Global vs. Local Model Checking of Infinite State Systems

Tobias Schuele and Klaus Schneider
Reactive Systems Group
Department of Computer Science, University of Kaiserslautern
P.O. Box 3049, 67653 Kaiserslautern, Germany
{Tobias.Schuele,Klaus.Schneider}@informatik.uni-kl.de
http://rsg.informatik.uni-kl.de

Abstract. The verification of systems with infinite state spaces has attained considerable attention in recent years. In particular, both global and local model checking techniques have been applied to the verification of such systems. However, new symbolic representations like predicate logics and finite automata have only been used in global model checking. In this paper, we therefore propose a technique for local model checking that can benefit from these representations. Moreover, we compare both approaches and show that the classes of specifications which can be verified automatically are different.

1 Introduction

During the last decades, much effort has been spent on the development of automatic verification methods for systems with finite state spaces. In particular, symbolic model checking has received much attention with the introduction of binary decision diagrams (BDDs) [5]. This is due to the fact that BDDs are an efficient data structure for storing and manipulating large, but finite sets.

However, in many areas one has to deal with infinite state spaces. According to [11], there are at least the following five ‘sources of infinity’:

1. data structures over infinite domains: natural numbers, integers, etc.
2. control structures: unbounded call stacks or dynamic creation of processes
3. asynchronous communication: unbounded queues (FIFOs) for process communication
4. parameterization: infinite families of distributed systems
5. real-time constraints: timing constraints based on real-valued clocks

In the sequel, we concentrate on systems with a finite state control flow, but with data structures over infinite domains. Systems of this class appear in different disguises, e.g., as extended finite state machines [12], and belong to the most powerful machine models, since they are equivalent to Turing machines. They are frequently used in early phases of many design flows.

Regardless of whether finite or infinite state systems are considered, one generally distinguishes between two classes of model checking: global and local model checking [14]. Global model checking aims at computing all states where a given specification holds by means of fixpoint iterations. In contrast, local model checking directly answers the question whether a given set of states satisfies the specification. This is accomplished by constructing a proof tree using syntax directed decomposition rules. There are many differences between these two approaches. The most relevant one is that in global model checking, the syntax tree of a specification is traversed in a bottom-up manner, whereas in local model checking, a specification is evaluated top-down. A major advantage of the latter approach is that subformulas can be checked by need, i.e., in the spirit of lazy evaluation.

For finite state systems, termination of the model checking algorithms is guaranteed. Unfortunately, this does not hold for systems with infinite state spaces. Consequently, techniques are required...
to achieve termination, e.g., by using additional information such as invariants and well-founded orderings. As another problem, propositional logic and hence BDDs are naturally limited to the representation of finite sets and do not allow us to reason about systems with infinitely many states. Hence, we need more powerful representations that enable us to deal with infinite sets. To this end, Presburger arithmetic [13, 10] has been proposed which can be translated to finite automata to obtain efficient tools.

In this paper, we present the first local model checking procedure for infinite state systems based on Presburger arithmetic. Moreover, we propose a set of rules that allow us to verify certain specification indirectly where global model checking otherwise fails. Thus, we are able to compare these two approaches, in particular regarding termination. We will show by a number of examples that difficult problems can become trivial in local model checking, whereas they require non-obvious user interactions when global model checking is used.

Initial work on the verification of infinite state systems was done by Bradfield and Stirling in 1991 where the authors established proof rules for a tableau calculus [4]. However, they did not investigate how these rules can be used for the implementation of local model checking tools. Some years later, Bultan, Gerber, and Pugh proposed a method for global model checking of infinite state systems by means of Presburger arithmetic [6, 7]. In 1998, finite automata were used to represent infinite sets in reachability analysis [12]. In [20, 1], it was shown how Presburger arithmetic can be efficiently translated to finite automata. Using this representation, global model checking procedures for extensions of the $\mu$-calculus have been implemented [15] and applied to other applications like worst case execution time analysis [16]. A wealth of results about other types of infinite state systems has been published. For more detailed information, the reader is referred to [8, 19].

