Bounded Underapproximations

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

[1] UCLA, USA rupak@cs.ucla.edu
[2] IMDEA Software, Spain pierre.ganty@imdea.org
[3] ENS Cachan, France bmonmege@dptinfo.ens-cachan.fr

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 subsets of 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.

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. Incidentally, we show that our approach subsumes 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 [19,4]. 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* [12,11]. 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 [12]. 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 [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 falls into two parts. First, using Newton's iteration [9] 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.

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 multi-threaded programs with procedures. Using the above construction, we obtain a semi-algorithm for reachability analysis of multi-threaded 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 [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. $L' \subseteq w_1^* \cdots w_k^*$); and (3) rerun the analysis using for $L \cap \overline{w_1^* \cdots w_k^*}$ 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 have been recently proposed by Kahlon for tractable reachability analysis of multithreaded programs [14]. 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 [12]). In contrast, our results are applicable even if the boundedness property does not hold.

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 step. We leave the empirical evaluation of our algorithms for future work.

2 Preliminaries

An alphabet is a finite non-empty set of symbols. We use the letter Σ to denote some alphabet. We assume the reader is familiar with the basics of language theory (see [13]). 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 $w_1, \ldots, w_k \in \Sigma^*$.

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 $(x_1, \ldots, x_p), (y_1, \ldots, y_p) \in \mathbb{Z}^p$ $(x_1, \ldots, x_p) + (y_1, \ldots, y_p) = (x_1 + y_1, \ldots, x_p + y_p)$. Given $\lambda \in \mathbb{N}$ and $x \in \mathbb{Z}^p$, we write λx as the λ-times sum $x + \cdots + x$.

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. We also define, using vector addition, the operation $\dot{+}$ on sets of Parikh vectors as follows: given $Z, Z' \subseteq \mathbb{N}^p$, let $Z \dot{+} Z' = \{z + z' \mid z \in Z \wedge z' \in Z'\}$. Thus, Π_Σ 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}.

The following lemma gives the properties of Π and Π^{-1} we need in the sequel.

Lemma 1. *For every $M \in 2^{\mathbb{N}^p}$ we have $\Pi \circ \Pi^{-1}(M) = M$.*
Let $\phi = \Pi^{-1} \circ \Pi$, for every $X, Y \subseteq \Sigma^*$ we have:

additivity of Π $\Pi(X \cup Y) = \Pi(X) \cup \Pi(Y)$*;*
monotonicity of ϕ $X \subseteq Y$ *implies* $\phi(X) \subseteq \phi(Y)$*;*
extensivity of ϕ $X \subseteq \phi(X)$*;*
idempotency of ϕ $\phi \circ \phi(X) = \phi(X)$*;*
structure-semipreservation of ϕ $\phi(X) \cdot \phi(Y) \subseteq \phi(X \cdot Y)$*;*
preservation of Π $\Pi(X \cdot Y) = \Pi(X) \dot{+} \Pi(Y)$*.*

Proof. For the first statement we first observe that Π is a surjective function, for each vector of $\mathbb{N}^p$ there is a word that is mapped to that vector. Next,

$$\begin{aligned} \Pi \circ \Pi^{-1}(M) &= \Pi(\{y \mid \exists m \in M \colon m = \Pi(y)\}) && \text{def. of } \Pi^{-1} \\ &= \{\Pi(y) \mid \exists m \in M \colon m = \Pi(y)\} && \text{def. of } \Pi \\ &= M && \text{surjectivity of } \Pi \end{aligned}$$

For the additivity, the monotonicity, the extensivity and the idempotency properties, we simply show the equivalence given below. Hence the properties immediately follows by property of Galois connection (we refer the reader to [7] for detailed proofs). We show that for every $L \in 2^{\Sigma^*}, M \in 2^{\mathbb{N}^\Sigma}$ we have : $\Pi(L) \subseteq M$ iff $L \subseteq \Pi^{-1}(M)$.

$$\begin{aligned} & L \subseteq \Pi^{-1}(M) \\ & \text{iff } L \subseteq \{y \mid \exists m \in M \colon m = \Pi(y)\} && \text{def. of } \Pi^{-1} \\ & \text{iff } \forall \ell \in L \, \exists m \in M \colon m = \Pi(\ell) \\ & \text{iff } \forall h \in \Pi(L) \, \exists m \in M \colon m = h && \text{def. of } \Pi \\ & \text{iff } \Pi(L) \subseteq M \end{aligned}$$

