Reducing Model Checking of the Many to the Few

E. Allen Emerson and Vineet Kahlon

Department of Computer Sciences,
The University of Texas at Austin, Austin TX-78712, USA,
{emerson,kahlon}@cs.utexas.edu,
http://www.cs.utexas.edu/users/{emerson,kahlon}

Abstract. Systems with an arbitrary number of homogeneous processes occur in many applications. The *Parametrized Model Checking Problem* (PMCP) is to determine whether a temporal property is true for every size instance of the system. Unfortunately, it is undecidable in general. We are able to establish, nonetheless, decidability of the PMCP in quite a broad framework. We consider asynchronous systems comprised of an arbitrary number n of homogeneous copies of a generic process template. The process template is represented as a synchronization skeleton while correctness properties are expressed using Indexed CTL*\X. We reduce model checking for systems of arbitrary size n to model checking for systems of size (up to) a small *cutoff* size c. This establishes decidability of PMCP as it is only necessary model check a finite number of relatively small systems. The results generalize to systems comprised of multiple heterogeneous classes of processes, where each class is instantiated by many homogenous copies of the class template (e.g., m readers and n writers).

1 Introduction

Systems with an arbitrary number of homogeneous processes can be used to model many important applications. These include classical problems such as mutual exclusion, readers and writers, as well as protocols for cache coherence and data communication among others. It is often the case that correctness properties are expected to hold irrespective of the size of the system, as measured by the number of processes in the system. However, time and space constraints permit us to verify correctness only for instances with a small number of processes. This makes it impossible to guarantee correctness in general and thus motivates consideration of automated methods to permit verification for arbitrary size instances. The general problem, known in the literature as the *Parametrized Model Checking Problem (PMCP)* is the following: to decide whether a temporal property is true of every size instance of a given system. This problem is known to be undecidable in general [AK86,Suz88]. However, by imposing certain stipulations on the organization of the processess we can get a useful framework with a decidable PMCP.

In our framework, processes are modeled as *Synchronization Skeletons* (cf. [CE81]) which are abstractions of concurrent programs where details irrelevant to synchronization are suppressed. This is because for most actual concurrent programs the portions

of each process responsible for interprocess synchronization can be cleanly separated from the sequential application-oriented computations performed by the process. The synchronization skeleton of each process P may then be viewed as a state transition graph where each state represents a region of sequential code intended to perform some serial computation, and each arc—of the form $p \xrightarrow{g \to A} q$, where g is an enabling condition and A is the action to be performed—represents a conditional transition used to enforce synchronization constraints.

Given a family $(U_1, ..., U_\mathsf{k})$ of k process classes and a k-tuple $(n_1, ..., n_\mathsf{k})$ of natural numbers, we let $(U_1, ..., U_\mathsf{k})^{(n_1,...,n_\mathsf{k})}$ denote the concrete system comprised of n_1 copies or instances of U_1 through n_k copies or instances of U_k running in parallel asynchronously (i.e., with interleaving semantics). By abuse of notation, we also write $(U_1, ..., U_\mathsf{k})^{(n_1,...,n_\mathsf{k})}$ for the associated state graph, where each process starts in its designated initial state.

Correctness properties are expressed using the following two basic formats (i) "for all processes $\mathsf{A}h$," and (ii) "for all processes $\mathsf{E}h$," where h is an LTL$\setminus$X formula (built using F "sometimes," G "always," U, "until," but without X "next-time") over propositions indexed just by the processes being quantified over, and A "for all futures," and E "for some future" are the usual path quantifiers. Use of such an indexed, stuttering-insensitive logic is natural for parameterized systems. Moreover, allowing the next-time operator X in formulas specifying correctness properties often gives us the ability to 'count' leading to undecidability of the PMCP [EK03]. Specifically, we consider correctness properties of the following types:

1. Over all individual processes of a single class U_l:
   $\bigwedge_{i_l} \mathsf{A}h(i_l)$ and $\bigwedge_{i_l} \mathsf{E}h(i_l)$, where i_l ranges over (indices of) individual processes in U_l.
2. Over pairs of different processes of a single class U_l:
   $\bigwedge_{i_l \neq j_l} \mathsf{A}h(i_l, j_l)$ and $\bigwedge_{i_l \neq j_l} \mathsf{E}h(i_l, j_l)$, where i_l, j_l range over pairs of distinct processes in U_l.
3. Over one process from each of two different classes U_l, U_m:
   $\bigwedge_{i_l, j_m} \mathsf{A}h(i_l, j_m)$ and $\bigwedge_{i_l, j_m} \mathsf{E}h(i_l, j_m)$, where i_l ranges over U_l and j_m ranges over U_m.

We say that the k-tuple $(c_1, ..., c_\mathsf{k})$ of natural numbers is a *cutoff* for the family $(U_1, ..., U_\mathsf{k})$ of process classes for formula f iff : $\forall\ (n_1, ..., n_\mathsf{k}) : (U_1, ..., U_\mathsf{k})^{(n_1,...,n_\mathsf{k})} \models f$ iff $\forall (m_1, ..., m_\mathsf{k}) \preceq (c_1, ..., c_\mathsf{k})\ :\ (U_1, ..., U_\mathsf{k})^{(m_1,...,m_\mathsf{k})} \models f$, where we write $(m_1, ..., m_\mathsf{k}) \preceq (c_1, ..., c_\mathsf{k})$ to mean $(m_1, ..., m_\mathsf{k})$ is *component-wise* less than or equal to $(c_1, ..., c_\mathsf{k})$ and $(m_1, ..., m_\mathsf{k}) \succeq (c_1, ..., c_\mathsf{k})$ to mean $(c_1, ..., c_\mathsf{k}) \preceq (m_1, ..., m_\mathsf{k})$.

In this chapter, we show that for systems in the synchronization skeleton framework with transition guards of a particular *Disjunctive* or *Conjunctive* form, there exists a small cutoff. This, in effect, reduces the PMCP to standard model checking over a relatively few small, finite-sized systems. In some cases, depending on the kind of property and guards, we can get an efficient solution to the PMCP.

Each process class is described by a generic process—a process *template* for the class. A system with k classes is given by the family of templates $(U_1, ..., U_\mathsf{k})$. For such a system, define $c_i = 2|U_i| + 1$ and $d_i = |U_i| + 3$, where $|U_i|$ is the size of

U_i as given by the number of local states of U_i. Then, for conjunctive and disjunctive guards, cutoffs of $(c_1, ..., c_k)$ and $(d_1, ..., d_k)$, respectively, suffice for all three types of formulas described above. These results give decision procedures for the PMCP for conjunctive and for disjunctive guards. Since these are broad frameworks and the PMCP is undecidable in general, we view this as quite a positive result.

However, the decision procedures are not necessarily efficient ones, although they may certainly be usable on small examples. Because the cutoff is proportional to the sizes of the template processes, the global state graph of the cutoff system is of size exponential in the template sizes, resulting in exponential time decision procedures. In the case of disjunctive guards, it turns out that if we restrict ourselves to formulas with the A path quantifier, but still permit all three type of properties, then the cutoff can be reduced, in quadratic time in the size of the template processes, to $(1, ..., 2, ..., 1)$ or $(1, ..., 3, ..., 1)$. In fact, depending on the type of property, we can show that it is possible to simplify the guards to ensure that only two or three classes need be retained. On the other hand, for conjunctive guards, if we restrict ourselves to model checking purely over infinite paths or purely over finite paths, then sharper cutoffs of the form (1,...,3,...,1), (1,...,2,...,1) or even (1,...,1) can, in some cases, be obtained.

The rest of the chapter is organized as follows. Section 2 defines the system model. We show how to exploit symmetry inherent in the model and correctness properties in section 3. Cutoff results pertaining to disjunctive and conjunctive guards are given in sections 4 and 5, respectively. Applications are considered in section 6 and we conclude with some remarks in section 7.

2 The System Model

We focus on systems comprised of multiple heterogeneous *classes* of processes modeled as *synchronization skeletons* (cf. [CE81]). Here, an individual concrete process has a transition of the form $p \xrightarrow{g} q$ indicating that the process can transit from local state p to local state q, provided the guard g is true. Each class is specified by giving a generic process *template*. If I is (an) index set $\{1, \ldots, n\}$, then we use U^I, or $(U)^{(n)}$, for short, to denote the concurrent system $U^1 \| \ldots \| U^n$ comprised of the n isomorphic (up to re-indexing) processes U^i running in parallel asynchronously. For a system with k classes associated with the given templates $U_1, ..., U_k$, we have corresponding (disjoint) index sets $I_1, \ldots I_k$. Each index set I_j is (a copy of) an interval $\{1, \ldots, m\}$ of natural numbers, denoted $\{1_j, \ldots, m_j\}$ for emphasis[1]. In practice, we assume the k index sets are specified by giving a k-tuple $(n_1, ..., n_k)$ of natural numbers, corresponding to I_1 being (a copy of) interval $\{1, \ldots, n_1\}$ through I_k being (a copy of) interval $\{1, \ldots, n_k\}$.

Given a family $(U_1, ..., U_k)$ of k template processes and a k-tuple $(n_1, ..., n_k)$ of natural numbers, we let $(U_1, ..., U_k)^{(n_1, \ldots, n_k)}$ denote the concrete system comprised of n_1 copies of U_1 through n_k copies of U_k running in parallel asynchronously (i.e., with interleaving semantics). A template process $U_l = (S_l, R_l, \imath_l)$ for class l, is comprised of a finite non-empty set S_l of (local) states, a set of transition edges R_l, and an initial

[1] e.g., if I_1 is a copy of $\{1, 2, 3\}$, the said copy is denoted $\{1_1, 2_1, 3_1\}$. Informally, subscripted index 3_1 means process 3 of class 1; formally, it is the ordered pair $(3, 1)$ as is usual with indexed logics.

