Bounded Underapproximations

Pierre Ganty, Rupak Majumdar, Benjamin Monmege

To cite this version:

Pierre Ganty, Rupak Majumdar, Benjamin Monmege. Bounded Underapproximations. Proceedings of the 22nd International Conference on Computer Aided Verification (CAV'10), Jul 2010, Edinburgh, United Kingdom. pp.600-614, 10.1007/978-3-642-14295-6_52 . hal-00608175

Bounded Underapproximations[*]

Pierre Ganty[1] , Rupak Majumdar[2] , Benjamin Monmege[3]

[1] IMDEA Software, Spain [2] UCLA, USA [3] ENS Cachan, France

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 $w_1^* w_2^* \cdots w_k^*$ for some $w_1, \ldots, w_k \in \Sigma^*$. In particular bounded context-free languages have nice structural and decidability properties. Our proof proceeds in two parts. First, using Newton's iterations on the language semiring, we construct a context-free subset L_N of L that can be represented as a sequence of substitutions on a linear language and has the same Parikh image as L. Second, we inductively construct a Parikh-equivalent bounded context-free subset of L_N.
As an application of this result in model checking, we show how to underapproximate the reachable state space of multithreaded procedural programs. The bounded language constructed above provides a decidable underapproximation for the original problem. 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.

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 [16,2].

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

[*] This research was sponsored in part by the NSF grants CCF-0546170 and CCF-0702743, DARPA grant HR0011-09-1-0037, EU People/COFUND program 229599 AMAROUT, EU IST FET Project 231620 HATS, and Madrid Regional Government project S2009TIC-1465 PROMETIDOS.

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* [9,8]. A language L is *bounded* if there exist $k \in \mathbb{N}$ and finite words $w_1, w_2, \ldots, w_k$ such that L is a subset of the regular language $w_1^* \cdots w_k^*$. 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 [9]. 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 symbol 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 symbols that matches a string in L'.

The Parikh-boundedness property was first proved in [13,1], 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 two contributions in this paper.

Explicit construction of Parikh-equivalent bounded subsets. Our constructive proof has two parts. First, using Newton's iteration [5] on the semiring of languages, we construct, for a given context-free language L, a finite sequence of linear substitutions which denotes a Parikh-equivalent (but not necessarily bounded) subset of L. (A linear substitution maps a symbol to a language defined by a *linear* grammar, that is, a context-free grammar where each rule has at most one non-terminal on the right-hand side.) The Parikh equivalence follows from a convergence property of Newton's iteration on the related commutative semiring.

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.

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 $L_1' = L_1^0 \cap w_1^* \cdots w_k^*$ 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 $L_1^1 = L_1^0 \cap \overline{w_1^* \cdots w_k^*}$ 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 [15].

We omit proofs for space reasons. Detailed proofs, as well as one more application of our result, can be found in [7].

Related Work. Bounded languages were introduced and studied by Ginsburg and Spanier [9] (see also [8]). The existence of a bounded, Parikh-equivalent subset for a context-free language was shown in [1] using previous results on languages in the Greibach hierarchy [13]. 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 was independently shown in [6].

Bounded languages have been recently proposed by Kahlon for tractable reachability analysis of multithreaded programs [11]. 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 [9]). In contrast, our results are applicable even if the boundedness property does not hold.

For multithreaded reachability, *context-bounded reachability* [15,17] 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. We leave the empirical evaluation of our algorithms for future work.

2 Preliminaries

We assume the reader is familiar with the basics of language theory (see [10]). An alphabet Σ is a finite non-empty set of symbols. The concatenation $L \cdot L'$ of two languages $L, L' \subseteq \Sigma^*$ is defined using word concatenation as $L \cdot L' = \{l \cdot l' \mid l \in L \wedge l' \in L'\}$.

An *elementary bounded language* over Σ is a language of the form $w_1^* \cdots w_k^*$ for some fixed $w_1, \ldots, w_k \in \Sigma^*$.

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 $(x_1, \ldots, x_p), (y_1, \ldots, y_p) \in \mathbb{Z}^p$ $(x_1, \ldots, x_p) + (y_1, \ldots, y_p) = (x_1 + y_1, \ldots, x_p + y_p)$. 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 symbol $a_i \in \Sigma$, 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 $2^{\mathbb{N}^p}$. We also define the inverse of the Parikh image $\Pi_\Sigma^{-1} \colon 2^{\mathbb{N}^p} \to 2^{\Sigma^*}$ as follows: given a subset M of $\mathbb{N}^p$, $\Pi_\Sigma^{-1}(M)$ is the set $\{y \in \Sigma^* \mid \exists m \in M \colon m = \Pi_\Sigma(y)\}$. When it is clear from the context we generally omit the subscript in Π_Σ and Π_Σ^{-1}.

