Noname manuscript No.
(will be inserted by the editor)

Bounded Underapproximations

Pierre Ganty · Rupak Majumdar · Benjamin Monmege

the date of receipt and acceptance should be inserted later

Abstract We show a new and constructive proof of the following language-theoretic result: for every context-free language L, there is a *bounded* context-free language $L' \subseteq L$ which has the same Parikh (commutative) image as L. Bounded languages, introduced by Ginsburg and Spanier, are subsets of regular languages of the form $w_1^* w_2^* \cdots w_m^*$ for some $w_1, \ldots, w_m \in \Sigma^*$. In particular bounded context-free languages have nice structural and decidability properties. Our proof proceeds in two parts. First, we give a new construction that shows that each context free language L has a subset L_N that has the same Parikh image as L and that can be represented as a sequence of substitutions on a linear language. Second, we inductively construct a Parikh-equivalent bounded context-free subset of L_N.

We show two applications of this result in model checking: to underapproximate the reachable state space of multithreaded procedural programs and to underapproximate the reachable state space of recursive counter programs. The bounded language constructed above provides a decidable underapproximation for the original problems. By iterating the construction, we get a semi-algorithm for the original problems that constructs a sequence of underapproximations such that no two underapproximations of the sequence can be compared. This provides a progress guarantee: every word $w \in L$ is in some underapproximation of the sequence, and hence, a program bug is guaranteed to be found. In particular, we show that verification with bounded languages generalizes context-bounded reachability for multithreaded programs.

This research was sponsored in part by the NSF grants CCF-0546170 and CCF-0702743, DARPA grant HR0011-09-1-0037, Comunidad de Madrid's Program PROMETIDOS-CM (S2009TIC-1465), PEOPLE-COFUND's program AMAROUT (PCOFUND-2008-229599), and by the Spanish Ministry of Science and Innovation (TIN2010-20639).

Pierre Ganty
IMDEA Software, Spain E-mail: pierre.ganty@imdea.org

Rupak Majumdar
MPI-SWS, Germany and UC Los Angeles, USA E-mail: rupak@mpi-sws.org

Benjamin Monmege
LSV, ENS Cachan, CNRS & INRIA, France E-mail: monmege@lsv.ens-cachan.fr

Keywords Context-free grammar · Bounded languages · Parikh-boundedness · Multithreaded reachability · Recursive counter machines

1 Introduction

Many problems in program analysis reduce to undecidable problems about context-free languages. For example, checking safety properties of multithreaded recursive programs reduces to checking emptiness of the intersection of context-free languages [19,3]. Checking reachability for recursive counter programs relies on context-free languages to describe valid control flow paths.

We study underapproximations of these problems, with the intent of building tools to find bugs in systems. In particular, we study underapproximations in which one or more context-free languages arising in the analysis are replaced by their subsets in a way that (P1) the resulting problem after the replacement becomes decidable and (P2) the subset preserves "many" strings from the original language. Condition (P1) ensures that we have an algorithmic check for the underapproximation. Condition (P2) ensures that we are likely to retain behaviors that would cause a bug in the original analysis.

We show in this paper an underapproximation scheme using *bounded languages* [13,12]. A language L is *bounded* if there exist $k \in \mathbb{N}$ and finite words $w_1, w_2, \ldots, w_m$ such that L is a subset of the regular language $w_1^* \cdots w_m^*$. In particular, context-free bounded languages (hereunder bounded languages for short) have stronger properties than general context-free languages: for example, it is decidable to check if the intersection of a context-free language and a bounded language is non-empty [13]. For our application to verification, these decidability results ensure condition (P1) above.

The key to condition (P2) is the following *Parikh-boundedness* property: for every context-free language L, there is a bounded language $L' \subseteq L$ such that the Parikh images of L and L' coincide. (The *Parikh image* of a word w maps each letter of the alphabet to the number of times it appears in w, the Parikh image of a language is the set of Parikh images of all words in the language.) A language L' meeting the above conditions is called a *Parikh-equivalent bounded subset* of L. Intuitively, L' preserves "many" behaviors as for every string in L, there is a permutation of its letters that matches a string in L'.

The Parikh-boundedness property was first proved in [16,2], however, the chain of reasoning used in these papers made it difficult to see how to explicitly construct the Parikh-equivalent bounded subset. Our paper gives a direct and constructive proof of the theorem. We identify three contributions in this paper.

Explicit construction of Parikh-equivalent bounded subsets. Our constructive proof has two parts. First, given a context-free language L, we show how to compute a subset of L that has the same Parikh image and that can be represented as a finite sequence of substitutions on a linear grammar. (A *linear* grammar is a context-free grammar where each rule has at most one non-terminal on the right-hand side.)

Second, we provide a direct constructive proof that takes as input such a sequence of linear substitutions, and constructs by induction a Parikh-equivalent bounded subset of the language denoted by the sequence. Our constructions are

optimal, and construct a bounded expression which is exponential in the size of the original grammar. We show that the exponential is necessary.

Along the way, for regular languages, we show an analogous Parikh-boundedness property (where the bounded language is also regular), and give a construction that is exponential in the size of the alphabet but polynomial in the size of the regular expression. Again, we show that the construction is optimal by showing an exponential lower bound in the size of the alphabet.

Reachability analysis of multithreaded programs with procedures. Using the above construction, we obtain a semi-algorithm for reachability analysis of multithreaded programs with the intent of finding bugs. To check if configuration (c_1, c_2) of a recursive 2-threaded program is reachable, we construct the context-free languages $L_1^0 = L(c_1)$ and $L_2^0 = L(c_2)$ respectively given by the execution paths whose last configurations are c_1 and c_2, and check if either $L_1' \cap L_2^0$ or $L_1^0 \cap L_2'$ is non-empty, where $L_1' = L_1^0 \cap w_1^* \cdots w_m^*$ and $L_2' = L_2^0 \cap v_1^* \cdots v_l^*$ are two Parikh-equivalent bounded subsets of L_1^0 and L_2^0, respectively. If either intersection is non-empty, we have found a witness trace. Otherwise, we construct $L_1^1 = L_1^0 \cap \overline{w_1^* \cdots w_m^*}$ and $L_2^1 = L_2^0 \cap \overline{v_1^* \cdots v_l^*}$ in order to exclude, from the subsequent analyses, the execution paths we already inspected. We continue by rerunning the above analysis on L_1^1 and L_2^1. If (c_1, c_2) is reachable, the iteration is guaranteed to terminate; if not, it could potentially run forever. Moreover, we show our technique subsumes and generalizes context-bounded reachability [18].

Reachability analysis of programs with counters and procedures. We also show how to underapproximate the set of reachable states of a procedural program that manipulates a finite set of counters. This program is given as a counter automaton A (see [17] for a detailed definition) together with a context-free language L over the transitions of A. Our goal is to compute the states of A that are reachable using a sequence of transitions in L.

A possibly non terminating algorithm to compute the reachable states of A through executions in L is to (1) find a Parikh-equivalent bounded subset L' of L; (2) compute the states that are reachable using a sequence of transitions in L' (as explained in [17], this set is computable if (i) some restrictions on the transitions of A ensures the set is Presburger definable and (ii) L' is bounded, i.e. $L' \subseteq w_1^* \cdots w_m^*$); and (3) rerun the analysis using for $L \cap \overline{w_1^* \cdots w_m^*}$ so that runs already inspected are omitted in every subsequent analyses. Again, every path in L is eventually covered in the iteration.

Related Work. Bounded languages were introduced and studied by Ginsburg and Spanier [13] (see also [12]). The existence of a bounded, Parikh-equivalent subset for a context-free language was shown in [2] using previous results on languages in the Greibach hierarchy [16]. In an earlier version of our proof [11], we showed the existence of a language representable as a sequence of linear transformations of a linear language which is Parikh-equivalent to a context-free language using *Newton's iterations* [9] on the language and Parikh semirings. (Such a construction was independently done in [8].) In this paper, we give a new, and greatly simplified, construction using some recent observations by [7,9].

Bounded languages have been proposed by Kahlon for tractable reachability analysis of multithreaded programs [15]. His observation is that in many practical instances of multithreaded reachability, the languages are actually bounded. If this is true, his algorithm checks the emptiness of the intersection (using the algorithm

in [13]). In contrast, our results are applicable even if the boundedness property does not hold. Recently, [6] introduced a technique for verification of multithreaded programs based on patterns which are expressions of the form $w_1^* \cdots w_m^*$.

For multithreaded reachability, *context-bounded reachability* [18,20] is a popular underapproximation technique which tackles the undecidability by limiting the search to those runs where the active thread changes at most k times. Our algorithm using bounded languages *subsumes* context-bounded reachability, and can capture unboundedly many synchronizations in one analysis.

For underapproximating reachable states of a counter machine under context-free restrictions, we are not aware of any other work trying to solve this problem. Notice that the problem can be analysed with existing techniques by encoding the stack using counters (after all, counter machines are Turing-powerful). However we believe that keeping the natural structure of context-free languages and approximating it through bounded languages allows us to compute reachable configurations which cannot be computed using existing techniques. This is because bounded languages allows to isolate the control flow from the data in programs.

2 Preliminaries

We assume the reader is familiar with the basics of language theory (see, e.g., [14]). An alphabet Σ is a finite non-empty set of letters. The concatenation $L \odot L'$ of two languages $L, L' \subseteq \Sigma^*$ is defined using word concatenation as $L \odot L' = \{l \cdot l' \mid l \in L \wedge l' \in L'\}$. For the sake of clarity we sometimes abbreviate $w \cdot w'$ and $L \odot L'$ as ww' and LL', respectively. A *bounded expression* over Σ is a language of the form $w_1^* \cdots w_m^*$ for some fixed $w_1, \ldots, w_m \in \Sigma^*$. The size of a bounded expression $w_1^* \ldots w_m^*$ is defined as $\sum_{i=1}^m |w_i|$.

Vectors. For $p \in \mathbb{N}$, we write $\mathbb{Z}^p$ and $\mathbb{N}^p$ for the set of p-dim vectors (or simply vectors) of integers and naturals, respectively. We write $\mathbf{0}$ for the vector $(0, \ldots, 0)$ and $\mathbf{e}_i$ the vector $(z_1, \ldots, z_p) \in \mathbb{N}^p$ such that $z_j = 1$ if $j = i$ and $z_j = 0$ otherwise. *Addition* on p-dim vectors is the componentwise extension of its scalar counterpart, that is, given $(x_1, \ldots, x_p), (y_1, \ldots, y_p) \in \mathbb{Z}^p$ $(x_1, \ldots, x_p) + (y_1, \ldots, y_p) = (x_1 + y_1, \ldots, x_p + y_p)$. Given $\lambda \in \mathbb{N}$ and $x \in \mathbb{Z}^p$, we write λx as the λ-times sum $x + \cdots + x$. Using vector addition, we define the operation $\dot{+}$ on sets of vectors as follows: given $Z, Z' \subseteq \mathbb{N}^p$, let $Z \dot{+} Z' = \{z + z' \mid z \in Z \wedge z' \in Z'\}$.

Parikh Image. Give Σ a fixed linear order: $\Sigma = \{a_1, \ldots, a_p\}$. The Parikh image of a letter $a_i \in \Sigma$, written $\Pi_\Sigma(a_i)$, is $\mathbf{e}_i$. The Parikh image is extended to words of Σ^* as follows: $\Pi_\Sigma(\varepsilon) = \mathbf{0}$ and $\Pi_\Sigma(u \cdot v) = \Pi_\Sigma(u) + \Pi_\Sigma(v)$. Finally, the Parikh image of a language on Σ^* is the set of Parikh images of its words. Thus, the Parikh image maps 2^{Σ^*} to $2^{\mathbb{N}^p}$. When it is clear from the context we generally omit the subscript in Π_Σ. Two languages L_1 and L_2 are *Parikh equivalent* if $\Pi(L_1) = \Pi(L_2)$.

The following lemma establishes a simple property Π we need in the sequel.

Lemma 1 (preservation of Π) *Let $X, Y \subseteq \Sigma^*$ we have $\Pi(X \odot Y) = \Pi(X) \dot{+} \Pi(Y)$.*

Proof

$$\begin{aligned}
\Pi(X \odot Y) &= \{\Pi(w) \mid w \in X \odot Y\} && \text{def. of } \Pi \\
&= \{\Pi(x \cdot y) \mid x \in X \wedge y \in Y\} && \text{def. of } \odot \\
&= \{\Pi(x) + \Pi(y) \mid x \in X \wedge y \in Y\} && \text{def. of } \Pi \\
&= \{a + b \mid a \in \Pi(X) \wedge b \in \Pi(y)\} \\
&= \Pi(X) \dot{+} \Pi(Y) && \text{def. of } \dot{+}
\end{aligned}$$

□

Context-free Languages. A *context-free grammar* (CFG) G is a tuple $(\mathcal{X}, \Sigma, \mathcal{P})$ where $\mathcal{X}$ is a finite non-empty set of *variables* (non-terminal letters), Σ is an alphabet of terminal *letters* and $\mathcal{P} \subseteq \mathcal{X} \times (\Sigma \cup \mathcal{X})^*$ a finite set of productions (the production (X, w) may also be noted $X \to w$). Given two strings $u, v \in (\Sigma \cup \mathcal{X})^*$ we define the *derivation* relation $u \Rightarrow v$, if there exists a production $(X, w) \in \mathcal{P}$ and some words $y, z \in (\Sigma \cup \mathcal{X})^*$ such that $u = yXz$ and $v = ywz$. A *partial derivation* is a finite sequence $u_1 \Rightarrow u_2 \Rightarrow u_l$ for some l and strings $u_1, \ldots, u_l \in (\Sigma \cup \mathcal{X})^*$. We use $\Rightarrow^*$ for the reflexive transitive closure of $\Rightarrow$. A word $w \in \Sigma^*$ is recognized by the grammar G from the state $X \in \mathcal{X}$ if $X \Rightarrow^* w$. Given $X \in \mathcal{X}$, the language $L_X(G)$ is given by $\{w \in \Sigma^* \mid X \Rightarrow^* w\}$. A language L is *context-free* (CFL) if there exists a context-free grammar $G = (\mathcal{X}, \Sigma, \mathcal{P})$ and an *initial variable* $X \in \mathcal{X}$ such that is $L = L_X(G)$. A *linear* (resp. *regular*) grammar G is a CFG where each production is in $\mathcal{X} \times \Sigma^*(\mathcal{X} \cup \{\varepsilon\})\Sigma^*$ (resp. $\mathcal{X} \times \Sigma^*(\mathcal{X} \cup \{\varepsilon\})$). A language L is *linear* (resp. *regular*) if $L = L_X(G)$ for some linear (resp. regular) grammar G and initial variable X of G. In what follows we usually write regular grammar as follows $R = (Q, \Sigma, \delta)$. A CFL L is *bounded* if it is a subset of some bounded expression.

