Efficient Image Computation in Infinite State Model Checking *

Constantinos Bartzis and Tevfik Bultan

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

Abstract. In this paper we present algorithms for efficient image computation for systems represented as arithmetic constraints. We use automata as a symbolic representation for such systems. We show that, for a common class of systems, given a set of states and a transition, the time required for image computation is bounded by the product of the sizes of the automata encoding the input set and the transition. We also show that the size of the result has the same bound. We obtain these results using a linear time projection operation for automata encoding linear arithmetic constraints. We also experimentally show the benefits of using these algorithms by comparing our implementation with LASH and BRAIN.

1 Introduction

Symbolic representations enable verification of systems with large state spaces which cannot be analyzed using enumerative approaches [15]. Symbolic model checking has been applied to verification of infinite-state systems using symbolic representations that can encode infinite sets [13, 8, 10]. One class of infinite-state systems is systems that can be specified using linear arithmetic formulas on unbounded integer variables. Verification of such systems has many interesting applications such as monitor specifications [20], mutual exclusion protocols [8, 10], parameterized cache coherence protocols [9], and static analysis of access errors in dynamically allocated memory locations (buffer overflows) [11].

There are two basic approaches to symbolic representation of linear arithmetic constraints in verification: 1) *Polyhedral representation*: In this approach linear arithmetic formulas are represented in a disjunctive form where each disjunct corresponds to a convex polyhedron. Each polyhedron corresponds to a conjunction of linear constraints [12, 13, 10]. This approach can be extended to full Presburger arithmetic by including divisibility constraints (which can be represented as equality constraints with an existentially quantified variable) [8, 16]. 2) *Automata representation*: An arithmetic constraint on v integer variables can be represented by a v-track automaton that accepts a string if it corresponds

* This work is supported in part by NSF grant CCR-9970976 and NSF CAREER award CCR-9984822.

to a v-dimensional integer vector (in binary representation) that satisfies the corresponding arithmetic constraint [5, 18, 19]. For both of these symbolic representations one can implement algorithms for intersection, union, complement, existential quantifier elimination operations, and subsumption, emptiness and equivalence tests, and therefore use them in model checking.

In [17] a third representation was introduced: *Hilbert's basis*. A conjunction of atomic linear constraints C can be represented as a unique pair of sets of vectors (N, H), such that every solution to C can be represented as the sum of a vector in N and a linear combination of vectors in H. Efficient algorithms for backward image computation, satisfiability checking and entailment checking on this representation are discussed in [17]. Based on these results an invariant checker called BRAIN which uses backward reachability is implemented [17]. The experimental results in [17] show that BRAIN outperforms polyhedral representation significantly.

In automata based symbolic model checking, the most time consuming operation is the image computation (either forward or backward). This is due to the fact that image computation involves determinization of automata, an operation with exponential worst case complexity. In this paper we propose new techniques for image computation that are provably efficient for a restricted but quite common class of transition systems. We investigate systems where the transition relation can be characterized as a disjunction of guarded updates of the form $guard \wedge update$, where *guard* is a predicate on current state variables and *update* is a formula on current and next state variables. We assume that the update formula is a conjunction of equality constraints. We show that for almost all such update formulas, image computation can be performed in time proportional to the size of the automata encoding the input set times the size of the automata encoding the guarded update. The size of the result of the image computation has the same bound. We discuss efficient implementation of the algorithms presented in this paper using BDD encoding of automata. Furthermore, we present experimental results that demonstrate the usefulness of our approach and its advantages over methods using other representations.

The rest of the paper is organized as follows. Section 2 gives an overview of automata encoding of Presburger formulas. Section 3 presents our main results. We first define four categories of updates. We present bounds for pre and post-condition computations for each category and we give algorithms that meet the given bounds. In Section 4 we describe the implementation of the given algorithms and discuss how we can integrate boolean variables to our representation. In Section 5 we present the experimental results and in Section 6 we give our conclusions.

