A logical reconstruction of reachability

Tatiana Rybina and Andrei Voronkov

University of Manchester
{rybina,voronkov}@cs.man.ac.uk

Abstract. In this paper we discuss reachability analysis for infinite-state systems. Infinite-state systems are formalized using transition systems over a first-order structure. We analyze the connections between the symbolic representation of transition systems and formulas used in various reachability algorithms. Our main results are related to the so-called *guarded assignment systems*.
Keywords: theoretical foundations, model theory, infinite-state systems, reachability analysis.

1 Introduction

Reachability properties arise is many applications of verification. In this paper we discuss reachability algorithms in infinite-state systems. Infinite-state systems are formalized using transition systems over a first-order structure. We analyze the connections between the symbolic representation of transition systems and formulas which are used in various reachability algorithms. Our main results are related to the so-called *guarded assignment systems*.

This paper serves two main purposes. First, it formalizes infinite-state systems using model-theoretic notions and discusses reachability algorithms based on this formalization. Though many results and observations of this paper form part of folklore circulating in the infinite-state model checking community, we believe that our formalization is useful since it gives a common model-theoretic approach to otherwise quite diverse formalizations. Second, we observe that for a large class of systems, called guarded assignment systems (GAS), the reachability analysis is simpler since only formulas of a special form are used for satisfiability- and entailment-checking. Many known formalizations of broadcast, cache coherence, and other protocols belong to the simplest kind of GAS, called simple GAS. It follows from our results that the so-called local backward reachability algorithms can be used for simple GAS over structures in which satisfiability-checking is decidable for conjunctions of atomic formulas. This, for example, allows one to extend the existing reachability algorithms to some theories of queues.

This paper is organized as follows. In Section 2 we briefly overview relevant parts of model theory and define several classes of first-order formulas. In Section 3 we give a model theory-based formalization of transition systems and their symbolic representations and introduce basic forward and backward reachability algorithms. We also discuss requirements on the underlying first-order structures. In Section 4 we define guarded assignment systems and reachability algorithms for them. In Section 5 we introduce local reachability algorithms for guarded assignment systems. Finally, in Section 6 we discuss some other issues related to our formalization of infinite-state systems and reachability.

2 Preliminaries

In this section we define notation and several classes of formulas which will be used in the rest of the paper. We assume knowledge of standard model-theoretic definitions, such as first-order formulas, structure and truth, which can be found in any standard textbook on logic or model theory.

Unless stated otherwise, we will deal with a fixed first-order structure $\mathbb{M}$ with a domain $\mathcal{D}$ and assume that all formulas are formulas of the signature of this structure. A *valuation* for a set of variables V in $\mathbb{M}$ is any mapping $s : V \to \mathcal{D}$. We will use the standard model-theoretical notation $\mathbb{M}, s \vDash A$ to denote that the formula A is true in the structure $\mathbb{M}$ under a valuation s. When we use this notation, we assume that s is defined on all free variables of A.

A formula A with free variables V is said to be *satisfiable* (respectively, *valid*) in $\mathbb{M}$ if there exists a valuation s for V in $\mathbb{M}$ such that $\mathbb{M}, s \vDash A$ (respectively, for every valuation s we have $\mathbb{M}, s \vDash A$).

A formula A is called

- *quantifier-free* if A contains no quantifiers.
- *positive existential* if A can be built from atomic formulas using just $\exists, \wedge, \vee$;
- a *conjunctive constraint* if A can be built from atomic formulas using just $\exists, \wedge$;
- a *simple constraint* if A is a conjunction of atomic formulas.

3 Infinite-state transition systems and reachability

In this section we introduce a logical formalization of transition systems.

3.1 Infinite-state transition systems and their symbolic representation

Our formalization of transition systems is as follows. A transition system has a finite number of variables with values in a possibly infinite domain $\mathcal{D}$. A state is a mapping from variables to values. Transitions may change values of variables. A symbolic representation of such a system uses first-order formulas interpreted in a structure with the domain $\mathcal{D}$.

DEFINITION 1 (Transition System) A *transition system* is a tuple $\mathbb{S} = (\mathcal{V}, \mathcal{D}, \mathcal{T})$, where

1. $\mathcal{V}$ is a finite set of *state variables*.
2. $\mathcal{D}$ is a non-empty set, called the *domain*. Elements of $\mathcal{D}$ are called *values*.

A *state* of the transition system $\mathbb{S}$ is a function $s : \mathcal{V} \to \mathcal{D}$.

3. $\mathcal{T}$ is a set of pairs of states, called the *transition relation* of $\mathbb{S}$.

A transition system $\mathbb{S}$ is said to be *finite-state* if $\mathcal{D}$ is finite, and *infinite-state* otherwise. ❑

We call any set of pairs of states a *transition*. Transition systems arising in practical applications often have variables ranging over different domains. For example, some state variables may range over the natural numbers, while others over the boolean values or other finite domains. We introduce a single domain for simplicity. It is not hard to generalize our formalization to several domains using many-sorted first order logic. In the sequel we assume a fixed transition system $\mathbb{S} = (\mathcal{V}, \mathcal{D}, \mathcal{T})$.

Suppose that $\mathbb{M}$ is a first-order structure whose domain is $\mathcal{D}$. For example, if $\mathcal{D}$ is the set of natural numbers, then the structure $\mathbb{M}$ can be the set of natural numbers together with the order $<$, operation $+$ and constants $0, 1$. In addition to the set of state variables $\mathcal{V}$, we also introduce a set $\mathcal{V}'$ of *next state variables* of the same cardinality as $\mathcal{V}$. We fix a bijection $' : \mathcal{V} \to \mathcal{V}'$ such that $v \in \mathcal{V}$ we have $v' \in \mathcal{V}'$.

We can treat the variables in $\mathcal{V} \cup \mathcal{V}'$ also as logical variables. Then any mapping $s : \mathcal{V} \to \mathcal{D}$ can be considered as both a state of the transition system $\mathbb{S}$ and a valuation for $\mathcal{V}$ in the structure $\mathbb{M}$, and similarly for $s' : \mathcal{V}' \to \mathcal{D}$.

DEFINITION 2 (Symbolic Representation) Let S be a set of states and A be a formula with free variables in $\mathcal{V}$. We say that A *symbolically represents* S *in* $\mathbb{M}$, or simply *represents* S if for every valuation s for $\mathcal{V}$ in $\mathbb{M}$ we have

$$s \in S \Leftrightarrow \mathbb{M}, s \vDash A.$$

Likewise, we say that a formula B with free variables in $\mathcal{V} \cup \mathcal{V}'$ *(symbolically) represents a transition* T *in* $\mathbb{M}$ if for every pair of valuations s, s' in $\mathbb{M}$ for $\mathcal{V}$ and $\mathcal{V}'$ respectively we have

$$(s, s') \in T \Leftrightarrow \mathbb{M}, s, s' \vDash B. \qquad ❑$$

In the sequel we will follow the following convention. We will often identify a symbolic representation of a transition with the transition itself. For example, when T is a formula with free variables in $\mathcal{V} \cup \mathcal{V}'$ we can refer to T as a transition.

