Widening Arithmetic Automata*

Constantinos Bartzis and Tevfik Bultan

Department of Computer Science
University of California
Santa Barbara CA 93106, USA
{bar,bultan}@cs.ucsb.edu

Abstract. Model checking of infinite state systems is undecidable, therefore, there are instances for which fixpoint computations used in infinite state model checkers do not converge. Given a widening operator one can compute an upper approximation of a least fixpoint in finite number of steps even if the least fixpoint is uncomputable. We present a widening operator for automata encoding integer sets. We show how widening can be used to verify safety properties that cannot be verified otherwise. We also show that the dual of the widening operator can be used to detect counter examples for liveness properties. Finally, we show experimentally how the same technique can be used to verify properties of complex infinite state systems efficiently.

1 Introduction

Symbolic verification of large and complex infinite state systems may require an unreasonable number of fixpoint iterations. Furthermore, since the problem of verification of temporal properties of infinite state systems is in general undecidable, the fixpoint computations might not converge at all. To overcome this problem one can use approximations. Abstract interpretation framework [9] provides a technique known as widening, to compute a least fixpoint's upper bound in finite time. Widening has been successfully applied to Polyhedra based verification of systems specified with arithmetic constraints [10, 13, 7]. On the other hand, similar work for the automata encoding of arithmetic constraints has been limited. We present a widening operator for automata encoding of integer sets as described in [3]. We also show how to verify properties of infinite state systems using an approximate fixpoint computation based on our widening technique. Note that, for these properties the exact fixpoint computation does not converge. Finally, we show experimentally how the same technique can be used to improve the efficiency of our infinite state model checking tool, Action Language Verifier (ALV) [8], and compare its performance with BRAIN [15].

Most reachability properties can be formulated as least fixpoints over sets of states. If the state space is infinite, these fixpoints may not be computable. Widening is a well known technique [9, 10] that facilitates the convergence of a fixpoint computation by extrapolating an upper approximation of the exact least

* This work is supported by NSF grants CCR-9984822 and CCR-0341365.

R. Alur and D.A. Peled (Eds.): CAV 2004, LNCS 3114, pp. 321–333, 2004.

fixpoint. In [10, 12] a widening operator was defined for systems whose transition relation and sets of states can be described by linear arithmetic constraints, symbolically represented as sets of convex Polyhedra. This technique has been successfully used in the analysis of various types of systems such as concurrent systems (by extending it to Presburger arithmetic formulas [7]), synchronous programs and linear hybrid systems [13].

Systems described by Presburger arithmetic formulas can also be symbolically represented by finite automata [6, 16]. Experimental results in [3], indicate that the automata representation often outperforms the polyhedral representation. However, until lately the use of approximation techniques for the automata representation has been limited. In a series of recent papers [14, 5] "Regular Model Checking" (RMC) has been defined as a framework for algorithmic verification of systems with transition relations represented by a regular length-preserving relation on strings. Typical examples of such systems are linear parameterized networks of processes. Widening techniques have been used to compute the set of reachable configurations of such systems in finite time. However, the arithmetic relations considered in [14, 5] are restricted because of the unary encoding used (only addition of constants is allowed). Our goal is to develop a widening technique for automata representing Presburger formulas.

Another way to deal with non-termination of exact infinite state model checking is to compute the reflexive-transitive closure R^* of the transition relation R of the system (or an upper approximation of R^*). Given R^*, one can compute the set of states forward or backward reachable from an initial set of states I with a single image computation. Therefore computing R^* is at least as hard as computing sets of reachable states. In fact, it may be the case that the set of states reachable from I is regular but R^* is not. Nevertheless, ways to approximate R^* for regular transition relations R have been studied in [5, 11, 4]. In particular, in [4] the authors present a generic technique for computing R^* (sometimes precisely) for transition relations representing arithmetic relations. The technique is generic in the sense that it can handle relations that are not in a restricted form. Our technique is also generic but is based on widening instead of iterating relations. As we show in Section 4 our approach can verify some properties of systems with non-regular R^*. The algorithm in [4] is complicated and involves determinization of automata which is potentially an expensive operation even when heuristics are used to improve efficiency. There is no full implementation of this algorithm that allows it to be used for verification applications. As we show in Section 5, our widening technique can be used to verify properties of complex systems efficiently. Finally, note that one can use our widening technique to iterate transducers but the opposite is not true.

The rest of the paper is organized as follows. First we discuss fixpoint computations and how widening technique can be used to help them converge in Section 2. Next, we briefly present the automata representation we are using for integer sets satisfying Presburger formulas in Section 3. In Section 4 we formally define our widening operator for arithmetic automata, prove some interesting properties and illustrate how it can be applied to a set of characteristic systems. Finally in Section 5 we present experimental results that demonstrate how our widening technique can be used to verify properties of complex systems efficiently.

2 Fixpoint Computations and Widening

We consider systems whose states can be described by the values of v integer variables $x_1, \ldots, x_v$. A set of states of the system, $S \subseteq \mathbb{Z}^v$, is a relation on the v integer variables. The transition relation of the system, $R \subseteq \mathbb{Z}^{2v}$, is a relation on the current state and next state variables $x_1, \ldots, x_v, x'_1, \ldots, x'_v$. In particular, we consider systems where S and R can be represented as Presburger arithmetic formulas, i.e., $S = \{(x_1, \ldots, x_v) \mid \phi_S(x_1, \ldots, x_v)\}$ and $R = \{(x_1, \ldots, x_v, x'_1, \ldots, x'_v) \mid \phi_R(x_1, \ldots, x_v, x'_1, \ldots, x'_v)\}$, where ϕ_S and ϕ_R are Presburger arithmetic formulas. In Section 3, we show how to represent S and R symbolically using finite automata.

