Automatic Verification of Recursive Procedures with one Integer Parameter

Ahmed Bouajjani, Peter Habermehl, and Richard Mayr

LIAFA - Université Denis Diderot - Case 7014 - 2, place Jussieu,
F-75251 Paris Cedex 05. France. E-mail: {abou,haberm,mayr}@liafa.jussieu.fr
Fax: +33 1 44 27 68 49

Abstract. Context-free processes (BPA) have been used for dataflow-analysis in recursive procedures with applications in optimizing compilers [6]. We introduce a more refined model called BPA($\mathbb{Z}$) that can model not only recursive dependencies, but also the passing of integer parameters to subroutines. Moreover, these parameters 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.

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 [3, 1, 11, 4]. 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 [5]. 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 [6].

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").

$P(x)$: If $x \geq 16$
  If $8|x$ then $Q(x+1)$
  else $P(x-2)$
else $F(x)$

$Q(x)$: If $2|x$ then $R(x)$
  else $S(x+1)$

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 integer parameters. 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 [7, 8] 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 integer parameters.

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:*

  - $e_i = k_i$ *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* $X(k) \xrightarrow{a} \epsilon, \quad P(k)$*. 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* $X(q)\alpha \xrightarrow{a} X_1(q_1)X_2(q_2)\dots X_n(q_n)\alpha$ *if there is a rewrite rule* $X(k) \xrightarrow{a} X_1(e_1)X_2(e_2)\dots X_n(e_n), \quad P(k)$ *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 $X(k) \xrightarrow{a} Y(k-7)Z(k+1), \quad 3|k \wedge k \geq 8$ 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 $X(5) \xrightarrow{a} Y(2)Z(17)$ can be expressed by $X(k) \xrightarrow{a} Y(2)Z(17), \quad k = 5$. In the following we sometimes use rules with constants on the left side as a shorthand notation.

Definition 5. *We say that a BPA(*$\mathbb{Z}$*) is in* normal form *if it only contains the following three types of rules:*

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

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 6. 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 $X(k) \xrightarrow{a} Y(k+1).Z(k-2).W(k+7)$ is replaced by $X(k) \xrightarrow{a} X'(k).W(k+7)$ and $X'(k) \xrightarrow{\epsilon} Y(k+1).Z(k-2)$. 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.

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.
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 7. Consider the BPA(ℤ) with just one rule $X(k) \xrightarrow{a} X(k+1)X(k)$ 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(ℤ).

Definition 8. *Let Δ be the set of rules of a BPA(ℤ) and L a language of ISS (describing configurations of the BPA(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

Definition 9. *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 $P \times \Gamma^*$.*
An alternating 1-counter machine is a special case of an APDA, because the counter can be simulated by the pushdown stack.

Definition 10. *A pushdown counter automaton (PCA) [7] 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 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 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. ℤ-input 1-counter machines will be used in Section 4 to represent sets of reachable configurations of BPA(ℤ).

Definition 11. *A 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. The machine 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. $(q:$ If $c \geq 0$ then goto q' else goto $q'')$.
4. $(q:$ If $c = 0$ then goto q' else goto $q'')$.
5. $(q:$ *Read input* $S(i)$. If $S = X$ and $i = K$ then goto q' else goto $q'')$.
6. $(q:$ *Read input* $S(i)$. If $S = X$ and $i = c$ then goto q' else goto $q'')$.
7. $(q:$ If $P(c)$ then goto q' 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 7 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 if necessary. We use them only as a convenient shorthand notation.

Lemma 12. *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 that it accepts the same input (initial counter value), and the same language.)*
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 s of M we define a set of control-states of M' of the form $(s, j_1, \ldots, j_m)$ 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 $j_i = c \bmod k_i$. 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'. □

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 13. *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. □

4 Constructing Post*

Theorem 14. *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 automata 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 15. *[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 12. Then, with the stack we can easily simulate the counter. We obtain the following:

Corollary 16. *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.*

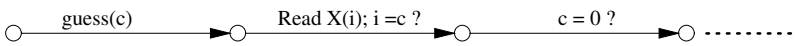
Now we can prove the following two lemmas.