2 Finite Automata Representation for Presburger Formulas

The representation of Presburger formulas by finite automata has been studied in [5, 19, 3, 2]. Here we briefly describe finite automata that accept the set of natural

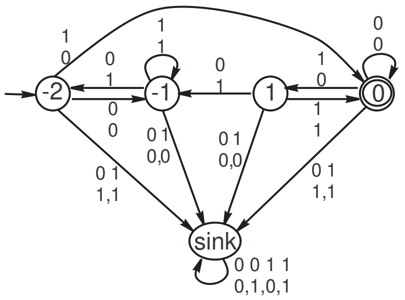

Fig. 1. An automaton for $2x - 3y = 2$

number tuples that satisfy a Presburger arithmetic formula on v variables. All the results we discuss in this paper are also valid for integers. We use natural numbers to simplify the presentation. Our implementation also handles 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 $K = \{k \mid \sum_{a_i<0} a_i \le k \le \sum_{a_i>0} a_i \lor 0 \le k \le -c \lor -c \le k \le 0\} \cup \{sink\}$, $\Sigma = \{0,1\}^v$, $e = -c$, $F = \{0\}$, and the transition function δ is defined as:

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

For inequalities, FA($\sum_{i=1}^{v} a_i \cdot x_i < 0$) $= (K, \Sigma, \delta, e, F)$, where $K = \{k \mid \sum_{a_i<0} a_i \le k \le \sum_{a_i>0} a_i \lor 0 \le k \le -c \lor -c \le k \le 0\}$, $\Sigma = \{0,1\}^v$, $e = -c$, $F = \{k \mid k \in K \land k < 0\}$, and the transition function is $\delta(k, (b_1, ..., b_v)) = \lfloor (k + \sum_{i=1}^{v} a_i \cdot b_i)/2 \rfloor$. An example automaton for the equation $2x - 3y = 2$ is shown in Figure 1.

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 size of the resulting deterministic FA can be exponential on the size of the initial FA, i.e. $|\mathrm{F}A(\exists x_i.\phi)| = 2^{|\mathrm{F}A(\phi)|}$ in the worst case. In this paper we show that for many interesting cases we can avoid this exponential blowup. 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.

3 Pre- and Post-condition computations

Two fundamental operations in symbolic verification algorithms are computing the pre- or post-conditions of a set of states (configurations) of a system. One interesting issue is investigating the sizes of the FA that would be generated by the pre- and post-condition operations.

Given a set of states $S \subseteq \mathbb{Z}^v$ of a system as a relation on v integer state variables $x_1,\ldots,x_v$ and the transition relation $R \subseteq \mathbb{Z}^{2v}$ of the system as a relation on the current state and next state variables $x_1,\ldots,x_v,x'_1,\ldots,x'_v$, we would like to compute the pre- and post-condition of S with respect to R, where $pre(S,R) \subseteq \mathbb{Z}^v$ and $post(S,R) \subseteq \mathbb{Z}^v$. 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. For example, consider a system with three integer variables x_1, x_2 and x_3. Let the current set of states be $S = \{(x_1,x_2,x_3) \mid x_1 + x_2 = x_3\}$ and the transition relation be $R = \{(x_1,x_2,x_3,x'_1,x'_2,x'_3) \mid x_1 > 0 \wedge x'_1 = x_1 - 1 \wedge x'_2 = x_2 \wedge x'_3 = x_3\}$. Then the post-condition of S with respect to R is $post(S,R) = \{(x_1,x_2,x_3) \mid x_1 > -1 \wedge x_1 + x_2 = x_3 - 1\}$. The pre-condition of S with respect to R is $pre(S,R) = \{(x_1,x_2,x_3) \mid x_1 > 0 \wedge x_1 + x_2 = x_3 + 1\}$.

One can compute $post(S,R)$ by first conjoining ϕ_S and ϕ_R, then existentially eliminating the current state variables, and finally renaming the variables, i.e., $\phi_{post(S,R)}$ is equivalent to $(\exists x_1 \ldots \exists x_v.(\phi_S \wedge \phi_R))_{[x'_1 \leftarrow x_1,\ldots,x'_v \leftarrow x_v]}$ where $\psi_{[y \leftarrow z]}$ is the formula generated by substituting z for y in ψ. On the other hand, $pre(S,R)$ can be computed by first renaming the variables in ϕ_S, then conjoining with ϕ_R, and finally existentially eliminating the next state variables, i.e., $\phi_{pre(S,R)}$ is equivalent to $\exists x'_1 \ldots \exists x'_v.(\phi_{S[x_1 \leftarrow x'_1,\ldots,x_v \leftarrow x'_v]} \wedge \phi_R)$. Hence, to compute $post(S,R)$ and $pre(S,R)$ we need three operations: conjunction, existential variable elimination and renaming.

