Model Checking of Systems Employing Commutative Functions

A. Prasad Sistla, Min Zhou, and Xiaodong Wang

University of Illinois at Chicago

Abstract. The paper presents methods for model checking a class of possibly infinite state concurrent programs using various types of bi-simulation reductions. The proposed methods work for the class of programs in which the functions that update the variables are mutually commutative. A number of bi-simulation relations are presented for such systems. Explicit state model checking methods that employ on-the-fly reductions with respect to these bi-simulations are given. Some of these methods have been implemented and have been used to verify some well known protocols that employ integer variables. Various applications of the methods and optimization techniques for special cases are also given in appendix.

1 Introduction

Two of the bottlenecks that hinder wider applicability of model checking approach is the state explosion problem and its less effectiveness in handling infinite state systems. In this paper, we present an approach for model checking that works for certain classes of infinite state systems and that can also be used to contain the state explosion problem.

One standard model checking method, employed often, is to construct the reachability graph of the given program and then check the correctness property against this graph. One way of reducing the size of the explored graph is to employ a reduction with respect to a bi-simulation relation U on the states of the reachability graph. Such a relation U is either known a priori through an implicit representation or has been computed by other means.

In this paper, we give a method that does not require a priori computation of a bi-simulation relation. Instead, we give a sufficient condition on any two states to determine if they are bi-similar. This condition requires equivalence of certain predicates associated with the two states. In fact, we present a number of bi-simulation relations that can be used in on-the-fly model checking methods. Our approach works for certain classes of programs that employ commutative unary functions for updating variables. Since bi-similarity of two states is based on the future behavior from these states, in general, it is not possible to check their bi-similarity by looking only at the values of the variables in these states.

We assume that the concurrent program is given by a Transition Diagram (TD) [11] which is an edge labeled directed graph. Each edge label consists of a condition, called guard, and an action which is a concurrent assignment of values to variables. We consider a class of TDs, called simple TDs, in which an expression that is assigned to a variable x is either a constant, or a variable, or of the form $f(x)$ where f is a unary function. Further more, we require that the functions that are used be mutually commutative, that is, for any two functions f, g, $fg = gf$. (Note that such TDs are as powerful as Turing M/Cs).

Our approach works as follows. First we preprocess the TD and compute a set of *predicate templates* with respect to each node q in the TD. (A predicate template is a predicate together with a function that renames some of its variables). These sets of predicate templates are computed, using a terminating fix-point computation on the graph of the TD, from guards of the transitions of the TD and from predicates that appear in the correctness formula. In the second step, the reachability graph is constructed in a symbolic form. Each of its states consists of a node in the TD and other components that give the values of the variables in symbolic form. We define an equivalence relation, $\sim_0$, on the states by instantiating the predicate templates associated with the corresponding TD node. Two states are equivalent if they are at the same TD node and the instantiations of the predicate templates in both the states are equivalent. We show that this equivalence relation is a bi-simulation. In general checking equivalence of predicates may require a theorem prover. However, for certain types of programs, such as those that use integer variables and additions of constants as functions, this equivalence can be checked efficiently if the predicates only involve standard comparison operators such as $<, >$, etc.

The requirements for the bi-simulation $\sim_0$ can some times be too strong. In order for two symbolic states s, t at a node q to be related by $\sim_0$, we require that the instantiations, of each predicate template *pt* associated with q, in the states s and t be equivalent. Each such predicate template *pt* corresponds to a guard of a transition of the TD from some node r or to an atomic predicate in the correctness formula. Suppose that none of the guards of the transitions entering node r are ever satisfiable; then, we don't need to require equivalence of the instantiations of *pt* with respect to both s and t because r will be never reached from either s or t. As a consequence, we can relax the equivalence requirement as follows. Suppose e is a transition entering the node r; then we require equivalence of the instantiations of *pt* only if the transition e is enabled with respect to both the states s and t. Thus we require conditional equivalence of template instantiations. The above relaxation in the requirement is done with respect to all the transitions entering node r and for every such node. The resulting binary relation $\sim_1$ on symbolic states is also going to be a bi-simulation.

The above notion of relaxing the requirement with respect to edges entering each node can be generalized to paths of length i entering each node.. When we do this, we get the relations $\sim_i$ for each $i > 0$. The relation $\sim_0$, defined earlier, can be considered to be the relation when we consider paths of length zero, i.e. null paths, entering a node. We show that each $\sim_i$ is a bi-simulation and that $\sim_i \subseteq \sim_{i+1}$ for each $i \geq 0$. Thus we get a chain of non-decreasing bi-simulations. For each i, we also show that there exists a TD for which $\sim_i$ is strictly contained in $\sim_{i+1}$. In fact, we can get a TD for which $\sim_i \subset \sim_{i+1}$ for every $i \geq 0$. It is to be noted that using $\sim_{i+1}$ gives us a smaller bi-simulation reduction, however there will be more conditional equivalence checks for $\sim_{i+1}$ than for $\sim_i$.

We also show, in the appendix, that when we consider TDs over integer domain, i.e., when the variables range over integer and the functions that update variables only add constants, then we don't need to use the symbolic graph and instead can use the standard reachability graph and define the bi-simulations on this graph.

In all the above approaches, checking equivalence of predicates or conditional equivalence of predicates assumes an implicit universal quantification over their free variables. We show in appendix that by analyzing the TD in advance, i.e., by performing static analysis, the domain over which the universal quantifiers range can be reduced. (In fact, this domain itself can be defined by a formula.) This allows more states to be considered equivalent. When we do this over the symbolic graph, we get the bi-simulation relations $\sim'_i$ for each $i \geq 0$. For TDs on the integer domains, as above, we can define these bi-simulations on the reachability graph itself. We show, in the appendix, that our techniques can be used to show decidability of the reachability problems for simple sub-classes of Hybrid automata [15, 19] when they are restricted to integer domains.

All the above bi-simulations preserve correctness under fairness also. We have implemented the above methods and applied them to examples such as the sliding window protocol, etc. Experimental results showing the effectiveness of our approach are presented.

In summary, we have given methods based on bi-simulation reductions for model checking a class of TDs where the update functions satisfy certain commutativity conditions. We define a sequence of non-decreasing bi-simulations. These bi-simulations can be checked on-the-fly as the reachability graph is constructed. The methods employ static analysis for defining different predicate templates that are used for checking the bi-simulation of states. These bi-simulations preserve correctness under fairness, and hence the corresponding bi-simulation reductions can be used to verify both safety and liveness properties in their full generality. We also show some of the results of the paper can be used to prove the decidability of certain hybrid automata when restricted to the integer domains. Some of the proposed methods are implemented and have been used to verify real protocols. To the best our knowledge, this is the first time when these special types of TDs have been considered in a comprehensive manner and such a wide spectrum of bi-simulations are proposed and considered for model checking.

The paper is organized as follows. Section 2 contains definitions and notation. Section 3 presents our method based on the bi-simulation relation $\sim_0$. Section 4 defines the bi-simulation relations with respect to paths of the TD, i.e. it defines the relations $\sim_i$ for $i > 0$ and presents results relating them. Section 5 presents experiment results. Section 6 contains conclusions and comparison to related work. Section 7.1 in appendix presents extensions to the method. Section 7.2 in appendix shows how hybrid automata over discrete time domain can be handled and it also shows how the commutativity requirement can be relaxed. The proofs for theorems are given in section 7.3 in appendix.

2 Definitions and Notation

2.1 Transition Diagram

We use *Transition Diagram* (TD) to model a concurrent system. Formally, a TD is a triple $G = (Q, X, E)$ such that Q denotes a set of nodes, X is a set of variables, and E is a set of transitions which are quadruples of the form $\langle q, C, \Lambda, q' \rangle$ where $q, q' \in Q$, C is a condition involving the variables in X and Λ is a set of assignments of the form $x := \rho$ where $x \in X$ and ρ is an expression involving the variables in X. For a transition $\langle q, C, \Lambda, q' \rangle$, we call C the condition part or *guard* of the transition and Λ the action part of the transition and we require that Λ contains at most one assignment for each variable. For any node q of G, we let $guards(q)$ denote the set of guards of transitions from the node q. We also let $guards(G)$ denote the set of guards of all transitions of G.

An evaluation h of a set of variables X is a function that assigns type-consistent values to each variable in X. A state of a TD $G = (Q, X, E)$ is a pair (q, h) where $q \in Q$ and h is an evaluation of X. We say that a transition $e = (q_1, C, \Lambda, q_2)$ is enabled in the state (q, h) if $q = q_1$ and the condition C is satisfied by h, i.e., the values of the variables given by h satisfy C. We say that a state (q', h') is obtained from (q, h) by executing the transition e if e is enabled in (q, h), $q' = q_2$ and the following property is satisfied: for each variable $x \in X$, if there is an assignment of the form $x := \rho$ in Λ then $h'(x) = h(\rho)$, otherwise $h'(x) = h(x)$.

A path in G from node q to node r is a sequence of transitions starting with a transition from q and ending with a transition leading to r such that each successive transition starts from the node where the preceding transition ends. Let $\pi = e_0, e_1, ..., e_{m-1}$ be a path in G from node q and let $s_0 = (q, h_0)$ be a state. We say that π is *feasible* from s_0 if there exists a sequence of states $s_1, ..., s_m$ such that for each i, $0 \leq i < m$, the transition e_i is enabled in s_i and state s_{i+1} is obtained by executing e_i in the state s_i. In this case, i.e., when π is feasible from s_0, we say that s_m is the state obtained by executing the path π from s_0.

The left part of figure 1 shows a TD with node set $\{0, 1, 2\}$, variable set $\{a, b, x, y\}$ and transition set $\{t_1, t_2, t_3, t_4\}$. Notice that the transition t_1 and t_2 both have empty guards meaning that they are always enabled. It is easy to see that the reachability graph from an initial state may be infinite since x, y can grow arbitrarily large.

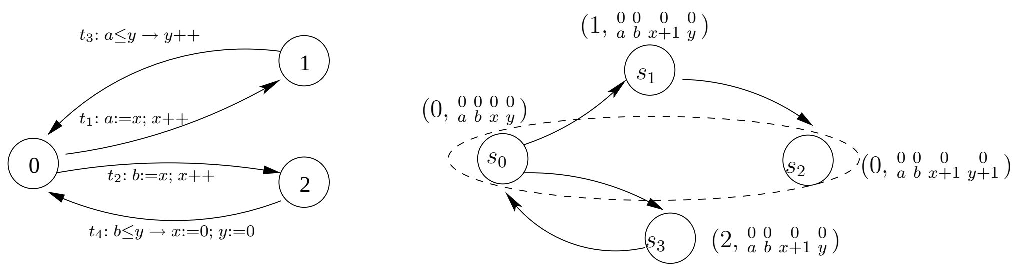


Fig. 1. Example of a TD and its reduced symbolic state graph

Commutativity Requirement

In this paper, we consider the TDs whose action parts only have assignments of the following forms: $x := c$ where c is a constant, or $x := \psi(x)$, or $x := y$ where y is another variable of the same type as x. We require all functions that are applied to variables of the same type to be commutative. We call such TDs as *simple* TDs. In section 7.2, we show how the commutativity requirement can be further relaxed.

2.2 Kripke Structures, Bi-simulation, etc.

A labeled Kripke structure H over a set of atomic propositions AP and over a set of labels Σ is a triple (S, R, L) where S is a set of states, $R \subseteq S \times \Sigma \times S$ and $L : S \rightarrow 2^{AP}$ associates each state with a set of atomic propositions. The Kripke structure H is said to be deterministic if for every $s \in S$ and every $\alpha \in \Sigma$ there exists at most one $s' \in S$ such that $(s, \alpha, s') \in R$.

