Push-Down Automata with Gap-Order Constraints

Parosh Abdulla, Mohamed Atig, Giorgio Delzanno, Andreas Podelski

Push-Down Automata with Gap-Order Constraints

Parosh Aziz Abdulla[1], Mohamed Faouzi Atig[1],
Giorgio Delzanno[2], and Andreas Podelski[3]

[1] Uppsala University
[2] University of Genova
[3] University of Freiburg

Abstract. We consider push-down automata with data (PDAD) that operate on variables ranging over the set of natural numbers. The conditions on variables are defined via gap-order constraint. Gap-order constraints allow to compare variables for equality, or to check that the gap between the values of two variables exceeds a given natural number. The messages inside the stack are equipped with values that are natural numbers reflecting their "values". When a message is pushed to the stack, its value may be defined by a variable in the program. When a message is popped, its value may be copied to a variable. Thus, we obtain a system that is infinite in two dimensions, namely we have a stack that may contain an unbounded number of messages each of which is equipped with a natural number. We present an algorithm for solving the control state reachability problem for PDAD based on two steps. We first provide a translation to the corresponding problem for context-free grammars with data (CFGD). Then, we use ideas from the framework of well quasi-orderings in order to obtain an algorithm for solving the reachability problem for CFGDs.

1 Introduction

Model checking has become one of the main techniques for algorithmic verification of computer systems. The original applications were found in context of finite-state systems, such as hardware circuits, where the behavior of the system can be captured by a finite state machine. In the last two decades, there has also been a large amount of work devoted to extending model checking so that its can handle models with *infinite* state spaces such as Petri nets, timed automata, push-down systems, counter automata, and channel machines. Recent works have considered systems that are infinite in *multiple* dimensions. For instance, many classes of timed protocols are *parameterized* (consist of unbounded numbers of components), and hence they can be naturally modeled by *timed Petri nets* [10]. Also, many message passing protocols have behaviors that are constrained by timing conditions, giving rise to *timed channel systems* [5].

In particular, Push-Down Automata (PDA) have been studied extensively as a model for the analysis of recursive programs (e.g., [12, 33, 23, 25]). The model of PDA has been extended to allow quantitative reasoning with respect to time [1] and probabilities [26, 24]. However, all existing models assume finite-state control, which means that variables in the program are assumed to range over finite domains. In this paper, we consider an extension of PDA, which we call PDAD, that strengthens the model in

two ways. First, in addition to the stack, a PDAD also operates on a number of variables ranging over the natural numbers. Furthermore, each message inside the stack is equipped with a natural number which represents its "value". Thus, we get a model that is possibly unbounded in two dimensions, namely we have an unbounded number of messages inside the stack each of which has an attribute that is a natural number. The operations allowed on the stack are the standard *push* and *pop* operations. However, when pushing a symbol to the stack, its value may be defined to be the value of a program variable. Also, when a message is popped, then its value may be copied to a variable. A PDAD allows comparing the values of variables according to the *gap-order* constraint system, where two variables may be tested for equality, or for checking that there is a minimal gap (defined by a natural number) between the values of the two variables. Also, a variable may be assigned a new arbitrary value, the value of another variable, or a value that is at least some (given) natural number larger than the value of another variable. In this manner, the model of PDAD subsumes two known models, namely that of PDA (which we get by removing the variables in the program and by neglecting the values of the symbols in the stack), and the model of *Integral Relational Automata* [15] (which we get by removing the stack).

In this paper, we show decidability of the control reachability problem for PDAD. Given a control (local) state of the automaton, we check whether the automaton reaches the state from its initial configuration. We solve the problem in two steps. We introduce a class of *Context-Free Grammars with Data* (CFGD). In a CFGD, each non-terminal has an arity. The grammar generates *terms* each of which is either a terminal or a non-terminal equipped with a tuple of natural number (as many as its arity). An application of a production rewrites a term to a *set* of terms. Such an application is constrained by the arguments of the involved non-terminals. The constraints are defined by gap-order conditions. For CFGD, we solve a reachability problem in which we ask whether it is possible to derive a set of terms each of which is a terminal belonging to a given set of terminals. In the first step of our method, we give a reachability analysis algorithm that solves the above mentioned problem for CFGDs.

The algorithm is based on a constraint representation of infinite sets of terms, and it is formulated within the framework of well structured transition systems [4, 6].

The second step of our method translates a given PDAD into a CFGD so as to exploit the corresponding reachability analysis procedure to solve control state reachability for PDADs.

To our knowledge our result yields a new decidable fragment of pushdown automata with data (see Section 10).

2 Preliminaries

In this section, we introduce some notations and definitions that we will use in the rest of the paper. We use $\mathbb{N}$ to denote the set of natural numbers.

We fix a finite set $\mathcal{V}$ of variables that range over $\mathbb{N}$. A *valuation* is a mapping $Val : \mathcal{V} \to \mathbb{N}$, i.e., it assigns a natural number to each variable. Given a variable $x \in \mathcal{V}$, a natural number $c \in \mathbb{N}$, and a valuation $Val : \mathcal{V} \to \mathbb{N}$, we use $Val[x \leftarrow c]$ to denote the valuation Val' defined as follows: $Val'(x) = c$, and $Val'(y) = Val(y)$ for all $y \in (\mathcal{V} \setminus \{x\})$.

A *renaming* is a mapping $Ren : \mathcal{V} \to \mathcal{V}$, i.e., it renames each variable to another one. A renaming *Ren* does not need to be injective, i.e., several variables may be renamed to the same variable by *Ren*. We say that *Ren* is a renaming for W if $Ren(x) \in W$ for all $x \in \mathcal{V}$.

For a set A, we use A^* to denote the set of finite words over A. We use ε to denote the empty word. For words $\alpha_1, \alpha_2 \in A^*$, we use $\alpha_1 \cdot \alpha_2$ to denote the concatenation of α_1 and α_2.

A *transition system* is a tuple $\langle \Upsilon, \gamma_{init}, \longrightarrow \rangle$ where Υ is a (potentially infinite) set of configurations, $\gamma_{init} \in \Upsilon$ is the initial configuration, and $\longrightarrow \subseteq \Upsilon \times \Upsilon$ is the transition relation. As usual, we write $\gamma \longrightarrow \gamma'$ to denote that $\langle \gamma, \gamma' \rangle \in \longrightarrow$, and use $\xrightarrow{*}$ to denote the reflexive transition closure of $\longrightarrow$. For a configuration $\gamma \in \Upsilon$ and a set $\Gamma \subseteq \Upsilon$ of configurations, we use $\gamma \xrightarrow{*} \Gamma$ to denote that $\gamma \xrightarrow{*} \gamma'$ for some $\gamma' \in \Gamma$.

3 Push-Down Automata with Data

In this section, we introduce *Push-Down Automata with Data* (PDAD) that are extensions of the classical model of Push-Down Automata (PDA). First, we define the model, then we define the operational semantics, i.e., the transition system induced by a PDAD, and finally we introduce the reachability problem. As in the case of a PDA a PDAD operates on an unbounded stack to which it can push (append) messages and from which it can pop (remove) message in last-in-first-out manner. The messages are chosen from a finite alphabet. PDADs extend PDAs in two ways. First, in addition to the stack, the automaton is equipped with a finite set of variables ranging over natural numbers. Second, each message inside the stack is equipped by a natural number that represents its "value". The allowed operations on variables are defined by the *gap-order* constraint system [15, 31]. More precisely, the model allows non-deterministic value assignment, copying the value of one variable to another, and assignment of a value v to some variable such that v is larger of at least a given natural number than the current value of another variable. The transitions may be conditioned by tests that compare the values of two variables for equality, or that give the minimal allowed gap between two variables. A *push* operation may copy the value of variable to the pushed message, and a *pop* operation may copy the value of the popped message to a variable.