In the next section, we describe how infinite state systems can be symbolically represented by means of Presburger arithmetic and propose an extension of the $\mu$-calculus as a suitable specification logic. In Section 3, we briefly sketch how global model checking procedures work and present our approach to local model checking which is based on Bradfield and Stirling’s rules. Then, we compare these two approaches in Section 4. In particular, we consider the termination problem and show by simple examples, that global model checking may require user interactions to achieve termination, while local model checking can directly solve these problems. Finally, we conclude with a summary and directions for future work.

2 Representing Infinite State Systems by Presburger Arithmetic

In this section, we explain how Presburger arithmetic can be used for the representation of infinite state systems. In principle, one could use any other decidable logic for that purpose such as monadic second order logic (MSO). The syntax of Presburger formulas is defined as follows1:

**Definition 1 (Syntax of Presburger Arithmetic).** The set of Presburger formulas $\mathcal{PA}$ over a finite set of variables $V$ is defined as follows where $c \in \mathbb{Z}$, $v \in V$, and $\sim$ is one of the relations $\lt, \leq, =, \geq, \gt$:

$$
\begin{align*}
\text{Term} & : = c \mid v \mid \text{Term} + \text{Term} \mid \text{Term} - \text{Term} \mid c \cdot \text{Term} \\
\mathcal{PA} & : = \text{Term} \sim \text{Term} \mid \neg \mathcal{PA} \mid \mathcal{PA} \land \mathcal{PA} \mid \mathcal{PA} \lor \mathcal{PA} \mid \exists v. \mathcal{PA} \mid \forall v. \mathcal{PA}
\end{align*}
$$

For example, the formula $\exists y. (x = 2 \cdot y) \land x \geq 0$ is a Presburger formula that holds for every positive even number $x$. In general, we interpret Presburger formulas with variable assignments that map the free variables of a formula to integers. The set of assignments that satisfy a formula $\varphi \in \mathcal{PA}$ is denoted as $[\varphi]$. Note that the above definition already contains some syntactic sugar, and conversely, further operators can be defined easily like divisibility by a constant: $(c \mid \tau) := \exists x. \tau = c \cdot x$.

1 In contrast to the original definition, we interpret the logic over the integers rather than over the natural numbers.
An important aspect regarding the implementation of our tools is that every Presburger formula can be translated to a finite automaton that encodes its models. As there exists for every finite automaton an equivalent minimal one, automata can serve as canonical representations for Presburger formulas. This is very much in the same spirit as binary decision diagrams are used as canonical normal forms for propositional logic [5]. Hence, automata can be viewed as generalizations of BDDs for representing infinite sets. For more detailed information on the translation of Presburger arithmetic to finite automata, the reader is referred to [9, 2, 20, 15, 1, 16]. However, we do not presuppose that Presburger formulas are represented and manipulated by means of finite automata. Every other decision procedure, e.g., quantifier elimination [10], can be used as well to decide Presburger formulas.

Since our aim is to reason about reactive systems that have ongoing computations, we will concentrate on model checking and the $\mu$-calculus. To this end, we use an extension of the $\mu$–calculus where the atomic propositions are Presburger formulas.

**Definition 2 (Syntax of Presburger $\mu$-Calculus).** The set of Presburger $\mu$-calculus formulas $PA^\mu$ is defined as follows:

$$PA^\mu := PA \mid \neg PA^\mu \mid PA^\mu \land PA^\mu \mid PA^\mu \lor PA^\mu \mid \Diamond PA^\mu \mid \Box PA^\mu \mid \mu x.PA^\mu \mid \nu x.PA^\mu$$

The basic formulas of $PA^\mu$ are exactly the formulas of Presburger arithmetic. These formulas express particular stages of a system’s computation but cannot be used to reason about its temporal behavior. Hence, we added operators for successors and fixpoints in the above definition to obtain a temporal logic. As a result, $PA^\mu$ formulas are obtained by replacing variables in the propositional $\mu$–calculus by formulas of Presburger arithmetic. The semantics of $PA^\mu$ is given over integer Kripke structures, which are transition systems whose states are variable assignments.

