Verifying Quantitative Temporal Properties of Procedural Programs

Mohamed Faouzi Atig
Uppsala University, Sweden

Ahmed Bouajjani[1]
IRIF, Paris Diderot University, France

K. Narayan Kumar[2]
Chennai Mathematical Institute and UMI RELAX, India

Prakash Saivasan
TU Braunschweig, Germany

Abstract

We address the problem of specifying and verifying quantitative properties of procedural programs. These properties typically involve constraints on the relative cumulated costs of executing various tasks (by invoking for instance some particular procedures) within the scope of the execution of some particular procedure. An example of such properties is "within the execution of each invocation of procedure P, the time spent in executing invocations of procedure Q is less than 20% of the total execution time". We introduce specification formalisms, both automata-based and logic-based, for expressing such properties, and we study the links between these formalisms and their application in model-checking. On one side, we define Constrained Pushdown Systems (CPDS), an extension of pushdown systems with constraints, expressed in Presburger arithmetics, on the numbers of occurrences of each symbol in the alphabet within invocation intervals (sub-computations between matching pushes and pops), and on the other side, we introduce a higher level specification language that is a quantitative extension of CaRet (the Call-Return temporal logic) called QCaRet where nested quantitative constraints over procedure invocation intervals are expressible using Presburger arithmetics. Then, we investigate (1) the decidability of the reachability and repeated reachability problems for CPDS, and (2) the effective reduction of the model-checking problem of procedural programs (modeled as visibly pushdown systems) against QCaRet formulas to these problems on CPDS.

2012 ACM Subject Classification Theory of computation → Logic and verification, Software and its engineering → Model checking, Software and its engineering → Software verification

Keywords and phrases Verification, Formal Methods, Pushdown systems, Visibly pushdown, Quantitative Temporal Properties

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2018.15

1 Introduction

Reasoning about performances requires checking properties on the cumulated costs of actions along program computations. Different types of costs can be considered corresponding to consumption of resources such as time, memory, energy, etc. To be able to reason about the action costs, amounts to the ability to count numbers of occurrences of different actions in

[1] Partially supported by Indo-French project AVeCSo
[2] Partially supported by Indo-French project AVeCSo, Infosys Foundation, DST-VR Project P-02/2014

computations (since weights can be associated to actions representing their various costs). Therefore, it is important to develop formal program models and specification languages (1) that allow the expression of counting constraints in different computation segments, and (2) that are useful for algorithmic verification of programs against quantitative properties involving these counting constraints. The goal of this paper is to propose such formalisms, both automata and logic based, for reasoning about the behaviors of procedural programs, i.e., sequential programs with (potentially recursive) procedure calls.

Quantitative properties of procedural programs are typically temporal properties including cost constraints on execution intervals corresponding to procedure invocations. An example of such a property is the invariant "within the execution of every terminating call to procedure P, the cumulated cost of executing all the calls by P to the procedure Q is less that 20% of the total cost of executing P". Then, formalisms for expressing such properties must have mechanisms allowing to express counting constraints in the scope of computation intervals between procedure calls and returns.

In the framework of automata-based formalisms, it is well know that pushdown systems (PDS) are natural models for procedural programs. Our first contribution is to introduce *Constrained Pushdown Systems* (CPDS), an extension of PDS by counting constraints on execution intervals between two matching push and pop operations (i.e., the push of an element to the stack, and the corresponding pop of that element from the stack). The counting constraints, expressed in Presburger arithmetics, concern the numbers of occurrences of the input alphabet symbols in the computation segment between these two matching operations. In order to impose these constraints, we consider an extended stack alphabet, where, in addition to plain stack symbols, push operations can push to the stack a pair of stack symbol γ and a counting constraint f. When a pair (γ, f) is popped from the stack, the automaton checks the satisfaction of the constraint f by the word read since it was pushed to the stack.

In the framework of logic-based formalisms, the temporal logic CaRet [6] has been introduced as a suitable specification formalisms for procedural programs. The second contribution is to introduce *Quantitative CaRet* (QCaRet), and extension of CaRet by counting constraints over procedure call-return intervals. Counting constraints within a call-return interval concern the cumulated lengths of the *outer-most* call-return intervals of each procedure. So, basically, QCaRet is the extension of CaRet with an operator W_f parametrized by a Presburger formula f. A QCaRet formula $W_f(\varphi)$ is satisfied at a point of a computation if that point corresponds to the call of a procedure, say P, and if both f and the QCaRet subformula φ are satisfied in the call-return interval of the procedure P. Notice that this allows nesting of temporal properties with counting constraints.

Then, we investigate the decision problems for these two formalisms. First, we prove that the reachability problem of CPDS is undecidable in general. However, we prove that under the assumption that the number of constraints in the stack is bounded – we call *constraint height-bounded CPDS* the class of CPDS corresponding to this assumption, the reachability problem becomes decidable and the same holds for the repeated reachability problem. Constraint height-bounded CPDS is a powerful class of automata allowing to express interesting non context-free languages. Interestingly, this class allows also to prove that QCaRet is decidable. Indeed, we show that the satisfiability problem of QCaRet can be reduced to solving repeated reachability in CPDS. The same reduction allows to show that the model-checking problem of procedural program, modeled as visibly PDS, against QCaRet formulas is decidable. A crucial point that leads to the decidability of the satisfiability and model-checking problems for QCaRet is the fact that counting constraints are about outermost calls of procedures in a call-return interval. This is necessary for the reduction

to constraint height-bounded CPDS decision problems. Another important and nontrivial contribution is that the complexity is shown to be elementary. Indeed, a more direct way of doing the reduction would use nested computations of Parikh images that would lead to a tower of exponentials depending on the size of the formula.

Related work. Extensions of word automata with counting constraints (such as Parikh automata and CQDDs) have been studied in the literature [21, 20, 12, 13, 19, 11]. These works cannot be used for reasoning about nested words (except [19] which extends visibly pushdown automata with reversal bounded counters). There are also works allowing to reason about unbounded-width trees using counting constraints (e.g., [22, 21, 23]), however, these constraints concerning the immediate successors of a node in a tree, cannot encode the type of constraints imposed by CPDS on nested words (that would correspond to global constraints on a whole subtree). CaRet is the first logic that was tailored to the specification of procedural programs. Extensions and variants of this logic have been proposed [7, 5], but none of them allow reasoning about quantitative properties. Extensions of temporal logics with counting constraints have been studied, e.g., in [21, 22], but again they are interpreted on words or on trees but without allowing to express (nested) constraints on nested words.

Several extensions of pushdown systems with either data or time have been studied in the literature (see e.g., [3, 2, 16, 4, 9, 14, 17, 1, 10, 15]). However, all these works are orthogonal to ours since they do not allow counting constraints on execution intervals between two matching push and pop operations.

2 Preliminaries

Let Σ be a finite alphabet. We use Σ^* and Σ^+ to denote the set of all finite words and non-empty finite words respectively over Σ; and use ϵ to denote the empty word. We also write Σ_ϵ for $\Sigma \cup \{\epsilon\}$. A language is a (possibly infinite) set of words. We let $|w|$ denote the length of the word w. Let $w = a_1 a_2 \dots a_n$. We write $w(i)$ for a_i and $w[i..j]$ for $w(i) \cdots w(j)$. For $w \in \Sigma^*$, $\Gamma \subseteq \Sigma$, we use $w{\downarrow_\Gamma} \in \Gamma^*$ for the projection of w on the Γ. We will also consider infinite words and languages of infinite words over Σ.

The set of linear constraints over a set V (written $\mathcal{C}(V)$) is the set of expressions of the form $c_1 x_1 + c_2 x_2 \dots c_k x_k \ \# \ 0$, where $x_i \in V$, $\# \in \{<, >, =\}$ and $c_i \in \mathbb{Z}$. The size of such a constraint φ, written $|\varphi|$, is sum of k and the number of bits needed to describe the sequence $c_1, c_2, ...c_k$. That is, we assume that the values c_i are provided in binary notation. A valuation v is a map that assigns a value from $\mathbb{N}$ to each element of V. We write $v \models \varphi$ to mean that φ is satisfied by the valuation v (and whose meaning is evident). We shall write $\mathcal{BC}(V)$ to denote formulas over $\mathcal{C}(V)$ constructed using $\wedge$ and $\vee$. The satisfaction relation $\models$ is extended to $\mathcal{BC}(V)$ in the obvious manner.

Given a word w over an alphabet Σ, we write $\pi(w)$ to denote the *Parikh map* defined by $\pi(w)(a)$ is the number of occurrences of a in w for all $a \in \Sigma$. Let L be a language over $\Sigma \uplus \Sigma'$ and let σ be a function from Σ' to languages over Σ. We write $\sigma(L)$ for the language $\{x_1 y_1 x_2 y_2 \dots y_k x_{k+1} \mid \forall i.\ x_i \in \Sigma^*, \exists a_1, \dots, a_k \in \Sigma'.\ x_1 a_1 x_2 a_2 \dots a_k x_{k+1} \in L, \forall i.\ y_i \in \sigma(a_i)\}$.

