control and modular systems are PSPACE-complete which means that these problems can be solved efficiently in computation space. Also, many of the problems we discussed can be easily converted to to problems of deciding reachability in a labelled transition system, which is a well known problem in the computer science community. It is shown in [23] that reachability for a directed graph is NLOGSPACE-complete, meaning that the besides the space needed to store the transition system, a nondeterministic computation device needs at most a space logarithmic in the size of the encoding of the transition to decide reachability. Of course no nondeterministic computation machine exists, but as we alluded to before, Savitch [55] shows a way to convert a nondeterministic com-

putation using space S to a deterministic computation using space S^2. Savitch's limit is the best known space-efficient conversion for the conversion of nondeterministic computations to deterministic computations and can be used to get a general feel for a lower limit on the amount of computation space required for many of the verification and control decision problems we discuss in this paper.

Although powerful, Savitch's method is not entirely intuitive or time-efficient for converting nondeterministic computations to deterministic computations. Savitch's method is a "one-size-fits-all" approach that makes no consideration of special system structure that may be exploited to develop more intuitive or effective algorithms. Savitch's nondeterministic conversion method is essentially a test of reachability in a directed graph representing the computation tree of a nondeterministic space-bounded Turing machine where the nodes of the graph are the various instantaneous descriptions of the Turing machine and the edges represent valid computation steps. This method has been used to give an intuitive space-efficient deterministic method for deciding coobservability using the $\mathcal{M}$-machine construction [49]. This method was developed to attempt to take the maximum known theoretical advantage possible of the time-space tradeoff inherent to PSPACE-complete problems using a method inspired by Savtich's method. However, as may be expected, the algorithm developed in [49] uses more time than reasonable.

This highlights a very interesting area for future research related to modular systems which we mentioned earlier. It would be worthwhile to develop algorithms for deciding control and verification problems for decentralized and modular problems that trade a *reasonable* amount of computation time for a more efficient use of computation space. There has been a large volume of related work from the computer science community that explores the time-space tradeoff in reachability problems. Space-efficient methods for undirected *st*-connectivity is discussed in [2]. There is a heuristic method presented in [8] for searching with restricted memory. A sweep-line state exploration method is discussed in [25] that uses less space than traditional state space exploration methods but still suffers from the state explosion problem. Although the method discussed in [38] stores a large fraction of the system states in memory to decide reachability, the authors use a pseudo-root state technique to cause a reduction in the number of states that need to be enumerated. There has also been a large volume of work related to the verification of systems from the formal methods community in computer science. See the texts [10] and [19] for an introduction.

5 Discussion

We have discussed the computational limitations of various problems being researched in the supervisory control community. We have explored the boundary between decidability and undecidability for deciding controller existence for systems with range specifications. It has been shown that the problem of deciding if there are nonblocking decentralized controllers for range specifications is undecidable, meaning that there are no algorithms for solving this problem. However, this problem is decidable if we assume centralized control and/or if blocking is disregarded.

We also discussed computational issues related to modular discrete-event system control problems that are also being researched in the supervisory control community. There is no shift from decidability to undecidability when modular systems are used instead of monolithic systems although there is a marked increase in the computational complexity of these problems. It is shown that a large number of simple problems which are polynomial time decidable for monolithic systems are PSPACE-complete when we assume modular systems. Verification is found to be com-

putationally difficult in general along with problems of deciding controller existence.

The increased computational complexity for modular problems is most likely due to the necessity of enumerating over all possible system states in the modular systems for control and verification. This exhaustive state enumeration is computationally difficult because the number of system states is exponential in the number of system components in the worst case. However, although the modular control and verification problems discussed are in PSPACE, current methods for solving these computationally expensive problems typically use a large amount of computation space by exhaustively storing all system states in memory in order to save on computation time. This is problematic for modern computation devices as computation space can be filled very quickly and hence, many problems cannot be solved due to space limitations rather than time limitations. Further research is needed on the development of approaches to solve these problems by balancing the time-space tradeoff inherent to many computationally difficult problems.

Our results on deciding controller existence also show that synthesizing controllers for modular systems is likewise computationally difficult. Another approach that has been proposed to circumvent this computational difficulty for control system synthesis has been to use online control methods where the control actions are computed on the fly as system behavior evolves. This approach as been successfully applied to reduce intensive precomputation of control actions in both the centralized and decentralized control of monolithic systems ([15] and [46] respectively). Online methods for centralized control allows for maximal behavior, but there is no known similar result for online decentralized control.

When using online control on monolithic systems, control actions can be updated as observations of system behavior are made in polynomial time with respect to the size of the monolithic system. However, in the worst case, the problem of updating online control actions is PSPACE-complete for modular systems [47]. The high computational complexity is due to the large current state spaces when the online control algorithms make estimates of the current system state. However, we feel that for many real-world systems, this computational difficulty is not problematic if we make system assumptions such that system state estimates are not overly large and the modular online controllers can therefore compute control actions in a computationally reasonable manner.