For structure semipreservation, we prove that $\phi(x) \cdot \phi(y) \subseteq \phi(x \cdot y)$ for $x, y \in \Sigma^*$ as follows:

$$\begin{aligned}
\phi(x) \cdot \phi(y) &= \Pi^{-1} \circ \Pi(x) \cdot \Pi^{-1} \circ \Pi(y) \\
&= \{x' \mid \Pi(x') = \Pi(x)\} \cdot \{y' \mid \Pi(y') = \Pi(y)\} && \text{def. of } \Pi^{-1} \\
&= \{x' \cdot y' \mid \Pi(x') = \Pi(x) \wedge \Pi(y') = \Pi(y)\} \\
&\subseteq \{x' \cdot y' \mid \Pi(x') + \Pi(y') = \Pi(x) + \Pi(y)\} \\
&= \{x' \cdot y' \mid \Pi(x' \cdot y') = \Pi(x \cdot y)\} && \text{def. of } \Pi \\
&= \Pi^{-1} \circ \Pi(x \cdot y) \\
&= \phi(x \cdot y)
\end{aligned}$$

The result generalizes to languages in a natural way. Finally, the preservation of Π is proved as follows:

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

□

Context-free Languages. A *context-free grammar* 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.

Moreover observe that for every $L' \subseteq L$ such that $\Pi(L) = \Pi(L')$, and for every elementary bounded B, if B is a solution to Pb. 1 on instance L' then B is also a solution for Pb. 1 on instance L. We solve Pb. 1 for instance L as follows: (1) we provide an effective representation of L' where $L' \subseteq L$, $\Pi(L') = \Pi(L)$, and L' has a "simple" structure (Sect. 3) and (2) then solve Pb. 1 for languages with this structure (Sect. 4). Section 5 provides applications of the result for program analysis problems.

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 s.t. 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, partial order, linear form, monomial and polynomial (transformation). 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 $\bar{0}$. The operations $\oplus$, $\odot$ are naturally extended to valuations. The *partial order* $\sqsubseteq$ on $\mathcal{S}$ can be lifted to a partial order on valuations, to this end we stack a point above $\sqsubseteq$ (viz. $\dot{\sqsubseteq}$) to denote the pointwise inclusion, given by $\boldsymbol{v} \dot{\sqsubseteq} \boldsymbol{v}'$ if and only if $\boldsymbol{v}(X) \sqsubseteq \boldsymbol{v}'(X)$ for every $X \in \mathcal{X}$.

A *linear form* is a mapping $l \colon \mathcal{S}^{\mathcal{X}} \to S$ satisfying $l(\boldsymbol{v} \oplus \boldsymbol{v}') = l(\boldsymbol{v}) \oplus l(\boldsymbol{v}')$ for

every $\boldsymbol{v}, \boldsymbol{v}' \in \mathcal{S}^{\mathcal{X}}$ and $l(\ddot{0}) = \bar{0}$.
A *monomial* is a mapping $\mathcal{S}^{\mathcal{X}} \to S$ described by a finite expression $m = a_1 \odot X_1 \odot a_2 \dots a_k \odot X_k \odot a_{k+1}$ where $k \geq 0$, $a_1, \dots, a_{k+1} \in S$ and $X_1, \dots X_k \in \mathcal{X}$ such that $m(\boldsymbol{v}) = a_1 \odot \boldsymbol{v}(X_1) \odot a_2 \dots a_k \odot \boldsymbol{v}(X_k) \odot a_{k+1}$ for $\boldsymbol{v} \in \mathcal{S}^{\mathcal{X}}$. The empty monomial is given by an empty expression coincides with $\bar{1}$.
A *polynomial* is a finite combination of monomials : $f = m_1 \oplus \dots \oplus m_k$ where $k \geq 0$ and $m_1, \dots, m_k$ are monomials. The set of polynomials w.r.t. $\mathcal{S}$ and $\mathcal{X}$ will be denoted by $\mathcal{S}[\mathcal{X}]$. The empty polynomial is given by an empty combination of monomials and coincides with $\bar{0}$.
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 valuation $\boldsymbol{v} \in \mathcal{S}^{\mathcal{X}}$, $\boldsymbol{F}(\boldsymbol{v})$ is a valuation that assigns each variable $X \in \mathcal{X}$ to $\boldsymbol{F}_X(\boldsymbol{v})$.

Differential. For every $X \in \mathcal{X}$, let $\mathrm{d}X$ denote the linear form defined by $\mathrm{d}X(\boldsymbol{v}) = \boldsymbol{v}(X)$ for every $\boldsymbol{v} \in \mathcal{S}^{\mathcal{X}}$: $\mathrm{d}X$ is the *dual variable* associated with the variable X. Let $\mathrm{d}\mathcal{X}$ denote the set $\{\mathrm{d}X \mid X \in \mathcal{X}\}$ of dual variables.