Model. A PDAD $\mathcal{A}$ is a tuple $\langle Q, q_{init}, A, \Delta \rangle$ where Q is the finite set of states, $q_{init} \in Q$ is the initial state, A is the stack alphabet, and Δ is the transition relation. We remark that the stack alphabet is infinite since it consists of pairs $\langle a, \ell \rangle$ where a is taken from a finite set and ℓ is a natural number. A transition $\delta \in \Delta$ is a triple $\langle q_1, op, q_2 \rangle$ where $q_1, q_2 \in Q$ are states and *op* is an *operation* of one of the following forms: (i) *nop* is an empty operation that does not change the values of the variables or the content of the stack, (ii) $x \leftarrow *$ assigns non-deterministically an arbitrary value in $\mathbb{N}$ to the variable x, (iii) $y \leftarrow x$ copies the value of variable x to y, (iv) $y \leftarrow (>_c x)$ assigns non-deterministically to y a value that exceeds the current value of x by c (so the new value of y is $> x + c$), (v) $y = x$ checks whether the value of y is equal to the value of x, (vi) $x <_c y$ checks whether the gap between the values of y and x is larger than c, (vii) $push(a)(x)$ pushes the symbol $a \in A$ to the stack and assigns to it the value of x, and (viii) $pop(a)(x)$ pops the symbol $a \in A$ (if a is the top-most symbol at the stack) and assigns its value to the variable x.

Transition System. A PDAD induces a transition system as follows. A *configuration* γ is a triple $\langle q, Val, \alpha \rangle$ where $q \in Q$ is a state, $Val : \mathcal{V} \mapsto \mathbb{N}$ is a valuation, and $\alpha \in (A \times \mathbb{N})^*$ defines the content of the stack (each element of the word is a pair $\langle a, c \rangle$ where a is the symbol and c is its value).

We define the transition relation $\longrightarrow := \cup_{\delta \in \Delta} \xrightarrow{\delta}$, where $\xrightarrow{\delta}$ describes the effect of the transition δ. For configurations $\gamma = \langle q, Val, \alpha \rangle$, $\gamma' = \langle q', Val', \alpha' \rangle$, and a transition $\delta = \langle q_1, op, q_2 \rangle \in \Delta$, we write $\gamma \xrightarrow{\delta} \gamma'$ to denote that $q = q_1$, $q' = q_2$, and one of the following conditions is satisfied:

- op is nop, $Val' = Val$, and $\alpha' = \alpha$. The values of the variables and the stack content are not changed.
- op is $x \leftarrow *$, $Val' = Val[x \leftarrow c]$ where $c \in \mathbb{N}$, and $\alpha' = \alpha$. The value of the variable x is changed non-deterministically to some natural number. The values of the other variables and the stack content are not changed.
- op is $y \leftarrow x$, $Val' = Val[y \leftarrow Val(x)]$, and $\alpha' = \alpha$. The value of the variable x is copied to the variable y. The values of the other variables and the stack content are not changed.
- op is $y \leftarrow (>_c x)$, $Val' = Val[y \leftarrow c']$, where $c' > Val(x) + c$, and $\alpha' = \alpha$. The variable y is assigned non-deterministically a value that exceeds the value of x by c. The values of the other variables and the stack content are not changed.
- op is $y = x$, $Val(y) = Val(x)$, $Val' = Val$, and $\alpha' = \alpha$. The transition is only enabled if the value of y is equal to the value of x. The values of the variables and the stack content are not changed.
- op is $x <_c y$, $Val(y) > Val(x) + c$, $Val' = Val$, and $\alpha' = \alpha$. The transition is only enabled if the value of y is larger than the value of x by more than c. The values of the variables and the stack content are not changed.
- op is $push(a)(x)$, $Val' = Val$, and $\alpha' = \langle a, Val(x) \rangle \cdot \alpha$. The symbol a is pushed onto the stack with a value equal to that of x.
- op is $pop(x)(a)$, $\alpha = \langle a, c \rangle \cdot \alpha'$ for some $c \in \mathbb{N}$, and $Val' = Val[x \leftarrow c]$. The symbol a is popped from the stack (if it is the top-most symbol), and its value is copied to the variable x.

We define the *initial configuration* $\gamma_{init} := \langle q_{init}, Val_{init}, \varepsilon \rangle$, where $Val_{init}(x) = 0$ for all $x \in \mathcal{V}$. In other words, we start from a configuration where the automaton is in its initial state, the values of all variables are equal to 0, and the stack is empty (the fact that we choose to initialize the variables to 0 is not crucial for solving the problem).

For a configuration and a state $q \in Q$, we write $\gamma \xrightarrow{*} q$ to denote that $\gamma \xrightarrow{*} \gamma' = \langle q, Val, \alpha \rangle$ for some $Val : \mathcal{V} \mapsto \mathbb{N}$ and $\alpha \in (A \times \mathbb{N})^*$.

In other words, from γ we can reach a configuration whose state is q.

Reachability Problem. In the reachability problem PDAD-REACH, given a PDAD $\mathcal{A} = \langle Q, q_{init}, A, \Delta \rangle$ and a state $q_{target} \in Q$, we ask whether $\gamma_{init} \xrightarrow{*} q_{target}$.

4 Context-Free Grammars with Data

In this section, we introduce *Context-Free Grammars with Data* (CFGD) that are extensions of the classical model of Context-Free Grammars (CFG) in which (terminal

and non terminal) symbols are defined by terms with free variables and productions have conditions defined by gap order constraints. We define the model, the operational semantics, and the reachability problem.

Model. A *Context-Free Grammars with Data* (CFGD) is a tuple $\mathcal{G} = \langle \mathcal{S}, X_{init}, P \rangle$, where $\mathcal{S}$ is a finite set of *symbols*. $X_{init} \in \mathcal{S}$ is the *start (or initial) symbol*, and P is the set of *productions*. Each symbol X has an *arity* $\rho(X) \in \mathbb{N}$ that is a natural number. Without loss of generality, we assume that $\rho(X_{init}) = 1$. A *term* has the form $X(x_1, \ldots, x_n)$ where $X \in \mathcal{S}$, $\rho(X) = n$ and $x_1, \ldots x_n \in \mathcal{V}$ are variables. A *ground term* has the form $X(c_1, \ldots, c_n)$ where $X \in \mathcal{S}$, $\rho(X) = n$ and $c_1, \ldots c_n \in \mathbb{N}$ are natural numbers. For a term σ of the form $X(x_1, \ldots, x_n)$ we define $Sym(\sigma) = X$ and $Var(\sigma) = \{x_1, \ldots, x_n\}$. We define $Sym(\sigma)$ for a ground term σ similarly. A *(ground) sentence* α is a finite set $\{\sigma_1, \sigma_2, \cdots, \sigma_n\}$, where each σ_i is a (ground) term. We define $Sym(\alpha) := \{Sym(\sigma_1), \ldots, Sym(\sigma_n)\}$, i.e., it is the set of symbols that occur in α. For a term $\sigma = X(x_1, \ldots, x_n)$ and a valuation *Val*, we define $Val(\sigma) := X(Val(x_1), \ldots, Val(x_n))$ to be the ground term we get by substituting each variable x_i in σ by $Val(x_i)$. For a sentence α, we define $Val(\alpha)$ similarly.

A *condition* θ is a finite conjunction of formulas of the forms: $x <_c y$ or $x = y$, where $x, y \in \mathcal{V}$ and $c \in \mathbb{N}$. Here $x <_c y$ stands for $x + c < y$. Sometimes, we treat a condition θ as set, and write e.g. $(x <_c y) \in \theta$ to indicate that $x <_c y$ is one of the conjuncts in θ. For a valuation *Val*, we use $Val(\theta)$ to denote the result of substituting each variable x in θ by $Val(x)$. We use $Val \models \theta$ to denote that $Val(\theta)$ evaluates to *true*. We use $Var(\theta)$ to denote the set of variables that occur in θ.

A *production* p is of the form $\sigma \rightsquigarrow \alpha : \theta$, where σ is a term, α is a non-empty sentence, and θ is a condition. We often use the notation $\sigma \rightsquigarrow \sigma_1 \cdots \sigma_n : \theta$ to denote the production $\sigma \rightsquigarrow \{\sigma_1, \ldots, \sigma_n\} : \theta$ (i.e. a sequence in the right-hand side denotes a set of terms). We use $\mathcal{N}$ to denote the set of non-terminals consisting of symbols that occur in the left-hand side of a production (we say that they are defined by a production). We use $\mathcal{T}$ to denote the set of terminals consisting of symbols that do not occur in the left-hand side of a production. Furthermore, we use $\mathcal{A}_T$ to denote the set of ground terms with symbols in $\mathcal{T}$.