DEFINITION 3 We say that a state s_n is *forward reachable* from a state s_0 w.r.t. $\mathcal{T}$ if there exists a sequence of states $s_1, \ldots, s_{n-1}$ such that for all $i \in \{0, \ldots, n-1\}$ we have $(s_i, s_{i+1}) \in \mathcal{T}$. In this case we also say that s_n is reachable from s_0 *in* n *steps* and that s_0 is *backward reachable* from s_n in n steps. ❑

Instead of "forward reachable" we will usually say "reachable". When we speak about reachability with respect to $\mathcal{T}$, and $\mathcal{T}$ is clear from the context, we will simply say "reachable".

The reachability problem can now be defined as follows.

DEFINITION 4 (Reachability Problem) The *reachability problem for* $\mathbb{M}$ is the following decision problem. Given formulas In, Fin, and Tr such that

1. In represents a set of states I, called the set of *initial states*;

2. *Fin* represents a set of states F, called the set of *final states*;
3. *Tr* represents the transition relation of a transition system $\mathbb{S}$,

do there exist states $s_1 \in I$, $s_2 \in F$ such that s_2 is reachable from s_1 w.r.t. *Tr*? ❑

In fact, reachability is a family of decision problems parametrized by the structure $\mathbb{M}$.

When we discuss instances of the reachability problem, we will call the formulas *In* and *Fin* the *initial* and *final conditions*, respectively, and *Tr* the *transition formula*.

The reachability problem for infinite-state systems is, in general, undecidable. Various results on reachability are discussed in many papers, including [16, 2, 21, 3].

3.2 Algorithms for checking reachability

The algorithms for checking reachability are based on the idea of building a symbolic representation of the set of reachable states. There are two main kinds of reachability algorithms: *forward reachability* and *backward reachability*. Forward reachability algorithms try to build the set of states reachable from the initial states and check whether this set of states contains any final state. Backward reachability algorithms try to build the set of states backward reachable from the final states and check if this set of states contains any initial state.

Before defining these algorithms, let us discuss symbolic representations of forward and backward reachable states. We assume fixed initial and final conditions $In(\mathcal{V})$, $Fin(\mathcal{V})$ and the transition formula $Tr(\mathcal{V}, \mathcal{V}')$.

Let $A(\mathcal{V})$ be a formula which represents a set of states S. It is not hard to argue that the set of states reachable in one step from a state in S can be represented by the formula $\exists \mathcal{V}_1(A(\mathcal{V}_1) \wedge Tr(\mathcal{V}_1, \mathcal{V}))$. Likewise, the set of states backward reachable in one step from a state in S is represented by the formula $\exists \mathcal{V}_1(A(\mathcal{V}_1) \wedge Tr(\mathcal{V}, \mathcal{V}_1))$. This observation implies the following facts about forward and backward reachability.

LEMMA 5 (Forward Reachability) *Consider the sequence of formulas FR_i defined as follows:*

$$\begin{aligned} FR_0(\mathcal{V}) &= In(\mathcal{V}); \\ FR_{i+1}(\mathcal{V}) &= FR_i(\mathcal{V}) \vee \exists \mathcal{V}_1(FR_i(\mathcal{V}_1) \wedge Tr(\mathcal{V}_1, \mathcal{V})). \end{aligned}$$

Then each FR_i symbolically represents the set of formulas reachable from I in at most i steps. ❑

LEMMA 6 (Backward Reachability) *Consider the sequence of formulas BR_i defined as follows:*

$$\begin{aligned} BR_0(\mathcal{V}) &= Fin(\mathcal{V}); \\ BR_{i+1}(\mathcal{V}) &= BR_i(\mathcal{V}) \vee \exists \mathcal{V}_1(BR_i(\mathcal{V}_1) \wedge Tr(\mathcal{V}, \mathcal{V}_1)). \end{aligned}$$

Then each BR_i symbolically represents the set of formulas backward reachable from F in at most i steps. ❑

Using these lemmas, one can prove two theorems which form the basis of reachability algorithms.

Theorem 1 (Reachability). *(i) There exists a final state reachable from an initial state if and only if there exists a number $i \geq 0$ such that $\mathbb{M} \models \exists \mathcal{V}(FR_i(\mathcal{V}) \wedge Fin(\mathcal{V}))$. (ii) If there exists a number $i \geq 0$ such that $\mathbb{M} \not\models \exists \mathcal{V}(FR_i(\mathcal{V}) \wedge Fin(\mathcal{V}))$ and $\mathbb{M} \models \forall \mathcal{V}(FR_{i+1}(V) \rightarrow FR_i(\mathcal{V}))$, then there exists no final state reachable from an initial state.*

The same statements hold if one replaces FR by BR and Fin by In.

This theorem reduces, in some sense, reachability-checking to checking whether formulas of a special form are true in the structure $\mathbb{M}$. The form of these formulas depends on the symbolic representations In, Fin, Tr of the sets of initial and finite states and the transition relation.

```
procedure Forward
input: formulas In, Fin, Tr
output: "reachable" or "unreachable"
begin
  current(V) := In(V)
  while M ⊭ ∃V(current(V) ∧ Fin(V))
    next(V) :=
      simp(current(V) ∨
            ∃V1(current(V1) ∧ Tr(V1, V)) )
    if M ⊨ ∀V(next(V) → current(V))
      then return "unreachable"
    current(V) := next(V)
  return "reachable"
end
```

Fig. 1. Forward reachability algorithm

```
procedure Backward
input: formulas In, Fin, Tr
output: "reachable" or "unreachable"
begin
  current(V) := Fin(V)
  while M ⊭ ∃V(current(V) ∧ In(V))
    prev(V) :=
      simp(current(V) ∨
            ∃V1(current(V1) ∧ Tr(V, V1)) )
    if M ⊨ ∀V(prev(V) → current(V))
      then return "unreachable"
    current(V) := prev(V)
  return "reachable"
end
```

Fig. 2. Backward reachability algorithm

The basic reachability algorithms are based on explicit computation of the formulas FR_i and BR_i and the use of Theorem 1.

DEFINITION 7 The *forward reachability algorithm* Forward and the *backward reachability algorithm* Backward are shown in Figures 1 and 2. They are parametrized by a function simp, called a *simplification function*. ❏

Usually, this formula transforms formulas into equivalent ones, i.e., $\mathbb{M} \models \forall \mathcal{V}(A \leftrightarrow \mathsf{simp}(A))$ for all formulas A.