As stated earlier, given FA(ϕ) representing the set of solutions of ϕ, FA($\exists x_1,\ldots,\exists x_n.\phi$) can be computed using projection and the size of the resulting FA is at most $O(2^{|\text{FA}(\phi)|})$. Note that existential quantification of more than one variable does not increase the worst case complexity since the determinization can be done once at the end, after all the projections are done. As discussed earlier, conjunction operation can be computed by generating the product automaton, and the renaming operation can be implemented as a linear time transformation of the transition function. Hence, given formulas ϕ_S and ϕ_R representing S and R, and corresponding FA, FA(ϕ_S) and FA(ϕ_R), the size of FA($\phi_{post(S,R)}$) and FA($\phi_{pre(S,R)}$) is $O(2^{|\text{FA}(\phi_S)|\cdot|\text{FA}(\phi_R))|})$ in the worst case. Below, we show that under some realistic assumptions, the size of the automaton resulting from pre- or post-condition computations is much better.

We assume that the formula ϕ_R defining the transition relation R is a guarded-update of the form $guard(R) \wedge update(R)$, where $guard(R)$ is a Presburger formula on current state variables $x_1, \ldots, x_v$ and $update(R)$ is of the form $x'_i = f(x_1, \ldots, x_v) \wedge \bigwedge_{j \neq i} x'_j = x_j$ for some $1 \leq i \leq v$, where $f : Z^v \to Z$ is a linear function. This is a realistic assumption, since in asynchronous concurrent systems, the transition relation is usually defined as a disjunction of such guarded-updates. Also, note that, the post-condition of a transition relation which is a disjunction of guarded-updates is the union of the post-conditions of individual guarded-updates, and can be computed by computing post-condition of one guarded-update at a time. The same holds for pre-condition.

We consider four categories of updates:

1. $x'_i = c$
2. $x'_i = x_i + c$
3. $x'_i = \sum_{j=1}^{v} a_j \cdot x_j + c$, where a_i is odd
4. $x'_i = \sum_{j=1}^{v} a_j \cdot x_j + c$, where a_i is even

Note that categories 1 and 2 are subcases of categories 4 and 3 respectively. We can prove that pre-condition computation can be performed efficiently for categories 1-4 and post-condition computation can be performed efficiently for categories 2-3. For each of these cases we give algorithms for computing pre- and post-conditions, and derive upper bounds for the time complexity of the algorithms and the size of the resulting automata. We define ϕ'_S as $\phi_{S[x_1 \leftarrow x'_1, \ldots, x_v \leftarrow x'_v]}$. The following Theorem will be used later for the complexity proofs.

Theorem 1. *Given a formula ψ of the form $\phi(x_1, \ldots, x_v) \wedge \sum_{j=1}^{v} a_j \cdot x_j = c$, where a_i is odd for some $0 \leq i \leq v$, the deterministic automaton FA($\exists x_i.\psi$) can be computed from FA(ψ) in linear time and it will have the same number of states.*

Proof. For all $(v-1)$-bit tuples $\sigma \in \{0,1\}^{v-1}$ we define $\sigma_{b_i=b}$ to be the v-bit tuple resulting from σ if we insert the bit b in the ith position of σ. For example if $\sigma = (1,1,0,0)$ then $\sigma_{b_3=0} = (1,1,0,0,0)$ and $\sigma_{b_3=1} = (1,1,1,0,0)$. Let FA($\psi$)=$(K, \{0,1\}^v, \delta, e, F)$. Then the non-deterministic automaton FA($\exists x_i.\psi$) is $(K, \{0,1\}^{v-1}, \delta', e, F)$, where $\delta'(k, \sigma) = \{\delta(k, \sigma_{b_i=0}), \delta(k, \sigma_{b_i=1})\}$. Since a_i is odd, we know that $\delta(k, \sigma_{b_i=0})$ or $\delta(k, \sigma_{b_i=1})$ is $sink$. This is because $\forall k \in K, (b_1, \ldots, b_v) \in \{0,1\}^v$ either $k + \sum_{j \neq i} a_j \cdot b_j$ or $k + \sum_{j \neq i} a_j \cdot b_j + a_i$ is odd. By the definition of automata for equalities in Section 2, one of the two transitions goes to $sink$ state. We also know that transitions that go to the $sink$ state in non-deterministic automata can be safely removed, since they can never be part of an accepting path. So in order to determinize FA($\exists x_i.\psi$) we only need to remove from $\delta'(k, \sigma)$ one of its two members that is $sink$. Figure 2 shows the algorithm that computes deterministic FA($\exists x_i.\psi$) from FA(ψ). Clearly, the complexity of the algorithm is $O(|\mathrm{FA}(\psi)|)$.

```
Input FA(ψ) = (K, {0,1}^v, δ, e, F), where ψ = φ ∧ Σ_{j=1}^{v} a_j · x_j = c
        integer i, 0 ≤ i ≤ v
Output FA(∃x_i.ψ) = (K, {0,1}^{v-1}, δ', e, F)


FOR ALL k ∈ K, σ ∈ {0,1}^{v-1} DO
   IF δ(k, σ_{b_i=0}) = sink THEN
      δ'(k, σ) = δ(k, σ_{b_i=1})
   ELSE
      δ'(k, σ) = δ(k, σ_{b_i=0})
```