Context-free Languages. A *context-free grammar* G is a tuple $(\mathcal{X}, \Sigma, \delta)$ where $\mathcal{X}$ is a finite non-empty set of variables (non-terminal letters), Σ is an alphabet of terminal letters and $\delta \subseteq \mathcal{X} \times (\Sigma \cup \mathcal{X})^*$ 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 relation $u \Rightarrow v$, if there exists a production $(X, w) \in \delta$ and some words $y, z \in (\Sigma \cup \mathcal{X})^*$ such that $u = yXz$ and $v = ywz$. 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* (written CFL) if there exists a context-free grammar $G = (\mathcal{X}, \Sigma, \delta)$ and an initial variable $X \in \mathcal{X}$ such that is $L = L_X(G)$. A *linear grammar* G is a context-free grammar where each production is in $\mathcal{X} \times \Sigma^*(\mathcal{X} \cup \{\varepsilon\})\Sigma^*$. A language L is *linear* if $L = L_X(G)$ for some linear grammar G and initial variable X of G. A CFL L is *bounded* if it is a subset of some elementary bounded language.

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 CFL L, compute an elementary bounded language B such that $\Pi(L \cap B) = \Pi(L)$.

If we can compute such a 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 language L as follows: (1) we find an L' such that $L' \subseteq L$, $\Pi(L') = \Pi(L)$, and L' has a "simple" structure (Sect. 3) and (2) then show how to find an elementary bounded B with $\Pi(L' \cap B) = \Pi(L')$, assuming this structure (Sect. 4). Observe that if $L' \subseteq L$ and $\Pi(L) = \Pi(L')$, then for every elementary bounded B, we have $\Pi(L' \cap B) = \Pi(L')$ implies $\Pi(L \cap B) = \Pi(L)$ as well. So the solution B for L' in step (2) is a solution for L as well. Section 5 provides an application of the result for multithreaded program analysis and compares it with an existing technique.

3 A Parikh-Equivalent Representation

Our proof to compute the above L' relies on a fixpoint characterization of CFLs and their Parikh image. Accordingly, we introduce the necessary mathematical notions to define and study properties of those fixpoints.

Semiring. A *semiring* $\mathcal{S}$ is a tuple $\langle S, \oplus, \odot, \bar{0}, \bar{1}\rangle$, where S is a set with $\bar{0}, \bar{1} \in S$, $\langle S, \oplus, \bar{0}\rangle$ is a commutative monoid with neutral element $\bar{0}$, $\langle S, \odot, \bar{1}\rangle$ is a monoid with neutral element $\bar{1}$, $\bar{0}$ is an annihilator w.r.t. $\odot$, *i.e.* $\bar{0} \odot a = a \odot \bar{0} = \bar{0}$ for all $a \in S$, and $\odot$ distributes over $\oplus$, *i.e.* $a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$, and $(a \oplus b) \odot c = (a \odot c) \oplus (b \odot c)$. We call $\oplus$ the *combine* operation and $\odot$ the *extend* operation. The natural order relation $\sqsubseteq$ on a semiring $\mathcal{S}$ is defined by $a \sqsubseteq b \Leftrightarrow \exists d \in S \colon a \oplus d = b$. The semiring $\mathcal{S}$ is *naturally ordered* if $\sqsubseteq$ is a partial order on S. The semiring $\mathcal{S}$ is *commutative* if $a \odot b = b \odot a$ for all $a, b \in S$, *idempotent* if $a \oplus a = a$ for all $a \in S$, *complete* if it is naturally ordered and $\sqsubseteq$ is such that ω-chains $a_0 \sqsubseteq a_1 \sqsubseteq \cdots \sqsubseteq a_n \sqsubseteq \cdots$ have least upper bounds. Finally, the semiring $\mathcal{S}$ is *ω-continuous* if it is naturally ordered, complete and for all sequences $(a_i)_{i\in\mathbb{N}}$ with $a_i \in S$, $\sup\{\bigoplus_{i=0}^{n} a_i \mid n \in \mathbb{N}\} = \bigoplus_{i\in\mathbb{N}} a_i$. We define two semirings we shall use subsequently.

Language Semiring. Let $\mathcal{L} = \left\langle 2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\varepsilon\}\right\rangle$ denote the idempotent ω-continuous semiring of languages. The natural order on $\mathcal{L}$ is given by set inclusion (viz. $\subseteq$).

Parikh Semiring. The tuple $\mathcal{P} = \left\langle 2^{\mathbb{N}^p}, \cup, \dot{+}, \emptyset, \{\mathbf{0}\}\right\rangle$ is the idempotent ω-continuous commutative semiring of Parikh vectors. The natural order is again given by $\subseteq$.

Valuation, polynomial. In what follows, let $\mathcal{X}$ be a finite set of variables and $\mathcal{S} = \langle S, \oplus, \odot, \bar{0}, \bar{1}\rangle$ be an ω-continuous semiring.