3 Constrained Pushdown Systems

A *constrained pushdown system (CPDS) A* is a tuple $(Q, \Gamma, \Sigma, \delta, s)$ where Q is the set of states, Γ is the stack alphabet, Σ is the tape alphabet, $s \in Q$ is the initial state and δ is the transition relation. We use $\perp \notin \Gamma$ to denote the stack bottom symbol. Let $\Gamma_C = \Gamma \times \mathcal{BC}(\Sigma)$

and $\Gamma_e = \Gamma \cup \Gamma_C$. The transition set δ is a subset of $Q \times \mathcal{SO} \times \Sigma \times Q$ where $\mathcal{SO}$ is the set of stack operations given by $\{push(X), pop(X), Y?, int \mid X \in \Gamma_e, Y \in \Gamma \cup \{\bot\}\}$. The operation $push(X)$ pushes X, with $X \in \Gamma_e$, on to the stack. The operation $pop(X)$ removes such an X from the stack. The $Y?$ operation checks if the current top of stack is either Y or in $\{Y\} \times \mathcal{BC}(\Sigma)$. Finally, the operation int is an internal action (i.e., independent of the stack). Thus, CPDS are PDS enriched with the ability to add constraints to stack symbols.

A configuration of the CPDS A is a pair (q, γ) with $q \in Q$ and $\gamma \in \Gamma_e^* \bot$. The *initial configuration* is the pair $(s, \bot)$. The transition relation $\xrightarrow{\tau}_A$, with $\tau \in \delta$, on the set of configurations is defined as follows: (1) if $\tau = (q, a, int, q')$ then $(q, \gamma) \xrightarrow{\tau}_A (q', \gamma)$ (Internal move), (2) if $\tau = (q, a, push(X), q')$ then $(q, \gamma) \xrightarrow{\tau}_A (q', X\gamma)$ (Push), (3) if $\tau = (q, a, pop(X), q')$ then $(q, X\gamma) \xrightarrow{\tau}_A (q', \gamma)$ (Pop), and (4) if $\tau = (q, a, Y?, q')$ then $(q, X\gamma) \xrightarrow{\tau}_A (q', X\gamma)$ if $X = Y$ or $X \in \{Y\} \times \mathcal{BC}(\Sigma)$ (Test). We often write $\rightarrow$ for $\rightarrow_A$ when A is clear from the context.

The transition relation extends naturally to sequences of transitions: $(q, \gamma) \xrightarrow{\epsilon} (q, \gamma)$ and $(q, \gamma) \xrightarrow{\sigma.\tau} (q', \gamma')$ if there is (q'', γ'') such that $(q, \gamma) \xrightarrow{\sigma} (q'', \gamma'')$ and $(q'', \gamma'') \xrightarrow{\tau} (q', \gamma')$. We call this an unconstrained run on the sequence of transitions σ. Given two unconstrained runs $\rho_1 = (q, \gamma) \xrightarrow{\sigma_1} (q', \gamma')$ and $\rho_2 = (q', \gamma') \xrightarrow{\sigma_2} (q'', \gamma'')$, the concatenation $\rho_1\rho_2$ denotes the unique run $(q, \gamma) \xrightarrow{\sigma_1.\sigma_2} (q'', \gamma'')$.

Let $\rho := (q_0, \gamma_0) \xrightarrow{\tau_1} (q_1, \gamma_1) \ldots, (q_{i-1}, \gamma_{i-1}) \xrightarrow{\tau_i} (q_i, \gamma_i) \ldots \xrightarrow{\tau_n} (q_n, \gamma_n)$ be an unconstrained run. We define a binary relation $\curvearrowright$ on positions in ρ as follows: $i \curvearrowright j$ then the ith transition is a push and the symbol pushed in this transition is popped by the jth transition. Formally $i \curvearrowright j$ if $0 < i < j \leq n$ and further $\tau_i = (q_{i-1}, a_i, push(X), q_i)$, $\tau_j = (q_{j-1}, a_j, pop(X), q_j)$, for each $i \leq k < j$, $\gamma_i = X\gamma_{i-1}$ is a suffix of γ_k and $\gamma_j = \gamma_{i-1}$.

Clearly, if $i \curvearrowright j$ and $i' \curvearrowright j$ then $i = i'$ and if $i \curvearrowright j$ and $i \curvearrowright j'$ then $j = j'$. We note that that if γ_0 is a suffix of γ_k for each $0 \leq k \leq n$ then for any pop transition, say j, there is a unique i such that $i \curvearrowright j$. In particular, this is true whenever $\gamma_0 = \bot$.

We are now in a position to define the *constrained runs* (or simply *runs*) of the CPDS A. An unconstrained run $\rho = (q_0, \gamma_0) \xrightarrow{\tau_1} (q_1, \gamma_1) \ldots, (q_{k-1}, \gamma_{k-1}) \xrightarrow{\tau_k} (q_k, \gamma_k) \ldots \xrightarrow{\tau_n} (q_n, \gamma_n)$ with $\tau_k = (q_{k-1}, a_k, op_k, q_k)$ is a constrained run if for every transition of the form $\tau_j = (q_{j-1}, a_j, pop(Y, \varphi), q_j)$ and $i \curvearrowright j$, we have $\pi(a_i \ldots a_j) \models \varphi$. Pushing a stack symbol of the form (Y, φ) enforces the requirement that the sequence of letters read from this transition upto the transition that pops this symbol from the stack satisfies the constraint φ. However, observe that constraints that are pushed on to the stack but not popped along the run do not place any requirements. In what follows, we shall write *run* to mean *constrained run*.

We also write $(q, \gamma) \xrightarrow{a,op} (q', \gamma')$ if there is a transition $\tau = (q, a, op, q')$ and $(q, \gamma) \xrightarrow{\tau} (q', \gamma')$ as this simplifies notation at many places. We write $(q, \gamma) \xrightarrow{w} (q', \gamma')$ if either $w = \epsilon$, $q = q'$ and $\gamma = \gamma'$ or $w = a_1 a_2 \ldots a_k$, $k \geq 1$, with $a_i \in \Sigma_\epsilon$ for $1 \leq i \leq k$, and further we can find configurations (q_i, γ_i) and operations op_i, $0 \leq i < k$, such that $(q, \gamma) = (q_0, \gamma_0) \xrightarrow{a_1, op_1} (q_1, \gamma_1) \cdots (q_{k-1}, \gamma_{k-1}) \xrightarrow{a_k, op_k} (q_k, \gamma_k) = (q', \gamma')$ is a run. We write $(q, \gamma) \xrightarrow{*} (q', \gamma')$ to mean that there is some w with $(q, \gamma) \xrightarrow{w} (q', \gamma')$.

Our aim is to study the *reachability problem* for CPDS. That is, given a CPDS A and a state $q \in Q$, determine whether there is a run $(s, \bot) \xrightarrow{*} (q, \gamma)$. We will also consider the *repeated reachability problem*, where the aim is to determine if there is an infinite run $(s, \bot) \xrightarrow{*} (q, \gamma_1) \xrightarrow{*} (q, \gamma_2) \ldots$ that visits the state q infinitely often.

We may equip a CPDS $A = (Q, \Gamma, \Sigma, \delta, s)$ with a set of accepting states $F \subseteq Q$ to obtain a constrained pushdown automaton (CPDA) $A' = (Q, \Gamma, \Sigma, \delta, s, F)$. The language, $\mathcal{L}(A)$, accepted by A' is defined naturally as $\{w \in \Sigma^* \mid (s, \bot) \xrightarrow{w} (q, \gamma), q \in F\}$. Clearly, the language emptiness problem for CPDAs is equivalent to the reachability problem for CPDS.

The CPDA model is quite expressive as indicated by the following examples.

- $L_1 = \{a^n b^m c^n d^m \mid n, m \geq 0\}$. Not a CFL, but recognized by Parikh Automata [20, 12].
- $L_2 = \{w\#w^R \mid w \in \{a,b\}^*, |\pi(w)(a) = \pi(w)(b)\}$. Not a CFL, not recognized by Parikh Automata, recognized by Parikh Pushdown Automata [20, 21]
- L_1^*, L_2^*. Unlikely to be recognizable by Parikh Pushdown Automata.
- $L^0 = \{w \in \{a_0, b_0\}^* \mid \pi(w)(a_0) = \pi(w)(b_0)\}$
  $L^{i+1} = \{w \in (a_{i+1}L^i + b_{i+1}L^i)^* \mid \pi(w)(a_{i+1}) = \pi(w)(b_{i+1})\}$.
  The automaton for L^i stores upto $i+1$ constraints at any point in the stack. Automata for L_1, L_2, L_1^* and L_2^* store at most one.

4 Visibly Pushdown Systems

Visibly Pushdown Systems (VPDS) [7] are natural formal model of procedural programs.

Formally, a VPDS is a PDS whose alphabet $\Sigma = \Sigma_\downarrow \cup \Sigma_\uparrow \cup \Sigma_L$ and any transition on a letter from $\Sigma_\downarrow$ must push a value on the stack, any transition on $\Sigma_\uparrow$ must pop a value from the stack and any transition on a letter from Σ_L must be an internal move or a test. Transitions on ϵ must also be internal or test moves and hence leave the stack unchanged. VPDSs have been extensively studied in literature and have several advantages over PDSs [7]. They enjoy a host of other algorithmic and language theoretic properties: the class of languages definable in the model is effectively closed under boolean operations, emptiness and universality are decidable.