Fig. 2. Projection algorithm of Theorem 1

3.1 Image Computation for $x'_i = c$ Updates

For each category of updates we discuss both pre-condition and post-condition computations. For each case we simplify the formulas $\phi_{pre(S,R)}$ and $\phi_{post(S,R)}$ and discuss the computation of their automata representations.

Pre-condition computation

$$\begin{aligned}
\phi_{pre(S,R)} &\Leftrightarrow \exists x'_1 \ldots \exists x'_v . (\phi'_S \wedge \phi_R) \\
&\Leftrightarrow \exists x'_1 \ldots \exists x'_v . (\phi'_S \wedge \mathit{guard}(R) \wedge \mathit{update}(R)) \\
&\Leftrightarrow (\exists x'_1 \ldots \exists x'_v . (\phi'_S \wedge \mathit{update}(R))) \wedge \mathit{guard}(R) \\
&\Leftrightarrow (\exists x'_1 \ldots \exists x'_v . (\phi'_S \wedge x'_i = c \wedge \bigwedge_{j \neq i} x'_j = x_j)) \wedge \mathit{guard}(R) \\
&\Leftrightarrow (\exists x'_i . (\phi_{S[x_i \leftarrow x'_i]} \wedge x'_i = c)) \wedge \mathit{guard}(R) \\
&\Leftrightarrow (\exists x_i . (\phi_S \wedge x_i = c)) \wedge \mathit{guard}(R).
\end{aligned}$$

FA($x_i = c$) can be constructed in $O(\log_2 c)$ time and has $O(\log_2 c)$ states. Thus, FA($\phi_S \wedge x_i = c$) has $O(|\mathrm{FA}(\phi_S)| \cdot \log_2 c)$ states. By Theorem 1, the size of FA($\exists x_i.(\phi_S \wedge x_i = c)$) and the time needed to compute it is $O(|\mathrm{FA}(\phi_S)| \cdot \log_2 c)$.

Post-condition computation

$$\begin{aligned}
\phi_{post(S,R)} &\Leftrightarrow (\exists x_1 \ldots \exists x_v . (\phi_S \wedge \phi_R))_{[x'_1 \leftarrow x_1, \ldots, x'_v \leftarrow x_v]} \\
&\Leftrightarrow (\exists x_1 \ldots \exists x_v . (\phi_S \wedge \mathit{guard}(R) \wedge \mathit{update}(R)))_{[x'_1 \leftarrow x_1, \ldots, x'_v \leftarrow x_v]} \\
&\Leftrightarrow (\exists x_1 \ldots \exists x_v . (\phi_S \wedge \mathit{guard}(R) \wedge x'_i = c \wedge \bigwedge_{j \neq i} x'_j = x_j))_{[x'_1 \leftarrow x_1, \ldots, x'_v \leftarrow x_v]} \\
&\Leftrightarrow (\exists x_i . (\phi_S \wedge \mathit{guard}(R))) \wedge x_i = c.
\end{aligned}$$

Unfortunately, we cannot use Theorem 1 in this case. We can compute FA($\exists x_i.(\phi_S \wedge \mathit{guard}(R))$) from FA($\phi_S \wedge \mathit{guard}(R)$) by projection with a worst case exponential time and space complexity.

3.2 Image Computation for $x'_i = x_i + c$ Updates

Suppose ϕ_S and $guard(R)$ consist of atomic linear constraints of the form ϕ_k : $\sum_{i=1}^{v} a_{i,k} \cdot x_i \sim c_k, 1 \leq k \leq l$, where $\sim \in \{=, \neq, >, \geq, \leq, <\}$, and Boolean connectives.