A *valuation* $\boldsymbol{v}$ is a mapping $\mathcal{X} \to S$. We denote by $\mathcal{S}^{\mathcal{X}}$ the set of all valuations and by $\ddot{0}$ the valuation which maps each variable to $\bar{0}$. We define $\dot{\sqsubseteq} \subseteq \mathcal{S}^{\mathcal{X}} \times \mathcal{S}^{\mathcal{X}}$ as the order given by $\boldsymbol{v} \mathrel{\dot{\sqsubseteq}} \boldsymbol{v}'$ if and only if $\boldsymbol{v}(X) \sqsubseteq \boldsymbol{v}'(X)$ for every $X \in \mathcal{X}$. A *monomial* is a mapping $\mathcal{S}^{\mathcal{X}} \to S$ given by a finite expression $m = a_1 \odot X_1 \odot a_2 \cdots a_k \odot X_k \odot a_{k+1}$ where $k \geq 0$, $a_1, \ldots, a_{k+1} \in S$ and $X_1, \ldots, X_k \in \mathcal{X}$ such that $m(\boldsymbol{v}) = a_1 \odot \boldsymbol{v}(X_1) \odot a_2 \cdots a_k \odot \boldsymbol{v}(X_k) \odot a_{k+1}$ for $\boldsymbol{v} \in \mathcal{S}^{\mathcal{X}}$.
A *polynomial* is a finite combination of monomials: $f = m_1 \oplus \cdots \oplus m_k$ where $k \geq 0$ and $m_1, \ldots, m_k$ are monomials. The set of polynomials w.r.t. $\mathcal{S}$ and $\mathcal{X}$ will be denoted by $\mathcal{S}[\mathcal{X}]$. Finally, a *polynomial transformation* $\boldsymbol{F}$ is a mapping $\mathcal{S}^{\mathcal{X}} \to \mathcal{S}^{\mathcal{X}}$ described by the set $\{\boldsymbol{F}_X \in \mathcal{S}[\mathcal{X}] \mid X \in \mathcal{X}\}$ of polynomials: hence, for every vector $\boldsymbol{v} \in \mathcal{S}^{\mathcal{X}}$, $\boldsymbol{F}(\boldsymbol{v})$ is a valuation of each variable $X \in \mathcal{X}$ to $\boldsymbol{F}_X(\boldsymbol{v})$.

Least Fixpoint. Recall that a mapping $f \colon S \to S$ is *monotone* if $a \sqsubseteq b$ implies $f(a) \sqsubseteq f(b)$, and *continuous* if for any infinite chain $a_0, a_1, a_2, \ldots$ we have $\sup\{f(a_i)\} = f(\sup\{a_i\})$. The definition can be extended to mappings $\boldsymbol{F} \colon \mathcal{S}^{\mathcal{X}} \to \mathcal{S}^{\mathcal{X}}$ in the obvious way (using $\dot{\sqsubseteq}$). Then we may formulate the following proposition (cf. [12]).

Proposition 1. *Let $\boldsymbol{F}$ be a polynomial transformation. The mapping induced by $\boldsymbol{F}$ is monotone and continuous and $\boldsymbol{F}$ has a unique least fixpoint $\mu\boldsymbol{F}$.*

Fixpoints of polynomial transformations relates to CFLs as follows. Given a grammar $G = (\mathcal{X}, \Sigma, \delta)$, let $L(G)$ be the valuation which maps each variable $X \in \mathcal{X}$ to the language $L_X(G)$. We first characterize the valuation $L(G)$ as the least fixpoint of a polynomial transformation $\boldsymbol{F}$ defined as follows: each $\boldsymbol{F}_X$ of $\boldsymbol{F}$ is given by the combination of α's for $(X, \alpha) \in \delta$ where α is interpreted as a monomial on the semiring $\mathcal{L}$. From [3] we know that $L(G) = \mu\boldsymbol{F}$.

Example 1. Let $G = (\{X_0, X_1\}, \{a, b\}, \delta)$ where $\delta = \{(X_0 \to aX_1|a), (X_1 \to X_0b|aX_1bX_0)\}$. It defines the polynomial transformation $\boldsymbol{F}$ on $\mathcal{L}^{\mathcal{X}}$ such that $\boldsymbol{F}_{X_0} = a \cdot X_1 \cup a$ and $\boldsymbol{F}_{X_1} = X_0 \cdot b \cup a \cdot X_1 \cdot b \cdot X_0$, and $L(G)$ is the least fixpoint of $\boldsymbol{F}$ in the language semiring. □

3.1 Relating the Language and Parikh Semirings

Given a polynomial transformation $\boldsymbol{F}$, we now characterize the relationship between the least fixpoints $\mu\boldsymbol{F}$ taken over the language and the Parikh semiring, respectively. Either fixpoint is given by the limit of a sequence of *iterates* which is defined by Newton's iteration scheme [4,5]. Our characterization operates at the level of those iterates: we inductively relate the iterates of each iteration sequence (over the Parikh and language semirings). We use Newton's iteration instead of the usual Kleene's iteration sequence because Newton's iteration is guaranteed to converge on the Parikh semiring in a finite number of steps, a property that we shall exploit. Kleene's iteration sequence, on the other hand, may not converge. Due to lack of space, we refer the reader to [4,5] for the definition of Newton's iteration scheme.