(local) state i_l. Each transition in R_l is labeled with a guard which is a boolean expression over atomic propositions corresponding to local states of other template processes. Then given template process U_l and index $i \in [1 : n_l]$, we use $U_l^i = (S_l^i, R_l^i, \mathsf{i}_l^i)$ to denote the ith copy of the template process U_l. Here S_l^i, the state set of U_l^i, R_l^i its transition relation and i_l^i its initial state are obtained from S_l, R_l and i_l, respectively, by uniformly superscripting the states of U_l with i. Thus, for local states p_l and q_l of template process U_l, p_l^i and q_l^i denote concrete local states of U_l^i, and $p_l^i \rightarrow q_l^i \in R_l^i$ iff $p_l \rightarrow q_l \in R_l$.

From the guards labeling the transitions of a given template process U_l, we now describe how to get the guards for the concrete process U_l^i of $(U_1, ..., U_\mathsf{k})^{(n_1,...,n_\mathsf{k})}$. We consider the following two types of guards.

- *Disjunctive guards*—of the general form $(a_1 \vee ... \vee b_1) \bigvee ... \bigvee (a_k \vee ... \vee b_\mathsf{k})$—label transition $(p_l, q_l) \in R_l$, where $a_m, ..., b_m$ are (propositions identified with) the local states of template U_m. In concrete process U_l^i of the system $(U_1, ..., U_\mathsf{k})^{(n_1,...,n_\mathsf{k})}$, the corresponding transition $(p_l^i, q_l^i) \in R_l^i$ is then labeled by the guard $\bigvee_{r \neq i}(a_l^r \vee ... \vee b_l^r) \vee \bigvee_{j \neq l}(\bigvee_{k \in [1:n_j]} (a_j^k \vee ... \vee b_j^k))$, where, for $c_j \in S_j$, proposition c_j^k is understood to be true when process k in class U^j, i.e., U_j^k, is in local state c_j for template process U_j.
- *Conjunctive guards with initial state*—of the general form $(\mathsf{i}_1 \vee a_1 \vee ... \vee b_1) \bigwedge ... \bigwedge (\mathsf{i}_k \vee a_k \vee ... \vee b_k)$. In concrete process i of class l, U_l^i, the corresponding transition is then labeled by the guard $\bigwedge_{r \neq i}(\mathsf{i}_l^r \vee a_l^r \vee ... \vee b_l^r) \wedge \bigwedge_{j \neq l}(\bigwedge_{k \in [1:n_j]} (\mathsf{i}_j^k \vee a_j^k \vee ... \vee b_j^k))$.

Note that the initial local states of processes must be present in the expressions for the conjunctive guards. Thus, the initial state of each process has a neutral or non-blocking character so that when a process is in its initial state, it does not prevent progress by another process distinct from it. This natural condition permits modeling a broad range of applications (and is helpful technically).

We now formalize the asynchronous concurrent (interleaving) semantics. A process transition labeled with guard g is *enabled* in global state s iff $s \models g$, i.e., g is true when evaluated over the local states in s. The global state transition diagram, $(U_1, ..., U_\mathsf{k})^{(n_1,...,n_\mathsf{k})}$, of the system instance corresponding to the tuple $(n_1, ..., n_\mathsf{k})$ is given by the tuple $(U_1, ..., U_\mathsf{k})^{(n_1,...,n_\mathsf{k})} = (S^{(n_1,...,n_\mathsf{k})}, R^{(n_1,...,n_\mathsf{k})}, \mathsf{i}^{(n_1,...,n_\mathsf{k})})$. A state $s \in S^{(n_1,...,n_\mathsf{k})}$ is written as a $(n_1 + ... + n_\mathsf{k})$-tuple $(u_1^1, ..., u_1^{n_1}, u_2^1, ..., u_\mathsf{k}^{n_\mathsf{k}})$, where the projection of s onto process i of class l, denoted $s[l, i]$, equals u_l^i, the local state in s of the ith copy of the template process U_l. The initial state $\mathsf{i}^{(n_1,...,n_\mathsf{k})} = (\mathsf{i}_1^1, ..., \mathsf{i}_\mathsf{k}^{n_\mathsf{k}})$. A global transition $s \longrightarrow t \in R^{(n_1,...,n_\mathsf{k})}$ iff t results from s by firing an enabled transition of some process, i.e., there exist i, l such that the guard labeling $p_l^i \rightarrow q_l^i \in R_l^i$ is enabled at s, $s[l, i] = p_l^i$, $t[l, i] = q_l^i$, and for all $(j, k) \neq (i, l)$, $s[k, j] = t[k, j]$. We write $(U_1, ..., U_\mathsf{k})^{(n_1,...,n_\mathsf{k})} \models f$ to indicate that the global state graph of $(U_1, ..., U_\mathsf{k})^{(n_1,...,n_\mathsf{k})}$ satisfies f at initial state $\mathsf{i}^{(n_1,...,n_\mathsf{k})}$.

For global state s, let *Set(s)* denote the set $\{a \mid s \textit{ contains an indexed local copy of } a\}$. For computation path $x = x_0, x_1, ...$ we define $PathSet(x) = \bigcup_i Set(x_i)$. The definition of projection is extended to include computation sequences as follows: for computation path $x = x_0, x_1, ...$ and $i \in [1 : n_l]$, the sequence of local states

$x_0[l,i], x_1[l,i], ...$ is denoted by $x[l,i]$. We say that the sequence of global states $y = y_0, y_1, ...$ is a *stuttering* of computation path x iff there exists a parsing $P_0 P_1 ...$ of y such that for all $j \geq 0$ there is some $r > 0$ with $P_j = (x_j)^r$ (cf. [BCG89]).

3 Appeal to Symmetry

We can exploit symmetry inherent in the system model and the correctness properties in the spirit of "state symmetry" as codified by [ES93] (cf. [ID96], [McM99]) to simplify our proof obligation. For formulas of types $\bigwedge_{i_l} f(i_l)$, $\bigwedge_{i_l \neq j_l} f(i_l, j_l)$ and $\bigwedge_{i_l, j_m} f(i_l, j_m)$, it suffices to show the results with the formulas replaced by $f(1_l)$, $f(1_l, 2_l)$ and $f(1_l, 1_m)$, respectively. The basic idea is that in a system comprised of fully interchangeable processes 1 through n of a given class, symmetry considerations dictate that process 1 satisfies a property iff each process $i \in [1:n]$ satisfies the property.

4 Systems with Disjunctive Guards

In this section, we consider the PMCP for systems with Disjunctive Guards. As defined before, these guards can be used to test whether there exists another process in one of a specified set of local states. Disjunctive Guards are used, for example, in cache coherence protocols when a processor cache wants to check whether another cache has the memory block it needs and based on that decide from where to fetch the required block. For such systems, we show how to reduce the model checking problem for systems with an arbitrary number of copies of each process class to systems with up to a cutoff number of copies in each class. The size of the cutoff for each class is essentially the number of local states of individual process template for the class. This yields decidability for this formulation of the PMCP, a pleasant result since the PMCP is undecidable in full generality. We go on to show that in the case of universal path quantified specification formulas $\mathsf{A}h$, small constant-size cutoffs can be obtained yielding provably efficient—polynomial time—decision procedures.

4.1 Properties ranging over all processes in a single class

For the sake of notational simplicity, we consider systems with just the two process classes V_1 and V_2. We begin by proving the Disjunctive Monotonicity and Disjunctive Bounding lemmas which allow us to, respectively, increase and decrease the system size one coordinate at a time while preserving properties of the from $\mathsf{E}h$ for process index 1 from each class.

Lemma 1 (Disjunctive Monotonicity Lemma).

(i) $\forall n \geq 1 : (V_1, V_2)^{(1,n)} \models \mathsf{E}h(1_2)$ *implies* $(V_1, V_2)^{(1,n+1)} \models \mathsf{E}h(1_2)$.
(ii) $\forall n \geq 1 : (V_1, V_2)^{(1,n)} \models \mathsf{E}h(1_1)$ *implies* $(V_1, V_2)^{(1,n+1)} \models \mathsf{E}h(1_1)$.

Proof

(i) For any computation x of $(V_1, V_2)^{(1,n)}$, there exists an analogous computation y of $(V_1, V_2)^{(1,n+1)}$ wherein the $(n+1)$st copy of template process V_2 stutters in its initial state and the rest of the processes behave as along x. Note that from the semantics of disjunctive guards, it follows that having the 'extra' process V_2^{n+1} in its initial state does not disable any of the transitions of processes distinct from it that were fired along x. Thus y is a valid computation sequence.

(ii) This part follows by using a similar argument. □

Lemma 2 (Disjunctive Bounding Lemma).

(i) $\forall n \geq |V_2| + 2 : (V_1, V_2)^{(1,n)} \models \mathsf{E}h(1_2)$ *iff* $(V_1, V_2)^{(1,c_2)} \models \mathsf{E}h(1_2)$, *where* $c_2 = |V_2| + 2$.