Transition System. A *configuration* γ is a ground sentence. We define a transition relation $\longrightarrow_{\mathcal{G}}$ on the set of configurations by $\longrightarrow_{\mathcal{G}} := \cup_{p \in P} \xrightarrow{p}$ where $\xrightarrow{p}$ represents the effect of applying the production p. More precisely, for a production $p \in P$ of the form $\sigma \rightsquigarrow \alpha : \theta$, we have $\gamma_1 \xrightarrow{p} \gamma_2$ if there is a valuation $Val \models \theta$ such that $\gamma_1 = \alpha' \cup \{Val(\sigma)\}$ and $\gamma_2 = \alpha' \cup \{Val(\alpha)\}$.

For a set S of ground terms, we define $Pre(S)$ to be the set of ground terms σ which can, through the single application of a production, generate a configuration $\gamma \subseteq S$ (i.e., $\sigma \longrightarrow_{\mathcal{G}} \gamma$). Let $Pre^*(\cdot)$ denote the transitive closure of $Pre(\cdot)$.

We will use the following lemmata later in the paper.

Lemma 1. *Let α be a ground sentence of $\mathcal{G}$. Then, if for every ground term $\sigma \in \alpha$, we have $\sigma \xrightarrow{*}_{\mathcal{G}} \alpha''$ for some ground sentence α'' such that $Sym(\alpha'') \subseteq \mathcal{T}$, then $\alpha \xrightarrow{*}_{\mathcal{G}} \alpha'$ for α' such that $Sym(\alpha') \subseteq \mathcal{T}$.*

Lemma 2. *Let S be a set of ground terms and σ be a ground term such that $\sigma \in Pre^*(S)$. If $\sigma \notin S$ then there is a ground term $\sigma' \in (Pre(S) \setminus S)$.*

Reachability Problem. In the *reachability* problem CFGD-REACH, we are given a CFGD $\mathcal{G} = \langle \mathcal{S}, X_{init}, P \rangle$ and we are asked the question whether $X_{init}(0) \xrightarrow{*}_{\mathcal{G}} \alpha$ for some ground sentence α such that $Sym(\alpha) \subseteq \mathcal{T}$. In other words, we start from a configuration consisting of the start symbol with its parameter is equal to zero, and ask whether the system can reach a configuration where all its ground terms have symbols in $\mathcal{T}$.

CFGD *vs* CFG A Context-Free Grammars (CFG) is defined by production of the form $S \rightarrow w$ where w is a word defined over terminal and non terminal symbols. We can encode a CFG as a CFGD by associating to each terminal/non terminal symbol X (except the initial) a term $X(a,b)$ in which (a,b) are used to maintain an order in the right-hand side of a rule. For instance, the production $S \rightarrow SaS$ is encoded via the CFGD production $S(x,y) \rightarrow \{S(x,z), a(z,t), S(t,y)\} : x < z, z < t, t < y$.

CFGD *vs* CMRS CFGD also differ from the CMRS model [7]. CMRS is obtained by combining multiset rewriting and Gap Order constraints and it is aimed at modeling concurrent processes. CMRS rules have multiple heads and work over multisets of monadic terms (i.e. with a single argument, no nested terms). Differently from CMRS, CFGD productions have a single term in the left-hand side and a set of terms in the right-hand side. This implies that multiple occurrences (with the same variables) of a term like $p(x,y)$ are counted only once. Furthermore, non-terminal symbols have arbitrary finite arity.

5 Symbolic Encoding

In this section, we define the symbolic representation used in the definition of the reachability algorithm (Section 6). The algorithm operates on *constraints*, where each constraint ϕ characterizes a (potentially) infinite set $[\![\phi]\!]$ of ground terms. A *constraint* ϕ is of the form $\sigma : \theta$ where σ is a term and θ is a condition. We define $Sym(\phi) = Sym(\sigma)$ and $Var(\phi) = Var(\sigma) \cup Var(\theta)$.

Definition 3. *The constraint ϕ characterizes a set of ground terms defined by $[\![\phi]\!] = \{\sigma' \mid \exists Val.\ (Val \models \theta) \wedge (\sigma' = Val(\sigma)\}$. For a finite set of constraints Φ, $[\![\Phi]\!] = \bigcup_{\phi \in \Phi} [\![\phi]\!]$.*

Without loss of generality, we can assume that $Var(\theta) = Var(\sigma)$, and that θ is consistent (constraints with inconsistent conditions characterize empty sets of configurations, and can therefore be safely discarded from the reachability analysis). A term $X(x_1, \ldots, x_n)$ is said to be *pure* if $x_i \neq x_j$ whenever $i \neq j$. A constraint $\sigma : \theta$ is said *pure* if σ is pure. We can assume without loss of generality that all constraints are pure. The reason is that if a variable x occurs (say) twice then the two occurrences of x can be replaced by two different variables y_1 and y_2 provided that we add a new conjunct $y_1 = y_2$ to the condition θ. For constraints ϕ_1, ϕ_2, we use $\phi_1 \sqsubseteq \phi_2$ to denote that ϕ_1 *subsumes* ϕ_2, i.e., $[\![\phi_1]\!] \supseteq [\![\phi_2]\!]$. Then, it is easy to see that checking whether $\phi_1 \sqsubseteq \phi_2$ can be reduced to the satisfiability problem for an existential Presburger formula (which is known to be NP-COMPLETE [34]).

Lemma 4. *For constraints ϕ_1, ϕ_2, the problem of checking whether $\phi_1 \sqsubseteq \phi_2$ is decidable.*

The following lemma states that we can transform any constraint ϕ of the form $\sigma : \theta$ to an equivalent constraint $clean(\phi)$ of the form $\sigma : \theta'$ such that $Var(\theta') = Var(\sigma)$ (i.e., we remove the extra-variables $(Var(\theta) \setminus Var(\sigma))$ from θ in order to satisfy the assumption that $Var(\theta) = Var(\sigma)$).

Lemma 5. *[31] Given a constraint ϕ of the form $\sigma : \theta$, we can construct a constraint $clean(\phi)$ of the form $\sigma : \theta'$ such that $Var(\theta') = Var(\sigma)$ and $[\![clean(\phi)]\!] = [\![\phi]\!]$.*

Given two terms σ_1 and σ_2, we say that σ_1 matches σ_2 iff $Sym(\sigma_1) = Sym(\sigma_2)$. For matching terms $\sigma_1 = X(x_1,\ldots,x_n)$ and $\sigma_2 = X(y_1,\ldots,y_n)$, where σ_2 is pure, we define $Ren_{\sigma_1}^{\sigma_2}$ to be a renaming such that $Ren_{\sigma_1}^{\sigma_2}(y_i) = x_i$ for all $i : 1 \leqslant i \leqslant n$. Consider a production $p = \sigma \rightsquigarrow \sigma_1 \cdots \sigma_n \ : \ \theta$ and constraints $\phi_1 = \sigma_1' : \theta_1, \ldots, \phi_n = \sigma_n' : \theta_n$ such that σ_i and σ_i' are matching, and such that σ_i' is pure for all $i : 1 \leqslant i \leqslant n$. We define $p \otimes \phi_1 \otimes \cdots \otimes \phi_n$ to be the constraint $\sigma : \theta \wedge Ren_{\sigma_1}^{\sigma_1'}(\theta_1) \wedge \cdots \wedge Ren_{\sigma_n}^{\sigma_n'}(\theta_n)$. For a set Φ of constraints, and production $p \in P$, we define $Pre_p(\Phi) := \{clean(\phi') | \exists \phi_1, \ldots, \phi_n \in \Phi. \phi' = p \otimes \phi_1 \cdots \otimes \phi_n\}$. We define $Pre(\Phi) := \cup_{p \in P} Pre_p(\Phi)$. Intuitively, $Pre(\Phi)$ defines a finite set of constraints that characterize the terms which can, through the single application of a production, generate a set of terms each of which belongs to Φ.