In different reachability algorithms, the simplification function simp may make a simple transformation, for example into a normal form, but may also perform more complex ones, such as quantifier elimination. We will sometimes need the simplification function simp to preserve some class of formulas. To this end, we introduce the following definition. We call a function simp on formulas *stable* w.r.t. a class of formulas $\mathcal{C}$, if for every formula $A \in \mathcal{C}$ we have $\mathsf{simp}(A) \in \mathcal{C}$.

3.3 First-order theories of infinite structures

In this section we analyze first-order structures which are suitable for reachability algorithms. In both forward and backward reachability algorithms one has to solve repeatedly the following problems:

1. *Satisfiability:* whether the formula $\mathsf{current}(\mathcal{V})$ is satisfiable in $\mathbb{M}$.
2. *Entailment:* whether the formula $\mathsf{next}(\mathcal{V}) \rightarrow \mathsf{current}(\mathcal{V})$ or $\mathsf{prev}(\mathcal{V}) \rightarrow \mathsf{current}(\mathcal{V})$ is valid in $\mathbb{M}$.

Of course the reachability algorithms of Figures 1 and 2 can only be considered as algorithms modulo the assumption that satisfiability- and entailment-checking are decidable.

It is not hard to argue that the formulas $\mathsf{current}(\mathcal{V})$ at the ith iteration of the while-loop are equivalent to the formulas FR_i for forward reachability and BR_i for backward reachability, provided that simp is equivalence-preserving. If $\mathsf{simp}(A) = A$ for every formula A, then the formulas $\mathsf{current}(\mathcal{V})$ are exactly the formulas FR_i for the forward reachability algorithm and BR_i for the backward reachability algorithm. This implies that the reachability algorithms have the following properties.

THEOREM 8 (Soundness and Semi-Completeness) *The algorithms* Forward *and* Backward *have the following properties:*

1. *there is a final state reachable from an initial state if and only if the algorithm returns "reachable";*
2. *if the algorithm returns "unreachable", then there is no final state reachable from an initial state.* ❏

On some inputs the algorithms do not terminate. In this case we say that the reachability analysis *diverges*.

Complexity of the satisfiability and entailment problems depends on the structure $\mathbb{M}$ and the form of the formulas $\mathsf{current}$ used in the algorithm.

If the first-order theory of $\mathbb{M}$ is decidable, one can implement satisfiability- and entailment-checking using a decision procedure for the first-order theory of $\mathbb{M}$. In some cases, one can even use off-the-shelf decision procedures or libraries. For example, [9] use the Omega Library [30] for deciding Presburger arithmetic. The use of general-purpose algorithms to decide specific classes of formulas may be inefficient. Then specialized tools may be needed to decide these classes of formulas more efficiently.

It is not hard to see that in many important special cases the formulas FR_i and BR_i have a special form.

LEMMA 9 *Let In, Fin, and Tr be positive existential formulas and* simp *be stable w.r.t. positive existential formulas. Then the formulas* current *in the algorithms* Forward *and* Backward *are positive existential too.* ❏

It follows from this lemma that it is enough to solve satisfiability and entailment for positive existential formulas only, as soon as the initial and final conditions and the transition formula are positive existential formulas, which they often are. As we shall see below, for some important special cases even quantifier-free formulas suffice.

If the first-order theory of $\mathbb{M}$ admits quantifier elimination, i.e., every formula is effectively equivalent to a quantifier-free formula, then it is enough to check satisfiability and entailment of quantifier-free formulas only. This can be achieved by applying quantifier elimination, i.e., replacing quantified formulas by equivalent quantifier-free formulas, whenever quantified formulas appear, for example, when transitions are applied. However, quantifier elimination may be expensive.

3.4 Integer-based systems

Denote by I the set of integers. Consider the following structure

$$\mathbb{I} = (I, >, <, \geq, \leq, +, -, 0, 1, 2, \ldots),$$

where all the function and predicate symbols (for example $>$) have their standard interpretation over integers. The first-order theory of $\mathbb{I}$ is decidable, which means that one can use the reachability algorithms Forward and Backward even when the initial and final conditions and the transition formula are arbitrary first-order formulas. In addition, infinite-state transition systems (of a very simple form) over the domain of integers can be used for specifying safety properties of a large classes of protocols. The first-order theory of $\mathbb{I}$ (essentially, the Presburger arithmetic) admits quantifier elimination, if we extend the signature of $\mathbb{I}$ by the predicates expressing $m \mid (x - n)$ for concrete natural numbers m, n. We will call the transition systems over $\mathbb{I}$ the *integer transition systems* and the reachability problem over $\mathbb{I}$ the *integer reachability problem*.

Off-the-shelf tools, such as Omega Library [30] or the Composite Library [33], are available to decide the Presburger arithmetic or its fragments.

The fact that satisfiability- and entailment-checking are decidable does not necessarily imply that the reachability problem is decidable. Neither does it imply that termination of the reachability algorithms is decidable. In this section we note the following undecidability result which will carry over to all classes of transition systems and reachability algorithms studied in this paper.

THEOREM 10 *Consider the instances of the integer reachability problem with three state variables c_1, c_2, s whose transition relation is a disjunction of conjunctions of the following formulas: $c_i > 0$ or $s = n$, where n is a natural number, $c_i' = c_i + 1$, $c_i' = c_i - 1$, $c_i' = c_i$, or $s' = m$, where m is a natural number. There is exactly one initial and one final state. Then*

1. *the reachability problem for this class is undecidable.*
2. *termination of the algorithm* Forward *is undecidable.*
3. *termination of the algorithm* Backward *is undecidable.* ❏

One can easily prove this theorem by encoding two-counter machines by transition systems of this form. The systems used in the theorem are simple guarded assignment systems (the definition is given below).

Some decidability results for integer-based systems are given in e.g., [16, 2]. For example, safety properties of broadcast protocols can be represented as instances of the integer reachability problem, and the backward reachability algorithm terminates on these instances [16].

4 Guarded assignment transition systems

In this section we consider an important special case of transition systems in which quantifier-free formulas can be used for backward reachability, even without the use of expensive quantifier elimination algorithms. Guarded assignment systems are in the core of the model-checking system BRAIN [32]

4.1 Guarded assignments

As usual, we assume that $\mathcal{V}$ the set of state variables of the transition system.

DEFINITION 11 (Guarded Assignment) A *guarded assignment* is any formula (or transition) of the form

$$P \wedge v_1' = t_1 \wedge \ldots \wedge v_n' = t_n \wedge \bigwedge_{v \in \mathcal{V} - \{v_1, \ldots, v_n\}} v' = v,$$

where P is a formula with free variables $\mathcal{V}$, $\{v_1, \ldots, v_n\} \subseteq \mathcal{V}$, and $t_1, \ldots, t_n$ are terms with variables in $\mathcal{V}$. We will write guarded assignments as

$$P \ \Rightarrow \ v_1 \ := \ t_1, \ldots, v_n \ := \ t_n. \tag{1}$$

The formula P is called the *guard* of this guarded assignment.