Let $f \in \mathcal{S}[\mathcal{X}]$ be a polynomial and let $X \in \mathcal{X}$ be a variable. The *differential w.r.t. X* of f is the mapping $D_X f \colon \mathcal{S}^{\mathcal{X}} \to \mathcal{S}^{\mathcal{X}} \to S$ that assigns to every valuation $\boldsymbol{v}$ the linear form $D_X f|_{\boldsymbol{v}}$ defined by induction as follows:

$$D_X f|_{\boldsymbol{v}} = \begin{cases} \bar{0} & \text{if } f \in S \text{ or } f \in \mathcal{X} \setminus \{X\} \\ \mathrm{d}X & \text{if } f = X \\ D_X g|_{\boldsymbol{v}} \odot h(\boldsymbol{v}) \oplus g(\boldsymbol{v}) \odot D_X h|_{\boldsymbol{v}} & \text{if } f = g \odot h \\ D_X g|_{\boldsymbol{v}} \oplus D_X h|_{\boldsymbol{v}} & \text{if } f = g \oplus h \end{cases} .$$

Then, the *differential* of f is defined by

$$Df = \bigoplus_{X \in \mathcal{X}} D_X f \ .$$

Consequently, the linear form $Df|_{\boldsymbol{v}}$ is a polynomial of the following form:

$$(a_1 \odot \mathrm{d}X_1 \odot a'_1) \oplus \dots \oplus (a_k \odot \mathrm{d}X_k \odot a'_k)$$

where each $a_i, a'_i \in S$ and $X_i \in \mathcal{X}$. We extend the definition of differential on polynomial transformation. Hence, $D\boldsymbol{F} : \mathcal{S}^{\mathcal{X}} \to \mathcal{S}^{\mathcal{X}} \to \mathcal{S}^{\mathcal{X}}$ is defined for every $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{S}^{\mathcal{X}}$ and every variable X as follows:

$$(D\boldsymbol{F}|_{\boldsymbol{v}}(\boldsymbol{w}))(X) = D\boldsymbol{F}_X|_{\boldsymbol{v}}(\boldsymbol{w}) \ .$$

Least Fixpoint. Recall that a mapping $f \colon \mathcal{S} \to \mathcal{S}$ is monotone if $a \sqsubseteq b$ implies $f(a) \sqsubseteq f(b)$, and continuous if for any infinite chain $a_0, a_1, a_2, \dots$ we have $\sup\{f(a_i)\} = f(\sup\{a_i\})$. The definition can be extended to mappings $\boldsymbol{F} \colon \mathcal{S}^{\mathcal{X}} \to \mathcal{S}^{\mathcal{X}}$ from valuations to valuations in the obvious way (component-wise). Then we may formulate the following proposition (cf. [15]).

Proposition 1. *Let $\boldsymbol{F}$ be a polynomial transformation. The mapping induced by $\boldsymbol{F}$ is monotone and continuous. Hence, by Kleene's theorem, $\boldsymbol{F}$ has a unique least fixpoint $\mu\boldsymbol{F}$. Further, $\mu\boldsymbol{F}$ is the supremum (w.r.t. $\dot{\sqsubseteq}$) of the Kleene's iteration sequence given by $\boldsymbol{\eta}_0 = \boldsymbol{F}(\ddot{0})$, and $\boldsymbol{\eta}_{i+1} = \boldsymbol{F}(\boldsymbol{\eta}_i)$.*

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 [6] 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 $\mathcal{L}^{\mathcal{X}}$ mapping X_0 to $a \cdot X_1 \cup a$ and X_1 to $X_0 \cdot b \cup a \cdot X_1 \cdot b \cdot X_0$, and $L(G)$ is the least fixpoint of $\boldsymbol{F}$ in the language semiring. □

We now recall the iteration sequence of [8,9] whose limit is the least fixpoint of $\boldsymbol{F}$. In some cases, the iteration sequence converges after a finite number of iterates while the Kleene iteration sequence does not.

Newton's Iteration Sequence. Given a polynomial transformation $\boldsymbol{F}$ on a ω-continuous semiring $\mathcal{S}$, *Newton's iteration sequence* is given by the following sequence:

$$\boldsymbol{\mu}_0 = \boldsymbol{F}(\ddot{0}) \qquad \text{and} \qquad \boldsymbol{\mu}_{i+1} = D\boldsymbol{F}|^*_{\boldsymbol{\mu}_i}(\boldsymbol{F}(\boldsymbol{\mu}_i))$$

the limit of which coincides with $\mu\boldsymbol{F}$ (see [9,8] for further details).