Lemma 17. *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.*

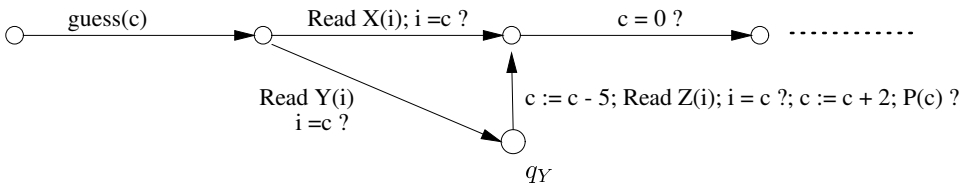
Proof. We construct an alternating one-counter machine M such that: M with initial counter value k has an accepting computation iff $X(k) \rightarrow^*_\Delta \epsilon$. Then, we apply Corollary 16.
We construct M as follows: To each process constant Y of the BPA($\mathbb{Z}$) we associate a state q_Y in M. The initial state of the counter machine is q_X and it's final state $accept$. Each non-decreasing rewrite rule $X(k) \xrightarrow{a} X_1(e_1)X_2(e_2)\ldots X_n(e_n), \quad P(k)$ is translated into a transition of the counter machine as follows: q_X goes to a conjunction of the states $q_{X_1}, \ldots, q_{X_n}$ by testing $P(k)$ and changing the counter according to $e_1, \ldots, e_n$. 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. □

Lemma 18. *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$^*_{\Delta'}(M')$.*

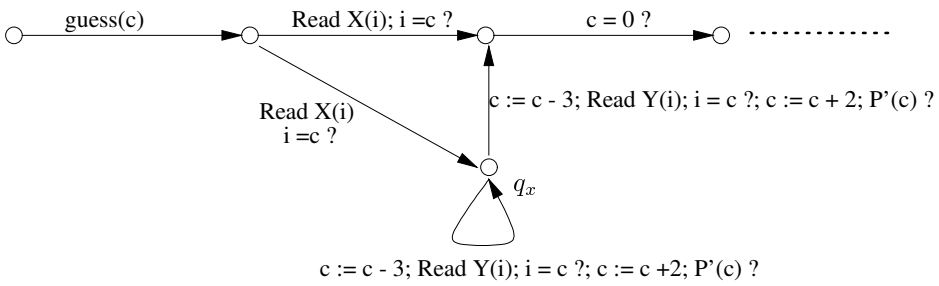
We use this lemma to prove Theorem 14. To construct a counter machine M' representing $Post^*_\Delta(M)$, given a counter machine M and a set of BPA($\mathbb{Z}$) rules Δ, it suffices to consider non-decreasing rules. We explain the main idea with an example: Suppose we have a rule of the form $X(k) \xrightarrow{a} Y(k+3)Z(k-2), \quad P(k)$ 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 full paper [2]). 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 $X(k) \xrightarrow{a} X(k+1)Z(k-2) \quad P'(k)$ 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. A lot of cases have to be considered. The details of the proof can be found in the full paper [2].

Remark 19. While the language of reachable states of any BPA(ℤ) can be described by a ℤ-input 1-CM, the converse is not true. Some ℤ-input 1-CMs describe languages that cannot be generated by any BPA(ℤ). Consider the language

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

It is easy to construct a ℤ-input 1-CM for this language (it just ignores the values of the j_i). However, no BPA(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(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(ℤ)

Instance: A BPA(ℤ) Δ with initial state $X(0)$.
Question: $X(0) \to^* \epsilon$?

