Automatic Verification of Recursive Procedures with one Integer Parameter

Ahmed Bouajjani [a] Peter Habermehl [a] Richard Mayr [b,*]

[a] *LIAFA - Université Denis Diderot - Case 7014 - 2, place Jussieu, F-75251 Paris Cedex 05. France.*

[b] *Department of Computer Science, Albert-Ludwigs-University Freiburg, Georges-Koehler-Allee Geb. 051, D-79110 Freiburg, Germany.*

Abstract

Context-free processes (BPA) have been used for dataflow-analysis in recursive procedures with applications in optimizing compilers [5]. We introduce a more refined model called BPA($\mathbb{Z}$) that can model not only recursive dependencies, but also the passing of an integer parameter to subroutines. Moreover, this parameter can be tested against conditions expressible in Presburger-arithmetic. This new and more expressive model can still be analyzed automatically. We define $\mathbb{Z}$-input 1-CM, a new class of one-counter machines that take integer numbers as input, to describe sets of configurations of BPA($\mathbb{Z}$). We show that the *Post** (the set of successors) of a set of BPA($\mathbb{Z}$)-configurations described by a $\mathbb{Z}$-input 1-CM can be effectively constructed. The *Pre** (set of predecessors) of a regular set can be effectively constructed as well. However, the *Pre** of a set described by a $\mathbb{Z}$-input 1-CM cannot be represented by a $\mathbb{Z}$-input 1-CM in general and has an undecidable membership problem. Then we develop a new temporal logic based on reversal-bounded counter machines that can be used to describe properties of BPA($\mathbb{Z}$) and show that the model-checking problem is decidable.

Key words: Verification, Model Checking, Context-free Processes

---

* An extended abstract of this paper appeared in the proceedings of MFCS 2001. This work was partially supported by the European Commission (Project Advance IST 1999-29082)

* Corresponding author.

Email addresses: `abou@liafa.jussieu.fr` (Ahmed Bouajjani), `haberm@liafa.jussieu.fr` (Peter Habermehl), `mayrri@informatik.uni-freiburg.de` (Richard Mayr).

1 Introduction

Besides their classical use in formal language theory, pushdown automata have recently gained importance as an abstract process model for recursive procedures. Algorithms for model checking pushdown automata have been presented in [2,1,10,3]. Reachability analysis for pushdown automata is particularly useful in formal verification. Polynomial algorithms for reachability analysis have been presented in [1] and further optimized in [4]. For most purposes in formal verification it is sufficient to consider BPA ('Basic Process Algebra'; also called context-free processes), the subclass of pushdown automata without a finite control. BPA have been used for dataflow-analysis in recursive procedures with applications in optimizing compilers [5].

The weakness of BPA is that it is not a very expressive model for recursive procedures. It can model recursive dependencies between procedures, but not the passing of data between procedures or different instances of a procedure with different parameters.

Example 1 *Consider the following abstract model of recursive procedures P, Q, R, S and F, which take an integer number as argument: ($x|y$ means "x divides y").*

$$
\begin{array}{llll}
P(x): & \textit{If } x \geq 16 & Q(x): & \textit{If } 2|x \textit{ then } R(x) \\
 & \qquad \textit{If } 8|x \textit{ then } Q(x+1) & & \textit{else } S(x+1) \\
 & \qquad \textit{else } P(x-2) & & \\
 & \textit{else } F(x) & &
\end{array}
$$

If one starts by calling procedure P (with any parameter) then procedure R will never be called, because P never calls Q with an even number as parameter. However, a BPA model for these procedures cannot detect this.

Thus, we define a new more expressive model called BPA($\mathbb{Z}$) that extends BPA with an integer parameter. Procedures are now called with an integer parameter that can be tested, modified and passed to subroutines. We limit ourselves to one integer parameter, because two would give the model full Turing power and make all problems undecidable. BPA($\mathbb{Z}$) is a compromise between expressiveness and automatic analysability. On the one hand it is much more expressive than BPA and can model more aspects of full programs. On the other hand it is still simple enough such that most verification problems about BPA($\mathbb{Z}$) stay decidable. For the verification of safety properties, it is particularly useful to have a symbolic representation of sets of configurations and to be able to effectively construct representations of the Pre^* (the set of predecessors) and the $Post^*$ (the set of successors) of a given set of configurations. While finite automata suffice for describing sets of configurations of BPA, a more expressive formalism is needed for BPA($\mathbb{Z}$). We define $\mathbb{Z}$-input 1-CM, a new class of one-counter machines that take integer numbers as input, to describe sets of configurations of BPA($\mathbb{Z}$). We show that the *Post** (the set of successors) of a set described by a $\mathbb{Z}$-input 1-CM can be effectively constructed. The *Pre** (the set of predecessors) of a regular set can be effectively constructed as well. However, the *Pre** of a set described by a $\mathbb{Z}$-input 1-CM cannot be represented by a $\mathbb{Z}$-input 1-CM in general and has an undecidable membership problem.

We develop a new temporal logic based on reversal-bounded counter machines that can be used to describe properties of BPA($\mathbb{Z}$). By combining our result on the constructibility of the *Post** with some results by Ibarra et al. on reversal bounded counter machines [6,7] we show that the model-checking problem is decidable.

2 BPA($\mathbb{Z}$)

We define BPA($\mathbb{Z}$), an extension of BPA, as an abstract model for recursive procedures with an integer parameter.

Definition 2 *A n-ary Presburger predicate $P(k_1, \ldots, k_n)$ is an expression in Presburger-arithmetic of type boolean (i.e., the outermost operator is a logical operator or quantifier) that contains exactly n free variables $k_1, \ldots, k_n$ of type integer.*

Definition 3 *We define* integer symbol sequences *(ISS) to describe configurations of processes. ISS are finite sequences of the form $X_1(k_1)X_2(k_2)\ldots X_n(k_n)$, where the X_i are symbols from a given finite set and the $k_i \in \mathbb{Z}$ are integers. (The brackets are mere 'syntactic sugar' and can be omitted.) Greek letters $\alpha, \beta, \ldots$ are used to denote ISS. The constant ϵ denotes the empty sequence.*

Definition 4 *Let $Act = \{\epsilon, a, b, c, \ldots\}$ and $Const = \{\epsilon, X, Y, Z, \ldots\}$ be disjoint sets of* actions *and* process constants*, respectively. A BPA($\mathbb{Z}$) (α, Δ) is given by an initial configuration α (where α is an ISS) and a finite set Δ of conditional rewrite rules of the form*

$$X(k) \xrightarrow{a} X_1(e_1)X_2(e_2)\ldots X_n(e_n), \quad P(k)$$

where

- *$X \in Const$, $a \in Act$, k is a free variable of type integer.*
- *$\forall i \in \{1, \ldots, n\}. X_i \in Const$.*

- *For every $i \in \{1,\ldots,n\}$ e_i is an expression of one of the following two forms:*
  - $e_i = k_i$ *for some constant* $k_i \in \mathbb{Z}$, *or*
  - $e_i = k + k_i$ *for some constant* $k_i \in \mathbb{Z}$.
- $P(k)$ *is a unary Presburger predicate.*

Note that n can be 0. In this case the rule has the form $X(k) \stackrel{a}{\longrightarrow} \epsilon, \quad P(k)$. We denote the finite set of constants used in Δ by $Const(\Delta)$ and the finite set of actions used in Δ by $Act(\Delta)$. These rewrite rules induce a transition relation on ISS by prefix-rewriting as follows: For any α we have $X(q)\alpha \stackrel{a}{\longrightarrow} X_1(q_1)X_2(q_2)\ldots X_n(q_n)\alpha$ if there is a rewrite rule

$$X(k) \stackrel{a}{\longrightarrow} X_1(e_1)X_2(e_2)\ldots X_n(e_n), \quad P(k)$$

such that the following conditions are satisfied.