Lemma 6. $\bigcup_{\phi' \in Pre(\Phi)} [\![\phi']\!] = Pre([\![\Phi]\!])$.

For the set $\mathcal{T}$ of terminals, we define

$$\Phi_{\mathcal{T}} := \{a(x_1,\ldots,x_n) : \mathsf{true} | \, a \in \mathcal{T}, \ \rho(a) = n\}$$

Notice that $\Phi_{\mathcal{T}}$ denotes the set of configurations whose symbols are in $\mathcal{T}$.

6 Reachability Analysis

In this section, we present an algorithm for solving the reachability analysis problem for CFGDs, and prove its partial correctness. The algorithm (Algorithm 1) inputs a CFGD $\mathcal{G} = \langle \mathcal{S}, X_{init}, P \rangle$ and answers the question whether we can reach a sentence where all the occurring terms are in $\mathcal{A}_T$ (i.e. terms with symbols in $\mathcal{T}$). The algorithm maintains two sets of constraints: a set `ToExplore`, initialized to $\Phi_{\mathcal{T}}$, of constraints that have not yet been analyzed; and a set `Explored`, initialized to the empty set, of constraints that contain constraints that have already been analyzed.

The algorithm preserves the following four invariants:

1. For each $\sigma \in [\![\texttt{ToExplore} \cup \texttt{Explored}]\!]$, $\sigma \xrightarrow{*} \alpha$ for some α s.t. $Sym(\alpha) \subseteq \mathcal{T}$.
2. If $X_{init}(0) \xrightarrow{*} \alpha$ for some α s.t. $Sym(\alpha) \subseteq \mathcal{T}$, then there is a ground term $\sigma \in [\![\texttt{ToExplore}]\!]$ such that $\sigma \notin [\![\texttt{Explored}]\!]$.
3. $X_{init}(0) \notin [\![\texttt{Explored}]\!]$.
4. $[\![\Phi_{\mathcal{T}}]\!] \subseteq [\![\texttt{ToExplore} \cup \texttt{Explored}]\!]$.

It is easy to see that the third and fourth invariants will be preserved. More precisely, for the third invariant, `Explored` is initially empty, and the condition at line 5 prevents

Algorithm 1: Reachability analysis for a CFGD.

Input: A CFGD $\mathcal{G} = \langle \mathcal{S}, X_{init}, P \rangle$
Output: Is there a subset of terminal symbols $T \subseteq \mathcal{T}$ reachable in $\mathcal{G}$?

```
1  ToExplore ← Φ_T ;
2  Explored ← ∅
3  while ToExplore ≠ ∅ do
4      remove some φ from ToExplore;
5      if X_init(0) ∈ [[φ]] then return true;
6      else if ∃φ' ∈ Explored. φ' ⊑ φ then discard φ;
7      else
8          ToExplore ← ToExplore ∪ Pre(Explored ∪ {φ});
9          Explored ← {φ} ∪ {φ' | φ' ∈ Explored ∧ (φ ⋢ φ')};
10 return false
```

adding any constraint whose symbol is X_{init} and parameter equals to 0 to Explored. The fourth invariant holds initially since $\texttt{ToExplore} \cup \texttt{Explored} = \Phi_{\mathcal{T}} \cup \emptyset = \Phi_{\mathcal{T}}$. This invariant is preserved since each time we remove a constraint from ToExplore (line 4), it is either eventually moved to Explored (line 9), or (in case it is discarded at line 6) there is already a constraint $\phi' \in \texttt{Explored}$ with $[\![\phi']\!] \supseteq [\![\phi]\!]$. Also, each time we remove a constraint ϕ' from Explored (line 9), we add the constraint ϕ to Explored where $[\![\phi]\!] \supseteq [\![\phi']\!]$.

Below, we show that the first two invariants are also preserved. Initially, the first invariant holds since $(\texttt{ToExplore} \cup \texttt{Explored}) = \Phi_{\mathcal{T}}$. The second invariant also holds initially since $\texttt{Explored} = \emptyset$ and $[\![\texttt{ToExplore}]\!] = [\![\Phi_{\mathcal{T}}]\!] \neq \emptyset$. Due to the first two invariants, the following two conditions can be checked during each step of the algorithm:

- From the second invariant, if ToExplore becomes empty then the algorithm terminates with a negative answer.
- From the first invariant, if a constraint ϕ is detected such that $X_{init}(0) \in [\![\phi]\!]$, then the algorithm terminates with a positive answer.

If neither of the two conditions is satisfied, the algorithm proceeds by picking and removing a constraint ϕ from ToExplore. Two possibilities arise depending on the value of σ:

- If there exists a constraint $\phi' \in \texttt{Explored}$ with $\phi' \sqsubseteq \phi$, then we discard ϕ. The first invariant is preserved since this operation will not add any new elements to $[\![\texttt{ToExplore} \cup \texttt{Explored}]\!]$. If $X_{init}(0) \xrightarrow{*} \alpha$ for some α s.t. $Sym(\alpha) \subseteq \mathcal{T}$, then the second invariant and the fact that $[\![\phi]\!] \subseteq [\![\texttt{Explored}]\!]$ imply that there is still some $\sigma \in \texttt{ToExplore}$ such that $\sigma \notin [\![\texttt{Explored}]\!]$. This means that the second invariant will also be preserved by this step.
- Otherwise, we compute the elements of $Pre(\texttt{Explored} \cup \phi)$, add them in ToExplore, move ϕ to Explored, and remove all constraints in Explored that are subsumed by ϕ. Let $\texttt{Explored}^{\texttt{old}}$ and $\texttt{Explored}^{\texttt{new}}$ be the contents of the set Explored before resp. after performing the operation. Define $\texttt{ToExplore}^{\texttt{old}}$ and $\texttt{ToExplore}^{\texttt{new}}$ analogously. The operation preserves the

first invariant as follows. Pick any $\sigma \in [\![\texttt{ToExplore}^{\texttt{new}} \cup \texttt{Explored}^{\texttt{new}}]\!]$. If $\sigma \in [\![\texttt{ToExplore}^{\texttt{old}} \cup \texttt{Explored}^{\texttt{old}}]\!]$ then the result follows by the first invariant. Otherwise we know that $\sigma \in [\![Pre(\texttt{Explored}^{\texttt{old}} \cup \{\phi\})]\!]$, i.e., $\sigma \longrightarrow_{\mathcal{G}} \alpha$ where $\alpha \subseteq [\![\texttt{Explored}^{\texttt{old}} \cup \{\phi\}]\!]$ (see Lemma 6). By the induction hypothesis and the first invariant, we know that every ground term $\sigma' \in \alpha$, $\sigma' \xrightarrow{*}_{\mathcal{G}} \alpha'$ for some α' s.t. $Sym(\alpha') \subseteq \mathcal{T}$. Hence $\alpha \xrightarrow{*}_{\mathcal{G}} \alpha''$ for some α'' s.t. $Sym(\alpha'') \subseteq \mathcal{T}$ (see Lemma 1). In other words, $\sigma \longrightarrow_{\mathcal{G}} \alpha \xrightarrow{*}_{\mathcal{G}} \alpha''$ s.t. $Sym(\alpha'') \subseteq \mathcal{T}$. The operation also preserves the second invariant as follows. Assume that $X_{init}(0) \xrightarrow{*}_{\mathcal{G}} \alpha$ for some α s.t. $Sym(\alpha) \subseteq \mathcal{T}$. There are two cases. If there is a $\sigma \in [\![\Phi_{\mathcal{T}}]\!]$ such that $\sigma \notin [\![\texttt{Explored}^{\texttt{new}}]\!]$, then by the fourth invariant $\sigma \in [\![\texttt{ToExplore}^{\texttt{new}}]\!]$ and the invariant holds immediately. Otherwise, $[\![\Phi_{\mathcal{T}}]\!] \subseteq [\![\texttt{Explored}^{\texttt{new}}]\!]$. Since $X_{init}(0) \xrightarrow{*}_{\mathcal{G}} \alpha$ we have also that $X_{init}(0) \in Pre^*([\![\texttt{Explored}^{\texttt{new}}]\!])$. By the third invariant, we know that $X_{init}(0) \notin [\![\texttt{Explored}^{\texttt{new}}]\!]$. By Lemma 2 that there is a ground term $\sigma \in (Pre([\![\texttt{Explored}^{\texttt{new}}]\!]) \setminus [\![\texttt{Explored}^{\texttt{new}}]\!])$. Since $[\![\texttt{Explored}^{\texttt{new}}]\!] = [\![\texttt{Explored}^{\texttt{old}} \cup \{\phi\}]\!]$ it follows that $\sigma \in [\![Pre(\texttt{Explored}^{\texttt{old}} \cup \{\phi\})]\!]$ and hence $\sigma \in [\![\texttt{ToExplore}^{\texttt{new}}]\!]$.