A guarded assignment is *quantifier-free* (respectively, *positive existential*), if so is its guard. A guarded assignment is called *simple existential* if its guard is a conjunctive constraint, and *simple* if its guard is a simple constraint. ❏

Formula (1) represents a transition which applies to states satisfying P and changes the values of variables v_i to the values of the terms t_i. Note that a guarded assignment T is a deterministic transition: for every state s there exists at most one state s' such that $(s, s') \in T$. Moreover, such a state s' exists if and only if the guard of this guarded assignment is true at s, i.e., $\mathbb{M}, s \models P$.

DEFINITION 12 (Guarded Assignment System) A transition system is called a *guarded assignment system*, or simply *GAS*, if its transition relation is a union of a finite number of guarded assignments. A GAS is called *quantifier-free* (respectively, *positive existential*, *simple existential*, or *simple*) if every guarded assignment in it is also quantifier-free (respectively positive existential, simple existential, or simple). ❏

$$\begin{array}{rcll} drinks > 0 \land customers > 0 \Rightarrow drinks & := & drinks - 1 & \texttt{(* dispense-drink *)} \\ true \Rightarrow drinks & := & drinks + 64 & \texttt{(* recharge *)} \\ true \Rightarrow customers & := & customers + 1 & \texttt{(* customer-coming *)} \\ customers > 0 \Rightarrow customers & := & customers - 1 & \texttt{(* customer-going *)} \end{array}$$

Fig. 3. Guarded assignment system

An example integer guarded assignment system for representing a drink dispenser is given in Figure 3. This system is a union of transitions shown in this figure. Since all guards are simple constraints, the system is simple.

Note that every guarded assignment represents a deterministic transition, but a guarded assignment system may represent a non-deterministic transition system because several guards may be true in the same state. Not every transition system is a guarded assignment system. Indeed, in every guarded assignment system with a transition relation $\mathcal{T}$ for every state s there exists a finite number of states s' such that $(s, s') \in \mathcal{T}$. The greatest transition $true$ over any infinite structure $\mathbb{M}$ does not have this property.

THEOREM 13 *Every integer quantifier-free GAS is also a simple GAS.*

Proof. Since every quantifier-free formula is equivalent to a formula in disjunctive normal form, we can assume that the guards are in disjunctive normal form. Since in $\mathbb{I}$ the negation of an atomic formula is equivalent to a disjunction of atomic formulas (for example $\neg x = y$ is equivalent to $x > y \lor y > x$ and $\neg x > y$ is equivalent to $y \geq x$), we can assume that every guard is a disjunction $C_1 \lor \ldots \lor C_n$ of conjunctions of atomic formulas. But every guarded assignment

$$C_1 \lor \ldots \lor C_n \Rightarrow A \tag{2}$$

is equivalent to the union of the following simple guarded assignments

$$C_1 \;\Rightarrow\; A, \quad \ldots, \quad C_n \;\Rightarrow\; A,$$

so we can equivalently replace guarded assignment (2) by n simple guarded assignments.

One can generalize this theorem to structures different from $\mathbb{I}$. Indeed, the only property of $\mathbb{I}$ used in this proof is that the negation of an atomic formula is equivalent to a positive quantifier-free formula.

The notion of a guarded assignment system is not very restrictive. Indeed, broadcast protocols and Petri nets can be represented as integer simple guarded assignment systems. All transition systems for cache coherence protocols described in [13] are integer simple GAS.

Let us note one interesting property of GAS related to properties other than reachability. Let us call a state s a *deadlock state* if there is no state s' such that $(s, s') \in \mathcal{T}$, where $\mathcal{T}$ is the transition relation.

THEOREM 14 *Let the transition relation $\mathcal{T}$ be a union of guarded assignments* $(P_1 \Rightarrow A_1), \ldots, (P_n \Rightarrow A_n)$. *Then the set of all deadlock states is represented by the formula* $\neg(P_1 \vee \ldots \vee P_n)$. ❑

This theorem shows that checking *deadlock-freedom* (i.e., non-reachability of a deadlock state) of GAS may be as easy as checking reachability properties, for example, if the GAS is quantifier-free, then the set of deadlock states can also be represented by a quantifier-free formula. Theorem 14 is also used in the system BRAIN to generate the deadlock-freedom conditions automatically.

4.2 Reachability algorithms for guarded assignment systems

We introduced simple guarded assignment systems because they have a convenient property related to backward reachability algorithms. Essentially, one can use backward reachability algorithms for these systems when the underlying structure has a much weaker property than the decidability of the first-order theory.

Let us see how the formulas FR_i and BR_i look like in the case of simple guarded assignment systems. Let u be a guarded assignment of the form

$$P(v_1, \ldots, v_n) \Rightarrow v_1 \ := \ t_1, \ldots, v_n \ := \ t_n.$$

For simplicity we assume that $\mathcal{V} = \{v_1, \ldots, v_n\}$. This can be achieved by adding "dummy" assignments $v \ := \ v$ for every variable $v \in \mathcal{V} - \{v_1, \ldots, v_n\}$. Let also $A(v_1, \ldots, v_n)$ be a formula whose free variables are in $\mathcal{V}$. For every term t denote by t' the term obtained from t by replacing every occurrence of every state variable v_i by v_i'.

Define the following formulas:

$$A^u(v_1, \ldots, v_n) \stackrel{\text{def}}{=} \exists \mathcal{V}'(A(v_1', \ldots, v_n') \wedge P(v_1', \ldots, v_n') \wedge v_1 = t_1' \wedge \ldots \wedge v_n = t_n');$$
$$A^{-u}(v_1, \ldots, v_n) \stackrel{\text{def}}{=} P(v_1, \ldots, v_n) \wedge A(t_1, \ldots, t_n).$$

LEMMA 15 *Let $A(v_1, \ldots, v_n)$ represent a set of states S. Then*

1. *the formula $A^u(v_1, \ldots, v_n)$ represents the set of states reachable in one step from S using u;*
2. *the formula $A^{-u}(v_1, \ldots, v_n)$ represents the set of states backward reachable in one step from S using u.*

Proof. For forward reachability the statement is straightforward, so let us prove the backward reachability part. The set of states backward-reachable in one step from S is represented by the formula

$$\exists \mathcal{V}'(P(v_1, \ldots, v_n) \wedge A(v_1', \ldots, v_n') \wedge v_1' = t_1 \wedge \ldots \wedge v_n' = t_n). \tag{3}$$

Using the fact that in first-order logic the formula $\exists x(x = t \wedge B(x))$, where t does not occur in x, is equivalent to $B(t)$, it is not hard to argue that formula (3) is equivalent to $A^{-u}(v_1, \ldots, v_n)$.