For the Kripke structure $H = (S, R, L)$, an execution σ is an infinite sequence $s_0, e_0, s_1, e_1, ..., s_i, e_i, ...$ of alternating states and labels in Σ such that for each $i \geq 0$, $(s_i, e_i, s_{i+1}) \in R$. A finite execution is a finite sequence of the above type ending in a label in Σ. Corresponding to the execution σ that is finite or infinite, let $trace(\sigma)$ denote the sequence $L(s_0), e_0, ..., L(s_i), e_i, ...$ A finite trace from state s is the sequence $trace(\sigma)$ corresponding to a finite execution from s. The length of a finite trace is the number of transitions in it. For any integer $k > 0$, let $Finite\_Traces_k(H, s)$ denote the set of finite traces of length k from s.

Let $H = (S, R, L)$ and $H' = (S', R', L')$ be two structures over the same set of atomic propositions AP and the same set Σ of labels. A relation $B \subseteq S \times S'$ is a bi-simulation between H and H' iff for all $s \in S$ and $s' \in S'$, if $(s, s') \in B$, then $L(s) = L(s')$ and the following conditions hold: (a) for every

$(s, \alpha, s_1) \in R$, there exists a state $s'_1 \in S'$ such that $(s', \alpha, s'_1) \in R'$ and $(s_1, s'_1) \in B$; (b) similarly, for every $(s', \alpha, s'_1) \in R'$, there exists a state $s_1 \in S$ such that $(s, \alpha, s_1) \in R$ and $(s_1, s'_1) \in B$.

Let $G = (Q, X, E)$ be a TD and $Reach(G, u) = (S, R, L)$ denote the Kripke structure over the set of atomic propositions AP and the set of labels E defined as follows: S is the set of reachable states obtained by executing the TD G from u; R is the set of triples (s, e, s') such that the transition $e \in E$ is enabled in state s and s' is obtained by executing e in state s; for any $s \in S$, $L(s)$ is the set of atomic propositions in AP that are satisfied in s. It is not difficult to see that $Reach(G, u)$ is a deterministic structure.

Let B be a bi-simulation relation from $Reach(G, u)$ to itself. Instead of constructing $Reach(G, u)$, we can construct a smaller structure using the relation B. We incrementally construct the structure by executing G starting from u. Whenever we get a state w by executing a transition from an already reached state v, we check if there exists an already reached state w' such that (w, w') or (w', w) is in B; if so, we simply add an edge to w' or else we include w into the set of reached states and add an edge to w. This procedure is carried until no more new nosed can be added to the set of reached states. We call the resulting structure as the bi-simulation reduction of $Reach(G, u)$ with respect to B. This reduction has the property that no two states in it are related by B. The number of states in this reduction may not be unique and may depend on the the order of execution of the enabled transitions. However, if B is an equivalence relation then the number of states in the reduction is unique and equals the number of equivalence classes of S with respect to B.

Let G be a TD that captures the behavior of a concurrent program. The Kripke structure $Reach(G, u)$ is deterministic. An infinite execution σ is said to be weakly fair if every process which is enabled continuously from a certain point in σ is executed infinitely often. Let $Fair\_traces(s)$ denote the set of all $trace(\sigma)$ where σ is an infinite weakly fair execution s. For any bi-simulation B from $Reach(G, u)$ to itself, it is easy to show that for every $(s, t) \in B$, $Fair\_traces(s) = Fair\_traces(t)$. This condition holds many other fairness conditions such as strong fairness, etc.

We use the temporal logic CTL* to specify properties of $Reach(G, u)$. Each atomic proposition in the formulas is a predicate involving variables in X or the special variable lc which refers to the nodes of G. We let AP be the set of predicates that appear in the temporal formula that we want to check. For any formula or predicate p, we let $var(p)$ denote the set of variables appearing in it.

If K is a reduction of $Reach(G, u)$ with respect to a bi-simulation relation then a state which is present in both $Reach(G, u)$ and K satisfies the same set of CTL* formulas in both structures even if we restrict the paths to fair paths. Also, any two states in $Reach(G, u)$ that are bi-similar to each other satisfy the same set of CTL* formulas.

We also use the following notation. If Φ represents an expression, then $\Phi\{\beta/\alpha\}$ is the expression obtained from Φ by substituting β for α.

2.3 Symbolic State graph

Let $G = (Q, X, E)$ be a TD, $u = (q_0, h_0)$ be the initial state of G and AP be the set of predicates that appear in the temporal formula to be checked. We execute G symbolically starting with u, to obtain a structure $Sym\_Reach(G, u) = (S', R', L')$. We call the structure $Sym\_Reach(G, u)$ as the *symbolic graph* and the states of S' *symbolic states* since the variables are represented by expressions. (It should be noted that our use of the term symbolic state is different from the traditional use where it is meant to be some representation for sets of actual states). Each state s in S' is a triple of the form $(s.lc, s.val, s.exp)$ where $s.lc \in Q$, $s.val$ is an evaluation of the variables in X and $s.exp$ is a function that assigns each variable x an expression which involves only the variable x. Intuitively,

$s.lc$ denotes the node in Q where the control is, $s.val(x)$ denotes the latest constant assigned to x and $s.exp(x)$ denotes the composition of functions that were applied to x since then. We associate each symbolic state s with a state $act\_state(s)$ of G defined as follows: $act\_state(s) = (q, h)$ where $q = s.lc$ and $h(x) = s.exp(x)\{s.val(x)/x\}$ for each $x \in X$; that is the value of a variable x is obtained by evaluating $s.exp(x)$ after substituting $s.val(x)$ for x in the expression. We say that a transition e is enabled in a symbolic state s if it is enabled in the corresponding actual state, i.e., it is enabled in $act\_state(s)$.

The successor states of a symbolic state s are the states obtained by enabled transitions in s. Assume that $e = (q, C, \Lambda, q')$ is enabled in s. The new symbolic state s' obtained by executing e from s is defined as follows: $s'.lc = q'$ and for each variable x, if there is no assignment to x in Λ then $s'.val(x) = s.val(x)$ and $s'.exp(x) = s.exp(x)$. If there is an assignment of the form $x := c$ where c is a constant then $s'.val(x) = c$ and $s'.exp(x) = x$. If there is an assignment of the form $x := \psi(x)$ in Λ then $s'.val(x) = s.val(x)$ and $s'.exp(x) = \psi(s.exp(x))$; that is the value remains unchanged and the new expression is obtained by applying the function ψ to the old expression. If there is an assignment of the form $x := y$ in Λ then $s'.val(x) = s.val(y)$ and $s'.exp(x) = s.exp(y)\{x/y\}$; that is the value of $s.val(y)$ is copied and the expression of y in s is also copied after replacing every y by x in the expression. If s' is obtained by executing an enabled transition e from a state s in S', then s' is a state in S' and $(s, e, s') \in R'$. Also for any $s \in S'$, $L'(s) = L(act\_state(s))$.

Consider the TD given in the left part of figure 1 with initial value $a = b = x = y = 0$. We represent each variable $v \in X$ as a pair $(v.val, v.exp)$. The actual value of v is $v.exp\{v.val/v\}$. For figure 1, the initial state s_0 is $(0, \substack{0\\a}\,\substack{0\\b}\,\substack{0\\x}\,\substack{0\\y})$, where the first 0 denotes the node, the vectors $(0, 0, 0, 0)$ and (a, b, x, y) represent the functions $s_0.val$ and $s_0.exp$ respectively. In s_0, transition t_1 and t_2 are enabled. Suppose we execute t_1 from s_0 and get state s_1. The node in s_1 is 1. For variable a, since x is assigned to it, we copy $x.val$ and $x.exp$ to $a.val$ and $a.exp$ respectively. For variable x, which is updated by a function of itself, we keep $x.val$ as before and change $x.exp$ from x to $x+1$ according to the updating function. Since there is no assignment to b and y in t_1, their val and exp remain unchanged. So, s_1 is $(1, \substack{0\\a}\,\substack{0\\b}\,\substack{0\\x+1}\,\substack{0\\y})$. We know t_3 is enabled in s_1 since the actual values of a, y satisfy the guard. We execute t_3 from s_1 and get $s_2 = (0, \substack{0\\a}\,\substack{0\\b}\,\substack{0\\x+1}\,\substack{0\\y+1})$ similarly. Similarly, executing t_2 from s_0 we get s_3. Executing t_4 from s_3 we get s_0; notice that since x is assigned 0, in the successor state s_0 we have $s.val(x) = 0$ and $s.exp(x) = x$ and same holds for variable y.

Lemma 1 *Let G be a TD, u be a state of G. Then the binary relation $\{(act\_state(s), s) : s$ is a symbolic state in $Sym\_Reach(G, u)\}$ is a bi-simulation between $Reach(G, u)$ and $Sym\_Reach(G, u)$.*

3 Our method

3.1 Intuitive Description

Let $G = (Q, X, E)$ be a TD. Recall that AP is the set of predicates appearing in the temporal formula. We motivate our definition of the bi-simulation relation and give an intuitive explanation for the commutativity requirement. For ease of explanation, assume that all the assignments in the transitions of G are of the form $x := \psi(x)$. Also assume that all the predicates in $guards(G) \cup AP$ have at most one variable. It is not difficult to see that any two states $s = (q, h)$ and $t = (q, h')$ satisfying the following conditions are bisimilar in $Reach(G, u)$: (i) for every path π from node q, π is feasible in s iff it is feasible from t; (ii) for every path π from node q to any node r such that π is feasible from s, if s', t' are the states obtained by executing π from s, t respectively then s', t' satisfy the same predicates in $guards(r) \cup AP$. For any path π in G and variable $x \in X$, let $F_{\pi,x}$ denote the composition of the functions that update the variable x in the path π where the composition is taken in the order they appear on the path. Using the above observation, it can be seen that the following relation U over the states of $Reach(G, u)$ is a bi-simulation. U is the set of all pairs (s, t) of states where $s = (q, h)$, $t = (q, h')$ for some $q \in Q$ and some evaluations h, h' such that the following condition is satisfied: for every node r and for every path π from q to r in G, and for every predicate $p(x) \in guards(r) \cup AP$, the truth values $p(x)\{F_{\pi,x}(h(x))/x\}$ and $p(x)\{F_{\pi,x}(h'(x))/x\}$ are the same.

Analogous to the relation U, we can define a relation V over the states of the symbolic graph $Sym\_Reach(G, u)$ as follows. V is the set of all pairs (v, w) of symbolic states where $v.lc = w.lc = q$ for some $q \in Q$, $v.val(x) = w.val(x) = c$ for some constant c and for every node r to which there is a path in G from node q, and for every predicate $p(x) \in guards(r) \cup AP$ the following condition (A) is satisfied: (A) for every path π from q to r in G, the truth values $p(x)\{F_{\pi,x}(\rho_1(c))/x\}$ and $p(x)\{F_{\pi,x}(\rho_2(c))/x\}$ are the same where $\rho_1(x), \rho_2(x)$ are the expressions $v.exp(x)$, $w.exp(x)$ respectively. Note that $\rho_1(c)$ is the value of $\rho_1(x)$ when c is substituted for x. It is not difficult to see that V is also a bi-simulation over $Sym\_Reach(G, u)$. Due to the commutativity requirement on the functions that update variables, we can see that $F_{\pi,x}(\rho_1(c)) = \rho_1(F_{\pi,x}(c))$ and a similar equality holds for the state w. As a consequence, condition (A) can be rewritten as follows: (B) for every path π from q to r in G, the truth values $p(x)\{\rho_1(F_{\pi,x}(c))/x\}$ and $p(x)\{\rho_2(F_{\pi,x}(c))/x\}$ are equal. Now we see that condition (B) is automatically satisfied if the two predicates $p(x)\{\rho_1(x)/x\}$ and $p(x)\{\rho_2(x)/x\}$ are equivalent (this is seen by substituting the value $F_{\pi,x}(c)$ for x in these two predicates). As a consequence we can replace condition (A) by condition (C) which requires the equivalence of the above two predicates. Checking condition (A) requires considering every path from q to r which is not needed for checking (C). (Using (C) in place of (A) might give us a smaller relation ρ_1; this disadvantage is minimized using the extensions of section 7.1). The above argument holds even if we have predicates with more than one free variable in $guards(G) \cup AP$. However, if we have other types of assignments to variables then we need to rename some of the variables to obtain the predicates whose equivalence needs to be checked. This is done by computing a set of predicate templates with respect to each node in G.