- $P(q)$
- *If* $e_i = k_i$ *then* $q_i = k_i$.
- *If* $e_i = k + k_i$ *then* $q_i = q + k_i$.

The Presburger predicates can be used to describe side conditions for the application of rules, e.g., the rule

$$X(k) \stackrel{a}{\to} Y(k-7)Z(k+1), \quad 3|k \wedge k \geq 8$$

can only be applied to ISS starting with $X(q)$ where q is at least 8 and divisible by 3. Furthermore, we can use Presburger predicates to express rules with constants on the left side, e.g., the rule $X(5) \stackrel{a}{\to} Y(2)Z(17)$ can be expressed by $X(k) \stackrel{a}{\to} Y(2)Z(17), \quad k=5$. In the following we sometimes use rules with constants on the left side as a shorthand notation.

Remark 5 *If one extends the model BPA($\mathbb{Z}$) by allowing two integer parameters instead of one, it becomes Turing-powerful, because it can simulate a Minsky 2-counter machine.*
If one extends the model by allowing multiplication and division on the one integer parameter, it becomes Turing-powerful as well. This is because in this case one can encode two counters into one by Gödel-coding. Two counters that hold numbers n_1 and n_2 are represented by one counter holding $2^{n_1}3^{n_2}$.

Definition 6 *We say that a BPA($\mathbb{Z}$) is in* normal form *if it only contains the following three types of rules:*

$$X(k) \stackrel{a}{\longrightarrow} X_1(e_1)X_2(e_2), \quad P(k)$$
$$X(k) \stackrel{a}{\longrightarrow} Y(e), \quad P(k)$$

$$X(k) \xrightarrow{a} \epsilon, \quad P(k)$$

where e, e_1, e_2 *are expressions and* $P(k)$ *is a unary Presburger predicate as in Def. 4.*

We call the rules of the third type decreasing *and the first two types* nondecreasing.

Remark 7 *It is easy to see that general BPA(*$\mathbb{Z}$*) can be simulated by BPA(*$\mathbb{Z}$*) in normal form with the introduction of some auxiliary constants. Long rules are split into several short rules. For example the long rule* $X(k) \stackrel{a}{\to} Y(k+1).Z(k-2).W(k+7)$ *is replaced by* $X(k) \stackrel{a}{\to} X'(k).W(k+7)$ *and* $X'(k) \stackrel{\epsilon}{\to} Y(k+1).Z(k-2)$*. If one is only interested in the set of reachable configurations of the original BPA(*$\mathbb{Z}$*) then one has to filter out the intermediate configurations that contain auxiliary constants. It will turn out in Section 4 that this is possible. We will show that the set of reachable configurations of a BPA(*$\mathbb{Z}$*) can be represented by a* $\mathbb{Z}$*-input 1-CM (a special type of one-counter machine) that is closed under synchronization with finite automata. Thus, to filter out the intermediate configurations it suffices to synchronize with the finite automaton that accepts exactly all sequences not containing auxiliary constants.*

It is clear that a BPA($\mathbb{Z}$) can simulate a 1-counter machine. However, the set of reachable configurations of a BPA($\mathbb{Z}$) cannot be described by a normal 1-counter machine.

Example 8 *Consider the BPA(*$\mathbb{Z}$*) with just one rule* $X(k) \stackrel{a}{\to} X(k+1)X(k)$ *and initial state* $X(0)$*. The set of reachable configurations are all decreasing sequences of the form* $X(n)X(n-1)X(n-2)\dots X(0)$ *for any* $n \in \mathbb{N}$*. The language consisting of these sequences cannot be accepted by a normal 1-counter machine, no matter how the integer numbers are coded (e.g., in unary coding or in binary as sequences of 0 and 1). The reason is that one cannot test the equality of the counter against the input without losing the content of the counter during the test.*

The central problem in this paper is to compute a representation of the set of reachable states of a BPA($\mathbb{Z}$).

Definition 9 *Let* Δ *be the set of rules of a BPA(*$\mathbb{Z}$*) and* L *a language of ISS (describing configurations of the BPA(*$\mathbb{Z}$*)). By* $Post^*_\Delta(L)$ *we denote the set of all successors (reachable configurations) of elements of* L *w.r.t.* Δ*.* $Post^*_\Delta(L) = \{\beta \mid \exists \alpha \in L.\, \alpha \to^*_\Delta \beta\}$*. By* $Pre^*_\Delta(L)$ *we denote the set of all predecessors of elements of* L *w.r.t.* Δ*.* $Pre^*_\Delta(L) = \{\alpha \mid \exists \beta \in L.\, \alpha \to^*_\Delta \beta\}$*.*

3 Automata

We define several classes of automata that are used in our constructions.

Definition 10 *An alternating pushdown automaton (APDA for short) is a triple* $\mathcal{P} = (P, \Gamma, \Delta_A)$ *where* P *is a finite set of* control locations, Γ *is a finite* stack alphabet, Δ_A *is a function that assigns to each element of* $P \times \Gamma$ *a positive (i.e. negation free) boolean formula over elements of* $P \times \Gamma^*$.

We can add an accepting state *accept* to P and designate an initial state $q_0 \in P$. Then the language $L(\mathcal{P}) \subseteq \Gamma^*$ *accepted* by $\mathcal{P}$ is defined as the set of initial stack contents for which $\mathcal{P}$ starting in q_0 accepts. An alternating 1-counter machine is a special case of an APDA, because the counter can be simulated by the pushdown stack.

Pushdown counter automata (PCA) have been introduced by Ibarra in [6].

Definition 11 *A pushdown counter automaton (PCA) [6] is a pushdown automaton that is augmented with a finite number of reversal-bounded counters (containing integers). A counter is reversal bounded iff there is a fixed constant* k *s.t. in any accepting computation the counter can change at most* k *times between increasing and decreasing.*

Now we define a new class of 1-counter machines with infinite input. These $\mathbb{Z}$-input 1-counter machines consider whole integer numbers as one piece of input and can compare them to constants, or to the internal counter without changing the counter's value. Additionally, they have several other useful features like Presburger-tests on the counter. $\mathbb{Z}$-input 1-counter machines will be used in Section 4 to represent sets of reachable configurations of BPA($\mathbb{Z}$).

Definition 12 *A* $\mathbb{Z}$*-input 1-counter machine* M *is described by a finite set of states* Q*, an initial state* $q_0 \in Q$*, a final state* $accept \in Q$*, a non-accepting state* $fail \in Q$*, and a counter* c *that contains a (possibly negative) integer value. The initial configuration is given by* q_0 *and some initial counter value. It reads pieces of input of the form* $S(i)$ *where* S *is a symbol out of a given finite set and* $i \in \mathbb{Z}$ *is an integer number. The instructions have the following form:*

(1) $(q:\ c := c+1; \mathsf{goto}\ q')$
(2) $(q:\ c := c-1; \mathsf{goto}\ q')$
(3) $(q:\ \mathsf{If}\ c \geq 0\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$.
(4) $(q:\ \mathsf{If}\ c = 0\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$.
(5) $(q:$ *Read input* $S(i)$. $\mathsf{If}\ S = X\ \mathsf{and}\ i = K\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$.
(6) $(q:$ *Read input* $S(i)$. $\mathsf{If}\ S = X\ \mathsf{and}\ i = c\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$.
(7) $(q:\ \mathsf{If}\ P(c)\ \mathsf{then\ goto}\ q'\ \mathsf{else\ goto}\ q'')$, *where* P *is a unary Presburger*

predicate.

where $X \in Const$ is a symbol constant and $K \in \mathbb{Z}$ is an integer constant.

