Space Efficient Methods for Testing Reachability with Applications to Coobservability and Decentralized 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

Technical Report CGR 03-8

March 28, 2003

Abstract

All known methods of deciding coobservability use exponential space and we demonstrate that we can decide coobservability using polynomial space. Coobservability is one of the necessary and sufficient conditions for deciding decentralized controller existence. Our method for deciding coobservability is based on a deterministic space efficient method we introduce for deciding state reachability for finite state systems. This reachability decision method runs in space

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

$O(q^2)$ where q is the size of the encoding of a system state. This memory bound allows us to decide reachability in large state space systems in a manner that avoids space constraints caused by the state explosion problem. However, there is a space-time tradeoff and our methods are potentially expensive in computation time. We discuss some experimental results that demonstrate this new method of deciding coobservability is more space efficient than the current known method but the time-space tradeoff causes these algorithms to be very inefficient in time.

1 Introduction

Many real-world problems require decentralized control. Central to decentralized control is the concept of coobservability. It has been shown that deciding coobservability is very difficult. Generally space considerations are more problematic. There has been no known work in the literature on space-efficient methods for deciding coobservability. We show a method that runs in polynomial space at the expense of increased computation time. More work will hopefully be done at balancing the time-space tradeoff.

We base our method for deciding reachability on the Rudie M-machine. Deciding reachability. This is an important problem for directed graphs and has been researched since the beginning of computer science.

Given a finite state system, a fundamental problem is to decide if one system state is reachable from another. Indeed, many important properties of systems modeled as finite state systems such as safety, controllability and coobservability can be reduced to problems of deciding state reachability. Algorithms are usually designed to be time efficient while space(memory) efficiency is usually ignored because time efficiency implies space efficiency. However, constraints on space for computation devices can be non-trivial

for many problems when we have to deal with the state explosion problem inherent to many concurrent systems. We show a space efficient method for deciding reachability in finite state labelled transition systems and show how this result can be used to decide coobservability in a space-efficient manner.

The reachability problem for finite state systems has several different names in the computer science literature. It is also called *st*-connectivity and it is equivalent to searching over a directed graph. The traditional time-efficient methods for deciding reachability in finite-state systems include depth-first and breadth-first searches [6]. These methods store all reachable states in memory in the worst case. Our method conserves computation space by storing in memory at most a logarithm of all the system states.

It is shown in [8] that reachability for a directed graph is NLOGSPACE-complete, meaning that the besides the space needed to store the directed graph, a nondeterministic computation device needs at most a space logarithmic in the size of the encoding of the graph to decide reachability. Of course no nondeterministic computation machines exist, but Savitch [17] shows a way to convert a nondeterministic computation using space S to a deterministic computation using space S^2. With Savitch's result, we can convert the nondeterministic algorithm for deciding reachability to a deterministic algorithm that uses space $O\left((\log(n))^2\right)$ where n is the length of the encoding of the directed graph.

Although powerful, Savitch's method is a not entirely intuitive method for converting nondeterministic computations to deterministic computations. Savitch's method is a "one-size-fits-all" approach that takes no consideration of special system structure that may be exploited to develop more intuitive or effective algorithms. This is the main motivation for this paper; although there are many space efficient and intuitive *non*deterministic methods for deciding many of the problems discussed here, there are few or no space efficient deterministic methods besides those found using the conversion presented by

Savitch.

The method we present for deciding reachability of finite state systems is inspired by Savitch's algorithm which is essentially a test of st-connectivity 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. Our method is more general than Savitch's reachability test in that it can be applied to more than just testing the reachability of a bounded space nondeterministic Turing machine computation tree accepting state. Our method is also more intuitive than Savitch's conversion and can be easily adapted for subclasses of finite state systems where we can take advantage of specialized system structure. We present our reachability method for a general type of finite state system representation called a *labelled transition system* in order to make our presentation independent of any particular modeling formalism.

Our reachability method can be applied to modular discrete-event systems [13], the $\mathcal{M}$-machine from [15] and tape-bounded Turing machines. Modular discrete-event systems are sets of finite automata that are combined using the parallel composition operation and a "state" of an n-modular discrete-event system is an n-tuple of the module states. We can model the computation tree of a tape-bounded Turing machine as a directed graph where the "states" represent instantaneous descriptions of the Turing machine as was done in [17]. The $\mathcal{M}$-machine is used to decide coobservability. The "states" of the $\mathcal{M}$-machine of an n-observer system are modeled as $(n+2)$-tuples of the specification and uncontrolled system states. We will discuss the $\mathcal{M}$-machine in more detail below.

