Automatic Verification of Parameterized Synchronous Systems*

E. Allen Emerson and Kedar S. Namjoshi

Department of Computer Sciences,
The University of Texas at Austin, U.S.A.

Abstract

Systems with an arbitrary number of homogeneous processes occur in many applications. The *Parameterized Model Checking Problem* (PMCP) is to determine whether a temporal property is true of every size instance of the system. We consider systems formed by a synchronous parallel composition of a single control process with an arbitrary number of homogeneous user processes, and show that the PMCP is decidable for properties expressed in an indexed propositional temporal logic. While the problem is in general PSPACE-complete, our initial experimental results indicate that the method is usable in practice.

1 Introduction

Systems with an arbitrary number of homogeneous processes occur in many contexts, especially in protocols for data communication, cache coherence, and classical synchronization problems. Current verification work on such systems has focussed mostly on verifying correctness for instances with a small number of processes. This does not indicate whether larger size instances are error-free, and so does not guarantee correctness in general. We are thus interested in methods that verify correctness for arbitrary size instances. Even though sometimes there is indeed a specific upper bound on the number of processes in a system, verifying such large size instances is intractable because of state explosion.

The general problem, then, is the *Parameterized Model Checking Problem* (PMCP): to determine whether a temporal property is true of every size instance of the the system. This is known to be undecidable in general [AK 86, Su 88]; however, it is decidable algorithmically for restricted classes [GS 92, EN 95], and there are methods with some degree of automation [Lu 84, ShG 89, KM 89, WL 89, V 93, CGJ 95]. This previous work (with the exception of [KM 89]) was oriented toward asynchronous systems.

We propose a fully automated approach to the PMCP for synchronous systems. We consider synchronous systems with a unique control process and an arbitrary number of homogeneous user processes. Each system is thus parameterized by the number of user processes. The processes are specified by labeled transition graphs, in which guards on each transition check the state of the control process as well as certain conditions on the global state. The correctness properties are expressed in an indexed propositional branching temporal logic, and are of the following types:

1. Over the control process : formulae of the form $\mathsf{A}h$ and $\mathsf{E}h$, where h is a linear-time formula with atomic propositions over control process states,

*This work was supported in part by NSF grant CCR 9415496 and SRC Contract 95-DP-388. The authors can be reached at `emerson, kedar@cs.utexas.edu` and at `http://www.cs.utexas.edu/users/{emerson, kedar}`.

2. Over all user processes: $\bigwedge_i \mathsf{A}h(i)$, and $\bigwedge_i \mathsf{E}h(i)$, where $h(i)$ is a linear-time formula with atomic propositions over control process states, and over user process states indexed with i.

3. Over every distinct pair of user processes : $\bigwedge_{i\neq j} \mathsf{A}h(i,j)$, and $\bigwedge_{i\neq j} \mathsf{E}h(i,j)$, where $h(i,j)$ is a linear-time formula with atomic propositions over control process states, and over user process states indexed with either i or j.

We show that the PMCP for the first type of formulae is decidable for this class of systems, and is PSPACE-complete. This decidability result is based on constructing an abstract graph in which *every* computation of *every* size instance of the system is represented by some path in the graph. However, the abstract graph may have "bad" paths that do not correspond to computations of any size instance. The heart of the algorithm is a method for identifying good paths in the abstract graph. This algorithm can be implemented in space polynomial in the size of the control and user processes. We show by a generic reduction that the PMCP is PSPACE-hard. As a result of the symmetry inherent in the system, the PMCP for the other types of formulae reduces to the PMCP for the first type. We have implemented this algorithm in SMV [McM92] and used it to check correctness of a bus arbitration protocol. Our initial experimental results indicate that the algorithm should be useful in practice.

Section 2 defines the system model and the logic used for expressing correctness properties. Section 3 describes the abstract graph representation, and Section 4 the algorithm for the PMCP for formulae of type (1). Section 5 shows the reduction of the PMCP for formulae of types (2) and (3) to the PMCP for formulae of type (1). Section 6 describes our implementation of the algorithm, and the application to the bus protocol. Section 7 concludes the paper with a discussion of related work.

2 The system model and logic

We refer to the collection of system instances formed by control process C and copies of a generic user process U as a (C,U) *family*. The control and user processes are specified as finite-state labeled transition graphs. We use the terms "process" and "labeled transition graph" interchangeably. For a process P, let S_P denote its set of states, R_P its transition relation, and ι_P its initial state[1].

The system *instance* of size n is a synchronous parallel composition of C with n copies of process U, and is denoted as $C \parallel U^n = C \parallel U_1 \parallel U_2 \ldots \parallel U_n$. U_i is the ith copy of U, which is obtained from U by uniformly subscripting the states of U with i as shown in the example below [2]:

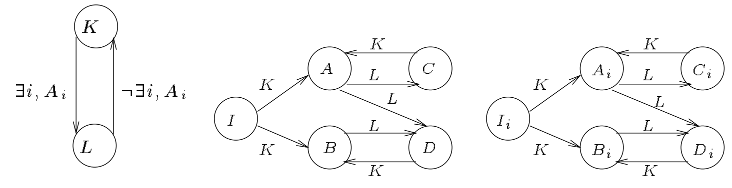


FIG 1a: The control process FIG 1b: The generic user process FIG 1c: The ith user process

Thus for all i,j, U_i and U_j are isomorphic up to re-indexing. Transitions in both C and U_i are labeled with *guards*. Every guard is a boolean combination of *users conditions*, which have the

[1]The results of this paper carry over for processes with a *set* of initial states.