3.1 Relating the Semirings

We naturally 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: $\Pi(\boldsymbol{v})(X) = \Pi(\boldsymbol{v}(X))$. The following lemma relates polynomial transformations on $\mathcal{L}$ and $\mathcal{P}$.

Lemma 2. *Let $f_{\mathcal{L}} \in \mathcal{L}[\mathcal{X}]$, that is a polynomial over the semiring $\mathcal{L}$ and variables $\mathcal{X}$. Define $f_{\mathcal{P}} = \Pi \circ f_{\mathcal{L}} \circ \Pi^{-1}$, we have $f_{\mathcal{P}} \in \mathcal{P}[\mathcal{X}]$.*

Proof. By induction on the structure of $f_{\mathcal{L}}$. The polynomial $f_{\mathcal{L}}$ is given by $m_1 \cup \cdots \cup m_\ell$. Hence,

$$\begin{aligned} \Pi \circ f_{\mathcal{L}} \circ \Pi^{-1} &= \Pi \circ (m_1 \cup \cdots \cup m_\ell) \circ \Pi^{-1} \\ &= \Pi \circ m_1 \circ \Pi^{-1} \cup \cdots \cup \Pi \circ m_\ell \circ \Pi^{-1} \end{aligned}$$

where each m_i is of the form $a_1 \cdot X_1 \cdot a_2 \ldots a_k \cdot X_k \cdot a_{k+1}$ with $a_1, \ldots, a_{k+1} \subseteq \Sigma^*$, $X_1, \ldots, X_k \in \mathcal{X}$. Let m be a monomial, we have:

$$\begin{aligned} \Pi \circ m \circ \Pi^{-1} &= \Pi \circ a_1 \cdot X_1 \cdot a_2 \ldots a_k \cdot X_k \cdot a_{k+1} \circ \Pi^{-1} \\ &= \Pi(a_1) \dot{+} X_1 \dot{+} \Pi(a_2) \ldots \Pi(a_k) \dot{+} X_k \dot{+} \Pi(a_{k+1}) \quad \text{id. of } \Pi \circ \Pi^{-1}\text{, preser. of } \Pi \end{aligned}$$

We now prove a commutativity results on polynomials and the Parikh mapping.

Lemma 3. *Let $f_\mathcal{L} \in \mathcal{L}[\mathcal{X}]$, for every valuation $\boldsymbol{v} \in \mathcal{L}^\mathcal{X}$, we have:*

$$\Pi(f_\mathcal{L}(\boldsymbol{v})) = f_\mathcal{P}(\Pi(\boldsymbol{v})) \ .$$

Proof. First, the definition of $f_\mathcal{P}$ shows that for every $\boldsymbol{v} \in \mathcal{L}^\mathcal{X}$:

$\Pi \circ f_\mathcal{L}(\boldsymbol{v}) = f_\mathcal{P} \circ \Pi(\boldsymbol{v})$
iff $\Pi \circ f_\mathcal{L}(\boldsymbol{v}) = \Pi \circ f_\mathcal{L} \circ \Pi^{-1} \circ \Pi(\boldsymbol{v})$
only if $\Pi^{-1} \circ \Pi \circ f_\mathcal{L}(\boldsymbol{v}) = \Pi^{-1} \circ \Pi \circ f_\mathcal{L} \circ \Pi^{-1} \circ \Pi(\boldsymbol{v})$ — appl. of Π^{-1}

Moreover,

$\Pi^{-1} \circ \Pi \circ f_\mathcal{L}(\boldsymbol{v}) = \Pi^{-1} \circ \Pi \circ f_\mathcal{L} \circ \Pi^{-1} \circ \Pi(\boldsymbol{v})$
only if $\Pi \circ \Pi^{-1} \circ \Pi \circ f_\mathcal{L}(\boldsymbol{v}) = \Pi \circ \Pi^{-1} \circ \Pi \circ f_\mathcal{L} \circ \Pi^{-1} \circ \Pi(\boldsymbol{v})$ — appl. of Π
only if $\Pi \circ f_\mathcal{L}(\boldsymbol{v}) = \Pi \circ f_\mathcal{L} \circ \Pi^{-1} \circ \Pi(\boldsymbol{v})$ — identity of $\Pi \circ \Pi^{-1}$
iff $\Pi \circ f_\mathcal{L}(\boldsymbol{v}) = f_\mathcal{P} \circ \Pi(\boldsymbol{v})$ — def. of $f_\mathcal{P}$