(ii) $\forall n \geq |V_2| + 1 : (V_1, V_2)^{(1,n)} \models \mathsf{E}h(1_1)$ *iff* $(V_1, V_2)^{(1,|V_2|+1)} \models \mathsf{E}h(1_1)$.

Proof

(i) Given a computation x of $(V_1, V_2)^{(1,n)}$, we construct a computation y of the system $(V_1, V_2)^{(1,c_2)}$ such that $x[2,1]$, i.e., the local computation in x of process 1 of template class V_2, is a stuttering of $y[2,1]$; and vice versa.

($\Rightarrow$) Let $x = x_0, x_1, ...$ be a computation sequence of $(V_1, V_2)^{(1,n)}$. We construct a formal sequence $y = y_0, y_1, ...$ of global states of $(V_1, V_2)^{(1,c_2)}$ from x as described below.

1. Set $y[1,1] = x[1,1]$ and $y[2,1] = x[2,1]$, i.e., the local computation paths in x of process index 1 of each of the classes V_1 and V_2 is preserved. This ensures that $x[2,1]$ and $y[2,1]$ are stuttering equivalent, as desired.

2. If x is an infinite computation, then we need to make sure that y is also an infinite computation. Towards that end, note that if any one of $x[1,1]$ or $x[2,1]$ is an infinite local computation, then we are already done. Otherwise, let v be an infinite local computation of x, in case x is an infinite computation; else, if x is finite, let $v = (\mathrm{i}_2)^{|x|}$, where $|x|$ denotes the length of x. Set $y[2, c_2] = v$.

3. In order for y to be a valid computation, we still need to ensure that the guard labeling each transition of x mimicked along y is enabled in y. This is accomplished by 'flooding' each local state of template V_2 occurring along x, as described below. Let $Reach = \{a_1, ..., a_l\}$ be the set of all local states of template process V_2 occurring along x. Consider $a_j \in Reach$. Let $b_1, ..., b_m$, where $b_m = a_j$, be the finite local computation of minimum length in x leading to local state a_j. We denote m by $MinLen(a_j)$. Furthermore, if x is an infinite computation we use $MinComp(a_j)$ to denote the sequence $b_1, ..., b_{m-1}, (b_m)^\omega$, else if x is an finite computation, then we use it denote a sequence of the form $b_1, ..., b_{m-1}, (b_m)^*$, where local state b_m stutters sufficiently many times to ensure that $MinComp(a_j)$ is of length $|x|$. Then, for each state $a_j \in Reach$, we set $y[2, j+1] = MinComp(a_j)$, i.e., we let the $(j+1)$st copy of V_2 execute the local computation of minimum length in x leading to a_j and subsequently let it stutter in a_j. Thus each state in *Reach* is now flooded. This has the implication that each local transition of x mimicked along y is enabled. Indeed let global transition $y_i \longrightarrow y_{i+1}$ result by firing local transition $tr : c \rightarrow d$ of template V_2. Let tr be labeled with guard g. By our construction, global state x_{i+1} also results from x_i by the firing of

local transition tr. Since g is disjunctive in nature, in global state x_i, there exits a process V in local state e, say, that enables guard g. If $V = V_1$, then $y_i[1,1] = x_i[1,1] = e_1$. Else if V is a copy of V_2, then $e = a_k$, for some $a_k \in Reach$. In that case, we have $MinLength(a_k) \leq i$ and so, by our construction, $y_i[2, k+1] = e_{k+1}$. Thus in either case there is a process other than the one firing tr that is in local state e thereby ensuring that guard g is enabled in y_i.

However, it might be the case that sequence y violates the interleaving semantics requirement. Indeed, consider the following scenario. Let states $a_i, a_j \in Reach$, be such that $MinComp(a_i)$ and $MinComp(a_j)$ are realized by the same local computation of x and suppose that $MinLen(a_i) \leq MinLen(a_j)$. If for $k < MinLen(a_i)$, $b_k \rightarrow b_{k+1}$ is a transition of $MinComp(a_i)$, then $y_k[2, i+1] \rightarrow y_{k+1}[2, i+1]$ and $y_k[2, j+1] \rightarrow y_{k+1}[2, j+1]$ are both local transitions driving y_k to y_{k+1}. This violates the interleaving semantics condition requiring that there be at most one local transition driving each global transition. There are two things to note here. First, for a transition $y_{k'} \rightarrow y_{k'+1}$, the violation occurs only for values of k' less than or equal to the maximum of $MinLen(a_{j'})$ over all $j' \in [0 : l]$. Secondly, for a fixed i', all violations are caused by a unique template transition $c \rightarrow d$ of V_2, namely one responsible for firing the global transition $x_{i'} \longrightarrow x_{i'+1}$.

To solve this problem, we 'stagger" copies of the local transition that are fired simultaneously, as described below. Let $y_i \longrightarrow y_{i+1}$ be a transition where the interleaving semantics requirement is violated by process indices $in_1, ..., in_f$ of V_2 executing indexed copies $c_2^{in_1} \rightarrow d_2^{in_1}, ..., c_2^{in_f} \rightarrow d_2^{in_f}$ respectively of the template transition $c_2 \rightarrow d_2$ of V_2. Replace the single global transition $y_i \longrightarrow y_{i+1}$ with a sequence $u_1, ..., u_{f+1}$ such that $u_1 = y_i$, $u_{f+1} = y_{i+1}$ and for each $j \in [1 : f]$, transition $u_j \rightarrow u_{j+1}$ results by executing local transition $c_2^{in_j} \rightarrow d_2^{in_j}$. Clearly the interleaving semantics requirement is met as at most one local transition is executed for each global transition. Finally, note that states with indices other than $in_1, ..., in_f$ are made to stutter finitely often in $u_1, ..., u_{f+1}$ which is allowed since we are considering formulas without the next-time operator X.

Thus, given a computation path x of $(V_1, V_2)^{(1,n)}$, we have constructed a stuttering computation path y of $(V_1, V_2)^{(1,c_2)}$, such that the local computation sequence $y[2,1]$ is a stuttering of the local computation sequence $y[2,1]$. This stuttering path correspondence, gives us the result.

($\Leftarrow$) The proof follows by repeated application of the Disjunctive Monotonicity Lemma.

(ii) This part follows by using a similar argument. □

Using the previous result, we get the Disjunctive Truncation Lemma that allows reduction in system size over multiple coordinates simultaneously (2 coordinates for notational brevity) instead of just one while preserving properties of the form $\mathsf{E}h$ over process index 1 in each class. The cutoff result then follows as an immediate corollary.

Lemma 3 (Disjunctive Truncation Lemma).
$\forall n_1, n_2 \geq 1 : (U_1, U_2)^{(n_1,n_2)} \models \mathsf{E}h(1_2)$ *iff* $(U_1, U_2)^{(m_1,m_2)} \models \mathsf{E}h(1_2)$, *where* m_1 *is the minimum of* n_1 *and* $|U_1| + 1$, *and* m_2 *is the minimum of* n_2 *and* $|U_2| + 2$.

Proof
If $n_2 > |U_2| + 2$, set $V_1 = (U_1)^{(n_1)}$ and $V_2 = U_2$. Then, $(U_1, U_2)^{(n_1,n_2)} \models \mathsf{E}h(1_2)$ iff $(V_1, V_2)^{(1,n_2)} \models \mathsf{E}h(1_2)$ iff $(V_1, V_2)^{(1,m_2)} \models \mathsf{E}h(1_2)$ (by the Disjunctive Bounding Lemma) iff $(U_1, U_2)^{(n_1,m_2)} \models \mathsf{E}h(1_2)$.

If $n_1 \leq |U_1| + 1$, then $n_1 = m_1$ and we are done; else set $V_1 = (U_2)^{(m_2)}$ and $V_2 = U_1$. Then, $(U_1, U_2)^{(n_1,m_2)} \models \mathsf{E}h(1_2)$ iff $(U_2, U_1)^{(m_2,n_1)} \models \mathsf{E}h(1_1)$ iff $(V_1, V_2)^{(1,n_1)} \models \mathsf{E}h(1_1)$ iff $(V_1, V_2)^{(1,m_1)} \models \mathsf{E}h(1_1)$ (by the Disjunctive Bounding Lemma) iff $(U_1, U_2)^{(m_1,m_2)} \models \mathsf{E}h(1_2)$. □

An easy but important consequence of the Disjunctive Truncation Lemma is the following.

Theorem 1 (Disjunctive Cutoff Result). *Let* f *be* $\bigwedge_{i_l} \mathsf{A}h(i_l)$ *or* $\bigwedge_{i_l} \mathsf{E}h(i_l)$, *where* h *is a LTL\X formula and* $l \in [1 : 2]$. *Then*
$\forall (n_1, n_2) \succeq (1, 1) : (U_1, U_2)^{(n_1,n_2)} \models f$ *iff*
$\forall (d_1, d_2) \preceq (c_1, c_2) : (U_1, U_2)^{(d_1,d_2)} \models f$
where the cutoff (c_1, c_2) *is given by* $c_l = |U_l| + 2$, *and for* $i \neq l : c_i = |U_i| + 1$.

Proof By appeal to symmetry and the fact that A and E are duals, it suffices to prove the result for formulas of the type $\mathsf{E}h(1_2)$. The *if* direction is trivial. For the *only if* direction, let $n_1, n_2 \geq 1$. Define n_1' to be the minimum of n_1 and $|U_1|+1$, and n_2', the minimum of n_2 and $|U_2|+2$. Then, by the Disjunctive Truncation Lemma, $(U_1, U_2)^{(n_1,n_2)} \models f(1_2)$ iff $(U_1, U_2)^{(n_1',n_2')} \models f(1_2)$. This proves the cutoff result. □

More generally for systems with $\mathsf{k} \geq 1$, different classes of processes, the cutoff results for systems with disjunctive guard can be formualted as below. The proof is along similar lines as for systems with two process classes.