[2]In this example, C has initial state K, and U has initial state I. Atomic propositions are identified with state names.

form $(\exists i\ \mathcal{E}(i))$, where $\mathcal{E}(i)$ is a boolean expression formed from atomic propositions over the states of C, and over the states of U_i. [3]

$\mathcal{G}_n$ denotes the global state transition graph of the instance of size n. A state s of $\mathcal{G}_n$ is written as an $(n+1)$-tuple $(c, u_1, \ldots, v_n)$, where c is the local state of C, and the $(i+1)$'th component of the tuple is the local state of U_i (for $i \in [1..n]$). The $(i+1)$'th component of s is denoted by $s(i)$. The initial state of $\mathcal{G}_n$ is $(\iota_C, (\iota_U)_1, \ldots, (\iota_U)_n)$. A transition (s, t) is in $\mathcal{G}_n$ iff

1. A transition of C from $s(0)$ to $t(0)$ is enabled in s, and
2. For all $i \in [1..n]$, a transition of U_i from $s(i)$ to $t(i)$ is enabled in s.

where a transition in a process is said to be enabled in a global state iff the corresponding guard is true when evaluated in that global state. We write $s \models g$ iff guard g is true in the global state s. $s \models (\exists i\ \mathcal{E}(i))$ iff for some $k \in [1..n]$, $\mathcal{E}(k)$ is true given the propositions that hold at $s(0)$ (the control state), and $s(k)$ (the state of process U_k). Boolean operators are handled in the standard manner. For a global state s, and state $a \in S_U$, we let $\#a(s) = |\{i \mid i \in [1..n] \wedge s(i) = a_i\}|$ (i.e., $\#a(s)$ is the number of user processes with local state a of the generic user process).

PLTL is the standard propositional linear temporal logic built up from atomic propositions, boolean connectives, and temporal operators G (always), F (sometime), X (next time), and U (until) [Pn 77, MP 92]. *CTL** is a branching temporal logic which extends PLTL by allowing the path quantifiers A (for all fullpaths) and E (for some fullpath). Many interesting correctness properties of parameterized systems can be expressed in one of the following forms:

1. Over the control process : formulae of the form $\mathsf{A}h$ and $\mathsf{E}h$, where h is a linear-time formula with atomic propositions over control process states,
2. Over all user processes: $\bigwedge_i \mathsf{A}h(i)$, and $\bigwedge_i \mathsf{E}h(i)$, where $h(i)$ is a linear-time formula with atomic propositions over control process states, and over states of U indexed with i.
3. Over all distinct pairs of user processes : $\bigwedge_{i \neq j} \mathsf{A}h(i,j)$, and $\bigwedge_{i \neq j} \mathsf{E}h(i,j)$, where $h(i,j)$ is a linear-time formula with atomic propositions over control process states, and over states of U indexed with either i or j.

The formal semantics of these logics is defined in the usual way [Em 90, BCG 89, ES 95], and we write $M, s \models f$ to mean that formula f is true in structure M at state s.

3 The abstract model

For a given (C, U) family, we construct an abstract process $\mathcal{A}$ which includes all computations of every size instance of the family. Intuitively, a state (c, S) of $\mathcal{A}$ *represents* any global state in which the control process is in state c, there is at least one user process in every user state in S, and no user process is in a state in $S_U \backslash S$. Transitions from a state (c, S) represent transitions enabled from global states that are represented by (c, S). Each such transition has a label which represents moves of individual processes.

Formally, let $\Lambda = 2^{S_U \times S_U} \backslash \{\emptyset\}$ be the set of edge labels. $\mathcal{A}$ is defined by a labeled transition graph, where

[3] There are two interesting special cases : (a) The guards in U_i involve only propositions over states of C. The control process may then be viewed as controlling the execution of the user processes. (b) The control process is a copy of the user process, and can be written as U_0. Then $C \parallel U^n$ is isomorphic to U^{n+1}. Our method applies in general, but often finds interesting application in these special cases.

1. $S_{\mathcal{A}} = S_C \times (2^{S_U} \backslash \{\emptyset\})$ is the set of states,
2. $R_{\mathcal{A}} \subseteq S_{\mathcal{A}} \times \Lambda \times S_{\mathcal{A}}$ is the set of transitions,
3. $\iota_{\mathcal{A}} = (\iota_C, \{\iota_U\})$ is the initial state.

To make the correspondence between global states and abstract states precise, we define families of abstraction functions $\{\phi_i\}$, $\{\psi_i\}$, where $\phi_n : S_{\mathcal{G}_n} \to S_{\mathcal{A}}$, and $\psi_n : S_{\mathcal{G}_n} \times S_{\mathcal{G}_n} \to \Lambda$. For a state $s \in S_{\mathcal{G}_n}$, $\phi_n(s) = (s(0), \{a \mid (\exists i \in [1..n]\, s(i) = a_i)\})$, and for a pair (s,t), $\psi_n(s,t) = \{(a,b) \mid \exists i \in [1..n]\, s(i) = a_i \wedge t(i) = b_i\}$. Then (c, S) represents $s \in \mathcal{G}_n$ iff $(c, S) = \phi_n(s)$.
For a guard g, and state (c, S) of $\mathcal{A}$, we define $(c, S) \,||\!-\, g$ as $(c, u_1, \ldots, v_k) \models g$, where $S = \{u \ldots v\}$ (for some ordering $u \ldots v$ of the elements of S) and $|S| = k$. The following proposition relates $\models$ and $||\!-$:

Proposition 1 *For any n, and any $s \in \mathcal{G}_n$, if $(c, S) = \phi_n(s)$, then for every guard expression g, $s \models g$ iff $(c, S) \,||\!-\, g$.* □

The set of transitions is defined as follows: A tuple $((c, S), X, (c', S')) \in R_{\mathcal{A}}$ iff

1. $(\exists p\; c \xrightarrow{p} c' \in R_C \wedge (c, S) \,||\!-\, p)$ (A transition from c to c' is enabled for the control process),
2. $(\forall a, b\; (a, b) \in X \Rightarrow a \in S \wedge b \in S' \wedge (\exists q\; a \xrightarrow{q} b \in R_U \wedge (c, S) \,||\!-\, q))$. (For every pair (a, b) in X, there is an enabled transition from a to b in the user process).
3. X is total on S, and X^{-1} is total on S', (Every state in S has a successor in S', and every state in S' has a predecessor in S).