We shall work with a specific variety of visible alphabets which makes explicit the set of procedures involved. Let Δ be a set of letters and Π be a set of *procedures*. The visible alphabet $\Sigma(\Delta, \Pi)$ is given by $\Sigma_\downarrow = \Delta \times \{call(P) \ : \ P \in \Pi\}$, $\Sigma_\uparrow = \Delta \times \{ret(P) \ : \ P \in \Pi\}$ and $\Sigma_L = \Delta \times \{int\}$. The words over such an alphabet that constitute behaviours of VPDSs have a particular form and we describe that now. A word σ over $\Sigma_\downarrow \cup \Sigma_\uparrow \cup \Sigma_L$ is well-nested if (1) for each prefix σ' of σ, $|\sigma'{\downarrow}\Sigma_\downarrow| \geq |\sigma'{\downarrow}\Sigma_\uparrow|$, and (2) if $\sigma(i) = (c, call(P))$, $\sigma(j) = (c', ret(P'))$ with $|\sigma[i..j]{\downarrow}\Sigma_\downarrow| = |\sigma[i..j]{\downarrow}\Sigma_\uparrow|$ then $P = P'$. In addition if σ is finite and has the same number of letters from $\Sigma_\downarrow$ and $\Sigma_\uparrow$ then we say it is a *complete well-nested word*.

We shall overload the symbol $\curvearrowright$ and write $i \curvearrowright j$ for positions i, j in a well-nested word σ if $\sigma(i) = (c, call(P))$, $\sigma(j) = (c', ret(P))$ with $|\sigma[i..j]{\downarrow}\Sigma_\downarrow| = |\sigma[i..j]{\downarrow}\Sigma_\uparrow|$. It captures the call-return relationship. We shall also write $i \curvearrowright_P j$ to explicitly indicate the associated procedure. Clearly if $i \curvearrowright j$ and $i' \curvearrowright j'$ then either the intervals $[i, j]$ and $[i', j']$ are completely disjoint or one is contained in the other. If $i \curvearrowright_P j$ and $[i, j]$ is not contained in any interval $[i', j']$ with $i' \curvearrowright_P j'$ then we say that $[i, j]$ is an *outermost* call to P in σ.

For any well-nested word σ, we define the map $\pi_{Pr}(\sigma)$ from $\Pi \cup \{\bot\}$ to $\mathbb{N}$ as follows: (1) $\pi_{Pr}(\sigma)(\bot) = |\sigma|$, (2) $\pi_{Pr}(\sigma)(P) = \sum\{j - i + 1 \mid [i, j]$ is an outermost call of P in $\sigma\}$. The function $\pi_{Pr}(\sigma)(P)$ computes the total length of all the outermost calls to the procedure P while $\pi_{Pr}(\sigma)(\bot)$ reports the length of σ. Notice that any word read by a VPDS along a run will be a well-nested word.

5 A Quantitative Extension of CaReT

We introduce in this section an extension of CaRet [6] which permits us to reason about quantitative properties of VPDSs using constraint formulas.

Let AP be a set of atomic propositions, and let Π be a finite set of procedure names. Then, we let $Prop = 2^{AP} \cup \{call(P), ret(P) \ : \ P \in \Pi\}$. We use $p, p_1, p_2, \ldots$ to refer to

elements of $Prop$ and $P, P', \ldots$ to refer to procedures in Π. We use $f, f_1, f_2, \ldots$ to refer to constraint formulas in $\mathcal{BC}(\Pi \cup \{\bot\})$. Formulas of *QCaRet* are given by the following syntax.

$$\varphi \ ::= \ p \,|\, \neg\varphi \,|\, \varphi \vee \varphi \,|\, \bigcirc^g \varphi \,|\, \bigcirc^a \varphi \,|\, \bigcirc^c \varphi \,|\, \varphi \mathcal{U}^g \varphi \,|\, \varphi \mathcal{U}^a \varphi \,|\, \varphi \mathcal{U}^c \varphi \,|\, W_f(\varphi)$$

The logic CaReT is the sub-logic without the W_f operator. As with CaReT, the formulas are interpreted over well-nested words, both finite and infinite, over a visible alphabet, in this case $\Sigma(AP, \Pi)$ where $\Sigma_\downarrow = 2^{AP} \times \{call(P) \ : \ P \in \Pi\}$, $\Sigma_\uparrow = 2^{AP} \times \{ret(P) \ : \ P \in \Pi\}$ and $\Sigma_L = 2^{AP} \times \{int\}$. For any well-nested word σ and position i, we define three different notions of successors as follows: (1) $suc_g(i)$ is $i+1$ if $|\sigma| > i$ and $\bot$ (to denote undefined) otherwise. (2) $suc_a(i) = suc_g(i)$ if $\sigma(i) \notin \Sigma_\downarrow$, $suc_a(i) = j$ if $\sigma(i) \in \Sigma_\downarrow$ and $i \curvearrowright j$, and $suc_a(i) = \bot$ otherwise. (3) $suc_c(i) = j$ if j is the largest number less than i for which there is a k with $j \curvearrowright k$ and $i \leq k$. $suc_c(i) = \bot$ if no such j exists. With all this we can define the semantics of the formulas w.r.t any well-nested word σ and any position i in σ:

$$\begin{array}{rll}
(\sigma, i) \models p & \textbf{iff} & \sigma(i) = (c, C),\ p \in c \cup \{C\} \\
(\sigma, i) \models \neg\varphi & \textbf{iff} & (\sigma, i) \not\models \varphi \\
(\sigma, i) \models \varphi_1 \vee \varphi_2 & \textbf{iff} & (\sigma, i) \models \varphi_1 \textbf{ or } (\sigma, i) \models \varphi_2 \\
(\sigma, i) \models \bigcirc^x \varphi & \textbf{iff} & suc_x(i) \neq \bot \textbf{ and } (\sigma, suc_x(i)) \models \varphi \text{ for } x \in \{a, c, g\} \\
(\sigma, i) \models \varphi_1 \mathcal{U}^x \varphi_2 & \textbf{iff} & \exists n.\, \exists i_0, i_1, \cdots, i_n.\, i_0 = i \textbf{ and} \\
 & & \forall k.\, 0 \leq k < n \textbf{ implies } \big(suc_x(i_k) = i_{k+1} \textbf{ and } (\sigma, i_k) \models \varphi_1\big) \\
 & & \textbf{and } (\sigma, i_n) \models \varphi_2 \\
(\sigma, i) \models W_f(\varphi) & \textbf{iff} & \exists j.\, i \curvearrowright j \textbf{ and } [i, j] \text{ is an outermost call in } \sigma \textbf{ and} \\
 & & \pi_{Pr}(\sigma[i+1, j-1]) \models f \textbf{ and } (\sigma[i+1, j-1], 1) \models \varphi
\end{array}$$

We say that $\sigma \models \varphi$ if $(\sigma, 1) \models \varphi$. We define the finite and infinite word languages defined by φ: $\mathcal{L}(\varphi) = \{\sigma \mid \sigma \models \varphi, |\sigma| < \infty\}$ and $\mathcal{L}^\omega(\varphi) = \{\sigma \mid \sigma \models \varphi, |\sigma| = \infty\}$.

In addition to properties expressible in CaRet, QCaRet allows to express (nested) quantitative constraints. Below are few examples of such QCaRet formulas (here $\Diamond^x(\Psi) = \text{True } \mathcal{U}^x \ \Psi$ and $\Box^x(\Psi) = \neg\Diamond^x(\neg\Psi)$):

- *For every outermost invocation of P, the time spent in executing outermost invocations to Q is less than 20% of the total execution time.*

  $$\Box^g(call(P) \wedge W_{\text{True}} \Rightarrow W_{(5Q \leq \bot)}\text{True})$$

- *For every outermost procedure execution interval where the cumulated time of executing Q is lower than half of the total execution time, the execution time of Q is less than the cumulated execution time of P in that same procedure interval execution, and there must be one invocation to Q in that interval that takes more that 5 time units*

  $$\Box^g\Big(W_{2Q \leq \bot}\text{True} \Rightarrow \big(W_{Q \leq P}(\Diamond^g(call(Q) \wedge W_{\bot > 5}\text{True}))\big)\Big)$$

For the logic CaReT obtained by omitting the W_f operator, it is known from [6, 5] that these languages are languages of Visibly Pushdown Automata (VPA) and Büchi Visibly Pushdown Automata (BVPA) respectively.

We investigate in the next sections the translation of QCaRet to (a visible version of) CPDSs. In [5], "qualitative" extensions of CaRet have been defined. We can extend them to quantitative versions in the same way as we did above by adding the operator W_f. The approach we will present in the rest of the paper can be applied in the same way to these extensions, and the results concerning decidability of the satisfiability and model checking problems and their complexity can be obtained for them in a similar way.

6 Reachability/Emptiness for CPDAs

In this section we shall examine the reachability problem for CPDSs (equivalently, the language emptiness problem for CPDAs). While the general problem is undecidable, we identify an interesting decidable under-approximation which provides an important tool in proving the decidability of QCaReT.