$\mathbb{Z}$-input 1-counter machines can be nondeterministic, i.e., there can be several instructions at the same control-state. Each transition arc to a new control-state can be labeled with an atomic action. The language $L(M)$ accepted by a machine M is the set of ISS which are read by M in a run from the initial configuration to the accepting state.

In the following we use several shorthand notations for operations which can be encoded by the standard operations above. We use $c := c + j$ (incrementing the counter by a constant j), $c := j$ (setting the counter to a given constant j) and the operation $guess(c)$ (setting the counter to a nondeterministically chosen integer).

It is now easy to see that the set of reachable states of Example 8 can be described by the following $\mathbb{Z}$-input 1-counter machine with initial configuration $(q_0, 0)$:

q_0 : $guess(c)$; goto q_1

q_1 : Read input $S(i)$. If $S = X$ and $i = c$ then goto q_2 else goto $fail$

q_2 : $c := c - 1$; goto q_1

q_2 : If $c = 0$ then goto $accept$ else goto $fail$

While instructions of type 6 (integer input) do increase the expressive power of 1-counter machines, this is not the case for instructions of type 7 (Presburger tests). The following lemma shows that instructions of type 7 can be eliminated from $\mathbb{Z}$-input or alternating 1-counter machines if necessary. We use them only as a convenient shorthand notation.

Lemma 13 *For every (alternating) ($\mathbb{Z}$-input) 1-counter machine M with Presburger tests (i.e., instructions of type 7), an equivalent (alternating) ($\mathbb{Z}$-input) 1-counter machine M' without Presburger tests can be effectively constructed (Equivalent means $L(M) = L(M')$).*

PROOF. Any Presburger formula can be written in a normal form that is a boolean combination of linear inequalities and tests of divisibility. As we consider only Presburger formulae with one free variable, it suffices to consider tests of the forms $c \geq k$, $c \leq k$ and $k|c$ for constants $k \in \mathbb{Z}$. Let K be the set of constants k used in these tests. K is finite and depends only on the Presburger predicates used in M. Let $K' = \{k_1, \ldots, k_m\} \subseteq K$ be the finite set of constants used in divisibility tests. For every control-state q of M we define a set of control-states of M' of the form $(q, j_1, \ldots, j_m)$ where $j_i \in \{0, \ldots, k_i - 1\}$ for every $i \in \{1, \ldots, m\}$. Now M' simulates the computation of M in such a way that M' is in a state $(s, j_1, \ldots, j_m)$ iff M is in state s and $j_i = c \mathsf{\ mod\ } k_i$. For example if $K' = \{2, 5\}$ then the step $(s, n) \stackrel{c:=c+1}{\longrightarrow} (s', n+1)$ of M yields e.g. the step $((s, 1, 2), n) \stackrel{c:=c+1}{\longrightarrow} ((s', 0, 3), n+1)$ of M'. The divisibility tests thus become trivial in M', because this information is now encoded in the control-states of M'. The linear inequality tests are even easier to eliminate. For example the test $c \geq 5$ can be done by decrementing the counter by 5, testing for ≥ 0 and re-incrementing by 5. Thus, the Presburger tests can be eliminated from M'. $\square$

It is only a matter of convention if a $\mathbb{Z}$-input 1-CM reads the input from left to right (the normal direction) or from right to left (accepting the mirror image as in the example above). It is often more convenient to read the input from right to left (e.g., in Section 5), but the direction can always be reversed, as shown by the following lemma.

Lemma 14 *Let M be a $\mathbb{Z}$-input 1-CM with initial configuration (q_0, k_0) (with $k_0 \in \mathbb{Z}$) and final state q_f that reads the input from right to left. A $\mathbb{Z}$-input 1-CM M' can be constructed that reads the input from left to right and accepts the same language as M.*

PROOF. (sketch) M' has the same control-states as M plus a new initial state q_0' and a new final state q_f'. M' starts in configuration $(q_0', 0)$. It guesses a value for its counter and goes to q_f. Then it does the computation of M in reverse (reading the input from left to right) until it reaches q_0. It tests if the counter has value k_0. If yes, it goes to q_f' and accepts. If no, then it doesn't accept. $\square$

4 Constructing Post*

In this section we prove the following theorem:

Theorem 15 *Let Δ be a set of BPA($\mathbb{Z}$) rules and M a $\mathbb{Z}$-input 1-counter machine. Then a $\mathbb{Z}$-input 1-counter machine M' with $L(M') = Post^*_\Delta(L(M))$ can be effectively constructed.*

To prove this theorem we generalize the proof of a theorem in [1] which shows that the $Post^*$ of a regular set of configurations of a pushdown automaton is regular. This proof uses a saturation method, i.e. adding a finite number of transitions and states to the automaton representing configurations.

We cannot directly adapt this proof to BPA($\mathbb{Z}$), because process constants in a configuration can disappear for certain values of the parameter by applying decreasing rules. We show how to calculate a Presburger formula to characterize these values. This allows us to eliminate decreasing rules from Δ. This means that symbols produced by rules in some derivation can not disappear later. Then, we can apply the saturation method.

First, we show how to characterize the set $\{k \mid X(k) \rightarrow^*_\Delta \epsilon\}$ by a Presburger formula. We transform the set of rules Δ into an alternating one-counter machine and show that the set of initial values of accepting computations is effectively semilinear. This follows from a corollary of a theorem from [1] which states that the Pre^* of a more general model (alternating pushdown systems) is regular.

Theorem 16 *[1] Given an APDA $\mathcal{P}$ and a regular set of configurations $\mathcal{C}$, $pre^*(\mathcal{C})$ is regular and effectively constructible.*

With an APDA we can easily simulate an alternating one-counter machine with Presburger tests: First, we eliminate the Presburger tests with Lemma 13. Then, with the stack we can easily simulate the counter. We obtain the following:

Corollary 17 *Let M be an (alternating) one-counter machine with Presburger tests. The set of initial counter values for which a computation of M is accepting is effectively Presburger definable.*

Now we can prove the following two lemmas.

Lemma 18 *Let Δ be a set of BPA($\mathbb{Z}$) rules and X a process constant. Then a Presburger formula $P_X(k)$ with $\{k \mid P_X(k)\} = \{k \mid X(k) \rightarrow^*_\Delta \epsilon\}$ can be effectively constructed.*

PROOF. We construct an alternating one-counter machine M with Presburger tests such that: M with initial counter value k has an accepting computation iff $X(k) \rightarrow^*_\Delta \epsilon$. Then, we apply Corollary 17.