**Definition 3 (Integer Kripke Structure (IKS)).** An integer Kripke Structure (IKS) over a finite set of variables $V$ is a transition system $K = (S, I, R)$ where $S$ is the possibly infinite set of states, $I \subseteq S$ are the initial states, and $R \subseteq S \times S$ is the transition relation. Every state $s \in S$ is a variable assignment $s : V \rightarrow \mathbb{Z}$. In addition, it is required that the set of initial states $I$ and the transition relation $R$ are definable in Presburger arithmetic.

According to the above definition, a state of an IKS is an assignment for the variables $V$. Regarding the modeling of a system, such an assignment describes the current values of the system’s variables. As the system proceeds with its execution, it changes some of the variables and therefore, we have a new assignment at the next point of time. We therefore represent the transition relation by a Presburger formula over the set of variables $V \cup V'$ where $V$ and $V'$ are the current and the next state variables, respectively. The formula that encodes a transition relation can be obtained by compiling high-level descriptions such as synchronous programs to Presburger arithmetic.

Global model checking procedures require to compute the existential and universal predecessors $\text{pre}_3^R(Q)$ and $\text{pre}_V^R(Q)$ of a set of states $Q \subseteq S$ with respect to a transition relation $R$. Local model checking procedures require to compute the successors of a set of states. These sets are defined as follows:

$$\text{pre}_3^R(Q) := \{ s \in S \mid \exists s' \in S. (s, s') \in R \land s' \in Q \}$$

$$\text{pre}_V^R(Q) := \{ s \in S \mid \forall s' \in S. (s, s') \in R \Rightarrow s' \in Q \}$$

$$\text{suc}_3^R(Q) := \{ s' \in S \mid s \in S. (s, s') \in R \land s' \in Q \}$$

$$\text{suc}_V^R(Q) := \{ s' \in S \mid \forall s \in S. (s, s') \in R \Rightarrow s \in Q \}$$

When using automata based representations of Presburger formulas, the above operations can be computed by intersection and projection [15, 16]. The semantics of Presburger $\mu$-calculus can then be recursively defined as the set of states that satisfy a formula.
Definition 4 (Semantics of Presburger $\mu$-Calculus). Given an IKS $\mathcal{K} = (S, I, R)$, we define the semantics of a formula $\Phi \in \text{PA}^\mu$ as follows:

- $[[\varphi]]_{\mathcal{K}} \equiv [[\varphi]] \cap S$ for any $\varphi \in \text{PA}$
- $[[\neg \varphi]]_{\mathcal{K}} := S \setminus [[\varphi]]_{\mathcal{K}}$
- $[[\varphi \land \psi]]_{\mathcal{K}} := [[\varphi]]_{\mathcal{K}} \cap [[\psi]]_{\mathcal{K}}$
- $[[\varphi \lor \psi]]_{\mathcal{K}} := [[\varphi]]_{\mathcal{K}} \cup [[\psi]]_{\mathcal{K}}$
- $[[\Diamond \varphi]]_{\mathcal{K}} := \text{pre}^R_K([[[\varphi]]_{\mathcal{K}}])$
- $[[\Box \varphi]]_{\mathcal{K}} := \text{pre}^C_K([[[\varphi]]_{\mathcal{K}}])$
- $[[\mu x. \varphi]]_{\mathcal{K}}$ is the least set $Q \subseteq S$ of states where $Q = [[\varphi]]_{K^Q}$ holds.
- $[[\nu x. \varphi]]_{\mathcal{K}}$ is the greatest set $Q \subseteq S$ of states where $Q = [[\varphi]]_{K^Q}$ holds.

$K^Q$ is the IKS that is obtained from $\mathcal{K}$ by changing the states $s$ of $\mathcal{K}$ such that exactly the states $s \in Q$ satisfy $x$. An IKS $\mathcal{K} = (S, I, R)$ satisfies a formula $\Phi$ if $I \subseteq [[\Phi]]_{\mathcal{K}}$ holds.

Although our base logic is decidable, this does not necessarily hold for the decidability of the model checking problem of $\text{PA}^\mu$. In fact, this is an undecidable problem [19].

