On Bounded Reachability of Programs with Set Comprehensions

Margus Veanes[1] and Ando Saabas[2⋆]

[1] Microsoft Research, Redmond, WA, USA
margus@microsoft.com
[2] Institute of Cybernetics, Tallinn University of Technology, Tallinn, Estonia
ando@cs.ioc.ee

Abstract. We analyze the bounded reachability problem of programs that use abstract data types and set comprehensions. Such programs are common as high-level executable specifications of complex protocols. We prove decidability and undecidability results of restricted cases of the problem and extend the Satisfiability Modulo Theories approach to support analysis of set comprehensions over tuples and bag axioms. We use the Z3 solver for our implementation and experiments, and we use AsmL as the modeling language.

1 Introduction

Programs that use high-level data types are commonly used to describe executable specifications [22] in form of so called *model programs*. An important and growing application area in the software industry is the use of model programs for specifying and documenting expected behavior of application-level network protocols [14]. Model programs typically use abstract data types such as sets and maps, and comprehensions to express complex state updates. Correctness assumptions about the model are usually expressed through state invariants. An important problem is to validate a model prior to its use as an oracle or final specification. One approach is to use Satisfiability Modulo Theories or SMT to perform bounded reachability analysis or bounded model-checking of model programs [36]. The use of SMT solvers for automatic software analysis has recently been introduced [1, 11] as an extension of SAT-based bounded model checking [5]. The SMT based approach makes it possible to deal with more complex background theories. Instead of encoding the verification task of a sequential program as a propositional formula the task is encoded as a quantifier free formula. The decision procedure for checking the satisfiability of the formula may use combinations of background theories [29].

⋆ Part of this work was done during an internship at Microsoft Research, Redmond.

The main contribution of this paper is a characterization of the decidable and undecidable cases of the bounded reachability problem of model programs. We show in Section 3 that already the single step reachability problem is undecidable if a single set-valued parameter is allowed. In Section 4 we show that the bounded reachability problem remains decidable provided that all parameters have basic (non-set valued) types. This result is orthogonal to the decidable fragment of bounded reachability of model programs that use the array property fragment [8]. In Section 5 the paper extends the work started in [36] through improved handling of quantifier instantiation and extended support for background axioms to support for example bag or multi-set axioms. We use the SMT solver Z3 [10] for our experiments and we use AsmL [16] as the modeling language. Related work is discussed in Section 6.

2 Model programs and bounded reachability

In this section we define some background material related to model programs, in order to make the paper self-contained. A more thorough exposition can be found in [36].

Model programs The main use of model programs is as high-level specifications in model-based testing tools such as Spec Explorer [37] and NModel [30]. In Spec Explorer, one of the supported input languages is the abstract state machine language AsmL [16]. AsmL is used in this paper as the concrete specification language for update rules that correspond to basic ASMs [15].

We let Σ denote the overall signature of function symbols. Part of Σ is denoted by Σ^{var} and contains nullary function symbols whose interpretation may vary from state to state, called *state variables*. The remaining part Σ^{static} contains symbols whose interpretation is fixed by the background theory. A ground term over Σ^{static} is called a *value term*. A subset of Σ^{static}, denoted by Σ^{acts} are free constructors called *action symbols*. Given an action symbol f, an *action* or *f-action* is a value term whose function symbol is f.

For all action symbols f with arity $n \geq 0$, and all i, $1 \leq i \leq n$, there is a unique *parameter variable* (not in Σ^{var}) denoted by f_i. We write Σ_f for $\{f_i\}_{1 \leq i \leq n}$. Note that if $n = 0$ then $\Sigma_f = \emptyset$.

Definition 1. A *model program* P is a tuple (V_P, A_P, I_P, R_P), where

- V_P is a finite set of *state variables*, let Σ_P denote $\Sigma^{\text{static}} \cup V_P$;

- A_P is a finite set of *action symbols*;
- I_P is a formula over Σ_P, called the *initial state condition*;
- R_P is a family $\{R_P^f\}_{f \in A_P}$ of *action rules* $R_P^f = (G_P^f, U_P^f)$, where
  - G_P^f is a quantifier free formula over $\Sigma_P \cup \Sigma_f$ called the *guard*;
  - U_P^f, called the *update rule*, is a block $\{v := t_v^f\}_{v \in V_P^f}$ of *assignments* where t_v^f is a term over $\Sigma_P \cup \Sigma_f$ and $V_P^f \subseteq V_P$.

This definition is a variation of model programs that syntactically restricts the update rules to be block assignments. This restriction is not a true limitation because *if-then-else* terms are allowed and nondeterministic choices can be encoded as branching based on action parameter values (i.e. the choices are made explicit). We often say *action* to also mean an action rule or an action symbol, if the intent is clear from the context.