We first extend the definition of the Parikh image to a valuation $\boldsymbol{v} \in \mathcal{L}^{\mathcal{X}}$ as the valuation of $\mathcal{P}^{\mathcal{X}}$ defined for each variable X by: $\Pi(\boldsymbol{v})(X) = \Pi(\boldsymbol{v}(X))$. Then, given $\boldsymbol{F}_{\mathcal{L}} \colon \mathcal{L}^{\mathcal{X}} \to \mathcal{L}^{\mathcal{X}}$, a polynomial transformation, we define a polynomial transformation $\boldsymbol{F}_{\mathcal{P}} \colon \mathcal{P}^{\mathcal{X}} \to \mathcal{P}^{\mathcal{X}}$ such that: for every $X \in \mathcal{X}$ we have $\boldsymbol{F}_{\mathcal{P}X} = \Pi \circ \boldsymbol{F}_{\mathcal{L}X} \circ \Pi^{-1}$. Lemma. 1 relates the iterates for $\mu\boldsymbol{F}_{\mathcal{L}}$ and $\mu\boldsymbol{F}_{\mathcal{P}}$ using the Parikh image mapping.

Lemma 1. *Let $(\boldsymbol{\nu}_i)_{i\in\mathbb{N}}$ and $(\boldsymbol{\kappa}_i)_{i\in\mathbb{N}}$ be Newton's iteration sequences associated with $\boldsymbol{F}_{\mathcal{L}}$ and $\boldsymbol{F}_{\mathcal{P}}$, respectively. For every $i \in \mathbb{N}$, we have $\Pi(\boldsymbol{\nu}_i) = \boldsymbol{\kappa}_i$.*

In [5], the authors show that Newton's iterates converges after a finite number of steps when defined over a commutative ω-continuous semiring. This shows, in our setting, that $(\boldsymbol{\kappa}_i)_{i\in\mathbb{N}}$ stabilizes after a finite number of steps.

Lemma 2. *Let $(\boldsymbol{\kappa}_i)_{i\in\mathbb{N}}$ be Newton's iteration sequence associated to $\boldsymbol{F}_{\mathcal{P}}$ and let n be the number of variables in $\mathcal{X}$. For every $k \geq n$, we have $\boldsymbol{\kappa}_k = \Pi(\mu\boldsymbol{F}_{\mathcal{L}})$. Hence, for every $k \geq n$, $\Pi(\boldsymbol{\nu}_k) = \Pi(\mu\boldsymbol{F}_{\mathcal{L}})$.*

We know Newton's iteration sequence $(\boldsymbol{\nu}_i)_{i\in\mathbb{N}}$, whose limit is $\mu\boldsymbol{F}_{\mathcal{L}}$, may not converge after a finite number of iterations. However, using Lem. 2, we know that the Parikh image of the iterates stabilizes after a finite number of steps. Precisely, if n is the number of variables in $\mathcal{X}$, then the language given by $\boldsymbol{\nu}_n$ is such that $\Pi(\boldsymbol{\nu}_n) = \Pi(L(G))$. Moreover because $(\boldsymbol{\nu}_i)_{i\in\mathbb{N}}$ is an ascending chain, for each variable $X \in \mathcal{X}$, we have that $\boldsymbol{\nu}_n(X)$ is a sublanguage of $L_X(G)$ such that $\Pi(\boldsymbol{\nu}_n(X)) = \Pi(L_X(G))$.

3.2 Representation of Iterates

We now show that Newton's iterates can be effectively represented as a combination of linear grammars and homomorphisms.

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 define the substitution $\sigma_{[a/b]} \colon \Sigma_1 \cup \{a\} \to \Sigma_1 \cup \{b\}$ which maps a to b and leaves all other symbols unchanged.
We show below that the iterates $(\boldsymbol{\nu}_k)_{k \leq n}$ have a "nice" representation.
k-fold composition. We effectively compute and represent each iterate as the valuation which maps each variable X to the language generated by a k-fold composition of a substitution. Since the substitution maps each symbol onto a language which is linear, it is effectively represented and manipulated as a linear grammar. To formally define the representation we need to introduce the following definitions.

Let $\tilde{G} = (\mathcal{X}, \Sigma \cup \{v_X \mid X \in \mathcal{X}\}, \tilde{\delta})$ be a linear grammar and let $k \in \mathbb{N}$, define $v_{\mathcal{X}}^k$ to be the set of symbols $\{v_X^k \mid X \in \mathcal{X}\}$. Given a language L on alphabet $\Sigma \cup \{v_X \mid X \in \mathcal{X}\}$, we define $L[v_{\mathcal{X}}^k]$ to be $\sigma_{[v_X / v_X^k]_{X \in \mathcal{X}}}(L)$ that is the language where each occurence of v_X is replaced by v_X^k.