Theorem 1 (Undecidability of $\text{PA}^\mu$ Model Checking). Checking whether a formula $\Phi \in \text{PA}^\mu$ holds in an IKS $\mathcal{K}$ is undecidable.

The proof is based on the fact that our descriptions are powerful enough to describe the halting problem for Turing machines which is well-known to be undecidable. As a consequence of theorem 1, there cannot exist algorithms for infinite state model checking that are able to prove or disprove every query. For this reason, it is important to have techniques that ensure termination for large classes of formulas. Such techniques can be classified into two categories: Automatic techniques that do not require user interaction and techniques that exploit user information such invariants and well-founded orderings. The latter are more powerful, but also more difficult to use in practice.

3 Global vs. Local Model Checking

As mentioned in the introduction, model checking algorithms can be classified into global and local approaches. Global model checking algorithms compute all states $[[\Phi]]_{\mathcal{K}}$ of a Kripke structure $\mathcal{K}$ where a formula $\Phi$ holds. To this end, the algorithms traverse the syntax tree of $\Phi$ in a bottom-up manner and attach to each subformula the corresponding set of states. In contrast, local model checking algorithms traverse the syntax tree in a top-down manner, and thereby check whether a subformula holds on a given set of states. In the following, we will describe both approaches.

3.1 Global Model Checking

Figure 1 shows the algorithm for global model checking. The boolean operations, i.e., negation, conjunction, and disjunction are computed by complementation, intersection, and union of the corresponding state sets. To compute the satisfying states for modal formulas $\Diamond \varphi$ and $\Box \varphi$, we have to compute the sets $\text{pre}^R_K([[[\varphi]]_{\mathcal{K}}])$ and $\text{pre}^C_K([[[\varphi]]_{\mathcal{K}}])$ (cf. Section 2). Finally, the satisfying states of fixpoint formulas are obtained by Tarski-Knaster iteration: For a least fixpoint formula $\mu x. f(x)$, we consider the iteration $Q_{i+1} := f(Q_i)$ starting with $Q_0 := \{\}$. The problem for model checking infinite state spaces is that we have $[[\mu x. f(x)]]_{\mathcal{K}} = \lim_{i \to \infty} Q_i$, but this fixpoint may never be reached. To achieve termination, it is therefore required to use special rules that we describe in more detail in Section 4.
3.2 Local Model Checking

Local model checking has been originally developed by Stirling and Walker [17, 18]. In [4, 3], Bradfield and Stirling proposed an extension for the verification of infinite state systems. All these procedures follow a top-down approach where the formula to be checked is successively decomposed into its subformulas. More precisely, local model checking procedures reduce queries \( s \vdash \varphi \) for a formula \( \varphi \) and a state (set) \( s \) by proof rules until axioms are finally reached. For finite state systems, this is advantageous if not the entire state space has to be considered. For infinite state spaces, this has the even more important aspect, that only a finite amount of it has to be considered.

In the following, we consider sequents \( \Phi \vdash \varphi \), where \( \Phi \) is a set of states represented by a Presburger formula and \( \varphi \in \text{PA}^\nu \) is the specification to be checked. A sequent \( \Phi \vdash \varphi \) holds iff every state \( s \in \Phi \) satisfies \( \varphi \), i.e., \( \Phi \subseteq \llbracket \varphi \rrbracket \). We may then use the rules given in Figure 2 to derive a proof tree for a sequent \( \Phi \vdash \varphi \). These are more or less Bradfield’s original rules [4, 3]. However, Bradfield was only interested in a sound and complete proof system and did neither consider the implementation nor the representation of the state sets.

Rules (1) and (3) follow the semantics of conjunctions and universal modal operators. Rule (2) is explained as follows: If \( \Phi \vdash \varphi \lor \psi \) holds, then the set of states encoded by \( \Phi \) can be partitioned into three classes: the states \( \psi_1 \) that satisfy \( \varphi \land \neg \psi \), the states \( \psi_2 \) that satisfy \( \neg \varphi \land \psi \), and the states \( \psi_3 \) that satisfy \( \varphi \land \psi \). We can then use \( \Phi_1 := \psi_1 \cup \psi_3 \) and \( \Phi_2 := \psi_2 \cup \psi_3 \) to see the correctness of the proof rule. Note that all subgoals are conjunctively connected, and that the third subgoal only requires a call to the decision procedure for Presburger arithmetic.