6.1 Undecidability of Reachability

In this section, we show that the reachability problem for CPDSs is undecidable, infact it is possible to simulate the runs of a 2 Counter Machine (2CM) using a CPDA over the alphabet $\Sigma = \{inc_1, inc_2, dec_1, dec_2, z_1, z_2\}$. Σ also also serves as its stack alphabet. The simulation proceeds in two phases. In the first, the CPDA guesses the sequence of transitions used in an accepting run of the 2CM, in reverse order, and pushes corresponding counter operations on the stack. While doing so, it also conjoins constraints with the decrements and test for zeros as follows: to simulating a decrement transition by counter i it pushes $(dec_i, (inc_i >= dec_i))$ and to simulate a test for zero on counter i it pushes $(z_i, (inc_i = dec_i))$ where $i \in \{1, 2\}$. This entire phase is executed without reading any input. For the guessed sequence of transitions (in reverse) to constitute an accepting run, we need to verify that the counters remained positive throughout the run and that all zero tests were successful. This is done in the second phase, where it repeatedly pops its stack and reads the same letter from the input tape till it reaches the empty stack and accepts if it does. The second phase processes operations of guessed accepting run in the correct order, and it is easy to see that the constraints inserted into the stack ensure that every zero test was indeed successful and none of the decrements resulted in a negative value for the counter, thus verifying the validity of the guessed run. This gives us the following theorem.

▶ **Theorem 1.** *The language emptiness problem for CPDAs (reachability problem for CPDS) is undecidable.*

6.2 Technical Preliminaries

Before we proceed to the under-approximation we introduce some notations and prove an useful technical lemma. Let $A = (Q, \Gamma, \Sigma, \delta, s)$ be an CPDS. We say that a run $(q_0, \gamma_0) \xrightarrow{\tau_1} (q_1, \gamma_1) \ldots, (q_{i-1}, \gamma_{i-1}) \xrightarrow{\tau_i} (q_i, \gamma_i) \ldots \xrightarrow{\tau_n} (q_n, \gamma_n)$ is a *weak X-run*, $X \in \Gamma_e \cup \{\bot\}$, if there is a γ such that $\gamma_0 = X\gamma$ and for each $0 \leq i \leq n$, $\gamma_i = \gamma_i' X\gamma$, that is, $X\gamma$ is a suffix of the stack contents of each configuration. In this case, for any γ', the following run $(q_0, X\gamma') \xrightarrow{\tau_1} (q_1, \gamma_1' X\gamma') \ldots, (q_{i-1}, \gamma_{i-1}' X\gamma') \xrightarrow{\tau_i} (q_i, \gamma_i' X\gamma') \ldots \xrightarrow{\tau_n} (q_n, \gamma_n' X\gamma')$ is also a weak X-run, where $\gamma_i' X\gamma = \gamma_i$, $1 \leq i \leq n$. Thus, we may say there is a weak X run from (q_0, X) to (q_n, X) to mean that there is such a run, without being specific about γ. We call such a run a *weak X-run from q_0 to q_n*. Further, if $\gamma_n = X\gamma = \gamma_0$ we say call it an X-run.

We let $\mathcal{L}^X_{q,q'}(A)$, $q, q' \in Q$, be the set of words w such that there is a weak X-run from q to q' on w. A CPDA for $\mathcal{L}^X_{q,q'}(A)$ can be constructed easily from A. The desired automaton $A^X_{q,q'}$ is $(Q, \Sigma, \Gamma, \delta^X_{q,q'}, q, \{q'\})$ where $\delta^X_{q,q'} = \delta \setminus \{(r, c, O, r') \mid O = (?\bot)\} \cup \{(r, c, (?\bot), r') \mid (r, c, (?X), r') \in \delta\}$. It treats $\bot$ as the symbol X for tests, never pops this "X" and never succeeds on a test for $\bot$. The size of the new automaton is linear in the size of A.

Another language of particular interest is the following: Suppose $X = (Y, \varphi)$ and further that $\tau = (p, a, push(X), q)$ and $\tau' = (p', b, pop(X), q')$ are transitions involving the push and pop of the same symbol. Then, we let $\mathcal{L}^X_{\tau,\tau'}(A)$ be $\{w = a.y.b \mid$ there is an X-run on y from q to $p'\}$. This identifies the languages of words recognized by runs consisting of τ, followed by

an X-run from q to p', followed by τ'. Please note that we use X-runs and not weak X-runs here. Finally, observe that this definition does not require that φ is satisfied by $\pi(ayb)$ (while the constraints along the run on y are enforced). This language is accepted by the CPDA $A^X_{\tau,\tau'} = (Q \cup \{s_a, t_b\}, \Sigma, \Gamma, \delta^X_{\tau,\tau'}, s_a, \{t_b\})$ where $\delta^X_{\tau,\tau'} = \{(s_a, a, int, q), (q', b, \bot?, t_b)\} \cup \delta \setminus \{(r, c, O, r') \mid O = (?\bot)\} \cup \{(r, c, (?\bot), r') \mid (r, c, (?X), r') \in \delta\}$. The size of this automaton is linear in the size of A. To summarize,

▶ **Lemma 2.** *Let $A =$ be a CPDS and let $X \in \Gamma_e$. Then for any $q, q' \in Q$, the language $\mathcal{L}^X_{q,q'}(A)$ is recognized by a CPDA $A^X_{q,q'}$ whose size is linear in A. Further, for any $X = (Y, \varphi) \in \Gamma_C$ and $\tau = (p, a, push(X), q), \tau' = (p', b, pop(X), q') \in \delta$, the language $\mathcal{L}^X_{\tau,\tau'}(A)$ is recognized by a CPDA $A^X_{\tau,\tau'}$ whose size is linear in A.*

6.3 Constraint height Bounded CPDAs

The *constraint height* of a configuration (q, γ) is defined by $|\gamma{\downarrow_{\Gamma \times \mathcal{C}(\Sigma)}}|$ (i.e., the number of constraint symbols in the stack). The constraint height of a finite run ρ is the maximum of the constraint heights of the configurations visited along ρ. The constraint height of an infinite run is defined similarly, with ∞ acting as the upper bound of the set of all integers.

For any CPDS A, we say that a state q is K *constraint height reachable* (or K*-reachable* for the sake of succinctness) if there is a run $(s, \bot) \xrightarrow{*} (q, \gamma)$ whose constraint height is bounded by K. The K-reachability problem is to determine if there is such a run. Similarly, for any CPDA A, $\mathcal{L}_K(A)$ is the set of all words accepted by runs with constraint height bounded by K. Note that all the example languages listed at the end of Section 3 are constraint height bounded.

When $K = 0$ we are effectively left with the pushdown system obtained by removing all the transitions involving constraints. Thus, 0-reachability is clearly decidable. Our main technical result is that the K-reachability problem for CPDS (or equivalently, the emptiness of $\mathcal{L}_K(A)$ for CPDAs) is decidable for any $K \geq 0$. Our proof of decidability establishes a stronger property as stated in the following theorem:

▶ **Theorem 3.** *Let A be a CPDA and let $K \in \mathbb{N}$. Then, $\pi(\mathcal{L}_K(A))$ is effectively semilinear and a finite-state automaton M with the same Parikh image can be computed in* 2-EXPTIME.

The rest of this section is devoted to the proof of this theorem. As a first step, we recall the Parikh's Theorem which states that the language of a pushdown automaton A can be simulated by a Nondeterministic Finite Automaton (NFA) upto Parikh-image equivalence.

▶ **Lemma 4** ([18])**.** *Let A be a pushdown automaton. We can construct an NFA M such that $\pi(\mathcal{L}(M)) = \{\pi(w) \mid w \in \mathcal{L}(A)\}$ and the size of M is bounded by $2^{p(|A|)}$ for a polynomial p.*

We now make use of a result from [8] to extend this to CPDAs.

▶ **Lemma 5.** *Let M be an NFA over Σ and φ be a formula in $\mathcal{BC}(\Sigma)$. Then we can construct an NFA M' with size bounded by $2^{|\varphi|}.(|M|.2^{|\varphi|})^{|\Sigma|^{d.k}}$ for some constant d, such that $\pi(\mathcal{L}(M')) = \pi(\{w \mid w \in \mathcal{L}(M)\ \&\ \pi(w) \models \varphi\})$, and where k is the depth of the formula and hence bounded by $|\varphi|$.*

The next lemma lists a couple of simple results about substitutions and Parikh-images.