We construct M as follows: To each process constant Y of the BPA($\mathbb{Z}$) we associate a state q_Y in M. The initial state of the counter machine is q_X and it's final state *accept*. Each non-decreasing rewrite rule $X(k) \xrightarrow{a} X_1(e_1)X_2(e_2)\dots X_n(e_n), \quad P(k)$ is translated into a transition of the counter machine as follows: q_X goes to a conjunction of the states $q_{X_1}, \dots, q_{X_n}$ by testing $P(k)$ and changing the counter according to $e_1, \dots, e_n$. Each decreasing rule $X(k) \xrightarrow{a} \epsilon, \quad P(k)$ is translated into a transition of the counter machine from q_X to *accept* by testing $P(k)$. It is a clear, that a run of M with some initial counter value k is accepting iff $X(k)$ can disappear with rules of Δ. $\square$

Lemma 19 *Let Δ be a set of BPA($\mathbb{Z}$) rules and M a $\mathbb{Z}$-input 1-counter machine representing a set of configurations. Then, we can effectively construct a set of rules Δ' without decreasing rules and a $\mathbb{Z}$-input 1-counter machine M', such that $Post^*_\Delta(M) = Post^*_{\Delta'}(M')$.*

PROOF. The proof is done in two steps. First we construct a machine M' such that $L(M') = Post^*_{\Delta_d}(M)$ (where $\Delta_d \subseteq \Delta$ is the set of decreasing rules in Δ), i.e. M' is the closure of M under decreasing rules. Then, we construct Δ' without decreasing rules such that $Post^*_\Delta(M) = Post^*_{\Delta'}(Post^*_{\Delta_d}(M))$.

Let us first construct M': The machine M represents a set of configurations. M' represents the closure of this set under application of decreasing rules. We add new transitions from the initial state q_0 to each state q. These transitions are Presburger tests $P_q(k)$ which characterize all the initial counter values for which there is a path from q_0 to q such that all the process constants read on this path can disappear by applying rules of Δ. The formula $P_q(k)$ can be calculated by constructing a counter machine without input from M by replacing all transitions which read a process constant X by the corresponding Presburger test $P_X(k)$ (with Lemma 18) and then using Corollary 17. Clearly, $L(M') = Post^*_{\Delta_d}(M)$.

Now we construct Δ' which provides rules which non-deterministically guess what process constants will disappear later in a derivation. Obviously, only the first process constant from the left can disappear. Δ' contains all non-decreasing rules of Δ. Furthermore, for each conditional rewrite rule in Δ of the form

$$X(k) \xrightarrow{a} X_1(e_1)X_2(e_2), \quad P(k)$$

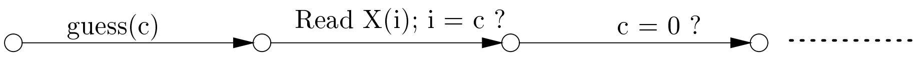
we add the rule

$$X(k) \xrightarrow{a} X_2(e_2), \quad P(k) \wedge P_{X_1}(e_1)$$

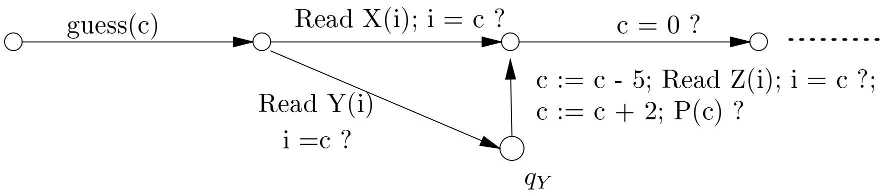
to Δ', where P_{X_1} is the Presburger formula of Lemma 18. Clearly, we have $Post^*_\Delta(M) = Post^*_{\Delta'}(Post^*_{\Delta_d}(M))$. $\square$

We use this lemma to prove Theorem 15. To construct a counter machine M' representing $Post^*_\Delta(M)$, given a counter machine M and a set of BPA($\mathbb{Z}$)

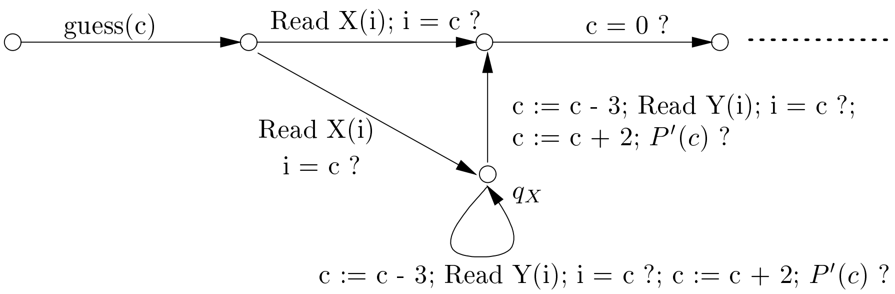
rules Δ, it suffices to consider Δ which doesn't contain decreasing rules. Before giving the detailed construction and it's correctness proof we explain the main idea with an example: Suppose we have a rule of the form $X(k) \xrightarrow{a} Y(k+3)Z(k-2), \quad P(k)$ in Δ and the automaton M is of the following form:



Notice that the counter is not tested before the input instruction. This is not a restriction (see Lemma 21). We add a new state q_Y for Y and transitions to M and obtain:



The transition going out of q_Y changes the counter value in such a way that if Y is read with parameter k then Z is read with a parameter $k-5$, where -5 is the difference between -2 and 3. Then, the transition restores the counter value to the value before application of the rule by adding 2 and tests $P(c)$. Now consider instead a rule $X(k) \xrightarrow{a} X(k+1)Y(k-2) \quad P'(k)$ in Δ. Following the same principle as before we add a state for X and transitions. This will create a loop:



It is clear that in this way we only add a finite number of states and transitions. In the following we give the detailed proof of our main theorem.

Proof of Theorem 15

First we show how to transform a $\mathbb{Z}$-input 1-CM into a special form:

Definition 20 *A $\mathbb{Z}$-input 1-CM M is said to be in special form iff*

- *M has one initial state q_0 from which there is an instruction $guess(c)$ going to q_1 and from q_1 there are only instructions reading input.*
- *no instructions go back to q_0 or q_1.*
- *all instructions from q_1 are of the form*
  (q : *Read input* $S(i)$. If $S = X$ and $i = c$ then goto q' else goto $fail$)
  or
  (q : *Read input* $S(i)$. If $S = X$ and $i \neq c$ then goto q' else goto $fail$)
  We call this instructions first input instructions.
- *all instructions with a test of the counter ($c \geq 0$ or $c = 0$) are of the form*
  (q : If $test(c)$ then goto q' else goto fail)

Machines in special form guess a counter value, read an input and only then can test the counter and continue. We can prove the following lemma:

Lemma 21 *A $\mathbb{Z}$-input 1-CM M with initial configuration (q_0, k_0) can be replaced by a $\mathbb{Z}$-input 1-CM M' in special form with an initial configuration (q'_0, k'_0) which accepts the same language.*

PROOF. Putting the test instructions into the required form is trivial. To construct a machine where no tests on the counter are done before an input, we have to characterize all the counter values which can be obtained by taking a path (without inputs) from state q_0 (with counter value k_0) to another state q'. We can construct a Presburger formula $P(c)$ for this (using a variation of Corollary 17). Then we can construct a machine with states q_0 and q_1 required by the special form (by putting the corresponding Presburger test behind each input instruction). At last, input instructions starting at q_1 of the form

(q_1 : Read input $S(i)$. If $S = X$ and $i = K$ then goto q' else goto $fail$)

can be replaced by

(q_1 : Read input $S(i)$. If $S = X$ and $i = c$ then goto q'' else goto $fail$)
(q'' : If $c = K$ then goto q'''else goto $fail$)
(q''' : $guess(c)$; goto q')

The same is done for instructions with inequations. Notice, that the behavior of the machine M' does not depend on the initial counter value k'_0. □

To prove Theorem 15 we have to show, that given a set Δ of BPA($\mathbb{Z}$) rules and a $\mathbb{Z}$-input 1-counter machine M we can construct a a $\mathbb{Z}$-input 1-counter machine M' with $L(M') = Post^*_\Delta(L(M))$. Because of Lemma 19 we can suppose that Δ does not contain decreasing rules. We suppose that rules of Δ are in normal form and that M is in special form (Lemma 21). We first give the construction of M' and then we prove that $L(M') = Post^*_\Delta(L(M))$.