Definition 1 *A* path *in $\mathcal{G}_n$ is a sequence of states such that adjacent states are in the global transition relation of $\mathcal{G}_n$.* □

Definition 2 *A* path *in $\mathcal{A}$ is a sequence starting at a state, with alternating states and transition labels such that for every $s, s' \in S_{\mathcal{A}}$ and $X \in \Lambda$, sXs' occurs in the sequence only if $(s, X, s') \in R_{\mathcal{A}}$.*
□

Define a family of functions $\{\gamma_i\}$ such that γ_n maps from paths in $\mathcal{G}_n$ to paths in $\mathcal{A}$ by $(\gamma_n(\sigma))_{2i} = \phi_n(\sigma_i)$, and $(\gamma_n(\sigma))_{2i+1} = \psi_n(\sigma_i, \sigma_{i+1})$ for all $i \in \mathbf{N}$.

Proposition 2 *For every path σ in $\mathcal{G}_n$, $\gamma_n(\sigma)$ is a path in $\mathcal{A}$.* □

It follows from Proposition 2 that if $\mathcal{A}$ satisfies a linear temporal formula over all paths, then so does every size instance of the family. However, if the formula is false for some path in $\mathcal{A}$, it does not follow that it is false for some instance, as not every path in $\mathcal{A}$ arises from a corresponding path in some instance; those that do are called "good".

Definition 3 *A path ρ in $\mathcal{A}$ is* good *iff $\exists n\, \exists \sigma \in \mathcal{G}_n\; \gamma_n(\sigma) = \rho$.* □

Definition 4 *A path σ' in $\mathcal{G}_i$ covers a path σ in $\mathcal{G}_j$ $(i \geq j)$ iff $\gamma_i(\sigma') = \gamma_j(\sigma)$, and for every $k \in \mathbf{N}$, $a \in U$, $\#a(\sigma'_k) \geq \#a(\sigma_k)$.* □

Lemma 1 *(Covering Lemma) For $n' \geq n$, every path in $\mathcal{G}_n$ has a covering path in $\mathcal{G}_{n'}$.*

Proof

Let σ be a path in $\mathcal{G}_n$. Define σ' in $\mathcal{G}_{n'}$ by the following: $\sigma'_k(0) = \sigma_k(0)$, and for $i \in [1..n']$, $\sigma'_k(i) = a_i$, where a is such that if $i \; mod \; n \neq 0$, then $\sigma_k(i \; mod \; n) = a_{i \; mod \; n}$, and and if $i \; mod \; n = 0$, then $\sigma_k(n) = a_n$.

It follows that $\phi_{n'}(\sigma'_k) = \phi_n(\sigma_k)$, and that $\#a(\sigma'_k) \geq \#a(\sigma_k)$ for all $a \in U$. To complete the proof, we need to show that $\psi_{n'}(\sigma'_k, \sigma'_{k+1}) = \psi_n(\sigma_k, \sigma_{k+1})$. Since (σ_k, σ_{k+1}) is a transition of $\mathcal{G}_n$, there exist guards $p, q_1, \ldots q_n$, such that $\sigma_k(0) \xrightarrow{p} \sigma_{k+1}(0)$ is the transition of the control process, and $\sigma_k(i) \xrightarrow{q_i} \sigma_{k+1}(i)$ is the transition of process U_i, for $i \in [1..n]$. As $\phi_{n'}(\sigma'_k) = \phi_n(\sigma_k)$, from Proposition 1, transition q_i is enabled for user processes with indices $j = i \,(mod \, n)$ in σ'_k. The resulting state is σ'_{k+1}. It follows that $\psi_{n'}(\sigma'_k, \sigma'_{k+1}) = \psi_n(\sigma_k, \sigma_{k+1})$, and so σ' is a path of $\mathcal{G}_{n'}$ that covers σ. $\square$

Lemma 2 *Every finite path of $\mathcal{A}$ is good.*

Proof

The proof is by induction on the number of states in the path. Suppose the path is a single state s. Let $s = (c, S)$, and let $n = |S|$. Consider the state $r = (c, u_1, \ldots v_n)$ in $\mathcal{G}_n$, where $S = \{u \ldots v\}$. As $\phi_n(r) = s$, the claim is true of paths with one state. Suppose that it is true for all paths with at most m states, for $m \geq 1$, and let ρ be a path with $m+1$ states. Then, $\rho = \rho' X t$, where if s is the last state in ρ', then $(s, X, t) \in R_{\mathcal{A}}$. By inductive hypothesis, for some n', there is a path $\sigma' \in \mathcal{G}_{n'}$ such that $\gamma_{n'}(\sigma') = \rho'$. Let r' be the last state in σ'.