Theorem 2 (Disjunctive Cutoff Theorem). *Let* f *be* $\bigwedge_{i_l} \mathsf{A}h(i_l)$ *or* $\bigwedge_{i_l} \mathsf{E}h(i_l)$, *where* $l \in [1 : \mathsf{k}]$ *and* h *is an LTL\X formula. Then*
$\forall (n_1, \ldots, n_\mathsf{k}) \succeq (1, \ldots, 1) : (U_1, \ldots, U_\mathsf{k})^{(n_1,\ldots,n_\mathsf{k})} \models f$ *iff*
$\forall (d_1, \ldots, d_\mathsf{k}) \preceq (c_1, \ldots, c_\mathsf{k}) : (U_1, \ldots, U_\mathsf{k})^{(d_1,\ldots,d_\mathsf{k})} \models f$,
where the cutoff $(c_1, \ldots, c_\mathsf{k})$ *is given by* $c_l = |U_l| + 2$ *and for* $i \neq l : c_i = |U_i| + 1$.

A useful corollary to the above result is the decidability results for the PMCP for systems with disjunctive guards.

Corollary 1 (Disjunctive Decidability Theorem). *The PMCP for systems with disjunctive guards and single-index assertions of the forms* $\bigwedge_{i_l} \mathsf{A}h(i_l)$ *and* $\bigwedge_{i_l} \mathsf{E}h(i_l)$ *is decidable in exponential time in the size of the tampltes defining the parameterized family.*

Proof By the Disjunctive Cutoff Theorem, it is enough to model check exponentially many state graphs each of exponential size for the systems $(U_1, \ldots, U_k)^{(d_1,\ldots,d_k)}$ for all $(d_1, \ldots, d_k) \preceq (c_1, \ldots, c_k)$. □

4.2 Efficient decidability for "for all future" properties

While the Disjunctive Cutoff Result yields decidability for the PCMP for disjunctive guards, the resulting decision procedure has a worst case complexity that is exponential in the size of each of the templates. It turns out that for universal-path-quantified formulas it is possible to be much more efficient. For such properties, we show that we can give a decision procedure for the PMCP that has a polynomial time worst case complexity in the size each of the templates. Towards that end, we first show that the PMCP for properties of the form $\mathsf{A}h$ reduces to model checking just the *single* system instance of size equal to the (small) cutoff (as opposed to all systems of size less than or equal to the cutoff).

Lemma 4 (Single-Cutoff Lemma). $\forall n_1, n_2 \geq 1 : (U_1, U_2)^{(n_1,n_2)} \models \mathsf{A}h(1_2)$ *iff* $(U_1, U_2)^{(c_1,c_2)} \models \mathsf{A}h(1_2)$, *where* $c_1 = |U_1| + 1$ *and* $c_2 = |U_2| + 2$.

Proof

($\Rightarrow$) This direction follows easily by instantiating $n_1 = c_1$ and $n_2 = c_2$ on the left hand side.

($\Leftarrow$) Choose arbitrary $k_1, k_2 \geq 1$. Set k_1' to be the minimum of k_1 and c_1, and k_2' to be the minimum of k_2 and c_2. By the Disjunctive Truncation Lemma, we have $(U_1, U_2)^{(k_1,k_2)} \models \mathsf{E}h(1_2)$ iff $(U_1, U_2)^{(k_1',k_2')} \models \mathsf{E}h(1_2)$. Then, by repeated application of the Disjunctive Monotonicity Lemma, we get $(U_1, U_2)^{(c_1,c_2)} \models \mathsf{E}h(1_2)$. Finally, by contraposition, $(U_1, U_2)^{(c_1,c_2)} \models \mathsf{A}h(1_2)$ implies $(U_1, U_2)^{(k_1,k_2)} \models \mathsf{A}h(1_2)$. Since k_1 and k_2 were arbitrarily chosen, we are done. □

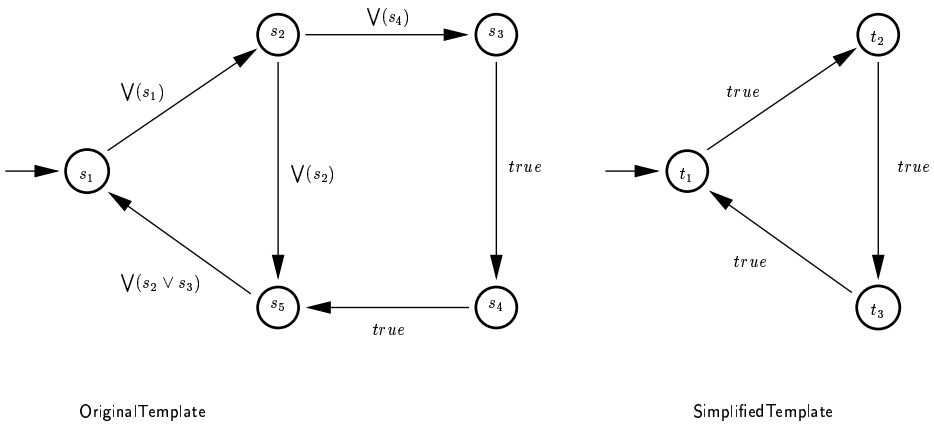

Fig 1.1 An Example of Simplification

Simplification Procedure. We now describe a transformation that given the templates $U_1, ..., U_k$ with disjunctive guards allows us to construct the simplified templates $U_1', ..., U_k'$, such that

1. For each i, the state transition graph for U_i' is a subgraph of the state transition diagram of U_i, and
2. All guards labeling transitions of U_i' are *true*.

We follow that up by showing that the transformation preserves single and double index properties of the type Ah.

Given templates $U_1, ..., U_k$, define *Reachable-States*$(U_1, ..., U_k) = (Q_1, ..., Q_k)$, where $Q_i = \{ q \mid q \in S_i$, *such that for some* $n_1, ..., n_k \geq 1$, *there exists a computation path of* $(U_1, ..., U_k)^{(n_1, ..., n_k)}$ *leading to a global state that contains a local indexed copy of* $q\}$. For all $i \geq 0$, and for all $j \in [1 : k]$, we define P_j^i as follows:

$P_j^0 = \{i_j\}$.
$P_j^{i+1} = P_j^i \bigcup \{ q \mid \exists p \in P_j^i : \exists p \xrightarrow{g} q \in R_j$ *and the expression for* g *contains a state in* $\bigcup_k P_k^i\}$.

For $j \in [1 : k]$, define $P_j = \bigcup_i P_j^i$.

Lemma 5 (Soundness Lemma). *For a fixed* i *and for all* $j \in [1 : k]$, *define* $m_j = |P_j^i|$. *Then there exists a finite computation sequence of* $(U_1, ..., U_k)^{(m_1, ..., m_k)}$ *leading to a global state that has for each* $j \in [1 : k]$ *and for each* $a_j \in P_j^i$, *a local indexed copy of* a_j.

Proof The proof is by induction on i. The base case, $i = 0$, is vacuously true. Assume that the result holds for $i \geq 0$. For each $j \in [1 : k]$, let $n_j = |P_j^i|$ and let $x = x_0, ..., x_l$ be a finite computation of $(U_1, ..., U_k)^{(n_1, ..., n_k)}$ such that global state x_l has for each $j \in [1 : k]$ and for each $a_j \in P_j^i$, a local indexed copy of a_j.

If for each j, $P_j^{i+1} = P_j^i$, then x is the desired computation and we are done. Now assume that $P_k^{i+1} \neq P_k^i$. Let $c_k \in P_k^{i+1} \setminus P_k^i$. Furthermore, let $b_k \to c_k$ be the transition that led to the inclusion of c_k into P_k^{i+1}. Clearly, $b_k \in P_k^i$. Then, by the induction hypothesis, in global state x_l there exists a copy of template process U_k, say $U_k^{n_k}$, in local state b_k. We now construct a finite sequence $y = y_0, ..., y_{2l}$ of states of $(U_1, ..., U_k)^{(n_1, ..., n_k+1, ..., n_k)}$ as follows.

1. For $j \in [1 : k] : k' \in [1 : n_j] : y[j, k'] = x[j, k'](x_l[j, k'])^l$, i.e., we let all processes other than the 'extra' process $U_k^{n_k+1}$ execute the same local computations as along x and then stutter.

2. $y[k, n_k + 1] = (i_k^{n_k+1})^l u (c_k^{n_k+1})^*$, where u is the local computation $x[k, n_k]$ leading to b_k with the index n_k replaced by $n_k + 1$, i.e., we let $U_k^{n_k+1}$ stutter in its initial state for the first l steps; then execute the same local computation as process $U_k^{n_k}$ along x leading to a copy of b_k; then fire the local transition $b_k \to c_k$ resulting in a new copy of local state c_k; and finally let it stutter in local state c_k to ensure that the length of the resulting local computation is $2l$.