▶ **Lemma 6.** *Let L be a language over $\Sigma \uplus \Sigma'$ and let σ assign a language $\sigma(a)$ over Σ for each $a \in \Sigma'$.*

1. *If L' is Parikh-equivalent to L and L'_a is Parikh-equivalent to $\sigma(a)$ for each $a \in \Sigma'$ then, $\sigma'(L')$, with $\sigma'(a) = L'_a$, is Parikh equivalent to $\sigma(L)$.*

2. *If M is an NFA for L and M_a is an NFA for $\sigma(a)$, $a \in \Sigma'$, then we can obtain an NFA for $\sigma(L)$ by replacing each transition on any letter $a \in \Sigma'$ by a copy of M_a. Thus, there is an NFA for $\sigma(L)$ whose size is bounded by $|M|(Max_{a\in\Sigma'}|M_a|)$.*

We now have the technical ingredients in place to address the proof of Theorem 3. We begin by observing that, given a CPDA $A = (Q, \Sigma, \Gamma, \delta, s, F)$ and a number K we can construct an CPDA $A[K]$ such that $\mathcal{L}_K(A) = \mathcal{L}_K(A[K]) = \mathcal{L}(A[K])$. Further, $A[K]$ faithfully records information regarding the constraint height of the configuration in its control state. This automaton is defined as follows: $A[K] = (Q\times\{0, 1, \ldots, K\}, \Sigma, \Gamma, \delta^K, (s, K), F\times\{0, 1, \ldots, K\})$. The transition relation δ^K is defined as follows:

- $((q, i), a, push((Y, \varphi)), (q', i-1)) \in \delta^K$ whenever $(q, a, push((Y, \varphi)), q') \in \delta$ and $1 \leq i \leq K$
- $((q, i), a, pop((Y, \varphi)), (q', i+1)) \in \delta^K$ whenever $(q, a, pop((Y, \varphi)), q') \in \delta$ and $0 \leq i < K$
- $((q, i), a, O, (q', i)) \in \delta^K$ whenever $(q, a, O, q') \in \delta$ and O does not involve constraints.

The transition relation δ^K faithfully simulates δ, moves from copy i to copy $i-1$ while pushing a constraint and moves from copy i to copy $i+1$ on popping a constraint. The constraint height of any reachable configuration with control state (q, i) is therefore $K - i$. A state of the form $(q, 0)$ does not permit pushing any constraint symbol. Also note that, for any $0 \leq j \leq K$, δ^j is just δ^K restricted to the state space of $A[j]$ $(Q \times \{0, 1, \ldots, j\})$.

Our strategy for the proof of Theorem 3 is the following: We will argue by induction on j, $0 \leq j \leq K$ that for any symbol $X = (Y, \varphi) \in \Gamma_C$ and any pair of transitions $\tau, \tau' \in \delta^j \setminus \delta^{j-1}$ which push and pop X respectively, we can construct an NFA Parikh-equivalent to $\mathcal{L}^X_{\tau,\tau'}(A[j])$ whose size is bounded by a function $f(j)$. We shall then derive an expression bounding $f(j)$, $j \leq K$. This will form a key ingredient in the proof of Theorem 3.

We observe that if $j = 0$ then there no transitions that push (or pop) symbols Γ_C and hence nothing is to be proved. We take $f(0) = 1$ (since $\mathcal{L}^X_{\tau,\tau'}(A[0]) = \emptyset$).

We proceed inductively as follows: We write B for $A[j]^X_{\tau,\tau'}$ to simplify the notation. We construct a simple pushdown automaton P from B. This automaton simulates B on all transitions other than those that push/pop elements of Γ_C. From any state $(p, j-1)$, instead of executing a push transition of the form $\mu = ((p, j-1), c, push((Z, \psi)), (p', j-2))$ it nondeterministically guesses a corresponding pop transition $\mu' = ((q, j-2), d, pop((Z, \psi)), (q', j-1))$ (which must exist along any accepting run of $B = A[j]^X_{\tau,\tau'}$ – as the run must return to level j before acceptance) and simply *outputs* (i.e. reads from the input tape) a symbol $(\mu, (Z, \psi), \mu')$ to indicate this guess and changes state to $(q', j-1)$. Thus, this automaton does not need states of the form (p, i) for $p \in Q$ and $i < j-1$ and it never leaves "level j-1" (except when executing the transitions τ andτ').

Let $\Sigma_C[j] = \{(\mu, (Z, \psi), \mu') \mid \exists p, p', q, q'.\ \mu = ((p, j-1), c, push((Z, \psi)), (p', j-2))$ and $\mu' = ((q, j-2), d, pop((Z, \psi)), (q', j-1))\}$. The alphabet of P is $\Sigma \cup \Sigma_C[j]$ and its stack alphabet Γ. Let s_a and t_b be the initial and final states of B (recall the definition of $A[j]^X_{\tau,\tau'}$ from Section 3), with a the letter read by τ and b the letter read by τ'. Then, $P = (\{s_a, t_b\} \cup (Q \times \{j\}), \Sigma \cup \Sigma_C[j], \Gamma, s_a, \Delta, \{t_b\})$ and Δ is given by

$$\delta[j]^X_{\tau,\tau'} \setminus \{(p, c, O, p') \mid \exists (Z, \psi).O = push((Z, \psi)) \text{ or } O = pop((Z, \psi))\} \bigcup$$
$$\{((p, j-1), (\mu, (Z, \psi), \mu'), (q', j-1)) \mid \mu = ((p, j-1), c, push((Z, \psi)), (p', j-2)),$$
$$\mu' = ((q, j-2), d, pop((Z, \psi)), (q', j-1)),\ \mu, \mu' \in \delta[j]^X_{\tau,\tau'}\}$$

Fact 1. With $\sigma((\mu, (Z, \psi), \mu')) = \mathcal{L}^{(Z,\psi)}_{\mu,\mu'}(A[j-1])$, $\forall(\mu, (Z, \psi), \mu') \in \Sigma_C[j]$, $\mathcal{L}(B) = \sigma(\mathcal{L}(P))$.

We now construct an NFA M_P Parikh-equivalent to P using Lemma 4. Then using the inductive hypothesis we construct, for each pair of transitions μ,μ' that push and pop,

respectively, the same symbol $(Z, \psi) \in \Gamma_C$, an NFA $M'_{\mu,\mu'}$ Parikh-equivalent to $\mathcal{L}^{(Z,\psi)}_{\mu,\mu'}(A[j-1])$. Then, we apply Lemma 5 to obtain an NFA $M_{\mu,\mu'}$ a language Parikh-equivalent to $\{w \in L(M'_{\mu,\mu'}) \mid \pi(w) \models \psi\}$. We let σ' be the map assigning $\mathcal{L}(M_{\mu,\mu'})$ to $(\mu, (Z, \psi), \mu')$. Then, by Lemma 6, $\sigma(\mathcal{L}(P))$ is Parikh-equivalent to $\sigma'(\mathcal{L}(M_P))$. Thus, by Fact 1, $\sigma'(\mathcal{L}(M_P))$ is Parikh-equivalent to $\mathcal{L}(B)$.

The state size of M_P is bounded by $2^{p(|B|)}$ for some polynomial p. But the state space of B is linear in the state space of A, its alphabet is polynomial in the size of A and its number of transitions also polynomial in the size of A. Thus, the size of the state space of M_P is bounded by $2^{r(|A|)}$ for some polynomial r. The number of transitions is bounded by the product of the size of the alphabet and the number of pairs of states. The number of new letters is quadratic in the number of transitions of A (in $(\mu, (Z, \psi), \mu')$, the value of (Z, ψ) is determined by μ and μ'). Thus the number of transitions is also bounded by $2^{r(|A|)}$ for some polynomial r. Equivalently it is bounded by $2^{|A|^c}$ for some fixed constant c.

The size of each automaton of the from $M'_{\mu,\mu'}$, by the induction hypthesis, is bounded by $f(j-1)$. Then, by Lemma 5, the size of $M_{\mu,\mu'}$ is bounded by $2^{|\varphi|}.(f(j-1).2^{|\varphi|})^{|\Sigma|^{d|\varphi|}}$ for some constant d. Thus, by Lemma 6, we have an NFA Parikh-equilvalent to $\mathcal{L}(B)$ whose size is bounded by $2^{|A|^c}.2^{|\varphi|}.(f(j-1).2^{|\varphi|})^{|\Sigma|^{d|\varphi|}}$. Simplifing, we get the following recurrences for $f(j)$: (1) $f(0) = 1$, and (2) $f(j) = 2^{|A|^c}.2^{|\varphi||\Sigma|^{d|\varphi|}+1}.f(j-1)^{|\Sigma|^{d|\varphi|}}$.

This gives $f(j)$ an upperbound of the form $\mathcal{O}(2^{\mathcal{O}(|A|^c.|\Sigma|^{d|\varphi|}.j)}.2^{\mathcal{O}(|\varphi||\Sigma|^{d|\varphi|}.j)}$. Thus, we have the following Lemma.

▶ **Lemma 7.** *There is an NFA Parikh-equivalent to $\mathcal{L}^X_{\tau,\tau'}(A[j])$, $0 \le j \le K$, whose size is bounded by $\mathcal{O}(2^{\mathcal{O}(|A|^c.|\Sigma|^{d|\varphi|}.j)}.2^{\mathcal{O}(|\varphi||\Sigma|^{d|\varphi|}.j)})$.*

Next we observe that to compute the Parikh-image of $\mathcal{L}^X_{(q,j),(q',j)}(A[j])$ for any $X \in \Gamma_e$, $q \in Q$ and $0 \le j \le K$, we may proceed as follows: Any weak X run from (q, j) to (q', j) can be broken up as, a segment involving no pushing or popping of letters from Γ_C, followed by a segment from the push of a symbol from Γ_C all the way till corresponding pop, followed by another segment involving no push or pop of letters from Γ_C, followed by one beginning with a push of a constraint and ending with the corresponding pop, and so on. (Recall that (q, j) and (q', j) are at the same level j). In particular, any symbol from Γ_C that is pushed must also be popped along such a run. We can use the same idea as in the proof of Lemma 7 and summarize the segments between push and the corresponding pop of $(Z, \psi) \in \Gamma_C$ with a letter of the form $(p, j), (\mu, (Z, \psi), \mu'), (p', j))$ and construct a simple pushdown system with no constraints. We then compute the Parikh-image of this system. Finally, we substitute these letters with the language of the corresponding NFAs computed in Lemma 7, and use Lemma 6 to obtain the desired NFA. This gives us the following Lemma.