For each $a \in U$, let $m_a = |\{b \mid (a,b) \in X\}|$. If for some a, $m_a > \#a(r')$, one can construct a path covering σ' such that if u is the final state on that path, then $m_a \leq \#a(u)$. Repeating this construction for each user state a for which it is necessary, we obtain, for some n, a path σ in $\mathcal{G}_n$ such that σ covers σ', and for every a, $m_a \leq \#a(r)$, where r is the last state on σ.

As $m_a \leq \#a(r)$ for each a, one can associate at least one index $i \in [1..n]$ with each pair (a, b) in X. For every pair (a, b) in X, there is an enabled transition from a to b in the user process. Thus, there is a state $u \in \mathcal{G}_n$ generated by performing the enabled transition from a_i to b_i in each process U_i where index i is associated with the pair (a, b), and the enabled transition for the control process. It is easy to verify that $\phi_n(u) = t$, and hence, σu is a path in $\mathcal{G}_n$ such that $\gamma_n(\sigma u) = \rho$. $\square$

4 Verifying properties of the control process

The properties of the control process are of the form $\mathsf{A}h$ or $\mathsf{E}h$, where h is a linear-time temporal formula with atomic propositions over the states of C. To model-check such a property, we follow the automata-theoretic approach of [VW 86] : To determine if $M, \iota_M \models \mathsf{E}h$, construct a Büchi automaton $\mathcal{B}_h$ for h, and check that the language of the product Büchi automaton of M and $\mathcal{B}_h$ is non-empty (cf. [LP85]). The check for the property $\mathsf{A}h$ is easily reduced to that for the earlier case by noting that $M, \iota_M \models \mathsf{A}h$ iff $M, \iota_M \not\models \mathsf{E}\neg h$.

We say that formula $\mathsf{A}h$ is *universal* iff it is true for every size instance of the family. To determine if $\mathsf{A}h$ is universal, we model check it over the abstract graph, by constructing a Büchi automaton $\mathcal{B}$ for $\neg h$, and forming the product Büchi automaton $\mathcal{M}$ of $\mathcal{A}$ and $\mathcal{B}$. $\mathcal{B}$ accepts a computation σ labeled with propositions over states of C iff there is a run of $\mathcal{B}$ on σ such that a "green" state of $\mathcal{B}$ is entered infinitely often. An *accepting* path in $\mathcal{M}$ is one which starts in an initial state, and along which a green state occurs infinitely often. For a path δ in $\mathcal{M}$, let $\delta_{\mathcal{A}}$ be its projection on $\mathcal{A}$. A path in $\mathcal{M}$ is *good* iff its projection on $\mathcal{A}$ is a good path in $\mathcal{A}$.

Theorem 1 *Formula* $\mathsf{A}h$ *is not universal iff there is an accepting good path in* $\mathcal{M}$.

Proof

Suppose δ is an accepting good path in $\mathcal{M}$. As $\delta_{\mathcal{A}}$ is good, for some n, there is a path in $\mathcal{G}_n$ that matches $\delta_{\mathcal{A}}$ on the sequence of states of C, and is hence accepted by $\mathcal{B}$. Therefore, $\mathsf{A}h$ is false in $\mathcal{G}_n$, and hence is not universal.

In the other direction, if $\mathsf{A}h$ is not universal, then for some n, there is a path σ in $\mathcal{G}_n$ from the initial state that is accepted by $\mathcal{B}$. From Lemma 2, $\gamma_n(\sigma)$ is a path in $\mathcal{A}$, which is good by construction. The sequence of states of C in $\gamma_n(\sigma)$ is the same as in σ, hence there is a run of $\mathcal{B}$ on $\gamma_n(\sigma)$ that forms an accepting good path in $\mathcal{M}$. □

4.1 Finding accepting good paths in $\mathcal{M}$

From Theorem 1, to determine if $\mathsf{A}h$ is not universal, we have to check if there is an accepting good path in $\mathcal{M}$. The following lemmas provide the basis for a PSPACE algorithm to check universality. For a cycle δ in $\mathcal{M}$, we say that δ is good iff the infinite path δ^ω is good.

Lemma 3 *There is an accepting good path in* $\mathcal{M}$ *iff there are finite paths* α *and* β *in* $\mathcal{M}$, *such that*

1. α *is a path from the initial state to a green state* s, *and*

2. β *is a good cycle starting at* s. □

Intuitively, a cycle in $\mathcal{M}$ is good if, starting at some global state which maps to a state in the cycle, there is no transition in that cycle that causes the count of processes in a specific local state to be "drained" (i.e. decreased monotonically) as the sequence of transitions along the cycle is executed repeatedly. For example, a self-loop with the transition label $\{(a,b)\}$ will decrease the count of processes in state a with every execution of the transition, while one with transition label $\{(a,b),(b,a)\}$ may not. Notice that in the latter case, there is a cycle $a \to b \to a$ in the transition label considered as a graph. This presence of cycles in the transition labels is the intuition behind the characterization of good cycles of $\mathcal{M}$.

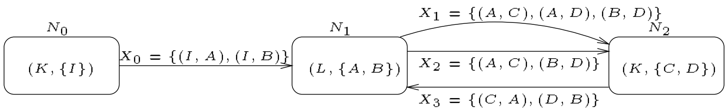


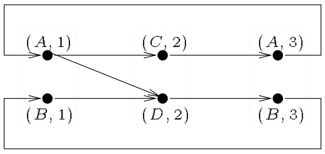
FIG 2 : A portion of the abstract graph for the example in FIG 1.