Construction of M'

In the following we will omit the else-part of all the instructions (they all go to *fail*). The construction of M' is done by adding states and instructions to the machine M. The basic idea is the following: Each rule of Δ can replace a process constant read starting from the initial state of the automaton M by other process constants. Therefore, instructions have to be added to M. These instructions have to change also the counter in order to simulate correctly the change in the parameters. Therefore, the counter has to be changed in such a way, that after reading the symbols on the right-hand side of a rule its value is the same as before. The special form of M insures that the counter is not tested before the input instructions. Furthermore, because there are no decreasing rules, only symbols read in the first input instructions can be replaced.

Let q_0 be the initial state of M and q_1 the state after the initial state. To simplify the presentation we only show how to treat input instruction with test $i = c$. The same can be done for $i \neq c$. For each process constant X we add a new state q_X and an instruction

$$(q_1 : \text{ Read input } S(i). \textsf{ If } S = X \textsf{ and } i = c \textsf{ then goto } q_X \,).$$

Now, for each input rule of the form

$$(q_1 : \text{ Read input } S(i). \textsf{ If } S = X \textsf{ and } i = c \textsf{ then goto } q')$$

in M (including instructions added before) and a rule in Δ of the following forms, we add instructions to M to obtain M'.

- $X(k) \xrightarrow{a} X_1(k + k_1), \quad P(k)$:
  add one instruction

  $$(q_{X_1} : \ c := c - k_1; \textsf{ If } P(c) \textsf{ then goto } q')$$

- $X(k) \xrightarrow{a} X_1(k_1), \quad P(k)$:
  add two instructions (q_n is a new state)

$(q_{X_1}: \textsf{If } c = k_1 \textsf{ then goto } q_n)$
$(q_n: \textsf{guess}(c); \ \textsf{If } P(c) \textsf{ then goto } q')$

- $X(k) \xrightarrow{a} X_1(k+k_1)X_2(k+k_2), \quad P(k)$:
  add two instructions (q_n is a new state)

  $(q_{X_1}: c := c - k_1 + k_2; \ \text{Read input } S(i).$
  $\textsf{If } S = X_2 \textsf{ and } i = c \textsf{ then goto } q_n)$
  $(q_n: c := c - k_2; \ \textsf{If } P(c) \textsf{ then goto } q')$

- $X(k) \xrightarrow{a} X_1(k_1)X_2(k+k_2), \quad P(k)$:
  add three instructions (q_{n_1}, q_{n_2} are new states)

  $(q_{X_1}: \textsf{If } c = k_1 \textsf{ then goto } q_{n_1})$
  $(q_{n_1}: \textsf{guess}(c); \ \text{Read input } S(i). \textsf{ If } S = X_2 \textsf{ and } i = c \textsf{ then goto } q_{n_2})$
  $(q_{n_2}: c := c - k_2; \ \textsf{If } P(c) \textsf{ then goto } q')$

- $X(k) \xrightarrow{a} X_1(k+k_1)X_2(k_2), \quad P(k)$:
  add two instructions (q_n is a new state)

  $(q_{X_1}: c := c - k_1; \ \text{Read input } S(i).$
  $\textsf{If } S = X_2 \textsf{ and } i = k_2 \textsf{ then goto } q_n)$
  $(q_n: \textsf{If } P(c) \textsf{ then goto } q')$

- $X(k) \xrightarrow{a} X_1(k_1)X_2(k_2), \quad P(k)$:
  add three instructions (q_{n_1}, q_{n_2} are new states)

  $(q_{X_1}: \textsf{If } c = k_1 \textsf{ then goto } q_{n_1})$
  $(q_{n_1}: \text{Read input } S(i). \textsf{ If } S = X_2 \textsf{ and } i = k_2 \textsf{ then goto } q_{n_2})$
  $(q_{n_2}: \textsf{guess}(c); \ \textsf{If } P(c) \textsf{ then goto } q')$

It is obvious, that only a finite number of instructions are added. In M' loops containing the states q_X can be created by the construction.

Correctness of M'

We have to show that $Post^*(L(M)) = L(M')$.

First, $Post^*(L(M)) \subseteq L(M')$:
We show by induction that $Post^n(L(M)) \subseteq L(M')$ for all $n \in \mathbb{N}$. Base case: Obviously, we have $L(M) \subseteq L(M')$ because M' is obtained by adding states to M. Induction step: Consider an $\alpha \in Post^{n+1}(L(M))$. Then, there exists an $\alpha' \in Post^n(L(M))$ with $\alpha' \rightarrow_\Delta \alpha$. By induction hypothesis $\alpha' \in L(M')$. Now, the construction of M' insures that $\alpha \in L(M')$, because for each rule of Δ there are corresponding transitions which are added to obtain M'.

Second, $L(M') \subseteq Post^*(L(M))$:
We prove this by induction on the number of new instructions (added to M by the construction) taken in accepting runs of M'. Let $\alpha \in L(M')$. If no new instruction is taken in an accepting run of α, then $\alpha \in L(M)$ and therefore $\alpha \in Post^*(L(M))$. Now, suppose that an accepting run of α in M' takes some new instructions. These are all taken at the beginning of the run. α must be of the form $X_1(m)\alpha'$. The machine M' takes first the $guess(c)$ instruction and then an input instruction of the form

$$(q_1: \text{ Read input } S(i). \textsf{ If } S = X_1 \textsf{ and } i = c \textsf{ then goto } q_{X_1} \textsf{ else goto } fail)$$

Then, there are several cases to consider depending on the next instruction taken. We give the proof in detail for one case. All the others are done analogously. Suppose that the next instruction taken by the machine is

$$(q_{X_1}: \ c := c - k_1; \textsf{ If } P(c) \textsf{ then goto } q')$$

Therefore, $P(m - k_1)$ is true. By construction there is an instruction in M'

$$(q_1: \text{ Read input } S(i). \textsf{ If } S = X \textsf{ and } i = c \textsf{ then goto } q')$$

for some process constant X. Furthermore there is a rule $X(k) \xrightarrow{a} X_1(k + k_1), \quad P(k)$ in Δ. Because of $\alpha \in L(M')$ we have $X(m - k_1)\alpha' \in L(M')$ with an accepting run which takes less new instructions than the one for α. By induction hypothesis, $X(m - k_1)\alpha' \in Post^*(L(M))$ and furthermore $X(m - k_1)\alpha' \rightarrow_\Delta X(m)\alpha' = \alpha$. It follows, that $\alpha \in Post^*(L(M))$.

Thus Theorem 15 is proven. □

Remark 22 *While the language of reachable states of any BPA($\mathbb{Z}$) can be described by a $\mathbb{Z}$-input 1-CM, the converse is not true. Some $\mathbb{Z}$-input 1-CMs describe languages that cannot be generated by any BPA($\mathbb{Z}$). Consider the language*

$$\{X(k)Y(j_1)Y(j_2)\ldots Y(j_n)X(k) \mid k \in \mathbb{N}, n \in \mathbb{N}, j_1, \ldots, j_n \in \mathbb{N}\}$$

It is easy to construct a $\mathbb{Z}$-input 1-CM for this language (it just ignores the values of the j_i). However, no BPA($\mathbb{Z}$) generates this language, since it cannot guess the values of the arbitrarily many j_i without losing the value for k, which it needs again at the end.