We define the pre-condition of S with respect to R, $\mathit{pre}(S, R) \subseteq \mathbb{Z}^v$, as the set of states that can reach some state in S in one step. Similarly we define $\mathit{post}(S, R) \subseteq \mathbb{Z}^v$ as the set of states reachable from some state in S in one step. One can compute $\mathit{pre}(S, R)$ and $\mathit{post}(S, R)$ as follows

$$\mathit{pre}(S, R) = \{(x_1, \ldots, x_v) \mid \exists x'_1 ... \exists x'_v . (\phi_{S[x_1 \leftarrow x'_1, \ldots, x_v \leftarrow x'_v]} \wedge \phi_R)$$

$$\mathit{post}(S, R) = \{(x_1, \ldots, x_v) \mid (\exists x_1 ... \exists x_v . \phi_S \wedge \phi_R)_{[x'_1 \leftarrow x_1, \ldots, x'_v \leftarrow x_v]}\}.$$

where $\psi_{[y \leftarrow z]}$ is the formula generated by substituting z for y in ψ. Hence, to compute $\mathit{pre}(S, R)$ and $\mathit{post}(S, R)$ we need to be able to compute three operations: conjunction, existential variable elimination and renaming. In Section 3, we will show that these operations can be implemented using an automata representation for R and S.

We can formulate the verification problem of invariants based on pre- and post-condition functions as follows. We are given a set of initial states I, a transition relation R, pre- and post-condition functions $\mathit{pre}(S, R)$ and $\mathit{post}(S, R)$, and a property P. To verify the property we have two alternatives. The first is to compute $FR(I)$, the set of states forward reachable from the initial states I, and then check whether

$$FR(I) \subseteq P \qquad (1)$$

The second way is to compute $BR(\neg P)$ the set of states backward reachable from the negation of the property and then check that

$$BR(\neg P) \cap I = \emptyset \qquad (2)$$

Since the problem is undecidable, we might not be able to compute $FR(I)$ or $BR(\neg P)$ exactly. In that case we can follow a conservative approach by replacing $FR(I)$ and $BR(\neg P)$ by over approximations $FR(I)^+ \supseteq FR(I)$ and $BR(\neg P)^+ \supseteq BR(\neg P)$ in equations 1 and 2 respectively. Note that, we may not be able to verify a property that actually holds when we use approximations. Below we describe how to compute $FR(I)$ and $FR(I)^+$. Computation of $BR(\neg P)$ and $BR(\neg P)^+$ is similar.

The set $FR(I)$ of reachable states from I is the least fixpoint of the functional $\lambda X\ .\ I \cup post(X, R)$. This fixpoint is the limit of the sequence $S_0, S_1, \ldots$, where $S_0 = I$ and $S_{i+1} = S_i \cup post(S_i, R)$. This sequence may not converge. However we can compute an over approximation $FR(I)^+$ which is the limit of a new sequence $S'_0, S'_1, \ldots$, such that for each i, $S_i \subseteq S'_i$ and the sequence $S'_0, S'_1, \ldots$ converges after a finite number of iterations. We compute the S'_is by using a widening operator ∇, which satisfies the following property:

$$\text{Given two sets } A \text{ and } B,\ A \cup B \subseteq A \nabla B. \tag{3}$$

Now we can define S'_i as:

$$S'_i = \begin{cases} S_i & \text{if } 0 \le i \le s \\ S'_{i-1} \nabla (S'_{i-1} \cup post(S'_{i-1}, R)) & \text{if } i > s \end{cases}$$

where s is the seed of the widening sequence. Experiments show that higher seeds are likely to result in better approximations. The goal is to find a widening operator such that the sequence $S'_0, S'_1, \ldots$ converges as fast as possible to a fixpoint that is as close as possible to the exact set of reachable states. We present a widening operator for automata representing integer sets.

3 Automata Representation for Integer Sets

The representation of Presburger formulas by finite automata has been studied in [6, 16, 3]. Here we briefly describe finite automata that accept the set of natural number tuples that satisfy a Presburger arithmetic formula on v variables. The representation we discuss below can be extended to integers using 2's complement arithmetic [3]. We present the construction for natural numbers to simplify the presentation. Our implementation of widening technique and our verification tool also handles negative integers.

We encode numbers using their binary representation. A v-tuple of natural numbers $(n_1, n_2, ..., n_v)$ is encoded as a word over the alphabet $\{0, 1\}^v$, where the i_{th} letter in the word is $(b_{i1}, b_{i2}, ..., b_{iv})$ and b_{ij} is the i_{th} least significant bit of number n_j. Given a Presburger formula ϕ, we construct a finite automaton FA(ϕ)=$(K, \Sigma, \delta, e, F)$ that accepts the language L(ϕ) over the alphabet $\Sigma = \{0, 1\}^v$, which contains all the encodings of the natural number tuples that satisfy the formula. K is the set of automaton states, Σ is the input alphabet, $\delta : K \times \Sigma \to K$ is the transition function, $e \in K$ is the initial state, and $F \subseteq K$ is the set of final or accepting states.