We use the reachability method presented here to develop a method to decide coobservability for languages specified by finite state automata, a known PSPACE-complete problem [14]. The previously known algorithm

for deciding coobservability takes exponential space in the worst case, but our new method takes polynomial space in the worst case. We present discuss experimental results for this new coobservability algorithm and compare our results to the traditional method of testing coobservability shown in [15].

We know PSPACE-complete problems (such as 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. Although our method for deciding coobservability takes more time than the previous method in the worst case, we justify our method by stating that space constraints are usually the more important and limiting factors when dealing with the state explosion problem [5]. When solving state reachability by exhaustively storing system states in memory, such as with breadth-first and depth-first searching methods in the worst case, memory can be filled very fast and it would make sense to trade some computation time for computation space in order to solve more problems. In our research we wish to explore a lower bound for the space required to decide coobservability. This captures our strategy for an alternative solution to the coobservability problem; we trade an increase in computation time for a reduction in required computation space.

The methods discussed here can also be easily applied to modular control problems [13] and [12] and other modeling formalisms such that the necessary preconditions are satisfied. This is an area of ongoing research as current methods for the control of modular systems also suffer due to the state explosion problem.

There has been some previous research in space-efficient methods for deciding reachability in finite-state systems. Space-efficient methods for undirected *st*-connectivity is discussed in [1]. There is a heuristic method presented in [3] for searching with restricted memory. A sweep-line state exploration method is discussed in [9] that uses less space than traditional state

space exploration methods but still suffers due to the state explosion problem aggravating memory constraints. Although the method discussed in [11] 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.

In Section 2 we present our general space efficient method for deciding reachability in finite state systems and analyze the the worst-case space and time complexity. In Section 3 we review the concept of coobservability and apply our reachability method to deciding coobservability. We discuss the results of this paper and several experimental results in Section 5.

2 Space-Efficient Reachability Methods

As was mentioned in the introduction, we present our space-efficient reachability method for *labelled transition systems* to make our presentation as general as possible and independent of any particular modeling formalism that we may wish to apply this method to.

Definition 1 *A* labelled transition system *is a 3-tuple* $T = (X, \Sigma, \delta)$ *where* X *is a finite set of states,* Σ *is a finite set of transition labels and* $\delta \subseteq X \times \Sigma \times X$ *is the transition relation.*

Labelled transition systems capture the inherent system features that we exploit to develop our space-efficient reachability method. For one, labelled transition systems have a finite state space X with cardinality $|X|$ We assume that we can encode a state $x_a \in X$ in space q. Also, for a labelled transition system with two system states x_a and x_b, we should be able to verify that x_b can be reached from x_a using one transition in a computationally trivial manner. We do not quantify what we mean by "computationally trivial", but we justify this assumption later when we discuss applications of our method.

We use the notation $x_a \vdash x_b$ to represent that x_b can be reached from x_a using one transition. We say that x_k is reachable in d steps from x_g if there exists a string of states $x_g, x_{i_1}, ..., x_{i_n}, x_k$ such that $x_g \vdash x_{i_1}, x_{i_1} \vdash x_{i_2}, ..., x_{i_n} \vdash x_k$ and the string $x_g, x_{i_1}, ..., x_{i_n}, x_k$ has at most $d+1$ elements. We use the notation $x_g \vdash^d x_k$ to represent that x_k is reachable in d steps from x_g.

We now present our method for space-efficient reachability in labelled transition systems. $Reachable_T(x_a, x_b, d)$ returns true for a labelled transition system T, two states $x_a, x_b \in X$ and a distance d if x_b is reachable in at most d steps from x_a. If d is not given, we can set d to be $|X|$ in the worst case. When $Reachable_T(x_a, x_b, d)$ is run, we assume T is stored as a global constant.

We assume without loss of generality that the states in X can be ordered from x_0 to x_f. We use the shorthand notation $\{x_{m+1} := x_m + 1\ \}$ to indicate that x_m is incremented to represent the next state in the ordered list of states $\{x_0, ..., x_f\}$. In order to avoid unnecessary complication, we do not concern ourselves with the specifics of the encoding of the state transition information. For many common large state-space systems such as linear bounded automata, Petri nets and modular discrete-event systems, this information can be encoded intuitively in a computationally efficient manner so we can disregard this detail.