We claim that y is a valid stuttering computation path of $(U_1, ..., U_k)^{(n_1,...,n_k+1,...,n_k)}$. Note that in the subsequence $y_1, ..., y_l$ all processes other than $U_k^{n_k+1}$ exhibit the same behavior as along x with $U_k^{n_k+1}$ stuttering in its initial non-blocking local state. Thus all transitions fired along $y_0, ..., y_l$ are enabled. Subsequently, the only transitions fired along y are by process $U_k^{n_k+1}$—first the local sequence u and then the transition $b_k \rightarrow c_k$. Note that, by our construction, each local state occurring along x, i.e., each state in $\bigcup_j P_j^i$, has a copy in y_l and therefore in each of the global states $y_l, ..., y_{2l-1}$ and y_{2l}. This ensures that any transition of $y[k, n_k + 1]$ that was fired along x is enabled. Specifically, all transitions fired along u are enabled. Finally since y_m, where $m \geq l$, has a copy of each state in $\bigcup_j P_j^i$, therefore transition $b_k \rightarrow c_k$ of $U_k^{n_k+1}$ is also enabled thus proving our claim.

Note that global state y_{2l} has at least one copy of each state in $\bigcup_j P_j^i$ plus a copy of c_k. Repeating the above procedure for each state in $P_j^{i+1} \setminus P_j^i$, we get a computation path with the desired property. This completes the induction step and proves the lemma. □

Lemma 6 (Completeness Lemma). $(Q_1, ..., Q_k) = (P_1, ..., P_k)$.

Proof By the above lemma, $\forall i \in [1 : k] : P_i \subseteq Q_i$. If possible, suppose that $(Q_1, ..., Q_k) \neq (P_1, ..., P_k)$. Then the set $D = \bigcup_i (Q_i - P_i) \neq \emptyset$. For definiteness, let $c_j \in D \bigcap S_j$. Then by definition of Q_j, there exists a finite computation sequence $x = x_0, ..., x_l$ such that for some k, $x_l[j, k] = c_j^k$. Let $min \in [0 : l]$ be the smallest index such that $Set(x_{min}) \bigcap D \neq \emptyset$. Then there exists a local transition $a_j \xrightarrow{g} b_j \in R_j$ driving global state x_{min-1} into x_{min} such that $x_{min-1} \models g$ and $b_j \in D$. Clearly, $Set(x_{min-1}) \subseteq PathSet(x_0, ..., x_{min-1}) \subseteq \bigcup_i P_i$. This implies that for some i', $Set(x_{m-1}) \subseteq P_j^{i'}$. Since g is enabled, there exists a state occurring in the expression for g that is included in the set $Set(x_{min-1})$ and therefore in the set $P_j^{i'}$. But then b_j would be included in $P_j^{i'+1} \subseteq P_j$, a contradiction to our assumption that $b_j \in D$. Thus $D = \emptyset$. This completes the proof. □

We now modify the k-tuple of template processes $(U_1, ..., U_k)$ to get the k-tuple $(U_1', ..., U_k')$, where $U_i' = (Q_i, R_i', i_i)$, with $p_i \rightarrow q_i \in R_i'$ iff the expression for the guard g_i labeling $p_i \rightarrow q_i$ in U_i contains a state in $\bigcup_{j \in [1:k]} Q_j$. Furthermore, any transition in the new system is labeled with $true$. See the above figure for an example. The motivation behind these definitions is that since for any $n_1, n_2, ..., n_k \geq 1$, no indexed copy of any state in $S_i \setminus Q_i$ is reachable in any computation of $(U_1, ..., U_k)^{(n_1,...,n_k)}$, we can safely delete these states from their respective template process. Also, any guard of a template process involving only states in $S_i \setminus Q_i$, will then always evaluate to false and hence any transition labeled with such a guard will never be fired. This justifies deleting such transitions from the transition graphs of the respective template processes. We now show that we can reduce the PMCP for properties of the form Ah to model checking a system comprised of just two of these simplified templates.

Theorem 3 (Reduction Result). *Let $V = U_k'$ if for some $k \in [1 : k]$, the transition graph for U_k' has a nontrivial strongly connected component, else let $V = U_1'$. Then,*

$(U_1, ..., U_k)^{(c_1,...,c_k)} \models \mathsf{A}h(1_p)$ *iff* $(U_p^{'}, V)^{(1,1)} \models \mathsf{A}h(1_1)$, *where* $c_p = |U_p| + 2$ *and* $c_i = |U_i| + 1$ *for* $i \neq p$.

Proof We show that $(U_1, ..., U_k)^{(c_1,...,c_k)} \models \mathsf{E}h(1_p)$ iff $(U_p^{'}, V)^{(1,1)} \models \mathsf{E}h(1_1)$. For definiteness, assume that $V = U_r^{'}$.

($\Rightarrow$) Given a computation x of $(U_1, ..., U_k)^{(c_1,...,c_k)}$, we construct a computation y of $(U_p', V)^{(1,1)} = (U_p', U_r')^{(1,1)}$ such that (i) $y[1,1]$ is stuttering equivalent to $x[p,1]$, and (ii) if x is an infinite computation then so is y.

Define a formal sequence $y = y_1, y_2, ...$ of states of $(U_p', U_r')^{(1,1)}$ as described below. To ensure that y satisfies (i), we set $y[1,1] = x[p,1]$. In case $x[p,1]$ is an infinite computation sequence then we simply set $y[2,1] = (\mathsf{i}_r)^\omega$ and we are done. Now assume that x is an infinite computation but the local computation $x[p,1]$ is finite and of length f, say. Since x is an infinite computation, there exists a process that executes an infinite local computation in x. This can happen only if there exists a template whose transition diagram has an infinite path and hence a non-trivial strongly connected component. Thus, by definition of V, the transition diagram for template U_r' has an infinite local path, say u. Then set $y[2,1] = (\mathsf{i}_r)^f u$.

To prove that y is a valid computation sequence of $(U_p^{'}, V)^{(1,1)}$, it suffices to show that all transitions fired along y are valid. This follows by noting that all local states occurring along x are reachable and thus belong to $\bigcup_j Q_j$. In particular, all local states occurring along $x[p,1]$ and u belong to $\bigcup_j Q_j$. But by the Completeness Lemma, we have $\bigcup Q_j = \bigcup P_j$. Thus all local states occurring along $x[p,1]$ and u belong to $\bigcup_j P_j$. Furthermore, all local transitions fired along $x(p,1)$ and u are labeled by guards whose expressions involve a state in $\bigcup_j Q_j = \bigcup_j P_j$ and hence they occur in $\bigcup_j R_j^{'}$. Finally these transition are now labeled with the guard *true* and are thus enabled along y.

($\Leftarrow$) Given a computation y of $(U_p', U_r')^{(1,1)}$, we construct a computation x of $(U_1, ..., U_k)^{(c_1,...,c_k)}$, such that $y[1,1]$ and $y[2,1]$ are stuttering equivalent to $x[p, c_p]$ and $x[r, c_r]$, respectively.

By the Soundness Lemma, it follows that there exists a finite computation path $u = u_0, ..., u_l$ of $(U_1, ..., U_k)^{(|U_1|,...,|U_k|)}$ starting at $\mathsf{i}^{(|U_1|,...,|U_k|)}$, such that $\forall j \in [1 : k] : \forall q_j \in P_j : \exists k \in [1 : |U_j|] : u_l[j,k] = q_j^k$, i.e., u_l has a copy of each reachable local state of each template. By the Completeness Lemma, $\bigcup_j Q_j = \bigcup_j P_j$ and so u_l has a copy of each local state in $\bigcup_j Q_j$. In other words, each state of $\bigcup_j Q_j$ is now 'flooded'. This enables every local transitions of $\bigcup_j R_j$ labeled with a guard whose expression involves a state of $\bigcup_j Q_j$. But these are precisely all the transitions of the simplified templates.

Then given a computation x of $(U_p', U_r')^{(1,1)}$ to get the desired computation y all we need to do is 'append' x at the end of u as described below. First we let all processes U_j^i, where $j \in [1 : k]$ and $i \in [1 : |U_j|]$, execute the same finite local computations as along u and then stutter in their respective final states, i.e., $\forall j \in [1 : k] : \forall k \in [1 : |U_j|] : x[j,k] = u[j,k](u_l[j,k])^\omega$. By the above remark, all template transitions in $R_p^{'} \cup R_r^{'}$ of processes $U_p^{c_p}$ and $U_r^{c_r}$ are now enabled in x_l. These two process can therefore mimic y by letting $x[p, c_p] = ((\mathsf{i}_p)^l)y[1,1]$, $x[r, c_r] = ((\mathsf{i}_r)^l)y[2,1]$. Thus for each $i \geq l$, transition $x_i \longrightarrow x_{i+1}$ is a valid global transition. Hence x is valid computation path. $\square$

The above result enables us to give a polynomial time decision procedure for properties of the form $\mathsf{A}h$ as we now show.