In general, model programs can have a rich background theory, including the theory of maps. In the following example, the fragment of interest is the so-called *array theory* fragment where all map sorts have domain sort $\mathbb{Z}$ and the theory of $\mathbb{Z}$ is Presburger arithmetic

Example 1 (Credits). The following model program is written in AsmL. It specifies how a client and a server need to use message ids, based on a sliding window protocol. It models part of the credits-algorithm in the SMB2 [34] protocol.

```
var window as Set of Integer = {0}
var maxId as Integer = 0
var requests as Map of Integer to Integer = {->}

[Action("Req(_,m,c)")] Req(m as Integer, c as Integer)
  require m in window and c > 0
  requests := Add(requests,m,c)
  window := window difference {m}

[Action("Res(_,m,c,_)")] Res(m as Integer, c as Integer)
   require m in requests
   require requests(m) >= c
   require c >= 0
   window := window union {maxId + i | i in {1..c}}
   requests := RemoveAt(requests,m)
   maxId := maxId + c

[Invariant] ClientHasEnoughCredits()
   require requests = {->} implies window <> {}
```

The *Credits* model program illustrates a typical use of model programs as protocol-specifications. Actions use parameters, maps and sets are used as state variables and a comprehension expression is used to compute a set. (Since the domain of the maps and sets is $\mathbb{Z}$, the example is in the

array theory fragment.) Each action has a guard and an update rule given by a basic ASM. For example, the guard of the `Req` action requires that the id of the message is in the current window of available ids and that the number of credits that the client requests from the server is positive. The state invariant associated with the model program is that the client must not starve, i.e. there should always be a message id available at some point, so that the client can issue new requests.

Let P be a fixed model program. A *P-state* is a mapping of V_P to values.[3] Given an action $a = f(a_1, \ldots, a_n)$, let θ_a denote the parameter assignment $\{f_i \mapsto a_i\}_{1 \leq i \leq n}$. Given a P-state S, an extension of S with the parameter assignment θ is denoted by $(S; \theta)$.

Let S be a P-state, an f-action a is *enabled* in S if $(S; \theta_a) \models G_P^f$ (where $\models$ is the standard satisfaction relation of first-order logic). The action *a causes a transition from S to S'*, where

$$S' = \{v \mapsto t_v^{f(S;\theta_a)}\}_{v \in V_P^f} \cup \{v \mapsto v^S\}_{v \in V_P \setminus V_P^f}.$$

A *labeled transition system* or *LTS* is a tuple $(\mathcal{S}, \mathcal{S}_0, L, T)$, where $\mathcal{S}$ is a set of *states*, $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of *initial states*, L is a set of labels and $T \subseteq \mathcal{S} \times L \times \mathcal{S}$ is a *transition relation*.

Definition 2. Let P be a model program. The *LTS of P*, denoted by $[\![P]\!]$ is the LTS $(\mathcal{S}, \mathcal{S}_0, L, T)$, where $\mathcal{S}_0$, is the set of all P-states s such that $s \models I_P$; L is the set of all actions over A_P; T and $\mathcal{S}$ are the least sets such that, $\mathcal{S}_0 \subseteq \mathcal{S}$, and if $s \in \mathcal{S}$ and there is an action a that causes a transition from s to s' then $s' \in \mathcal{S}$ and $(s, a, s') \in T$.

A *run* of P is a sequence of transitions $(s_i, a_i, s_{i+1})_{i<\kappa}$ in $[\![P]\!]$, for some $\kappa \leq \omega$, where s_0 is an initial state of $[\![P]\!]$. The sequence $(a_i)_{i<\kappa}$ is called an (*action*) *trace* of P. The run or the trace is *finite* if $\kappa < \omega$.

Bounded reachability of model programs Let P be a model program and let φ be a Σ_P-formula. The main problem we are addressing is whether φ is reachable in P within a given bound.

Definition 3. Given φ and $k \geq 0$, *φ is reachable in P within k steps*, if there exists an initial state s_0 and a (possibly empty) run $(s_i, a_i, s_{i+1})_{i<l}$ in P, for some $l \leq k$, such that $s_l \models \varphi$. If so, the action sequence $\alpha = (a_i)_{i<l}$ is called a *reachability trace for φ* and s_0 is called an *initial state for α*.

[3] More precisely, this is the foreground part of the state, the background part is the canonical model of the background theory $\mathcal{T}$.

Note that, given a trace α and an initial state s_0 for it, the state where the condition is reached is reproducible by simply executing α starting from s_0. This provides a cheap mechanism to check if a trace produced by a solver is indeed a witness. In a typical model program, the initial state is uniquely determined by an initial assignment to state variables, so the initial state witness is not relevant.

The *bounded reachability formula* for a given model program P, step bound k and reachability condition φ is:

$$\begin{aligned} \mathit{Reach}(P,\varphi,k) \stackrel{\text{def}}{=} I_P \wedge \Big(\bigwedge_{0\le i<k} P[i]\Big) \wedge \Big(\bigvee_{0\le i\le k} \varphi[i]\Big) \\ P[i] \stackrel{\text{def}}{=} \bigvee_{f\in A_P} \Big(\mathit{action}[i] = f(f_1[i],\ldots,f_n[i]) \wedge G_P^f[i] \\ \bigwedge_{v\in V_P^f} v[i+1] = t_v^f[i] \bigwedge_{v\in V_P\setminus V_P^f} v[i+1] = v[i]\Big) \end{aligned}$$

where an expression $E[i]$ denotes E where each state variable and parameter variable has been given index i if $i > 0$. A *skip* action has the action rule $(\mathit{true}, \emptyset)$. We use the following Theorem from [36].