Algorithm 1 $Reachable_T(x_a, x_b, d)$

return_flag := *FALSE*
If $(x_a = x_b)$ *Then* $\{$*return_flag* := *TRUE*$\}$
Else
If $(x_a \vdash x_b) \wedge (d \geq 1)$ *Then* $\{$*return_flag* := *TRUE*$\}$
Else
If $(d > 1)$
$\{$
$x_m := x_0$

last_loop := FALSE

do

{

If $Reachable_T(x_a, x_m, \lceil d/2 \rceil)$ *Then*

{ *If* $Reachable_T(x_m, x_b, \lfloor d/2 \rfloor)$ *Then* {*return_flag := TRUE*} }

If $(x_m = x_f)$ *Then* {*last_loop := TRUE* } *Else* $\{x_m := x_m + 1$ }

}

while $(\neg$*return_flag* $\wedge \neg$*last_loop*)

}

RETURN return_flag

Now that the algorithm has been presented we need to prove it actually accomplishes the task it was written for.

Theorem 1 *Given two states* x_a *and* x_b *in a labelled transition system* T *and a distance* d, $Reachable_T(x_a, x_b, d)$ *returns true if and only if* x_b *can be reached from* x_a *in* d *steps according to the transition rules of* T.

Proof:

Case 1 *For the trivial case of* $d = 0$, *it should be apparent from the statement of the algorithm that* $Reachable_T(x_a, x_b, d)$ *returns true if and only if* $x_a = x_b$.

Case 2 *We demonstrate the case of* $d \geq 1$ *by generalized induction on* d.

We use for the base of the induction that $d = 1$. *For* $d = 1$ *it should be apparent from the statement of the algorithm that* $Reachable_T(x_a, x_b, d)$ *returns true if and only if* $x_a \vdash x_b$.

We take as the generalized induction hypothesis that $Reachable_T(x_a, x_b, d')$ *returns true if and only if* x_b *can be reached from* x_a *in at most* d' *steps for some* $d' \leq d$.

For the induction step we show that given the induction hypothesis, $Reachable_T(x_a, x_b, (d+1))$ *returns true if and only if* x_b *can be reached from* x_a *in at most* $(d+1)$ *steps.*

Notice from the the statement of the algorithm that given $d > 1$, $Reachable_T(x_a, x_b, (d+1))$ *returns true if and only if there exists a state* x_m *such that* $Reachable_T(x_a, x_m, \lceil (d+1)/2 \rceil)$ *and* $Reachable_T(x_m, x_b, \lfloor (d+1)/2 \rfloor)$.

$Reachable_T(x_a, x_m, \lceil (d+1)/2 \rceil)$ *returns true if and only if* x_m *can be reached from* x_a *in* $\lceil (d+1)/2 \rceil$ *steps.*

$Reachable_T(x_m, x_b, \lfloor (d+1)/2 \rfloor)$ *returns true if and only if* x_b *can be reached from* x_m *in* $\lfloor (d+1)/2 \rfloor$ *steps.*

x_m *can be reached from* x_a *in* $\lceil (d+1)/2 \rceil$ *steps and* x_b *can be reached from* x_m *in* $\lfloor (d+1)/2 \rfloor$ *steps if and only if* x_b *can be reached from* x_a *in at most* $\lceil (d+1)/2 \rceil + \lfloor (d+1)/2 \rfloor$ *steps.*

Notice that $\lceil (d+1)/2 \rceil + \lfloor (d+1)/2 \rfloor = (d+1)$. *Therefore for* $d > 1$ *with these facts,* $Reachable_T(x_a, x_b, (d+1))$ *returns true if and only if* x_b *can be reached from* x_a *in at most* $(d+1)$ *steps. This completes our generalized proof by induction.*

■

2.1 Space and Time Complexity

We now discuss the space and time complexity of Algorithm 1. Let us assume that each state can be uniquely represented using space q. Given a set of states X it should be readily apparent that q is at least $\log(|X|)$. As mentioned earlier, we assume that we can verify $x_a \vdash x_b$ and $x_a = x_b$ for $\{x_a, x_b\} \subseteq X$ in a computationally trivial manner, but for the sake of discussion in this subsection we assume we can verify these properties in space v_s and time v_t.