Proof Plan. The main result of the paper is the following.

Theorem 1 *For every* CFL L*, there is an effectively computable* CFL L' *such that* (i) $L' \subseteq L$*,* (ii) $\Pi(L) = \Pi(L')$*, and* (iii) L' *is bounded.*

We actually solve the following related problem in our proof.

Problem 1 Given a language L, compute a bounded expression B such that

$$\Pi(L \cap B) = \Pi(L) \ .$$

In this paper we study the particular problem of solving Pb. 1 when the language L is a CFL. If we can compute such a bounded expression B, then we can compute the CFL $L' = B \cap L$ which satisfies conditions (i) to (iii) of the Th. 1. Thus, solving Pb. 1 proves the theorem constructively.

We solve Pb. 1 for a CFL L as follows: (1) we find a CFL L' such that $L' \subseteq L$, $\Pi(L') = \Pi(L)$, and L' has a "simple" structure (Sect. 3) and (2) then we show how to compute a bounded expression B which is a solution to Pb. 1 for instance L'. Observe that because $L' \subseteq L$ and $\Pi(L) = \Pi(L')$, we find that if B is a solution to Pb. 1 for instance L', then so is B for instance L. Along Sect. 4, we also give some bounds on the size the smallest bounded expression which solves Pb. 1 for

various classes of instances like regular, linear and context-free languages. Finally, Sect. 5 provides applications of the result for program analysis problems.

We conclude this section to point out that the existence of a solution to Pb. 1 is not a trivial property of every class of languages. In fact, the existence of a solution to Pb. 1 for context-sensitive languages is not guaranteed as shown by the following example taken from [2]. Consider Pb. 1 for the language

$$L_1 = \{1010^2 \dots 10^h \mid h \geq 1\}$$

there is no bounded expression which solves Pb. 1, namely $\Pi(L_1 \cap B) \neq \Pi(L_1)$ for every bounded expression B. Observe that $\Pi(L_1)$ is not semilinear.

Moreover, consider $L_2 = L_1 \cup \{1^i 0^j \mid (j,i) \notin \Pi(L_1)\}$[1]. It follows from [2] that Pb. 1 for instance L_2 has no solution. It is worth pointing that $\Pi(L_2) = \mathbb{N}^2$ is semilinear, but $L_2 \cap 1^*0^*$ is not.

3 A Parikh-Equivalent Representation

This section is devoted to show that given a CFL L, we can compute a language L' such that $L' \subseteq L$, $\Pi(L') = \Pi(L)$, and L' has a "simple" representation.

3.1 Derivation tree, yield and dimension

In this section, we follow ideas from [9] and give tree versions of the semantics of context-free grammars. Let $G = (\mathcal{X}, \Sigma, \mathcal{P})$ be a fixed CFG. In the following, we will define some ordered unbounded trees with nodes labeled by production rules $(X, \alpha) \in \mathcal{P}$. For a given tree t, define $\lambda_v(t) = X$ if the root of t is labeled by $(X, \alpha) \in \mathcal{P}$.

Definition 1 (Derivation tree, yield) The *derivation trees* of G and their *yields* are inductively defined as follows:

- For every production $(X, \alpha) \in \mathcal{P} \cap (\mathcal{X} \times \Sigma^*)$, the tree t consisting of one single node labeled by (X, α) is a derivation tree of G. Its yield $\mathcal{Y}(t)$ is equal to α.
- For every production $(X, \alpha) \in \mathcal{P}$ such that $\alpha = a_1 X_1 a_2 X_2 \cdots a_k X_k a_{k+1}$ for some $k \geq 1$, let $t_1, \dots, t_k$ be derivation trees of G such that for every $1 \leq i \leq k$, $\lambda_v(t_1) = X_i$. Then the tree t with root labelled by (X, α) and having $t_1, \dots, t_k$ as (ordered) children is also a derivation tree of G, and its yield $\mathcal{Y}(t)$ is equal to $a_1 \mathcal{Y}(t_1) \cdots a_k \mathcal{Y}(t_k) a_{k+1}$.

The yield $\mathcal{Y}(T)$ of a set T of derivation trees is defined by $\mathcal{Y}(T) = \bigcup_{t \in T} \mathcal{Y}(t)$. We denote by $\mathcal{T}(G)$ the set of derivation trees of G, simply writing $\mathcal{T}$ if G is clear from the context. Moreover we define $\mathcal{T}_X(G)$ to be the subset of all derivation trees $t \in \mathcal{T}(G)$ such that $\lambda_v(t) = X$. In what follows, we often abbreviate *derivation tree* to *tree*.

[1] $\Sigma = \{0, 1\}$ is ordered as $0 \prec 1$

Observe first that, since every word in $L_X(G)$ is the yield of some derivation tree, we have $L_X(G) = \mathcal{Y}(\mathcal{T}_X)$.

We want to find a Parikh-equivalent sublanguage of $L_X(G)$ for a fixed $X \in \mathcal{X}$. This will be achieved by showing that a subset of $\mathcal{T}_X$ is a Parikh-equivalent to $L_X(G)$. This subset of $\mathcal{T}_X$ is characterized using the notion of *dimension* of a derivation tree.

Definition 2 (Dimension) The dimension $d(t)$ of a tree t is inductively defined as follows:

1. If t is a node with no children, then $d(t) = 0$.
2. If t has exactly one child t_1, then $d(t) = d(t_1)$.
3. If t has at least two children, let t_1, t_2 be two distinct children of t such that $d(t_1) \geq d(t_2)$ and $d(t_2) \geq d(t')$ for every child $t' \neq t_1$. Then

$$d(t) = \begin{cases} d(t_1) + 1 & \text{if } d(t_1) = d(t_2) \\ d(t_1) & \text{if } d(t_1) > d(t_2) \end{cases}$$

We denote by $\mathcal{D}^i(G)$ (resp. $\mathcal{D}^i_X(G)$) the set of derivation trees of dimension at most i (resp. and with λ_v-labels X). We omit the argument G if it is clear from the context.

The next lemma states that every tree is Parikh-equivalent to a tree of at most dimension n where $n = |\mathcal{X}|$ is the number of variables in G. This result has been proved in [9,7] with slightly different notations and in a different context so we do not reformulate the proof here.

Lemma 2 *Let G be a* CFG *with n variables. For each tree $t \in \mathcal{T}$, there is a tree $t' \in \mathcal{D}^n$ such that t and t' have the same number of nodes, the set of λ_v-labels in for all subtrees in t and t' coincide and $\Pi(\mathcal{Y}(t)) = \Pi(\mathcal{Y}(t'))$.*

We shall use the following simple corollary of this result.

Corollary 1 *Let $G = (\mathcal{X}, \Sigma, \mathcal{P})$ be a context-free grammar and $n = |\mathcal{X}|$. For every variable $X \in \mathcal{X}$, the language $\mathcal{Y}(\mathcal{D}^n_X)$ is a subset of $L_X(G)$ and is Parikh-equivalent to $L_X(G)$.*

Given n and a context-free grammar $G = (\mathcal{X}, \Sigma, \mathcal{P})$, we define the context-free grammar $G^{[n]} = (\mathcal{X}^{[n]}, \Sigma, \mathcal{P}^{[n]})$ which annotates the variables of $\mathcal{X}$ with a positive integer superscript bounding the dimension of the underlying derivation tree. We will then show that for every $X \in \mathcal{X}$ we have $L_{X^{[n]}}(G^{[n]}) = \mathcal{Y}(\mathcal{D}^n_X)$.

Our definition is inspired by an example appearing in [9].

Definition 3 Let $G = (\mathcal{X}, \Sigma, \mathcal{P})$ be a CFG and $n \in \mathbb{N}$. Define the CFG $G^{[n]} = (\mathcal{X}^{[n]}, \Sigma, \mathcal{P}^{[n]})$ as follows. Define $\mathcal{X}^{[n]} = \{X^{[i]} \mid 0 \leq i \leq n \wedge X \in \mathcal{X}\}$. Define $\mathcal{P}^{[n]}$ as the smallest set such that $(X^{[i]}, \alpha) \in \mathcal{P}^{[n]}$ if $(X, \mathit{erase}(\alpha)) \in \mathcal{P}$ and either $\alpha \in \Sigma^*$ or $\exists \alpha_1 \in (\mathcal{X} \cup \Sigma)^*, \alpha_2 \in (\mathcal{X} \cup \Sigma)^*, Y \in \mathcal{X}$: $\alpha = \alpha_1 Y^{[i]} \alpha_2$ and $\mathit{Sidx}(\alpha_1) \cup \mathit{Sidx}(\alpha_2) \subseteq \{i-1\}$ where $\mathit{erase}(\alpha)$ removes the superscript from every variable occurrence in α and $\mathit{Sidx}(\alpha)$ returns the set of superscripts that occurs in α.

Example 1 Let us define the CFG G and $G^{[1]}$ thereof which are given by:

$$\begin{array}{lll} X \to AX' \mid \$ & X^{[1]} \to A^{[0]}X'^{[1]} \mid A^{[1]}X'^{[0]} \mid \$ & A^{[1]} \to a \\ X' \to XB & X^{[0]} \to \$ & A^{[0]} \to a \\ A \to a & X'^{[1]} \to X^{[1]}B^{[0]} \mid X^{[0]}B^{[1]} & B^{[1]} \to b \\ B \to b & & B^{[0]} \to b \end{array}$$

Note that there is no rule with left hand side having the decorated variable $X'^{[0]}$ by definition.

Lemma 3 *Let $G = (\mathcal{X}, \Sigma, \mathcal{P})$ be a* CFG*, $X \in \mathcal{X}$ and $n \in \mathbb{N}$. Let $G^{[n]}$ be given as in def. 3, we have $L_{X^{[n]}}(G^{[n]}) = \mathcal{Y}(\mathcal{D}^n_X)$.*

Proof We extend the substitution *erase* to derivation trees by applying it recursively over every label $(X^{[i]}, \alpha)$, mapping it to $(\mathit{erase}(X^{[i]}) = X, \mathit{erase}(\alpha))$. Then, we show by induction on $h \in \mathbb{N}$ that for every derivation tree t of G of height at most h we have for every $n \leq h$ (the dimension of a tree is always less or equal than its height):

$$t \in \mathcal{D}^n_X(G) \quad \iff \quad t \in \mathit{erase}(\mathcal{T}_{X^{[n]}}(G^{[n]}))$$

For the base case, let $h = 0$. Let $t \in \mathcal{D}^0_X(G)$ be a tree of height 0. This is a childless root labelled by $(X, \alpha) \in \mathcal{P}$ with $\alpha \in \Sigma^*$. By definition of $G^{[0]}$ we find that $(X^{[0]}, \mathit{erase}(\alpha) = \alpha) \in \mathcal{P}^{[0]}$. Hence, the tree t' consisting of the childless root labelled $(X^{[0]}, \alpha)$ belongs to $\mathcal{T}_{X^{[0]}}(G^{[0]})$ and $t = \mathit{erase}(t')$.

We now suppose that $h > 0$ and that the result holds for every tree of height at most $h - 1$. Let $t \in \mathcal{D}^n_X(G)$ with height h. By definition of the dimension, we have three cases.