One can deduce from this lemma that the formulas A^u and A^{-u} have a special form, as expressed by the following lemma.

LEMMA 16 *Let u be a guarded assignment. Then the function $A \mapsto A^u$ is stable w.r.t. positive existential formulas and conjunctive constraints. The function $A \mapsto A^{-u}$ is stable w.r.t. positive existential formulas, conjunctive constraints, quantifier-free formulas, and simple constraints.* ❑

So far forward and backward reachability were treated symmetrically. Lemma 16 shows an asymmetry between forward and backward reachability for simple GAS: the predicate transformer corresponding to backward reachability yields simpler formulas.

Using Lemma 15, one can modify forward and backward reachability algorithms of Figures 1 and 2 for guarded assignment transition systems.

DEFINITION 17 The *forward reachability algorithm* GASForward and the *backward reachability algorithm* GASBackward for GAS are shown in Figures 4 and 5. When we check reachability for a GAS whose transition relation is a union of guarded assignments $u_1, \ldots, u_n$, we let $U = \{u_1, \ldots, u_n\}$ in the input of the algorithm. ❑

procedure GASForward
input: formulas In, Fin,
    finite set of guarded assignments U
output: "reachable" or "unreachable"
begin
  current := In
  while $\mathbb{M} \not\models \exists\mathcal{V}(\mathsf{current} \wedge Fin)$
    next := $\mathsf{current} \vee \bigvee_{u \in U} \mathsf{current}^u$
    if $\mathbb{M} \models \forall\mathcal{V}(\mathsf{next} \rightarrow \mathsf{current})$
      then **return** "unreachable"
    current := next
  return "reachable"
end

Fig. 4. Forward reachability algorithm for GAS

procedure GASBackward
input: formulas In, Fin,
    finite set of guarded assignments U
output: "reachable" or "unreachable"
begin
  current := In
  while $\mathbb{M} \not\models \exists\mathcal{V}(\mathsf{current} \wedge In)$
    prev := $\mathsf{current} \vee \bigvee_{u \in U} \mathsf{current}^{-u}$
    if $\mathbb{M} \models \forall\mathcal{V}(\mathsf{prev} \rightarrow \mathsf{current})$
      then **return** "unreachable"
    current := prev
  return "reachable"
end

Fig. 5. Backward reachability algorithm for GAS

THEOREM 18 (Soundness and Semi-Completeness) *The algorithms* GASForward *and* GASBackward *have the following properties:*

1. *there is a final state reachable from an initial state if and only if the algorithm returns "reachable";*

2. *if the algorithm returns "unreachable", then there is no final state reachable from an initial state.* ❏

Moreover, GASForward *(respectively,* GASBackward*) terminates if and only if* Forward *(respectively* Backward*) terminates.* ❏

THEOREM 19 *If the input GAS and the formulas In, Fin are quantifier-free, then all of the formulas* current *in the algorithm* GASBackward *are quantifier-free too. If, in addition, the input GAS and In, Fin are negation-free, then* current *is also negation-free.* ❏

This theorem shows that for quantifier-free GAS satisfiability and entailment in GASBackward should only be checked for quantifier-free formulas. It does not hold for the forward reachability algorithm GASForward. Checking satisfiability and validity of quantifier-free formulas is possible if the existential theory (i.e., the set of all sentences of the form $\exists X A$, where A is quantifier-free) of the structure $\mathbb{M}$ is decidable. There exist theories whose full first-order theory is undecidable but existential theory is decidable. A simple example is Presburger arithmetic extended by the divisibility predicate [5, 28, 24], though it is not clear how useful is this theory for applications. A more practical examples are some theories of queues considered in [8, 7]. This shows that there are infinite domains for which backward reachability may be easier to organize than forward reachability. Of course one can omit the entailment checks from the algorithms, but then they will only be applicable for checking reachability, but not for checking non-reachability.

5 Local algorithms

In this section we study so-called *local* algorithms, which are different from the algorithms discussed above. They are simpler since checking for satisfiability and entailment for simple GAS is only performed for conjunctive constraints, but not so powerful since they may diverge for instances of reachability for which the previously described algorithms terminate.

The idea of these algorithms is to get rid of disjunctions which appear in the formulas current even when the input contains no disjunctions at all. Instead of a disjunction $C_1 \vee \ldots \vee C_n$, one deals with the set of formulas $\{C_1, \ldots, C_n\}$. The entailment check is not performed on the disjunction, but separately on each member C_i of the disjunction, and is therefore called a *local entailment check*.

DEFINITION 20 (Local Reachability Algorithms) The *local forward reachability algorithm* LocalForward and *local backward reachability algorithm* LocalBackward are given in Figures 6 and 7. They are parametrized by a function *select* which selects a formula in a set of formulas. The input set of guarded assignments U is defined as in the algorithms GASForward and GASBackward. The input sets of formulas IS and FS are any sets of formulas such that $In = \bigvee_{A \in IS} A$ and $Fin = \bigvee_{A \in FS} A$. ❏

Local algorithms can be used not only for guarded assignment systems, but for every system in which the transition relation is represented by a finite union of transitions.

procedure LocalForward
input: sets of formulas IS, FS,
finite set of guarded assignments U
output: "reachable" or "unreachable"
begin
  if there exist $I \in IS, F \in FS$ such that
    $\mathbb{M} \models \exists\mathcal{V}(I \wedge F)$ **then**
    return "reachable"
  $unused$:= IS
  $used$:= $\emptyset$
  while $unused \neq \emptyset$
    S := $select(unused)$
    $used$:= $used \cup \{S\}$
    $unused$:= $unused - \{S\}$
    forall $u \in U$
      N := S^u
      if there exists $F \in FS$ such that
        $\mathbb{M} \models \exists\mathcal{V}(N \wedge F)$ **then**
        return "reachable"
      if for all $C \in used \cup unused$
        $\mathbb{M} \not\models \forall\mathcal{V}(N \rightarrow C)$ **then**
        $unused = unused \cup \{N\}$
        forall $C' \in used \cup unused$
          if $\mathbb{M} \models \forall\mathcal{V}(C' \rightarrow N)$ **then**
            remove C' from $used$ or $unused$
  return "unreachable"
end