Rule (4) is more difficult: \( \Phi \vdash \Box \varphi \) holds iff every state \( s \in \Phi \) satisfies \( \Box \varphi \), i.e., iff every state \( s \in \Phi \) has at least one successor state \( s' \) that satisfies \( \varphi \). We may therefore select for every state \( s \in \Phi \) a successor state \( s' \) to obtain the state set \( \Phi' \), so that \( \Phi' \vdash \varphi \) holds. To see if our selection was correct, we have to check whether every state \( s \in \Phi \) has at least one successor state \( s' \in \Phi' \). This is ensured by the additional subgoal \( \Phi \subseteq \text{pre}^R(\Phi') \) that can be checked directly using the Presburger decision procedure. Rules (2) and (4) require user interaction: The user has to provide the system with appropriate formulas that represent sets of states \( \Phi_1, \Phi_2, \) and \( \Phi' \), respectively. The sequent holds
that have already been processed in a list avoid that the algorithm applies the unwinding rules infinitely often, it keeps track of all the queries and terminates and returns \( \Delta \). The reason for this end, the procedure maintains a so-called definition list \( \Delta \) that consists of pairs \((x, \mu x. \varphi)\) that associate a bound variable \(x\) with the subformula where it is bound. Given \((x, \sigma x. \varphi) \in \Delta\) holds, we write \(\Delta(x)\) for the second component of this pair.

Figure 3 shows our algorithm for local model checking that incorporates the above rules. To avoid that the algorithm applies the unwinding rules infinitely often, it keeps track of all the queries that have already been processed in a list \(\text{CallStack}\). As soon as a query reappears, the procedure terminates and returns 1 in the case of greatest fixpoint formulas. For least fixpoint formulas the algorithm has to check whether the entries on the call stack are well-founded [4, 3]. The reason for this is a deeply rooted property of the \(\mu\)-calculus that intuitively states that least fixpoints are related with finite recursion, while greatest fixpoints allow infinite recursion. Note further that the function \(\text{LoopTest}\) does not simply check whether a query already appears in \(\text{CallStack}\) [14].

### 4 Termination of Global and Local Model Checking Procedures

It follows from Theorem 1, that there is no procedure that could prove or disprove all queries for model checking Turing powerful infinite state systems. For this reason, it is particularly useful to have different approaches that can deal with different classes of queries. In this section, we will show by some examples, that the global and the local approaches indeed cover different classes of problems.

To this end, consider the algorithm shown in Figure 4. It adds two numbers \(a\) and \(b\) by successively incrementing \(a\) and decrementing \(b\) until \(b = 0\) holds. The variable \(s\) is an observer variable which is used to store the sum before the loop starts. As a precondition, we assume that \(b\) is positive since otherwise the algorithm does not terminate. Hence, for the set of initial states, we have \( I = \left[ b \geq 0 \land (a + b = s) \right] \). The following specifications are to be checked:

- \( S_1: \text{AG} \left( a + b = s \right) \)
- \( S_2: \text{AG} \left( b = 0 \rightarrow a = s \right) \)
- \( S_3: \text{AF} \left( b = 0 \right) \)

Specification \(S_1\) means that the sum of \(a\) and \(b\) does not change during the computation (loop invariant), \(S_2\) means that \(a\) is the sum when \(b = 0\) holds, and \(S_3\) means that for all inputs, \(b\) will finally become zero. Hence, proving \(S_3\) implies the termination of the algorithm. In the following, we consider how these specifications can be checked by the global and the local model checking procedures.