We are only concerned with worst-case analysis of Algorithm 1 so we disregard input states x_a and x_b passed to the algorithm and always assume worst case computations. This means that we assume the recursive calls of Algorithm 1 are always made if possible. With these assumptions, the distance measurement d is the only input variable that we concern ourselves with. We therefore only analyze worst-case time and space complexity of $Reachable(d)$.

Theorem 2 *The space complexity of* $Reachable(d)$ *is in* $O(q \log(d))$.

Proof: We use $Space_R(d)$ to denote the space required by $Reachable(d)$ for recursion. During the execution of Algorithm 1, if the while loop is executed, the algorithm needs to store the three states x_a,x_b and x_m, the integer d and the two boolean values $last\_loop$ and $return\_flag$. x_a,x_b and x_m can be stored in space q, d can be stored in space $log(d)$ and the binary variables can be stored in two bits. There are at most three tests of $x_a \vdash x_b$ or $x_a = x_b$ which are performed in space $3v_s$, but this space is reclaimed before the recursive calls to $Reachable$ are made and therefore this quantity does not affect memory required for the recursion. Therefore, $Reachable(d)$ needs space $3q + \log(d) + 2$ to be saved during each recursive call.

Therefore,

$Space_R(d) \leq (3q + \log(d) + 2) + \max\{Space_R(\lceil d/2 \rceil), Space_R(\lfloor d/2 \rfloor)\}$.

It should be apparent that $Space_R(d)$ is always increasing. Therefore,

$\max\{Space_R(\lceil d/2 \rceil), Space_R(\lfloor d/2 \rfloor)\} = Space_R(\lceil d/2 \rceil)$ and

$Space_R(d) \leq (3q + \log(d) + 2) + Space_R(\lceil d/2 \rceil)$.

Without loss of generality we may assume that $\log(d) \leq q$ because the maximum path length is equal to the size of the state-space.

Therefore,

$Space_R(d) \leq (4q + 2) + Space_R(\lceil d/2 \rceil)$.

So for d a power of 2,

$Space_R(d) \leq (4q + 2) + Space_R(d/2)$

Solving this recurrence, we find

$Space_R(d) \in O(q \log(d))$.

Therefore, with the three tests of $x_a \vdash x_b$ or $x_a = x_b$ which are performed in space $3v_s$, the space complexity of $Reachable(d)$ is in $O(\max\{q \log(d), v_s\})$. In almost all real-world examples we can test $x_a \vdash x_b$ and $x_a = x_b$ in space $O(q)$. Therefore, with our assumption that testing $x_a \vdash x_b$ and $x_a = x_b$ are computationally trivial, the space complexity of $Reachable(d)$ is in $O(q \log(d))$. ■

We know that d is at most the size of the state space, so $\log(d) \leq q$. This prompts the following corollary.

Corollary 1 *Given that state information can be encoded in space q, the space complexity of Algorithm 1 is in $O(q^2)$*

We now analyze the time complexity of Algorithm 1. Unfortunately there is no free lunch in our reduction of computation space for testing reachability. The basis of our method in Algorithm 1 is that we trade extra computation time to get a reduction in computation space. In the worst case at every level of recursion Algorithm 1 enumerates over all states in X.

Theorem 3 *The time complexity of $Reachable(d)$ is in $O\left(d^{\lg(2|X|)}\right)$.*

Proof: We use $Time_R(d)$ to denote the time required for $Reachable(d)$. During the execution of Algorithm 1, in the worst case, the x_m enumerate over all possible states in X. This means there are $|X|$ recursive calls to $Reachable(\lceil d/2 \rceil)$ and $Reachable(\lfloor d/2 \rfloor)$. Besides the recursion, there are also 3 tests of $x_a \vdash x_b$ or $x_a = x_b$ which are performed in at most time $3v_t$

Therefore,

$Time_R(d) \leq 3v_t + |X| \left(Time_R(\lceil d/2 \rceil) + Time_R(\lfloor d/2 \rfloor)\right)$

It should be apparent that $Time_R(d)$ is always increasing. Therefore,

$\max\{Time_R(\lceil d/2 \rceil), Time_R(\lfloor d/2 \rfloor)\} = Time_R(\lceil d/2 \rceil)$ and

$Time_R(d) \leq 3v_t + 2|X|Time_R(\lceil d/2 \rceil)$.