Theorem 1. *Let P be a model program that includes a skip action, $k \ge 0$ a step bound and φ a reachability condition. Then $\mathit{Reach}(P,\varphi,k)$ is satisfiable if and only if φ is reachable in P within k steps. Moreover, if M satisfies $\mathit{Reach}(P,\varphi,k)$, let $M_0 = \{v \mapsto v^M\}_{v\in V_P}$, let $a_i = \mathit{action}[i]^M$ for $0 \le i < k$, and let α be the sequence $(a_i)_{i<k}$. Then α is a reachability trace for φ and M_0 is an initial state for α.*

3 One step reachability

The bounded reachability problem of model programs is undecidable in the general case. In this section we pin down various minimal cases of the undecidability with respect to certain background theories. In all cases it is enough to restrict the reachability bound and the number of action symbols to 1, i.e. the undecidability arises already using a single step and a single action symbol. We call it the *one step reachability problem*. In Section 4 we argue that these undecidable cases are minimal in some sense.

First, we define a theory $\mathit{TS}(\mathcal{A})$ that extends a given theory $\mathcal{A}$ (for example Presburger arithmetic) with *tuples* and *sets*. It is assumed that the language of $\mathcal{A}$ does not include the new symbols. It is convenient to

$$
\begin{array}{ll}
\textit{Basic elements}: & E ::= T_{\mathcal{A}} \mid \langle E, \ldots, E\rangle \mid \pi_i(E) \mid x \mid \mathit{ite}(F, E, E) \\
\textit{Sets of basic elements}: & S ::= \{E \mid_{\overline{x}} F\} \mid \emptyset \mid S \cup S \mid S \cap S \mid S \setminus S \mid v \mid \mathit{ite}(F, S, S) \\
\textit{Formulas}: & F ::= F_{\mathcal{A}} \mid \neg F \mid F \wedge F \mid F \vee F \mid \forall x\, F \mid \exists x\, F \mid \\
 & \quad E = E \mid S \subseteq S \mid S = S \mid E \in S
\end{array}
$$

Fig. 1. Well-formed expressions in $TS(\mathcal{A})$. The theory $\mathcal{A}$ has terms $T_{\mathcal{A}}$ and Formulas $F_{\mathcal{A}}$. It is assumed that all terms in $T_{\mathcal{A}}$ have sort $\mathbb{A}$. Set variables are denoted by v and basic variables (tuple variables or variables of sort $\mathbb{A}$) are denoted by x. The grammar omits sorts (type annotations) for ease of readability, but it is that of standard many-sorted first order logic. For example in a set operation term $s_1 \diamond s_2$, it is assumed that both s_1 and s_2 have the same sort (so sets contain only homogeneous elements), in an element-of atom $t \in s$ it is assumed that if the sort of t is σ then the sort of s is $\{\sigma\}$, a tuple (t_1, t_2) has the sort $\sigma_1 \times \sigma_2$ provided that t_i has sort σ_i, etc.

restrict the set of all possible expressions of $TS(\mathcal{A})$ to a set of well-formed expressions that are shown in Figure 1. When considering a formula of $TS(\mathcal{A})$ as defined in Figure 1, it is assumed that by default all set variables are *existentially quantified*, i.e. have an outermost existential quantifier. We write $TS(\mathcal{A})$ both for the class of expressions as defined in Figure 1, as well as the axioms of $TS(\mathcal{A})$.

The axioms of $TS(\mathcal{A})$ include the axioms of $\mathcal{A}$, the axioms for tuples stating that for each arity k the k-tuple constructor is a free constructor, axioms for set union, set intersection, element-of relation, subset relation, and the extensionality axiom for sets. Given a model $\mathfrak{A}$ of $TS(\mathcal{A})$, i.e., a structure $\mathfrak{A}$ in the language of $TS(\mathcal{A})$ that is a model of the axioms of $TS(\mathcal{A})$, the *comprehension term* $s = \{t(\overline{x}) \mid_{\overline{x}} \varphi(\overline{x})\}$, where t and φ may include parameters, has the interpretation $s^{\mathfrak{A}}$ in $\mathfrak{A}$ such that $\mathfrak{A} \models \forall y(y \in s^{\mathfrak{A}} \leftrightarrow \exists \overline{x}(t(\overline{x}) = y \wedge \varphi(\overline{x})))$ which is well-defined due to the extensionality axiom: $\forall v\, w(\forall y(y \in v \leftrightarrow y \in w) \rightarrow v = w)$.

Example 2. Let $\mathcal{P}$ be Presburger arithmetic. The following is a *range expression*, in $TS(\mathcal{P})$: $\{z \mid x \leq z \wedge z \leq y\}$ where we omit the z from $|_z$. We often use the abbreviation $\{x..y\}$ for a range from x to y. The following is a *direct product* $v \times w$ between two sets v and w: $\{\langle x, y\rangle \mid x \in v \wedge y \in w\}$.