Pre-condition computation

$$\begin{aligned}\phi_{pre(S,R)} &\Leftrightarrow (\exists x'_1 \ldots \exists x'_v . (\phi'_S \wedge x'_i = x_i + c \wedge \bigwedge_{j \neq i} x'_j = x_j)) \wedge guard(R) \\ &\Leftrightarrow \phi_{S[x_i \leftarrow x_i + c]} \wedge guard(R) \;\; \Leftrightarrow \;\; \phi_{S[c_k \leftarrow c_k - a_{i,k} \cdot c]} \wedge guard(R).\end{aligned}$$

Post-condition computation

$$\begin{aligned}\phi_{post(S,R)} &\Leftrightarrow (\exists x_1 \ldots \exists x_v . (\phi_S \wedge guard(R) \wedge x'_i = x_i + c \wedge \\ &\qquad \bigwedge_{j \neq i} x'_j = x_j))_{[x'_1 \leftarrow x_1, \ldots, x'_v \leftarrow x_v]} \\ &\Leftrightarrow (\phi_S \wedge guard(R))_{[x_i \leftarrow x_i - c]} \;\; \Leftrightarrow \;\; (\phi_S \wedge guard(R))_{[c_k \leftarrow c_k + a_{i,k} \cdot c]}.\end{aligned}$$

It is clear that for both pre- and post-condition computation only the constant term changes in each atomic linear constraint. An algorithm that changes the constant term in an atomic equation is shown in Figure 3. The algorithm for inequations is similar. Note that the complexity of both algorithms is proportional to the number of new states introduced, which is possibly 0 or at most $|c'|$. These algorithms assume that atomic formulas are stored with the corresponding automata. In our implementation this is not the case, and we actually use the more general approach presented next.

3.3 Image Computation for $x'_i = \sum_{j=1}^{v} a_j \cdot x_j + c$ Updates

Pre-condition computation

$$\begin{aligned}\phi_{pre(S,R)} &\Leftrightarrow (\exists x'_1 \ldots \exists x'_v . (\phi'_S \wedge x'_i = \sum_{j=1}^{v} a_j \cdot x_j + c \wedge \bigwedge_{j \neq i} x'_j = x_j)) \wedge guard(R) \\ &\Leftrightarrow (\exists x'_i . (\phi_{S[x_i \leftarrow x'_i]} \wedge x'_i = \sum_{j=1}^{v} a_j \cdot x_j + c)) \wedge guard(R).\end{aligned}$$

Again we can use Theorem 1 to prove that existential variable elimination can be performed in linear time without increasing the automaton size. We use the algorithm in Figure 2 to compute $\mathrm{FA}(\exists x'_i . (\phi_{S[x_i \leftarrow x'_i]} \wedge x'_i = \sum_{j=1}^{v} a_j \cdot x_j + c))$.

```
Input FA(∑_{i=1}^{v} a_i · x_i = c) = (K, Σ, δ, e, F)
Output FA(∑_{i=1}^{v} a_i · x_i = c') = (K', Σ', δ', e', F')


IF −c' ∈ K THEN
    K' = K  Σ' = Σ  δ' = δ  e' = −c'  F' = F
ELSE
    K' = K ∪ {−c'}  Σ' = Σ  δ' = δ  e' = −c'  F' = F
    WHILE ∃k ∈ K', σ ∈ Σ' s.t. δ'(k, σ) = null DO
        FOR ALL σ = (b_1, ..., b_v) ∈ Σ' DO
            IF l := (∑_{i=1}^{v} a_i · b_i + k)/2 ∈ ℤ THEN
                K' := K' ∪ {l}
                δ'(k, σ) := l
            ELSE
                δ'(k, σ) := sink
```

Fig. 3. Algorithm for changing the constant term in equations

Post-condition computation

$$\phi_{post(S,R)} \Leftrightarrow (\exists x_1 \ldots \exists x_v . (\phi_S \wedge guard(R) \wedge x'_i = \sum_{j=1}^{v} a_j \cdot x_j + c \wedge \bigwedge_{j \neq i} x'_j = x_j))_{[x'_1 \leftarrow x_1, \ldots, x'_v \leftarrow x_v]}$$

$$\Leftrightarrow (\exists x_i . (\phi_S \wedge guard(R) \wedge x'_i = \sum_{j=1}^{v} a_j \cdot x_j + c))_{[x'_i \leftarrow x_i]}.$$

Note that in this case, Theorem 1 applies only when a_i is an odd integer and in that case we can use the algorithm in Figure 2.