Theorem 4 (Efficient Decidability Theorem). *For systems with disjunctive guards and properties of the type $\bigwedge_{i_l} \mathsf{A}h(i_l)$, the PMCP is decidable in time quadratic in the size of the given family $(U_1, ..., U_k)$, where size is defined as $\sum_j(|S_j|+|R_j|)$, and linear in the size of the Büchi Automaton for $\neg h(1_l)$.*

Proof We first argue that we can construct the simplified system U_l' efficiently. By definition, $\forall j \geq 0 : P_l^j \subseteq P_l^{j+1}$. Let $P^i = \bigcup_l P_l^i$. Then, it is easy to see that, $\forall j \geq 0 : P^j \subseteq P^{j+1}$ and if $P^j = P^{j+1}$, then $\forall i \geq j : P^i = P^j$. Also, $\forall i : P^i \subseteq \bigcup_l S_l^{'}$. Thus to evaluate sets P_l^j, for all j, it suffices to evaluate them for values of $j \leq \sum_i |S_i|$. Furthermore, given P_l^j, to evaluate P_l^{j+1} it suffices to make a pass through all transitions leading to states in $S_l \setminus P_l^j$ to check if a guard leading to any of these states contains a state in $\bigcup_l P_l^j$. This can clearly be accomplished in time $\sum_j(|S_j| + |R_j|)$. The above remarks imply that evaluation of sets P_l^j, can be done in time $O((\sum_j(|S_j| + |R_j|))^2)$. Furthermore, given i, whether $U_i^{'}$ has a nontrivial strongly connected component can be decided in time $O(|S_i^{'}| + |R_i^{'}|)$ by constructing all strongly connected components of $U_i^{'}$. Thus, determining whether such an i exists can be done in time $O(\sum_j(|S_j|+|R_j|))$.

The Reduction Theorem reduces the PMCP to the model checking problem for the system $(U_l^{'}, V)^{(1,1)}$, where $V = U_r^{'}$ if for some $i \in [1 : k]$, the transition graph for $U_i^{'}$ has a nontrivial strongly connected component else $V = U_1^{'}$. Now, $(U_l^{'}, V)^{(1,1)} \models Ah(1_1)$ iff $(U_l^{'}, V)^{(1,1)} \models \neg E\neg h(1_1)$. Thus it suffices to check whether $(U_l^{'}, V)^{(1,1)} \models E\neg h(1_1)$, for which we use the automata-theoretic approach of [VW86]. We construct a Büchi Automaton $\mathcal{B}_{\neg h}$ for $\neg h(1_1)$, and check that the language of the product Büchi Automaton $\mathcal{P}$, of $(U_l^{'}, V)^{(1,1)}$ and $\mathcal{B}_{\neg h}$ is non-empty (cf [LP85]). Since the non-emptiness check for $\mathcal{P}$ can be done in time linear in the size of $\mathcal{P}$, and the size of $(U_l^{'}, V)^{(1,1)}$ is $O((\sum_j(|S_j| + |R_j|))^2)$, we are done.

4.3 Properties ranging over pairs of processes from two classes

Using similar kinds of arguments as were used in proving assertions in the sections 4.1 and 4.2, we can prove the following results.

Theorem 5 (Cutoff Theorem). *Let f be $\bigwedge_{i_l,j_m} \mathsf{A}h(i_l, j_m)$ or $\bigwedge_{i_l,j_m} \mathsf{E}h(i_l, j_m)$, where h is an LTL\X formula and $l, m \in [1 : k]$. Then*

$$\forall (n_1, \ldots, n_k) \succeq (1, \ldots, 1) : (U_1, \ldots, U_k)^{(n_1,\ldots,n_k)} \models f \text{ iff}$$
$$\forall (d_1, \ldots, d_k) \preceq (c_1, \ldots, c_k) : (U_1, \ldots, U_k)^{(d_1,\ldots,d_k)} \models f,$$

where the cutoff $(c_1, \ldots, c_k)$ is given by $c_l = |U_l| + 2$, $c_m = |U_m| + 2$ and for $i \neq l, m : c_i = |U_i| + 1$.

Theorem 6 (Reduction Theorem).
$(U_1, ..., U_k)^{(c_1,\ldots,c_k)} \models \bigwedge_{i_l,j_m} \mathsf{A}h(i_l, j_m)$ iff $(U_l^{'}, U_m^{'})^{(1,1)} \models \bigwedge_{i_l,j_m} \mathsf{A}h(i_l, j_m)$, where $c_l = |U_l| + 2, c_m = |U_m| + 2$ and $\forall i \neq l, m : c_i = |U_i| + 1$.

Again, we get the analogous Decidability Theorem and Efficient Decidability Theorem. Moreover, we can specialize these results to apply when l=m. This permits reasoning about formulas of the type $\bigwedge_{i_l \neq j_l} \mathsf{A}h(i_l, j_l)$ or $\bigwedge_{i_l \neq j_l} \mathsf{E}h(i_l, j_l)$, for properties ranging over all pairs of processes in a single class l.

5 Systems with Conjunctive Guards

The development of results for conjunctive guards closely resembles that for disjunctive guards.

Lemma 7 (Conjunctive Monotonicity Lemma).

(i) $\forall n \geq 1 : (V_1, V_2)^{(1,n)} \models \mathsf{E}h(1_2)$ *implies* $(V_1, V_2)^{(1,n+1)} \models \mathsf{E}h(1_2)$.
(ii) $\forall n \geq 1 : (V_1, V_2)^{(1,n)} \models \mathsf{E}h(1_1)$ *implies* $(V_1, V_2)^{(1,n+1)} \models \mathsf{E}h(1_1)$.

Proof For any computation x of $(V_1, V_2)^{(1,n)}$, there exists an analogous computation y of $(V_1, V_2)^{(1,n+1)}$ wherein the $(n+1)$st copy of template process V_2 stutters in its initial state and the rest of the processes behave as along x. Note that since, by definition of our system model, the initial state of each template appears in the expressions for all the conjunctive guards of $(V_1, V_2)^{(1,n+1)}$, process V_2^{n+1} stuttering in its initial state does not disable any transition fired by the other processes along y that was also fired along x. Thus y is a valid computation. □

Lemma 8 (Conjunctive Bounding Lemma).

(i) $\forall n \geq 2|V_2| + 1 : (V_1, V_2)^{(1,n)} \models \mathsf{E}h(1_2)$ *iff* $(V_1, V_2)^{(1,c_2)} \models \mathsf{E}h(1_2)$, *where* $c_2 = 2|V_2| + 1$.
(ii) $\forall n \geq 2|V_2| : (V_1, V_2)^{(1,n)} \models \mathsf{E}h(1_1)$ *iff* $(V_1, V_2)^{(1,2|V_2|)} \models \mathsf{E}h(1_1)$.

Proof

(i) ($\Rightarrow$) Let x be a full path of $(V_1, V_2)^{(1,n)}$. There are two cases to consider (a) x is an infinite computation, and (b) x is deadlocked.

First assume that x is an infinite computation of $(V_1, V_2)^{(1,n)}$. We show how to construct an infinite computation y of $(V_1, V_2)^{(1,c_2)}$ such that $y[2, 1]$ is stuttering equivalent to $x[2, 1]$. Towards that end, set $y[1, 1] = x[1, 1]$ and $y[2, 1] = x[2, 1]$. If one of $x[1, 1]$ and $x[2, 1]$ is an infinite local computation then the resulting computation y is also infinite. If however none of $x[1, 1]$ or $x[2, 1]$ is an infinite local computation then there exists $k \neq 1$ such that $x[2, k]$ is an infinite local computation. In that case, set $y[2, 2] = x[2, k]$, thus ensuring that y is infinite. We let the remaining copies of V_2 stutter in their respective initial states. Then using the fact that a process in its initial state does not disable any other process we can, as in the proof of Conjunctive Monotonicity Lemma, show that y is a valid computation.

Now consider the case when $x = x_0, ..., x_l$ is a deadlocked computation sequence of $(V_1, V_2)^{(1,n)}$. In constructing computation path y of $(V_1, V_2)^{(1,c_2)}$ apart from preserving the local computation path of V_2^1, modulo stuttering, we have to make sure that all process eventually deadlock along y. The first condition is ensured by projecting on

to the local computations of processes V_1^1 and V_2^1, i.e., setting $y[1,1] = x[1,1]$ and $y[2,1] = x[2,1]$.

To satisfy the second condition, we have to make sure that for each process V in $(V_1, V_2)^{(1,c_2)}$ there exists a set of process in local states that deadlocks every transition emanating from the local state of V in the last global state of y. Note that we just need one copy of each of these 'deadlocking' local states. However V might be in local state a in the last global state of y and, due to the irreflexive nature of the guards, the quantification being over all *other* processes, we might need another process in local state a to deadlock V. Thus in the last global state occurring along y, to deadlock all the processes we need at most two copies of each local state occurring therein. With this in mind, we project onto processes indices of V_2 making sure that in the resulting global state y_l we have at least one copy of each local state occurring in x_l and furthermore, exactly two copies of each local state that has two or more copies on y_l. As before, we let the remaining processes stutter in their respective initial states. This completes the construction and proves the result.

($\Leftarrow$) The proof follows by repeated application of the Conjunctive Monotonicity Lemma.

(ii) Similar to the above proof. □

We next present the Conjunctive Truncation Lemma which is analogous to the Disjunctive Truncation Lemma, in that it allows reduction in system size over multiple coordinates simultaneously (2 coordinates for notational brevity).

Lemma 9 (Conjunctive Truncation Lemma). $\forall n_1, n_2 \geq 1 : (U_1, U_2)^{(n_1,n_2)} \models \mathsf{E}h(1_2)$ *iff* $(U_1, U_2)^{(n_1',n_2')} \models \mathsf{E}h(1_2)$, *where* n_2' *is the minimum of* n_2 *and* $2|U_2| + 1$, *and* n_1' *is the minimum of* n_1 *and* $2|U_1|$.

Proof Idea
Use the Conjunctive Bounding Lemma and associativity of the $||$ operator. □

An easy corollary of the Conjunctive Truncation Lemma is the cutoff result for systems with conjunctive guards.