Fig. 6. Local forward reachability algorithm

procedure LocalBackward
input: sets of formulas IS, FS,
finite set of guarded assignments U
output: "reachable" or "unreachable"
begin
  if there exist $I \in IS, F \in FS$ such that
    $\mathbb{M} \models \exists\mathcal{V}(I \wedge F)$ **then**
    return "reachable"
  $unused$:= FS
  $used$:= $\emptyset$
  while $unused \neq \emptyset$
    S := $select(unused)$
    $used$:= $used \cup \{S\}$
    $unused$:= $unused - \{S\}$
    forall $u \in U$
      N := S^{-u}
      if there exists $I \in IS$ such that
        $\mathbb{M} \models \exists\mathcal{V}(N \wedge I)$ **then**
        return "reachable"
      if for all $C \in used \cup unused$
        $\mathbb{M} \not\models \forall\mathcal{V}(N \rightarrow C)$ **then**
        $unused = unused \cup \{N\}$
        forall $C' \in used \cup unused$
          if $\mathbb{M} \models \forall\mathcal{V}(C' \rightarrow N)$ **then**
            remove C' from $used$ or $unused$
  return "unreachable"
end

Fig. 7. Local backward reachability algorithm

THEOREM 21 (Soundness and Semi-Completeness) *The algorithms* LocalForward *and* LocalBackward *have the following properties:*

1. *there is a final state reachable from an initial state if and only if the algorithm returns "reachable";*
2. *if the algorithm returns "unreachable", then there is no final state reachable from an initial state.* ❑

However, we cannot guarantee that the algorithms terminate if and only if their non-local counterparts terminate.

Example 1. Consider an example integer GAS with one variable v whose transition relation is represented by a single guarded assignment

$$true \Rightarrow v \ := \ v - 1.$$

Take $0 > 0$ as the initial condition and $v \neq 0$ as the final condition. The non-local backward reachability algorithm for GAS at the second iteration will generate the formula $v \neq 0 \vee v \neq 1$, and at the third iteration $v \neq 0 \vee v \neq 1 \vee v \neq 2$. Since these formulas entail each other, the algorithm terminates. However, the local backward reachability algorithm for GAS generates formulas $v \neq 0$, $v \neq 1$, $v \neq 2$ which do not entail one another, and hence diverges. One can give a similar example for forward reachability.

It is not difficult to prove the following result.

THEOREM 22 *If* LocalForward *(respectively,* LocalBackward*) terminates, then so does* GASForward *(respectively,* GASBackward*).* ❑

Note that termination of the local algorithms may depend on the selection function $select$. Let us call the selection function *fair* if no formula remains in $unused$ forever.

THEOREM 23 *If the local forward (respectively backward) algorithm terminates for some selection function, then it terminates for every fair selection function.* ❑

It is difficult to say if there are problems coming from real applications for which a non-local algorithm terminates but its local counterpart does not. One can prove, for example, that for broadcast protocols LocalBackward always terminates. In fact, this algorithm is implicitly used in the proof of decidability of broadcast protocols in [16].

The main property of the local algorithms is that subsumption and entailment-checking is only performed on simpler classes of formulas.

THEOREM 24 *If the input GAS is conjunctive and the sets IS, FS are sets of conjunctive constraints, then all of the formulas* current *in both* LocalForward *and* LocalBackward *are conjunctive constraints. If the input GAS is simple and the sets IS, FS are simple constraints then all of the formulas* current *in* LocalBackward *are simple constraints too.* ❑

The last property of local reachability algorithms is heavily used in the system BRAIN. For some structures (for example the reals or the integers) the algorithms for satisfiability-checking of simple constraints are a heavily investigated subject. If the structure is $\mathbb{I}$, then a simple constraint is essentially a system of linear equations and inequations. Checking satisfiability of simple constraints means solving such a system. For solving systems of linear equations and inequations over integers or reals several off-the-shelf tools are available. For example, [14] implement satisfiability- and entailment-checking for conjunctive constraints over real numbers using the corresponding built-in functions of the SICStus Prolog constraint library.

When a backward reachability algorithm is performed on GAS, we get a quantifier elimination effect due to the special form of formulas A^{-u}. Theorems 19 and 24 show that in many cases entailment can be checked only between quantifier-free formulas or even simple constraints. Let us show that in some cases entailment-checking can be performed using satisfiability-checking. It is easy to see that a formula $\forall\mathcal{V}(A \rightarrow B)$ is valid if and only if the formula $\exists\mathcal{V}(A \wedge \neg B)$ is unsatisfiable. Therefore, for quantifier-free GAS and backward reachability algorithms one can use satisfiability-checking for quantifier-free formulas also for entailment-checking.

In the case of simple GAS the situation is not much more complicated. Suppose that we would like to check an entailment problem of the form $\forall\mathcal{V}(A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \wedge \ldots \wedge B_m)$. This problem is equivalent to unsatisfiability of the formula $\exists\mathcal{V}(A_1 \wedge \ldots \wedge A_n \wedge (\neg B_1 \vee \ldots \vee \neg B_m))$. This formula is equivalent to

$$\begin{array}{cc} \exists\mathcal{V}(A_1 \wedge \ldots \wedge A_n \wedge \neg B_1) & \vee \\ \cdots & \vee \\ \exists\mathcal{V}(A_1 \wedge \ldots \wedge A_n \wedge \neg B_m). & \end{array}$$

Therefore, to check the original entailment problem, one has to check m satisfiability problems for the formulas $A_1 \wedge \ldots \wedge A_n \wedge \neg B_i$. These formulas are not simple constraints any more, but can be made into simple constraints or disjunctions of simple constraints if the first-order theory of the structure $\mathbb{M}$ has the following property: the negation of any atomic formula is effectively equivalent to a disjunction of atomic formulas. The first-order theory of $\mathbb{I}$ has this property, for example the formula $\neg x = y$ is equivalent to $x > y \vee y > x$.

This observation shows that in many situations it is enough to implement satisfiability-checking for simple constraints in order to implement backward reachability. For example, for $\mathbb{I}$ it is enough to implement algorithms for solving systems of linear equations and inequations.

One general obstacle for efficiency of reachability algorithms is the problem of *accumulated variables*, i.e., existentially quantified variables introduced by applications of the corresponding predicate transformers. We have shown that for quantifier-free GAS and backward reachability algorithms no new variables are accumulated. For forward reachability algorithms, GAS still have an advantage of (normally) accumulating only a small number of extra variables. Indeed, consider a guarded assignment u of the form

$$P(v_1, \ldots, v_k) \Rightarrow v_1 \ := \ t_1, \ldots, v_n \ := \ t_n.$$

and assume that $\mathcal{V} = \{v_1, \ldots, v_n, v_{n+1}, \ldots, v_k\}$. Denote by $t'_1, \ldots, t'_n$ the terms obtained from $t_1, \ldots, t_n$ by replacing each variable v_i for $i \in \{1, \ldots, n\}$ by v'_i (note that the variables $v_{n+1}, \ldots, v_k$ are not replaced). It is not hard to argue that $A^u(v_1, \ldots, v_k)$ is equivalent to

$$\exists v'_1 \ldots \exists v'_n (A(v'_1, \ldots, v'_n, v_{n+1}, \ldots, v_k) \wedge P(v'_1, \ldots, v'_n, v_{n+1}, \ldots, v_k) \wedge v_1 = t'_1 \wedge \ldots \wedge v_n = t'_n).$$

In this formula the new quantifiers bind the variables $v'_1, \ldots, v'_n$, i.e., only those variables whose values are changed by the transition. In transition systems formalizing protocols (e.g., from [13]) the number of variables whose values are changed by the transition is usually small compared to the overall number of state variables.