Using the master theorem [6] and our assumption that v_t is insignificant compared to $n^{\lg(2|X|)}$, we know:

$Time_R(d) \in O\left(d^{\lg(2|X|)}\right)$.

Therefore, the time complexity of $Reachable(d)$ is in $O\left(d^{\lg(2|X|)}\right)$.

■

3 Coobservability

The concept of coobservability, defined below, is central to the existence of decentralized controllers for discrete-event systems [4], [16]. Coobservability captures the notion for decentralized control systems that if an illegal event is about to occur there is always a controller that knows to disable the illegal event and can disable the illegal event. Coobservability is part of the set of necessary and sufficient conditions for the existence of decentralized controllers in all known results in the literature for various decentralized architectures. Different architectures have different versions of coobservability, but the coobservability seen in Definition 1 below is for the "conjunctive" architecture discussed in [2] and [16]. The methods in this paper are easily applied to deciding other versions of coobservability (as seen in [19]) in a space efficient manner.

We now review the definition of coobservability seen in [16]. We assume the reader has a basic understanding of discrete-event system theory. For more information please consult the text [2]. Let Σ_{ci} and Σ_{oi} be the locally controllable and observable events respectively, for the local controllers $S_i, i \in \{1, ..., s\}$ for the system under consideration. Let $P_i : \Sigma^* \rightarrow \Sigma_{oi}^*, i \in \{1, ..., s\}$

be the corresponding natural projections for the observations of the local controllers that "erase" locally unobservable events (see [2]).

Definition 2 *A language K is* coobservable *with respect to M, P_i, and Σ_{ci}, $i = 1, ..., s$ if for all $t \in \overline{K}$ and for all $\sigma \in \Sigma_c = \cup_{i=1}^{s} \Sigma_{ci}$,*

$\left(t\sigma \notin \overline{K}\right)$ *and* $(t\sigma \in M) \Rightarrow$

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

Coobservability is the decentralized generalization of "observability" [10]. It is well known that observability can be decided in polynomial time [18], but this result does not hold for coobservability. Be aware that coobservability is not non-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.

It is shown in [14] that the problem of deciding coobservability for languages specified by finite-state automata is PSPACE-complete when the number of controllers is unbounded. PSPACE-complete problems are known to be at least as hard as NP-complete problems and are thought to be much more so. Although PSPACE-complete problems can be solved in a space-efficient manner, it is generally believed that they cannot be solved in a time-efficient manner.

3.1 Methods for Testing Coobservability

Rudie and Willems [15] present the current standard method for testing coobservability for two-controller systems. Given an uncontrolled system automaton G a specification automaton H, the two controller specifications Σ_{c1} and Σ_{c2} and the two projections P_1 and P_2, a nondeterministic finite-state automata $\mathcal{M}$ is constructed. The only marked state (the "dump" state) is reachable from the initial state in $\mathcal{M}$ if and only if $\mathcal{L}_m(H)$ is not coobservable

with respect to $\mathcal{L}(G)$, P_1, P_2, Σ_{c1} and $\Sigma c2$. Therefore, we can test if languages specified by finite-state automata are coobservable in polynomial time by performing a reachability test on $\mathcal{M}$. We do not present the details of the construction of $\mathcal{M}$ here, but the reader is encouraged to consult [15] for more information.

The $\mathcal{M}$-machine construction can be easily generalized so that we can reduce coobservability tests for n-controller systems to tests of reachability over a finite-state automaton. Unfortunately, the $\mathcal{M}$-machine construction suffers from the state explosion problem. In the worst case the size of the state space of the $\mathcal{M}$-machine is exponential in the number of controllers.

When dealing with the state explosion problem, space constraints are usually more problematic than time constraints. All current tools that we are aware of for testing coobservability perform a reachability test on $\mathcal{M}$ by storing all reachable states in memory in the worst case. This method is currently the most time-efficient test for coobservability. However, when dealing with the state explosion problem, space constraints are usually more problematic than time constraints [5]. We can easily write to the memory of a modern computer in very short time (on the order of fractions of a second), so it is very easy for us to fill up the usable memory of a computation device and therefore space constraints are the limiting factor when testing coobservability for any system that is more complicated than a toy example. It would therefore be advantageous and straightforward to apply our space-efficient reachability test to the $\mathcal{M}$-machine which is a special type of labelled transition system.