This give us the following theorem.

Theorem 7. *Algorithm 1, under termination assumption, always return the correct answer.*

7 Termination

In this section, we show that Algorithm 1 is guaranteed to terminate. To do that, we first recall some basics of the theory of well and better quasi-orderings. Then, we introduce a new class of constraints that we call *flat constraints* and show that they are better quasi-ordered. We show that each condition can be translated into a number of flat constraints. We use this to show that the set of conditions is well quasi-ordered under set inclusion. This leads to the well quasi-ordering of the set of constraints (of Section 5). Finally, we show the termination of the algorithm.

WQOs *and* BQOs. A *Quasi-Ordering* (or a QO for short), is a pair $\langle A, \preceq \rangle$ where $\preceq$ is a reflexive and transitive binary relation on the set A. A QO $\langle A, \preceq \rangle$ is a *Well Quasi-Ordering* (WQO), if for each infinite sequence $a_1, a_2, a_3, \ldots$ of elements of A , there are $i < j$ such that $a_i \preceq a_j$. The following lemma follows from the definition of a WQO.

Lemma 8. *For* QOs $\preceq$ *and* $\preceq'$ *on some set A, if* $\preceq \subseteq \preceq'$ *and* $\preceq$ *is a* WQO *then* $\preceq'$ *is a* WQO.

Given a QO $\langle A, \preceq \rangle$, we define a QO $\langle A^*, \preceq^* \rangle$ on the set of words A^* such that $a_1 a_2 \cdots a_m \preceq^* a'_1 a'_2 \cdots a'_n$ if there is an injection $h : \{1, \ldots, m\} \mapsto \{1, \ldots, n\}$ such that $i < j$ implies $h(i) < h(j)$ for all $i, j : 1 \leqslant i, j \leqslant m$, and $a_i \preceq a'_{h(i)}$ for each $i : 1 \leqslant i \leqslant m$. We define the relation $\preceq^{\mathcal{P}}$ on the powerset $\mathcal{P}(A)$ (finite set of elements in A) of A, so that $A_1 \preceq^{\mathcal{P}} A_2$ if $\forall a_2 \in A_2 . \exists a_1 \in A_1 . a_1 \preceq a_2$.

We define the relation $\preceq^p$ on the Cartesian product $A_1 \times \ldots \times A_n$ of orders $\langle A_i, \leqslant_i \rangle$ for $i : 1, \ldots, n$, so that $\langle a_1, \ldots, a_n \rangle \preceq^p \langle a'_1, \ldots, a'_n \rangle$ if $a_i \preceq_i a'_i$ for $i : 1, \ldots, n$.

In the following lemma we state some properties of BQOs[4] [10, 28].

Lemma 9. – *Each* BQO *is* WQO.
- *If A is finite, then $\langle A, = \rangle$ is a* BQO*, and $\langle \wp(A), \subseteq \rangle$ is a* BQO.
- $\langle \mathbb{N}, \leqslant \rangle$ *is a* BQO.
- *If $\langle A_i, \leqslant_i \rangle$ is a* BQO *for $i : 1, \ldots, n$ then $\langle A_1 \times \ldots \times A_n, \preceq^p \rangle$ is a* BQO.
- *If $\langle A, \preceq \rangle$ is a* BQO*, then $\langle \wp(A), \preceq^{\wp} \rangle$ is a* BQO.

Flat Constraints. Fix a set $\mathcal{V} = \{x_1, \ldots, x_n\}$ of variables. A flat constraint ψ over $\mathcal{V}$ if of the form $A_0 c_1 A_1 \cdots c_m A_m$, where $c_1, \ldots, c_m \in \mathbb{N}$, and $A_0, A_2, \ldots, A_m$ is a partitioning of $\mathcal{V}$, i.e., $\mathcal{V} = A_0 \cup A_1 \cup \cdots \cup A_m$, $A_i \neq \emptyset$, and $A_i \cap A_j = \emptyset$ if $i \neq j$. In other words, a flat constraint is a word which alternatively contains sets of variables and natural numbers, starting and ending with a set of variables. The flat constraint ψ characterizes an infinite set $[\![\psi]\!]$ of vectors over $\mathbb{N}$ of length n, i.e., $[\![\psi]\!] \subseteq \mathbb{N}^n$. More precisely, define $h_\psi : \{1, \ldots, n\} \mapsto \{0, \ldots, m\}$ such that $h_\psi(i) = k$ if $x_i \in A_k$. $v = \langle d_1, \ldots, d_n \rangle \in [\![\psi]\!]$ iff the following conditions are satisfied for all $i, j : 1 \leqslant i, j \leqslant n$:

- $d_i = d_j$ if $h_\psi(i) = h_\psi(j)$.
- If $h_\psi(i) = k$. and $h_\psi(j) = k+1$ then $c_{k+1} < d_j - d_i$.

In other words, the variable x_i represents d_i in ψ. If two variables are mapped to the same set then their values should be identical. Furthermore, the natural numbers c_i define the gaps between values of variables belonging to the different sets. For flat constraints $\psi = A_0 c_1 A_1 \cdots c_m A_m$ and $\psi' = A'_0 c'_1 A'_1 \cdots c'_m A'_m$ over $\mathcal{V}$, we write $\psi \preceq \psi'$ to denote that (i) $A'_i = A_i$ for all $i : 0 \leqslant i \leqslant m$, and (ii) $c_i \leqslant c'_i$ for all $i : 1 \leqslant i \leqslant m$. The following lemma follows from the definitions.

Lemma 10. $\psi \preceq \psi'$ *implies that* $[\![\psi]\!] \supseteq [\![\psi']\!]$.

By Lemma 9 it follows that

Lemma 11. $\preceq$ *is a* BQO *on the set of flat constraints.*

Proof. We first observe that flat contraints can be viewed as tuples with at most $K = |\mathcal{V}|$ partitions and $|\mathcal{V}| - 1$ constants and we can always add finite sequences such as $0\emptyset 0 \ldots 0\emptyset$ to consider K-tuples only. From Lemma 9, we know that $\langle \mathbb{N}, \leqslant \rangle$ and $\langle \wp(\mathcal{V}), = \rangle$ are BQOs. Thus, the Cartesian product $(\wp(\mathcal{V}) \times \mathbb{N})^{K-1} \times \wp(\mathcal{V})$ with $\preceq$ is still a BQO.

Flattening. Consider a condition θ with $Var(\theta) = \{x_1, \ldots, x_n\}$ (recall the definitions of conditions and constraints from Section 5). We define $[\![\theta]\!]$ to be the set of vectors $v = \langle d_1, \ldots, d_n \rangle \in \mathbb{N}^n$, such that there is a valuation Val with $Val \models \theta$ and $Val(x_i) = d_i$ for all $i : 1 \leqslant i \leqslant n$. Furthermore, for two conditions on the same set of variables we define $\theta \sqsubseteq \theta'$ iff $[\![\theta]\!] \supseteq [\![\theta']\!]$. A *flattening* of θ is a flat constraint ψ over $Var(\theta)$, of the form $A_0 c_1 A_1 \cdots c_m A_m$ where $c_1, \ldots, c_m \geqslant 0$ are minimal natural numbers such that the following conditions are satisfied:

[4] The technical definition of BQOs is quite complicated and can be found in e.g. [10]. The actual definition is not needed for understanding the rest of the paper, and is therefore omitted here.