3.2 Predicate Templates

To define the bi-simulation, we associate a set of predicate templates, denoted $ptemplates(q)$, with each node q in the TD. Intuitively, $ptemplates(q)$ is the set of pairs of predicates and renaming functions on their variables; roughly speaking, our bi-simulation condition requires that the predicates, obtained by renaming the variables and substituting them by the corresponding expressions in two symbolic states, should be equivalent. Formally, a predicate template is a pair (p, f) where p is a predicate and f, called renaming function, is a total function from $var(p)$ to $X \cup \{*\}$.

First we need the following definition. Let π be a path in G. Each such path denotes a possible execution in G. With respect to π, we define a function $depends_\pi$ from X to $X \cup \{*\}$. Intuitively, if $depends_\pi(x)$ is a variable, say y, then this denotes that the value of x at the end of the execution of π depends on the value of y at the beginning of this execution; otherwise, i.e., $depends_\pi(x) = *$, the value of x at the end of π does not depend on the value of any variable at the beginning of π; for example, this happens if x is assigned a constant some where along π. We define $depends_\pi$ inductively on the length of π. If π is a single transition $\langle q, C, \Lambda, q' \rangle$ then $depends_\pi(x)$ is given as follows: if Λ has the assignment $x := y$ then $depends_\pi(x) = y$; if Λ has no assignment to x or has an assignment of the form $x := \psi(x)$ then $depends_\pi(x) = x$; when x is assigned a constant, $depends_\pi(x) = *$. If π is the path consisting of π_1 followed by π_2 then $depends_\pi$ is defined as follows: for each $x \in X$, if $depends_{\pi_2}(x)$ is a variable then $depends_\pi(x) = depends_{\pi_1}(depends_{\pi_2}(x))$, otherwise $depends_\pi(x) = *$. For the TD given in figure 1 and the path π given by the single transition from node 0 to 1, we see that $depends_\pi(a) = x$.

For a node q, $ptemplates(q) = \{(p, depends_\pi) : \pi$ is a path from node q to some node r and $p \in guards(r) \cup AP\}$. Although the number of paths from q can be infinite, the number of functions $depends_\pi$ and hence $ptemplates(q)$ is a bounded set. We can compute $ptemplates(q)$ without examining all the paths from q as follows.

For a template (p, f) and a set of assignments Λ, let $(p, f)_\Lambda$ be the template (p, f') where f' is given as follows: (note that (p, f') is different from the the weakest precondition of p with respect to Λ)

- if $f(x) = *$, then $f'(x) = *$.
- if $f(x) = y$ where $y \in X$, then if the action part Λ has
  - no assignment for y or an assignment of the form $y := \psi(y)$, then $f'(x) = y = f(x)$.
  - an assignment of the form $y := z$, then $f'(x) = z$.
  - an assignment of the form $y := c$ where c is a constant, then $f'(x) = *$.

Let f_{id} be the identity function. For each node $q \in Q$, the set $ptemplates(q)$ is the least fix point solution for the variables $temp(q)$ in the following set of equations:

$$\begin{aligned} temp(q) = &\{(p, f_{id}) | p \in AP \lor p \in guards(q)\} \cup \\ &\{(p, f)_\Lambda | (p, f) \in temp(q') \land \exists (q, C, \Lambda, q') \in E\} \end{aligned}$$

Consider the system given in figure 1. Suppose we want to check the formula $\forall \Box (x \geq y)$. Let p_0 denote $x \geq y$, p_1 denote $a \leq y$, p_2 denote $b \leq y$. Template (p_0, f_{id}) will appear in templates of each location since it is in AP. (p_1, f_{id}) will appear in $ptemplates(1)$ since p_1 is in $guards(1)$. In the remainder of our description, we will also represent a predicate template (p, f) where f maps v_1, v_2 to z_1, z_2 respectively by the tuple $(p, v_1 : z_1, v_2 : z_2)$. Suppose t is a transition, let Λ_t denote the action part of t. By definition, $(p_1, f_{id})_{\Lambda_{t_1}}$ will appear in $ptemplate(0)$. Since Λ_{t_1} contains the assignments $a := x$ and $x := x + 1$, the template $(p_1, f_{id})_{\Lambda_{t_1}}$ is given by $(p_1, a : x, y : y)$. Using transition t_4, we see that the template $(p_0, x : *, y : *)$ is in $ptemplates(2)$; note that in this template both the variables are mapped to $*$ since both these variables are assigned constant values in t_4. By doing this, eventually, we will have

$$\begin{aligned} ptemplates(1) = &\{(p_0, f_{id}), (p_1, f_{id}), \\ &(p_1, a : x,\ y : y), (p_2, b : x,\ y : y)\} \end{aligned}$$

We have only given the templates associated with node 1 and even from this we omitted the templates whose renaming function maps all the variables to $*$. From the above definition, we see that $ptemplates(q)$ contains all the templates in $\{(p, f_{id}) | p \in AP \cup guards(q)\}$.

We can use standard fix point algorithm to compute the set $ptemplates(q)$ for each $q \in Q$. It is easy to see that the algorithm terminates since the total number of predicate templates is bounded.

Let N_n, N_t, N_v be the number of nodes in G, number of transitions in G and the number of variables respectively. Similarly let N_p and N_a respectively be the number of predicates in

$guards(G) \cup AP$ and the maximum number of variables appearing in any predicate. Since the maximum number of renaming functions is $(N_v + 1)^{N_a}$, we see that the maximum number of predicate templates is $N_p \cdot (N_v + 1)^{N_a}$. Thus the number of the outer iterations of the fix point computations is at most $N_n \cdot N_p \cdot (N_v + 1)^{N_a}$. The time complexity of the algorithm is $O(N_n \cdot N_p \cdot N_t \cdot (N_v + 1)^{N_a})$. Thus we see that the number of predicates is exponential in the number of variables that can appear in a predicate. In most cases we have unary or binary predicates and hence the complexity will not be a problem. Also this worst case complexity occurs when every variable is assigned to every other variable directly or indirectly. We believe that this is a rare case.

3.3 Definition of the bi-simulation relation

Now we define the instantiation of a predicate template in a symbolic state. Suppose s is a state of the symbolic state graph $Sym\_Reach(G, u)$, (p, f) is a predicate template and $x_1, x_2, \cdots, x_n$ are variables appearing in p. Let p' be the predicate obtained by replacing every occurrence of the variable x_i (for $1 \leq i \leq n$), for those x_i such that $f(x_i) \neq *$, by the expression $s.exp(y_i)\{x_i/y_i\}$ where y_i is the variable $f(x_i)$. Note that the variables x_i for which $f(x_i) = *$ are not replaced. We define $(p, f)[s]$ to be p' as given above.

For the system given in figure 1, it is easy to see that for the state s_1, $(p_1, a : x,\ y : y)[s_1]$ is $a + 1 \leq y$. Note that for those templates whose renaming function maps all the variables to $*$, the instantiation of them in any two symbolic states will be identical and thus they are trivially equivalent. We can just ignore such templates.

Definition 1. *Define relation $\sim_0$ as follows: For any two states s and t, $s \sim_0 t$ iff $s.lc = t.lc$, $s.val = t.val$ and for each $(p, f) \in ptemplates(s.lc)$, $(p, f)[s] \equiv (p, f)[t]$ is a valid formula.*

Theorem 1. *$\sim_0$ is a bi-simulation on the symbolic state graph $Sym\_Reach(G, u)$.*

It is possible that two symbolic states s, t correspond to the same actual state but $(s, t) \notin \sim_0$. To overcome this problem, we consider the relation $\sim_{equal} = \{(s, t)\ :\ s, t$ are symbolic states and $act\_state(s) = act\_state(t)\}$. Clearly, $\sim_{equal}$ is a bi-simulation on $Sym\_Reach(G, u)$. Given symbolic states s, t checking if $(s, t) \in \sim_{equal}$ is simple. We simply compute $act\_state(s)$ and $act\_state(t)$ and check if they are equal. Now, we consider the relation $\sim_0\ \cup \sim_{equal}$. It is well known that the union of bi-simulation relations is also a bi-simulation. ¿From this, we see that $\sim_0\ \cup \sim_{equal}$ is a bi-simulation on $Sym\_Reach(G, u)$. The reduction of $Sym\_Reach(G, u)$ with respect to the bi-simulation $\sim_0 \cup \sim_{equal}$ has at most as many states as the number of states in $Reach(G, u)$.

Checking if $s \sim_0 t$ requires checking equivalence of predicates $(p, f)[s]$ and $(p, f)[t]$ for each template $(p, f) \in ptemplates(s.lc)$. Section 7.1 in the appendix also introduces a bi-simulation $\sim_0'$ which is obtained by constraining the universal quantifiers over variables in the above predicate equivalence. These constraints on the quantifiers are generated by performing a static analysis of the TD. Subsection 7.1 in the appendix discusses how this equivalence check can be done efficiently for the integer domains. It also shows how the regular reachability graph. $Reach(G, u)$, instead of $Sym\_Reach(G, u)$, can be used to define a bi-simulations $\approx_0$, $\approx_0'$ and use them for model checking.

4 Bi-simulation Relations with respect to the Paths in the TD

In section 3, we defined the bi-simulation relation $\sim_0$. Two symbolic states s, t are related by this relation, if $s.lc = t.lc$ and for every node r and for every $p \in guards(r) \cup AP$ and for every path

π from $s.lc$ to r, the two predicates $(p, depends_\pi)[s]$ and $(p, depends_\pi)[t]$ are equivalent (note that $(p, depends_\pi)$ is a template in $ptemplates(s.lc)$). If none of the guards of transitions entering r is satisfiable then we don't need to require the equivalence of the above two predicates since r is never reached. We define a bi-simulation relation $\sim_1$ in which we relax the condition equivalence condition. Suppose e is a transition entering node r, then in the definition of $\sim_1$, we require the equivalence of $(p, depends_\pi)[s]$ and $(p, depends_\pi)[t]$ only for those cases when the transition e is enabled with respect to both s and t. Such a requirement will be made with respect to every transition entering r. This notion of relaxing the requirement can be generalized to paths of length k entering node r leading to bi-simulation relations $\sim_k$ for each $k > 0$. We describe this below.