To determine if such cycles are present, we resolve a cycle in $\mathcal{M}$ into a "threaded graph" (cf. [ES 95]) which shows explicitly which local user state in an abstract state is driven into which other local user state in the next abstract state. This information is obtained from the transition label. The threaded graph is defined below:

Definition 5 Threaded Graph

Let δ be a finite path in $\mathcal{M}$ with m states. Let the ith state of δ be called s_i, and the ith transition be called X_i. For a state $s = ((c, S), u)$ of $\mathcal{M}$, let $Ustates(s) = S$. Define H_δ to be the following graph :
$V(H_\delta) = \{(x, i) \mid i \in [1..m] \wedge x \in Ustates(s_i)\}$
$E(H_\delta) = \{((x, i), (y, i+1)) \mid i \in [1..m-1] \wedge (x, y) \in X_i\}$
If δ is a cycle, then $s_1 = s_m$. Define G_δ to be the graph where $V(\mathcal{G}_\delta) = V(H_\delta)$, and $E(\mathcal{G}_\delta) = E(H_\delta) \cup \{((x, m), (x, 1)) \mid x \in Ustates(s_1)\}$.
A graph is *isolated* iff its edge set is empty. For any directed graph G, let $maxscc(G)$ be the graph representing the decomposition of G into its maximal strongly connected components (scc's).
$V(maxscc(G)) = \{C \mid C \textit{ is a maximal strongly connected component of } G\}$
$E(maxscc(G)) = \{(C, D) \mid \exists s \in C, t \in D \ (s, t) \in E(G)\}$
We refer to vertices of $maxscc(G)$ as max-scc's. It is a fact that $maxscc(G)$ is acyclic for any graph G. For any max-scc D in $maxscc(G)$, define max-scc C to be *above* D if there is a path in $maxscc(G)$ from C to D. □

The following figure shows the threaded graphs for the cycles $N_1 \xrightarrow{X_1} N_2 \xrightarrow{X_3} N_1$ and $N_1 \xrightarrow{X_2} N_2 \xrightarrow{X_3} N_1$ in figure 2:



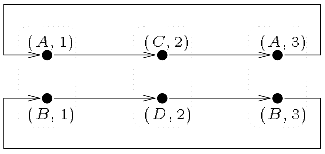


FIG 3a : Threaded graph G_δ for the first cycle FIG 3b : Threaded graph G_δ for the second cycle

Lemma 4 *δ is a good cycle in $\mathcal{M}$ iff $maxscc(G_\delta)$ is isolated.*

Proof Sketch
(LHS $\Rightarrow$ RHS): Suppose that $maxscc(G_\delta)$ is not isolated but δ is good. Hence, there are max-scc's C and D such that some pair of vertices (x, i) in C and (y, j) in D is connected in G_δ. For any n, consider an infinite path σ in $\mathcal{G}_n$ such that $\gamma_n(\sigma) = \delta^\omega_{\mathcal{A}}$. We say that a process with index $l \in [1..n]$ and local state a_l is in component F at the kth state in σ iff $(a, k) \in F$.
Let m be the number of states in δ. Starting with the ith transition in σ, at every mth successive transition, at least one of the processes in C, say one with index l, must change its local state from x_l to y_l. Thus, the count of processes in components above D decreases at each such step. As the max-scc decomposition is acyclic, this number cannot increase. Thus, eventually, the number of processes in components above D must become negative, which is impossible as σ is infinite. Hence, δ is not good.
(RHS $\Rightarrow$ LHS): Suppose that $maxscc(G_\delta)$ is isolated. For each max-scc of G_δ, construct a cycle in G_δ that includes each edge in that component at least once. For each $a \in U$, let m_a be the number of occurrences of the vertex $(a, 1)$ in the set of cycles. Let $n = \Sigma_{a \in U} m_a$. We will construct a path σ in $\mathcal{G}_n$ such that $\gamma_n(\sigma) = \delta^\omega_{\mathcal{A}}$. The idea behind the construction is to allot a set of processes for each constructed cycle, and to ensure that each transition of every process is along the cycle that it is alloted to.
The inductive assumption is that at the ith step ($i < m$), a path σ' has been constructed such that $\gamma_n(\sigma')$ is the prefix of $\delta_{\mathcal{A}}$ up to the ith state, and if s is the last state of σ', then $\#a(s)$ is the

number of occurrences of (a, i) in the set of constructed cycles. Hence, after m steps, the last state s_m is a permutation of the first state s_1. Repeating the construction at most n times produces a path σ with last state identical to s_1, and such that $\gamma_n(\sigma) = \delta^k_{\mathcal{A}}$. Thus, $\gamma_n(\sigma^\omega) = (\delta^k_{\mathcal{A}})^\omega = \delta^\omega_{\mathcal{A}}$, and so δ is a good cycle. $\square$

For a finite path α with m states in $\mathcal{A}$ define $\overline{\alpha}$ to be the relation over $S_U \times S_U$ where $(a, b) \in \overline{\alpha}$ iff there is a path from $(a, 1)$ to (b, m) in H_α. We say that relation R is cyclic iff for every edge in the graph of R, there is there is a cycle in the graph that includes that edge.

Lemma 5 *For a cycle δ in $\mathcal{M}$, $maxscc(G_\delta)$ is isolated iff $\overline{\delta}$ is cyclic.* $\square$

Theorem 2 *Formula* $\mathsf{A}h$ *is not universal iff there is a finite path in $\mathcal{M}$ from an initial state to a green state and a cycle δ from that state such that $\overline{\delta}$ is cyclic.*

Proof Follows from Theorem 1 and Lemmas 4, 5. $\square$

Let L be the maximum length of a guard in C and U processes. Note that $L \leq |C| + |U|$.