- If t has no child, then its height is 0, and we can conclude by the base case.
- If t has exactly one child t_1, then $d(t_1) = d(t)$. Let Y be such that $Y = \lambda_v(t_1)$. Clearly the height of t_1 is less than the one of t and so, by induction hypothesis, there exists $t'_1 \in \mathcal{T}_{Y^{[n]}}(G^{[n]})$ with $t_1 = \mathit{erase}(t'_1)$. Moreover, if (X, α) is the root of t, we necessarily can decompose α as $\alpha_1 Y \alpha_2$, with $\alpha_1, \alpha_2 \in \Sigma^*$. Then the tree t' rooted by $(X^{[n]}, \alpha_1 Y^{[n]} \alpha_2) \in \mathcal{P}^{[n]}$ with only child t'_1 is clearly in $\mathcal{T}_{X^{[n]}}(G^{[n]})$ and it further verifies $t = \mathit{erase}(t')$.
- If t has at least two children, let $(X, a_1 Y_1 a_2 \cdots a_k Y_k a_{k+1})$ be the label of the root of t. The definition of dimension shows that there exists two distinct subtrees t_i, t_j of t such that $d(t_i) \geq d(t_j)$ and $d(t_j) \geq d(t')$ for every child $t' \neq t_i$. Moreover, we have:

$$d(t) = \begin{cases} d(t_i) + 1 & \text{if } d(t_i) = d(t_j) \\ d(t_i) & \text{if } d(t_i) > d(t_j) \end{cases}$$

  Every child t_ℓ has height less or equal than $h - 1$, so we conclude by induction hypothesis that for $\ell \neq i$ there exists a derivation tree $t'_\ell \in \mathcal{T}_{Y_\ell^{[n-1]}}(G^{[n]})$ (as $G^{[n-1]}$ is included in $G^{[n]}$) such that $t_\ell = \mathit{erase}(t'_\ell)$, and that there exists a derivation tree $t'_i \in \mathcal{T}_{Y_\ell^{[n]}}(G^{[n]})$ such that $t_i = \mathit{erase}(t'_i)$. Then the tree t' with root labelled $(X^{[n]}, a_1 Y_1^{[n-1]} a_2 \cdots Y_{i-1}^{[n-1]} a_i Y^{[n]} a_{i+1} Y_{i+1}^{[n-1]} \cdots a_k Y_k^{[n-1]} a_{k+1}) \in \mathcal{P}^{[n]}$ and with children $t'_1, \ldots, t'_k \in \mathcal{T}(G^{[n]})$ is a derivation tree of $G^{[n]}$ with $t = \mathit{erase}(t')$.

The reverse implication is proved similarly. □

Lemma 3 and Cor. 1: let L be a CFL given by $L_X(G)$ where $G = (\mathcal{X}, \Sigma, \mathcal{P})$ is a CFG with n variables and $X \in \mathcal{X}$, if B be solution to Pb. 1 for the instance $L_{X^{[n]}}(G^{[n]})$ then so is B for the instance L.

In order to compute the bounded expression B solving Pb. 1, in the following section, we give an equivalent representation of $L_{X^{[n]}}(G^{[n]})$. Roughly, the idea is to give a representation in $n+1$ layers, such that layer i will simulate the derivation steps of $G^{[n]}$ where only non-terminals with superscript i are expanded. Formally each layer will correspond to a linear context-free grammar. In order to simulate a derivation of $L_{X^{[n]}}(G^{[n]})$, we need the layers to interact with each other. Interactions will be formally defined using substitutions.

3.2 t-fold composition

A *substitution* σ from alphabet Σ_1 to alphabet Σ_2 is a function which maps every word over Σ_1 to a set of words of Σ_2^* such that $\sigma(\varepsilon) = \{\varepsilon\}$ and $\sigma(u \cdot v) = \sigma(u) \cdot \sigma(v)$. A *homomorphism* h is a substitution such that for each word u, $h(u)$ is a singleton. We write $\sigma_{[a/b]} \colon \Sigma_1 \cup \{a\} \to \Sigma_1 \cup \{b\}$ for the substitution which maps a to b and leaves all other letters unchanged.

Given $\mathcal{X}$ and $i \geq 0$, define the set $v_{\mathcal{X}^{(i)}} = \{v_{X^{(i)}} \mid X \in \mathcal{X}\}$ of letters. Moreover, define the substitution τ_i which replaces each $X^{[i]}$ by $v_{X^{(i)}}$. Finally, define τ_{-1} as the identity.

Definition 4 Given the CFG $G^{[n]} = (\mathcal{X}^{[n]}, \Sigma, \mathcal{P}^{[n]})$ define the family $\{G^{(0)}, \ldots, G^{(n)}\}$ of CFGs as follows. For each $i \in \{0, \ldots, n\}$, define $G^{(i)} = (\mathcal{X}^{(i)}, \Sigma^{(i)}, \mathcal{P}^{(i)})$ where $\mathcal{X}^{(i)} = \{X^{(i)} \mid X^{[i]} \in \mathcal{X}^{[n]}\}$,

$$\Sigma^{(i)} = \Sigma \cup \begin{cases} v_{\mathcal{X}^{(i-1)}} & \text{if } i > 0 \\ \varnothing & \text{otherwise} \end{cases}$$

and $\mathcal{P}^{(i)}$ is the smallest set given by:

- if $(A^{[i]} \to \alpha) \in \mathcal{P}^{[n]}$ where $\alpha \in \Sigma^*$ then $(A^{(i)} \to \alpha) \in \mathcal{P}^{(i)}$;
- if $(A^{[i]} \to \alpha_1 Y^{[i]} \alpha_2) \in \mathcal{P}^{[n]}$ then $(A^{(i)} \to \tau_{i-1}(\alpha_1) Y^{(i)} \tau_{i-1}(\alpha_2)) \in \mathcal{P}^{(i)}$.

We conclude from Definition 4 that each $G^{(i)}$ is a linear CFG, and the grammars differ to each other by their superscript only (except for $G^{(0)}$). Informally, they represent layers of derivations of the grammar $G^{[n]}$. Our next step is to relate the layers: we do that iteratively by applying substitutions.

We will use the notions of substitution and linear grammar to define t-fold compositions. For $t \in \{1, \ldots, n\}$, we define $\sigma_t \colon \Sigma \cup v_{\mathcal{X}^{(t)}} \to \Sigma \cup v_{\mathcal{X}^{(t-1)}}$ as the substitution which maps each $v_{X^{(t)}}$ onto $L_{X^{(t)}}(G^{(t)})$ and leaves Σ unchanged. Note that for $t = 0$, the substitution σ_0 has the signature $\Sigma \cup v_{\mathcal{X}^{(0)}} \to \Sigma$.

Let ℓ, t be such that $0 \leq \ell \leq t \leq n$. We define σ_ℓ^t to be σ_ℓ if $t = \ell$ and $\sigma_\ell^{t-1} \circ \sigma_t$ otherwise (or equivalently $\sigma_\ell \circ \sigma_{\ell+1}^t$). Hence, σ_0^n is such that: $(\Sigma \cup v_{\mathcal{X}^{(n)}})^* \xrightarrow{\sigma_n} (\Sigma \cup v_{\mathcal{X}^{(n-1)}})^* \cdots (\Sigma \cup v_{\mathcal{X}^{(1)}})^* \xrightarrow{\sigma_1} (\Sigma \cup v_{\mathcal{X}^{(0)}})^* \xrightarrow{\sigma_0} \Sigma^*$.

Definition 5 Given a CFG $G = (\mathcal{X}, \Sigma, \mathcal{P})$, variable $X \in \mathcal{X}$ and $t \in \mathbb{N}$, define the *t-fold composition* to be $\sigma_0^t(v_{X^{(t)}})$.

The following lemma establishes equivalence between representations.

Lemma 4 *Given a* CFG $G = (\mathcal{X}, \Sigma, \mathcal{P})$*, variable* $X \in \mathcal{X}$ *and* $t \in \mathbb{N}$*, we have*

$$\sigma_0^t(v_{X^{(t)}}) = L_{X^{[t]}}(G^{[t]}) \ .$$

Proof The result is shown by induction on t.

$t = 0$**.** We have that $\sigma_0^0(v_{X^{(0)}}) = \sigma_0(v_{X^{(0)}}) = L_{X^{(0)}}(G^{(0)})$ which in turn equals $L_{X^{[0]}}(G^{[0]})$ by definition of $G^{[0]}$.

$t > 0$**.** For each word w we have:

$$\begin{aligned} w \in L_{X^{[t]}}(G^{[t]}) \text{ iff } & X^{[t]} \Rightarrow^* w && \text{def. of derivation (in } G^{[t]}) \\ \text{iff } & \exists \alpha_i \quad \varphi(\alpha_i) \wedge \alpha_i \Rightarrow^* w && \text{derive highest index first} \end{aligned}$$

where $\varphi(\alpha_i)$ is the property

$$X^{[t]} \Rightarrow^* \alpha_{i-1} \Rightarrow \alpha_i \quad \text{where } \mathit{Sidx}(\alpha_{i-1}) \subseteq \{t, t-1\}, \mathit{Sidx}(\alpha_i) \subseteq \{t-1\}$$

Let α_i be a word such that $\varphi(\alpha_i)$ holds. Then, α_i is necessarily of the form $x_1 A_1^{[t-1]} x_2 \dots x_s A_s^{[t-1]} x_{s+1}$ with $x_j \in \Sigma^*$ for every j. Then,

$$\begin{aligned} \alpha_i \Rightarrow^* w \text{ iff } & w \in x_1 L_{A_1^{[t-1]}}(G^{[t]}) \dots L_{A_s^{[t-1]}}(G^{[t]}) x_{s+1} && \text{def. of derivation} \\ \text{iff } & w \in x_1 L_{A_1^{[t-1]}}(G^{[t-1]}) \dots L_{A_s^{[t-1]}}(G^{[t-1]}) x_{s+1} && \text{only prod. of } G^{[t-1]} \\ \text{iff } & w \in x_1 \sigma_0^{t-1}(v_{A_1^{[t-1]}}) \dots \sigma_0^{t-1}(v_{A_s^{[t-1]}}) x_{s+1} && \text{ind. hyp.} \\ \text{iff } & w \in \sigma_0^{t-1}\Big(x_1 v_{A_1^{[t-1]}} \dots v_{A_s^{[t-1]}} x_{s+1}\Big) && \text{prop. of subst.} \\ \text{iff } & w \in \sigma_0^{t-1}\Big(x_1 \tau_{t-1}(A_1^{[t-1]}) \dots \tau_{t-1}(A_s^{[t-1]}) x_{s+1}\Big) && \text{def. of } \tau_{t-1} \\ \text{iff } & w \in \sigma_0^{t-1}\Big(\tau_{t-1}(\alpha_i)\Big) && \text{prop. of } \tau_{t-1} \end{aligned}$$

Finally, w belongs to $L_{X^{[t]}}(G^{[t]})$ iff there exists such a α_i verifying property $\varphi(\alpha_i)$ such that $w \in \sigma_0^{t-1}\Big(\tau_{t-1}(\alpha_i)\Big)$. As i is the least value such that $t \notin \mathit{Sidx}(\alpha_i)$, we have reached the end of a layer of the grammar $G^{(t)}$, so $w \in L_{X^{[t]}}(G^{[t]})$ iff $w \in \sigma_0^{t-1}\Big(L_{X^{(t)}}(G^{(t)})\Big)$, *i.e.* $L_{X^{[t]}}(G^{[t]}) = \sigma_0^{t-1}\Big(L_{X^{(t)}}(G^{(t)})\Big)$. By definition of $\sigma_t(v_{X^{(t)}})$, we get $L_{X^{[t]}}(G^{[t]}) = \sigma_0^{t-1}\Big(\sigma_t(v_{X^{(t)}})\Big)$, and by definition of σ_0^t, we finally have proved $L_{X^{[t]}}(G^{[t]}) = \sigma_0^t(v_{X^{(t)}})$. □

4 Constructing a Parikh Equivalent Bounded Subset

We now show how to solve Pb. 1 for the class of t-fold compositions. This will complete the solution to Pb. 1 for the class of CFLs, hence the proof of Th. 1. In this section, we give an effective construction of a bounded expression that solve Pb. 1 first for regular languages, then for linear languages, and finally for t-fold compositions.

First we need to introduce the notion of semilinear sets. Recall that a set $S \subseteq \mathbb{N}^k$, $k \geq 1$, is *linear* if there is an *offset* $\mathbf{b} \in \mathbb{N}^k$ and *periods* $\mathbf{p}_1, \ldots, \mathbf{p}_j \in \mathbb{N}^k$ such that $S = \{\mathbf{b} + \sum_{i=1}^{j} \lambda_i \mathbf{p}_i \mid \lambda_1, \ldots, \lambda_j \in \mathbb{N}\}$. Let $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_j\}$, we write S as $\mathcal{L}(\mathbf{b}; P)$. A set is *semilinear* if it is the union of a finite number of linear sets. Parikh's theorem (cf. [12]) shows that the Parikh image of every CFL is a semilinear set that is effectively computable.

4.1 Regular Languages

The construction of a bounded expression that solves Pb. 1 for a regular language L is known from [16] (see also [17], Lem. 4.1). We give here a proof of this result inspired by recent development about the computation of Parikh images of regular languages (see [21]).

In all this subsection, we fix a regular grammar $R = (Q, \Sigma, \delta)$ with $Q = \{q_1, \ldots, q_n\}$ and $\Sigma = \{a_1, \ldots, a_k\}$, where we assign Σ a fixed linear order.

The partial derivation $\tau \equiv p_0 \Rightarrow u_1 \cdot p_1 \Rightarrow u_1 u_2 \cdot p_2 \Rightarrow^* u_1 \ldots u_r \cdot p_r$, with $p_1, \ldots, p_r \in Q$ and $u_1, \ldots, u_r \in \Sigma^*$, is said *elementary* if there is no $0 \leq i < j \leq r$ such that $p_i = p_j$. We say that τ is a *cycle* if $p_0 = p_r$, and finally that τ is an *elementary cycle* if it is a cycle but its prefix $p_0 \Rightarrow^* u_{r-1} \cdot p_{r-1}$ is elementary. We extend the notion of elementary to the derivations of R.