First, we need the following definitions. Let π be a path in G and p be any predicate. We define the weakest precondition of p with respect to π, denoted by $WP(\pi, p)$, inductively on the length of π as follows. If π is of length zero, i.e., π is an empty path then $WP(\pi, p) = p$. If π is a single transition given by (r, C, A, r'), where A is the set of assignments $x_1 := \rho_1, ..., x_k := \rho_k$, then $WP(\pi, p)$ is the predicate $p\{\rho_1/x_1, ..., \rho_k/x_k\}$. If π has more than one transition and consists of the path π' followed by the single transition e then $WP(\pi, p) = WP(\pi', WP(e, p))$. The following lemma is proved by a simple induction on the length of π. (It is to be noted that our definition of the weakest precondition is slightly different from the traditional one; for example, the traditional weakest precondition is $C \supset WP(\pi, p)$ for the case when π is a single transition with guard C).

Lemma 2 *If path π of G is feasible from state s, and t is the state obtained by executing π from s, then t satisfies p iff s satisfies $WP(\pi, p)$.* $\Box$

Let $\pi = e_0, e_1, ..., e_{k-1}$ be a path in G where, for $0 \leq i < k$, C_i is the guard of the transition e_i. For each i, $0 < i \leq k$, let $\pi(i)$ denote the prefix of π consisting of the first i transitions, i.e., the prefix up to e_{i-1}. Define $Cond(\pi)$ to be the predicate $C_0 \wedge \bigwedge_{0<i<k} WP(\pi(i), C_i)$. The following lemma is proved using the property of the weakest preconditions given by the previous lemma.

Lemma 3 *A path π of G is feasible from state s iff s satisfies the predicate $Cond(\pi)$.* $\Box$

Let $k > 0$ be an integer. Now for each node q of G, we define a set called $extended\_templates(q, k)$ as follows. The set $extended\_templates(q, k)$ is defined to be the set of all triples of the form $(Cond(\pi''), WP(\pi'', p), depends_{\pi'})$ where π', π'' are paths such that $\pi'\pi''$ (i.e., π' followed by π'') is a path from q to some node r, the length of π'' is k and $p \in guards(r) \cup AP$. Consider the TD given in figure 2. By taking π' to be the empty path and π'' to be the single edge from q_0 to q_1, it is easy to see that the triple $(x_2 = 0, x_1 \geq 20, f_{id})$ is in $extended\_templates(q_0, 1)$. Similarly the tuple $(x_1 = 0, x_2 \geq 20, f_{id})$ is also in this set. The only other significant tuples in the above set are $(true, x_1 = 0, f_{id})$ and $(true, x_2 = 0, f_{id})$. These tuples are obtained by respectively using the paths consisting of the self loop that increments x_1 and the edge to q_2, and a similar path containing the self loop that increments x_2 and the edge to q_1.

Now we define the binary relation $\sim_k$ on the states of $Sym\_Reach(G, u)$ as follows. $\sim_k$ is the set of all pairs (s, t) where s, t are states of $Sym\_Reach(G, u)$ such that (a) $s.lc = t.lc$ and $s.val = t.val$, (b) $Finite\_Traces_k(Sym\_Reach(G, u), s) =$
$Finite\_Traces_k(Sym\_Reach(G, u), t)$ and (c) for every $(p_1, p_2, f) \in extended\_templates(s.lc, k)$ the formula $((p_1, f)[s] \wedge (p_1, f)[t]) \supset ((p_2, f)[s] \equiv (p_2, f)[t])$ is a valid formula. Observe that in the template (p_1, p_2, f), the predicate p_1 corresponds to a path π'' of length k in G, p_2 corresponds to a predicate p which is a member of AP or is a guard of some transition e from the last node in π''; roughly speaking condition (c) asserts an inductive hypothesis stating that if it is possible to reach

from s, t states s', t' (respectively) so that that π'' is feasible from both s', t' and if state s'', t'' are reached by executing π'' from s', t' (respectively) then p has the same truth value in both s'' and t''. Thus (c) together with (b) ensure inductively that the same set of infinite traces are possible from both s and t.

Theorem 2. *For each $k > 0$, $\sim_k$ is a bi-simulation relation.* $\Box$

The bi-simulation relation $\sim_k$ is defined using paths of length k. We can consider the relation $\sim_0$ defined in section 3 as $\sim_k$ for $k = 0$; in this case, we consider an empty path as a path of length zero. The following theorem states that $\sim_k$ is contained in $\sim_{k+1}$ for each $k \geq 0$ and that for for every k there exists a TD in which this containment is strict. The proof of the theorem is given in the appendix.

Theorem 3. *(1) For every $k \geq 0$, every TD G, and for every set AP of atomic predicates, $\sim_k \subseteq \sim_{k+1}$. (2) For every $k \geq 0$, there exists a TD G and a set of atomic predicates AP for which the above containment is strict, i.e. $\sim_k \subset \sim_{k+1}$.*

The set $extended\_templates(q, k)$ for any node q and for any $k > 0$ can be computed using a fix point computation just like the computation of $ptemplates(q)$. An efficient way to check condition (b) in the definition of $\sim_k$, i.e., to check $Finite\_Traces_k(Reach(G, u), s) = Finite\_Traces_k(Reach(G, u), s)$, is as follows. First check that s and t satisfy the same predicates from AP. For every transition e enabled from s, check that e is also enabled from t and vice versa; further, if s', t' are the states obtained by executing e from s, t respectively, then inductively check that $Finite\_Traces_{k-1}(Reach(G, u), s') = Finite\_Traces_{k-1}(Reach(G, u), t')$. This method works because $Reach(G, u)$ is a deterministic structure.

It is to be noted that for any $i < j$, checking if $(s, t) \in \sim_j$ is going to be more expensive than checking if $(s, t) \in \sim_i$. This is because checking condition (c) in the definition of $\sim_i$ is less expensive since it requires checking equivalence of fewer formulas since $extended\_templates(q, i)$ is a smaller set than $extended\_templates(q, i)$. Similarly, checking condition (b) is less expensive for $\sim_i$ since the number of traces of length i less than the number of traces of length j. However, the bi-simulation reduction with respect to $\sim_j$ will be smaller. Thus there is a trade off. We believe that, in general, it is practical to use $\sim_i$ for small values of i such as $i = 0, 1$, etc.

As in the section 3, for each $i > 0$, we can use the bi-simulation $\sim_i \cup \sim_{equal}$ on the graph $Sym\_Reach(G, u)$ to get a smaller bi-simulation reduction. Section 7.1 in the appendix defines bi-simulations $\sim'_k$ analogous to $\sim'_0$ by constraining the domain of universal quantifiers in the predicate equivalences. Analogous to $\approx_0$ and $\approx'_0$, it also defines bi-simulations $\approx_k$ and $\approx'_k$ for integer domains over $Reach(G, u)$ for each $k > 0$. The appendix discusses and gives an example TD, shown in figure 2, for which the bi-simulation $\approx_1$ gives more reduction than $\approx_0$.

5 Experimental Results

We implemented the method given in the paper for checking invariance properties. This implementation uses the reduction with respect to the bi-simulation $\sim_0$.

Our implementation takes a concurrent program given as a transition system T. The syntax of the input language is similar to that of SMC [17]. The input variables can be either binary variables or integer variables. The implementation has two parts. The first part reads the description of T and converts it in to a TD G by executing the finite state part of it. All the parts in the conditions and actions that refer to integer variables in the transitions of T are transferred to guards and actions in the transitions of G. The second part of the implementation constructs the bi-simulation reduction of G given in section 3 and checks the invariance property on the fly.

We tested our implementation for a variety of examples. Our implementation terminated in all cases excepting for the example of an unbounded queue implementation. In those cases where it terminated, it gave the correct answer. One of our examples is the sliding window protocol with bounded channel. It is an infinite state system due to the sequence numbers. This protocol has a sender and receiver process communicating over a channel. The sender transmits messages tagged with sequence numbers and the receiver sends acknowledgments (also tagged with sequence numbers) after receiving a message. We checked the safety property that every message value received by the receiver was earlier transmitted by the sender. We tested this protocol with a bounded channel under different assumptions. The results are given in Table 2. The table shows the window sizes, the time in seconds, the number of states and edges in the reduced graph and the environment for which the protocol is supposed to work (here "duplicates" means the channel can duplicate messages, "lost" means the channel can lose messages).

Problem Instance	Property	t in sec	# of states
Ticket algorithm	$\forall\Box(\neg(pc_1 = C_1 \wedge pc_2 = C_2))$	0	9
ProducerConsumer size of buffer	Property	t in sec	# of states
30	$\forall\Box(0 \leq p_1 + p_2 - (c_1 + c_2) \leq s)$	0.01	31
100	$\forall\Box(0 \leq p_1 + p_2 - (c_1 + c_2) \leq s)$	0.09	101
Circular Queue size of queue	Property	t in sec	# of states
10	$\forall\Box(h \leq s \wedge t \leq s)$	0.24	121
	$\forall\Box(t \geq h \rightarrow p - c = t - h)$	0.12	121
	$\forall\Box(t \leq h \rightarrow p - c = s - (h - t) + 1)$	0.12	121
	$\forall\Box(0 \leq p - c \leq s)$	0.15	121
30	$\forall\Box(h \leq s \wedge t \leq s)$	16.4	961
	$\forall\Box(t \geq h \rightarrow p - c = t - h)$	2.8	961
	$\forall\Box(t \leq h \rightarrow p - c = s - (h - t) + 1)$	2.7	961
	$\forall\Box(0 \leq p - c \leq s)$	3.2	961

Table 1. Summary of the tests.

We also tested with three other examples taken from [6]. These are the Ticket algorithm for mutual exclusion, Producer-Consumer algorithm and Circular queue implementation. The detailed descriptions of these examples can be found in [6]. All these examples use integer variables as well as binary variables. The ticket algorithm is an algorithm similar to the bakery algorithm. We checked the mutual exclusion property for this algorithm. The Producer-Consumer algorithm consists of two producer and two consumer processes. The producer processes generate messages and place them in a bounded buffer which are retrieved by the consumer processes. In this algorithm, the actual content of the buffer is not modeled. The algorithm uses some binary variables, four integer variables p_1, p_2, c_1, c_2 and a parameter s denoting the size of the buffer. Here p_1, p_2 denote the total number of messages generated by each of the producers respectively; similarly, c_1, c_2 denote the number of messages consumed by the consumers. The circular queue example has the bounded variables h, t which are indexes into a finite array, positive integer variables p, c and a parameter s that denotes the size of the buffer. Here p, c respectively denote the number of messages that are enqueued and dequeued since the beginning. In all these examples, our method terminated and the results are shown in table 1. Time in the table is the time used to get the reduced symbolic state graph. Checking the property does not take that much time.

We have also implemented the method using the relation $\sim_1$. We run it on examples such as the one given in figure 2 and get a smaller bi-simulation reduction. More realistic examples need to be further examined.

Sender Window	Receiver Window	t[s]	# of states	# of edges	Environment
1	1	0.016	47	164	duplicate, lost
1	2	0.203	447	1076	duplicate
1	2	0.296	509	1731	duplicate, lost
2	1	0.860	1167	3832	duplicate
2	2	11.515	4555	11272	duplicate

Table 2. Experiment on sliding window protocol.

6 Conclusion and Related Work

In this paper we gave methods for model checking of concurrent programs modeled by simple Transition Diagrams. We have given a chain of non-decreasing bi-simulation relations, $\sim_i$ for each $i \geq 0$, on these TDs. They are defined by using predicate templates and extended predicate templates that can be computed by performing a static analysis of the TD. Recall that $\sim_i$ is defined by using templates with respect paths of length i in the TD. Each of these bi-simulations require that the two related states be indistinguishable with respect to the corresponding templates. All these relations can be used to compute a reduced graph for model checking. We have also given how some of these bi-simulation conditions can be checked efficiently for certain types of programs. We have also shown how they can be used to show the decidability of the reachability problem of some simple hybrid automata when the time domain is discrete. We have also presented variants of the above approaches. We have implemented the model checking method using the bi-simulations $\sim_0, \sim_1$. We have given experimental results showing their effectiveness. To the best our knowledge, this is the first time such a comprehensive analysis of simple TDs has been done and the various types of bi-simulations have been defined and used.