The complexity of constructing a representation of Post* must be at least as high as the complexity of the reachability problem for BPA($\mathbb{Z}$). A special case of the reachability problem is the problem if the empty state ϵ is reachable from the initial state.

ϵ-REACHABILITY FOR BPA($\mathbb{Z}$)
Instance: A BPA($\mathbb{Z}$) Δ with initial state $X(0)$.
Question: $X(0) \to^* \epsilon$?

It is clear that for BPA($\mathbb{Z}$) with Presburger constraints the complexity of ϵ-reachability is at least as high as that of Presburger-arithmetic. Consider a closed (i.e., without free variables) Presburger formula P and a BPA($\mathbb{Z}$) with one rule $X(k) \to \epsilon, \quad P(k)$. Then we have $X(0) \to^* \epsilon$ iff P is true. Presburger-arithmetic is complete for the class $\bigcup_{k>1} TA[2^{2^{n^k}}, n]$ (see [9]), and thus requires at least doubly exponential time. Now we consider the restricted case of BPA($\mathbb{Z}$) without Presburger constraints.

Theorem 23 *The ϵ-reachability problem for BPA($\mathbb{Z}$) without full Presburger constraints, but with constants on the left sides of rules, is $\mathcal{NP}$-hard.*

PROOF. We reduce 3-SAT to ϵ-reachability. Let $Q := Q_1 \wedge \ldots \wedge Q_j$ be a boolean formula in 3-CNF with j clauses over the variables $x_1, \ldots, x_n$. We construct a BPA($\mathbb{Z}$) Δ with initial state $X(0)$ s.t. $X(0) \to^*_\Delta \epsilon$ iff Q is satisfiable. Let p_l be the l-th prime number. We encode an assignment of boolean values to $x_1, \ldots, x_n$ in a natural number x by Gödel coding, i.e., x_i is true iff x is divisible by p_i. The set of rules Δ is defined as follows:

$$
\begin{array}{lll}
X(k) & \to X(k+1) & \\
X(k) & \to Q_1(k+1).Q_2(k+1).\ldots.Q_j(k+1) & \\
Q_i(k) & \to X_l(k) & \text{if } x_l \text{ occurs in clause } Q_i. \\
Q_i(k) & \to \bar{X}_l(k) & \text{if } \bar{x}_l \text{ occurs in clause } Q_i. \\
X_l(k) & \to X_l(k-p_l) & \\
X_l(0) & \to \epsilon & \\
\bar{X}_l(k) & \to \bar{X}_l(k-p_l) & \\
\bar{X}_l(r) & \to \epsilon & \text{for every } r \in \{1, \ldots, p_l - 1\}
\end{array}
$$

The $X(k)$ is used to guess a number k that encodes an assignment to $x_1, \ldots, x_n$. If follows from the construction that $Q_i(k) \to^* \epsilon$ iff k encodes an assignment that makes clause Q_i true, $X_l(k) \to^* \epsilon$ iff k encodes an assignment where x_l is true and $\bar{X}_l(k) \to^* \epsilon$ iff k encodes an assignment where x_l is false. Thus we get $X(0) \to^* \epsilon$ iff Q is satisfiable. As the l-th prime number is $\mathcal{O}(l \cdot \log l)$, the size of Δ is $\mathcal{O}(jn + n^2 \log n)$. $\square$

5 The Constructibility of Pre*

In this section we show that the Pre^* of a regular set of configurations (w.r.t. a BPA($\mathbb{Z}$)) is effectively constructible. However, the Pre^* of a set of configurations described by a $\mathbb{Z}$-input 1-CM is not constructible. It is not even representable by a $\mathbb{Z}$-input 1-CM in general. Regular sets are given by finite automata. We define that finite automata ignore all integer input and are only affected by symbols. So, in the context of BPA($\mathbb{Z}$) we interpret the language $(ab)^*$ as $\{a(k_1)b(k'_1)\ldots a(k_n)b(k'_n) \mid n \in \mathbb{N}_0,\ \forall i.\, k_i, k'_i \in \mathbb{Z}\}$.

Theorem 24 *Let Δ be a BPA($\mathbb{Z}$) and R a finite automaton. Then a $\mathbb{Z}$-input 1-CM M can be effectively constructed s.t. $M = Pre^*_\Delta(R)$.*

PROOF. Every element in $Pre^*_\Delta(R)$ can be written in the form $\alpha X(k)\gamma$ where $\alpha \to^* \epsilon$, $X(k) \to^* \beta$ and $\beta\gamma \in R$. Thus there must exist a state r in R s.t. there is a path from the initial state r_0 of R to r labeled β and a path from r to a final state of R labeled γ. We consider all (finitely many) pairs (X, r) where $X \in Const(\Delta)$ and $r \in states(R)$. Let R_r be the finite automaton that is obtained from R by making r the only final state. We compute the set of integers k for which there exists a β s.t. $X(k) \to^* \beta$ and $\beta \in R_r$. First we compute the $\mathbb{Z}$-input 1-CM M_X that describes $Post^*(X(k))$ as in Theorem 15. Then we compute the product of M_X with R_r, which is again a $\mathbb{Z}$-input 1-CM. The set of initial values k for which $M_X \times R_r$ is nonempty is Presburger and effectively computable. Let $P_{X,r}$ be the corresponding unary Presburger predicate. Let R'_r be the finite automaton that is obtained from R by making r the initial state. We define $M_{X,r}$ to be the $\mathbb{Z}$-input 1-CM that behaves as follows: First it accepts $X(k)$ iff $P_{X,r}(k)$ and then it behaves like R'_r. Let M_ϵ be the $\mathbb{Z}$-input 1-CM that accepts all sequences α s.t. $\alpha \to^* \epsilon$. M_ϵ is effectively constructible, since for every symbol Y the set of k for with $Y(k) \to^* \epsilon$ is Presburger and effectively constructible by Lemma 18. Then finally we get

$$M = M_\epsilon \cdot \bigcup_{X,r} M_{X,r}$$

and $M = Pre^*_\Delta(R)$. $\square$

Now we consider the problem of the Pre^* of a set of configurations described by a $\mathbb{Z}$-input 1-CM.

<u>MEMBERSHIP IN Pre^* OF $\mathbb{Z}$-INPUT 1-CM</u>

Instance: A BPA($\mathbb{Z}$) Δ, a $\mathbb{Z}$-input 1-CM M and a state $X_0(0)$
Question: $X_0(0) \in Pre^*_\Delta(M)$?

Theorem 25 *Membership in Pre^* of $\mathbb{Z}$-input 1-CM is undecidable.*

PROOF. We reduce the undecidable halting problem for Minsky 2-counter machines (with both counters initially 0) to the membership in Pre^* of a $\mathbb{Z}$-input 1-CM. The first observation is that $X_0(0) \in Pre^*_\Delta(M)$ iff $Post^*_\Delta(X_0(0)) \cap L(M) \neq \emptyset$. Let M' be a Minsky 2-counter machine. We will define the BPA($\mathbb{Z}$) Δ and the $\mathbb{Z}$-input 1-CM M in such a way that each of them simulates a 1-counter machine and together they simulate the 2-counter machine M'.

We define the BPA($\mathbb{Z}$) Δ in such a way that it correctly simulates the part of the computation of M' that only affects the first counter c_1. The integer parameter is used to store the first counter c_1.