Note that, not all well-formed $TS(\mathcal{P})$ expressions can be used in a model program, in a model program all expressions are quantifier free and each set comprehension variable has a finite range.

Theorem 2. *One can effectively associate a deterministic 2-register machine M with a formula $halts_M(m, n)$ in $TS(\mathcal{P})$ with integer parameters m and n, such that M halts on (m, n) if and only if $halts_M(m, n)$ holds.*

```
type Config = (Integer, Integer, Integer)
steps as Set of (Integer,Config,Config)
length as Integer
[Action] halts_M(m as Integer, n as Integer)
  require valid_M(m, n, steps, length)
```

Fig. 2. Model program P_M.

Proof (Outline). Let $STEP_M(\langle i, m, n\rangle, \langle i', m', n'\rangle)$ be the Presburger program formula for M as defined in [6, Theorem 2.1.15], where i, m, n and i', m', n' denote the current and the next configuration of the 2-register machine. Let $1 \ldots k$ be the instructions of M and assume that M is such that the initial instruction is 1 and the final instruction is $k > 1$ and when the final instruction is reached then both registers are zero. Let $halts_M$ be the following formula:

$$
\begin{aligned}
&halts_M(m, n) \stackrel{\text{def}}{=} \exists s \exists l(valid_M(m, n, s, l)) \\
&valid_M(m, n, s, l) \stackrel{\text{def}}{=} \\
&\quad s = \{\langle j, x, y\rangle \mid \langle j, x, y\rangle \in s \wedge STEP_M(x, y) \wedge 1 \leq j \wedge j \leq l\} \wedge \\
&\quad \{\langle \pi_0(z), \pi_1(z)\rangle \mid z \in s\} \cup \{\langle l, \langle k, 0, 0\rangle\rangle\} = \\
&\quad \{\langle 1, \langle 1, m, n\rangle\rangle\} \cup \{\langle \pi_0(z) + 1, \pi_2(z)\rangle \mid z \in s\}
\end{aligned}
$$

The statement is now straightforward to prove through an argument similar to *shifted pairing* [17, Theorem 15]. ⊠

The following is an immediate consequence of the proof of Theorem 2.

Corollary 1. *TS($\mathcal{P}$) is undecidable. Undecidability arises already for formulas of the form $\exists v \exists x \varphi$, where φ is quantifier free and uses at most three unnested comprehensions.*

The construction of $halts_M$ in Theorem 2 shows that comprehensions together with pairing (or tuples) leads to undecidability of the one step reachability problem, because $valid_M$ can be used as an enabling condition of an action as illustrated in Figure 2, and the halting problem of 2-register machines is undecidable.

Only a small fragment of Presburger arithmetic is needed. In particular, divisibility by a constant is not needed. The proof of the theorem does not change if M is assumed to be a Turing machine (assume M has two input symbols and the configuration (i, m, n) represents a snapshot of M where i is the finite state of M, m represents the tape content to the left of the tape head and n represents the tape content to the right

of the tape head), only the construction of `STEP` is different. However, in that case one needs to express divisibility by 2 to determine the input symbol represented by the lowest bit of the binary representation of m or n, which can be encoded using an additional existential quantifier.

Another consequence of the construction in Theorem 2 is that decidability of the bounded reachability problem cannot in general be obtained by fixing the model program or by limiting the number of set variables (without disallowing them).

Corollary 2. *There is a fixed model program P_u over TS($\mathcal{P}$) with one set-valued state variable, one integer-valued state variable, and an action symbol with two integer-valued parameters, such that the following problem is undecidable: given an action a, decide if a is enabled in P_u.*

Proof. Let M_u be a 2-register machine that is *universal* in the following sense, given a Turing machine M and an input v (over a fixed alphabet), let $\ulcorner M, v \urcorner$ be an effective encoding of M and v as an input for M_u, so that M_u accepts $\ulcorner M, v \urcorner$ if and only if M accepts v. Such a 2-register machine exists and can be constructed effectively [21, Theorem 7.9]. Let P_u be like P_{M_u} in Figure 2. Let M be a Turing machine and v an input for M. Then $\texttt{halts}_{M_u}(\ulcorner M, v \urcorner)$ is enabled in P_u iff (by Theorem 2) M_u halts on $\ulcorner M, v \urcorner$ iff M accepts v. ⊠

A *basic* value or sort is a non-set value or sort. A parameter or state variable is *basic* if its sort is basic.

Definition 4. A model program is *basic* if all of its action parameters are basic, each state variable is either basic or a set of basic elements, and set-sorted state variables are initialized with expressions that contain no set-sorted state variables.