For $k \in \{1, \ldots, n\}$, we define $\sigma_k \colon \Sigma \cup v_{\mathcal{X}}^k \to \Sigma \cup v_{\mathcal{X}}^{k-1}$ as the substitution which maps each v_X^k onto $L_X(\tilde{G})[v_{\mathcal{X}}^{k-1}]$ and leaves Σ unchanged. For $k = 0$ the substitution σ_0 maps each v_X^0 on $\boldsymbol{F}(\ddot{0})(X)$ and leaves Σ unchanged. σ_0 basically applies the terminal rules of the grammar. Let k, ℓ be such that $0 \leq k \leq \ell \leq n$ we define σ_k^ℓ to be $\sigma_k \circ \cdots \circ \sigma_\ell$. Hence, σ_0^k is such that: $(\Sigma \cup v_{\mathcal{X}}^k)^* \xrightarrow{\sigma_k} (\Sigma \cup v_{\mathcal{X}}^{k-1})^* \cdots (\Sigma \cup v_{\mathcal{X}}^1)^* \xrightarrow{\sigma_1} (\Sigma \cup v_{\mathcal{X}}^0)^* \xrightarrow{\sigma_0} \Sigma^*$.

Finally, the *k-fold composition* of a linear grammar $\tilde{G}$ and initial variable X is given by $\sigma_0^k(v_X^k)$. Lemma 3 relates k-fold compositions with $(\boldsymbol{\nu}_k)_{k \in \mathbb{N}}$. Moreover we characterize the complexity of computing $\tilde{G}$ given a polynomial transformation $\boldsymbol{F}$ the size of which is defined to be the number of bits needed to write the set $\{\boldsymbol{F}_X\}_{X \in \mathcal{X}}$ where each $\boldsymbol{F}_X$ is a string of symbols.

Lemma 3. *Given a polynomial transformation $\boldsymbol{F}$, there is a polynomial time algorithm to compute a linear grammar $\tilde{G}$ such that for every $k \geq 0$, every $X \in \mathcal{X}$ we have $\boldsymbol{\nu}_k(X) = \sigma_0^k(v_X^k)$.*

We refer the reader to our technical report [7] for the polynomial time construction of $\tilde{G}$ given $\boldsymbol{F}$. However, let us give a sample output of the construction.

Example 2. Let $\boldsymbol{F}$ be a polynomial transformation on $\mathcal{L}^{\mathcal{X}}$ where $\boldsymbol{F}_{X_0} = aX_1 \cup a$ and $\boldsymbol{F}_{X_1} = X_0 b \cup aX_1 bX_0$. The construction outputs $\tilde{G} = (\{X_0, X_1\}, \{a, b, v_{X_0}, v_{X_1}\}, \tilde{\delta})$ where $\tilde{\delta}$ is given by:

$$\begin{aligned} &X_0 \to aX_1 \mid av_{X_1} \mid a \\ &X_1 \to X_0 b \mid aX_1 bv_{X_0} \mid av_{X_1} bX_0 \mid v_{X_0} b \mid av_{X_1} bv_{X_0} \enspace . \end{aligned}$$

We have that $\boldsymbol{\nu}_1(X_0) = \sigma_0 \circ \sigma_1(v_{X_0}^1)$ and $\boldsymbol{\nu}_1(X_1) = \sigma_0 \circ \sigma_1(v_{X_1}^1)$.

Lem. 3 completes our goal to define a procedure to effectively compute and represent the iterates $(\boldsymbol{\nu}_k)_{k\in\mathbb{N}}$. This sequence is of interest since, given a CFL L and $\boldsymbol{\nu}_n$ the n-th iterate (where n equals the number of variables in the grammar of L so that $\Pi(\boldsymbol{\nu}_n) = \Pi(L)$), if B is a solution to Pb. 1 for the instance $\boldsymbol{\nu}_n$, B is also a solution to Pb. 1 for L. Notice that k-fold compositions relate to indexed grammars used to represent Newton iterates in [6].

4 Constructing a Parikh Equivalent Bounded Subset

We now show how, given a k-fold composition L', to compute an elementary bounded language B such that $\Pi(L' \cap B) = \Pi(B)$, that is we give an effective procedure to solve Pb. 1 for the instance L'. This will complete the solution to Pb. 1, hence the proof of Th. 1. In this section, we give an effective construction of elementary bounded languages that solve Pb. 1 first for regular languages, then for linear languages, and finally for a linear substitution.