For equalities, FA($\sum_{i=1}^{v} a_i \cdot x_i = c$) = $(K, \Sigma, \delta, e, F)$, where

$$K = \{k \mid \sum_{a_i<0} a_i \le k \le \sum_{a_i>0} a_i \vee 0 \le k \le -c \vee -c \le k \le 0\} \cup \{sink\},$$

$$\Sigma = \{0, 1\}^v, \quad e = -c, \quad F = \{0\},$$

$$\delta(k, (b_1, ..., b_v)) = \begin{cases} (k + \sum_{i=1}^{v} a_i \cdot b_i)/2 & \text{if } k + \sum_{i=1}^{v} a_i \cdot b_i \text{ is even and } k \ne sink \\ sink & \text{otherwise} \end{cases}$$

For inequalities, $\text{FA}(\sum_{i=1}^{v} a_i \cdot x_i < c) = (K, \Sigma, \delta, e, F)$, where

$$K = \{k \mid \sum_{a_i<0} a_i \leq k \leq \sum_{a_i>0} a_i \vee 0 \leq k \leq -c \vee -c \leq k \leq 0\}$$

$$\Sigma = \{0,1\}^v, \quad e = -c, \quad F = \{k \mid k \in K \wedge k < 0\},$$

$$\delta(k, (b_1, ..., b_v)) = \lfloor (k + \sum_{i=1}^{v} a_i \cdot b_i)/2 \rfloor.$$

Moreover, conjunction, disjunction and negation of constraints can be implemented by automata intersection, union and complementation, respectively. Finally, if some variable is existentially quantified, we can compute a non-deterministic FA accepting the projection of the initial FA on the remaining variables and then determinize it. The resulting FA may not accept *all* satisfying encodings (with any number of leading zeros). We can overcome this by recursively identifying all rejecting states k such that $\delta(k, (0, 0, ..., 0)) \in F$, and make them accepting. Universal quantification can be similarly implemented by the use of the FA complementation.

4 Widening Arithmetic Automata

Before formally defining a widening operator for arithmetic automata we briefly describe the intuition behind it. Let A and A' be two automata representing two consecutive members of a sequence $A_0, A_1, \ldots$, whose limit A_∞ is the exact least fixpoint we are trying to compute. Since A_∞ is the union of all A_is, it can be seen as a product automaton with each state being a tuple (of possibly infinite size) containing a state from each A_i. First, consider a string w and assume that after consuming w A, A' and A_∞ move to state k, k' and k_∞ respectively. Then k_∞ contains k and k'. Second, consider states k and k' of A and A' respectively, such that the languages accepted from k and k' are the same. Then again there exists a state k_∞ of A_∞ that contains both k and k'. For either scenario, our widening method, given A and A' as an input, produces an automaton that accepts both languages of A and A' and in which k and k' are merged in a single state.

Given two finite automata $A = (K, \Sigma, \delta, e, F)$ and $A' = (K', \Sigma, \delta', e', F')$ we define the binary relation $\equiv_\nabla$ on $K \cup K'$ as follows. Given $k \in K$ and $k' \in K'$, we say that $k \equiv_\nabla k'$ and $k' \equiv_\nabla k$ if and only if

$$\forall w \in \Sigma^*.\ \delta^*(k, w) \in F \Leftrightarrow \delta'^*(k', w) \in F' \tag{4}$$

$$\text{or} \quad k, k' \neq \mathit{sink} \wedge \exists w \in \Sigma^*.\ \delta^*(e, w) = k \wedge \delta'^*(e', w) = k', \tag{5}$$

where $\delta^*(k, w)$ is defined as the state A reaches after consuming w starting from state k. In other words, condition 4 states that $k \equiv_\nabla k'$ if the languages accepted by A from k and by A' from k' are the same. Condition 5 states that $k \equiv_\nabla k'$ if for some word w, A ends up in state k and A' ends up in state k' after consuming w. For $k_1 \in K$ and $k_2 \in K$ we say that $k_1 \equiv_\nabla k_2$ if and only if

$$\exists k' \in K'.\ k_1 \equiv_\nabla k' \wedge k_2 \equiv_\nabla k' \quad \vee \quad \exists k \in K.\ k_1 \equiv_\nabla k \wedge k_2 \equiv_\nabla k \tag{6}$$

Similarly we can define $k'_1 \equiv_\nabla k'_2$ for $k'_1 \in K'$ and $k'_2 \in K'$.

It is easy to prove that $\equiv_\nabla$ is an equivalence relation. Call C the set of equivalence classes of $\equiv_\nabla$. We define $A \nabla A' = (K'', \Sigma, \delta'', e'', F'')$ by:

$$\begin{aligned}
K'' &= C \\
\delta''(c_i, \sigma) &= c_j \;\text{ s.t. }\; (\forall k \in c_i \cap K.\ \delta(k,\sigma) \in c_j \vee \delta(k,\sigma) = sink) \wedge \\
&\qquad\qquad (\forall k' \in c_i \cap K'.\ \delta'(k',\sigma) \in c_j \vee \delta'(k',\sigma) = sink) \\
e'' &= c \;\text{ s.t. }\; e \in c \wedge e' \in c \\
F'' &= \{c_1, c_2, ..., c_n\} \;\text{ s.t. }\; \forall c_i, \exists k \in F \cup F'.\ k \in c_i
\end{aligned}$$

In other words, the set of states of $A \nabla A'$ is the set C of equivalence classes of $\equiv_\nabla$. Transitions are defined from the transitions of A and A'. The initial state is the class containing the initial states e and e'. The set of final states is the set of classes that contain some of the final states in F and F'. The following Theorem states that ∇ satisfies condition (3).