Example 3. The model program P_u in Corollary 2 is not basic because the initial value of `steps` is undefined. The following model program on the other hand is basic, where `STEP` and `k` are the same as above.

```
[Action] halts(maxCounter as Integer, l as Integer)
  let steps = {(j,(i,m,n),(i',m',n')) | i,i' in {1..k}, j in {1..l},
               m,n,m',n' in {1..maxCounter}, STEP((i,m,n),(i',m',n'))}
  require {(j,x) | (j,x,y) in steps} union {(l,(k,0,0))} =
          {(1,(1,m,n))} union {(j+1,y) | (j,x,y) in steps}
```

It seems as if it is possible to express the halting problem just using bounded reachability of basic model programs. This is not the case as is shown in Section 4. Intuitively, a comprehension adds "too many" elements.

An extension of basic model programs that leads to undecidability of the one step reachability problem is if we allow *set cardinality*. We can then express integer multiplication as follows, given two (non-negative) integers m and n: $m \cdot n \stackrel{\text{def}}{=} |\{1..m\} \times \{1..n\}|$. Also, if we allow *bag comprehensions* we can define the cardinality of a set s as $|s| \stackrel{\text{def}}{=} \{\!\{0 | x \in s\}\!\}[0]$. Either of these extensions allows us to effectively encode diophantine equations (e.g. $5x^2y + 6z^3 - 7 = 0$ is a diophantine equation). Let $p(\overline{x})$ be a diophantine equation and let $\mathtt{P}(\overline{x})$ be an action whose enabling condition is the encoding of $p(\overline{x})$. Then $\mathtt{P}(\overline{n})$ is enabled iff $\overline{n}$ is an integer solution for $p(\overline{x})$. The problem of deciding whether a diophantine equation has an integer solution is known as *Hilbert's 10th problem* and is undecidable [28].

4 Bounded reachability of basic model programs

We show that the bounded reachability problem of *basic* model programs over a background $TS(\mathcal{A})$ is decidable provided that $Th(\mathcal{A})$ is decidable, where $Th(\mathcal{A})$ is the closure of $\mathcal{A}$ under entailment. We say that $Th(\mathcal{A})$ is decidable, if for an arbitrary closed first-order formula φ in the language of $\mathcal{A}$ it is decidable whether $\varphi \in Th(\mathcal{A})$.

The proof has two steps. First, we show that there is a fragment of $TS(\mathcal{A})$ formulas, denoted by $TS(\mathcal{A})_{\prec}$, for which the validity or satisfiability problem reduces effectively to $\mathcal{A}$, by showing that there is an effective equivalence preserving mapping from formulas in $TS(\mathcal{A})_{\prec}$ to formulas in $\mathcal{A}$. Second, we show that the bounded reachability problem of basic model programs over $TS(\mathcal{P})$ reduces to (satisfiability in) $TS(\mathcal{P})_{\prec}$. Let $\mathcal{A}$ be fixed. Let $V(\varphi)$ denote the collection of all set variables that occur in a formula φ over $TS(\mathcal{A})$.

Definition 5. A $TS(\mathcal{A})$ formula φ is in $TS(\mathcal{A})_{\prec}$ (also called *stratified*) if

- φ has the form $\psi \wedge \bigwedge_{v \in V(\varphi)} v = S_v$, and
- the relation $\prec \stackrel{\text{def}}{=} \{(w, v) | v \in V(\varphi), w \in V(S_v)\}$ is well-founded.

The equation $v = S_v$ is called the *definition of* v in φ.

Theorem 3. *$TS(\mathcal{A})_{\prec}$ reduces effectively to $\mathcal{A}$.*

Proof (Outline). The definition of $TS(\mathcal{A})_{\prec}$ is equivalent to the following construction in the case when all tuples are required to be flat. Let L_0 be the language of $\mathcal{A}$ and let $\mathcal{A}_0 = \mathcal{A}$. Given L_i and $\mathcal{A}_i$, create L_{i+1} and $\mathcal{A}_{i+1}$ as follows: expand L_i with a relation symbol R_φ of arity n for each L_i-formula $\varphi(x_1, \ldots, x_n)$ and add the definition $\forall \overline{x}(R_\varphi(\overline{x}) \leftrightarrow \varphi(\overline{x}))$

to $\mathcal{A}_i$. Now $TS(\mathcal{A})_\prec$ corresponds to $\bigcup_i \mathcal{A}_i$ as follows. Due to the well-founded ordering, each set variable v with the definition $v = \{\langle \overline{x} \rangle \mid_{\overline{x}} \varphi(\overline{x})\}$ corresponds to a relation symbol R_φ. Given a formula φ in $TS(\mathcal{A})_\prec$, it corresponds thus to a formula φ_k in $\mathcal{A}_k$ for some k. The statement follows by using the theorem of the existence of definitional expansions [20, Theorem 2.6.4] to reduce φ_{i+1} in L_{i+1} to an equivalent φ_i in L_i. ⊠

It follows that $TS(\mathcal{P})_\prec$ is decidable. We also get the following corollary that is the main result of this section.