Definition 6 Given $q \in Q$ we define the set $W_q = \{w \mid q \Rightarrow^* w \cdot q \wedge |w| \leq n\}$. Moreover, for every language L, we define $[L]_{\Pi}$ to be a Parikh-equivalent subset of L such that, for every vector $\mathbf{b} \in \Pi(L)$, there is exactly one word $w \in [L]_{\Pi}$ such that $\Pi(w) = \mathbf{b}$. Then we inductively define the set of bounded expressions $\{B_i\}_{i \geq 0}$ over Σ as follows:

$$B_0 = \bigodot_{q \in Q} \{w^* \mid w \in [W_q]_{\Pi}\}$$
$$B_i = B_{i-1} \odot \left(\bigodot_{a \in \Sigma} a^* \right) \odot B_0 \ .$$

Lemma 5 *Let $R = (Q, \Sigma, \delta)$ be a regular grammar and let $q \in Q$, the bounded expression $B = B_{(n-1)^2}$ solves Pb. 1 for the instance $L_q(R)$.*

Proof We assume wlog that q does not occur on the right-hand side of any productions of δ. We have to prove that $\Pi(B_{(n-1)^2} \cap L_q(R)) \supseteq \Pi(L_q(R))$. In order to prove it, let $w \in L_q(R)$, and $\tau \equiv q = p_0 \Rightarrow u_1 \cdot p_1 \Rightarrow u_1 u_2 \cdot p_2 \Rightarrow \cdots \Rightarrow u_1 \ldots u_r \cdot p_r \Rightarrow u_1 \ldots u_r u_{r+1} = w$ be a derivation of w. We denote the subderivation $u_1 \ldots u_i \cdot p_i \Rightarrow^* u_1 \ldots u_j \cdot p_j$ of τ by $\tau[i, j]$ in the sequel.

We associate to each state p occurring in τ, the maximum index $i \in \{0, \dots, r\}$ such that $p_i = p$. We can order all these indexes increasingly: $i_0 < i_1 < \cdots < i_s$ with $s < n$. Observe that $i_s = r$ and also that, since q does not occur on the right-hand side of any productions of δ, $i_0 = 0$. Using techniques of graph theory, we can easily decompose for every $j \in \{0, \dots, s-1\}$ each subderivation $\tau[i_j, i_{j+1}]$ of τ as the interleaving of

- an elementary partial derivation $\tau_j \equiv p_{i_j} \Rightarrow^* v_j \cdot p_{i_{j+1}}$ of length at most $n-1$,
- finitely many elementary cycles $C_1^{(j)} \equiv p_1^{(j)} \Rightarrow^* w_1^{(j)} \cdot p_1^{(j)}, \dots, C_{h_j}^{(j)} \equiv p_{h_j}^{(j)} \Rightarrow^* w_{h_j}^{(j)} \cdot p_{h_j}^{(j)}$, producing words $w_1^{(j)}, \dots, w_{h_j}^{(j)}$ each of length at most n,

such that $\Pi(u_{i_j+1} \dots u_{i_{j+1}}) = \Pi(v_j) + \sum_{i=1}^{h_j} \Pi(w_i^{(j)})$.

Observe that $w_i^{(j)} \in W_{p_i^{(j)}}$ for every $j \in \{0, \dots, s\}$ and $i \in \{1, \dots, h_j\}$. We will moreover assume wlog that $w_i^{(j)} \in \left[W_{p_i^{(j)}}\right]_\Pi$.

Such a decomposition result, however, does not guarantee that every $C_i^{(j)}$ for $i \in \{1, \cdots, h_j\}$ meet with τ_j (see [21, Fact 7.3.3]) , which means that some $p_i^{(j)}$ may not appear in the partial derivation τ_j. On the other hand, $C_i^{(j)}$ must visit states from $p_{i_0}, \dots, p_{i_s}$ as this sequence contains all states in τ.

Therefore, we can conclude that there exists a derivation τ' given by some interleaving of $\tau_0 \cdots \tau_{s-1}$ with the elementary cycles in $C_1^{(0)}, \dots, C_{h_0}^{(0)}, \dots, C_1^{(s)}, \dots, C_{h_s}^{s}$ such that for w', the word generated by τ', we have $\Pi(w') = \Pi(w)$. Moreover, since every $w_i^{(j)}$ belongs to $\left[W_{p_i^{(j)}}\right]_\Pi$, we conclude from the definition of B_0, $s < n$ and the fact that each τ_j is no more than $n-1$ steps, that $w' \in (B_{n-1})^{n-1} \subseteq B_{(n-1)^2}$. This proves that $\Pi(w) \in \Pi(B_{(n-1)^2} \cap L_q(R))$. □

We now derive a bound on the size of $B_{(n-1)^2}$. We start by bounding the size of B_0. First, for a fixed alphabet size k, we have $|\Sigma^{\leq n}| = k^n$, but $\left|\left[\Sigma^{\leq n}\right]_\Pi\right| \leq \binom{n+k}{k}$. This is because the latter is the number of solutions to the equation $x_1 + \cdots + x_k \leq n$ for non-negative integers $x_1, \dots, x_k$. The term $\binom{n+k}{k} = 2^{O(k \log n)}$ is polynomial in n for each fixed k.

By definition of the operator $[\cdot]_\Pi$, the number of words in $[W_q]_\Pi$ is bounded above by $\left[\Sigma^{\leq n}\right]_\Pi$, and hence $\left|[W_q]_\Pi\right| \leq \binom{n+k}{k}$. Moreover, each word in W_q has length at most n. So, the concatenation $\bigodot \{w^* \mid w \in [W_q]_\Pi\}$ has size at most $n \cdot \binom{n+k}{k}$. The size of the bounded expression B_0 is then bounded by

$$n \cdot n \cdot \binom{n+k}{n} = n^2 \cdot 2^{O(k \log n)} = 2^{O(k \log n)} \quad .$$

Thus, for a fixed value of k the size of B_0 is polynomial in n. From the definition of B_i, for each polynomial P, we have $B_{P(n)}$ is polynomial in n (and exponential in k). To compute $B_{(n-1)^2}$, inspired again by [21, Lem. 7.3.6], we will use dynamic programming. Note that we just have to compute the bounded expression B_0, and then repeat it $(n-1)^2$ times interleaved with $\bigodot_{a \in \Sigma} a^*$ to compute $B_{(n-1)^2}$.

We denote by $\preceq$ the lexicographic order over Σ^*. Let $\perp$ be a fresh letter, and let $W = \{\perp\} \cup \Sigma^*$: we can extend the order $\preceq$ to W by $w \preceq \perp$ for every $w \in \Sigma^*$.

For every vector $\mathbf{b} \in \mathbb{N}^k$, we define $M^{\mathbf{b}} = (m^{\mathbf{b}}_{i,j})_{1 \leq i,j \leq n}$ the matrix over W where $m^{\mathbf{b}}_{i,j}$ is the minimal word $w \in \Sigma^*$ with Parikh image $\Pi(w) = \mathbf{b}$ such that there exists a partial derivation $\tau \equiv q_i \Rightarrow^* w \cdot q_j$, if such a word exists, and $\bot$ otherwise.

We can define a minimization operation $\vee : W \times W \to W$ defined for $w, w' \in W$ by $\vee(w, w')$ the minimal element in $\{w, w'\}$ for the order $\preceq$. We can extend this operation over the finite sets of elements of W: hence, if $S \subseteq W$, $\bigvee S$ is the unique minimal element in S, for the order $\preceq$. Moreover, we can extend the concatenation $\odot$ of words to elements of W by defining for $w \in \Sigma^*$, $\bot \odot w = w \odot \bot = \bot \odot \bot = \bot$. These two operations make possible to multiply matrices over W. Finally, let $\leq$ be the partial order over $\mathbb{N}^k$ defined by $\mathbf{b} \leq \mathbf{b}'$ if and only if for every $i \in \{1, \ldots, k\}$, $\mathbf{b}_i \leq \mathbf{b}'_i$.

Lemma 6 *Let $\mathbf{b} = (r_1, r_2, \ldots, r_{i-1}, r_i + 1, r_{i+1}, \ldots, r_k)$ with each $r_i \in \mathbb{N}$. Then, the following identity holds*

$$M^{\mathbf{b}} = \bigvee_{\mathbf{c},\mathbf{d}} M^{\mathbf{c}} \odot M^{\mathbf{e}_i} \odot M^{\mathbf{d}} \tag{1}$$

where $(\mathbf{e}_i)_{1 \leq i \leq k}$ is the standard basis for $\mathbb{R}^k$, $\mathbf{c}$ ranges over all vectors $\leq \mathbf{b}$ whose i-th entry is 0, and $\mathbf{d}$ is the vector $\mathbf{b} - \mathbf{e}_i - \mathbf{c}$.

Proof Let $i, j \in \{1, \ldots, n\}$. Any partial derivation $q_i \Rightarrow^* u \cdot q_j$ with $\Pi(u) = \mathbf{b}$ is uniquely decomposed into three partial derivations $\tau_1 \equiv q_i \Rightarrow^* v \cdot p, \tau_2 \equiv p \Rightarrow a_i \cdot p', \tau_3 \equiv p' \Rightarrow^* w \cdot q_j$ with the word v containing no letter a_i. Thus, the vector $\mathbf{c} = \Pi(v)$ is $\leq \mathbf{b}$ and its i-th entry is 0, $\Pi(a_i) = \mathbf{e}_i$ and $\mathbf{d} = \Pi(w) = \mathbf{b} - \mathbf{e}_i - \mathbf{c}$. Hence, the coefficient $m^{\mathbf{b}}_{i,j}$ is $\succeq$ than the coefficient of index (i, j) of the matrix $N = \bigvee_{\mathbf{c},\mathbf{d}} M^{\mathbf{c}} \odot M^{\mathbf{e}_i} \odot M^{\mathbf{d}}$.

Reciprocally, the concatenation of three *compatible* partial derivations leading to words v, a_i, w verifying $\Pi(v) \leq \mathbf{b}$ with i-th entry equal to 0, and $\Pi(w) = \mathbf{b} - \mathbf{e}_i - \Pi(v)$, is a partial derivation from q_i to q_j leading to a word u with Parikh image $\Pi(u) = \mathbf{b}$. Hence, $m^{\mathbf{b}}_{i,j}$ is $\preceq$ than the coefficient of index (i, j) of the matrix N. □

Applying this lemma and dynamic programming, we get

Lemma 7 *We can compute the matrices $M^{\mathbf{b}}$ for vector $\mathbf{b}$ such that $b_1 + \ldots + b_k \leq n$ in time $2^{O(k \log n)}$. Hence, for every state $q_i \in Q$, we can compute the set $[W_q]_\Pi = \bigcup_{b_1 + \ldots + b_k = n} \{m^{\mathbf{b}}_{i,i}\}$ in time $2^{O(k \log n)}$.*

Finally, we show that the exponential dependency on k cannot be improved.

Lemma 8 *There is a family $\{L_k\}_{k \in \mathbb{N}}$ of regular languages, where each L_k is over an alphabet of size $2k$, such that every bounded expression B_k solving Pb. 1 for instance L_k has size $2^{\Omega(k)}$.*

Proof Define L_k to be a regular language over alphabet $\{a_1, b_1, \ldots, a_k, b_k\}$ given by the regular expression

$$\big((a_1 \mid b_1) \cdot (a_2 \mid b_2) \cdots (a_k \mid b_k)\big)^* \ .$$

We show that every bounded B_k such that $\Pi(L_k \cap B_k) = \Pi(L_k)$ must be of size exponential in k by contradiction.

Fix a k. Assume that $B = w_1^* \dots w_m^*$ solves Pb. 1 for instance L_k. Let $\widehat{L_k} = (a_1 \mid b_1) \cdot (a_2 \mid b_2) \dots (a_k \mid b_k)$. Since for all $y, z \in \widehat{L_k}$, we have $\Pi(y) = \Pi(z)$ implies $z = y$, we have that for each $i \geq 0$ and for each $z \in \widehat{L_k}$, the word z^i is in B.

Pick $z \in \widehat{L_k}$. For each $i_z \geq 0$, we know that

$$z^{i_z} = w_1^{i_{z1}} w_2^{i_{z2}} \dots w_m^{i_{zm}}$$

for some $i_{z1}, i_{z2}, \dots, i_{zm}$, and further, by picking i_z big enough, we can ensure that $i_{zj} > 1$ for some j. Then, $w_j \odot w_j$ contains z as a subword (since z does not have any letters repeated). Also, for any $y \in \widehat{L_k} \setminus \{z\}$, it cannot be that $w_j \odot w_j$ contains y as a subword, even though by the same argument as for z, there is some j' such that $w_{j'} \odot w_{j'}$ contains y as a subword. This means that $m \geq 2^k$. □

4.2 Linear Languages

We now extend the previous construction to the case of linear languages. Recall that linear languages are the main ingredient of t-fold compositions. Lemma 9 gives a characterization of linear languages based on regular languages, homomorphism, and some additional structures.