Definition 1. *Given an automaton $A = (K, \Sigma, \delta, e, F)$ and a state $k \in K$, we define $L(k)$ to be the language accepted by the automaton $(K, \Sigma, \delta, k, F)$. Also $L(A) = L(e)$.*

Theorem 1. *Given two automata A and A', $L(A) \cup L(A') \subseteq L(A \nabla A')$.*

Proof. Essentially we want to prove that given $w \in \Sigma^*$ such that w is accepted by A or A' then w is also accepted by $A \nabla A'$. Without loss of generality we may assume that w is accepted by A. Let $w = \sigma_0 \sigma_1 \ldots \sigma_n$. Then there is a sequence of non-sink states $k_0, k_1, \ldots, k_{n+1}$ such that $k_0 = e$, $\delta(k_i, \sigma_i) = k_{i+1}$ and $k_{n+1} \in F$. From the definition of $A \nabla A' = (K'', \Sigma, \delta'', e'', F'')$ it follows that there exists a sequence $c_0, c_1, \ldots, c_{n+1}$ such that $c_0 = e''$, $\delta''(c_i, \sigma_i) = c_{i+1}$ and $k_i \in c_i$ for all $0 \leq i \leq n+1$. Since $k_{n+1} \in c_{n+1}$ and $k_{n+1} \in F$ it follows that $c_{n+1} \in F''$ and thus w is accepted by $A \nabla A'$.

According to the original definition [9], a widening operator has to guarantee convergence. Our widening operator does not guarantee convergence. Nevertheless, we can force it to converge by a slight modification. If we discard the condition $k, k' \neq sink$ from equation 5, for each state in one automaton there exists an equivalent state in the other. Thus, the produced automaton has at most as many states as the smaller operand. As a result, the automata in the sequence can not increase in size. There are finitely many automata with a given number of states and a fixed alphabet. On the other hand, the size of the set of states represented by the automata in the sequence is monotonically increasing, otherwise we would have reached a fixpoint. Consequently, the sequence will converge. One can deploy this technique when the number of iterations has become too high. However, we decided not to use it in our implementation (see Section 5), because in most of the cases this modification in the widening operator makes the approximation too coarse to prove any property.

On the other hand, in the sequel we show that for a class of systems, if the approximate computation converges, it computes the exact set of reachable states.

Definition 2. *An automaton $A_1 = (K_1, \Sigma, \delta_1, e_1, F_1)$ is called weakly equivalent to automaton $A_2 = (K_2, \Sigma, \delta_2, e_2, F_2)$ iff there exists a total function $f : K_1 \setminus \{sink\} \to K_2$ such that $\delta_1(k, \sigma) = sink$ or $f(\delta_1(k, \sigma)) = \delta_2(f(k), \sigma)$ for all $k \in K_1 \setminus \{sink\}$ and $\sigma \in \Sigma$. Furthermore, $f(e_1) = e_2$ and for all $k_1 \in F_1$, $f(k_1) \in F_2$.*

Lemma 1. *If automaton $A_1 = (K_1, \Sigma, \delta_1, e_1, F_1)$ is weakly equivalent to automaton $A_2 = (K_2, \Sigma, \delta_2, e_2, F_2)$ then state $k_1 \in K_1 \setminus \{sink\}$ is mapped to state $k_2 \in K_2$ iff for all $w \in \Sigma^*$, $\delta_1^*(e_1, w) = k_1 \Rightarrow \delta_2^*(e_2, w) = k_2$.*

Lemma 2. *If automaton $A_1 = (K_1, \Sigma, \delta_1, e_1, F_1)$ is weakly equivalent to automaton $A_2 = (K_2, \Sigma, \delta_2, e_2, F_2)$ then, if state $k_1 \in K_1$ is mapped to state $k_2 \in K_2$ then $L(k_1) \subseteq L(k_2)$.*

The proofs of Lemma 1 and Lemma 2 are trivial and have been omitted.

Definition 3. *An automaton $A = (K, \Sigma, \delta, e, F)$ is called state-disjoint iff $L(k_1) \cap L(k_2) = \emptyset$ for all $k_1 \neq k_2 \in K$.*

Lemma 3. *Consider a transition system and an approximate sequence S'_0, $S'_1, \ldots$ as defined in Section 2. If $\lambda X \,.\, I \cup post(X, R)$ has a least fixpoint represented by a state-disjoint automaton A_∞ and the automaton A_i representing S'_i, $i \geq s$ is weakly equivalent to A_∞, then the automaton A_{i+1} representing S'_{i+1} is also weakly equivalent to A_∞.*

Proof. Let A'_i represent $S'_i \cup post(S'_i, R)$. Then $A_{i+1} = A_i \nabla A'_i$. Let $A_i = (K, \Sigma, \delta, e, F)$, $A'_i = (K', \Sigma, \delta', e', F')$ and $A_\infty = (K_\infty, \Sigma, \delta_\infty, e_\infty, F_\infty)$. Note that by Lemma 2 and the monotonicity of $\lambda X \,.\, I \cup post(X, R)$, $L(A_i) \subseteq L(A'_i) \subseteq L(A_\infty)$. Hence, the states of A_{i+1} are of two kinds: classes of $\equiv_\nabla$ that contain states from both A_i and A'_i and singleton classes containing one state from A'_i. Recall that A_i is weakly equivalent to A_∞ with respect to a function f.