4 Implementation

A problem with the FA representation for arithmetic constraints is the size of the transition function, since the number of transitions from each state is exponential on v, the number of integer variables. Hence, it is impractical to store the transition function as a table. Actual implementations use different solutions to this problem. We have used the approach used in MONA [14]. MONA is an automata package that uses BDDs [6] to store the transition function. In particular, for each FA state n, there is a BDD representing the function $\delta(n, (b_1, ..., b_v))$. The terminal nodes are also FA states and internal nodes can be shared. To evaluate $\delta(n, (b_1, ..., b_v))$ one should start at the root of the BDD for state n and move from node to node depending on the values of $b_1, ..., b_v$ until a leaf node is reached. That leaf node corresponds to the state $\delta(n, (b_1, ..., b_v))$. Since BDDs are a canonical representation for Boolean functions, given a fixed variable ordering, the size of the transition relation can be kept minimal, e.g., variables with zero coefficients do not appear in the BDD representing the transition function. We can also prove that the size of the BDD is linear in the number of variables v and not exponential [2].

The algorithm of Theorem 1 is linear in the number of transitions. When the transition function is represented as a BDD we would like the algorithm to be linear in the number of BDD nodes in the automaton. For the case of pre-condition computation, this is feasible for a new category of updates: $x'_i = \sum_{j=1}^{i} a_j \cdot x_j + c$, i.e. updates where the new value of x_i depends on the old value of itself or variables with smaller indices. This new category includes the original categories 1 and 2. For a system with v variables, we fix the order of the variables in the BDDs to be $x_1, x'_1, ..., x_v, x'_v$. In $\mathrm{FA}(\phi_{S[x_i \leftarrow x'_i]} \wedge x'_i = \sum_{j=1}^{i} a_j \cdot x_j + c)$, according to Theorem 1, we know that at least one of the children of any node for variable x'_i points to the *sink* state. Consequently, we can compute $\mathrm{FA}(\exists x'_i.(\phi_{S[x_i \leftarrow x'_i]} \wedge x'_i = \sum_{j=1}^{i} a_j \cdot x_j + c))$ by visiting all nodes for variable x_i and delete one of the children that goes to the *sink* state. Every BDD node is visited once, thus the whole operation can be performed in time linear in the total number of BDD nodes in the automaton.

The BDD representation of the FA transition function also allows efficient handling of boolean formulas [2]. To accommodate boolean variables, we encode $false$ with $0(0 \cup 1)^*$ and $true$ with $1(0 \cup 1)^*$. This way, in an automaton that represents a composite formula with both boolean and integer variables, only the BDD rooted at the initial state will contain nodes that depend on the boolean variables. All other BDDs will contain only nodes for the integer variables and thus their size is independent of the number of boolean variables. In other words, the BDD rooted at the initial state evaluates the boolean part of the formula and the rest of the automaton evaluates the integer part.

Given a formula ϕ containing both boolean and integer variables, we define a boolean subformula of ϕ to be either a boolean variable appearing in ϕ, a negated boolean variable, a constant *true* or $false$, or two boolean subformulas connected by a logical connective ($\wedge, \vee$, etc) in ϕ. A maximal boolean subformula of ϕ is a boolean subformula of ϕ that is not contained in any other boolean subformula of ϕ.

Now suppose that ϕ is a general formula containing distinct maximal boolean subformulas $(B_1, ..., B_n)$ and distinct atomic linear integer arithmetic constraints $(P_1, ..., P_m)$ combined with boolean connectives. We can prove that the total size of the BDD representing the transition function of $\mathrm{FA}(\phi)$ is $O(\prod_{i=1}^{n} |\mathrm{FA}(B_i)| + \prod_{i=1}^{m}(|\mathrm{FA}(P_i)| + 1))$, where $|\mathrm{FA}(B_i)|$ and $|\mathrm{FA}(P_i)|$ are the sizes (in BDD nodes) of the automata representing B_i and P_i respectively [2].