In some cases accumulated existentially quantified variables are not the only source of complexity. For example, for finite domains the number of variables is usually large, and repeated substitutions of terms t_i for v_i make the formula $\mathsf{current}$ grow exponentially. One way to make $\mathsf{current}$ shorter is to introduce extra existentially quantified variables to name common subexpressions, in which case the formulas $\mathsf{current}$ will not be quantifier-free even for simple GAS.

6 Other issues

In this section we discuss some other issues related to our formalization of transition systems. There are still many topics not covered here, such as fairness constraints, or concurrent applications of transitions.

6.1 Widening

When the reachability algorithms diverge, one can apply technique known as *approximation* or *widening*. This technique can be formalized as follows. Instead of using an equivalence-preserving simplification function simp we use a *widening function* wide, such that $\mathbb{M} \models \forall \mathcal{V}(A \rightarrow \mathsf{wide}(A))$. As a result, the set of states computed by the reachability algorithms in the limit, is a superset of the set of reachable states. The algorithms using widening are not semi-decision procedures and correct only if they return "unreachable". Widening can be used to make diverging algorithms converge in some cases, but also to make the formulas $\mathsf{current}$ simpler.

6.2 Transitive closure

Transitive closure is a technique used by [9] to make the reachability analysis converge. The idea is to try to use a transitive closures of some transitions instead of the transitions themselves. For example, the transition

$$v_1 > 0 \Rightarrow v_1 \ := \ v_1 - 1, v_2 \ := \ v_2 + 1$$

over $\mathbb{I}$ can be replaced by its transitive closure

$$v_1 > 0 \wedge v'_1 < v_1 \wedge v'_1 + v'_2 = v_1 + v_2.$$

The use of transitive closure can make otherwise non-terminating reachability analysis terminate. Unlike widening, it does not change the set of computed reachable states, so it is sound and semi-complete in the same sense as the other algorithms presented here. However, transitive closure applied to a GAS gives a system which is not a GAS, and hence our results on GAS are not applicable any more.

6.3 Relaxation

The system DMC [14] uses the technique known in integer programming as *relaxation*. Instead of solving the reachability problem over the integers, one solves "the same" reachability problem over the reals. Similarly, [13] analyzes integer reachability problems using the Hy-Tech model checker in which the state variables are real-valued [17]. Relaxation is attractive since solving satisfiability and entailment problems over the reals is easier than solving these problems over the integers. For example, satisfiability and entailment of simple constraints can be solved in polynomial time over the reals but are respectively NP-complete and coNP-complete over the integers. In this section we formalize relaxation in a general setting.

Let $\mathbb{M}_1$ and $\mathbb{M}_2$ be two structures of the same signature Σ over the domains $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively, and ϵ be an *isomorphic embedding* of $\mathbb{M}_1$ into $\mathbb{M}_2$, i.e., a function from $\mathcal{D}_1$ to $\mathcal{D}_2$ with the following properties

1. For every predicate symbol P of arity n of the signature Σ and valuation s for a set of variables $\{x_1, \ldots, x_n\}$ in $\mathbb{M}_1$ we have $\mathbb{M}_1, s \vDash P(x_1, \ldots, x_n)$ if and only if $\mathbb{M}_2, \epsilon \circ s \vDash P(x_1, \ldots, x_n)$.
2. For every function symbol P of arity n of the signature Σ and valuation s for a set of variables $\{x, x_1, \ldots, x_n\}$ in $\mathbb{M}_1$ we have $\mathbb{M}_1, s \vDash x = f(x_1, \ldots, x_n)$ if and only if $\mathbb{M}_2, \epsilon \circ s \vDash x = f(x_1, \ldots, x_n)$.

These properties guarantee that for any quantifier-free formula A and valuation s for its variables in $\mathbb{M}_1$ we have $\mathbb{M}_1, s \vDash A$ if and only if $\mathbb{M}_2, s \vDash A$.

DEFINITION 25 (Relaxation) Let $R = (Tr, In, Fin)$ be an instance of the reachability problem over a structure $\mathbb{M}_1$ and there exists an isomorphic embedding of $\mathbb{M}_1$ into a structure $\mathbb{M}_2$. The instance (Tr, In, Fin) of the reachability problem over $\mathbb{M}_2$ is called the *relaxation* of R. ❏

That is, the original problem and its relaxation use the same formulas but are considered over different structures.

THEOREM 26 *Let $R_1 = (Tr, In, Fin)$ be an instance of the reachability problem over $\mathbb{M}_1$ and R_2 be its relaxation over $\mathbb{M}_2$. Suppose that all formulas in (Tr, In, Fin) are equivalent to existential formulas in $\mathbb{M}_1$. If R_2 does not hold, then R_1 does not hold too.* ❏

This theorem means that one can solve relaxation instead of the original problem to show non-reachability as soon as the initial and final conditions and the transition formula are existential or equivalent to existential formulas in $\mathbb{M}_1$. All special classes of transition systems (for example, positive existential) introduced in this paper have this property. One should be careful about using relaxation when properties other than reachability are checked. For example, if guards in a GAS are existentially quantified formulas, than one cannot in general check deadlock-freedom using relaxation. As stated in Theorem 14, the deadlock formula for GAS is $\neg(P_1 \vee \ldots \vee P_n)$, which may not be equivalent to an existential formula, as soon as at least one of the P_i is existentially quantified.

6.4 Other temporal formulas

As observed in many papers, a possibility to build fixpoints implies that properties more complex than reachability could be verified. It is not hard to argue that the following property, related to the fixpoint construction, is true for all of the algorithms of this paper.

THEOREM 27 *If the forward reachability algorithm terminates with the answer "unreachable" then* current *symbolically represents the set of all states reachable from initial states. If the backward reachability algorithm terminates with the answer "unreachable" then* current *symbolically represents the set of all states backward reachable from the final states.* ❏

This property allows one to extend the reachability algorithms and some results to more general classes of temporal formulas than those expressing reachability, but the analysis of such extended algorithms is beyond the scope of this paper.

7 Related work

[29] formalizes model checking procedures as constraint solving using a rather general framework. He treats a class of properties more general than reachability. An approach to formalizing reachability for hybrid systems using first-order logic is presented by [22]. They also note that one can use arbitrary first-order formulas (over reals) due to quantifier elimination. [18] present a classification of state transition systems, but over the reals.