Corollary 3. *Bounded reachability of basic model programs over $TS(\mathcal{P})$ is decidable.*

Proof. Let P be a basic model program over $TS(\mathcal{P})$ let φ be a reachability condition, and let k be a step bound. It is easy to see that $\psi = \mathit{Reach}(P, \varphi, k)$ can be written as a stratified $TS(\mathcal{P})$ formula: First, we can assume that there is only one action symbol (with a specific parameter that identifies a particular action). Since P is basic, the initial value of each state variable $v_{(0)}$ must be defined. In each step formula for step i, the value $v_{(i+1)}$ is given a definition that uses only variables or parameters from state i and parameters are basic. The definition can be written on a form that uses *ite* and is a top level equation of the generated formula. The only variables that are not given definitions are parameters, but all parameters are basic. Satisfiability of ψ in the language that includes the state variables reduces to entailment of the existential closure of ψ from $TS(\mathcal{P})$, which by Theorem 3, reduces to $\mathcal{P}$ and is thus decidable. ⊠

General integer arrays and array read and write operations are, strictly speaking, not in the $TS(\mathcal{P})$ fragment but can easily be encoded using tuples and comprehensions. For example, given an array variable v from integers to integers with the default value 0, encode it as the graph $\tilde{v}$ of v. The relation $\mathit{Read}(\tilde{v}, l, x)$ that holds when $v[l] = x$, can be defined through $\mathit{Read}(\tilde{v}, l, x) \stackrel{\text{def}}{=} \mathit{ite}(\{x\} = \{\pi_1(y) \mid y \in \tilde{v} \wedge \pi_0(y) = l\}, \mathit{true}, x = 0)$ and the corresponding write operation $\mathit{Write}(\tilde{v}, l, x)$ can be defined through $\mathit{Write}(\tilde{v}, l, x) \stackrel{\text{def}}{=} \{y \mid y \in \tilde{v} \wedge \pi_0(y) \neq l\} \cup \{(l, x)\}$. Using this encoding one can for example transform the *Credits* model program in Example 1 into an equivalent model program over $TS(\mathcal{P})$.

5 Implementation

We use the state of the art Z3 SMT solver for the implementation of bounded model checking. The initial implementation was described in [36].

We have extended this work in several aspects, including support for comprehensions with multiple comprehension variables and non-invertible comprehension expressions, bag(multiset) support etc., all of which make use of the iterated model refinement technique (explained in the following paragraphs). This is possible due to the fact that model programs are executable, so the feasibility of traces provided by the solver can be checked.

While, in principle, the traces could be executed directly on the model program via the ASML compiler, we use the approach to translate them to C# and executed the traces on C# code. This provides several benefits: we can conveniently use .Net API's for reflection, we can add auxiliary methods for evaluating and saving intermediate results for pinpointing error locations (in case an erroneous trace is provided by the solver) etc. Additionally, this eases the adoption of other languages for describing model programs. For example, NModel [30] uses C# as the modelling language. In this case, we would only need to provide a parser from C# to the internal abstract syntax to be able to use the framework.

The refinement loop works as follows. A trace provided by the Z3 solver is executed step by step on the generated program via reflection, and after each step it is checked whether the state given in the model matches the actual state (simply by comparing the variable values as assigned by the solver to the values in the actual state). If it does not match, we know at which action the mismatching state was reached. By examining the statements in the action we can check which of the axioms was not instantiated correctly and on which variables, consequently pinpointing the exact error source. The interpretation on this operation can then be fixed, by adding new formulas to the original model formula, giving explicit instantiations of the "misinterpreted" axiom on each index term. The new formula can be sent back to Z3 and a new trace obtained, which might again be erroneous (on some other axiom application), in which case it is again fixed and the refinement loop continues. This technique helps us circumvent SMT solvers' difficulties in coping with quantifiers. The approach is similar to CEGAR [9] (counter example guided abstraction refinement), the main difference being that we do not refine the level of abstraction, but instead lazily instantiate axioms in case their use has not been triggered during proof search.

We make use of the iterative refinement in several cases. In comparison to [36], we have added support for comprehensions which include more than 1 variable, and for the case where the element term is not invertible. In this case we rewrite the comprehension into formulas $\forall \overline{y}(\varphi[\overline{x}] \rightarrow t[\overline{x}] \in$

s') and $\forall y(y \in s' \rightarrow \exists \overline{x}(y = t[\overline{x}] \wedge \wedge \varphi[\overline{x}]))$. Existential quantifiers can be eliminated from the latter by skolemization. During the refinement loop, the two axioms are instantiated for specific index terms if needed.

Similar approach is used when extending the framework with bag support. While adding the bag axioms to Z3 is straightforward (for example the definition of bag union is $\forall x, s_1, s_2.((s_1 \uplus s_2)[x] \equiv s_1[x] + s_2[x])$), traces given by Z3 might be incorrect. (Quantifiers are implemented via pattern matching in Z3, so axioms are instantiated only if the particular pattern is encountered during proof search. If the pattern is not encountered, the axiom never gets used.) In this case, we can again use the model checking technique to pinpoint the source of error, and refine the model. This procedure is complete in case of integer bags.