The following decomposition rules for local model checking:

1. \( \Phi \vdash \varphi \land \psi \)
   \[ \frac{}{\Phi \vdash \varphi \quad \Phi \vdash \psi} \]

2. \( \Phi \vdash \varphi \lor \psi \)
   \[ \frac{\Phi_1 \vdash \varphi \quad \Phi_2 \vdash \psi}{\Phi \subseteq \Phi_1 \cup \Phi_2} \]

3. \( \Phi \vdash \Box \varphi \)
   \[ \frac{\text{suc}_G(\Phi) \vdash \varphi}{\Phi \vdash \Box \varphi} \]

4. \( \Phi \vdash \Diamond \varphi \)
   \[ \frac{\Phi \vdash \Box \varphi \quad \Phi \subseteq \text{pre}_G(\Phi')}{\Phi' \vdash \Diamond \varphi} \]

5. \( \Phi \vdash \mu x. \varphi \)
   \[ \frac{\Phi \vdash \varphi}{\Delta := \Delta \cup \{(x, \mu x. \varphi)\}} \]

6. \( \Phi \vdash \nu x. \varphi \)
   \[ \frac{\Phi \vdash \varphi}{\Delta := \Delta \cup \{(x, \mu x. \varphi)\}} \]

7. \( \Phi \vdash x \)
   \[ \frac{\Phi \vdash \Delta(x)}{\Phi \vdash \Delta(x)} \]

8. \( \Phi \vdash \varphi \)
   \[ \frac{\Phi \subseteq [\varphi]}{\varphi \in \text{PA}} \]

Fig. 2. Decomposition Rules for Local Model Checking

\(\Phi\) iff such sets \(\Phi_1, \Phi_2, \text{and} \Phi'\) exist (completeness). Rules (5-7) simply unwind fixpoint formulas. To do this, the procedure maintains a so-called definition list \(\Delta\) that associates a bound variable \(x\) with the subformula where it is bound. Given \((x, \sigma x. \varphi) \in \Delta\) holds, we write \(\Delta(x)\) for the second component of this pair.

2 We assume that every bound variable is bound only once and has no additional free occurrences. Moreover, we assume that the formulas are given in guarded normal form [14], i.e., that every occurrence of a bound variable \(x\) in \(\mu x. \varphi\) is guarded by a modal operator. Guarded normal forms can be computed efficiently for every formula.

3 We use the following CTL-operators for the sake of brevity: \(\text{AF} \varphi := \mu x. \varphi \lor \Box x\) and \(\text{AG} \varphi := \nu x. \varphi \land \Box x\)
function LoopTest($V_\Psi$, $(\Phi_S, \sigma x.\varphi), CS)$
if $CS = []$ then return 0
else
  $(\Phi'_S, \sigma' x'.\varphi') := \text{HD}(CS)$;
  if $(\Phi'_S, \sigma' x'.\varphi') = (\Phi_S, \sigma x.\varphi)$ then return 1
  elseif $x' \in V_\Psi$ then return 0
  else return LoopTest($V_\Psi$, $(\Phi_S, \sigma x.\varphi), \text{TL}(CS)$)
end
end

function LocalCheck($\Phi_S, \Psi$)

  case $\Psi$ of
  isBoundVar($\Psi$): /* Rule (7) */
    $D_\Psi := \Delta(\Psi)$; $V_\Psi := \text{FreeVar}(D_\Psi)$;
    if LoopTest($V_\Psi$, $(\Phi_S, D_\Psi), \text{CallStack}$) then
      case $D_\Psi$ of
      $\mu x.\varphi$ : return CheckWellFound($\text{CallStack}$)
      $\nu x.\varphi$ : return 1
      end
    else return LocalCheck($\Phi_S, D_\Psi$)
  IsPres($\Psi$): /* Rule (8) */
    return CheckPresburger($\Phi_S \rightarrow \Psi$);
  $\varphi \land \psi$: /* Rule (1) */
    if LocalCheck($\Phi_S, \varphi$) then return LocalCheck($\Phi_S, \psi$)
    else return 0;
  $\varphi \lor \psi$: /* Rule (2) */
    $(\Phi_1, \Phi_2) := \text{ReadPartition}(\Phi_S, \varphi \lor \psi)$;
    if not(CheckPresburger($\Phi_S \rightarrow \Phi_1 \lor \Phi_2$)) then fail;
    if not(LocalCheck($\Phi_1, \varphi$)) then fail;
    return LocalCheck($\Phi_2, \psi$);
  $\lozenge \varphi$: /* Rule (4) */
    $\Phi' := \text{ReadPartition}(\Phi_S, \lozenge \varphi)$;
    if not(CheckPresburger($\Phi_S \rightarrow \text{pre}_R(\varphi)$)) then fail;
    return LocalCheck($\Phi', \varphi$);
  $\Box \varphi$: /* Rule (3) */
    return LocalCheck($\text{suc}_R(\Phi_S), \varphi$);
  $\sigma x.\varphi$: /* Rules (5) and (6) */
    \text{CallStack} := \text{CONS}((\Phi_S, \Psi), \text{CallStack});
    $\Delta := \Delta \cup \{(x, \sigma x.\varphi)\}$;
    $b := \text{LocalCheck}(\Phi_S, \varphi)$;
    \text{CallStack} := \text{TL}(\text{CallStack});
    return $b$
end
end