Further work is needed on relaxing some of the assumptions such as the commutativity and also on weakening the equivalence condition. Further investigation is also needed on identifying classes of programs to which this approach can be applied.

Related Work

The work that is most closely related is that given in [12]. In this work the authors present a method that syntactically transforms a program with variables ranging over infinite domain to boolean programs. Their approach involves two steps. In the first step, they perform a fix point computation to generate a set of predicates. In the second step, a boolean program is constructed using binary variables to represent the generated predicates and this program is checked using any existing model checker. The first step in their approach may not terminate. In our case, we use simple methods to compute a set of predicate templates with each node in the TD and this method always terminates. Also the predicate templates computed in our are different from those generated by [12]. Our second step involves constructing the reduced graph and this step may not terminate sometimes. Our methods are better suited for on-the-fly model checking; i.e., we can terminate with an error message when the first error state is encountered during the construction of the reduced graph. In case of the method of [12], the first step needs to be completed before actual model checking can begin, and the first step may not terminate sometimes. There are examples for which our system terminates but the method of [12] may not terminate. Consider the following TD with one node and with the following two transitions from that node back to itself: $x \leq 0 -> x := x + 2$ and $x \geq 1 -> x := x - 1$. The initial value of x is 0. Let the correctness property be $AG(x < 3)$. It can be shown that the method of [12] does not terminate for this example while ours does.

There have also been techniques that construct the bi-simulation reduction in an on-the-fly manner in [10]. The method given in [10] assumes symbolic representation of groups of states and requires efficient computation of certain operations on such representations. Our work is also different from the predicate abstraction methods used in [16, 9, 7] and also in [2] These works use predicates to abstract the concrete model, to get the abstract model and then perform model checking on it. This abstraction ensures that there is a simulation relation from the concrete model to the abstract model. They use ∀CTL for model checking. If the abstract model does not satisfy the correctness then they will have to refine the abstraction and try this process again. Since we use bi-simulation based approach the program satisfies the correctness formula iff the reduced graph satisfies it. So we do not need any further iterations for refinement. The SLAM project [2, 3] has also applied predicate abstraction and static analysis for verification of software. They have successfully used these methods for verifying device driver routines.

It should be noted that the commutativity assumption of TDs is different from the commutativity assumption in partial reductions [13, 8]; our assumption requires commutativity of functions employed to update the variables, while in partial-order based methods commutativity of transitions is used. Note that the commutativity definition of transitions, used in partial order reductions, takes into consideration both the guard and action parts of the transitions. It should not be difficult to come up with examples where the functions in the actions are commutative but the transitions themselves are not. Hence in some sense, ours is more general.

It is to be noted that our method based on the bi-simulation $\sim_0$ is itself a generalized method for [4] where a location based bisimulation is used. Our work is different from other works for handling large/infinite state systems such as the one in [5] where symbolic representation of periodic sets is employed.

The work given in [6] considers verification of systems with integer variables. Their method is based on computing invariants and employs approximation techniques based on widening operators. Our method is based on bi-simulation and can be employed for other domains also apart from systems with integer variables.

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

2. T. Ball and S. K. Rajmani. The slam toolkit. In *CAV2001*, 2001.
3. T. Ball and S. K. Rajmani. The slam project: Debugging system software via static analysis. In *ACM Symposium on Principles of Progarmming Languages*, 2002.
4. G. Behrmann, E. Bouyer, Patricia Fleury, and K. G. Larsen. Static guard analysis in timed automata verification. In *TACAS*, pages 254–270, 2003.
5. B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In D. L. Dill, editor, *CAV94, Stanford, California, USA, Proceedings*, volume 818 of *Lecture Notes in Computer Science*, pages 55–67. Springer, 1994.
6. T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: symbolic representations, approximations, and experimental results. *ACM Trans. Program. Lang. Syst.*, 21(4):747–789, 1999.
7. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counter-example guided abstraction refinement. In *CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, volume 1855 of *Lecture Notes in Computer Science*. Springer, 2000.
8. P. Godefroid. *Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem*, volume 1032. Springer-Verlag Inc., New York, NY, USA, 1996.
9. H.Saidi and N. Shankar. Abstract and model check while you prove. In N. Halbwachs and D. Peled, editors, *CAV99, Trento, Italy,Proceedings*, volume 1633 of *Lecture Notes in Computer Science*, pages 443–454. Springer, 1999.
10. D. Lee and M. Yannakakis. Online minimization of transition systems (extended abstract). In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 264–274. ACM Press, 1992.
11. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, Berlin, Jan. 1992.
12. K. S. Namjoshi and R. P. Kurshan. Syntactic program transformations for automatic abstraction. In *CAV2000*, pages 435–449, 2000.
13. D. Peled. All from one, one for all, on model-checking using representatives. In C. Courcoubetis, editor, *CAV 93, Elounda, Greece, Proceedings*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 1993.
14. C. C. R. Alur and et al. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
15. T. H. R. Alur, C. Courcubetis and P. H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229, 1993.
16. S. Graf and H. Saidi. Construction of abstract state graphs with PVS. In O. Grumberg, editor, *CAV97*, volume 1254, pages 72–83. Springer Verlag, 1997.
17. A. P. Sistla, V. Gyuris, and E. A. Emerson. Smc: A symmetry based model checker for safety and liveness properties. *ACM Transactions on Software Engineering Methodologies*, 9(2):133–166, 2000.
18. A. P. T. A. Henzinger, P. W. Kopke and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.
19. J. S. X. Nicollin, A. Olivero and S. Yovine. An approach to description and analysis of hybrid systems. In *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 149–178, 1993.
20. J. S. Y. Kesten, A. Pnueli and S. Yovine. Integration graphs: A class of decidable hybrid systems. In *Hybrid systems*, volume 736 of *Lecture Notes in Computer Science*, pages 179–208. Springer-Verlag, 1993.

7 Appendix

7.1 Extensions and Special cases

Using static analysis to constrain the quantifiers' range In subsection 3.3 we have defined a bi-simulation relation $\sim_0$ on the states of $Sym\_Reach(G,u)$. Under this definition, $s \sim_0 t$ if $s.lc = t.lc$, $s.val = t.val$ and for each template $(p,f) \in ptemplates(s.lc)$, $(p,f)[s] \equiv (p,f)[t]$ is valid. Note that $var(p)$ is also the set of variables that appear free in $(p,f)[s]$ and $(p,f)[t]$. It should also be noted that there is an implicit universal quantification on the variables in $var(p)$ in the equivalence $(p,f)[s] \equiv (p,f)[t]$ when we assert its validity. This universal quantification can be restricted to range over a smaller set of values in some cases. By this change, we get another relation which is also a bi-simulation that is larger than $\sim_0$.

Consider any state s in $Sym\_Reach(G,u)$. Let (p,f) be any predicate template in $ptemplates(s.lc)$ such that $p \in guards(G) \cup AP$. Let $\{x_1,...,x_n\}$ be all the variables in $var(p)$ and $Q_p \subseteq Q$ be as defined below: if $p \in AP$ then $Q_p = Q$, otherwise Q_p is the set of all nodes q in G such that $p \in guards(q)$. Let G' be the TD obtained from G by replacing all guards in its transitions by the constant predicate $True$. Now let $T_{p,s}$ be the set of n-tuples $(c_1,...,c_n)$ respectively denoting the values of variables $x_1,...,x_n$ in some state (q,h), where $q \in Q_p$, that is reachable from the state $(s.lc, s.val)$ in the $Reach(G',u)$. Essentially, $T_{p,s}$ is the set of all tuples of values for the variables $x_1,...,x_n$ when the control is at some node in Q_p when the TD G' is executed starting from the state $(s.lc, s.val)$. Let $g_{p,s}$ be a formula with free variables $x_1,...,x_n$ that defines any super set of $T_{p,s}$, i.e., every tuple in $T_{p,s}$ satisfies the constraint given by $g_{p,s}$. Essentially, $g_{p,s}$ defines a constraint on the values of the variables when the control is at some node in Q_p. Let $\sim'_0$ be the set of (s,t) such that s,t are symbolic states, $s.lc = t.lc$, $s.val = t.val$, for all $(p,f) \in ptemplates(s.lc)$ the formula $g_{p,s} \supset ((p,f)[s] \equiv (p,f)[t])$ is valid. Thus we see that the quantifiers over $x_1,...,x_n$ are restricted to the tuples that satisfy $g_{p,s}$, i.e., the tuples in $T_{p,s}$. Observe that $T_{p,s} = T_{p,t}$; thus it does not matter whether we use s or t for obtaining these sets and the formulas $g_{p,s}$. It should be easy to see that $\sim'_0 \supseteq \sim_0$.

The following theorem can be proved on the same lines as theorem 1.

Theorem 4. *$\sim'_0$ is a bi-simulation on symbolic state graph $Sym\_Reach(G,u)$.* $\square$

We can also define relations $\sim'_i$, for each $i > 0$, by performing static analysis and by constraining the domain of the universal quantifiers over the free variables in the conditional predicate equivalences of $\sim_i$ for each $i > 0$. Formally, $\sim'_i$ is the set of symbolic state pairs (s,t) such that $s.lc = t.lc$ and the following is satisfied: for each triple $(p_1, p_2, f) \in extended\_templates(s.lc, i)$, we require that $(g_{p,s} \wedge (p_1,f)[s] \wedge (p_1,f)[t]) \supset ((p_2,f)[s] \equiv (p_2,f)[t])$ be a valid formula where $g_{p,s}$ is as defined above. It is also not to difficult to see, using theorem 2, that each $\sim'_i$ (for $i > 0$) is a bi-simulation.

In the above definitions, we see that the formulas $g_{p,s}$ depend on p, $s.lc$ and $s.val$. Note that the set of possible values of $s.val$ can be determined from static analysis of G. Essentially, for any variable x, $s.val(x)$ is a constant that is assigned to x in the action part of some transition. Since G' has only the guards $True$ in all transitions, it is not difficult to see that the set of tuples $T_{p,s}$ or the formulas $g_{p,s}$ can be computed by performing a static analysis of G. The number of values for $s.val$ can be exponential. Many times we can get a constraining formula $g_{p,s}$ which is even independent of $s.val$.

Transition Diagrams over integer domains The use of symbolic graphs can be avoided for certain cases. Consider transition diagrams where the variables are integer variables and the functions updating them are increments by constants; that is, for any constant c, let $f_c(x)$ be the function $x + c$. It is trivial to see that for any two constants c, d the functions f_c and f_d are commutative.

Without constructing the symbolic state graph, we directly define a bi-simulation on the structure $Reach(G, u)$. Recall that each state of $Reach(G, u)$ is of the form (q, h) where $q \in Q$ and h assigns values to variables in X. Let $s = (q, h)$ be a state in $Reach(G, u)$. For a predicate template (p, f), define $(p, f)[s]$ to be the predicate obtained by replacing each occurrence of a variable x in p by $x + h(f(x))$ if $f(x)$ is a variable. (Note that we don't replace x if $f(x) = *$). Let $\approx_0$ be the set of pairs of states (s, t), where $s = (q, h)$ and $t = (q, h')$ for some $q \in Q$, such that for each $(p, f) \in ptemplates(q)$, $(p, f)[s] \equiv (p, f)[t]$ is valid. (This construction can be employed for a general case where $+$ is any commutative and associative binary operator over the domain of the variables.)