- If $(x = y) \in \theta$ then $x, y \in A_i$ for some $i : 1 \leqslant i \leqslant m$.
- If $(x <_c y) \in \theta$, $x \in A_i$, and $y \in A_j$ then $c \leqslant \left(\sum_{k=i+1}^{j}(c_k + 1) - 1\right)$.

Intuitively, variables which are required to be equal by θ, are put in the same X_i. Also, variables which are ordered according to θ, are placed sufficiently far apart to cover the corresponding gap. We define $\mathcal{F}(\theta)$ to be the set of flattening of θ. In general conditions induce a partial order between variables. The flattening contains all linearizations with minimal gaps (constants) between variables. Notice that this set is finite. As an example, consider the condition $x <_2 y, x <_1 z$. Since there are no constraints on y and z, we have three different flattening where $y < z$ or $y = z$ or $y > z$, namely $\{x\}2\{y\}0\{z\}$, $\{x\}2\{y,z\}$, and $\{x\}1\{z\}0\{y\}$.

We define an ordering $\preceq$ on conditions such that $\theta \preceq \theta'$ if for each $\psi' \in \mathcal{F}(\theta')$ there is a $\psi \in \mathcal{F}(\theta)$ with $\psi \preceq \psi'$. From Lemma 10 we get the following.

Lemma 12. *$\theta \preceq \theta'$ implies that $[\![\theta]\!] \supseteq [\![\theta']\!]$.*

The following lemma follows from Lemma 9 and Lemma 11.

Lemma 13. *$\preceq$ is a* BQO *(and hence* WQO*) on the set of conditions.*

From Lemma 13, Lemma 12, and Lemma 8 we get the following lemma.

Lemma 14. *The set of conditions is* WQO *under $\sqsubseteq$.*

The following lemma then holds.

Lemma 15. *The set of constraints is* WQO *under $\sqsubseteq$.*

Proof. Consider an infinite sequence of constraints: $\phi_1, \phi_2, \phi_3, \ldots$. Since the set $\mathcal{N} \cup \mathcal{T}$ is finite, there is an infinite sequence $i_1 < i_2 < i_3 < \cdots$ such that $Sym(\phi_{i_1}) = Sym(\phi_{i_2}) = Sym(\phi_{i_3}) = \cdots$. If $Sym(\phi_{i_j}) \in \mathcal{T}$ then the result follows immediately (since $[\![\phi_{i_j}]\!] = \{Sym(\phi_{i_j})\}$ for all $j \geqslant 1$). Otherwise, we can assume, without loss of generality, that ϕ_{i_j} is of the form $X(x_1, \ldots, x_n) : \theta_{i_j}$. Notice that each $Var(\theta_{i_j}) = \{x_1, \ldots, x_n\}$ is a condition over $\{x_1, \ldots, x_n\}$. By Lemma 14, there are $j < k$ such that $\theta_{i_j} \sqsubseteq \theta_{i_k}$, and hence $\phi_{i_j} \sqsubseteq \phi_{i_k}$.

Termination. The reason why the algorithm always terminates is that only a finite set of constraints can be added to `Explored`. This can be explained as follows. By definition, a new element ϕ is added to `Explored` only if $\phi' \not\sqsubseteq \phi$, for each ϕ' already added to `Explored`. This means that the constraints added to `Explored` form a sequence $\phi_1, \phi_2, \phi_3, \ldots$, such that $\phi_i \not\sqsubseteq \phi_j$ for all $i < j$. By WQO of $\sqsubseteq$ (Lemma 15) it follows that this sequence is finite. This gives the following theorem.

Theorem 16. *Algorithm 1 is guaranteed to terminate.*

8 Translation

Reachability with Empty Stacks. We consider a different variant of PDAD-REACH which we call PDAD-REACH-EMPTY. An instance of PDAD-REACH-EMPTY is defined by a PDAD $\mathcal{A} = \langle Q, q_{init}, A, \Delta \rangle$ and a state $q_{target} \in Q$, and we are asked whether

$\gamma_{init} \xrightarrow{*} \gamma$ for some γ of the form $\langle q_{target}, Val, \varepsilon \rangle$, i.e., we ask whether we reach q_{target} at a configuration where the stack is *empty*. Given an instance of PDAD-REACH, defined by a PDAD $\mathcal{A} = \langle Q, q_{init}, A, \Delta \rangle$ and a state $q_{target} \in Q$, we derive an equivalent instance of PDAD-REACH-EMPTY as follows. We construct a new PDAD $\mathcal{A}'$ from $\mathcal{A}$ by adding a new state q_{new} to Q, and adding a transition labeled with *nop* from q_{target} to q_{new}. For each member $a \in A$ of the stack alphabet, we add a self-loop on q_{new} that pops a (with any value). The two problem instances are equivalent as follows. Suppose that q_{new} is reachable with an empty stack in $\mathcal{A}'$. Then, the run of $\mathcal{A}'$ reaching q_{new} must have passed through q_{target} (since q_{new} can only be reached from q_{target}). This means that q_{target} is reachable in $\mathcal{A}$. On the other hand, suppose that q_{target} is reachable in $\mathcal{A}$. Then, $\mathcal{A}'$ can simulate the run of $\mathcal{A}$ until it reaches q_{target}. From there, it takes the transition to q_{new}, and starts executing the self-loops, popping all the symbols in the stack until the stack becomes empty.

From PDAD *to* CFGD. Suppose that we are given an instance of PDAP-REACH-EMPTY defined by a PDAD $\mathcal{A} = \langle Q, q_{init}, A, \Delta \rangle$ and a state $q_{target} \in Q$. Let $\{x_1, \ldots, x_n\}$ be the set of variables that occur in $\mathcal{A}$. We derive an equivalent instance of CFGD-REACH defined by a CFGD $\mathcal{G} = \langle \mathcal{S}, X_{init}, P \rangle$. The set $\mathcal{T}$ of $\mathcal{G}$ is defined by the singleton set $\{t\}$ and we assume that the arity of t is 0 (i.e., $\rho(t) = 0$). The set of $\mathcal{N}$ of $\mathcal{G}$ is defined as follows: For each pair of states $q_1, q_2 \in Q$ and symbol $a \in A \cup \{\bot\}$, with $\bot \notin A$, we have a nonterminal $X_{(q_1,a,q_2)} \in \mathcal{N}$ with arity $2n+1$. The symbol $\bot$ is used to denote that the stack of $\mathcal{A}$ is empty. The set of non-terminal set $\mathcal{N}$ contains the initial symbol X_{init} (by definition).

In the following, let $\overline{y}$ denote a vector $\langle y_1, \ldots, y_n \rangle$ of length n, and define $\overline{y}[i] := y_i$ for $i : 1 \leqslant i \leqslant n$. For vectors $\overline{z} = \langle z_1, \ldots, z_n \rangle$ and $\overline{y} = \langle y_1, \ldots, y_n \rangle$, we use $\overline{z} = \overline{y}$ (resp. $\overline{z} \neq_j \overline{y}$ for some $j : 1 \leqslant j \leqslant n$) to denote the condition $\bigwedge_{1 \leqslant i \leqslant n} z_i = y_i$ (resp. $\bigwedge_{(1 \leqslant i \leqslant n) \wedge (i \neq j)} z_i = y_i$). Furthermore, for brevity, we sometimes shorten a conjunction of conditions $\theta_1 \wedge \ldots \wedge \theta_n$ into a list $\theta_1, \ldots, \theta_n$.

Intuitively, a non-terminal of the form $X_{(q_1,a,q_3)}(\overline{y}, \overline{z}, \ell)$ represents a run of $\mathcal{A}$ from a configuration where the state is q_1, the topmost stack symbol is a and its corresponding value is given by the value ℓ (if $a = \bot$ then the stack is empty), and the valuation of the shared variables of $\mathcal{A}$ is given by the valuation of $\overline{y}$, to a configuration with a stack content where a has been popped and where the state is q_3 and the valuation of the shared variables of $\mathcal{A}$ is given by the valuation of $\overline{z}$.

The set P is derived from Δ, and it contains the productions of Fig. 1. Then the following property holds.