Theorem 7 (Conjunctive Cutoff Result). *Let* f *be* $\bigwedge_{i_l} \mathsf{A}h(i_l)$ *or* $\bigwedge_{i_l} \mathsf{E}h(i_l)$, *where* h *is a LTL*$\backslash X$ *formula and* $l \in [1:2]$. *Then*

$\forall (n_1, n_2) \succeq (1,1) : (U_1, U_2)^{(n_1,n_2)} \models f$ *iff*

$\forall (d_1, d_2) \preceq (c_1, c_2) : (U_1, U_2)^{(d_1,d_2)} \models f$,

where the cutoff (c_1, c_2) *is given by* $c_l = 2|U_l| + 1$, *and for* $i \neq l : c_i = 2|U_i|$.

More generally, for systems with $\mathsf{k} \geq 1$ class of processes we have

Theorem 8 (Conjunctive Cutoff Theorem). *Let* f *be* $\bigwedge_{i_l} \mathsf{A}h(i_l)$ *or* $\bigwedge_{i_l} \mathsf{E}h(i_l)$, *where* h *is a LTL*$\backslash X$ *formula and* $l \in [1:\mathsf{k}]$. *Then*

$\forall (n_1, ..., n_\mathsf{k}) \succeq (1, ..., 1) : (U_1, ..., U_\mathsf{k})^{(n_1,...,n_\mathsf{k})} \models f$ *iff*

$\forall (d_1, ..., d_\mathsf{k}) \preceq (c_1, ...c_\mathsf{k}) : (U_1, ..., U_\mathsf{k})^{(d_1,...,d_\mathsf{k})} \models f$,

where the cutoff $(c_1, ..., c_\mathsf{k})$ *is given by* $c_l = 2|U_l| + 1$, *and for* $i \neq l : c_i = 2|U_i|$.

Although the above results yield decidability for the PMCP in the Conjunctive guards case, the worst case complexity of the decision procedures may be exponential in the size of the given templates. We now show that if we limit path quantification to range over *infinite* paths only (i.e. ignore deadlocked paths); or *finite* paths only; then we can give an efficient decision procedure for this version of the PMCP. We use $\mathsf{A}_{\mathsf{inf}}$ for "for all infinite paths," $\mathsf{E}_{\mathsf{inf}}$ for "for some infinite path," $\mathsf{A}_{\mathsf{fin}}$ for "for all finite paths," and $\mathsf{E}_{\mathsf{fin}}$ for "for some finite path".

Theorem 9 (Infinite Conjunctive Reduction Theorem). *For any LTL\X formula h and $l \in [1:\mathsf{k}]$, we have*

(i) $\forall(n_1,\ldots,n_{\mathsf{k}}) \succeq (1,\ldots,1) : (U_1,...,U_{\mathsf{k}})^{(n_1,\ldots,n_{\mathsf{k}})} \models \bigwedge_{i_l} \mathsf{E}_{\mathsf{inf}} h(i_l)$, *iff* $(U_1,...,U_{\mathsf{k}})^{(c_1,\ldots,c_{\mathsf{k}})} \models \mathsf{E}_{\mathsf{inf}} h(1_l)$;

(ii) $\forall(n_1,\ldots,n_{\mathsf{k}}) \succeq (1,\ldots,1) : (U_1,...,U_{\mathsf{k}})^{(n_1,\ldots,n_{\mathsf{k}})} \models \bigwedge_{i_l} \mathsf{A}_{\mathsf{inf}} h(i_l)$, *iff* $(U_1,...,U_{\mathsf{k}})^{(c_1,\ldots,c_{\mathsf{k}})} \models \mathsf{A}_{\mathsf{inf}} h(1_l)$,

where $(c_1,...,c_{\mathsf{k}}) = (\underbrace{1,...,2}_{l},...,1)$.

Proof

By appeal to symmetry, to obtain (i), it suffices to establish that for each $(n_1,\ldots,n_{\mathsf{k}}) \succeq (1,\ldots,1) : (U_1,...,U_{\mathsf{k}})^{(n_1,\ldots,n_{\mathsf{k}})} \models \mathsf{E}_{\mathsf{inf}} h(1_l)$ iff $(U_1,...,U_{\mathsf{k}})^{(c_1,\ldots,c_{\mathsf{k}})} \models \mathsf{E}_{\mathsf{inf}} h(1_l)$. Using the duality between $\mathsf{A}_{\mathsf{inf}}$ and $\mathsf{E}_{\mathsf{inf}}$ on both sides of the latter equivalence, we can also appeal to symmetry to obtain (ii). We establish the latter equivalence as follows.

($\Rightarrow$) Let $x = x_0 \xrightarrow{in_0,g_0} x_1 \xrightarrow{in_1,g_1} \ldots$ denote an infinite computation of $(U_1,...,U_{\mathsf{k}})^{(n_1,\ldots,n_{\mathsf{k}})}$, where in_i denotes the index of the process that fires the local transition driving the system from global states x_i to x_{i+1} and g_i is the guard enabling the transition. Since x is infinite, it follows that there exists some process such that the result of projecting x onto that process results in a stuttering of an infinite local computation of the process. By appeal to symmetry, we can without loss of generality, assume that for each process class U_p, if a copy of U_p in $(U_1,...,U_{\mathsf{k}})^{(n_1,\ldots,n_{\mathsf{k}})}$ has the above property then that copy is in fact the concrete process U_p^1 in case $p \neq l$ and the concrete process U_p^2 in case $p = l$ with the local computation $x[l,1]$ being finite.

Define a (formal) sequence $y = y_0 \xrightarrow{in'_0,g'_0} y_1 \xrightarrow{in'_1,g'_1} \ldots$ by projecting each global state x_i onto process 1 coordinate for each class U_p for $p \neq l$ and onto process coordinates 1 and 2 for process class U_l to get a state y_i. We let $in'_i = 1_l$ if $in_i = 1_l$, $in'_i = 2_l$ if $in_i = 2_l$, else set $in' = \epsilon$, while g'_i is the syntactic guard resulting from g_i by deleting all conjuncts corresponding to indices not preserved in the projection. Then, by our construction and the fact that x was an infinite computation, we have that y denotes a stuttering of a genuine infinite computation of $(U_1,...,U_{\mathsf{k}})^{(c_1,\ldots,c_{\mathsf{k}})}$. To see this, note that for any i such that $y_i \neq y_{i+1}$, the associated (formal) transitions have their guard g'_i true, since for conjunctive guards g_i and their projections g'_i we have $x_i \models g_i$ implies $y_i \models g'_i$, and can thus fire in $(U_1,...,U_{\mathsf{k}})^{(c_1,\ldots,c_{\mathsf{k}})}$. For any stuttering i where $y_i = y_{i+1}$, the (formal) transition is labeled by $in' = \epsilon$.

Thus, given infinite computation path of $(U_1,...,U_{\mathsf{k}})^{(n_1,\ldots,n_{\mathsf{k}})}$, there exists a stuttering of an infinite computation path of $(U_1,...,U_{\mathsf{k}})^{(c_1,\ldots,c_{\mathsf{k}})}$, such that the local computation path of U_l^1 is the same in both. This path correspondence proves the result.

($\Leftarrow$) Let $y = y_0, y_1, ...$ be an infinite computation path of $(U_1, ..., U_k)^{(c_1,...,c_k)}$. Then, consider the sequence of states $= x_0, x_1, ...,$, where $x[l,1] = y[l,1]$, $x[l,2] = y[l,2]$ and $\forall (k,j) \neq (l,1),(l,2) : x(k,j) = (i_k^j)^\omega$. Let g_i be the guard labeling the local transition tr that causes global state y_i to transit to y_{i+1}. Then in state x_i all processes apart from the one executing (a copy of) tr are in their respective initial states. Since the guards do allow initial states of all template processes as non-blocking states in that their being present in the global state does not falsify any guards, we have $x_i \models g_i$.

Thus, given infinite computation path y of $(U_1, ..., U_k)^{(c_1,...,c_k)}$, there exists an infinite computation path x of $(U_1, ..., U_k)^{(n_1,...,n_k)}$, such that the local computation path of U_l^1 is the same in both. This path correspondence easily gives us the desired result. □

In a similar fashion, we may prove the following result.

Theorem 10 (Finite Conjunctive Reduction Theorem). *For any LTL\X formula h, and $l \in [1:k]$ we have*

(i) $\forall (n_1, \ldots, n_k) \succeq (1, \ldots, 1) : (U_1, ..., U_k)^{(n_1,...,n_k)} \models \bigwedge_{i_l} \mathsf{E}_{\mathsf{fin}} h(i_l)$, *iff* $(U_1, ..., U_k)^{(1,...,1)} \models \mathsf{E}_{\mathsf{fin}} h(1_l)$;

(ii) $\forall (n_1, \ldots, n_k) \succeq (1, \ldots, 1) : (U_1, ..., U_k)^{(n_1,...,n_k)} \models \bigwedge_{i_l} \mathsf{A}_{\mathsf{fin}} h(i_l)$, *iff* $(U_1, ..., U_k)^{(1,...,1)} \models \mathsf{A}_{\mathsf{fin}} h(1_l)$.