Theorem 5. *$\approx_0$ is a bi-simulation on $Reach(G, u)$.* $\square$

Now we discuss, how to check if two states s, t are related by $\approx_0$ efficiently for integer domain TDs where each predicate p (appearing in the guards or in the temporal formula) is linear, i.e. is of the form $\Sigma_{1 \leq i \leq n} a_i x_i > d'$ where $\{x_i : 1 \leq i \leq n\}$ is a subset of X and $a_1, ..., a_n$ are integer constants. Consider such a TD. Let $s = (q, h)$ and $t = (q, h')$ be any two states. Now consider any template (p, f) in $ptemplates(s.lc)$ where p is linear, i.e., p is $\Sigma_{1 \leq i \leq n} a_i x_i > d'$. It is not difficult to see that $(p, f)[s] \equiv (p, f)[t]$ is valid if $\Sigma_{1 \leq i \leq n} a_i h(y_i) = \Sigma_{1 \leq i \leq n} a_i h'(y_i)$ where the variables y_i, for $1 \leq i \leq m$, are given as follows: y_i is same as x_i if $f(x_i) = *$; otherwise y_i is $f(x_i)$. The later condition can be checked efficiently. It is also not difficult to see that checking equivalence can also be done efficiently when p is of the form $(\Sigma_{1 \leq i \leq n} a_i x_i) \; mod \; c = c'$ where c, c' are appropriate constants.

A subclass of integer domain TDs are those in which the variables range over natural numbers; this occurs if all the constants in the transitions are positive and the initial state u assigns only positive values to the variables; further each predicate p is a linear predicate and the coefficients of variables in the predicate are all positive. Let $s = (q, h)$ and $t = (q, h')$ be any two states. Now consider any template (p, f) in $ptemplates(s.lc)$ where p is given by $\Sigma_{1 \leq i \leq n} a_i x_i > d'$. It is not difficult to see that $(p, f)[s] \equiv (p, f)[t]$ is valid if either the two values $\Sigma_{1 \leq i \leq n} a_i h(y_i)$ and $\Sigma_{1 \leq i \leq n} a_i h'(y_i)$ are equal, or both of them are greater than d'; here the variables y_i are defined as before.

For each $p \in guards(G) \cup AP$, let Q_p be the set of nodes in G defined earlier. Let $x_1, ..., x_n$ be the variables that appear in p. For any state $s = (q, h)$ and $p \in guards(G) \cup AP$, let $T_{s,p}$ be the set of tuples $(c_1, ..., c_n)$ such that there exists a state of the form $s' = (q', h')$, where $q' \in Q_p$, that is reachable in G' from the state $(q, \mathbf{0})$ where $\mathbf{0}$ is the function that assigns value zero to all the variables and and $c_1, c_2, ..., c_n$ are the values of variables $x_1, ..., x_n$ in s', i.e., the values given by h'. (It is to be noted that the definition of $T_{s,p}$ is slightly different from the one in the previous sub-section; its value does not depend on h either). Using formulas that define super sets of $T_{s,p}$, as given in the previous subsection, we can define a bi-simulation $\approx'_0$ that is bigger than $\approx_0$. Such super sets can be defined using static analysis.

For each $i > 0$, we define the bi-simulations $\approx_i$ and $\approx'_i$ just like $\sim_i$ and $\sim'_i$ excepting that these are defined on the structure $Reach(G, u)$ (these are defined on the same lines as $\approx_0$ and $\approx'_0$).

The TD given in figure 2 is an example that shows that the bi-simulation $\approx_1$ gives more reduction than $\approx_0$. As indicated earlier the triples $(x_2 = 0, x_1 \geq 20, f_i d)$ and $(x_1 = 0, x_2 \geq 20, f_i d)$ are in

$extended\_templates(q_0, 1)$. In addition, the triples $(true, x_1 = 0, f_id)$ and $(true, x_2 = 0, f_id)$ are also in the above set. If $s = (q_0, h)$ and $t = (q_0, h')$ are two states such that $h(x_1) = 0, h(x_2) = 15$ and $h'(x_1) = 1, h'(x_2) = 25$, then it is not difficult to see that $s \approx_1 t$. However s, t are not related by $\approx_0$. For any two states $s = (q_0, h)$ and $t = (q_0, h')$ in $Reach(G, \mathbf{0})$, $s \approx_0 t$ if, for each $j = 1, 2$, $h(x_j) = h'(x_j)$ or both $h(x_j), h'(x_j)$ are greater than or equal 20. Thus the set of states of the above type form into 21^2 classes with respect to $\approx_0$. On the other hand, they are related by $\approx_1$ if their instantiations with respect to the above triples in $extended\_templates(q_0, 1)$ are equivalent. By a detailed combinatorial analysis, it can be shown that these set of states are divided into exactly 42 classes with respect to the bi-simulation $\approx_1$. In a more general case, if the constant in the transitions entering node q_3 is c then the set of states of the form (q_0, h) will be divided in to $(c+1)^2$ and $2(c+1)$ classes with respect to the bi-simulations $\approx_0, \approx_1$ respectively. In this general case the number of classes of states of the form (q_1, h) or (q_2, h) or (q_3, h) is $3(c+1)$ and 3 with respect to $\approx_0$ and $\approx_1$ respectively. Thus in the general case, the total number of states in the reductions with respect to $\approx_0, \approx_1$ are given by $(c+1)^2 + 3(c+1)$ and $3(c+1) + 3$ respectively. Thus $\approx_1$ gives a smaller bi-simulation reduction. In general, we can generalize the example by introducing n arbitrary variables where $\approx_1$ has much more reduction than $\approx_0$.

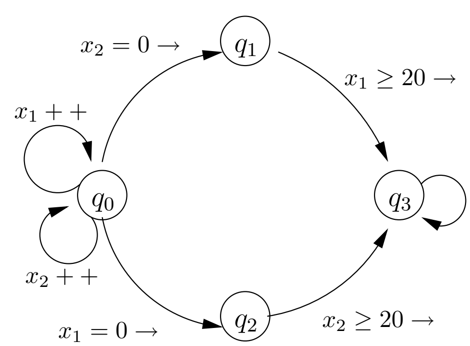


Fig. 2. Example of a TD for which $\approx_{i-1} \subset \approx_i$

7.2 Applications and Discussion

Applications In this subsection, we discuss how the bi-simulations defined in subsection 7.1 can be used to analyze different classes of hybrid automata that were proposed in [15, 19] for specification and verification of properties hybrid systems. Essentially, a hybrid automata is a TD in which all the variables range over real numbers and the values of these variables increase automatically with time at various rates. The rate at which a particular variable increases over time may be different depending on the node of the TD. A particular class of hybrid systems are the timed automata [1]. In these timed automata the values of all the variables increase at unit rate in all the nodes. The reachability problem for hybrid automata is to decide if there exists an execution that takes the system from a given initial node, when all the variables are started with value zero, to a given node in the automaton. This problem has been shown to be undecidable in general [20] and decidable in particular cases [14, 18]. The undecidability of the reachability problem was shown to hold even when time is restricted to be discrete, i.e., the set of natural numbers. We consider hybrid automata when time is restricted to be discrete, and show our bi-simulations can be used to show the decidability of the reachability problem for these cases.

We consider hybrid automata over discrete time domain where the time ranges over natural numbers. Usually hybrid automata have invariants associated with each node; such invariants are used to force the automaton to leave the state and hence satisfy a liveness condition; we ignore such invariants since they can be transferred to the guards of the out going transitions of the node without effecting the decidability of the reachability problem. Consider such an automaton $\mathcal{A}$. We capture such an automaton by TD G as follows. G is going to be identical to $\mathcal{A}$ excepting that we add additional transitions. In each nod q of G, we add an edge which is a self loop and the transition labeling this edge has an action part that increments all the variables by their respective rates and the guard of such a transition is $true$. We call these transitions as clock transitions. The clock transitions are always enabled, and each execution of such transitions models the advancement of the clock by one unit and it increments the variables by the appropriate rates. If the rates at which the variables are advanced is non-deterministic, then we can model this by multiple self loops in a node corresponding to the different combinations of rates of the variables.

To check if a node r is reachable from the initial state, we simply need to check if the CTL formula $EF(lc = r)$ holds in the initial state. Thus, we see that the set of atomic predicates involving variables are only the ones that appear in the guards. Now we consider various cases. Assume that the rates at which the variables increase in all the nodes are all positive. Also assume that all the predicates in $guards(G)$ are of the form $\Sigma_{1 \leq i \leq n} a_i x_i \; > d$ where the coefficients a_i, for $1 \leq i \leq n$, are all positive. Since all the variables range over natural numbers, we use the reachability graph $Reach(G, u)$ and the bi-simulation $\approx_0$. From the discussion in the subsection 7.1, we see that we get a finite quotient with respect to the bi-simulation $\approx_0$.

We can also use $\approx_0'$ defined in the subsection 7.1 to get finite quotients for a wider class of TDs. Recall that two states s, t are related by $\approx_0'$, if $s.lc = t.lc$ and for every template (p, f) in $ptemplates(s.lc)$, the formula $g \supset (p, f)[s] \equiv (p, f)[t]$ is valid. In these requirements, g is a formula that constrains the ranges of free variables and is obtained by static analysis of the TD. Suppose in a TD we allow comparisons of the form $ax_1 - bx_2 > d$ where a, b are positive numbers. Suppose for any predicate p of the above form, if U_1 is the set of all variables y such that for some path π in the TD $depends_\pi(x_1) \; = y$ and similarly U_2 is the set of variables that x_2 depends on. Then if the rates of increase of all variables in U_1 are equal in every node, and similarly the rates for variables in U_2 are same, and further for every node q, if u_1, u_2 are the rates for x_1, x_2 in node q respectively, and further $au_1 - bu_2 \geq 0$, then it can be shown that $\approx_0'$ gives a finite quotient.

It also to be noted that we can use the possibly bigger bi-simulations given by $\approx_i$ for $i > 0$ to get smaller bi-simulation reductions The hybrid automaton represented by the TD given in figure 2 is such an example.

Relaxing the Commutativity Requirement So far in this paper, we required that the set of all functions applied to variables of the same type are mutually commutative. This requirement can be relaxed as follows. Recall that for any path π in the G, $depends_\pi$ is a function that specifies dependencies of values of the variables at the end π to their values at the beginning of π. Now we introduce a binary relation $depends\_on$ on variable-node pairs, i.e., pairs of the form (x, r) where $x \in X$ and r is a node in G. The pair (y, r) $depends\_on$ the pair (x, q) if there exists a path π in G from node q to r such that $depends_\pi(y) \; = x$. It is easy to see that for the TD given in Figure 3,

(x, q_2) *depends_on* (x, q_1) while (x, q_3) does not *depends_on* (x, q_2) because the only transition from q_2 to q_3 assigns a constant to x. On the other hand, if t_2 did not have this assignment for x then (x, q_2) *depends_on* (x, q_3), and in addition if we have $x := y$ as the assignment in t_1 then (x, q_3) *depends_on* (y, q_1).

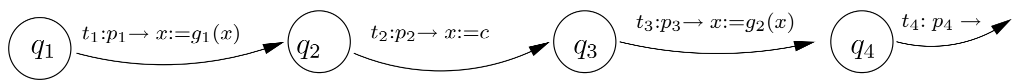


Fig. 3.