First we need to introduce the notion of semilinear sets. A set $A \subseteq \mathbb{N}^n$ is a *linear set* if there exist $c \in \mathbb{N}^n$ and $p_1, \ldots, p_k \in \mathbb{N}^n$ such that $A = \left\{c + \sum_{i=1}^{k} \lambda_j p_j \mid \lambda_j \in \mathbb{N}\right\}$: c is called the constant of A and $p_1, \ldots, p_k$ the periods of A. A *semilinear set* S is a finite union of linear sets: $S = \bigcup_{i=1}^{\ell} A_i$ where each A_i is a linear set. Parikh's theorem (cf. [8]) shows that the Parikh image of every CFL is a semilinear set that is effectively computable.

Lemma 4. *Let L and B be respectively a CFL and an elementary bounded language over Σ such that $\Pi(L \cap B) = \Pi(L)$. There is an effectively computable elementary bounded language B' such that $\Pi(L^t \cap B') = \Pi(L^t)$ for all $t \in \mathbb{N}$.*

Proof. By Parikh's theorem, we know that $\Pi_\Sigma(L)$ is a computable semilinear set. Let us consider $u_1, \ldots, u_\ell \in L$ such that $\Pi_\Sigma(u_i) = c_i$ for $i \in \{1, \ldots, \ell\}$.

Let $B' = u_1^* \cdots u_\ell^* B^\ell$, we see that B' is an elementary bounded language. Let $t > 0$ be a natural integer. We have to prove that $\Pi(L^t) \subseteq \Pi(L^t \cap B')$.

case $\boldsymbol{t \leq \ell}$. By property of Π and $\Pi(L) = \Pi(L \cap B)$ we find that:

$$\begin{aligned}
\Pi(L^t) &= \Pi((L \cap B)^t) & \\
&\subseteq \Pi(L^t \cap B^t) & \text{monotonicity of } \Pi \\
&\subseteq \Pi(L^t \cap B^\ell) & B^t \subseteq B^\ell \text{ since } \varepsilon \in B \\
&\subseteq \Pi(L^t \cap B') & \text{def. of } B'
\end{aligned}$$

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

$$\Pi(w) = \sum_{i=1}^{\ell} \mu_i c_i + \sum_{i=1}^{\ell} \sum_{j=1}^{k_i} \lambda_{ij} p_{ij} \quad .$$

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 $\Pi(z_i) = c_i + \sum_{j=1}^{k_i} \lambda_{ij} p_{ij}$ 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 $w'' \in L^t$. □

Regular Languages. The construction of an elementary bounded language that solves Pb. 1 for a regular language L is known from [13] (see also [14], Lem. 4.1). The construction is carried out by induction on the structure of a regular expression for L. Assuming $L \neq \emptyset$, the base case (*i.e.* a symbol or ε) is trivially solved. Note that if $L = \emptyset$ then every elementary bounded language B is such that $\Pi(L \cap B) = \Pi(L) = \emptyset$.

The inductive case decomposes into three constructs. Let R_1 and R_2 be regular languages, and B_1 and B_2 the inductively constructed elementary bounded languages such that $\Pi(R_1 \cap B_1) = \Pi(R_1)$ and $\Pi(R_2 \cap B_2) = \Pi(R_2)$.

concatenation For the instance $R_1 \cdot R_2$, the elementary bounded language $B_1 \cdot B_2$ is such that $\Pi((R_1 \cdot R_2) \cap (B_1 \cdot B_2)) = \Pi(R_1 \cdot R_2)$;
union For $R_1 \cup R_2$, the elementary bounded language $B_1 \cdot B_2$ suffices;
Kleene star Let us consider R_1 and B_1, Lem. 4 shows how to effectively compute an elementary bounded language B' such that for every $t \in \mathbb{N}$, $\Pi(R_1^t \cap B') = \Pi(R_1^t)$. Let us prove that B' solves Pb. 1 for the instance R_1^*. In fact, if w is a word of R_1^*, there exists a $t \in \mathbb{N}$ such that $w \in R_1^t$. Then, we can find a word w' in $R_1^t \cap B'$ with the same Parikh image as w. This proves that $\Pi(R_1^*) \subseteq \Pi(R_1^* \cap B')$. The other inclusion holds trivially.

Proposition 2. *For every regular language R, there is an effective procedure to compute an elementary bounded language B such that $\Pi(R \cap B) = \Pi(R)$.*

Linear Languages. We now extend the previous construction to the case of linear languages. Recall that linear languages are used to represent the iterates $(\boldsymbol{\nu}_k)_{k \in \mathbb{N}}$. Lemma 5 gives a characterization of linear languages based on regular languages, homomorphism, and some additional structures.

Lemma 5. *(from [10]) For every linear language L over Σ, 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 over A such that $L = h(R\widetilde{A}^* \cap S)$ where $S = \{w\widetilde{w}^r \mid w \in A^*\}$ and w^r denotes the reverse image of the word w. Moreover there is an effective procedure to construct h, A, and R.*

The next result shows that an elementary bounded language that solves Pb. 1 can be effectively constructed for every linear language L that is given by h and R such that $L = h(R\widetilde{A}^* \cap S)$.