Theorem 3 *There is a nondeterministic algorithm to decide if a temporal property over computations of C is not universal which uses space $O(|S_U|^2 + log(|S_C||S_{\mathcal{B}}|) + L)$. The algorithm uses space logarithmic in the size of $\mathcal{M}$.*

Proof

By Theorem 2, a property $\mathsf{A}h$ is not universal iff there is a finite path in $\mathcal{M}$ to a green state and a following cycle δ from that state such that $\overline{\delta}$ is cyclic. The algorithm "guesses" a path to a green state, and a cycle δ from it, recording only the current state of $\mathcal{M}$, and $\overline{\delta'}$ for the prefix δ' of δ that has been examined. As $\overline{(\alpha; X; s)} = \overline{\alpha} \circ X$, $\overline{\delta}$ can be computed incrementally.

Recording a state of $\mathcal{M}$ takes space $(log(|S_C||S_{\mathcal{B}}|) + |S_U|)$. Computing a successor state can be done in space proportional to $(log|S_{\mathcal{B}}| + log|S_C| + log|S_U| + L)$ (as this requires checking if $(c, S) \,||\!-\, p$ for guards p). Storing $\overline{\delta'}$ takes space $|S_U|^2$, and checking if $\overline{\delta}$ is cyclic can be done within the same space bound. Thus, the overall space usage is $O(|S_U|^2 + log(|S_C||S_{\mathcal{B}}|) + L)$. $\square$

Remark 1 *There are two special cases where the algorithm can be optimized. If the user processes are deterministic, every cycle δ in $\mathcal{M}$ is good (as G_δ must be isolated). If the correctness property is a safety property, the algorithm need check only finite accepting paths, which are good by Lemma 2. In both cases, the check for good cycles can be eliminated, which is a substantial saving.* $\square$

A reduction from a generic PSPACE Turing Machine shows that checking if $\mathsf{AG}\neg accept$ is not universal is PSPACE-hard.

Theorem 4 *Deciding if a property over computations of C is not universal is PSPACE-complete.*

Corollary 1 *Deciding if a property over computations of C is universal is PSPACE-complete.*

The algorithm given above for determining if a property is not universal is nondeterministic and uses polynomial space. So, using Savitch's construction, there is a deterministic algorithm with time complexity $O(2^{k(|S_U|^2 + log(|S_C||S_{\mathcal{B}}|)+L)^2})$ for some k. We present a "natural" deterministic algorithm with the same worst case time complexity in $|S_U|$. Let $K = |S_{\mathcal{M}}| \times 2^{|S_U|^2}$. The algorithm follows from this observation:

Proposition 3 *If ρ is a finite path in $\mathcal{M}$ from s to t of length greater than K, then there is a path δ from s to t in $\mathcal{M}$ of length at most K such that $\overline{\rho} = \overline{\delta}$.*

Proof

Define an equivalence relation on states s of ρ by $s_i \equiv s_j$ iff $s_i = s_j$ and $\overline{X_0 \circ X_1 \ldots X_{i-1}} = \overline{X_0 \circ X_1 \ldots X_{j-1}}$. Clearly there are at most K equivalence classes. So if the length of ρ is greater than K, there must be distinct indices i and j such that $s_i \equiv s_j$. Assume that $i < j$. Then the path ρ' formed by appending the suffix from s_j to the prefix up to s_i is a path in $\mathcal{M}$ that is shorter than the path ρ, and is such that $\overline{\rho'} = \overline{\rho}$. Repeating this construction a finite number of times produces a path δ with the desired properties. $\square$

Theorem 5 *There is a deterministic algorithm to determine if a property is not universal with exponential worst case time complexity in* $|S_U|$.

Proof Sketch

From Proposition 3, it suffices to look for cycles (in Theorem 2) of length at most K. This can be done using an iterative squaring of the transition relation of $\mathcal{M}$, with overall time complexity exponential in $|S_U|$. $\square$

5 Symmetry reduction

Let π be a permutation over the set $\{0 \ldots n\}$ that fixes 0. For a state $s = (c, u_1, \ldots, v_n)$ in $\mathcal{G}_n$, the permuted state $\pi(s)$ is defined by $(\pi(s))(i) = a_i$ iff $s(\pi^{-1}(i)) = a_{\pi^{-1}(i)}$, for $i \in [0..n]$. For example, the state (c, u_1, v_2, w_3) under the permutation $\pi = \{(1 \rightarrow 2), (2 \rightarrow 3), (3 \rightarrow 1)\}$ becomes (c, w_1, u_2, v_3). As $\phi_n(\pi(s)) = \phi_n(s)$, from Proposition 1, the truth value of any guard is the same in both s and $\pi(s)$. Hence there is complete symmetry among the user processes in any size instance of a (C, U) family, and the PMCP for formulae of type (2) and (3) reduces that for formulae of type (1). The following lemmas are based on those in [ES 93, CFJ 93] (cf. [ID 93]) Let $f(i)$ be a CTL^* formula with propositions over the states of C and over the states of U indexed with i, and let $f(i,j)$ be a CTL^* formula with propositions over the states of C and over the states of U indexed with either i or j.

Lemma 6 *For* $n \geq 1$, $\mathcal{G}_n, \iota_{\mathcal{G}_n} \models \bigwedge_i f(i)$ *iff* $\mathcal{G}_n, \iota_{\mathcal{G}_n} \models f(1)$.

Lemma 7 *For* $n \geq 2$, $\mathcal{G}_n, \iota_{\mathcal{G}_n} \models \bigwedge_{i \neq j} f(i,j)$ *iff* $\mathcal{G}_n, \iota_{\mathcal{G}_n} \models f(1,2)$.