5 Experiments

We integrated the construction algorithms in [3, 2] as well as the pre- and post-condition computation algorithms presented in this paper to an infinite state

CTL model checker called Action Language Verifier (ALV) [7] built on top of the Composite Symbolic Library [21]. The Composite Symbolic Library uses an object-oriented design to combine different symbolic representations [21]. In our experiments we compare the efficiency of our implementation with BRAIN [17] that uses Hilbert's basis as canonical representation for arithmetic constraints, and LASH [1] that uses the automata representation. To make the comparison with LASH fair, we integrated the automata construction and manipulation algorithms used in LASH to the Action Language Verifier.

We experimented with a large set of examples taken from 1) The Action Language Verifier distribution at: http://www.cs.ucsb.edu/˜bultan/composite/ and 2) The BRAIN distribution at: http://www.cs.man.ac.uk/˜voronkov/BRAIN/. All the examples used in our experiments and the executables of the tools are available at: http://www.cs.ucsb.edu/˜bar/image. 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. Time measurements appear in seconds. Entries of ↑↑ mean that the computation was aborted because the memory limit was exceeded. Entries of ↑ mean that the computation was aborted at 12000 seconds because no significant progress was made. For the automata representation used in ALV we also recorded the size (i.e., number of BDD nodes) of the largest automaton computed.

The experimental results show that the automata representation used in LASH is not efficient. There are two main reasons for this inefficiency. First, LASH stores the transition function of automata explicitly as opposed to the multi-terminal BDD representation used in MONA. Second, LASH implements the image computation using a standard automata projection algorithm which has a worst case exponential complexity, as opposed to the polynomial time image computations proposed here.

Problem instances can be categorized in three groups:

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

Except for the consistency protocol problem instance, it is clear that ALV is faster than BRAIN for groups 2 and 3 and BRAIN is faster only for problems in group 1. The problems with invariants are obtained from the original problems by adding 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. The presence of finite domain variables causes a problem for BRAIN, because the size of the Basis (the canonical representation used in BRAIN) can grow exponentially. On the other hand, systems with finite domain variables can be efficiently encoded by automata with transition functions stored as BDDs.

There are several advantages of the automata representation of arithmetic formulas over the Hilbert's basis representation used in BRAIN:

Table 1. Experimental results. Time measurements appear in seconds. Max size is the number of BDD nodes of the largest automaton computed for each problem instance

Problem Instance	BRAIN	ALV		LASH
	time	time	max size	time
CSM4	3.76	99.35	25910	↑
CSM6	25.01	540.88	110796	↑
CSM8	128.54	1772.85	238739	↑
CSM10	494.03	4809.13	484249	↑
CSM12	1644.33	9676.81	839870	↑
CSMinv10	0.93	0.58	485	217.76
CSMinv20	3.57	0.90	606	282.49
CSMinv30	9.59	1.09	727	416.46
CSMinv40	20.71	1.20	727	458.48
CSMinv50	38.58	1.45	910	601.21
incdec	195.48	2792.54	258945	↑
incdecinv	24.79	4.67	1194	↑
bakery3	0.35	0.38	509	10.65
bakery4	14.82	9.83	8762	244.67
bakery5	1107.75	577.45	230906	↑
barber4	0.74	0.25	76	11.79
barber8	52.05	0.78	136	27.01
barber12	14669.80	1.81	212	53.41
bigjava	11244.60	↑↑	↑↑	↑
bigjavainv	2641.05	82.33	6157	↑
bigjavainv1	30615.20	1160.09	45114	↑
consistencyprot	1.09	24.28	15049	↑
consistencyprotinv	7.75	59.38	31453	↑
consistencyprotinv1	0.05	0.16	212	182.84
consprod	11346.40	↑↑	↑↑	↑
consprodinv	1.27	0.66	253	↑

1. Automata can handle a larger class of systems, namely all transition systems representable by Presburger arithmetic formulas. BRAIN cannot handle quantified formulas or divisibility constraints.
2. For the class of systems for which BRAIN provides polynomial time image computation we prove polynomial bounds for the automata representation and give the algorithms. Even for problems in group 1 for which BRAIN is faster than ALV, the speedup achieved by BRAIN seems to be constant, which is what we would expect given that both techniques have equally efficient image computations. In particular, for problem CSM, ALV scales better even though BRAIN is faster.
3. The automata representation can handle forward image computation and solve problems for which the backward fixpoint computation does not converge, but the forward computation does. Such problems are not solvable using BRAIN. For example, we can verify mutual exclusion and starvation