Lemma 9 *(from [14]) Let $G = (\mathcal{X}, \Sigma, \mathcal{P})$ be a linear* CFG*. There exist an alphabet A and its distinct copy $\widetilde{A}$, an homomorphism $h : (A \cup \widetilde{A})^* \to \Sigma^*$ and a regular language $R = (\mathcal{X}, A, \delta)$ such that for every $X \in \mathcal{X}$ we have $L_X(G) = h(L_X(R)\widetilde{A}^*) \cap S$ where $S = \{w\widetilde{w}^{-1} \mid w \in A^*\}$ and w^{-1} denotes the reverse image of the word w. Moreover there is an effective procedure to construct h, A, and R.*

Proof Define the alphabet A to be $\{a_p \mid p \in \mathcal{P}\}$. Define the regular CFG $R = (\mathcal{X}, A, \delta)$ such that

$$\delta = \{(X, a_p Y) \mid p = (X, \alpha Y \beta) \in \mathcal{P} \wedge p \in \mathcal{X} \times \Sigma^* \mathcal{X} \Sigma^*\} \cup \{(X, a_p) \mid p = (X, \alpha) \in \mathcal{P} \wedge p \in \mathcal{X} \times \Sigma^*\} \ .$$

Note that $\delta \subseteq \mathcal{X} \times A(\mathcal{X} \cup \varepsilon)$ shows that for every $X \in \mathcal{X}$, $L_X(R)$ is a regular language. Next we define the homomorphism, h which, for each $p = (X, \alpha Y \beta) \in \mathcal{P}$, maps a_p and $\widetilde{a_p}$ to α and β, respectively. By construction and induction on the length of a derivation, it is easily seen that the result holds. □

Next, we have a technical lemma which relates homomorphism and the Parikh image operator.

Lemma 10 *Let $X, Y \subseteq \Sigma^*$ and a homomorphism $h\colon \Sigma^* \to A^*$, we have:*

$$\Pi(X) = \Pi(Y) \text{ implies } \Pi(h(X)) = \Pi(h(Y)) \ .$$

Proof It suffices to show that the result holds for $=$ replaced by $\subseteq$. Let $x' \in h(X)$. We know that there exists $x \in X$ such that $x' = h(x)$. The equality $\Pi(X) = \Pi(Y)$ shows that there exists $y \in Y$ such that $\Pi(y) = \Pi(x)$. It is clear by property of homomorphism that $\Pi(h(y)) = \Pi(h(x))$. □

The next result shows that a bounded expression that solves Pb. 1 can be effectively constructed for every linear language L.

Proposition 1 *For every linear language $L = h(R\widetilde{A}^* \cap S)$ where h and R are given, there is an effective procedure which solves Pb. 1 for the instance L.*

Proof Since R is a regular language, we can use the result of Prop. 5 to effectively compute the set $\{w_1, \ldots, w_m\}$ of words such that for $R' = R \cap w_1^* \cdots w_m^*$ we have $\Pi(R') = \Pi(R)$. Also, we observe that for every language $Z \subseteq A^*$ we have $Z\widetilde{A}^* \cap S = \{w\widetilde{w}^{-1} \mid w \in Z\}$.

$$
\begin{aligned}
&\Pi(R') = \Pi(R) && \text{by above}\\
&\text{only if } \Pi(R'\widetilde{A}^* \cap S) = \Pi(R\widetilde{A}^* \cap S) && \text{by above}\\
&\text{only if } \Pi(h(R'\widetilde{A}^* \cap S)) = \Pi(h(R\widetilde{A}^* \cap S)) && \text{Lem. 10}\\
&\text{only if } \Pi(h(R'\widetilde{A}^* \cap S)) = \Pi(L) && \text{def. of } L\\
&\text{only if } \Pi(h(R\widetilde{A}^* \cap S) \cap w_1^* \cdots w_m^* \widetilde{w_m^{-1}}^* \cdots \widetilde{w_1^{-1}}^*) = \Pi(L) && \text{def. of } R'\\
&\text{only if } \Pi(h(R\widetilde{A}^* \cap S) \cap h(w_1^* \cdots w_m^* \widetilde{w_m^{-1}}^* \cdots \widetilde{w_1^{-1}}^*)) = \Pi(L)\\
&\text{only if } \Pi(L \cap h(w_1^* \cdots w_m^* \widetilde{w_m^{-1}}^* \cdots \widetilde{w_1^{-1}}^*)) = \Pi(L) && \text{def. of } L\\
&\text{only if } \Pi(L \cap h(w_1)^* \cdots h(w_m)^* h(\widetilde{w_m^{-1}})^* \cdots h(\widetilde{w_1^{-1}})^*) = \Pi(L)
\end{aligned}
$$

which concludes the proof since $h(w) \in \Sigma^*$ if $w \in (A \cup \widetilde{A})^*$. □

From the proof, and the construction for regular languages, it is clear that the bounded expression B is exponential in the size of the alphabet k but polynomial in n. The exponential dependence on k is inevitable and follows from the lower bound for regular languages.

4.3 Linear languages with substitutions

Our goal is to solve Pb. 1 for the class of t-fold compositions, *i.e.* for languages of the form $\sigma_0^t(v_{X^{(t)}})$. Prop. 1 gives an effective procedure for the case $\sigma_\ell^\ell(v_{X^{(\ell)}})$ since it is a linear language. Prop. 2 generalizes to the case $\sigma_t^\ell(v_{X^{(\ell)}})$ where $t < \ell$: given a solution to Pb. 1 for the instance $\sigma_{t+1}^\ell(v_{X^{(\ell)}})$, there is an effective procedure for Pb. 1 for the instance $\sigma_t \circ \sigma_{t+1}^\ell = \sigma_t^\ell(v_{X^{(\ell)}})$.

But prior to that we need the following result.

Lemma 11 *Let L and B be respectively a* CFL *and a bounded expression over Σ such that B solves Pb. 1 for instance L, i.e. $\Pi(L \cap B) = \Pi(L)$. There is an effectively computable bounded expression B' which solves Pb. 1 for instance L^*, i.e. $\Pi(L^r \cap B') = \Pi(L^r)$ for all $r \in \mathbb{N}$.*

Proof By Parikh's theorem, we know that $\Pi(L)$ is a computable semilinear set $\bigcup_{i=1}^{\ell} \mathcal{L}(\mathbf{c}_i; P_i)$ where each $P_i = \{\mathbf{p}_{i1}, \ldots, \mathbf{p}_{ik_i}\}$. Let us consider $u_1, \ldots, u_\ell \in L$ such that $\Pi_\Sigma(u_i) = \mathbf{c}_i$ for $i \in \{1, \ldots, \ell\}$.

Let $B' = u_1^* \cdots u_\ell^* B^\ell$. We see that B' is a bounded expression. Let $r > 0$ be a natural integer. We have to prove that $\Pi(L^r) \subseteq \Pi(L^r \cap B')$.

case $r \leq \ell$. We conclude from the preservation of Π (Lem. 1) and the hypothesis $\Pi(L) = \Pi(L \cap B)$ that

$$\begin{aligned}
\Pi(L^r) &= \Pi((L \cap B)^r) \\
&\subseteq \Pi(L^r \cap B^r) && \text{monotonicity of } \Pi \\
&\subseteq \Pi(L^r \cap B^\ell) && B^r \subseteq B^\ell \text{ since } \varepsilon \in B \\
&\subseteq \Pi(L^r \cap B') && \text{def. of } B'
\end{aligned}$$

case $r > \ell$. Let us consider $w \in L^r$. For every $i \in \{1, \dots, \ell\}$ and $j \in \{1, \dots, k_i\}$, there exist some positive integers λ_{ij} and μ_i, with $\sum_{i=1}^{\ell} \mu_i = r$ such that

$$\Pi(w) = \sum_{i=1}^{\ell} \mu_i \mathbf{c}_i + \sum_{i=1}^{\ell} \sum_{j=1}^{k_i} \lambda_{ij} \mathbf{p}_{ij} \ .$$

We define a new variable for each $i \in \{1, \dots, \ell\}$: $\alpha_i = \begin{cases} \mu_i - 1 & \text{if } \mu_i > 0 \\ 0 & \text{otherwise.} \end{cases}$.
For each $i \in \{1, \dots, \ell\}$, we also consider z_i a word of $L \cup \{\varepsilon\}$ such that $z_i = \varepsilon$ if $\mu_i = 0$ and $\Pi(z_i) = \mathbf{c}_i + \sum_{j=1}^{k_i} \lambda_{ij} \mathbf{p}_{ij}$ else.

Let $w' = u_1^{\alpha_1} \dots u_\ell^{\alpha_\ell} z_1 \dots z_\ell$. Clearly, $\Pi(w') = \Pi(w)$ and $w' \in u_1^* \cdots u_\ell^* (L \cup \{\varepsilon\})^\ell$. For each $i \in \{1, \dots, \ell\}$, $\Pi(L \cap B) = \Pi(L)$ shows that there is $z_i' \in (L \cap B) \cup \{\varepsilon\}$ such that $\Pi(z_i') = \Pi(z_i)$. Let $w'' = u_1^{\alpha_1} \dots u_\ell^{\alpha_\ell} z_1' \dots z_\ell'$. We find that $\Pi(w'') = \Pi(w)$, $w'' \in B'$ and we can easily verify that $w'' \in L^r$. □

Proposition 2 *Let*

1. *L be a* CFL *over Σ;*
2. *B a bounded expression such that $\Pi(L \cap B) = \Pi(L)$;*
3. *σ and τ be two substitutions over Σ such that for each $a \in \Sigma$, (i) $\sigma(a)$ and $\tau(a)$ are respectively a* CFL *and bounded expression and (ii) $\Pi(\sigma(a) \cap \tau(a)) = \Pi(\sigma(a))$.*

Then, there is an effective procedure that computes B' such that B' solves Pb. 1 for the instance $\sigma(L)$.

Proof Let $w_1, \dots, w_m \in \Sigma^*$ be the words such that $B = w_1^* \cdots w_m^*$. Let $L_i = \sigma(w_i)$ for each $i \in \{1, \dots, m\}$. Since $\sigma(a)$ is a CFL so is $\sigma(w_i)$ by property of the substitutions and the closure of CFLs by finite concatenations. For the same reason, $\tau(w_i)$ is a bounded expression. Next, Lem. 11 where the bounded expression is given by $\tau(w_i)$, shows that we can construct a bounded expression B_i such that for all $r \in \mathbb{N}$, $\Pi(L_i^r \cap B_i) = \Pi(L_i^r)$. Define $B' = B_1 \dots B_m$ that is a bounded expression. We have to prove the inclusion $\Pi(\sigma(L)) \subseteq \Pi(\sigma(L) \cap B')$ since the reverse one trivially holds. So, let $w \in \sigma(L)$. Since $\Pi(L \cap w_1^* \cdots w_m^*) = \Pi(L)$, there is a word $w' \in \sigma(L \cap w_1^* \cdots w_m^*)$ such that $\Pi(w) = \Pi(w')$. Then we have

$$\begin{aligned}
w' &\in \sigma(L \cap w_1^* \cdots w_m^*) \\
&\in \sigma(w_1^{r_1} \dots w_m^{r_m}) && \text{for some } r_1, \dots, r_m \\
&\in \sigma(w_1^{r_1}) \dots \sigma(w_m^{r_m}) && \text{property of subst.} \\
&\in \sigma(w_1)^{r_1} \dots \sigma(w_m)^{r_m} && \text{property of subst.} \\
&\in L_1^{r_1} \dots L_m^{r_m} && \sigma(w_i) = L_i
\end{aligned}$$

Algorithm 1: Bounded Sequence

Data: Linear CFGs $\{G^{(0)}, \ldots, G^{(n)}\}$
Data: A set of regular CFGs $\{\tilde{B}^{(0)}, \ldots, \tilde{B}^{(n)}\}$ such that
$\forall i \in \{0, \ldots, n\} \forall X \in \mathcal{X}$: $L_X(\tilde{B}^{(i)})$ solves Pb. 1 for instance $L_{X^{(i)}}(G^{(i)})$
Result: A set of regular CFGs $\{B^{(0)}, \ldots, B^{(n)}\}$ such that
$\forall i \in \{0, \ldots, n\} \forall X \in \mathcal{X}$: $L_X(B^{(i)})$ solves Pb. 1 for instance $\sigma_i^n(v_{X^{(n)}})$
1 Let $B^{(n)} = \tilde{B}^{(n)}$;
for $i = n-1, n-2, \ldots, 0$ **do**
  Let τ_i be the substitution which maps each $v_{X^{(i)}}$ onto $L_X(\tilde{B}^{(i)})$ and leaves each letter of Σ unchanged;
  foreach $X \in \mathcal{X}$ **do**
    Let $L_X(B^{(i)})$ be the language returned by Prop. 2 on the languages
2 $\sigma_{i+1}^n(v_{X^{(n)}})$ and $L_X(B^{(i+1)})$, and the substitutions σ_i, τ_i;

For each $i \in \{1, \ldots, m\}$, we have $\Pi(L_i^{r_i} \cap B_i) = \Pi(L_i^{r_i})$, so we can find $w'' \in (L_1^{r_1} \cap B_1) \ldots (L_m^{r_m} \cap B_m)$ such that $\Pi(w'') = \Pi(w')$. Definition of B' also shows that $w'' \in B'$. Moreover

$$
\begin{aligned}
w'' &\in (L_1^{r_1} \cap B_1) \ldots (L_m^{r_m} \cap B_m) \\
&\in L_1^{r_1} \ldots L_m^{r_m} \\
&\in \sigma(w_1)^{r_1} \ldots \sigma(w_m)^{r_m} && \sigma(w_i) = L_i \\
&\in \sigma(w_1^{r_1}) \ldots \sigma(w_m^{r_m}) && \text{property of subst.} \\
&\in \sigma(w_1^{r_1} \ldots w_m^{r_m}) && \text{property of subst.} \\
&\in \sigma(L \cap w_1^* \ldots w_m^*) && w_1^{r_1} \ldots w_m^{r_m} \in L \cap w_1^* \ldots w_m^* \\
&\in \sigma(L)
\end{aligned}
$$

Finally, $w'' \in B'$ and $w'' \in \sigma(L)$ and $\Pi(w'') = \Pi(w')$, which in turn equals $\Pi(w)$, prove the inclusion. □

We use the above result inductively to solve Pb. 1 for t-fold composition as follows: fix L to be $\sigma_{\ell+1}^t(v_{X^{(t)}})$, B to be the solution of Pb. 1 for the instance L, σ to be σ_ℓ and τ a substitution which maps every $v_{X^{(\ell)}}$ to a solution of Pb. 1 for the instance $\sigma_\ell(v_{X^{(\ell)}})$. Then B' is the solution of Pb. 1 for the instance $\sigma_\ell^t(v_{X^{(t)}})$.