Proposition 17. *$\gamma_{init} \xrightarrow{*} \gamma$ for some $\gamma = \langle q_{target}, Val, \varepsilon \rangle$ iff $X_{init} \xrightarrow{*}_{\mathcal{G}} \alpha$ for some sentence α such that $Sym(\alpha) \subseteq \mathcal{T}$.*

As an immediate consequence of the above Proposition, Theorem 7, and Theorem 16, we get:

Theorem 18. *The* PDAD-REACH *and* PDAD-REACH-EMPTY *problems are decidable for* PDADs.

$$\frac{\langle q_1, nop, q_2\rangle \in \Delta \quad q_3 \in Q}{(X_{(q_1,a,q_3)}(\overline{y},\overline{z},\ell) \rightsquigarrow X_{(q_2,a,q_3)}(\overline{y'},\overline{z'},\ell') \ : \ \overline{y}=\overline{y'},\overline{z}=\overline{z'},\ell=\ell') \in P}$$

$$\frac{\langle q_1, x_i \leftarrow *, q_2\rangle \in \Delta \quad q_3 \in Q}{(X_{(q_1,a,q_3)}(\overline{y},\overline{z},\ell) \rightsquigarrow X_{(q_2,a,q_3)}(\overline{y'},\overline{z'},\ell') \ : \ \overline{y}\neq_i \overline{y'},\overline{z}=\overline{z'},\ell=\ell') \in P}$$

$$\frac{\langle q_1, x_i \leftarrow x_j, q_2\rangle \in \Delta \quad q_3 \in Q}{(X_{(q_1,a,q_3)}(\overline{y},\overline{z},\ell) \rightsquigarrow X_{(q_2,a,q_3)}(\overline{y'},\overline{z'},\ell') \ : \ \overline{y}\neq_i \overline{y'},\overline{z}=\overline{z'},\ell=\ell',\overline{y'}[i]=\overline{y}[j]) \in P}$$

$$\frac{\langle q_1, x_i \leftarrow (>_c x_j), q_2\rangle \in \Delta \quad q_3 \in Q}{(X_{(q_1,a,q_3)}(\overline{y},\overline{z},\ell) \rightsquigarrow X_{(q_2,a,q_3)}(\overline{y'},\overline{z'},\ell') \ : \ \overline{y}\neq_i \overline{y'},\overline{z}=\overline{z'},\ell=\ell',\overline{y}[j]<_c \overline{y'}[i]) \in P}$$

$$\frac{\langle q_1, x_j = x_i, q_2\rangle \in \Delta \quad q_3 \in Q}{(X_{(q_1,a,q_3)}(\overline{y},\overline{z},\ell) \rightsquigarrow X_{(q_2,a,q_3)}(\overline{y'},\overline{z'},\ell') \ : \ \overline{y}=\overline{y'},\overline{z}=\overline{z'},\ell=\ell',\overline{y}[i]=\overline{y}[j]) \in P}$$

$$\frac{\langle q_1, x_j <_c x_i, q_2\rangle \in \Delta \quad q_3 \in Q}{(X_{(q_1,a,q_3)}(\overline{y},\overline{z},\ell) \rightsquigarrow X_{(q_2,a,q_3)}(\overline{y'},\overline{z'},\ell') \ : \ \overline{y}=\overline{y'},\overline{z}=\overline{z'},\ell=\ell',\overline{y'}[j]<_c \overline{y}[i]) \in P}$$

$$\frac{\langle q_1, push\,(b)\,(x_i), q_2\rangle \in \Delta \quad q_3, q_4 \in Q}{(X_{(q_1,a,q_3)}(\overline{y},\overline{z},\ell) \rightsquigarrow X_{(q_2,b,q_4)}(\overline{y'},\overline{u},\ell')\, X_{(q_4,a,q_3)}(\overline{u'},\overline{z'},\ell'') \ : \ \overline{y}=\overline{y'},\overline{u}=\overline{u'},\overline{z}=\overline{z'},\ell=\ell'',\ell'=\overline{y}[i]) \in P}$$

$$\frac{\langle q_1, pop\,(x_i)\,(a), q_2\rangle \in \Delta}{(X_{(q_1,a,q_2)}(\overline{y},\overline{z},\ell) \rightsquigarrow t \ : \ \overline{y}\neq_i \overline{z},\overline{z}[i]=\ell) \in P}$$

$$\frac{}{(X_{init}(x) \rightsquigarrow X_{(q_{init},\perp,q_{target})}(\overline{y},\overline{z},\ell) \ : \ \bigwedge_{1\leqslant i\leqslant n}\overline{y}[i]=x) \in P}$$

$$\frac{}{(X_{(q_{target},\perp,q_{target})}(\overline{y},\overline{z},\ell) \rightsquigarrow t \ : \ \overline{y}=\overline{z}) \in P}$$

Fig. 1. From transitions of pushdown with data to productions.

9 Extended PDADs

In this section, we present generalizations of the basic PDAD model for which the results presented in this paper still hold.

The first extension consists in adding to conditions of the form $x = c$, $x > c$, and $x < c$ for a variable x and a constant value $c \geqslant 0$. The resulting formulas corresponds to the original Gap Order Constraints considered in [31].

The second extension consists in adding multiple data fields in each element pushed to the stack. For fixed number of data fields $k \geqslant 0$, the configuration of PDAD$_k$ becomes a triple $\langle q, Val, \alpha\rangle$ where $q \in Q$ is a state, $Val : \mathcal{V} \mapsto \mathbb{N}$ is a valuation, and $\alpha \in \left(A \times \mathbb{N}^k\right)^*$ defines the content of the stack (each element of the word is a pair $\langle a, c_1, \ldots, c_k\rangle$ where a is the symbol and c_i is its value for the i-th field).

We now consider operations that manipulate the data fields. We first extend the push operation and consider $push\,(a)\,(x_1, \ldots, x_k)$ to push the symbol $a \in A$ and to assign to the i-th field the value of x_i for $i : 1, \ldots, k$. We also consider operation $pop\,(a)\,(x_1, \ldots, x_k)$ to pop the symbol $a \in A$ from the stack and to assign to x_i the value of the i-th field on

the top of the stack $i : 1, \ldots, k$. The operational semantics can be naturally extended in order to cope with tuples of values instead of single one.

Finally, we consider operations that test and modify the data fields on the stack. We can use special identifiers $topx_1, \ldots, topx_k$ to denote such data fields and use them in conditions of transitions.

To encode the resulting model into CFGD, we need to introduce non-terminals with extra arguments that represent both the current value and the (guessed) updated value of data fields. More specifically, we need non-terminals of the form $X_{(q_1,a,q_2)}(\overline{x},\overline{y},\overline{z},\overline{u})$ to represent a run of a $\mathcal{A}_k$ from a configuration where the state is q_1, the topmost stack symbol is a and its corresponding data field values are given by the vector $\overline{z}$, and the valuation of the shared variables of $\mathcal{A}$ is given by the valuation of $\overline{x}$, to a configuration with the updated data fields $\overline{u}$ and where the state is q_2 and the valuation of the shared variables is given by the valuation of $\overline{y}$.

We leave a detailed treatment of this extension for future work.

10 Related Work and Conclusion

Decidability and complexity of reachability problems for pushdown systems with or without data have been extensively studied in the literature. In [12] the authors present an algorithm to compute $Post^*$ and Pre^* for a pushdown automata and a regular set of its configurations (represented as automata). Symbolic versions of the algorithms have been studied e.g. in [29]. In [11] the authors consider approximated verification methods for subclasses of pushdown systems called finite indices in which it is possible to handle counters without zero test (i.e. transitions of a Petri net). In [2, 1] the authors present decidability results for timed extensions of pushdown systems. In [14] the authors present decidability results for pushdown systems with either a well-quasi ordered set of control locations or of data values. In our model we do not consider a well-quasi ordered data domain, but introduce a well-quasi ordered relation over values pushed to and popped from the stack in order to decide reachability. Our extensions of pushdown system with Gap Order is orthogonal to the above mentioned models. Furthermore, it subsumes the model presented in [32], where the authors consider pushdown systems in which messages carry (object) identifiers that can be compared by equality. In addition to equality tests, Gap Order can be used to order messages in the stack.