Note that the above theorem permits us to verify safety properties efficiently. Informally, this is because if there is a finite path leading to a 'bad' state in the system $(U_1, ..., U_k)^{(n_1,...,n_k)}$, then there exists a finite path leading to a bad state in the system $(U_1, ..., U_k)^{(1,...,1)}$. Thus, checking that there is no finite path leading to bad state in $(U_1, ..., U_k)^{(n_1,...,n_k)}$ reduces to checking it for $(U_1, ..., U_k)^{(1,...,1)}$. We can use this to obtain an Efficient Conjunctive Decidability Theorem. Moreover, the results can be readily extended to formulas with multiple indices as in the disjunctive guards case.

6 Applications

Here, we consider a solution to the *mutual exclusion* problem. The template process is given below.

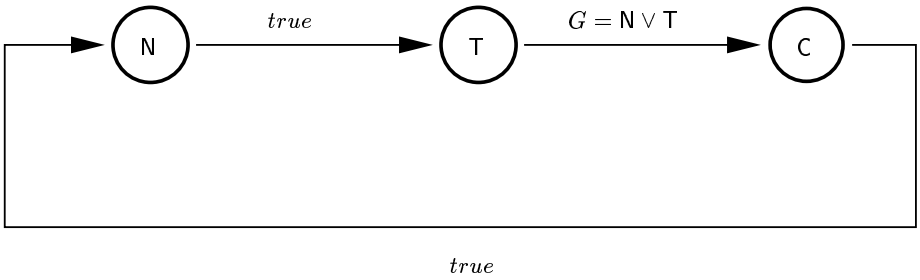

Fig 1.2 Template for Mutual Exclusion

Initially, every process is in local state N, the non-critical region. The guard $true$ is universally true irrespective of the current global state. If a process wants to enter the critical section C, it goes into the trying region T, which it can as guard $true$ is always enabled. Guard $G = N \vee T$, instantiated for process i of n processes, takes the conjunctive form $\bigwedge_{j \neq i}(N_j \vee T_j)$. When G is true, no other process is in the critical section, and the transition from T to C can be taken. Note that all guards are conjunctive with neutral (i.e., non-blocking) initial state N. Thus, by the Finite Conjunctive Reduction Theorem for multi-indexed properties, the PMCP for all sizes n with the mutual exclusion property $\bigwedge_{i,j,i \neq j} \mathsf{A_{fin}G}\neg(C_i \wedge C_j)$ can be reduced to checking a 2-process instance. Using the Conjunctive Cutoff Theorem, the starvation-freedom property $\bigwedge_i \mathsf{A}(\mathsf{G}(T_i \Rightarrow \mathsf{F}C_i))$ can be checked by a 7-process instance. In this simple example, mutual exclusion is maintained but starvation-freedom fails.

7 Concluding Remarks

The PMCP is, in general, undecidable [AK86]. However, under certain restrictions, a variety of positive results have been obtained. Early work includes [Lub84] which uses an abstract graph of exponential size "downstairs" to capture the behaviour of arbitrary sized parameterized asynchronous programs "upstairs" over 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. A semi-automated method requiring construction of a *closure process* which represents computations of an arbitrary number of processes is described in [CG87]; it is shown that, if for some k, $(C,U)^{(1,k)}$ is appropriately bi-similar to $(C,U)^{(1,k+1)}$, then it suffices to check instances of size at most k to solve the PMCP. But it is not shown that such a cutoff k exists and the method is not guaranteed to be complete. Kurshan and McMillan [KM89] introduce the related notion of a *process invariant* (cf. [WL89]). Ip and Dill [ID96] describe another approach to dealing with many processes using an abstract graph; it is sound but not guaranteed to be complete; [PD95] proposes a similar construction for verification of safety properties of cache coherence protocols, which is also sound but not complete. A theme is that most these methods suffer, first, from the drawback of being only partially automated and hence requiring human ingenuity, and, second, from being sound but not guaranteed complete (i.e., a path "upstairs" maps to a path "downstairs", but paths downstairs do not necessarily lift). Other methods can be fully automated but do not appear to have a clearly defined class of protocols on which they are guaranteed to terminate successfully (cf. [CGJ95], [Sis97], [Ver93]).

For systems comprised of CCS processes, German and Sistla [GS92] combine the automata-theoretic method with process closures to permit efficient solution to the PMCP for single index properties, modulo deadlock. But efficient solution is only yielded for processes in a single class. Even for systems of the form $(C,U)^{(1,n)}$ a doubly exponential decision procedure results, which likely limits its practical use. Emerson and Namjoshi [EN96] show that in a single class (or client-server) synchronous framework the PMCP is decidable but with PSPACE-complete complexity. Moreover, this framework is undecidable in the asynchronous case.

In some sense, the closest results might be those of Emerson and Namjoshi [EN95] who for the token ring model, reduce reasoning, for multi-indexed temporal logic formulas, for rings of arbitrary size to rings up to a small cutoff size. These results are significant in that, like ours, correctness over all sizes holds iff correctness of (or up to) the small cutoff size holds. But these results were formulated only for a single process class and, for a restricted version of the token ring model, namely one where the token cannot be used to pass values. Also, related are the results of Attie and Emerson [AE98]. In the context of program synthesis, rather than program verification, it is shown how certain 2-process solutions to synchronization problems could be inflated to n-process solutions. However, the correspondence is not an "iff", but is established in only one direction for conjunctive-type guards; disjunctive guards are not considered, nor are multiple process classes.

We believe that our positive results on the PMCP are significant for several reasons. Because the PMCP solves (a major aspect of) the state explosion problem and the scalability problem in one fell swoop, many researchers have attempted to make it more tractable, despite its undecidability in general. Of course, the PMCP seems to be prone to undecidability in practice as well, as is evidenced by the wide range of solution methods proposed that are only partially automated or incomplete or lack a well-defined domain of applicability. Our methods are fully automated returning a yes/no answer, they are sound and complete as they rely on establishing exact (up to stuttering) correspondences (yes upstairs iff yes downstairs). In many cases, our methods are efficient, making the problem genuinely tractable. An additional advantage, is that downstairs we have a small system of cutoff size that, but for its size, looks like a system of size n. This contrasts with methods that construct an abstract graph downstairs which may have a complex and non-obvious organization.

References

[AE98] P.C. Attie and E.A. Emerson. Synthesis of concurrent systems with many similar processes. In *ACM Transactions on Programming Languages and Systems*, volume 20(1), pages 51–115, 1998.

[AK86] K. Apt and D. Kozen. Limits for automatic verification of finite-state concurrent systems. In *Information Processing Letters*, volume 15, pages 307–309, 1986.

[BCG89] M.C. Browne, E.M. Clarke, and O. Grumberg. Reasoning about networks with many identical finite state processes. In *Information and Control*, volume 81(1), pages 13–31, 1989.

[CE81] E.M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, LNCS 131, pages 52–71, 1981.

[CG87] E.M. Clarke and O. Grumberg. Avoiding the state explosion problem in temporal logic model checking algorithms. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 294–303, 1987.

[CGJ95] E.M. Clarke, O. Grumberg, and S. Jha. Verifying parameterized networks using abstracion and regular languages. In *6th International Conference on Concurrency Theory*, LNCS 962, pages 395–407, 1995.

[EK03] E.A. Emerson and V. Kahlon. Model checking guarded protocols. In *Eighteenth Annual IEEE Symposium on Logic in Computer Science*, pages 361–370, 2003.

[EN95] E.A. Emerson and K.S. Namjoshi. Reasoning about rings. In *22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 85–94, 1995.

[EN96] E.A. Emerson and K.S. Namjoshi. Automatic verification of parameterized synchronous systems. In *Computer Aided Verification, Proceedings of the 8th International Conference*, LNCS 1102, pages 87–98, 1996.

[ES93] E.A. Emerson and A.P. Sistla. Symmetry and model checking. In *Computer Aided Verification, Proceedings of the 5th International Conference*, LNCS 697, pages 463–478, 1993.

[GS92] S.M. German and A.P. Sistla. Reasoning about systems with many processes. In *J. ACM*, volume 39(3), pages 675–735, 1992.

[ID96] C. Ip and D. Dill. Verifying systems with replicated components in mur*phi*. In *8th International Conference on computer Aided Verification*, LNCS 1102, pages 147–158, 1996.

[KM89] R.P. Kurshan and L. McMillan. A structural induction theorem for processes. In *Proceedings of the Eight Annual ACM Symposium on Principles of Distributed Computing*, pages 239–247, 1989.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specifications. In *12nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 97–107, 1985.

[Lub84] B. Lubachevsky. An approach to automating the verification of compact parallel coordination programs. In *Acta Informatica*, volume 21, 1984.

[McM99] K. McMillan. Verification of infinite state systems by compositional model checking. In *10th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 219–234, 1999.

[PD95] F. Pong and M. Dubois. A new approach for the verification of cache coherence protocols. In *IEEE Transactions on Parallel and Distributed Systems*, 1995.

[Sis97] A. P. Sistla. Parameterized verification of linear networks using automata as invariants. In *9th International Conference on Computer Sided Verification*, LNCS 1254, pages 412–423, 1997.

[Suz88] I. Suzuki. Proving properties of a ring of finite state systems. In *Information Processing Letters*, volume 28, pages 213–314, 1988.

[Ver93] I. Vernier. Specification and verification of parameterized parallel programs. In *Proceedings of the 8th International Symposium on Computer and Information Sciences*, pages 622–625, 1993.

[VW86] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *IEEE Symposium on Logic in Computer Science*, pages 332–344, 1986.

[WL89] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Automatic Verification Methods for Finite State Systems*, LNCS 407, pages 68–80, 1989.