4.4 t-fold compositions

Let L be a CFL such that $L = L_X(G)$ where $G = (\mathcal{X}, \Sigma, \mathcal{P})$ is a CFG with $n = |\mathcal{X}|$ and $X \in \mathcal{X}$. Let us now solve Pb. 1 where the instance is given by the n-fold composition $\sigma_0^n(v_{X^{(n)}})$. To do so, we compute, following def. 4, the linear CFGs $\{G^{(0)}, \ldots, G^{(n)}\}$ which define the n-fold composition $\sigma_0^n(v_{X^{(n)}})$ of def. 5. With the result of Prop. 1, we inductively construct the regular CFG $B^{(0)}, \ldots, B^{(n)}$ each of which is such that $\forall X \in \mathcal{X}$: $L_X(B^{(\ell)})$ solves Pb. 1 for instance $\sigma_\ell^n(v_{X^{(n)}})$. The above reasoning is formally explained in Alg. 1 for which we prove the following invariant.

Lemma 12 *When Alg. 1 returns we have:*

$$\forall i \in \{0, \ldots, n\} \forall X \in \mathcal{X}\colon L_X(B^{(i)}) \text{ solves Pb. 1 for instance } \sigma_i^n(v_{X^{(n)}}) \ .$$

Proof By induction on i:
Base case. ($i = n$) Algorithm 1, line 1 shows that $B^{(n)}$ is initialized with $\tilde{B}^{(n)}$. Therefore by property of $\tilde{B}^{(n)}$ we find that for every $X \in \mathcal{X}$ $L_X(\tilde{B}^{(n)})$ solves Pb. 1 for instance $L_{X^{(n)}}(G^{(n)})$, hence for instance $\sigma_n(v_{X^{(n)}})$ by definition of σ_n and finally for instance $\sigma_n^n(v_{X^{(n)}})$ because $\sigma_n = \sigma_n^n$.
Inductive case. ($0 \leq i < n$) At line 1, $L_X(B^{(i)})$ coincide with the language of the bounded expression returned by Prop. 2 provided some assumptions are satisfied. Let us show that those assumptions hold: (1) $\sigma_{i+1}^n(v_{X^{(n)}})$ is a CFL (CFLs are closed by context-free substitutions), (2) $L_X(B^{(i+1)})$ coincides with the language of a bounded expression which, by induction hypothesis, solves Pb. 1 for instance $\sigma_{i+1}^n(v_{X^{(n)}})$, (3) for every variable $X \in \mathcal{X}$ we have $\sigma_i(v_{X^{(i)}}) = L_{X^{(i)}}(G^{(i)})$ is a CFL, $\tau_i(v_{X^{(i)}}) = L_X(\tilde{B}^{(i)})$ coincides with the language of a bounded expression such that $\tau_i(v_{X^{(i)}})$ solves Pb. 1 for instance $\sigma_i(v_{X^{(i)}})$. Hence by Prop. 2 we find that for every $X \in \mathcal{X}$, $L_X(B^{(i)})$ solves Pb. 1 for instance $\sigma_i\big(\sigma_{i+1}^n(v_{X^{(n)}})\big)$, hence for instance $\sigma_i^n(v_{X^{(n)}})$ because $\sigma_i^n = \sigma_i \circ \sigma_{i+1}^n$. □

Finally the procedure which solves Pb. 1 for CFL instances, hence the proof of Th. 1, goes as follows.

Proof (of Th. 1) Let $G = (\mathcal{X}, \Sigma, \mathcal{P})$ be a CFG with initial variable $X \in \mathcal{X}$ where $|\mathcal{X}| = n$. Moreover let B be a bounded expression, Cor. 1 shows that B solves Pb. 1 for instance $L_X(G)$ iff so does B for instance $\mathcal{Y}(\mathcal{D}_X^n)$ iff so does B for instance $L_{X^{[n]}}(G^{[n]})$ (by Lem. 3) iff so does B for instance $\sigma_0^n(v_{X^{(n)}})$ (by Lem. 4). Moreover the CFGs $\{G^{(0)}, \ldots, G^{(n)}\}$ which represent $\sigma_0^n(v_{X^{(n)}})$ are effectively constructible given G and n. Finally the Alg. 1 and the result of Lem. 12 shows that there exists an effective procedure to solve Pb. 1 for $\sigma_0^n(v_{X^{(n)}})$. □

This concludes the proof of Th. 1. Unfortunately, as the example below illustrates, the bounded expression can be exponential in the size of the grammar even when the alphabet is held to a fixed size.

Lemma 13 *There is a family $\{G_n\}_{n\in\mathbb{N}}$ of CFG each of which defined over the alphabet $\Sigma = \{0, 1\}$ such that every bounded expression B_n solving Pb. 1 for $L_S(G_n)$ has size $2^{\Omega(n)}$.*

Proof Define G_n to have variables $\{S\} \cup \{A_0, \ldots, A_n\}$, and productions:

$$S \to \epsilon \qquad A_n \to A_{n-1}A_{n-1} \qquad A_0 \to 0$$
$$S \to 1A_nS \qquad \vdots$$
$$A_1 \to A_0A_0$$

The definition of G_n shows that $L_S(G_n) = (10^{2^n})^*$. A possible (and trivial) bounded expression B_n solving Pb. 1 for instance $L_S(G_n)$ is $(10^{2^n})^*$, which is exponential in the size of n (note that the size of B_n is given by the number of letters in its words). We now show that any bounded expression which solves Pb. 1 for instance $L_S(G_n)$ must be exponential n.

First, note that there is at most one word of any length in $L_S(G_n)$, and so for any $i, j \in \mathbb{N}$, we have that $\Pi(w^i) = \Pi(w^j)$ implies $i = j$. Thus, B must contain every word w^i, $i \geq 0$.

Suppose that $B = w_1^* \dots w_m^*$ is a bounded expression solving Pb. 1 for instance $L_S(G_n)$. Clearly, at least one of $w_1, \dots, w_m$ must have the letter 1. Since m is fixed, for large enough i (indeed, $i > m$ suffices), we will have that $w^i = w_1^{i_1} w_2^{i_2} \dots w_m^{i_m}$ such that there is a j with the following properties: (1) w_j contains the letter 1, and (2) $i_j > 1$. This is because w^i has exactly i occurrences of 1, and all these occurrences must be "captured" by B. However, in w^i, any two occurrences of 1 has exactly 2^n occurrences of 0 between them. This implies that the length of w_j must be at least 2^n. □

Iterative Algorithm. We conclude this section by showing a result related to the notion of progress if the result of Th. 1 is applied repeatedly.

Lemma 14 *Given a* CFL L*, define two sequences* $(L_i)_{i\in\mathbb{N}}$, $(B_i)_{i\in\mathbb{N}}$ *such that (1)* $L_0 = L$*, (2)* B_i *is a bounded expression and* $\Pi(L_i \cap B_i) = \Pi(L_i)$*, (3)* $L_{i+1} = L_i \cap \overline{B_i}$*. For every* $w \in L$*, there exists* $i \in \mathbb{N}$ *such that* $w \notin L_i$*. Moreover, given* L_0*, there is an effective procedure to compute* L_i *for every* $i > 0$*.*

Proof Let $w \in L$ and let $v = \Pi(w)$ be its Parikh image. We conclude from $\Pi(L_0 \cap B_0) = \Pi(L_0)$ that there exists a word $w' \in B_0$ such that $\Pi(w') = v$. Two cases arise: either $w' = w$ and we are done; or $w' \neq w$. In that case $L_1 = L_0 \cap \overline{B_0}$ shows that $w' \notin L_1$. Intuitively, at least one word with the same Parikh image as w has been selected by B_0 and then removed from L_0 by definition of L_1. Repeatedly applying the above reasoning shows that at each iteration there exists a word w'' such that $\Pi(w'') = v$, $w'' \in B_i$ and $w'' \notin L_{i+1}$ since $L_{i+1} = L_i \cap \overline{B_i}$. Because there are only finitely many words with Parikh image v we conclude that there exists $j \in \mathbb{N}$, such that $w \notin L_j$. The effectiveness result follows from the following arguments: (1) as we have shown above (our solution to Pb. 1), given a CFL L there is an effective procedure that computes a bounded expression B such that $\Pi(L \cap B) = \Pi(L)$; (2) the complement of B is a regular language effectively computable; and (3) the intersection of a CFL with a regular language is again a CFL that can be effectively constructed (see [14]). □

Intuitively this result shows that given a context-free language L, if we repeatedly compute and remove a Parikh-equivalent bounded subset of L ($L \cap \overline{B}$ is effectively computable since B is a regular language), then each word w of L is eventually removed from it.

5 Application I: Multithreaded Procedural Programs

A common programming model consists of multiple recursive threads communicating via shared memory. Formally, we model such systems as pushdown networks [20]. Let k be a positive integer, a *pushdown network* is a triple $\mathcal{N} = (G, \Gamma, (\Delta_i)_{1\leq i\leq k})$ where G is a finite non-empty set of *globals*, Γ is the *stack alphabet*, and for each $1 \leq i \leq k$, Δ_i is a finite set of *transition rules* of the form $\langle g, \gamma\rangle \hookrightarrow \langle g', \alpha\rangle$ for $g, g' \in G$, $\gamma \in \Gamma$, $\alpha \in \Gamma^*$.

A *local configuration* of $\mathcal{N}$ is a pair $(g, \alpha) \in G \times \Gamma^*$ and a *global configuration* of $\mathcal{N}$ is a tuple $(g, \alpha_1, \ldots, \alpha_k)$, where $g \in G$ and $\alpha_1, \ldots, \alpha_k \in \Gamma^*$ are individual stack content for each thread. Intuitively, the system consists of k threads, each of which with its own stack, and the threads can communicate by reading and manipulating the global storage represented by g.

We define the local transition relation of the i-th thread, written $\to_i$, as follows: $(g, \gamma\beta) \to_i (g', \alpha\beta)$ iff $\langle g, \gamma\rangle \hookrightarrow \langle g', \alpha\rangle$ in Δ_i and $\beta \in \Gamma^*$. The transition relation of $\mathcal{N}$, denoted $\to$, is defined as follows: $(g, \alpha_1, \ldots, \alpha_i, \ldots, \alpha_k) \to (g', \alpha_1, \ldots, \alpha'_i, \ldots, \alpha_k)$ iff $(g, \alpha_i) \to_i (g', \alpha'_i)$ for some $i \in \{1, \ldots, k\}$. By $\to_i^*$, $\to^*$, we denote the reflexive and transitive closure of these relations. Let C_0 and C be two global configurations, the *reachability problem* asks whether $C_0 \to^* C$ holds. An instance of the reachability problem is denoted by a triple $(\mathcal{N}, C_0, C)$.

A *pushdown system* is a pushdown network where $k = 1$, namely (G, Γ, Δ). A *pushdown acceptor* is a pushdown system extended with an initial configuration $c_0 \in G \times \Gamma^*$, labeled transition rules of the form $\langle g, \gamma\rangle \overset{\lambda}{\hookrightarrow} \langle g'\alpha\rangle$ for g, g', γ, α defined as above and $\lambda \in \Sigma \cup \{\varepsilon\}$. A pushdown acceptor is given by a tuple $(G, \Gamma, \Sigma, \Delta, c_0)$. The language of a pushdown acceptor is defined as expected where the acceptance condition is given by the empty stack.

In what follows, we reduce the reachability problem for a pushdown network of k threads to a language problem for k pushdown acceptors. The pushdown acceptors obtained by reduction from the pushdown network settings have a special global $\bot$ that intuitively models an inactive state. The reduction also turns the globals into input letters which label transitions. The firing of a transition labeled with a global models a context switch. When such transition fires, every pushdown acceptor synchronizes on the label. The effect of such a synchronization is that exactly one acceptor will change its state from inactive to active by updating the value of its global (i.e. from $\bot$ to some $g \in G$) and exactly one acceptor will change from active to inactive by updating its global from some g to $\bot$. All the others acceptors will synchronize and stay inactive.