6 Related work

The *unbounded* reachability problem for model programs without comprehensions and with parameterless actions is shown to be undecidable in [13], where it is called the hyperstate reachability problem. General reachability problems for transition systems are discussed in [33] where the main results are related to guarded assignment systems. A guarded assignment system is a union of guarded assignments or update rules. Detailed proofs of the theorems in this paper are given in [38]. The case when $\mathcal{A} = \mathcal{P}$ in Theorem 3 is related to decidable extensions of $\mathcal{P}$ that are discussed in [4].

The decidable fragment BAPA [26] is an extension of Boolean algebra with $\mathcal{P}$. The sets in BAPA are finite and bounded by a maximum size and the cardinality operator is allowed, which unlike for $TS(\mathcal{P})_{\prec}$, does not enable encoding of multiplication. Comprehensions are not possible and the element-of relation is not allowed, i.e. integers and sets can only be related through the cardinality operator. A decidable fragment of bag (multiset) constraints combined with summation constraints are considered in [31] where summation constraints can be used to express set cardinality (without using bag cardinality that is also included in the fragment). A related fragment of integer linear arithmetic with a star operator is considered in [32].

In [8] a decision procedure for an array fragment is introduced and in [36] it is shown that this decision procedure can be applied to the bounded reachability problem of a subclass of model programs. However, the fragment in [8] does not allow expressions that include universally quantified variables, other than the variable itself, to occur in array read operations.

Consequently, comprehensions where the comprehension expression is not invertible are not covered in [36]. In [19] another fragment of arrays is considered that allows universal variables in array read expressions that relate consecutive elements or talk about periodic properties.

The full fragment $TS(\mathcal{P})$ is also part of the data structures that are allowed in the Jahob verification system [7]. Formulas in this fragment are translated in Jahob to standard first-order formulas that can be proven using a resolution theorem prover.

A technique for translating common comprehension expressions (such as *sum* and *count*) into verification conditions is presented in [27] within the Spec# verification system that uses Boogie to generate verification conditions for SMT solvers [3]. The system does not support arbitrary set comprehension expressions as terms but allows axioms that enable explicit definitions of sets.

The reduction of the theories of arrays, sets and multisets to the theory of equality with uninterpreted function symbols and linear arithmetic is used in [24] for constructing interpolants for these theories. This work is based on the results of [25], where it is shown that the quantifier-free theories of arrays, sets and multisets can be reduced to quantifier-free theories of uninterpreted symbols with equality, constructors and Presburger arithmetic.

Using SAT for bounded reachability of transition systems was introduced in in [5] and the extension to SMT was introduced in [1]. Besides Z3 [10], other SMT solvers that support arrays are described in [2, 35]. The formula encoding we use [36] into SMT follows the same scheme but does not unwind comprehensions and makes the action label explicit.

Our quantifier elimination scheme is inspired by [8], and refines it by using model-checking to implement an efficient incremental saturation procedure on top of the SMT solver. The work here extends the work in [36] through support for set comprehensions with multiple comprehension variables and non-invertible comprehension expressions, as well as bag (multi-set) axioms. A recent application of the quantifier elimination scheme has been pursued by [23] in the context of railway control systems.

The following problems have not been addressed yet. Bounded reachability of model programs that use nested comprehensions, including for example sets and bags, is interesting for analysis of general purpose algorithms, see e.g. [18]. Given the (computational) complexity of $\mathcal{A}$, what is the complexity of $TS(\mathcal{A})_{\prec}$? It seems that a $TS(\mathcal{A})_{\prec}$ formula can be exponentially more succinct than the corresponding $\mathcal{A}$ formula. So, the complexity of $TS(\mathcal{P})_{\prec}$ could thus be $2^{2^{2^{cn}}}$, since the complexity of $\mathcal{P}$ is

$2^{2^{cn}}$ [12]. The proper instantiation of array indices and avoidance of false models generated by an SMT solver, due to the inherent incompleteness of the triggering mechanism of universally quantified axioms, is an important open problem in the general case.

Acknowledgement We thank Nikolaj Bjørner for support with Z3. We also thank the anonymous referees for their helpful comments and suggestions. The second author received support from the Estonian Doctoral School in ICT, the EITSA Tiger University Plus programme and the Estonian Association of Information Technology and Telecommunications (ITL).