Our notion of guarded assignment systems is similar to the action systems of [4], to UNITY [10], and to the programs of [3]. In particular, [3] prove some programs of programs related to monotonicity of certain operations, which are then used to prove termination. Other similar formalisms may be found in [26, 25, 23, 27].

Non-local reachability procedures were already formulated in [15, 31, 11], a local reachability procedure in [1].

[20] present a general approach to symbolic model checking of infinite-state systems, but based on a single language rather than different languages for different structures.

[6] show that the use of relaxation is complete for Petri nets. [13] proves a similar result for broadcast protocols. It would be interesting to find general conditions on relaxation to be complete which would imply these results.

Widening, or approximations, are studied in, e.g., [19, 12] and many other papers, too numerous to be mentioned here. There are also are many papers on (un)decidability results for infinite-state systems, both over integers and other structures, not mentioned here.

Acknowledgments

We thank Howard Barringer, Giorgio Delzanno, and Andreas Podelski for their comments on earlier versions of this paper.

References

1. P.A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS'96)*, pages 313–321, New Brunswick, NJ, July 1996. IEEE Computer Society Press.
2. P.A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1-2):109–127, January 2000.
3. P.A. Abdulla and B. Jonsson. Ensuring completeness of symbolic verification methods for infinite-state systems. *Theoretical Computer Science*, 256:145–167, 2001.
4. R.-J. Back and R. Kurki-Suonio. Distributed cooperation with action systems. *ACM Transactions on Programming Languages and Systems*, 10(4):513–554, 1988.
5. A.P. Beltyukov. Decidability of the universal theory of natural numbers with addition and divisibility (in Russian). *Zapiski Nauchnyh Seminarov LOMI*, 60:15–28, 1976. English translation in: Journal of Soviet Mathematics.
6. B. Bérard and L. Fribourg. Reachability analysis of (timed) Petri nets using real arithmetic. In J.C.M. Baeten and S. Mauw, editors, *CONCUR'99: Concurrency Theory, 10th International Conference*, volume 1664 of *Lecture Notes in Computer Science*, pages 178–193. Springer Verlag, 1999.
7. N.S. Bjørner. *Integrating Decision Procedures for Temporal Verification*. PhD thesis, Computer Science Department, Stanford University, 1998.
8. N.S. Bjørner. Reactive verification with queues. In *ARO/ONR/NSF/DARPA Workshop on Engineering Automation for Computer-Based Systems*, pages 1–8, Carmel, CA, 1998.
9. T. Bultan, R. Gerber, and W. Pugh. Model-checking concurrent systems with unbounded integer variables: symbolic representations, approximations, and experimental results. *ACM Transactions on Programming Languages and Systems*, 21(4):747–789, 1999.
10. K.M. Chandy and J. Misra. *Parallel Program Design: a Foundation*. Addison-Wesley Publishing Co., 1988.
11. E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.
12. T. Dang and O. Maler. Reachability verification for hybrid automata. In T.A. Henzinger, editor, *Hybrid Systems: Computation and Control. First International Workshop, HSCC'98*, volume 1386 of *Lecture Notes in Computer Science*, pages 96–109. Springer Verlag, 1998.
13. G. Delzanno. Automatic verification of parametrized cache coherence protocols. In A.E. Emerson and A.P. Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000*, volume 1855 of *Lecture Notes in Computer Science*, pages 53–68. Springer Verlag, 2000.

14. G. Delzanno and A. Podelski. Constraint-based deductive model checking. *International Journal on Software Tools for Technology Transfer*, 3(3):250–270, 2001.
15. E.A. Emerson and E.M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.
16. J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *14th Annual IEEE Symposium on Logic in Computer Science (LICS'99)*, pages 352–359, Trento, Italy, 1999. IEEE Computer Society.
17. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hy-Tech: a model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.
18. T.A. Henzinger and R. Majumdar. A classification of symbolic state transition systems. In H. Reichel and S. Tison, editors, *STACS 2000*, volume 1770 of *Lecture Notes in Computer Science*, pages 13–34. Springer Verlag, 2000.
19. T.A. Henzinger and V. Rusu. Reachability verification for hybrid automata. In T.A. Henzinger, editor, *Hybrid Systems: Computation and Control. First International Workshop, HSCC'98*, volume 1386 of *Lecture Notes in Computer Science*, pages 190–204. Springer Verlag, 1998.
20. Y. Kesten, O. Maler, M. Marcus, A. Pnueli, and E. Shahar. Symbolic model checking with rich assertional languages. *Theoretical Computer Science*, 256(1-2):93–112, 2001.
21. O. Kupferman and M. Vardi. Model checking of safety properties. *Formal Methods in System Design*, 19(3):291–314, 2001.
22. G. Lafferriere, G.J. Pappas, and S. Yovine. Symbolic reachability computation for families of linear vector fields. *Journal of Symbolic Computations*, 32(3):231–253, 2001.
23. L. Lamport. The temporal logic of actions. SRC Rep. 57, DEC, Palo Alto, 1991.
24. L. Lipshitz. The Diophantine problem for addition and divisibility. *Transactions of the American Mathematical Society*, 235:271–283, January 1978.
25. N.A. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151. ACM, August 1987.
26. Z. Manna and A. Pnueli. Adequate proof principles for invariance and liveness properties of concurrent programs. *Science of Computer Programming*, 4:257–289, 1984.
27. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer Verlag, 1992.
28. V.I. Mart'janov. Universal extended theories of integers. *Algebra i Logika*, 16(5):588–602, 1977.
29. A. Podelski. Model checking as constraint solving. In J. Palsberg, editor, *Static Analysis, 7th International Symposium, SAS 2000*, volume 1924 of *Lecture Notes in Computer Science*, pages 22–37. Springer Verlag, 2000.
30. W. Pugh. Counting solutions to Presburger formulas: how and why. *ACM SIGPLAN Notices*, 29(6):121–134, June 1994. Proceedings of the ACM SIGPLAN'94 Conference on Programming Languages Design and Implementation (PLDI).
31. J.P Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In M. Dezani-Ciancaglini and M. Montanari, editors, *International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer Verlag, 1982.
32. T. Rybina and A. Voronkov. Using canonical representations of solutions to speed up infinite-state model-checking. submitted, 2002.
33. T. Yavuz-Kahveci, M. Tuncer, and T. Bultan. A library for composite symbolic representations. In T. Margaria, editor, *Tools and Algorithms for Construction and Analysis of Systems, 7th International Conference, TACAS 2001*, volume 1384 of *Lecture Notes in Computer Science*, pages 52–66, Genova, Italy, 2001. Springer Verlag.