Given an instance of the reachability problem, that is a pushdown network $(G, \Gamma, (\Delta_i)_{1 \le i \le k})$ with k threads, two global configurations C_0 and C (assume wlog that C is of the form $(g, \varepsilon, \ldots, \varepsilon)$), we define a family of pushdown acceptors $\{(G', \Gamma, \Sigma, \Delta'_i, c_0^i)\}_{1 \le i \le k}$, where:

- $G' = G \cup \{\bot\}$, Γ is given as above, and $\Sigma = G \times \{1, \ldots, k\}$,
- Δ'_i is the smallest set such that:
  - $\langle g, \gamma\rangle \overset{\varepsilon}{\hookrightarrow} \langle g', \alpha\rangle$ in Δ'_i if $\langle g, \gamma\rangle \hookrightarrow \langle g', \alpha\rangle$ in Δ_i;
  - $\langle g, \gamma\rangle \overset{(g,j)}{\hookrightarrow} \langle \bot, \gamma\rangle$ for $j \in \{1, \ldots, k\} \setminus \{i\}$, $g \in G$, $\gamma \in \Gamma$;
  - $\langle \bot, \gamma\rangle \overset{(g,j)}{\hookrightarrow} \langle \bot, \gamma\rangle$ for $j \in \{1, \ldots, k\} \setminus \{i\}$, $g \in G$, $\gamma \in \Gamma$;
  - $\langle \bot, \gamma\rangle \overset{(g,i)}{\hookrightarrow} \langle g, \gamma\rangle$ for $g \in G$, $\gamma \in \Gamma$.
- let $C_0 = (g, \alpha_1, \ldots, \alpha_i, \ldots, \alpha_k)$, c_0^i is given by $(\bot, \alpha_i)$ if $i > 1$; (g, α_1) else.

Proposition 3 *Let k be a positive integer, and $(\mathcal{N}, C_0, C)$ be an instance of the reachability problem with k threads, one can effectively construct* CFL*s $(L_1, \ldots, L_k)$ (as pushdown acceptors) such that $C_0 \to^* C$ iff $L_1 \cap \cdots \cap L_k \neq \emptyset$.*

The converse of the proposition is also true, and since the emptiness problem for intersection of CFLs is undecidable [14], so is the reachability problem. We will

now compare two underapproximation techniques for the reachability problem: context-bounded switches [18] and bounded languages, which we first detail below.

Let $L_1, \ldots, L_k$ be CFLs, and consider the problem to decide if $\bigcap_{1\leq i\leq k} L_i \neq \emptyset$. We give a decidable sufficient condition: given an bounded expression B, we define the *intersection modulo* B of the languages $\{L_i\}_i$ as $\bigcap_i^{(B)} L_i = \left(\bigcap_i L_i\right) \cap B$. Clearly, $\bigcap_i^{(B)} L_i \neq \emptyset$ implies $\bigcap_i L_i \neq \emptyset$. Below we show that the problem $\bigcap_i^{(B)} L_i \neq \emptyset$ is decidable .

Lemma 15 *Given a bounded expression $B = w_1^* \cdots w_n^*$ and CFLs $L_1, \ldots, L_k$, it is decidable to check if $\bigcap_{1\leq i\leq k}^{(B)} L_i \neq \emptyset$.*

Proof Define the alphabet $A = \{a_1, \ldots, a_n\}$ disjoint from Σ. Let h be the homomorphism that maps the letters $a_1, \ldots, a_n$ to the words $w_1, \ldots, w_n$, respectively. We show that $\bigcap_{1\leq i\leq k} \Pi_A\left(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\right) \neq \emptyset$ iff $\bigcap_{1\leq i\leq k}^{(B)} L_i \neq \emptyset$.

We conclude from $w \in \bigcap_{1\leq i\leq k}^{(B)} L_i$ that $w \in B$ and $w \in L_i$ for every $1 \leq i \leq k$, hence there exist $t_1, \ldots, t_n \in \mathbb{N}$ such that $w = w_1^{t_1} \ldots w_n^{t_n}$ by definition of B. Then, we find that $(t_1, \ldots, t_n) \in \Pi_A(h^{-1}(w) \cap a_1^* \cdots a_n^*)$, hence that $(t_1, \ldots, t_n) \in \Pi_A(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*)$ for every $1 \leq i \leq k$ by above and finally that $(t_1, \ldots, t_n) \in \bigcap_{1\leq i\leq k} \Pi_A\left(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\right)$.

For the other implication, consider $(t_1, \ldots, t_n)$ a vector of $\bigcap_{1\leq i\leq k} \Pi_A\left(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\right)$ and let $w = w_1^{t_1} \ldots w_n^{t_n}$. For every $1 \leq i \leq k$, we will show that $w \in L_i \cap B$. As $(t_1, \ldots, t_n) \in \Pi_A\left(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\right)$, there exists a word $w' \in a_1^* \cdots a_n^*$ such that $\Pi_A(w') = (t_1, \ldots, t_n)$ and $h(w') \in L_i \cap B$. We conclude from $\Pi_A(w') = (t_1, \ldots, t_n)$, that $w' = a_1^{t_1} \ldots a_n^{t_n}$ and finally that, $h(w') = w$ belongs to $L_i \cap B$.

The class of CFLs is effectively closed under inverse homomorphism and intersection with a regular language [14]. Moreover, given a CFL, we can compute its Parikh image which is a semilinear set. Finally, we can compute the semilinear sets $\Pi_A\left(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\right)$ and the emptiness of the intersection of semilinear sets is decidable [12]. □

While Lem. 15 shows decidability for bounded expression language, in practice, we want to select B "as large as possible". We select B using Th. 1. We first compute for each language L_i the bounded expression $B_i = {w_1^{(i)}}^* \cdots {w_{n_i}^{(i)}}^*$ such that $\Pi(L_i \cap B_i) = \Pi(L_i)$. Finally, we choose $B = B_1 \cdots B_k$.

By repeatedly selecting and removing a bounded language B from each L_i where $1 \leq i \leq k$ we obtain a sequence $\{L_i^j\}_{j\geq 0}$ of languages such that $L_i = L_i^0 \supseteq L_i^1 \supseteq \ldots$ The result of Lem. 14 shows that for each word $w \in L_i$, there is some j such that $w \notin L_i^j$, hence that the above sequence is strictly decreasing, that is $L_i = L_i^0 \supsetneq L_i^1 \supsetneq \ldots$, and finally that if $\bigcap_{1\leq i\leq k} L_i \neq \emptyset$ then the iteration is guaranteed to terminate.

Alg. 2 gives the pseudocode for the special case of the intersection of two CFLs.

Comparison with Context-Bounded Reachability. A well-studied underapproximation for multithreaded reachability is given by context-bounded reachability [18]. We need a few preliminary definitions. We define the global reachability relation $\rightsquigarrow$ as a reachability relation where all the moves are made by a single thread: $(g, \alpha_1, \ldots, \alpha_i, \ldots, \alpha_n) \rightsquigarrow (g', \alpha_1, \ldots, \alpha_i', \ldots, \alpha_n)$ iff $(g, \alpha_i) \rightarrow_i^* (g', \alpha_i')$ for some $1 \leq i \leq n$. The relation $\rightsquigarrow$ holds between global configurations reachable

Algorithm 2: Intersection

Data: L_1^0, L_2^0 : CFLs
$L_1 \leftarrow L_1^0$, $L_2 \leftarrow L_2^0$;
repeat forever
  if $\Pi(L_1) \cap \Pi(L_2) = \emptyset$ **then**
    return $L_1^0 \cap L_2^0$ *is empty*
  else
    Compute B_1 and B_2 which solves Pb. 1 for instance L_1 and L_2, respectively;
    Compute $B = B_1 \odot B_2$ /* B solves Pb. 1 for instance $L_1 \cup L_2$ */;
    if $L_1 \cap^{(B)} L_2 \neq \emptyset$ **then**
      return $L_1^0 \cap L_2^0$ *is not empty*
  $L_1 \leftarrow L_1 \cap \overline{B}$, $L_2 \leftarrow L_2 \cap \overline{B}$

from each other in a single *context*. Furthermore we denote by $\rightsquigarrow_j$, where $j \geq 0$, the reachability relation within j contexts: $\rightsquigarrow_0$ is the identity relation on global configurations, and $\rightsquigarrow_{i+1} = \rightsquigarrow_i \circ \rightsquigarrow$.

Given a pushdown network, global configurations C_0 and C, and a number $k \geq 1$, the *context-bounded reachability problem* asks whether $C_0 \rightsquigarrow_k C$ holds, i.e. if C can be reached from C_0 in k context switches. This problem is decidable [18]. Context-bounded reachability has been successfully used in practice for bug finding. We show that underapproximations using bounded languages (Lem. 15) subsumes the technique of context-bounded reachability in the following sense.

Proposition 4 *Let $\mathcal{N}$ be a pushdown network, C_0, C global configurations of $\mathcal{N}$, and $(L_1, \ldots, L_n)$ CFLs over alphabet Σ such that $C_0 \rightarrow^* C$ iff $\cap_i L_i \neq \emptyset$. For each $k \geq 1$, there is an bounded expression B_k such that $C_0 \rightsquigarrow_k C$ only if $\bigcap_i^{(B_k)} L_i \neq \emptyset$. Also, $\bigcap_i^{(B_k)} L_i \neq \emptyset$ only if $C_0 \rightarrow^* C$.*

Proof Consider all sequences $C_0 \rightsquigarrow C_1 \cdots C_{k-1} \rightsquigarrow C_k$ of k switches. By the CFL encoding (Prop. 3) each of these sequences corresponds to a word in Σ^k. If $C_0 \rightsquigarrow_k C$, then there is a word $w \in \bigcap_i L_i$ and $w \in \Sigma^k$. Let $\Sigma = \{a_1, \ldots, a_n\}$, define B_k to be $(a_1^* \cdots a_n^*)^k$. We conclude from $w \in \Sigma^k$ and the definition of B_k that $w \in B_k$, hence that $\bigcap_i^{(B_k)} L_i \neq \emptyset$ since $w \in \bigcap_i L_i$. For the other direction we conclude from $\bigcap_i^{(B_k)} L_i \neq \emptyset$ that $\bigcap_i L_i \neq \emptyset$, hence that $C_0 \rightarrow^* C$. □

However, underapproximation using bounded languages can be more powerful than context-bounded reachability in the following sense. There is a family $\{(\mathcal{N}_k, C_{0k}, C_k)\}_{k \in \mathbb{N}}$ of pushdown network reachability problems such that $C_{0k} \rightsquigarrow_k C_k$ but $C_{0k} \not\rightsquigarrow_{k-1} C_k$ for each k, but there is a single bounded expression B such that $\bigcap_i^{(B)} L_{ik} \neq \emptyset$ for each k, where again $(L_{1k}, \ldots, L_{nk})$ are CFLs such that $C_{0k} \rightsquigarrow C_k$ iff $\cap_i L_{ik} \neq \emptyset$ (as in Prop. 3).

For clarity, we describe the family of pushdown networks as a family of two-threaded programs whose code is shown in Fig. 1. The programs in the family differs from each other by the value to which `k` is instantiated: `k` $= 0, 1, \ldots$ Each program has two threads. Thread one maintains a local counter `c` starting at `0`. Before each increment to `c`, thread one sets a global `bit`. Thread two resets `bit`. The target configuration C_k is given by the exit point of `p1`. We conclude from the program code that hitting the exit point of `p1` requires `c` $\geq$ `k` to hold. For every instance, C_k

is reachable, but it requires at least k context switches. Thus, there is no fixed context bound that is sufficient to check reachability for every instance in the family. In contrast, the bounded expression given by $\big((\texttt{bit} == \texttt{true}, 2) \cdot (\texttt{bit} == \texttt{false}, 1)\big)^*$ is sufficient to show reachability of the target for **every** instance in the family.

```
thread p1() {
  int c=0;
L:bit=true;
  if bit == false { ++c; }
  if c<k { goto L; }
}
```

```
thread p2() {
L1:bit = false;
   goto L1;
}
```

Fig. 1: The family of pushdown network with global `bit`.

We refer the interested reader to [6] for a treatment in depth of the underapproximation of the reachability problem using bounded expressions and in particular its complexity. Also see http://software.imdea.org/~pierreganty/vanocka.html for a prototype implementation.

6 Application II: Recursive Counter Machines

In verification, counting is a powerful abstraction mechanism. Often, counting abstractions are used to show decidability of the verification problem. Counting abstractions have been applied on a wide range of applications from parametrized systems specified as concurrent JAVA programs to cache coherence protocols (see [22]) and to programs manipulating complex data structures like lists (see for instance [5]). In those works, counting not only implies decidability, it also yields precise abstractions of the underlying verification problem. However, in those works recursion (or equivalently the call stack) is not part of the model. One option is to abstract the stack using additional counters, hence abstracting away the stack discipline. Because counting abstractions for the stack yields too much imprecision, we prefer to use a precise model of the call stack and perform an underapproximating analysis. This is what is defined below for a model of recursive programs that manipulate counters.

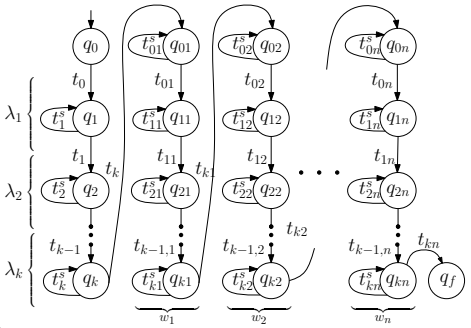
Counter Machine: Syntax and Semantics. An *n-dimensional counter machine* $M = (Q, T, \alpha, \beta, \{G_t\}_{t\in T})$ consists of the finite non-empty sets Q and T of locations and transitions, respectively; two mappings $\alpha\colon T \mapsto Q$ and $\beta\colon T \mapsto Q$, and a family $\{G_t\}_{t\in T}$ of semilinear (or *Presburger definable*) sets over $\mathbb{N}^{2n}$.