References

1. A. Armando, J. Mantovani, and L. Platania. Bounded model checking of software using SMT solvers instead of SAT solvers. In A. Valmari, editor, *SPIN*, volume 3925 of *LNCS*, pages 146–162. Springer, 2006.
2. A. Armando, S. Ranise, and M. Rusinowitch. A rewriting approach to satisfiability procedures. *Inf. Comput.*, 183(2):140–164, 2003.
3. M. Barnett, B.-Y. E. Chang, R. DeLine, B. Jacobs, and K. R. M. Leino. Boogie: A modular reusable verifier for object-oriented programs. In *FMCO 2005*, volume 4111 of *LNCS*, pages 364–387. Springer, 2006.
4. A. Bès. A survey of arithmetical definability, A tribute to Maurice Boffa, Special Issue of Belg. Math. Soc., pages 1–54, 2002.
5. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS'99*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.
6. E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer, 1997.
7. C. Bouillaguet, V. Kuncak, T. Wies, K. Zee, and M. Rinard. On using first-order theorem provers in the Jahob data structure verification system. Technical Report MIT-CSAIL-TR-2006-072, Massachusetts Institute of Technology, November 2006.
8. A. R. Bradley, Z. Manna, and H. B. Sipma. What's decidable about arrays? In *VMCAI'06*, volume 3855 of *LNCS*, pages 427–442. Springer, 2006.
9. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In E. A. Emerson and A. P. Sistla, editors, *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.
10. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, (TACAS'08)*, LNCS. Springer, 2008.
11. L. M. de Moura, H. Rueß, and M. Sorea. Lazy theorem proving for bounded model checking over infinite domains. In *Proc. of CADE '02*, volume 2392 of *LNCS*, pages 438–455. Springer, 2002.
12. M. J. Fischer and M. O. Rabin. Super-exponential complexity of Presburger arithmetic. In *SIAMAMS*, pages 27–41, 1974.
13. W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes. Generating finite state machines from abstract state machines. *SIGSOFT Softw. Eng. Notes*, 27(4):112–122, 2002.
14. W. Grieskamp, D. MacDonald, N. Kicillof, A. Nandan, K. Stobie, and F. Wurden. Model-based quality assurance of Windows protocol documentation. In *ICST'08*, Lillehammer, Norway, April 2008.

15. Y. Gurevich. *Specification and Validation Methods*, chapter Evolving Algebras 1993: Lipari Guide, pages 9–36. Oxford University Press, 1995.
16. Y. Gurevich, B. Rossman, and W. Schulte. Semantic essence of AsmL. *Theor. Comput. Sci.*, 343(3):370–412, 2005.
17. Y. Gurevich and M. Veanes. Logic with equality: partisan corroboration and shifted pairing. *Inf. Comput.*, 152(2):205–235, 1999.
18. Y. Gurevich, M. Veanes, and C. Wallace. Can abstract state machines be useful in language theory? *Theor. Comput. Sci.*, 376(1):17–29, 2007.
19. P. Habermehl, R. Iosif, and T. Vojnar. What else is decidable about arrays? In R. Amadio, editor, *FoSSaCS'08*, LNCS. Springer, 2008.
20. W. Hodges. *Model theory*. Cambridge Univ. Press, 1995.
21. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.
22. J. Jacky, M. Veanes, C. Campbell, and W. Schulte. *Model-based Software Testing and Analysis with C#*. Cambridge University Press, 2008.
23. S. Jacobs and V. Sofronie-Stokkermans. Applications of hierarchical reasoning in the verification of complex systems. *ENTCS*, 174(8):39–54, 2007.
24. D. Kapur, R. Majumdar, and C. G. Zarba. Interpolation for data structures. In *SIGSOFT FSE 2006*, pages 105–116. ACM, 2006.
25. D. Kapur and C. G. Zarba. A reduction approach to decision procedures, 2006.
26. V. Kuncak, H. H. Nguyen, and M. Rinard. An algorithm for deciding BAPA: Boolean algebra with Presburger arithmetic. In R. Nieuwenhuis, editor, *CADE 2005*, volume 3632 of *LNAI*, pages 260–277. Springer, 2005.
27. R. Leino and R. Monahan. Automatic verification of textbook programs that use comprehensions. In *FTfJP 2007*, Berlin, Germany, July 2007.
28. Y. V. Matiyasevich. *Hilbert's tenth problem*. MIT Press, 1993.
29. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *ACM Trans. Program. Lang. Syst.*, 1(2):245–257, 1979.
30. NModel. http://www.codeplex.com/NModel, public version released May 2008.
31. R. Piskac and V. Kuncak. Decision procedures for multisets with cardinality constraints. In *VMCAI*, volume 4905 of *LNCS*, pages 218–232. Springer, 2008.
32. R. Piskac and V. Kuncak. On Linear Arithmetic with Stars. Technical Report LARA-REPORT-2008-005, EPFL, 2008.
33. T. Rybina and A. Voronkov. A logical reconstruction of reachability. In *PSI 2003*, volume 2890 of *LNCS*, pages 222–237. Springer, 2003.
34. SMB2. url:http://msdn2.microsoft.com/en-us/library/cc246482.aspx, 2008.
35. A. Stump, C. W. Barrett, D. L. Dill, and J. R. Levitt. A decision procedure for an extensional theory of arrays. In *LICS'01*, pages 29–37. IEEE, 2001.
36. M. Veanes, N. Bjørner, and A. Raschke. An SMT approach to bounded reachability analysis of model programs. In *FORTE'08*, LNCS. Springer, 2008.
37. M. Veanes, C. Campbell, W. Grieskamp, W. Schulte, N. Tillmann, and L. Nachmanson. Model-based testing of object-oriented reactive systems with Spec Explorer. In R. Hierons, J. Bowen, and M. Harman, editors, *Formal Methods and Testing*, volume 4949 of *LNCS*, pages 39–76. Springer, 2008.
38. M. Veanes, A. Saabas, and N. Bjørner. Bounded reachability of model programs. Technical Report MSR-TR-2008-81, Microsoft Research, May 2008.