Let $C|U$ be the process where $S_{C|U} = S_C \times S_U$, and $(c, u) \stackrel{p' \wedge q'}{\rightarrow} (c', u') \in R_{C|U}$ iff $c \stackrel{p}{\rightarrow} c' \in R_C$ and $u \stackrel{q}{\rightarrow} u' \in R_U$, and p' (similarly q') is p (q) with every global condition $(\exists i\ \mathcal{E}(i))$ replaced with $\mathcal{E}(0) \vee (\exists i\ \mathcal{E}(i))$, where propositions labeled by 0 refer to the state of U in $C|U$.

Theorem 6 *A property of the form* $\bigwedge_i \mathsf{A}h(i)$ *is universal for a* (C, U) *family iff* $\mathsf{A}h(0)$ *is universal for the control process in the family* $(C|U, U)$.

Theorem 7 *A property of the form* $\bigwedge_{i \neq j} \mathsf{A}h(i,j)$ *is universal for a* (C, U) *family iff* $\mathsf{A}h(0, 0')$ *is universal for the control process in the family* $((C|U|U), U)$.

6 Applications

We have implemented this algorithm to verify a bus arbitration protocol based on the SAE J1850 draft standard [SAE 92] for automobile applications. This is a protocol where many microcontrollers can transmit symbols along a shared single-wire bus in a car. As a consequence of this restriction, symbols are encoded by the width of a pulse. Nodes on the bus may begin transmitting different messages simultaneously; only the node with the highest priority message should complete transmission after the arbitration process. Symbol 0 has priority over symbol 1, and priority between messages over the alphabet $\{0, 1\}$ is determined lexicographically. The microcontrollers are modeled as user processes, and the bus as the control process. The property which we have verified, using the result in Theorem 7, is that whenever two users begin simultaneous transmission of symbols 0 and 1 respectively, the user transmitting 1 continues transmission unless it loses arbitration. Hence, messages with lower priority cannot prevail over higher priority messages.

We implemented the algorithm by generating SMV [McM92] code to describe the abstract process transitions, given a description of the next-state relation of the user and control processes. Since the correctness property is a safety property, we were able to simplify the implementation as described following Theorem 3. Each user process has about 50 states, while the control process together with the automaton for the property has about 400 states. Verification took less than a minute on a SPARC 5. We emphasize that this establishes correctness of the bus protocol for an arbitrary number of attached microcontrollers.

7 Conclusions and Related Work

A variety of positive results on the PMCP have been obtained previously. All of them, however, possess certain limitations, which is perhaps not surprising since the PMCP is undecidable in general (cf. [AK 86],[Su 88]). Many of the methods are only partially automated, requiring human ingenuity to construct, e.g., a process invariant or closure process (cf. [CG 87], [BCG 89], [KM 89], [WL 89]). Some could be fully automated but do not appear to have a clearly defined class of protocols on which they are guaranteed to succeed (cf. [ShG 89], [V 93], [CGJ 95]).

Abstract graphs (for asynchronous systems) were considered in [ESr 90] for synthesis, [V 93] for automatic but incomplete verification, and in [CG 87], where they are called process closures. Interestingly, [CG 87] show (in our notation) that if, for some k, $C \parallel U^k \parallel \mathcal{A}$ is appropriately bisimilar to $C \parallel U^{k+1} \parallel \mathcal{A}$, then it suffices to model check instances of size at most k to solve the PMCP. However, they do not show that such a cutoff k always exists, and their method is not guaranteed to be complete. Pong and Dubois [PD 95] propose a similar abstract graph construction for verification of safety properties of cache coherence protocols. They consider a synchronous model with broadcast actions. Although sound for verification, their method appears to be incomplete. Lubachevsky [Lu 84] makes an interesting early report of the use of an abstract graph similar to a "region graph" for parameterized asynchronous programs using *Fetch-and-Add* primitives; however, while it caters for (partial) automation, the completeness of the method is not established and it is not clear that it can be made fully automatic.

Our approach, in contrast, is a fully automated, sound and complete one (i.e., always generates a correct "yes" or "no" answer to the PMCP). Another such approach appears in [GS 92]. They also consider systems with a single control process and an arbitrary number of user processes, but with asynchronous CCS-type interactions. Unfortunately, their algorithm has exponential space (double exponential time) worst case complexity.

Our framework thus differs from [GS 92] in these significant respects: (a) the parallel composition operator is synchronous; (b) we permit guards testing "everywhere" conditions (i.e., of the form $\forall i\, \mathcal{E}(i)$); (c) it is more tractable (PSPACE vs. EXPSPACE)[4]. Partial synchrony can also be handled

[4]On the other hand, for their model of computation with all user processes but no control process, there is a polynomial time algorithm [GS 92]. We believe that our PSPACE-completeness result is not an insurmountable barrier to practical utility, given BDD-based implementations, as suggested in section 6.

in our framework. These factors permit us to represent a wider range of concurrent systems. For example, the bus protocol described in Section 6 relies on the ability to test everywhere conditions, which are not permitted in [GS 92]. There is a noteworthy limitation in the modeling power of our present framework. Because of the covering lemma (Lemma 1), an algorithm for mutual exclusion cannot be implemented in our model (cf. [GS 92]'s control process-free model), even with the control process. We suspect it is possible to overcome this restriction, and are working on it.

Finally, it is interesting to note that we can show that for fully asynchronous computation (interleaving semantics), the PMCP for our model becomes undecidable. This is shown by a simple simulation of a two counter machine by a (C, U) family. Essentially, the zero-test of a two counter machine can be expressed as an everywhere condition, and increments can be encoded because precisely one process fires at each step in the computation.