Suppose we are given a system $G = (X^G, \Sigma, x_0^G, \delta^G, X_m^G)$, a specification $H = (X^H, \Sigma, x_0^H, \delta^H, X_m^H)$, observerable event sets Σ_{o1},...,Σ_{os} and controllable event sets Σ_{c1},...,Σ_{cs}. The $\mathcal{M}$-machine constructed from this configuration has a state space that is an $(s+2)$-tuple $X^G \times X^H \times X^H \times ... \times X^H$. Given that the states of G and H can each be encoded in space q, the states of the

$\mathcal{M}$-machine can be encoded in space $(s+2)q$. Given that we can verify that one state follows in one step from another in G and H in a computationally efficient manner, we can also verify that one state follows in one step from another in the $\mathcal{M}$-machine in a computationally efficient manner (it should take at most $(s+2)$ times as much time and space).

Theorem 4 *Given H, G, P_1,...,P_s and Σ_{c1},...,Σ_{cs}, we can test if $\mathcal{L}_m(H)$ is coobservable with respect to $\mathcal{L}(G)$, P_1,...,P_s and Σ_{c1},...,Σ_{cs} in space $O((s+2)^2q^2)$ and time $O\left(\left(|X^G| * |X^H|^{(s+1)}\right)^{\lg\left(2|X^G|*|X^H|^{(s+1)}\right)}\right)$.*

Proof: We prove this result in two parts.

Space efficiency: We know that reachability of the marked state in the $\mathcal{M}$-machine can be tested in space $O((s+2)q\log(d))$. In the worst case $d \leq |X^G| * |X^H|^{(s+1)}$. We know that $\max\{\log(|X^G|), \log(|X^H|)\} \leq q$, so $\log(d) \leq (s+2)q$, which proves our result.

Time efficiency: We know that reachability of the marked state in the $\mathcal{M}$-machine can be tested in time $O\left(d^{\lg(2|X|)}\right)$. In the worst case $d = |X|$ and $|X| = |X^G| * |X^H|^{(s+1)}$. So, the time required to test coobservability is in $O\left(\left(|X^G| * |X^H|^{(s+1)}\right)^{\lg\left(2|X^G|*|X^H|^{(s+1)}\right)}\right)$. ■

We have greatly shortened the required space for testing coobservability at the expense of increased worst-case computation time. We find in our experimental results that the worst-case computation space and time is actually rather close to the average case requirements for space and time.

4 Discussion

It is possible for a computer to access all of its memory in a relatively quick amount of time, and when we can therefore fill up a computer's memory very fast but storing all system states in memory. If we could trade time

for memory, we could solve many more problems in a reasonable amount of time, which is why we present these methods. We have introduced a method based on the proof of Savitch's theorem for testing reachability in labelled transition systems. This method uses very little computation space but potentially a lot of time. We have used the new reachability method to test the coobservability of decentralized control systems. We have attempted to minimize the computation space required to solve coobservability problem by using much more computation time. However, we traded too much time for space as shown in several informal experiments we ran to test coobservability.

The algorithm outlined above for testing coobservability was implemented in the UMDES system produced at the University of Michigan. The code was tested on several simple systems such as the dining philosopher system as seen in [2]. We used the system as its specification (this specification is therefore trivially coobservable) and established one controller for each philosopher that can control and observe all events local to the philosopher. When the code was run on this system for two philosophers, the code required several minutes to run on a Sun workstation, but when the code was run for three philosophers, the program did not halt in less than 48 hours. This is very disappointing because even relatively small systems take a prohibitively long time to run. The traditional methods for testing coobservability took on the order of several minutes to run. This indicates that the new coobservability algorithm trades too much computation space for more computation time. Therefore, it would be advantageous to find a better tradeoff between time and space. This would be an interesting area for future research.

References

[1] 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.

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

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

[4] 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.

[5] E. Clarke. Forte 2002 tutorial. Personal communciation.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, 2001.

[7] G. Holzmann. An improved protocol reachability analysis technique. *Software - Practice and Experience*, 18(2):137–161, 1988.

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

[9] 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.

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

[11] 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.

[12] K. Rohloff and S. Lafortune. Supervisor existence for modular discrete-event systems. Preprint.

[13] 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.

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

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

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

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

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

[19] 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.