First we show that for any two distinct states k_1, k_2 of A_i that belong to the same class, $f(k_1) = f(k_2)$. States k_1 and k_2 are in the same class iff they are both $\equiv_\nabla$ to some state k' of A'_i. This can happen in two ways. First, assume that there exist $w_1, w_2 \in \Sigma^*$ such that $\delta^*(e, w_1) = k_1$, $\delta^*(e, w_2) = k_2$, $\delta'^*(e', w_1) = k'$ and $\delta'^*(e', w_2) = k'$. Then by Lemma 1, $\delta_\infty^*(e_\infty, w_1) = f(k_1)$ and $\delta_\infty^*(e_\infty, w_2) = f(k_2)$. Since $L(A_i) \subseteq L(A'_i) \subseteq L(A_\infty)$, $\emptyset \neq L(k_1) \cup L(k_2) \subseteq L(k') \subseteq L(f(k_1)) \cap L(f(k_2))$. Since A_∞ is state-disjoint, we conclude that $f(k_1) = f(k_2)$. Second, assume that $L(k') = L(k_2)$ and for some $w \in \Sigma^*$, $\delta^*(e, w) = k_1$ and $\delta'^*(e', w) = k'$. Using similar arguments we conclude that $\emptyset \neq L(k_1) \subseteq L(k') = L(k_2) \subseteq L(f(k_1)) \cap L(f(k_2))$ and therefore $f(k_1) = f(k_2)$.

Now we can prove that A'_i is weakly equivalent to A_∞. To do so, we define a function $f' : K' \to K_\infty$ according to Definition 2. Given any state $k' \in K'$ such that $k' \equiv_\nabla k$ for some $k \in K$ (i.e., k' belongs to a non-singleton class), we define $f'(k') = f(k)$. Now we can show that all transitions from k' conform to Definition 2. First we consider transitions to states in non-singleton classes. If $L(k') = L(k)$ then for all $\sigma \in \Sigma$, $L(\delta'(k', \sigma)) = L(\delta(k, \sigma))$. Consequently, $\delta'(k', \sigma) \equiv_\nabla \delta(k, \sigma)$ and thus $f'(\delta'(k', \sigma)) = f(\delta(k, \sigma)) = \delta_\infty(f(k), \sigma) = \delta_\infty(f'(k'), \sigma)$. Now if $L(k') \neq L(k)$, then there exists $w \in \Sigma^*$ such that $\delta^*(e, w) =$

k and $\delta'^*(e', w) = k'$. Then $\delta^*(e, w.\sigma) = \delta(k, \sigma)$ and $\delta'^*(e', w.\sigma) = \delta'(k', \sigma)$, thus $\delta(k, \sigma) \equiv_\nabla \delta'(k', \sigma)$ and again we can conclude that $f'(\delta'(k', \sigma)) = \delta_\infty(f'(k'), \sigma)$. Now let us consider transitions going to singleton states, for which f' has not been defined yet. Assume $\delta'(k', \sigma)$ is such a state. We define $f'(\delta'(k', \sigma)) = \delta_\infty(f'(k'), \sigma)$. Now we need to prove that there is no state $k'' \in K'$ that belongs to a non-singleton class and symbol $\sigma' \in \Sigma$ such that $\delta'(k'', \sigma') = \delta'(k', \sigma)$ and $f'(\delta'(k'', \sigma')) \neq \delta_\infty(f'(k''), \sigma')$. If that were the case, we can show that $L(\delta_\infty(f'(k'), \sigma))$ would intersect $L(\delta_\infty(f'(k''), \sigma'))$, which contradicts the hypothesis. Proceeding in the same manner we can show that f' can be defined for the rest of the states, so that A'_i is weakly equivalent to A_∞. Finally, A_{i+1} is weakly equivalent to A_∞, since it is constructed by merging states of A'_i that have the same f'.

Theorem 2. *Consider a transition system and an approximate sequence* $S'_0, S'_1, \ldots$ *as defined in Section 2. If* $\lambda X \ . \ I \cup post(X, R)$ *has a least fixpoint represented by a state-disjoint automaton* A_∞ *and the automaton* A_s *representing* S'_s *is weakly equivalent to* A_∞*, then if the sequence converges, it will converge to the exact least fixpoint.*

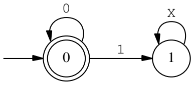
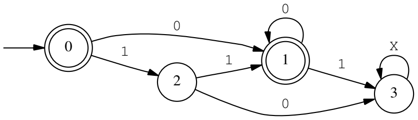
Proof. Automaton A_s, i.e., the last automaton that has been computed without widening, is weakly equivalent to A_∞. By Lemma 3, all A_i, $i > s$, are also weakly equivalent to A_∞. Since $L(A_0) \subseteq L(A_1) \subseteq \ldots \subseteq L(A_\infty)$, the limit of the sequence is $L(A_\infty)$, which represents the least fixpoint of $\lambda X \ . \ I \cup post(X, R)$.