Our results also imply that instead of working on computationally intensive methods for solving general modular verification problems, we should focus our efforts on heuristic methods to efficiently solve problems for special cases of important practical interest. An example of such a research effort is seen in [45] where the authors investigate modular system models where the modules exhibit a degree of symmetry that allow for a relaxation of the computational complexity.

Finally, we mention that system decomposition is another approach for tackling the computational difficulties associated with complex systems. Petri net models appear to be particularly well-suited for performing system decomposition due to the inherent distributed nature of the states in a Petri net model. This feature is exploited using the concept of place bounded Petri nets in the context of fault diagnosis problems in [13].

References

[1] S. Bala. Intersection of regular languages and star hierarchy. In *Proc. 29th Int. Colloquium Automata, Languages and Programming*, pages 159–169, 2002.

[2] G. Barnes and W. L. Ruzzo. Deterministic algorithms for undirected s-t connectivity using polynomial time and sublinear space (extended abstract). In *ACM Symposium on Theory of Computing*, pages 43–53, 1991.

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

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

[5] 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.

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

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

[8] P.P. Chakrabarti, S. Ghose, A. Acharya, and S.C. de Sarkar. Heuristic search in restricted memory. *Artificial Intellgience*, 41:197–221, 1989.

[9] 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.

[10] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, Cambridge, MA, 2002.

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

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

[13] S. Genc and S. Lafortune. Distributed diagnosis of DES using Petri nets. In E. Best and W.M.P van der Aalst, editors, *International Conference on Application and Theory of Petri Nets - ATPN 2003*. Springer-Verlag, 2003.

[14] 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.

[15] 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.

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

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

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

[19] M.D.A Huth and M.D. Ryan. *Logic in Computer Science Modelling and Reasoning about Systems*. Cambridge University Press, Cambridge, UK, 2000.

[20] K. İnan. Nondeterministic supervision under partial observation. In G. Cohen and J. Quadrat, editors, *11th International Conference on Analysis and Optimization of Systems: Discrete Event Systems*, pages 39–48. Springer-Verlag, 1994.

[21] 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.

[22] 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.

[23] N.D. Jones. Space-bounded reducibility among combinatorial problems. *Journal of Computer and System Sciences*, 11:68–75, 1975.

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

[25] L. Kristensen and T. Mailund. A compositional sweep-line state space exploration method. In D. Peled and M. Vardi, editors, *Formal Techniques for Networked and Distributed Systems - FORTE 2002, number 629 in LNCS*, pages 327–343. Springer-Verlag, 2002.

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

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] F. Lin. *On Controllability and Observability of Discrete Event Systems*. PhD thesis, Department of Electrical Engineering, The University of Toronto, 1987.

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

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

[36] 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.

[37] C.H. Papadimitriou. *Computational Complexity*. Addison Wesley, 1994.

[38] A. Parashkevov and J. Yantchev. Space efficient reachability analysis through use of pseudo-root states. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 50–64, 1997.

[39] 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.

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

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

[42] 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.

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

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

[45] K. Rohloff and S. Lafortune. The control and verification of similar agents operating in a broadcast network. Preprint.

[46] K. Rohloff and S. Lafortune. On the synthesis of safe control policies in decentralized control of discrete event systems. Manuscript to appear in IEEE Trans. on Automatic Control.

[47] K. Rohloff and S. Lafortune. PSPACE-completeness of automata intersection decision problems with applications to supervisory control. Preprint.

[48] K. Rohloff and S. Lafortune. On the computational complexity of the verification of modular discrete-event systems. In *Proc. 41st IEEE Conf. on Decision and Control*, Las Vegas, Nevada, December 2002.

[49] K. Rohloff and S. Lafortune. Space efficient methods for testing reachability with applications to coobservability and decentralized control. Technical Report CGR03-08, Department of Electrical Engineering and Computer Science, University of Michigan, 2003.

[50] K. Rohloff, T.-S. Yoo, and S. Lafortune. Deciding coobservability is PSPACE-complete. Manuscript accepted to IEEE Trans. on Automat. Control.

[51] K. Rohloff, T.S. Yoo, and S. Lafortune. Deciding coobservability is PSPACE-complete. Technical Report CGR03-06, Department of Electrical Engineering and Computer Science, University of Michigan, 2003.

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

[53] K. Rudie and W.M. Wonham. The infimal prefix-closed and observable superlanguage of a given language. *Systems & Control Letters*, 15:361–371, 1990.

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

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

[56] M Sipser. *Introduction to the Theory of Copmutation*. Brooks Cole, first edition, 1997.

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

[58] J. Tsitsiklis. On the control of discrete-event dynamical systems. *Mathematics of Control, Signals and Systems*, 2:95–107, 1989.

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

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

[61] 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.

[62] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, 1987.

[63] 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.

[64] T.-S. Yoo and S. Lafortune. On the computational complexity of some problems arising in partially-observed discrete-event systems. Manuscript submitted to IEEE Trans. on Automat. Control.

[65] T.S. Yoo. *Monitoring and Control of Partially-Observed Discrete-Event Systems*. PhD thesis, The University of Michigan, Ann Arbor, 2002.