- For every instruction of M' of the form $(X: \ c_1 := c_1 + 1; \mathsf{goto}\ X')$ we have a rule $X(k) \to X'(k+1)X(k)$.
- For every instruction of M' of the form $(X: \ \mathsf{If}\ c_1 = 0\ \mathsf{then\ goto}\ X'\ \mathsf{else}\ c_1 := c_1 - 1; \mathsf{goto}\ X'')$ we have two rules $X(0) \to X'(0)X(0)$ and $X(k) \to X''(k-1)X(k), \quad k > 0$.
- For every instruction of M' of the form $(X: \ c_2 := c_2 + 1; \mathsf{goto}\ X')$ we have a rule $X(k) \to X'(k)X(k)$.
- For every instruction of M' of the form $(X: \ \mathsf{If}\ c_2 = 0\ \mathsf{then\ goto}\ X'\ \mathsf{else}\ c_2 := c_2 - 1; \mathsf{goto}\ X'')$ we have two rules $X(k) \to X'(k)X(k)$ and $X(k) \to X''(k)X(k)$.

In the last of these four cases the BPA($\mathbb{Z}$) guesses the successor state, because it knows nothing about the counter c_2. Thus, $Post^*_\Delta(X_0(0))$ contains all correct computation sequences of M' starting at the initial control state X_0 and initial counter value 0, but also some wrong ones (if it has guessed wrongly in the fourth case). These sequences are read from right to left.

Then we use the $\mathbb{Z}$-input 1-CM M to simulate the other part of the computation of M' which affects the second counter c_2. The counter of M is used to store the second counter c_2 of M' (which is initially 0). M ignores all integer input and only checks the symbols.

- For every instruction of M' of the form $(X: \ c_1 := c_1 + 1; \mathsf{goto}\ X')$ the machine M reads the input, but ignores it, goes to control-state X' and leaves the internal counter unchanged.
- For every instruction of M' of the form $(X: \ \mathsf{If}\ c_1 = 0\ \mathsf{then\ goto}\ X'\ \mathsf{else}\ c_1 := c_1 - 1; \mathsf{goto}\ X'')$ the machine M reads the input symbol, which is either X or X', and changes the control-state accordingly to X or X'. M ignores the integer input and leaves the internal counter unchanged.
- For every instruction of M' of the form $(X: \ c_2 := c_2 + 1; \mathsf{goto}\ X')$ the machine M increases the internal counter by 1 and goes to the control-state

X'.

- For every instruction of M' of the form $(X:\ \mathsf{If}\ c_2 = 0\ \mathsf{then\ goto}\ X'\ \mathsf{else}\ c_2 := c_2 - 1; \mathsf{goto}\ X'')$ the machine M checks if the internal counter is 0.
  · If the internal counter is 0, then it reads the input symbol and checks if it is X'. If yes, then it goes to the control-state X'. If no, then it stops and rejects. The internal counter is left unchanged. The integer input is ignored.
  · If the internal counter is > 0 then it decrements the internal counter by 1, reads the input symbol and checks if it is X''. If yes, then it goes to the control-state X''. If no, then it stops and rejects. The integer input is ignored.

The machine M only accepts in the final control-state X_f, which is also the final state of M'. As for the BPA($\mathbb{Z}$) above, these computation sequences of M are read from right to left. This is not a restriction by Lemma 14.

Together Δ and M simulate the computation of M'. Δ ensures that the computation step is correct when the first counter is concerned. M does the same for the second counter and ensures that only those sequences are accepted that end in the final state X_f of M'.

So we get that $X_0(0) \in Pre^*_\Delta(M) \iff Post^*_\Delta(X_0(0)) \cap L(M) \neq \emptyset \iff$ M halts, and thus the membership problem in Pre^* is undecidable. □

Theorem 25 does not automatically imply that the Pre^* of a $\mathbb{Z}$-input 1-CM (w.r.t. Δ) cannot be represented by a $\mathbb{Z}$-input 1-CM. It leaves the possibility that this $\mathbb{Z}$-input 1-CM is just not effectively constructible. (Cases like this occur, e.g., the set of reachable states of a classic lossy counter machine is semi-linear, but not effectively constructible [8].) However, the following theorem shows that the Pre^* of a $\mathbb{Z}$-input 1-CM is not a $\mathbb{Z}$-input 1-CM in general.

Theorem 26 *Let Δ be a BPA($\mathbb{Z}$) and M a $\mathbb{Z}$-input 1-CM. The set $Pre^*_\Delta(M)$ cannot be represented by a $\mathbb{Z}$-input 1-CM in general.*

PROOF. Let M' be the 2-counter machine that accepts if and only if the initial counter value in the first counter c_1 is a power of 2, i.e., 2^m for some positive integer m. Let Δ and M be defined as in the proof of Theorem 25.

We assume that $Pre^*_\Delta(M)$ could be represented by a $\mathbb{Z}$-input 1-CM and derive a contradiction. If there were a $\mathbb{Z}$-input 1-CM that represents $Pre^*_\Delta(M)$ then there would also exist a $\mathbb{Z}$-input 1-CM that represents $Pre^*_\Delta(M) \cap \{X_0(n) \mid n \in \mathbb{N}\} = \{X_0(n) \mid \exists m \in \mathbb{N}.\, n = 2^m\}$. This is a contradiction, because the set $\{n \mid \exists m \in \mathbb{N}.\, n = 2^m\}$ is not Presburger definable. □

6 The Logic and its Applications

We define a logic called ISL (Integer Sequence Logic) that can be used to verify properties of BPA($\mathbb{Z}$). It is interpreted over ISS (see Def. 3). We define a notion of satisfaction of an ISL formula by a BPA($\mathbb{Z}$) and show that the verification problem is decidable.

Let *const* denote the projection of ISS on sequences of constants obtained by omitting the integers; formally $const(X_1(k_1)X_2(k_2)\ldots X_m(k_m)) = X_1X_2\ldots X_m$. Then, the logic ISL is defined as follows:

Definition 27 *ISL formulae have the following syntax:*

$$F \;:=\; (A_1, \ldots, A_n, P)$$

where $A_1, \ldots, A_n$ are finite automata over an alphabet of process constants, and P is an $(n-1)$-ary Presburger predicate. Formulae are interpreted over sequences w of the form $X_1(k_1)X_2(k_2)\ldots X_m(k_m)$, where the satisfaction relation is defined as follows:

$w \models F$ iff there exist words $w_1, \ldots, w_n$, constants $Y_1, \ldots, Y_{n-1}$ and integers $k_1, \ldots, k_{n-1}$ s.t. $w = w_1Y_1(k_1)w_2Y_2(k_2)\ldots w_{n-1}Y_{n-1}(k_{n-1})w_n$ and

- *$\forall i \in \{1, \ldots, n-1\}.\, A_i$ accepts $const(w_i)Y_i$.*
- *A_n accepts $const(w_n)$ and $P(k_1, \ldots, k_{n-1})$ is true.*

The set of sequences which satisfy a formula F is given by $[\![F]\!] = \{w \mid w \models F\}$.

Intuitively, ISL formulae specify regular patterns (using automata) involving a finite number of integer values which are constrained by a Presburger formula. We use ISL formulae to specify properties on the configurations of the systems and not on their computation sequences, the typical use of specification logics in verification. For instance, when BPA($\mathbb{Z}$)'s are used to model recursive programs with an integer parameter, a natural question that can be asked is whether some procedure X can be called with some value k satisfying a Presburger constraint P. This can be specified by asking whether there is a reachable configuration corresponding to the pattern $Const^*X(k)Const^*$, where $P(k)$ holds. Using ISL formulae, we can specify more complex questions such as whether it is possible that the execution stack of the recursive program can contain two consecutive copies of a procedure with the same calling parameter. This corresponds to the pattern $Const^*X(k_1)(Const-\{X\})^*X(k_2)Const^*$, where $k_1 = k_2$.