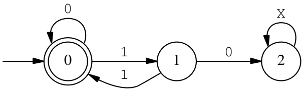
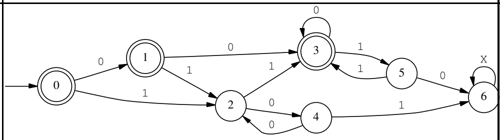
Corollary 1. *Consider a transition system with one integer variable* x *that is initially set to 0 and is increased by a constant* c *at each step. Then the approximate fixpoint corresponds to the exact set of reachable states.*

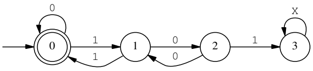
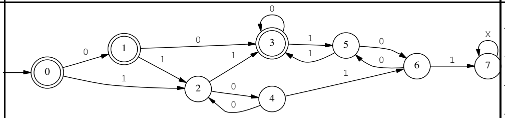
Proof. Clearly all reachable states satisfy $\exists y \geq 0 \ . \ x = c \cdot y$. If c is odd, the automaton A_∞ representing this constraint has states $0, 1, \ldots, c-1$, where $L(n)$ contains all non-negative integers for which the remainder of division with c is n. State 0 is the initial and only accepting state. Also $\delta(0, 0) = 0$. Clearly $L(n) \cap L(m) = \emptyset$ whenever $n \neq m$, therefore A_∞ is state-disjoint. Moreover, the automaton A_0 representing the initial state $x = 0$ has only one accepting and initial state that loops 0 and sends 1 to *sink*. Obviously A_0 is weakly equivalent to A_∞ and hence the hypothesis of Theorem 2 holds. Consequently, the exact set of reachable states will be computed. If c is even, it can be written as $c = 2^n \cdot d$, where d is odd. Then every number divisible by c consists of a prefix of n zeros and a suffix that is divisible by d. Following similar arguments as before we can conclude that the hypothesis of Theorem 2 holds, if the widening seed is 1.

Note that the class of systems that satisfy the hypothesis of Theorem 2 is quite large. Corollary 1 is just an example of such a system. Figure 1 illustrates an example fixpoint computation for the system described in Corollary 1, when $c = 3$. The first column shows the automata representing the approximate fixpoint iterate S_i, the second column shows the automata representing $S'_i = S_i \cup post(S_i)$ and the third column shows the equivalence classes on the states of S'_i. For each i, $S_{i+1} = S_i \nabla S'_i$.

i	Iterate S_i	S'_i	C'_i

1			{0, 1}
			{2}
			{3}

2			{0, 1, 3}
			{2, 5}
			{4}
			{6}

3			{0, 1, 3}
			{2, 5}
			{4, 6}
			{7}

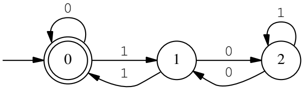
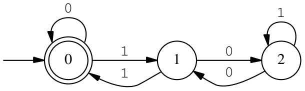
4			{0}
			{1}
			{2}

Fig. 1. Example fixpoint computation for increment by 3

```
module square()
 integer y,z;
 parameterized integer x;
 initial: y=0 && z=x;
 square: z>0 && z'=z-1 && y'=y+x;
 spec: AG([z=0 => ((x<=3 => y<=9) &&
 (x=4 => y=16) && (x>=5 => y>=25))])
endmodule
```

```
module con_decr()
 integer x,y;
 initial: x>=0 && y>=0;
 con_decr: y>0 => (x'=x-1 && y'=y-1);
 spec: AF([x=0])
endmodule
```

Fig. 2. Example specifications of transition systems

Our second example, shown in Figure 2 as module `square`, is a transition system that computes the square of an integer parameter x by iteratively adding x to a variable y that is initially set to 0. Variable z is used to count the iterations. The set of reachable states of the system contains exactly those states where $y = (x - z) \cdot x$ and is obviously non-regular. Hence, no forward fixpoint computation is expected to converge since the fixpoint cannot be represented in a finite way. Also, the closure of the transition relation of the system is not regular and thus cannot be computed exactly. On the other hand, there are meaningful properties, like the one shown in Figure 2, which can be verified using a backward fixpoint computation. The exact computation does not converge. However our algorithm terminates after 10 iterations and indeed verifies the property.

Finally, the third example illustrates the use of the collapse operator $\bar{\nabla}$, the dual of the widening operator ∇. The collapse operator should satisfy the condition that given two sets A and B, $A \cap B \supseteq A \bar{\nabla} B$. Given a widening operator ∇, one can trivially define a collapse operator as: $A \bar{\nabla} B = \neg(\neg A \nabla \neg B)$. For other representations for integer sets (e.g. Polyhedra [7], or composite disjunctive representations [17]), negation is an expensive operation and therefore this definition is not efficient. However, for deterministic automata, negation has linear complexity in the number of states (it suffices to make accepting states rejecting and vice versa), thus the above definition can be implemented efficiently.