Acknowledgements: We would like to thank Carl Pixley of Motorola for suggesting the bus protocol example, and the CAV '96 referees for bringing [PD 95] to our attention.

References

[AK 86] Apt, K., Kozen, D. Limits for automatic verification of finite-state concurrent systems. *IPL* 15, pp. 307-309.

[BCG 89] Browne, M. C., Clarke, E. M., Grumberg, O. Reasoning about Networks with Many Identical Finite State Processes, *Information and Computation*, vol. 81, no. 1, pp. 13–31, April 1989.

[CE 81] Clarke, E.M., Emerson, E.A. Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic. *Workshop on Logics of Programs*, Springer-Verlag LNCS 131.

[CES 86] Clarke, E.M., Emerson, E.A., and Sistla, A.P., Automatic Verification of Finite-State Concurrent Systems using Temporal Logic, *ACM Trans. Prog. Lang. and Sys.*, vol. 8, no. 2, pp. 244-263, April 1986.

[CFJ 93] Clarke, E.M., Filkorn, T., Jha, S. Exploiting Symmetry in Temporal Logic Model Checking, 5th CAV, Springer-Verlag LNCS 697.

[CG 87] Clarke, E.M., Grumberg, O. Avoiding the State Explosion Problem in Temporal Logic Model Checking Algorithms, PODC 1987.

[CGJ 95] Clarke, E.M., Grumberg, O., Jha, S. Verifying Parameterized Networks using Abstraction and Regular Languages. CONCUR 95.

[Em 90] Emerson, E.A., Temporal and Modal Logic, in Handbook of Theoretical Computer Science, vol. B, (J. van Leeuwen, ed.), Elsevier/North-Holland, 1991.

[EN 95] Emerson, E.A., Namjoshi, K.S. Reasoning about Rings. *Proc. ACM Symposium on Principles of Programming Languages*, 1995.

[ES 93] Emerson, E.A., Sistla, A.P. Symmetry and Model Checking, 5th CAV, Springer-Verlag LNCS 697.

[ES 95] Emerson, E.A., Sistla, A.P. Utilizing Symmetry when Model Checking under Fairness Assumptions: An Automata-theoretic approach. CAV 1995.

[ESr 90] Emerson, E.A., Srinivasan, J. A decidable temporal logic to reason about many processes. PODC 1990.

[GS 92] German, S.M., Sistla, A.P. Reasoning about Systems with Many Processes. *J.ACM*, Vol. 39, Number 3, July 1992.

[HB 95] Hojati, R., Brayton, R. Automatic Datapath Abstraction in Hardware Systems, CAV 1995.

[ID 93] Ip, C., Dill, D. Better verification through symmetry. Proc. 11th Intl. Symp. on Computer Hardware Description Languages and their Applications.

[KM 89] Kurshan, R.P., McMillan, K. A Structural Induction Theorem for Processes, PODC 1989.

[LSY 94] Li, J., Suzuki, I., Yamashita, M. Fair Petri Nets and structural induction for rings of processes. *Theoretical Computer Science*, vol. 135(2), 1994. pp. 337-404.

[LP85] Litchtenstein, O., and Pnueli, A., Checking That Finite State Concurrent Programs Satisfy Their Linear Specifications, POPL 85, pp. 97-107.

[Lo 93] Long, D. Model Checking, Abstraction, and Compositional Verification. Ph.D. Thesis, Carnegie-Mellon University, 1993.

[Lu 84] Lubachevsky, B. An Approach to Automating the Verification of Compact Parallel Coordination Programs I. *Acta Informatica* 21, 1984.

[MP 92] Manna, Z., Pnueli, A. Temporal Logic of Reactive and Concurrent Systems: Specification, Springer-Verlag, 1992.

[McM92] McMillan, K., Symbolic Model Checking: An Approach to the State Explosion Problem, Ph.D. Thesis, Carnegie-Mellon University, 1992.

[Pn 77] Pnueli, A. The Temporal Logic of Programs. FOCS 1977.

[PD 95] Pong, F., Dubois, M. A New Approach for the Verification of Cache Coherence Protocols. *IEEE Transactions on Parallel and Distributed Systems*, August 1995.

[RS 85] Reif, J., Sistla, A. P. A multiprocess network logic with temporal and spatial modalities. *JCSS* 30(1), 1985.

[RS 93] Rho, J. K., Somenzi, F. Automatic Generation of Network Invariants for the Verification of Iterative Sequential Systems. CAV 1993, LNCS 697.

[SAE 92] SAE J1850 Class B data communication network interface. Society of Automotive Engineers, Inc., 1992.

[ShG 89] Shtadler, Z., Grumberg, O. Network Grammars, Communication Behaviours and Automatic Verification. Springer-Verlag, LNCS 407.

[Su 88] Suzuki, I. Proving properties of a ring of finite state machines. *IPL* 28, pp. 213-214.

[Va9?] Vardi, M. An Automata-theoretic Approach to Linear Temporal Logic, Proceedings of Banff Higher Order Workshop on Logics for Concurrency, F. Moller, ed., Springer-Verlag LNCS, to appear.

[VW 86] Vardi, M., Wolper, P. An Automata-theoretic Approach to Automatic Program Verification, Proc. IEEE LICS, pp. 332-344, 1986.

[V 93] Vernier, I. Specification and Verification of Parameterized Parallel Programs. Proc. 8th Intl. Symp. on Computer and Information Sciences, Istanbul, Turkey, pp. 622-625.

[WL 89] Wolper, P., Lovinfosse, V. Verifying Properties of Large Sets of Processes with Network Invariants. Springer-Verlag, LNCS 407.