It is clear that for BPA(ℤ) with Presburger constraints the complexity of ϵ-reachability is at least as high as that of Presburger-arithmetic. (Presburger-arithmetic is complete for the class $\bigcup_{k>1} TA[2^{2^{n^k}}, n]$ [10], and thus requires at least doubly exponential time). Now we consider the restricted case of BPA(ℤ) without Presburger constraints.

Theorem 20. *The ϵ-reachability problem for BPA(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 $Q := Q_1 \wedge \dots \wedge Q_j$ be a boolean formula in 3-CNF with j clauses over the variables $x_1, \dots, x_n$. Let p_l be the l-th prime number. We encode an assignment of values to $x_1, \dots, x_n$ 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:

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

As the l-th prime number is $\mathcal{O}(l \cdot \log l)$, the size of Δ is $\mathcal{O}(j + n^2 \log n)$. It is easy to see that $X(0) \to^* \epsilon$ iff Q is satisfiable. □

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 $\{a(k_1)b(k_1') \dots a(k_n)b(k_n') \mid n \in \mathbb{N}_0, \forall i.\, k_i, k_i' \in \mathbb{Z}\}$.

Theorem 21. *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 14. 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 17. Then finally we get

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

□

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

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

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

Theorem 22. *Membership in Pre^* of $\mathbb{Z}$-input 1-CM is undecidable.*

The proof is by reduction of the halting problem for Minsky 2-counter machines. It is given in the full paper [2]. Theorem 22 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 semilinear, but not effectively constructible [9].) However, the following theorem (proved in the full paper [2]) shows that the Pre^* of a $\mathbb{Z}$-input 1-CM is not a $\mathbb{Z}$-input 1-CM in general.

Theorem 23. *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.*

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 $const(X_1(k_1)X_2(k_2)\dots X_m(k_m)) = X_1X_2\dots X_m$. Then, the logic ISL is defined as follows:

Definition 24. *ISL formulae have the following syntax: $F := (A_1, \dots, A_n, P)$, where $A_1, \dots, A_n$ 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 $X_1(k_1)X_2(k_2)\dots X_m(k_m)$, where the satisfaction relation is defined as follows: $w \models F$ iff there exist words $w_1, \dots, w_n$, constants $Y_1, \dots, Y_{n-1}$ and integers $k_1, \dots, k_{n-1}$ s.t. $w = w_1Y_1(k_1)w_2Y_2(k_2)\dots w_{n-1}Y_{n-1}(k_{n-1})w_n$ and*

- *$\forall i \in \{1, \dots, n-1\}$. A_i accepts $const(w_i)Y_i$.*
- *A_n accepts $const(w_n)$ and $P(k_1, \dots, k_{n-1})$ 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 25. *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 machine 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(ℤ)'s and ISL formulae.

Definition 26. *Let (w_0, Δ) be a BPA(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(ℤ) (w_0, Δ) and a formula F we need the following definition and a lemma.

Definition 27. *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 28. *Let (w_0, Δ) be a BPA(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 14 we construct a 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 29. *Let (w_0, Δ) be a BPA(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 28, Lemma 25, the fact that the intersection of a CA language with a PCA language is a PCA language (Lemma 5.1 of [7]), and Theorem 5.2 of [7] which states that the emptiness problem of PCA is decidable. □

Finally, we consider another interesting problem concerning the analysis of BPA(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 30. *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*

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

is effectively semilinear.

Proof. As in the proof of Theorem 29, 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 [7]). □

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] A. Bouajjani, P. Habermehl, and R. Mayr. Automatic Verification of Recursive Procedures with one Integer Parameter Technical Report LIAFA, 2001. available at `www.informatik.uni-freiburg.de/~mayrri/`.

[3] 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.

[4] 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.

[5] 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, 2000.

[6] 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.

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

[8] 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, 2000.

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

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

[11] I. Walukiewicz. Pushdown processes: games and model checking. In *International Conference on Computer Aided Verification (CAV'96)*, volume 1102 of *LNCS*. Springer, 1996.