The collapse operator is useful in disproving liveness properties and producing counter examples for them, whenever the exact fixpoint computation does not converge. As an example, consider the system shown in Figure 2 as module `con_decr`. There are two non-negative variables x and y with arbitrary initial values. We keep decrementing them concurrently while y is positive. We want to check whether $\mathrm{AF}(x = 0)$ i.e., x will eventually become 0 for all execution paths. It is easy to see that this does not hold when $x > y$. The exact backward fixpoint computation for $\mathrm{AF}(x = 0)$ does not converge. If we stop the computation after a fixed number of iterations, we can obtain an under-approximation of $\mathrm{AF}(x = 0)$ and observe that it does not include the set of initial states, thus the property might not hold. To prove that, we need to verify the negation of the initial property, namely $\mathrm{EG}(x \neq 0)$. Once again the exact fixpoint computation does not converge. However, an approximate fixpoint computation that uses our collapse operator converges in 3 steps and indeed computes the exact set of states that violate $\mathrm{AF}(x = 0)$, namely those states where $x > y$. Note that since the sequence of fixpoint iterates for EG is decreasing, we cannot use widening.

5 Experiments

Widening can be used for two purposes: 1) As we explained above, it can be used when the exact fixpoint computations do not converge; 2) It can be used to speed up fixpoint computations that would otherwise converge but only after a large number of iterations. To demonstrate this fact we repeated the experiments in [2] using widening as well.

We integrated the construction algorithms in [2, 3] as well as the approximate fixpoint computation algorithms based on the widening technique presented earlier to an infinite state CTL model checker called Action Language Verifier (ALV) [8] built on top of the Composite Symbolic Library [17]. In our experiments we compare the running times of the exact and approximate forward and backward fixpoint algorithms. For the same experiments we also present the running time of BRAIN [15]. BRAIN is a reachability analysis tool, that 1) uses Hilbert's basis as symbolic representation for integer sets, and 2) computes the exact backward fixpoint iterations. There is no approximation operation in BRAIN for the fixpoint computations which do not converge.

We experimented with a set of examples taken from the BRAIN distribution available at: `http://www.cs.man.ac.uk/~voronkov/BRAIN/` and the ALV distribution available at: `http://www.cs.ucsb.edu/~bultan/composite/`. We obtained the experimental results on a SUN ULTRA 10 work station with 768 Mbytes of memory, running SunOs 5.7. The results are presented in Table 1. For the approximate fixpoint computations we also report the seed used for widen-

Table 1. Experimental results. Time measurements appear in seconds

Problem Instance	BRAIN	ALV					
		exact forward	exact backward	approximate forward	seed	approximate backward	seed
CSM4	3.76	∞	99.35	0.21	0	79.29	5
CSM6	25.01	∞	540.88	0.21	0	482.11	7
CSM8	128.54	∞	1772.85	0.21	0	3782.51	9
CSM10	494.03	∞	4809.13	0.21	0	$\uparrow$	11
CSM12	1644.33	∞	9676.81	0.20	0	$\uparrow$	13
CSMinv10	0.93	∞	0.58	0.18	0	0.61	4
CSMinv20	3.57	∞	0.90	0.18	0	0.79	4
CSMinv30	9.59	∞	1.09	0.19	0	1.01	4
CSMinv40	20.71	∞	1.20	0.19	0	1.08	4
CSMinv50	38.58	∞	1.45	0.19	0	1.36	4
bigjava	11244.60	∞	$\uparrow$	$\uparrow$	0	$\uparrow$	0
bigjavainv	2641.05	2.85	82.33	9.78	0	43.17	10
bigjavainv1	30615.20	8.57	1160.09	8.32	0	26.13	1
consistencyprot	1.09	∞	24.28	0.34	0	26.85	7
consistencyprotinv	7.75	1.16	59.38	1.14	0	11.55	1
consistencyprotinv1	0.05	∞	0.16	0.20	0	0.17	4
consprod	11346.40	1.85	$\uparrow$	1.39	0	$\uparrow$	4
consprodinv	1.27	∞	0.66	139.17	0	0.54	5
ticket2	$\star$	0.13	∞	0.12	0	0.13	0
ticket3	$\star$	0.45	∞	0.45	0	0.56	0
ticket4	$\star$	2.93	∞	2.89	0	6.77	0
coherence	$\star$	∞	∞	0.23	0	0.13	0
bakery3	0.35	∞	0.38	7.95	0	0.44	4
bakery4	14.82	∞	9.83	1681.85	0	10.09	5
bakery5	1107.75	∞	577.45	$\uparrow$	0	582.43	6

ing. Entries of $\uparrow$ mean that the computation was aborted because it did not finish in 5 hours or the memory limit was exceeded. Entries of ∞ mean that the exact fixpoint computation does not converge. Finally, $\star$ means that we are checking a liveness property that cannot be handled by BRAIN. Problem instances can be categorized in three groups:

1. Pure integer problems (CSM, bigjava, consistencyprot and consprod)
2. Integer problems with invariants (those with the suffix inv)
3. Problems with both boolean and integer variables (ticket, coherence, bakery)

The problems with invariants are obtained from the original problems by conjoining the transition relation with a set of invariants. A typical invariant has the form $x_1 + ... + x_k < m$, where m is a natural number. Such invariants essentially bound the variables $x_1, ..., x_k$ to a finite region.