Concerning our proof techniques, the algorithm for solving the CFGD reachability problem is inspired to the seminal results on Datalog and context-free language reachability [35, 30] and to the evaluation of Datalog with Gap Order Constraints [31]. CLP programs with Gap Order constraints without conjunctions in the body have been used to model transition systems in [27]. The fixpoint semantics of CLP programs has been used to characterize model checking problems in [21] and applied to infinite-state systems in [18, 16, 17, 20]. In [15] extended automata with Gap Order conditions over variables are used as an approximated model of counter systems. The model however does not have recursion. The complexity of verification problems (expressed in temporal logic) for transitions systems with Gap Order Constraints has been studied in [13]. Allowing rules with sets of terms in the right-hand side, GFGD are more general than the model in [13]. Multiset rewriting systems with Gap Order Constraints (i.e. systems

with an arbitrary number of integral variables) have been introduced in [3] and applied to different types of systems in [8] extending the parameterized models described in [9, 22]. These systems are a subclass of multiset rewriting with (linear) constraints applied to infinite state verification, e.g., in [19].

The evaluation procedure for Datalog with Gap Order Constraints in [31] and its termination depend on specific data structures (weighted graphs kept in normal form) used to represent relations between variables that occur in Datalog clauses. In the present paper we formulate an algorithmic solution to CFGD reachability as an instance of the general framework of well-structured transition systems and apply the theory of better-quasi ordering to naturally infer its termination. This approach has the great advantage of capturing the essential ingredients needed for extending the algorithm to other classes of grammars with data. For instance, under some restrictions on the arity of terms, a slightly modified algorithm can be applied to grammars with sets of terms in the left-hand side of a production. A more formal treatment of this kind of generalization together with a deeper investigation of the complexity of the resulting algorithm is part of our future work.

References

1. Abdulla, P.A., Atig, M.F., Stenman, J.: Dense-timed pushdown automata. In: LICS. IEEE Computer Society (2012)
2. Abdulla, P.A., Atig, M.F., Stenman, J.: The minimal Cost reachability problem in priced timed pushdown systems. In: LATA. LNCS, vol. 7183 (2012)
3. Abdulla, P.A., Delzanno, G.: On the coverability problem for constrained multiset rewriting. In: Proc. AVIS'06, 5^{th} Int. Workshop on on Automated Verification of Infinite-State Systems (2006)
4. Abdulla, P.A.: Well (and Better) Quasi-Ordered Transition Systems. The Bulletin of Symbolic Logic 16(4), 457–515 (2010)
5. Abdulla, P.A., Atig, M.F., Cederberg, J.: Timed lossy channel systems. In: Proc. FSTTCS '05, 32^{nd} Conf. on Foundations of Software Technology and Theoretical Computer Science (2012)
6. Abdulla, P.A., Čerāns, K., Jonsson, B., Tsay, Y.K.: General decidabilitytheorems for infinite-state systems. In: Proc. LICS '96, 11^{th} IEEE Int. Symp. on Logic in Computer Science. pp. 313–321 (1996)
7. Abdulla, P.A., Delzanno, G., Begin, L.V.: A classification of the expressive power of well-structured transition systems. Inf. Comput. 209(3), 248–279 (2011)
8. Abdulla, P.A., Delzanno, G., Rezine, A.: Parameterized verification of infinite-state processes with global conditions. In: Proc. 19^{th} Int. Conf. on Computer Aided Verification. Lecture Notes in Computer Science, vol. 4590, pp. 145–157 (2007)
9. Abdulla, P.A., Delzanno, G., Rezine, A.: Approximated parameterized verification of infinite-state processes with global conditions. Formal Methods in System Design 34(2), 126–156 (2009)
10. Abdulla, P.A., Nylén, A.: Better is better than well: On efficient verification of infinite-state systems. In: Proc. LICS '00, 16^{th} IEEE Int. Symp. on Logic in Computer Science. pp. 132–140 (2000)
11. Atig, M.F., Ganty, P.: Approximating Petri net reachability along context-free traces. In: FSTTCS '11. LIPIcs, vol. 13, pp. 152–163. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2011)

12. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: CONCUR '97. pp. 135–150. LNCS 1243, Springer (1997)
13. Bozzelli, L., Pinchinat, S.: Verification of gap-order constraint abstractions of counter systems. In: VMCAI '12. pp. 88–103 (2012)
14. Cai, X., Ogawa, M.: Well-structured extensions of pushdown systems. In: RP '12 (2012)
15. Čerāns, K.: Deciding properties of integral relational automata. In: Abiteboul, Shamir (eds.) Proc. ICALP '94, 21^{st} International Colloquium on Automata, Languages, and Programming. Lecture Notes in Computer Science, vol. 820, pp. 35–46. Springer Verlag (1994)
16. Delzanno, G.: Automatic verification of cache coherence protocols. In: Emerson, Sistla (eds.) Proc. 12^{th} Int. Conf. on Computer Aided Verification. Lecture Notes in Computer Science, vol. 1855, pp. 53–68. Springer Verlag (2000)
17. Delzanno, G., Bultan, T.: Constraint-based verification of client-server protocols. In: Proc. CP '01. Lecture Notes in Computer Science, vol. 2239, pp. 286–301. Springer Verlag (2001)
18. Delzanno, G., Esparza, J., Podelski, A.: Constraint-based analysis of broadcast protocols. In: Proc. CSL'99 (1999)
19. Delzanno, G., Ganty, P.: Automatic verification of time sensitive cryptographic protocols. In: Proc. TACAS '04, 10^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. No. 2988 in Lecture Notes in Computer Science (2004)
20. Delzanno, G.: Constraint-based verification of parameterized cache coherence protocols. Formal Methods in System Design 23(3), 257–301 (2003)
21. Delzanno, G., Podelski, A.: Model checking in CLP. In: TACAS '99. pp. 223–239 (1999)
22. Delzanno, G., Rezine, A.: A lightweight regular model checking approach for parameterized systems. STTT 14(2), 207–222 (2012)
23. Esparza, J., Hansel, D., Rossmanith, P., Schwoon, S.: Efficient algorithms for model checking pushdown systems. In: CAV. LNCS, vol. 1855. Springer (2000)
24. Esparza, J., Kučera, A., Mayr, R.: Model checking probabilistic pushdown automata. In: Proc. LICS '04, 20^{th} IEEE Int. Symp. on Logic in Computer Science. pp. 12–21 (2004)
25. Esparza, J., Schwoon, S.: A BDD-based model checker for Büchi programs. In: CAV. LNCS, vol. 2102, pp. 324–336. Springer (2001)
26. Etessami, K., Yannakakis, M.: Algorithmic verification of recursive probabilistic state machines. In: Proc. TACAS '05, 11^{th} Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 3440, pp. 253–270 (2005)
27. Fribourg, L., Richardson, J.: Symbolic verification with gap-order constraints. In: LOPSTR '96. pp. 20–37 (1996)
28. Marcone, A.: Foundations of BQO theory. Transactions of the American Mathematical Society 345(2) (1994)
29. Reps, T.W., Schwoon, S., Jha, S., Melski, D.: Weighted pushdown systems and their application to interprocedural dataflow analysis. Sci. Comput. Program. 58(1-2), 206–263 (2005)
30. Reps, T.: Program analysis via graph reachability. Information & Software Technology 40(11-12), 701–726 (1998)
31. Revesz, P.Z.: A closed-form evaluation for datalog queries with integer (gap)-order constraints. TCS 116(1&2), 117–149 (1993)
32. Rot, J., de Boer, F.S., Bonsangue, M.M.: Pushdown System Representation For Unbounded Object Creation. Tech. Rep. KIT-13, Karlsruhe Institute of Technology (July 2010)
33. Schwoon, S.: Model-Checking Pushdown Systems. Ph.D. thesis, Technische Universität München (2002)
34. Verma, K.N., Seidl, H., Schwentick, T.: On the complexity of equational Horn clauses. In: CADE'05. pp. 337–352. LNCS 3632, Springer (2005)
35. Yannakakis, M.: Graph-theoretic methods in database theory. In: PODS '90. pp. 230–242 (1990)