Fig. 3. Local Model Checking
module IncDecAdder:
  input a, b, s : Z;
  while b > 0 do
    a := a + 1;
    b := b - 1;
  end;
end;

Fig. 4. A simple adder

4.1 Global Model Checking

We tried to check the above specifications using our global model checker. Specification $S_1$ can be verified directly, i.e., without any user interaction. For the specifications $S_2$ and $S_3$, the fixpoint iterations do not terminate. In the following, we present a technique that enables us to prove greatest fixpoints and to disprove least fixpoints by means of over- and under-approximation, respectively. Using this technique, it is possible to prove $S_2$ with global model checking.

Let $K$ be an IKS and $\mu x.\varphi, \nu x.\psi$ be two formulas whose fixpoint computations do not terminate. Then, we can disprove $\mu x.\varphi$ and prove $\nu x.\psi$ using the following observations:

\[
K \not\models \mu x.\varphi \iff I \not\subseteq [\mu x.\varphi]_K \iff \exists \Phi. [\mu x.\varphi]_K \subseteq [\Phi]_K \land I \not\subseteq [\Phi]_K
\]

\[
K \models \nu x.\psi \iff I \subseteq [\nu x.\psi]_K \iff \exists \Psi. [\nu x.\psi]_K \subseteq [\Psi]_K \land I \subseteq [\Psi]_K
\]

The problem is that these rules cannot be applied in a straightforward way since the fixpoint computations of $[\mu x.\varphi]_K$ and $[\nu x.\psi]_K$ do not terminate. However, if one considers the Tarski-Knaster fixpoint theorem (see [14]), we have

\[
[[\varphi]]_x^\Phi \subseteq [[\Phi]]_K \Rightarrow [[\mu x.\varphi]]_K \subseteq [[\Phi]]_K \quad \text{and} \quad [[\psi]]_x^\Psi \subseteq [[\psi]]_K \Rightarrow [[\Psi]]_K \subseteq [[\nu x.\psi]]_K.
\]

This follows from the fact that the least fixpoint of a function $f : D \rightarrow D$ is not only the minimum of all fixpoints, but also the minimum of the prefixed points, i.e., those elements with $f(x) \subseteq x$. The converse holds for greatest fixpoints. Given two formulas $\Phi$ and $\Psi$, we hence have the following rules:

\[
([\varphi]]_x^\Phi) \subseteq ([\Phi])_K \land K \not\models \Phi \Rightarrow K \not\models \mu x.\varphi \quad \text{and} \quad ([\psi]]_x^\Psi) \subseteq ([\psi]]_K \land K \models \Psi \Rightarrow K \models \nu x.\psi.
\]

Now we consider how to prove specification $S_2$ for the example in Figure 4. $S_2$ is a greatest fixpoint, namely $\nu x.(b = 0 \rightarrow a = s) \land \Box x$. Hence, we use the rule for greatest fixpoints with $\Psi \equiv S_1$. Since we already proved that $S_1$ holds, it remains to show that $[[S_1]]_K \subseteq [[(b = 0 \rightarrow a = s) \land \Box S_1]]_K$ holds. To sum up, our rules enabled the global model checker to prove $S_2$ indirectly.