Proposition 3. *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, that is a procedure returning an elementary bounded B such that $\Pi(L \cap B) = \Pi(L)$.*

Linear languages with Substitutions. Our goal is to solve Pb. 1 for k-fold compositions, *i.e.* for languages of the form $\sigma_j^k(v_X^k)$. Prop. 3 gives an effective procedure for the case $j = k$ since $\sigma_k^k(v_X^k)$ is a linear language. Prop. 4 generalizes to the case $j < k$: given a solution to Pb. 1 for the instance $\sigma_{j+1}^k(v_X^k)$, there is an effective procedure for Pb. 1 for the instance $\sigma_j \circ \sigma_{j+1}^k(v_X^k) = \sigma_j^k(v_X^k)$.

Proposition 4. *Let*

1. *L be a CFL over Σ;*
2. *B an elementary bounded language 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 an elementary bounded and (ii) $\Pi(\sigma(a) \cap \tau(a)) = \Pi(\sigma(a))$.*

Then, there is an effective procedure that solves Pb. 1 for the instance $\sigma(L)$, by returning an elementary bounded language B' such that $\Pi(\sigma(L) \cap B') = \Pi(\sigma(L))$.

We use the above result inductively to solve Pb. 1 for k-fold composition as follows: fix L to be $\sigma_{j+1}^k(v_X^k)$, B to be the solution of Pb. 1 for the instance L, σ to be σ_j and τ a substitution which maps every v_X^j to the solution of Pb. 1 for the instance $\sigma_j(v_X^j)$. Then B' is the solution of Pb. 1 for the instance $\sigma_j^k(v_X^k)$.

Due to lack of space we refer to reader to [7] for details.

We thus have an effective construction of an elementary bounded language that solves Pb. 1 for k-fold composition, hence a constructive proof for Th. 1.

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 6. *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 elementary bounded 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 an elementary bounded language 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 [10]). □

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 to Multithreaded Procedural Programs

We now give an application of our construction that gives a semi-algorithm for checking reachability of multithreaded procedural programs [16,11,2]. A common programming model consists of multiple recursive threads communicating via shared memory. Formally, we model such systems as pushdown networks [17]. Let k be a positive integer, a *pushdown network* is a triple $\mathcal{N} = (G, \Gamma, (\Delta_i)_{1 \le i \le k})$ where G is a finite non-empty set of *globals*, Γ is the *stack alphabet*, and for each $1 \le i \le k$, 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 $\rightarrow_i$, as follows: $(g, \gamma\beta) \rightarrow_i (g', \alpha\beta)$ iff $\langle g, \gamma \rangle \hookrightarrow \langle g', \alpha \rangle$ in Δ_i and $\beta \in \Gamma^*$. The transition relation of $\mathcal{N}$, denoted $\rightarrow$, is defined as follows: $(g, \alpha_1, \ldots, \alpha_i, \ldots, \alpha_k) \rightarrow (g', \alpha_1, \ldots, \alpha'_i, \ldots, \alpha_k)$ iff $(g, \alpha_i) \rightarrow_i (g', \alpha'_i)$ for some $i \in \{1, \ldots, k\}$. By $\rightarrow_i^*$, $\rightarrow^*$, we denote the reflexive and transitive closure of these relations. Let C_0 and C be two global configurations, the *reachability problem* asks whether $C_0 \rightarrow^* C$ 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 $\perp$ that intuitively models an inactive state. The reduction also turns the globals into input symbols 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 $\perp$ to some $g \in G$) and exactly one acceptor will change from active to inactive by updating its global from some g to $\perp$. All the others acceptors will synchronize and stay inactive.

Given an instance of the reachability problem, that is a pushdown network $(G, \varGamma, (\varDelta_i)_{1\leq i\leq k})$ 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 $\big\{(G', \varGamma, \varSigma, \varDelta'_i, c_0^i)\big\}_{1\leq i\leq k}$, where:

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

Proposition 5. *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 CFLs $(L_1, \ldots, L_k)$ (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 [10], so is the reachability problem. We will now compare two underapproximation techniques for the reachability problem: context-bounded switches [15] and bounded languages, which we first detail below.

Let $L_1, \ldots, L_k$ be context-free languages, 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 elementary bounded language B, we define the *intersection modulo B* of the languages $\{L_i\}_i$ as $\bigcap_i^{(B)} L_i = \big(\bigcap_i L_i\big) \cap B$. 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 7. *Given an elementary bounded language $B = w_1^* \cdots w_n^*$ 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 $\varSigma$. Let h be the homomorphism that maps the symbols $a_1, \ldots, a_n$ to the words $w_1, \ldots, w_n$, respectively. We show that $\bigcap_{1\leq i\leq k} \varPi_A\big(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\big) \neq \emptyset$ 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 $(t_1, \ldots, t_n) \in \varPi_A(h^{-1}(w) \cap a_1^* \cdots a_n^*)$, hence that $(t_1, \ldots, t_n) \in \varPi_A(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*)$ for every $1 \leq i \leq k$ by above and finally that $(t_1, \ldots, t_n) \in \bigcap_{1\leq i\leq k} \varPi_A\big(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\big)$.