▶ **Lemma 8.** *For any $X \in \Gamma_e$ and any pair of states $(q, j), (q', j)$ in $A[j]$, there is an NFA Parikh-equivalent to $\mathcal{L}^X_{(q,j),(q',j)}(A[j])$, whose size is bounded by $O(2^{\mathcal{O}(|A|^c.|\Sigma|^{d|\varphi|}.j)}.2^{\mathcal{O}(|\varphi||\Sigma|^{d|\varphi|}.j)})$.*

Now, we are in a position to complete the proof of Theorem 3. Suppose an accepting run of A reaches an accepting configuration (f, γ) where the constraint-height of γ is 0. Then, the corresponding run in $A[K]$ is a weak $\perp$-run from (s, K) to (f, K). Its emptiness can be checked using Lemma 8 by checking the emptiness of a double exponential sized NFA.

If the constraint-height of γ is j with $1 \le j \le K$ then, the corresponding run in $A[K]$ is a run from the state (s, K) to the state $(f, K - j)$.

We break up this run as follows (we let $\tau_\ell = ((q_\ell, \ell), a_\ell, push((Y_\ell, \varphi_\ell)), (p_{\ell-1}, \ell-1))$):

$$((s,K),\bot) \xrightarrow{w_1} ((q_K,K),\gamma_K) \xrightarrow{\tau_K} ((p_{K-1},K-1),(Y_K,\varphi_K)\gamma_K) \xrightarrow{w_2} ((q_{K-1},K-1),\gamma_{K-1})$$
$$\xrightarrow{\tau_{K-1}} \quad \ldots ((q_{j+1},j+1),\gamma_{j+1}) \xrightarrow{\tau_j} ((p_j,j),(Y_{j+1},\varphi_{j+1})\gamma_{j+1}) \xrightarrow{w_j} ((f_j,j),\gamma_j)$$

Here, we have identified that transitions that transfer a run from a state from at k to a state at level $k-1$ for the last time along the run, for each $K \geq k \geq j+1$. The existence of such a run is equivalent to firstly the existence of transitions $((q_k,k), a_k, push((Y_k,\varphi_k)), (p_{k-1}, k-1))$ for $K \geq k > j$ and secondly, the existence of a weak $\bot$-run from (s,K) to (q_K,K), (Y_{k+1},φ_{k+1})-run from (p_k,k) to (q_k,k) for $K > k > j$, and a weak (Y_{j+1},φ_{j+1})-run from (p_j,j) to (f_j,j). Once the transition sequence in the first part is fixed (and we cycle through there at most $|A|^j$ such sequences one by one), the existence of each of the weak runs in the second part can be determined using Lemma 8. Thus, we make at the most $|A|^j.j$ calls to the emptiness of NFAs of double exponential size and this can also be done in double exponential space. This completes the proof of Theorem 3. The following theorem provides a lower bound.

▶ **Theorem 9.** *The K-reachability problem for CPDS is* PSPACE*-hard.*

We end the section with the following theorem about decidability of repeated reachability.

▶ **Theorem 10.** *Let $A = (Q, \Gamma, \Sigma, \delta, s, F)$ be a CPDA let $K \in \mathbb{N}$. The problem of deciding if A has a K constraint height bounded infinite run that visits F infinitely often is decidable in* 2-EXPTIME.

7 Visible CPDS with procedural constraints

In this section, we develop a variant of our CPDS model with a view to establish the decidability of the logic QCaReT. The model by itself is interesting in its ability to model visible behaviours of recursive programs equipped with constraints. This model is a natural extension of the VPA model to our setting.

A *procedural CPDS (or PCPDS) A* is tuple $(Q, \Delta, \Pi, \Gamma, \delta, s)$. Its input tape alphabet is the visible alphabet $\Sigma(\Delta, \Pi)$. It is very similar to a CPDS over this alphabet, except for the language of constraints it uses (and their interpretation). The set of symbols that are pushed/popped is $\Gamma_P = \Pi \times (\Gamma_{PC} \cup \Gamma)$ where $\Gamma_{PC} = \Gamma \times \mathcal{BC}(\Pi \cup \bot)$. In particular, a push transition on an input letter $(c, call(P))$ must necessarily push a letter of the form (P, Z) for some $Z \in \Gamma_{PC} \cup \Gamma$. Similarly for pop transitions. Also note that the constraints only refer to the procedure in the input (and not to elements of the tape alphabet Δ).

The notions of configurations and unconstrained runs are defined as in the case of a CPDS. The key difference is in the interpretation of the constraints and thus in the definition of constrained runs. We note that any word (or prefix of a word) read by a PCPDS is necessarily well-nested.

An unconstrained run $\rho = (q_0,\gamma_0) \xrightarrow{\tau_1} (q_1,\gamma_1) \ldots, (q_{i-1},\gamma_{i-1}) \xrightarrow{\tau_i} (q_i,\gamma_i) \ldots \xrightarrow{\tau_n} (q_n,\gamma_n)$ with $\tau_k = (q_{k-1}, (c_k, P_k), o_k, q_k)$, $1 \leq k \leq n$, is a constrained run if for every transition τ_j with $o_j = pop((P,(Y,\varphi)))$ and $i \curvearrowright j$ (so that $P_i = call(P)$ and $P_j = ret(P)$ for some $P \in \Pi$) we have $\pi_{Pr}((c_{i+1}, C_{i+1}) \ldots (c_{j-1}, C_{j-1})) \models \varphi$. Observe here that the enclosing call and return points are omitted when checking the constraints, unlike CPDAs.

The reachability problem (as well as the associated language emptiness problem) for this model are defined as usual. These problems remain undecidable in general. We define the constraint height of configurations and runs of PCPDS analogous to those for CPDS. A configuration (or control state) is K *constraint height reachable* or K*-reachable*, if it can be

reached (from the initial configuration) through a run where the constraint height is bounded by K. The main result of this section is the following:

▶ **Theorem 11.** *The K-reachability for PCPDS is decidable and is in* 2-EXPTIME.

Proof-outline. Our main idea is the following: reduce the K-reachability in a PCPDS to K-reachability in a CPDS and use Theorem 3. Let $A = (Q, \Delta, \Pi, \Gamma, \delta, s)$ be the given PCPDS. The main difficulty is that, unlike in a CPDS, a constraint φ in a PCPDS is not expressed in terms of the tape letters read along the run, but instead it is expressed in terms of the number of transitions executed inside various procedures. We plan to handle this by using a more elaborate tape alphabet.

Consider a segment of a run of A of the form

$$\rho = (q_0, \gamma_0) \xrightarrow{\tau_1} (q_1, \gamma_1) \ldots, (q_{i-1}, \gamma_{i-1}) \xrightarrow{\tau_i} (q_i, \gamma_i) \ldots \xrightarrow{\tau_n} (q_n, \gamma_n)$$

where $1 \curvearrowright_{(P,(Y,\varphi))} n$. We shall focus our attention on verifying the constraint φ on this run (and ignore the verification of the other constraints pushed and popped along the run). Let $a_1 \ldots a_n$, $a_i \in \Sigma(\Delta, \Pi)$, be the word read on the tape in this run. Suppose, φ refers to $R \in \Pi$. We need to "determine" the value of $\pi_{Pr}(a_2 a_3 \ldots a_{n-1})(R)$. Our idea is to replace each letter a_i by an enriched version b_i (from an extended alphabet Σ') so that we may determine the value of $\pi_{Pr}(a_2 a_3 \ldots a_{n-1})(R)$, for each R, from $\pi(b_1 b_2 b_3 \ldots b_{n-1} b_n)$ and also replace the formula φ over Π with an equivalent formula over Σ'. Once we perform this transformation, the satisfaction of the constraint depends on the (enriched) letters read along the run and thus we have a obtained a CPDS instead of a PCPDS.

Observe that the contents of the stack at each configuration along ρ can be written as: $\gamma_i = \gamma'_i(P, (Y, \varphi))\gamma_0$, for all $1 \leq i \leq n-1$ and $\gamma_0 = \gamma_n$. The value of $\pi_{Pr}(a_2 a_3 \ldots a_{n-1})(R)$, for any R, is exactly the number of transitions taken from configurations where γ'_k includes an occurrence of an element of $\{R\} \times (\Gamma \cup \Gamma_C)$.

The automaton A' that we construct will simulate A and maintain additional information in its state and stack. Using this information, it outputs, in addition to a_i, the set $S_i \subseteq \Pi$ of the set of procedure symbols that appear in γ'_{i-1}. Thus, taking $b_i = (a_i, S_i)$, the value of $\pi_{Pr}(a_2 a_3 \ldots a_{n-1})(R)$ is the same as

$$\sum_{a \in \Sigma(\Delta,\Pi), R \in S \subseteq \Pi} \pi(b_1 b_2 b_3 \ldots b_{n-1} b_n)(a, S)$$

Using this equivalence we transform φ into a formula over the letters of the form (a, S).