As mentioned previously, the fixpoint computation for specification $S_3$ does neither terminate. In this case, the above rules are of little help, since we can only disprove least fixpoint formulas. A solution to this problem requires other rules, e.g., rules based on well-founded orderings which are compatible with the algorithm’s transition relation.
4.2 Local Model Checking

In this section, we consider the verification of our example with the local model checking procedure. Figure 5 shows the proof tree where the initial query is called with the set of initial states $I = [b \geq 0 \land (a + b = s)]$. The procedure terminates since an infinite recursion is detected in vertex 4: A regeneration of $x$ would lead to a vertex with the same query as vertex 0. Hence, the return value is 1 and we have shown that the specification holds. As in the case of the global model checking procedure, it can be checked automatically without any user interaction.

![Fig. 5. Local model checking of the formula $AG\ (a + b = s) :\equiv \nu x.(a + b = s) \land \square x$](image)

For specification $S_2$ the situation is different. Consider the proof tree shown in Figure 6. At a first glance, one would apply rule (2) in vertex 2 to resolve the disjunction. However, the algorithm of Figure 3 first checks if the formula is a Presburger formula. If so, it can be evaluated directly by a call to the Presburger decision procedure using rule (8), i.e., without further decomposition. Again, we have an infinite recursion in vertex 4 and return 1. In contrast to global model checking, specification $S_2$ is checked without additional user interaction.

![Fig. 6. Local model checking of the formula $AG\ (b = 0 \rightarrow a = s) :\equiv \nu x.(b \neq 0 \lor a = s) \land \square x$](image)

Finally, the queries of the local model checking procedure for specification $S_3$ are given in Figure 7. Since the disjunction contains a modal operator, we have to partition the set of states $[b = 0 \land (a + b = s)]$ in vertex 1. It follows immediately from the specification that $\Phi_1 :\equiv [b = 0 \land (a + b = s)]$ and $\Phi_2 :\equiv [b > 0 \land (a + b = s)]$ is a suitable partition. The formula in vertex 4 is a pure Presburger formula and can hence be checked directly. In vertex 5, an infinite recursion is detected since the regeneration of $x$ rederives the pair that occurs in vertex 0. According to [3], this means that we have to check the $\mu$-success, i.e., we have to endow the set of numbers $b \geq 0$ with an ordering such that for a particular number $b \geq 0$ the changes made by the path from vertex 0 to 5 is well-founded.
\[ 0 : b \geq 0 \land (a + b = s) \vdash \mu x. b = 0 \lor \Box x \]

\[ 1 : b \geq 0 \land (a + b = s) \vdash b = 0 \lor \Box x \]

\[ 2 : b = 0 \land (a + b = s) \vdash b = 0 \]

\[ 3 : b > 0 \land (a + b = s) \vdash \Box x \]

\[ 4 : \Phi \subseteq \Phi_1 \cup \Phi_2 \]

\[ 5 : b \geq 0 \land (a + b = s) \vdash x \]

\[ \Phi :\equiv \llbracket b \geq 0 \land (a + b = s) \rrbracket \quad \Phi_1 :\equiv \llbracket b = 0 \land (a + b = s) \rrbracket \quad \Phi_2 :\equiv \llbracket b > 0 \land (a + b = s) \rrbracket \]

**Fig. 7.** Local model checking of the formula $AF(b = 0) :\equiv \mu x. (b = 0) \land \Box x$

### 5 Summary and Conclusion

In this paper, we have presented a method for local model checking of infinite state systems where the state sets are represented using Presburger arithmetic. It is surprising that this combination has not yet been exploited for the implementation of verification tools. Moreover, we proposed a set of rules for global model checking that can sometimes be used to prove specifications when the fixpoint computations do not terminate.

We have shown by a small example, that the classes of problems that can be solved by these two approaches without interaction, are not the same. In particular, the local model checking procedure is able to prove two of the given specifications automatically and one with little interaction. In contrast, the global model checking procedure can only solve the first one without further help. Using the proposed rules, it is possible to prove the second specification, but not the third one, which requires more sophisticated user interaction.

### References