Using the relation *depends_on*, we specify the less restrictive commutative requirement. Let Π be the set of all functions that are applied to variables in the transitions of G. Let $\{\Pi_1, ..., \Pi_m\}$ be the finest partition of Π which satisfies the following condition: two functions ψ_1 and ψ_2 are in the same set Π_i if there exist pairs (y, r) and (x, q) such that (y, r) *depends_on* (x, q) and there exist transitions (q', C_1, Λ_1, q), (r, C_2, Λ_2, r') such that the assignment $x := \psi_1(x)$ is in Λ_1 and the assignment $y := \psi_2(y)$ is in Λ_2. It is easy to see that the partition can be computed efficiently from G using standard graph algorithms. Consider the example of Figure 3, the functions g_1, g_2 are in the different sets of the partition and hence they need not be commutative. On the other hand, if t_2 did not have any assignment for x then both g_1, g_2 would be in the same set of the partition and they would need to be commutative.

7.3 Proof for theorems

Proof of theorem 1: Assume that $s \sim_0 t$. For any $q \in Q$, $\forall p \in \{AP \cup guards(q)\}$, we have $(p, f_{id}) \in ptemplates(q)$. By definition of $\sim_0$, $(p, f_{id})[s] \equiv (p, f_{id})[t]$ is valid. $(p, f_{id})[s]$ and $(p, f_{id})[t]$ are obtained by replacing each variable x occurring in $(p, f_{id})[s]$ and in $(p, f_{id})[t]$ by $s.val(x)$ and $t.val(x)$ respectively. It is easy to see that the symbolic state s satisfies p iff t satisfies p. From this it follows that $L'(s) = L'(t)$ and a transition e from node $s.lc$ is enabled in s iff it is enabled in t. Now assume $(s, e, s') \in R'$ which means there is an enabled transition e in s and s' is reached after the execution of e. From above, we know that e is also enabled in t. Let t' be the symbolic state obtained by executing e from t. We show that $s' \sim_0 t'$. Obviously, $s'.lc = t'.lc$ and $s'.val = t'.val$.

Now, we show for all $(p, f') \in ptemplates(s'.lc)$, $(p, f')[s'] \equiv (p, f')[t']$ is a valid formula. Consider template $(p, f') \in ptemplates(s'.lc)$. From the definition of template, it is seen that there exists a $(p, f) \in ptemplates(s.lc)$ satisfying the following conditions for every x in $var(p)$. These conditions are divided into five cases. The first case is when $f'(x) = *$; in this case, $f(x) = *$ and the variable x in p remains unchanged when we obtain $(p, f)[s]$ and $(p, f')[s']$ from p. In the other cases $f'(x) \neq *$, i.e., $f'(x)$ is a variable. Let $f'(x)$ be the variable y. The remaining cases depend on whether there is an assignment to y in the action part of e and if so, what is assigned to it. The second case is when there is no assignment for the variable y. In this case, $f(x) = y$ and $s'.exp(y) = s.exp(y)$; hence the expressions substituted for x in p to obtain $(p, f)[s]$ and $(p, f')[s']$ are identical. The third case is when y is assigned a constant; in this case, $f(x) = *$ and from the construction of $Sym\_Reach(G, u)$, we see that $s'.exp(y) = y$ and hence the expressions substituted for x in p to obtain $(p, f)[s]$ and $(p, f')[s']$ are both x itself. The fourth case is when y is assigned a variable z; in this case, $f(x) = z$

and $s'.exp(y) = s.exp(z)\{y/z\}$; from this it should be easy to see that the expressions substituted for x in p to obtain $(p,f)[s]$ and $(p,f')[s']$ are identical.

In all above four cases, we see that the same expression is substituted for x in p to obtain both $(p,f)[s]$ and $(p,f')[s']$. The fifth and last case is when there is an assignment of the form $y := \psi(y)$ where ψ is a unary function; in this case, $f(x) = y$ and $s'.exp(y) = \psi(s.exp(y))$; since the function ψ is commutative with the functions appearing in $s.exp(y)$, it is easy to see that $s'.exp(y) = s.exp(y)\{\psi(y)/y\}$; it should be noted that the variable x in p is substituted by $s.exp(y)\{x/y\}$ and by $s'.exp(y)\{x/y\}$ respectively to obtain $(p,f)[s]$ and $(p,f')[s']$. From the above observations, we see that the expression substituted for x in p to obtain $(p,f')[s']$ is $s.exp(y)\{\psi(x)/y\}$. Let $x_1,...,x_n$ be the variables in $var(p)$ to which this last case applies. For each $i = 1,...,n$, let $f'(x_i) = y_i$ and $y_i := \psi_i(y_i)$ be the assignment to y_i in the action part of e.

From the above observations, we see that the expression substituted for x_i in p to obtain $(p,f)[s]$ is $s.exp(y_i)\{x_i/y_i\}$, while the expression substituted for x in p to obtain $(p,f')[s']$ is $s.exp(y_i)\{\psi_i(x_i)/y_i\}$. Using this fact for each $i = 1,...,n$ and the fact that in all the first four cases the same expressions are substituted for x in p to obtain both $(p,f)[s]$ and $(p,f')[s']$, it is not difficult to see that $(p,f')[s'] = (p,f)[s]\{\psi_1(x_1)/x_1,...,\psi_n(x_n)/x_n\}$. By the same argument, we see that $(p,f')[t'] = (p,f)[t]\{\psi_1(x_1)/x_1,...,\psi_n(x_n)/x_n\}$. Since, $(p,f)[s] \equiv (p,f)[t]$ is valid, its validity holds even when we replace each x_i by $\psi(x_i)$ for $i = 1,...,n$. When this replacement is done, we get that $(p,f')[s'] \equiv (p,f')[t']$ is valid. □

In Figure 1, state s_0 is bi-similar to state s_2. Actually, the right subgraph of Figure 1 is the whole reduced symbolic state graph. It only consists of 3 states while the original state graph has infinite states.

Proof of theorem 2: The theorem is proved on the same lines as theorem 1. Assume that $s \sim_k t$. From the definition, we have

$$Finite\_Traces_k(Sym\_Reach(G,u),s) = Finite\_Traces_k(Sym\_Reach(G,u),t)$$

¿From this it follows that $L'(s) = L'(t)$ and a transition τ from node $s.lc$ is enabled in s iff it is enabled in t. Now assume $(s,\tau,s') \in R'$. Let t' be the symbolic state such that $(t,\tau,t') \in R'$. Obviously, $s'.lc = t'.lc$ and $s'.val = t'.val$.

Now, using the same argument as given in the proof of theorem 1, it is easily seen that for every $(p_1,p_2,f) \in extended\_templates(s'.lc,k)$, the formula $((p_1,f)[s'] \wedge (p_1,f)[t']) \supset ((p_2,f)[s'] \equiv (p_2,f)[t'])$ is a valid formula. Now we show that

$$Finite\_Traces_k(Sym\_Reach(G,u),s') = Finite\_Traces_k(Sym\_Reach(G,u),t')$$

Let $\phi_0,e_0,...,\phi_{k-1},e_{k-1}$ be a trace of length k in $Finite\_Traces_k(Sym\_Reach(G,u),s')$. Let $\psi = L'(s)$. Clearly $\psi,\tau,\phi_0,e_0,...,\phi_{k-2},e_{k-2}$ is in $Finite\_Traces_k(Sym\_Reach(G,u),s)$ and hence is also in $Finite\_Traces_k(Sym\_Reach(G,u),t)$. As a consequence $\phi_0,e_0,...,\phi_{k-2},e_{k-2}$ is a finite trace of length $k-1$ from t'; this is due the fact that $Sym\_Reach(G,u)$ is deterministic. Now, we see that $\pi = \tau,e_0,...,e_{k-2}$ is a path of length k from node $s.lc$ in G to a node r. Clearly e_{k-1} is a transition from r. Let C_{k-1} be the guard of this transition. It is easy to see that the triple $(Cond(\pi), WP(\pi,C_{k-1}), f_{id})$ is in $extended\_templates(s.lc,k)$. It is also easy to see that the predicates $Cond(\pi), f_{id})[s]$ and $Cond(\pi), f_{id})[t]$ are both true when the values of the variables are given by $s.val$ (note that $s.val = t.val$). For any predicate $p \in AP$, $p \in \phi_{k-1}$ iff $WP(\pi,p), f_{id})[s]$ is true when the value of the variables is given by $s.val$. Since $s \sim_k t$, it is the case that $((Cond(\pi), f_{id})[s] \wedge (Cond(\pi), f_{id})[t]) \supset \ WP(\pi, p)[s] \equiv WP(\pi, p)[t]$ is a validity. Substituting the values of the variables given by $s.val$ in to the above valid formula and using earlier observations, we that the state t'' reached by executing π from the state t satisfies p iff $p \in \phi_{k-1}$. By a similar argument it is shown that t'' satisfies C_{k-1} and hence the transition e_{k-1} is enabled in t''. Putting all these together we see that $\phi_0, e_0, ..., \phi_{k-1}, e_{k-1}$ is also a trace from t', i.e. it is in $Finite\_Traces_k(Sym\_Reach(G, u), t')$. By a symmetric argument, we see that every trace in $Finite\_Traces_k(Sym\_Reach(G, u), t')$ is also in $Finite\_Traces_k(Sym\_Reach(G, u), s')$. This shows that $s' \sim_k t'$. $\Box$

Proof of theorem 3: We prove (1) of the theorem first. Let G be any TD, AP be any set of atomic predicates and let k be any positive integer. We show below that $\sim_k \subseteq \sim_{k+1}$. Assume s, t are two states in $Sym\_Reach(G, u)$ such that $s \sim_k t$. Clearly $s.lc = t.lc$ and $s.val = t.val$. Now we show that $s \sim_{k+1} t$. Since $\sim_k$ is a bi-simulation, it is not difficult to see that $Finite\_Traces_l(Sym\_Reach(G, u), s) = Finite\_Traces_l(Sym\_Reach(G, u), t)$ for all $l \geq 0$; hence this is true for $l = k + 1$. Now consider any triple $(p_1, p_2, f) \in extended\_templates(s.lc, k + 1)$. We want to show that the following formula is a validity.
(A): $((p_1, f)[s] \wedge (p_1, f)[t]) \supset (p_2, f)[s] \equiv (p_2, f)[t]$.
¿From the definition of $extended\_templates(s.lc, k + 1)$, we see that there exists a path in G of the form $\pi'\pi''$ from $s.lc$ to a node r in G such that π'' has $k + 1$ transitions and there exists a predicate $p_3 \in guards(r) \cup AP$ such that $p_1 = Cond(\pi'')$, $p_2 = WP(\pi'', p_3)$ and $f = depends_{\pi'}$.