We analyzed each of the problem instances with three different configurations of ALV using two exact fixpoint computation algorithms (forward and backward) and two approximate fixpoint computation algorithms based on our widening technique. For each problem we chose a widening seed that makes our approximation precise enough to allow us to verify the properties. For the forward case, the lowest possible value for the seed, 0, is adequate. However, for the backward case we had to set the seed higher to achieve the required precision. In this sense our technique is not fully automatic. Nevertheless, one could automate the choice of widening seed by iteratively trying all possible values, starting from 0, until the property is verified.

For almost all problem instances, one of the configurations of ALV (depending on the choice of exact or approximate algorithm, forward or backward fixpoint, and the value of the widening seed) is faster than BRAIN. Note that according to [15] BRAIN outperforms other infinite state model checkers: Hytech, DMC and a version of ALV that uses a Polyhedral representation for arithmetic sets. The only two exceptions are consistencyprotinv1, for which the difference of performance is very small, and bigjava, for which ALV runs out of memory.

Among all fixpoint computation algorithms used in ALV, the approximate forward algorithm is the faster for most of the problems with the approximate backward algorithm coming second. This indicates that the use of widening can speed up fixpoint computations significantly even when the exact computations converge. We believe that the fact that the forward algorithm is usually faster than the backward algorithm is due to the specifics of each problem instance and is not a general rule. A characteristic example is consprod and consprodinv. Another observation is that while the approximate backward algorithm performs well for the invariant versions of the problem instances, it usually performs poorly for the non-invariant versions. This does not happen with the forward algorithm. An explanation to this fact is that each of the forward fixpoint iterates naturally satisfies the invariants. However, this is not true for the backward fixpoint iterates, whose size is reduced when intersected with the invariants.

The problem instances ticket2, ticket3, ticket4 and coherence could not be solved by any exact backward algorithm. For ALV, the exact backward fixpoint computation diverges whereas the forward fixpoint computation converges. BRAIN cannot handle liveness properties as the ones specified in ticket and coherence. On the other hand, both approximate algorithms were able to verify the properties. Furthermore, while the exact forward fixpoint computation diverges for most of the problem instances, the approximate one converges relatively fast for almost all of these problem instances. This shows that in practice our widening technique can be successfully applied to non-trivial systems, whereas in [4] only very simple systems are considered. A problem of special interest is bakery. Our exact backward fixpoint computation always converges and scales better than BRAIN, whereas the exact forward fixpoint computation always diverges. Widening does not help much for this problem. The approximate forward fixpoint computation does not scale well. The approximate backward fixpoint computation is precise enough only when widening is used in the last iteration and thus it takes a little longer than the exact computation to finish.

Finally, we repeated all experiments using another version of ALV in which integer sets are symbolically represented as polyhedra and manipulated by the Omega Library [1]. This version uses an extension of Halbwachs' widening algorithm [10] to Presburger arithmetic [7]. For the ticket, coherence and bakery problems, we could verify the properties using both forward and backward approximate fixpoint computations but the running times are much higher than those for the automata version. For the CSM problem instances, the approximate forward fixpoint computed was not precise enough to verify the properties immediately. Due to internal limitations of the Omega Library we could not get results for the rest of the problem instances.

References

1. The Omega project. `http://www.cs.umd.edu/projects/omega/`
2. Constantinos Bartzis and Tevfik Bultan. Efficient image computation in infinite state model checking. In *Proceedings of the 15th International Conference on Computer Aided Verification (CAV 2003)*, volume 2725 of *Lecture Notes in Computer Science*, pages 249–261. Springer, 2003.
3. Constantinos Bartzis and Tevfik Bultan. Efficient symbolic representations for arithmetic constraints in verification. *International Journal of Foundations of Computer Science*, 14(4):605–624, 2003.
4. Bernard Boigelot, Axel Legay, and Pierre Wolper. Iterating transducers in the large. In *Proceedings of the 15th International Conference on Computer Aided Verification* , volume 2725 of *LNCS*, pages 223–235. Springer, 2003.
5. Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *Computer Aided Verification*, pages 403–418, 2000.
6. A. Boudet and H. Comon. Diophantine equations, Presburger arithmetic and finite automata. In *Proceedings of the 21st International Colloquium on Trees in Algebra and Programming*, volume 1059 of *LNCS*, pages 30–43. Springer-Verlag, April 1996.
7. 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, July 1999.
8. T. Bultan and T. Yavuz-Kahveci. Action language verifier. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, 2001.
9. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th Annual ACM Symposium on Principles of Programming Languages*, pages 238–252, 1977.
10. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th Annual ACM Symposium on Principles of Programming*, pages 84–97, 1978.
11. Denis Dams, Yassine Lakhnech, and Martin Steffen. Iterating transducers. In *Computer Aided Verification'01*, 2001.
12. N. Halbwachs. *Détermination automatique de relations linéaires vérifiées par les variables d'un programme.* PhD thesis, University of Grenoble, March 1979.
13. N. Halbwachs, Y. E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, August 1997.
14. Bengt Jonsson and Marcus Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 220–234, 2000.
15. Tatiana Rybina and Andrei Voronkov. Using canonical representations of solutions to speed up infinite-state model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification*, pages 400–411, 2002.
16. P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In *Proceedings of the 6th Conference on Tools and Algorithms for the Construction and Analysis of Systems*, LNCS, pages 1–19. Springer, April 2000.
17. T. Yavuz-Kahveci, M. Tuncer, and T. Bultan. Composite symbolic library. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 335–344. Springer-Verlag, April 2001.