A *M-configuration* (q, x) consists of a location $q \in Q$ and a vector $x \in \mathbb{N}^n$; we define C_M as the set of M-configurations. For each transition $t \in T$, its semantics is given by the reachability relation $R_M(t)$ over C_M defined as $(q, x)R_M(t)(q', x')$ iff $q = \alpha(t)$, $q' = \beta(t)$, and $(x, x') \in G_t$. The reachability relation is naturally extended to words of T^* by defining $R_M(\varepsilon) = \{((q, x), (q, x)) \mid (q, x) \in C_M\}$ and $R_M(u\cdot v) = R_M(u) \circ R_M(v)$. Also, it extends to languages as expected. Finally, we write (M, D) for a counter machine M with an initial set $D \subseteq C_M$ of configurations. Note that semilinear sets carry over subsets of C_M using a bijection from Q to $\{1, \ldots, |Q|\}$.

- $\gamma' = \gamma + (k+1)n$
- $Q' = \{q_i\}_{0\le i\le k} \cup \{q_{ij}\}_{0\le i\le k}^{1\le j\le n} \cup \{q_f\}$
- $T' = \{t_0\} \cup \{t_i, t_i^s\}_{1\le i\le k} \cup \{t_{ij}, t_{ij}^s\}_{0\le i\le k}^{1\le j\le n}$
- α' and β' are given by the automaton
- Let $i \in \{0,\dots,k\}$ and $j \in \{1,\dots,n\}$

$$G_{t_i} = \begin{cases} G^+_{\lambda_{01}} \circ \dots \circ G^+_{\lambda_{0n}} & \text{if } i = 0 \\ \{(x,x) \in \mathbb{N}^{2\gamma'}\} & \text{else} \end{cases}$$

$G_{t_i^s} = G^+_{\lambda_{i1}} \circ \dots \circ G^+_{\lambda_{in}}$,
$G_{t_{ij}^s} = G(w_j^{p_{ij}}) \circ G^-_{\lambda_{ij}}$, and
$G_{t_{ij}} = \{\big((x,v),(x,v)\big) \mid v_{i*n+j} = 0\}$



Let $\# \in \{+,-\}$, $G^{\#}_{\lambda_{ij}} = \{\big((x,v),(x,v')\big) \in \mathbb{N}^{2\gamma'} \mid v' = v\#\mathbf{e}_{i*n+j}\}$.

Let $w \in T^*$, $G(w)$ is s.t. $G(\varepsilon) = \{(x,x) \in \mathbb{N}^{2\gamma'}\}$, $G(t) = \{\big((x,v),(x',v)\big) \in \mathbb{N}^{2\gamma'} \mid (x,x') \in G_t\}$, and $G(w_p \cdot w_s) = G(w_p) \circ G(w_s)$ if $w = \varepsilon$, t and $w_s \cdot w_p$, respectively.

Fig. 2: The γ'-dim counter machine $M' = (Q', T', \alpha', \beta', \{G_t\}_{t\in T'})$.

Computing the Reachable Configurations. Let $R \subseteq C_M \times C_M$ and $D \subseteq C_M$, we define the set of configurations $\mathrm{post}[R](D)$ as $\{(q,x) \mid \exists (q_0,x_0) \in D \wedge (q_0,x_0)R(q,x)\}$. Given a n-dim counter machine $M = (Q,T,\alpha,\beta,\{G_t\}_{t\in T})$, a semilinear set D of configurations and a CFL $L \subseteq T^*$ (encoding execution paths), we want to underapproximate $\mathrm{post}[R_M(L)](D)$: the set of M-configurations reachable from D along words of L. Our underapproximation computes the set $\mathrm{post}[R_M(L')](D)$ where L' is a Parikh-equivalent bounded subset L such that $L' = L \cap B$ where $B = w_1^* \cdots w_n^*$.

We will construct, given (M,D), L and B (we showed above how to effectively compute such a B), a pair (M',D') such that the set of M-configurations reachable from D along words of $L \cap B$ can be constructed from the set of M'-configurations reachable from D'. Without loss of generality, we assume M is such that Q is a singleton. (One can encode locations using counters.)

Let $M = (Q,T,\alpha,\beta,\{G_t\}_{t\in T})$ a γ-dim counter machine with $Q = \{q_f\}$ and $B = w_1^* \cdots w_n^*$ such that $\Pi(L \cap B) = \Pi(L)$. Let h be the homomorphism that maps some fresh letters $a_1,\dots,a_n$ to the words $w_1,\dots,w_n$, respectively. We compute the language $L'_A = h^{-1}(L \cap B) \cap a_1^* \cdots a_n^*$. Let $S = \Pi_{\{a_1,\dots,a_n\}}(L'_A)$, and note that S is a semilinear set. For clarity, we first consider a linear set H where $p_0 = (p_{01},\dots,p_{0n})$ denotes the constant and $\{p_i = (p_{i1},\dots,p_{in})\}_{i\in I\setminus\{0\}}$ the set of periods of H and $I = \{0,\dots,k\}$. Let $J = \{1,\dots,n\}$. In the following, for every pair of vectors $x = (x_1,\dots,x_r)$ and $y = (y_1,\dots,y_s)$, we denote by (x,y) the vector $(x_1,\dots,x_r,y_1,\dots,y_s)$. The machine M' is defined in Fig. 2.

Between q_0 and q_{01}, M' non-deterministically picks values for all the additional counters which we denote $\{\lambda_{ij}\}_{i\in I, j\in J}$. When M' fires t_k, we have for all $i \in I$ and $j,j' \in J$: $\lambda_{ij} = \lambda_{ij'}$ and $\lambda_{0i} = 1$. Below, for every $i \in I$, we denote by λ_i the common value of the counters $\{\lambda_{ij}\}_{j\in J}$. Then, M' simulates the behavior of M for the sequence of transitions given by $w_1^{p_{01}+\lambda_1 p_{11}+\dots+\lambda_k p_{k1}} \dots w_n^{p_{0n}+\lambda_1 p_{1n}+\dots+\lambda_k p_{kn}}$ the Parikh image of which is $p_0 + \sum_{i\in I} \lambda_i p_i$. Let us define the set D' of configurations of $C_{M'}$ as $\{(q_0,(x,v)) \mid (q_f,x) \in D \wedge v = 0^{(k+1)n}\}$.

A sufficient condition for the set of reachable configurations of M' starting from D' to be effectively computable is that for each t in $\{t_i^s\}_{i\in I\setminus\{0\}} \cup \{t_{ij}^s\}_{i\in I, j\in J}$ (i.e. the loops in Fig. 2), it holds that t^* is computable and Presburger definable. Given t the problem of deciding if t^* is Presburger definable is undecidable [1]. However, there exist some subclasses C of Presburger definable sets such that if $t \in C$ then t^* is Presburger definable and effectively computable, hence the set of reachable configurations of (M', D') can be computed by quantifier elimination in Presburger arithmetic. A known subclass is that of guarded command Presburger relations. An n-dimensional *guarded command* is given by the closure under composition of $\{(x, x') \in \mathbb{N}^{2n} \mid x' = x + \mathbf{e}_i\}$ (increment), $\{(x, x') \in \mathbb{N}^{2n} \mid x' = x - \mathbf{e}_i\}$ (decrement) and $\{(x, x) \in \mathbb{N}^{2n} \mid x = (x_1, \ldots, x_n) \wedge x_i = 0\}$ (0-test) for $1 \leq i \leq n$.

Other subclasses are given in [4,10]. Note that if for each $t \in T$ of M, G_t is given by a guarded command then so is each $G_{t'}$ for $t' \in T'$ of M' by definition.

Hence, we find that the set $\mathrm{post}[R_{M'}(T'^*)](D')$ of reachable configurations of (M', D') is Presburger definable, effectively computable and relates to $\mathrm{post}[R_M(L')](D)$ for the bounded language L' as follows.

Lemma 16 *Let $(q_f, x) \in C_M$. We have $(q_f, x) \in \mathrm{post}[R_M(L')](D)$ iff there exists $v \in \mathbb{N}^{(k+1)n}$ such that $(q_f, (x, v)) \in \mathrm{post}[R_{M'}(T'^*)](D')$.*

We can easily compute the intersection of the two semilinear sets S and $\{q_f\} \times \mathbb{N}^\gamma$ over $Q' \times \mathbb{N}^\gamma$, because of the way we have carried the notion of semilinear set over $Q' \times \mathbb{N}^\gamma$. We take a bijection η from Q' to $\{1, \ldots, |Q|\}$, so a configuration $(q, x) \in Q' \times \mathbb{N}^\gamma$ is represented by $(p_1, \ldots, p_{|Q|}, x)^T$ with

$$p_j = \begin{cases} 1 & \text{if } \eta(q) = j \\ 0 & \text{otherwise} \end{cases}$$

Hence, the intersection consists of all the vectors of S with the component of q_f equal to one and the others equal to zero. Lem. 14 shows that by iterating the construction we obtain a semi-algorithm for a context-free language.

Acknowledgement. We thank Javier Esparza for his help on the lower bounds on the bounded expressions which solves Pb. 1 for regular and context-free languages given at Lemma 8 and 13, respectively. We also thank Stefan Kiefer and Michael Luttenberger for helpful discussions.

References

1. Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Ph. Schnoebelen. Flat acceleration in symbolic model checking. In *ATVA '05: Proc. 3rd Int. Symp. on Automated Technology for Verification and Analysis*, volume 3707 of *LNCS*, pages 474–488. Springer, 2005.
2. Meera Blattner and Michel Latteux. Parikh-bounded languages. In *ICALP '81: Proc. of 8th Int. Colloquium on Automata, Languages and Programming*, volume 115 of *LNCS*, pages 316–323. Springer, 1981.
3. Ahmed Bouajjani, Javier Esparza, and Taissir Touili. A generic approach to the static analysis of concurrent programs with procedures. In *POPL '03: Proc. 30th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 62–73. ACM Press, 2003.
4. Marius Bozga, Codruta Gîrlea, and Radu Iosif. Iterating octagons. In *TACAS '09: Proc. 15th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 5505 of *LNCS*, pages 337–351. Springer, 2009.

5. Marius Bozga, Radu Iosif, Pierre Moro, Tomás Vojnar, Ahmed Bouajjani, and Peter Habermehl. Programs with lists are counter automata. In *CAV '06: Proc. 18th Int. Conf. on Computer Aided Verification*, volume 4144 of *LNCS*, pages 517–531. Springer, 2006.
6. Javier Esparza and Pierre Ganty. Complexity of pattern-based verification for multithreaded programs. In *POPL '11: Proc. 38th ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages*, pages 499–510. ACM Press, 2011.
7. Javier Esparza, Pierre Ganty, Stefan Kiefer, and Michael Luttenberger. Parikh's theorem: A simple and direct automaton construction. *CoRR*, abs/1006.3825, 2010.
8. Javier Esparza, Stefan Kiefer, and Michael Luttenberger. Newton's method for ω-continuous semirings. In *ICALP '08: Proc. 35th Int. Coll. on Automata, Languages and Programming*, volume 5126 of *LNCS*, pages 14–26. Springer, 2008. Invited paper.
9. Javier Esparza, Stefan Kiefer, and Michael Luttenberger. Newtonian program analysis. *Journal of the ACM*, 57(6):33:1–33:47, 2010.
10. Alain Finkel and Jérôme Leroux. How to compose presburger-accelerations: Applications to broadcast protocols. In *FSTTCS '02: Proc. 22nd Int. Conf. on Fondation of Software Technology and Theoretical Computer Science*, volume 2556 of *LNCS*, pages 145–156. Springer, 2002.
11. Pierre Ganty, Rupak Majumdar, and Benjamin Monmege. Bounded underapproximations. In *CAV'10: Proc. 20th Int. Conf. on Computer Aided Verification*, volume 6174 of *LNCS*, pages 600–614. Springer, 2010. See also CoRR report abs/0809.1236.
12. Seymour Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, Inc., New York, NY, USA, 1966.
13. Seymour Ginsburg and Edwin H. Spanier. Bounded algol-like languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964.
14. John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1st edition, April 1979.
15. Vineet Kahlon. Tractable dataflow analysis for concurrent programs via bounded languages. Patent WO/2009/094439, July 2009.
16. Michel Latteux and Jeannine Leguy. Une propriété de la famille GRE. In *FCT '79: Fundamentals of Computation Theory Proc. of the Conf. on Algebraic, Arithmetic, and Categorial Methods in Computation Theory*, pages 255–261. Akademie-Verlag, 1979.
17. Jérôme Leroux and Grégoire Sutre. On flatness for 2-dimensional vector addition systems with states. In *CONCUR '04: Proc. 15th Int. Conf. on Concurrency Theory*, volume 3170 of *LNCS*, pages 402–416. Springer, 2004.
18. Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *TACAS '05: Proc. 11th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.
19. G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM TOPLAS*, 22(2):416–430, 2000.
20. Dejvuth Suwimonteerabuth, Javier Esparza, and Stefan Schwoon. Symbolic context-bounded analysis of multithreaded java programs. In *SPIN '08: Proc. of 15th Int. Model Checking Software Workshop*, volume 5156 of *LNCS*, pages 270–287. Springer, 2008.
21. Anthony Widjaja To. *Model Checking Infinite-State Systems: Generic and Specific Approaches*. PhD thesis, School of Informatics, University of Edinburgh, August 2010.
22. Laurent Van Begin. *Efficient Verification of Counting Abstractions for Parametric systems*. PhD thesis, Université Libre de Bruxelles, Belgium, 2003.