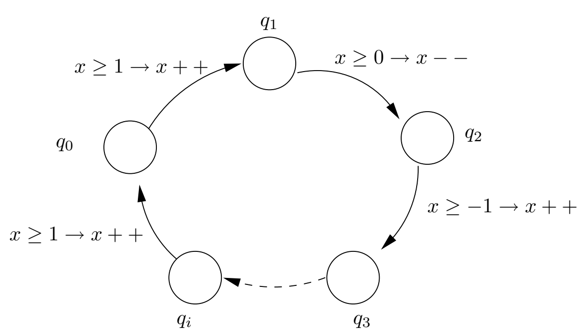
Let e_0 be the first edge in the path π'' and η' denote the path $\pi' e_0$, i.e., π' followed by the edge e_0. Let η'' be the suffix of π'' beyond e_0, i.e., $\pi'' = e_0\eta''$. Clearly $\eta'\eta''$ is the same path as $\pi'\pi''$ and the length of η'' is k. As a consequence the triple $(Cond(\eta''), WP(\eta'', p_3), depends_{\eta'})$ is in $extended\_templates(s.lc, k)$. Let $p'_1 = Cond(\eta'')$, $p'_2 = WP(\eta'', p_3)$ and $f' = depends_{\eta'}$. Since $s \sim_k t$, the following formula (C) is a validity.
(C) $(p'_1, f')[s] \wedge (p'_1, f')[t] \supset (p'_2, f')[s] \equiv (p'_2, f')[t])$

Using the validity of (C) we prove the validity of (A). Let C_0 be the guard of the transition e_0. Observe that $Cond(\pi'') = C_0 \wedge WP(e_0, Cond(\eta''))$ and $depends_{\eta'} = depends_{\pi'} \cdot depends_{e_0}$ and hence $f' = f \cdot depends_{e_0}$. Substituting $p_1 = Cond(\pi'')$ and using the above identity, the first conjunct on the left hand side of $\supset$ in (A) can be written as $(C_0, f)[s] \wedge (WP(e_0, Cond(\eta'')), f)[s]$. Similarly the second conjunct on the left hand side of (A) can be written as $(C_0, f)[t] \wedge (WP(e_0, Cond(\eta'')), f)[t]$. Let $p''_1 = WP(e_0, Cond(\eta''))$ and $p''_3 = WP(\pi'', p_3)$. Now consider the following formula (B):
(B) $(p''_1, f)[s] \wedge (p''_1, f)[t] \supset (p''_3, f)[s] \equiv (p''_3, f)[t]$

We show that (B) is a validity. From this it would follow that (A) is validity because the left hand side of (A) implies the left hand side of (A) as the former has more conjuncts. We show the validity of (B) by showing that it can be obtained from C by appropriate substitutions to some of the variables. First we need the following notation. Define a *term* to be a constant, a variable or an expression. Define a substitution to be a function from the set of variables X to the set of terms. We extend any substitution δ to apply to formulas, terms and predicates in a natural way; that is, for any such entity ϵ, $\delta(\epsilon)$ is the corresponding entity obtained by replacing every occurrence of every variable x in ϵ by $\delta(x)$. We define the composition of two substitutions δ_1, δ_2, denoted by $\delta_1 \cdot \delta_2$, so that $\delta_1 \cdot \delta_2(x) = \delta_1(\delta_2(x))$. Corresponding to the transition e_0, let δ be the substitution such that for any $x \in X$, $\delta(x)$ is defined as follows: if the assignment $x := \rho$ is in the assignments of e_0 then $\delta(x) = \rho$; if there is no such assignment for x then $\delta(x) = x$.

We show that we can obtain (B) from (C) by using the substitution δ. To show this, we prove the following property for any predicate p' and any symbolic state u: $(WP(e_0, p'), depends_{\pi'})[u]$ can be obtained from $(p', depends_{\eta'})[u]$ by using the substitution δ. Recall that $f = depends_{\pi'}$ and $depends_{\eta'} \ = f \cdot depends_{e_0}$. First we observe that $(WP(e_0, p'), depends_{\pi'})[u]$ and $(p', depends_{\eta'})[u]$ can be obtained from p' by respectively using the substitutions ϵ' and ϵ'' which are defined below. For any variable x, if x is assigned a constant c in e_0 then $\epsilon'(x) = c$ and $\epsilon''(x) = x$; if x is assigned a variable y then $\epsilon'(x) = u.exp(f(y))\{y/f(y)\}$ and $\epsilon''(x) = u.exp(f(y))\{x/f(y)\}$; ; if x is assigned $\psi(x)$ then $\epsilon'(x) = \psi(u.exp(f(x))\{x/f(x)\})$ and $\epsilon''(x) = u.exp(f(x))\{x/f(x)\}$; otherwise $\epsilon'(x) = x$ and $\epsilon''(x) = x$. Now it is not very difficult to see that $\delta((p', depends_{\eta'})[u]) = \delta(\epsilon''(p')) = (\delta \cdot \epsilon'')(p')$. Now, we see that for any variable x, $\delta \cdot \epsilon''(x) = \delta(\epsilon''(x))$. If there is no assignment for x in e_0 or x is assigned a constant or a variable in e_0 then it is straightforward to see that $\delta \cdot \epsilon''(x) \ = \epsilon'(x)$; if there is an assignment of the form $x := \psi(x)$ in e_0 then $\delta \cdot \epsilon''(x) = u.exp(f(x))\{\psi(x)/f(x)\}$; by using the commutativity property of the functions we see that $u.exp(f(x))\{\psi(x)/f(x)\} = \psi(u.exp(f(x))\{x/f(x)\}$ and hence $\delta \cdot \epsilon''(x) \ = \epsilon'(x)$. Using the fact that $\epsilon'(p') \ = (WP(e_0, p'), depends_{\pi'})[u]$, we see that $\delta((p', depends_{\eta'})[u]) = \delta \cdot \epsilon''(p') = \epsilon'(p') = (WP(e_0, p'), depends_{\pi'})[u]$. By similar argument, we can show that for any symbolic state u, $\delta((WP(\eta'', p_3), depends_{\eta'})[u]) = (WP(\pi'', p_3), depends_{\pi'})[u]$. Using the above observations with $u = s, t$ and with $p' \ = Cond(\eta'')$, it is easy to see that (B) can be obtained from (C) using the substitution δ. Hence (B) is a valid formula.

We prove part (2) of the theorem as follows. We consider the cases for odd k and even k separately. Let k be any positive odd integer and $i = k + 1$. Clearly i is even.

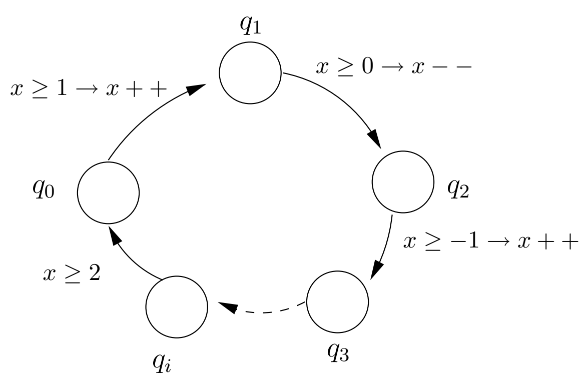


$q_0 : x \geq 1 \rightarrow x{+}{+}$
$q_{odd} : x \geq 0 \rightarrow x{-}{-}$
$q_{even} : x \geq -1 \rightarrow x{+}{+}$
$q_i : x \geq 1 \rightarrow x{+}{+}$

Fig. 4. Example of a TD for which $\sim_{i-1} \subset \sim_i$ where i is even

Consider the TD G given in figure 4. Let AP contain the single atomic predicate $x \geq 0$. The TD G consists of a cycle. The transitions entering and leaving q_0 are identical. For each even j such that $j \neq 0$ and $j \neq i$, the transitions from node q_j are all identical. Similarly, for all odd j, the transitions leaving q_j are identical. The transitions from q_0, from q_j for even j such that $j \neq 0$ and $j \neq i$, from q_j for odd j and from q_i are all given in the figure with appropriate labels. Let the initial state u be such that $u.lc = q_0$ and $u.x = 0$. It is not difficult to see that for any state v such that $v.lc = q_0$ and $v.x \geq 0$, the cycle from q_0 back to q_0 is feasible and executing this cycle increments x by 1. As a consequence, $Reach(G, u)$ has infinite number of states. Since G has no assignment in which x is assigned a constant or is assigned another variable, $Sym\_Reach(G, u)$ is isomorphic to $Reach(G, u)$. We show that the bi-simulation $\sim_i$ is the set of all (s, t) such that $s.lc = t.lc$. Let s, t be any two nodes in $Sym\_Reach(G, u)$ such that $s.lc = t.lc$. Consider a node q_j in G. Let π' be the simple path from $s.lc$ to q_j and π'' be the path of length of i from q_j. Since there are no assignments where x is assigned a constant or is assigned another variable, it should be obvious that $depends_{\pi'}$ is the identity function f_{id}. Let q_j be such that j is an odd number. It can be shown that $Cond(\pi'')$ is equivalent to $x \geq 2$. The path π'' is the path from q_j to q_{j-1}. The guard of the transition from q_{j-1} to q_j is either $x \geq 1$ or $x \geq -1$ (if $j = 1$ then it is the former else it is the later). As a consequence, the set $guards(q_{j-1}) \cup AP$ contains only the predicates $x \geq 1$, $x \geq -1$,$x \geq 0$. For each predicate p in the above set, it is easy to see that $WP(\pi'', p)$ is p itself. As a consequence $extended\_templates(s.lc, i)$ consists of triples of the form $(x \geq 2, p, f_{id})$ where p is one of the three predicates given above. For any p which is one of the above predicates, it is obvious that if $(x \geq 2, f_{id})[s]$ is true then $(p, f_{id})[s]$ is also true for every possible integer value of x; the same is true with respect to t. As a consequence, we see that for every triple (p', p'', f) in $extended\_templates(s.lc, i)$, the formula $(p', f)[s] \wedge (p', f)[t] \supset (p'', f)[s] \equiv (p'', t)[t]$ is a validity. In the above analysis we assumed that j is an odd integer such that $1 \leq j \leq i-1$. The above property can also be proved for other cases of j also.

Now we show that the bi-simulation relation $\sim_{i-1} \subset \sim_i$. We prove this by showing that there are some pairs (s, t) that are in $\sim_i$ but not in $\sim_{i-1}$. To see this, consider the path π'' of length $i-1$ from q_1 to q_i. It can be shown that $Cond(\pi'')$ is simply $x \geq 0$. Let p be the predicate $x \geq 1$ which is the guard of the transition from q_i. It is easy to see that $WP(\pi'', p)$ is $x-1 \geq 1$, i.e., $x \geq 2$. Let π' be the path consisting of the single edge from q_0 to q_1. From the definition of $extended\_templates(q_0, i-1)$, we see that $(x \geq 0, x \geq 2, depends_{\pi'})$ is in this set. It is trivial to see that $depends_{\pi'}$ is the identity function f_{id}. For any two symbolic states s, t where $s.lc = t.lc = q_0$ and $s.exp(x) = x + c$ and $t.exp(x) = x + d$, $(x \geq 0, f_{id})[s] \wedge (x \geq 0, f_{id})[s]$ is $x + c \geq 0 \wedge x + d \geq 0$; similarly, $(x \geq 2, f_{id})[s]$ and $(x \geq 2, f_{id})[t]$ are given by $x + c \geq 2$ and $x + d \geq 2$ respectively. In order for (s, t) to be in $\sim_{i-1}$, the formula $(x + c \geq 0 \wedge x + d \geq 0) \supset (x + c \geq 2 \equiv x + d \geq 2)$ needs to be a validity for the case when x ranges over all integers. Consider the case when c is any integer greater than or equal to zero and $d = c - 1$. For such cases the formula is not valid. To see this, take the value of x to be $2 - c$; it should be clear that the left hand side of the implication in the above formula is true while the right hand side is false. Thus such pairs of s, t are not in $\sim_{i-1}$. However these pairs are in $\sim_i$. when i is odd, we can use the similarly method with a slightly different TD given in figure 4. $\Box$



$q_0 : x \geq 1 \rightarrow x++$

$q_{odd} : x \geq 0 \rightarrow x--$

$q_{even} : x \geq -1 \rightarrow x++$

$q_i : x \geq 2$

Fig. 5. Example of a TD for which $\sim_{i-1} \subset \sim_i$ where i is odd