freedom properties for the ticket mutual exclusion protocol [8] using forward fixpoint computations, whereas this is not possible for BRAIN.
4. The automata representation can be combined with an efficient encoding of boolean and enumerated variables, however it is not clear if this could be done efficiently with the Hilbert's basis technique. In BRAIN specification of the problems in group 3, boolean and enumerated variables have been mapped to integers. The experimental results indicate the inefficiency of this mapping, which becomes more apparent when the problem size increases.
5. Using the automata representation we can perform full CTL verification, whereas BRAIN can only verify invariants. For example, for the bakery protocol we can verify liveness properties of the form: $AG(pc = try \Rightarrow AF(pc = cs))$, while BRAIN cannot.

However, on pure integer problems with large number of variables, BRAIN outperforms ALV. We plan to investigate if this is due to the efficiency of the Hilbert's Basis representation or due to the fixpoint computation algorithm used in BRAIN.

6 Conclusion

In this paper we show that for a common class of infinite state systems represented by linear arithmetic constraints, image computations can be done efficiently without an exponential blow up. We give algorithms for efficient image computations for updates that are expressed as linear equalities based on an automata encoding of the states of the system. We implemented these algorithms and experiments show that they improve the efficiency of automata based representations significantly. Experiments also indicate that, in a lot of cases, automata encoding with the proposed image computations is as efficient as other more restrictive canonical representations.

The results in this paper can also be used to show that image computation on bounded arithmetic constraints represented by BDDs can be done in polynomial time for a class of arithmetic constraints. We plan to develop algorithms for the bounded case based on the BDD encodings of arithmetic constraints presented in [4].

References

1. The Liège Automata-based Symbolic Handler (LASH). Available at `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.
2. Constantinos Bartzis and Tevfik Bultan. Efficient symbolic representations for arithmetic constraints in verification. *International Journal of Foundations of Computer Science*. To appear.
3. Constantinos Bartzis and Tevfik Bultan. Automata-based representations for arithmetic constraints in automated verification. In *Proceedings of the Seventh International Conference on Implementation and Application of Automata*, 2002.

4. Constantinos Bartzis and Tevfik Bultan. Construction of efficient BDDs for bounded arithmetic constraints. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2003.
5. A. Boudet and H. Comon. Diophantine equations, Presburger arithmetic and finite automata. In H. Kirchner, editor, *Proceedings of the 21st International Colloquium on Trees in Algebra and Programming - CAAP'96*, volume 1059 of *Lecture Notes in Computer Science*, pages 30–43. Springer-Verlag, April 1996.
6. R. Bryant. Graph-based algorithms for boolean function manipulation. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, 1990.
7. T. Bultan and T. Yavuz-Kahveci. Action language verifier. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, 2001.
8. Tevfik Bultan, Richard Gerber, and William 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.
9. G. Delzanno and T. Bultan. Constraint-based verification of client-server protocols. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, 2001.
10. Giorgio Delzanno and Andreas Podelski. Constraint-based deductive model checking. *Journal of Software Tools and Technology Transfer*, 3(3):250–270, 2001.
11. N. Dor, M. Rodeh, and M. Sagiv. Cleanness checking of string manipulations in C programs via integer analysis. In *Proceedings of the 8th International Static Analysis Symposium*, volume 2126 of *Lecture Notes in Computer Science*, pages 194–212. Springer-Verlag, 2001.
12. 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.
13. T. A. Henzinger, P. Ho, and H. Wong-Toi. Hytech: a model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.
14. Nils Klarlund and Anders Møller. *MONA Version 1.4 User Manual*. BRICS Notes Series NS-01-1, Department of Computer Science, University of Aarhus, 2001.
15. K. L. McMillan. *Symbolic model checking*. Kluwer Academic Publishers, Massachusetts, 1993.
16. W. Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis, 1992.
17. 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.
18. P. Wolper and B. Boigelot. An automata-theoretic approach to Presburger arithmetic constraints. In *Proceedings of the Static Analysis Symposium*, 1995.
19. P. Wolper and B. Boigelot. On the construction of automata from linear arithmetic constraints. In S. Graf and M. Schwartzbach, editors, *Proceedings of the 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 1–19. Springer, April 2000.
20. T. Yavuz-Kahveci and T. Bultan. Specification, verification, and synthesis of concurrency control components. In *Proceedings of the International Symposium on Software Testing and Analysis*, pages 169–179, July 2002.
21. 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.