The first result we show, is that we can characterize $[\![F]\!]$ by means of reversal bounded counter automata. However, elements of $[\![F]\!]$ are sequences over an infinite alphabet, since they may contain any integer. To characterize over a finite alphabet an element $w \in [\![F]\!]$ we can encode the integers in w *in unary*: a positive (resp. negative) integer k_i is replaced by k_i (resp. $-k_i$) occurrences of a symbol p_i (resp. n_i). Hence, given a set L of ISS, let $\widehat{L}$ denote the set of all sequences in L encoded in this way. We can characterize $\widehat{[\![F]\!]}$ with a reversal bounded counter automaton.

Lemma 28 *We can construct a reversal bounded counter automaton M over a finite alphabet Σ such that $\widehat{[\![F]\!]} = L(M)$.*

PROOF. The reversal bounded counter automaton M simulates sequentially the automata $A_1, \ldots, A_n$ in order to check if the input is of the correct regular pattern. After reading w_i (A_i has to be in an accepting state), the machine reads a sequence of symbols p_i or n_i and stores their length in corresponding reversal bounded counters. After the input has been completely read, the Presburger formula can be tested by using a finite number of other reversal bounded counters. □

Now, we define a notion of satisfaction between BPA($\mathbb{Z}$)'s and ISL formulae.

Definition 29 *Let (w_0, Δ) be a BPA($\mathbb{Z}$) with initial configuration w_0 and set of rules Δ. Let F be an ISL-formula. We define that (w_0, Δ) satisfies the formula F iff it has a reachable configuration that satisfies F. Formally*

$$(w_0, \Delta) \models F \iff \exists w \in Post^*_\Delta(w_0).\, w \models F$$

To prove the decidability of the verification problem $(w_0, \Delta) \models F$, for a given BPA($\mathbb{Z}$) (w_0, Δ) and a formula F we need the following definition and a lemma.

Definition 30 *Let L be a set of ISS. Then, $L|_k$ is the set of sequences w such that there exists a sequence $w' \in L$ with $k' \geq k$ integers such that w is obtained from w' by removing $k' - k$ integers and encoding the remaining integers in unary.*

Lemma 31 *Let (w_0, Δ) be a BPA($\mathbb{Z}$) with initial configuration w_0 and set of rules Δ. Then we can construct a PCA M such that $L(M) = Post^*_\Delta(w_0)|_k$.*

PROOF. First by Theorem 15 we construct a $\mathbb{Z}$-input 1-CM M that accepts $Post^*_\Delta(w_0)$. We construct a PCA from M by (1) using the pushdown store to encode the counter (2) choosing non-deterministically exactly k input values which are compared to the counter. For these comparisons we need k additional reversal bounded counters (to avoid losing the counter value). □

Theorem 32 *Let (w_0, Δ) be a BPA($\mathbb{Z}$) with initial configuration w_0 and set of rules Δ and $F = (A_1, \ldots, A_n, P)$ an ISL-formula. The problem $(w_0, \Delta) \models F$ is decidable.*

PROOF. Clearly, we have $Post^*_\Delta(w_0) \cap [\![F]\!] \neq \emptyset$ iff $\widehat{Post^*_\Delta(w_0)} \cap \widehat{[\![F]\!]} \neq \emptyset$, which is also equivalent to $\widehat{Post^*_\Delta(w_0)}|_{n-1} \cap \widehat{[\![F]\!]} \neq \emptyset$ since F cannot constrain more than $n-1$ integers. Then we show that $\widehat{Post^*_\Delta(w_0)}|_{n-1} \cap \widehat{[\![F]\!]} \neq \emptyset$ is decidable. This follows from Lemma 31, Lemma 28, the fact that the intersection of a CA language with a PCA language is a PCA language (Lemma 5.1 of [6]), and Theorem 5.2 of [6] which states that the emptiness problem of PCA is decidable. □

Finally, we consider another interesting problem concerning the analysis of BPA($\mathbb{Z}$)'s. When used to model recursive procedures, a natural question is to know the set of all the possible values for which a given procedure can be called. More generally, we are interested in knowing all the possible values of the vectors $(k_1, \ldots, k_n)$ such that there is a reachable configuration which satisfies some given ISL formula $F = (A_1, \ldots, A_{n+1}, P)$. We show that this set is effectively semilinear.

Theorem 33 *Let (w_0, Δ) be a BPA($\mathbb{Z}$) with initial configuration w_0 and set of rules Δ, and let F be an ISL formula. Then, the set*

$$\{(k_1, \ldots, k_n) \in \mathbb{Z}^n \,|\, \exists w = w_1 Y_1(k_1) \ldots w_n Y_n(k_n) w_{n+1} \in Post^*_\Delta(w_0).\, w \models F\}$$

is effectively semilinear.

PROOF. As in the proof of Theorem 32, we can construct a PCA which recognizes the language $\widehat{Post^*_\Delta(w_0)}|_n \cap \widehat{[\![F]\!]}$. Then, the result follows from the fact that the Parikh image of a PCA language is semilinear (see Theorem 5.1 of [6]). □

7 Conclusion

We have shown that BPA($\mathbb{Z}$) is a more expressive and more realistic model for recursive procedures than BPA. The price for this increased expressiveness is that a stronger automata theoretic model ($\mathbb{Z}$-input 1-CM) is needed to describe sets of configurations, while simple finite automata suffice for BPA. As a consequence the set of predecessors is no longer effectively constructible for BPA($\mathbb{Z}$) in general. However, the set of successors is still effectively constructible in BPA($\mathbb{Z}$) and thus many verification problems are decidable for BPA($\mathbb{Z}$), e.g., model checking with ISL. Thus, BPA($\mathbb{Z}$) can be used for verification problems like dataflow analysis, when BPA is not expressive enough. We expect that our results can be generalized to more expressive models (e.g., pushdown automata with an integer parameter), but some details of the constructions will become more complex.

References

[1] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: application to model checking. In *International Conference on Concurrency Theory (CONCUR'97)*, volume 1243 of *LNCS*. Springer Verlag, 1997.

[2] O. Burkart and B. Steffen. Pushdown processes: Parallel composition and model checking. In *CONCUR'94*, volume 836 of *LNCS*, pages 98–113. Springer Verlag, 1994.

[3] O. Burkart and B. Steffen. Model checking the full modal mu-calculus for infinite sequential processes. In *Proceedings of ICALP'97*, volume 1256 of *LNCS*. Springer Verlag, 1997.

[4] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proc. of CAV 2000*, volume 1855 of *LNCS*. Springer Verlag, 2000.

[5] J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *Proc. of FoSSaCS'99*, volume 1578 of *LNCS*, pages 14–30. Springer Verlag, 1999.

[6] O. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25:116–133, 1978.

[7] O. Ibarra, T. Bultan, and J. Su. Reachability analysis for some models of infinite-state transition systems. In *Proc. of CONCUR 2000*, volume 1877 of *LNCS*. Springer Verlag, 2000.

[8] R. Mayr. Undecidable problems in unreliable computations. In *Proc. of LATIN 2000*, volume 1776 of *LNCS*. Springer Verlag, 2000.

[9] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science: Volume A, Algorithms and Complexity.* Elsevier, 1990.

[10] I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.