This idea can easily be generalized to handle all constraints that are pushed/popped along a run by using the fact that the run is constraint height bounded. ◀

Furthermore, we can extend this result to the repeated reachability problem as follows: The CPDS A' constructed from the PCPDS A in Theorem-11 simulates A and in doing so maintains the current state of A as part of its state in each step of the simulation. The automaton A has a K constraint height bounded run visiting q infinite often if and only if A' has a K constraint height bounded run visiting some state in which q appears, infinitely often. By Theorem 10, this is decidable. Thus we have the following theorem.

▶ **Theorem 12.** *The K constraint bounded repeated reachability problem for PCPDS is decidable and is in* 2-EXPTIME.

8 Decidability of QCaReT

In the following, we show the decidability of the model-checking of our logic QCaReT. To that aim, we will need first to recall come algorithmic properties of the logic CaReT.

▶ **Theorem 13** ([6, 5]). *For any CaReT formula φ there is a VPA A_φ and a BVPA B_φ such that $\mathcal{L}(\varphi) = \mathcal{L}(A_\varphi)$ and $\mathcal{L}^\omega(\varphi) = \mathcal{L}(B_\varphi)$. Further, A_φ and B_φ are only exponentially larger than φ.*

VPAs and BVPAs are closed under intersection and have decidable emptiness problem [7]. This immediately gives decision procedures for checking the satisfiability of CaReT formulas as well as for model-checking VPAs/BVPAs w.r.t. CaReT formulas. Our aim is to lift these results to QCaReT and PCPDAs. We now utilize the theory of PCPDAs developed in the previous section to provide algorithms for deciding the satisfiability of QCaReT formulas as well their model checking w.r.t. PCPDAs (and hence VPAs as well).

For any formula φ in QCaReT, we may define its *depth*, denoting the maximum nesting of the operator W_f in it, as follows: $\mathbf{d}(p) = 0$, $\mathbf{d}(\neg\varphi) = \mathbf{d}(\varphi)$, $\mathbf{d}(\varphi_1 \vee \varphi_2) = max(\mathbf{d}(\varphi_1), \mathbf{d}(\varphi_2))$, $\mathbf{d}(\bigcirc^x \varphi) = \mathbf{d}(\varphi)$, $\mathbf{d}(\varphi_1 \mathcal{U}^x \varphi_2) = max(\mathbf{d}(\varphi_1), \mathbf{d}(\varphi_2))$ and $\mathbf{d}(W_f \varphi) = 1 + \mathbf{d}(\varphi)$.

We shall construct a PCPDS A_φ with $\mathcal{L}(\varphi) = \mathcal{L}(A_\varphi)$ as well as a Büchi PCPDS B_φ with $\mathcal{L}^\omega(\varphi) = \mathcal{L}(B_\varphi)$. We do so by proceeding inductively on the depth of formula φ.

If $\mathbf{d}(\varphi) = 0$, then φ is in CaReT and the associated automata are given by Theorem 13. Otherwise, we first turn φ into a CaReT formula as follows: Let $\mathcal{W} = \{W_{f_1}(\psi_1), \ldots, W_{f_k}(\psi_k)\}$ be the set of outer-most W_f formulas (that is, not within the scope of another W_f operator) in φ. We obtain φ' by replacing $W_{f_i}(\psi_i)$ by a new propositional variable $p(f_i, \psi_i)$. Let $AP' = \{p(f_i, \psi_i) \mid 1 \leq i \leq k\}$. Clearly, φ' is a CaReT formula over the set of propositions $AP \cup AP'$ and the set of procedures Π.

Let $\sigma\Uparrow$, for any well-nested word σ over $\Sigma(AP, \Pi)$, be the well-nested word over $\Sigma(AP \cup AP', \Pi)$ given by $\sigma\Uparrow(i) = (P', Y)$ where $\sigma(i) = (P, Y)$, $P' = P \cup \{p(f, \psi) \in AP' \mid (\sigma, i) \models W_f(\psi)\}$. It extends the labelling, interpreting $p(f, \psi)$ as the formula $W_f(\psi)$. Similarly $\sigma'\Downarrow$, for any well-nested word σ' over $\Sigma(AP \cup AP', \Pi)$, is the well-nested word over $\Sigma(AP, \Pi)$ given by $\sigma'\Downarrow(i) = (P, Y)$ where $\sigma'(i) = (P', Y)$ with $P = P' \cap AP$. It restricts the labels to the propositions in AP. Observe that $\sigma = \sigma\Uparrow\Downarrow$. The following lemma, whose proof is omitted, is an easy consequence of our construction:

▶ **Lemma 14.** *For any well-nested word σ over $\Sigma(AP, \Pi)$ $\sigma \models \varphi$ iff $\sigma\Uparrow \models \varphi'$.*

Now, with this Lemma in place, we proceed by constructing the VPA $A_{\varphi'}$ using Theorem 13 and use this in the construction of A_φ. The automaton A_φ does the following: It simulates $A_{\varphi'}$ by guessing a set of propositions from AP' at each step and verifies that its guess at each step is correct. That is, while reading a well-nested word σ over $\Sigma(AP, \Pi)$, (i) it simulates $A_{\varphi'}$ on a word σ' with $\sigma'\Downarrow = \sigma$ (ii) it verifies that $\sigma' = \sigma\Uparrow$. This would then mean, by Lemma 14, that A_φ accepts the language $\mathcal{L}(\varphi)$. We now describe the details of how to build an automaton satisfying (i) and (ii).

Clearly, (i) can be achieved by nondeterministically guessing a set of propositions from AP' at each step. The difficulty is in ensuring (ii), that is, for each i, $1 \leq i \leq |\sigma|$, if $C'_i \subseteq AP'$ is the set of propositions guessed in the ith step verify that $\sigma, i \models W_f(\psi)$ for each $p(f, \psi) \in C'_i$ and that $\sigma, i \not\models W_f(\psi)$ for each $p(f, \psi) \in AP' \setminus C'_i$. Let us examine the conditions under which $\sigma, i \models W_f(\psi)$. This requires the following properties:

1. $\sigma(i)$ must be in $\Sigma_\downarrow$. Say $\sigma(i) = (c, call(P))$.
2. This must be an outer-most call to P in σ.

3. There is a j with $i \curvearrowright_P j$ in σ.
4. $\sigma[i+1, j-1] \models \psi$
5. $\pi_{Pr}(\sigma[i+1, j-1]) \models f$.

Truth of item 1 is determined from the letter $\sigma(i)$ and so is easy to check. For item 2, we shall add a component to the state space of $A_{\varphi'}$ that keeps track of the list of procedures from Π that are currently active. To maintain this set correctly, we expand the stack alphabet of $A_{\varphi'}$ to tag the bottom-most occurrence of each procedure. With this modification we can determine, while reading an input letter $(c, call(P))$ whether it is an outer most call to P or not. Thus, w.l.o.g. we may assume this information is available with the simulation of $A_{\varphi'}$ and hence the truth of item 2 can be determined.

This leaves us with items 3,4 and 5. The truth of these items depends not only on the letter at i (and information about outer most calls stored in the state), but on the existence of a suitable j (as required by item 3) and the word read between positions i and j (to determine items 4 and 5). The automaton guesses whether such a j exists (and then ensures that along any accepting run, the guess is indeed correct).

If it guesses that such a j does not exist (it does so only if it also guessed $C'_i = \emptyset$, as implied by the semantics of the W_f operator) then instead of pushing the symbol, say Z, pushed by $A_{\varphi'}$, it pushes a symbol $Z_\perp$. This new symbol *feels* like Z (in that we allow a test for $Z_\perp$ to succeed whenever a test for Z succeeds) but there are no transitions that pop this symbol. This guarantees that we cannot read a return corresponding to the call at position i using any transition.

If it guesses that a j does exist (and in this case, it must ensure that a return corresponding to the call at position i is encountered along any accepting run in which such a guess is made. We shall return shortly to how this can be arranged.) then, for each $p(f, \psi) \in C'_i$, we must verify that (a) $\pi_{Pr}(\sigma[i+1, j-1]) \models f$ and (b) $\sigma[i+1, j-1] \models \varphi$. The former is dealt with the power of PCPDS to impose constraints. We simply push the constraint f onto the stack and the semantics of PCPDS ensures (a). We may have to push several such f, corresponding to different formulas in C'_i, but then it suffices to push the conjunction of these constraints. For (b), the idea is to start a copy of the automaton A_ψ to read the word until the position j where the matching return is encountered. Notice that $\mathbf{d}(\psi) < \mathbf{d}(\varphi)$ and by the induction hypothesis the existence of A_ψ is guaranteed. Observe that copies are started only at positions i that correspond to outer most calls and the copies terminate when the corresponding call returns. Thus, there are at most $|\Pi|.|AP'|$ such automata under simulation at any point.