For the other implication, consider $(t_1, \ldots, t_n)$ a vector of $\bigcap_{1\leq i\leq k} \varPi_A\big(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\big)$ 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 $(t_1, \ldots, t_n) \in \mathit{\Pi}_A\big(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\big)$, there exists a word $w' \in a_1^* \cdots a_n^*$ such that $\mathit{\Pi}_A(w') = (t_1, \ldots, t_n)$ and $h(w') \in L_i \cap B$. We conclude from $\mathit{\Pi}_A(w') = (t_1, \ldots, t_n)$, 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 [10]. Moreover, given a CFL, we can compute its Parikh image which is a semilinear set. Finally, we can compute the semilinear sets $\mathit{\Pi}_A\big(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\big)$ and the emptiness of the intersection of semilinear sets is decidable [8]. □

While Lem. 7 shows decidability for every elementary bounded 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 elementary bounded language $B_i = {w_1^{(i)}}^* \cdots {w_{n_i}^{(i)}}^*$ such that $\mathit{\Pi}(L_i \cap B_i) = \mathit{\Pi}(L_i)$. 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. 6 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.

Comparison with Context-Bounded Reachability. A well-studied underapproximation for multithreaded reachability is given by context-bounded reachability [15]. 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 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 [15]. Context-bounded reachability has been successfully used in practice for bug finding. We show that underapproximations using bounded languages (Lem. 7) subsumes the technique of context-bounded reachability in the following sense.

Proposition 6. *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 elementary bounded language 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 or fewer switches. By the CFL encoding (Prop. 5) each of these sequences corresponds to a word in $\mathit{\Sigma}^k$. If $C_0 \rightsquigarrow_k C$, then there is a word $w \in \bigcap_i L_i$ and $w \in \mathit{\Sigma}^k$. Define B_k to be $w_1^* \cdots w_m^*$ where $w_1, \ldots, w_m$ is an enumeration of all strings in $\mathit{\Sigma}^k$. We conclude from $w \in \mathit{\Sigma}^k$ 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 $C_0 \to^* C$. □

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 elementary bounded 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. 5).

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: $\texttt{k} = 0, 1, \ldots$ 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 $\texttt{c} \geq \texttt{k}$ 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 elementary bounded language 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`.

Acknowledgment. We thank Ahmed Bouajjani for pointing that the bounded languages approach subsumes the context-bounded switches one.

References

1. M. Blattner and M. Latteux. Parikh-bounded languages. In *ICALP '81*, LNCS 115, 316–323. Springer, 1981.
2. A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. In *POPL '03*, 62–73. ACM Press, 2003.
3. N. Chomsky and M.P. Schützenberger. The algebraic theory of context-free languages. *Comp. Programming and Formal Systems*, 118–161. North-Holland, 1963.
4. J. Esparza, S. Kiefer, and M. Luttenberger. An extension of Newton's method to ω-continuous semirings. In *DLT '07*, LNCS 4588, 157–168. Springer, 2007.
5. J. Esparza, S. Kiefer, and M. Luttenberger. On fixed point equations over commutative semirings. In *STACS '07*, LNCS 4393, 296–307. Springer, 2007.

6. J. Esparza, S. Kiefer, and M. Luttenberger. Newton's Method for ω-Continuous Semirings. In *ICALP '08*, LNCS 5126, 14–26. Springer, 2008.
7. P. Ganty, R. Majumdar, and B. Monmege. Bounded Underapproximations. *CoRR*, abs/0809.1236, 2009.
8. S. Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, NY, USA, 1966.
9. S. Ginsburg and E. Spanier. Bounded ALGOL-like languages. *Trans. Amer. Math. Soc.*, 113:333–368, 1964.
10. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation (1st Edition)*. Addison Wesley, 1979.
11. V. Kahlon. Tractable analysis for concurrent programs via bounded languages. Unpublished.
12. W. Kuich. Semirings and formal power series: their relevance to formal languages and automata. In *Handbook of formal languages, Vol. 1*, 609–677. Springer, 1997.
13. M. Latteux and J. Leguy. Une propriété de la famille GRE. In *Fundamentals of Computation Theory*, 255–261, 1979. Akademie-Verlag.
14. J. Leroux and G. Sutre. On flatness for 2-dimensional vector addition systems with states. In *CONCUR '04*, LNCS 3170, 402–416. Springer, 2004.
15. S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS '05*, LNCS 3440, 93–107. Springer, 2005.
16. G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM TOPLAS*, 22(2):416–430, 2000.
17. D. Suwimonteerabuth, J. Esparza and S. Schwoon. Symbolic Context-Bounded Analysis of Multithreaded Java Programs. In *SPIN '08*, LNCS 5156, 270–287. Springer, 2008.