We are not done yet. If the guess is that such a j exists, we are also obliged to show that for each $p(f, \psi) \notin C'_i$, either (a') $\pi_{Pr}(\sigma[i+1, j-1]) \not\models f$ or (b') $\sigma[i+1, j-1] \not\models \psi$. Again the automaton guesses which one to verify. To verify , $\pi_{Pr}(\sigma[i+1, j-1]) \not\models f$ observe that this is equivalent to $\pi_{Pr}(\sigma[i+1, j-1]) \models \neg f$. Thus, we simply do what we did in the previous paragraph. It suffices to push $\neg f$ as a constraint and let the semantics of PCPDS take care of the verification of $\pi_{Pr}(\sigma[i+1, j-1]) \models \neg f$. If we guess that $\sigma[i+1, j-1] \not\models \psi$ then we start a copy of the automaton for $A_{\neg\psi}$ (note that $\mathbf{d}(\varphi) > \mathbf{d}(\neg\psi)$), and verify that this automaton is in an accepting state when position j is reached.

Thus, the only thing left to explain is how we validate a guess that a j with $i \curvearrowright j$ exists. The visibility restriction prevents popping of the stack at the end to verify that there are no pending returns (also such a technique will not work in the construction of B_φ to deal with infinite words). The point is that, the number of constraints on the stack at any point during the run, due to the construction described here (and not counting those due to the automata A_ψ being simulated) is bounded by Π, one per outer most call that is currently

active. Let us call this height *outer constraint height*. Thus, we can keep track of the outer constraint height of the stack as part of the control state. Then, at step i, to ensure that the call at $\sigma(i)$ returns, we simply record the outer constraint height as a *target* in the control state. Whenever the current level falls to a target level we drop that from the target set. An accepting state verifies in addition that there are no pending targets to be reached. Actually it suffices to maintain the lowest target at any point and discharge it when it is visited.

In summary, we construct a PCPDS whose state has several components: a global component that tracks the state of $A_{\varphi'}$, records information to recover outer most calls, tracks the current outer constraint height and tracks the current target for the outer constraint height. It has also has one component for each pair $P \in \Pi$ and $W_f(\psi) \in \mathcal{W}$. This component maintains the state of the automaton A_ψ if a copy of this automaton has been started at the currently active outermost call to P, or else its value is $\perp$. Such a component gets reset to $\perp$ whenever the outer most call of P returns (after verifying that it had reached the accepting state). Finally, the accepting states verify that the simulation of $A_{\varphi'}$ is accepting, no target levels are pending and that all the additional components are in the $\perp$ state.

The changes needed to handle the Büchi automata construction in the case of B_φ are minor. The simulations still use A_ψ (since the calls are obliged to terminate at some j). The only issue is with tracking visits to accepting states of $A_{\varphi'}$ while ensuring that target levels are reached. This can be ensured as follows: we do not indicate visits to accepting states of $A_{\varphi'}$ when some target is pending. We simply record it in the local state and whenever we find that all targets have been attained we flag any visits to the accepting state in the intervening run. Since the setting and unsetting of target levels happens in a well-nested manner, we are guaranteed to indicate visits to accepting configurations infinitely often as long as the run met all its obligations (i.e. contains all the necessary returns) and visits accepting states infinitely often. This gives us the following theorem.

▶ **Theorem 15.** *For any QCaReT formula φ, we can construct a PCPDS A_φ and Büchi PCPDS B_φ such that $\mathcal{L}(\varphi) = \mathcal{L}(A_\varphi)$ and $\mathcal{L}^\omega(\varphi) = \mathcal{L}(B_\varphi)$. The resulting automata have $\mathcal{O}((2^{|\varphi|} \times |\Pi|^2)^{(|\Pi|.|\varphi|)^{\mathcal{O}(|\varphi|)}})$ states and they are $\mathcal{O}((|\Pi|.|\varphi|)^{\mathcal{O}(|\varphi|)})$ constraint height bounded.*

Closure under intersections and emptiness checking (via reachability/repeated reachability) of PCPDSs means that we may also model check VPAs (as well as constraint height bounded PCPDAs) against QCaReT specifications.

9 Conclusion

In this work, we provide a method to specify and verify the quantitative properties of procedural programs. For this purpose, we introduced an automaton model called the constrained pushdown system (CPDS). We showed that reachability on such systems in general is undecidable. We then showed that reachability and repeated reachability are decidable in 2-EXPTIME when the number of constraints in the stack remains bounded.

We also introduced the high level specification language called the QCaReT and an extension of visibly pushdown system called the procedural CPDS (PCPDS). Finally we provided an algorithm for satisfiability and model-checking QCaReT formulas against PCPDS (and hence a VPA) by a reduction to reachability/ repeated reachability on a CPDS.

One question that is left unanswered is whether the decision procedure for decidability of reachability in CPDS is optimal. While we provide a 2-EXPTIME procedure, we only have a PSPACE lower bound. As a future work, the language theoretic properties of the constraint height bounded CPDS is an interesting topic that can be explored.

References

1. P. A. Abdulla, M. F. Atig, and J. Stenman. The minimal cost reachability problem in priced timed pushdown systems. In *LATA*, volume 7183 of *LNCS*, 2012.
2. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Giorgio Delzanno, and Andreas Podelski. Push-down automata with gap-order constraints. In *Fundamentals of Software Engineering - 5th International Conference, FSEN 2013, Tehran, Iran, April 24-26, 2013, Revised Selected Papers*, volume 8161 of *Lecture Notes in Computer Science*. Springer, 2013.
3. Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jari Stenman. Dense-timed pushdown automata. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 35–44. IEEE Computer Society, 2012.
4. S. Akshay, Paul Gastin, and Shankara Narayanan Krishna. Analyzing Timed Systems Using Tree Automata. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory (CONCUR 2016)*, volume 59 of *Leibniz International Proceedings in Informatics (LIPIcs)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.
5. Rajeev Alur, Marcelo Arenas, Pablo Barceló, Kousha Etessami, Neil Immerman, and Leonid Libkin. First-order and temporal logics for nested words. *Logical Methods in Computer Science*, 4(4), 2008. doi:10.2168/LMCS-4(4:11)2008.
6. Rajeev Alur, Kousha Etessami, and P. Madhusudan. A temporal logic of nested calls and returns. In *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, 2004. doi:10.1007/978-3-540-24730-2_35.
7. Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, 2004. doi:10.1145/1007352.1007390.
8. Mohamed Faouzi Atig, Dmitry Chistikov, Piotr Hofman, K. Narayan Kumar, Prakash Saivasan, and Georg Zetzsche. The complexity of regular abstractions of one-counter languages. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8*, 2016. doi:10.1145/2933575.2934561.
9. Benedikt Bollig, Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. Model checking languages of data words. In *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7213 of *Lecture Notes in Computer Science*. Springer, 2012.
10. A. Bouajjani, R. Echahed, and R. Robbana. On the automatic verification of systems with continuous variables and unbounded discrete data structures. In *Hybrid Systems II*, volume 999 of *LNCS*. Springer, 1994.
11. Ahmed Bouajjani and Peter Habermehl. Symbolic reachability analysis of fifo-channel systems with nonregular sets of configurations. *Theor. Comput. Sci.*, 221(1-2):211–250, 1999.
12. Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. On the expressiveness of parikh automata and related models. In *Third Workshop on Non-Classical Models for Automata and Applications - NCMA 2011, Milan, Italy, July 18 - July 19, 2011. Proceedings*, 2011.
13. Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Bounded parikh automata. *Int. J. Found. Comput. Sci.*, 23(8):1691–1710, 2012.
14. X. Cai and M. Ogawa. Well-structured pushdown systems. In *CONCUR 2013*, volume 8052 of *LNCS*. Springer, 2013.

15 Krishnendu Chatterjee, Andreas Pavlogiannis, and Yaron Velner. Quantitative interprocedural analysis. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17*, 2015.

16 Lorenzo Clemente and Slawomir Lasota. Timed pushdown automata revisited. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*. IEEE Computer Society, 2015.

17 F. S. de Boer, M. M. Bonsangue, and J. Rot. It is pointless to point in bounded heaps. *Sci. Comput. Program.*, 112, 2015.

18 Javier Esparza, Pierre Ganty, Stefan Kiefer, and Michael Luttenberger. Parikh's theorem: A simple and direct automaton construction. *Inf. Process. Lett.*, 111(12), 2011. doi:10.1016/j.ipl.2011.03.019.

19 Oscar H. Ibarra. Visibly pushdown automata and transducers with counters. *Fundam. Inform.*, 148(3-4):291–308, 2016.

20 Felix Klaedtke and Harald Rueß. Monadic second-order logics with cardinalities. In JosC.M. Baeten, JanKarel Lenstra, Joachim Parrow, and GerhardJ. Woeginger, editors, *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003.

21 Christof Löding and Karianto Wong. On nondeterministic unranked tree automata with sibling constraints. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, 2009. doi:10.4230/LIPIcs.FSTTCS.2009.2328.

22 Helmut Seidl, Thomas Schwentick, and Anca Muscholl. Counting in trees. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*., 2008.

23 Helmut Seidl, Thomas Schwentick, Anca Muscholl, and Peter Habermehl. Counting in trees for free. In *Automata, Languages and Programming: 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004. Proceedings*, volume 3142 of *Lecture Notes in Computer Science*. Springer, 2004.