Hence,

$\Pi \circ f_\mathcal{L}(\boldsymbol{v}) = f_\mathcal{P} \circ \Pi(\boldsymbol{v})$ iff $\Pi^{-1} \circ \Pi \circ f_\mathcal{L}(\boldsymbol{v}) = \Pi^{-1} \circ \Pi \circ f_\mathcal{L} \circ \Pi^{-1} \circ \Pi(\boldsymbol{v})$

Let $\phi = \Pi^{-1} \circ \Pi$, we will thus show that for every $\boldsymbol{v} \in \mathcal{L}^\mathcal{X}$

$\phi \circ f_\mathcal{L}(\boldsymbol{v}) = \phi \circ f_\mathcal{L} \circ \phi(\boldsymbol{v})$

The inclusion $\phi \circ f_\mathcal{L}(\boldsymbol{v}) \subseteq \phi \circ f_\mathcal{L} \circ \phi(\boldsymbol{v})$ is clear since $\boldsymbol{v} \dot{\sqsubseteq} \phi(\boldsymbol{v})$, every function occuring in the above expression is monotone and the functional composition preserves monotonicity. For the reverse inclusion, we first show that for every $\boldsymbol{w} \dot{\sqsubseteq} \phi(\boldsymbol{v})$ we have $f_\mathcal{L}(\boldsymbol{w}) \subseteq \phi \circ f_\mathcal{L}(\boldsymbol{v})$. That is $\forall x \in f_\mathcal{L}(\boldsymbol{w})\colon x \in \phi \circ f_\mathcal{L}(\boldsymbol{v})$. $f_\mathcal{L} \in \mathcal{L}[\mathcal{X}]$ shows that $x \in m(\boldsymbol{w})$ for some monomial $m = a_1 \cdot X_1 \cdot a_2 \ldots a_k \cdot X_k \cdot a_{k+1}$, that is $x \in a_1 \cdot \boldsymbol{w}(X_1) \cdot a_2 \ldots a_k \cdot \boldsymbol{w}(X_k) \cdot a_{k+1}$. We have,

$$\begin{aligned}
\phi \circ f_\mathcal{L}(\boldsymbol{v}) &\supseteq \phi(a_1 \cdot \boldsymbol{v}(X_1) \cdot a_2 \ldots a_k \cdot \boldsymbol{v}(X_k) \cdot a_{k+1}) \\
&\supseteq \phi(a_1) \cdot \phi(\boldsymbol{v}(X_1)) \cdot \phi(a_2) \ldots \phi(a_k) \cdot \phi(\boldsymbol{v}(X_k)) \cdot \phi(a_{k+1}) && \text{struct. semipreserv.} \\
&\supseteq a_1 \cdot \phi(\boldsymbol{v}(X_1)) \cdot a_2 \ldots a_k \cdot \phi(\boldsymbol{v}(X_k)) \cdot a_{k+1} && \text{extensivity of } \phi \\
&\supseteq a_1 \cdot \boldsymbol{w}(X_1) \cdot a_2 \ldots a_k \cdot \boldsymbol{w}(X_k) \cdot a_{k+1} && \boldsymbol{w} \dot{\sqsubseteq} \phi(\boldsymbol{v}) \\
&\ni x && \text{def. of } x
\end{aligned}$$

The following reasoning concludes the proof:

$f_\mathcal{L} \circ \phi(\boldsymbol{v}) \subseteq \phi \circ f_\mathcal{L}(\boldsymbol{v})$ — from above with $\boldsymbol{w} = \phi(\boldsymbol{v})$
only if $\phi \circ f_\mathcal{L} \circ \phi(\boldsymbol{v}) \subseteq \phi \circ \phi \circ f_\mathcal{L}(\boldsymbol{v})$ — monotonicity of ϕ
iff $\phi \circ f_\mathcal{L} \circ \phi(\boldsymbol{v}) \subseteq \phi \circ f_\mathcal{L}(\boldsymbol{v})$ — idempotency of ϕ

Here follows a commutativity result between the differential and the Parikh image.

Lemma 4. *For every $f_{\mathcal{L}} \in \mathcal{L}[\mathcal{X}]$, every valuation $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{L}^{\mathcal{X}}$, every $X \in \mathcal{X}$ we have:*

$$\Pi(D_X f_{\mathcal{L}}|_{\boldsymbol{v}}(\boldsymbol{w})) = D_X f_{\mathcal{P}}|_{\Pi(\boldsymbol{v})}(\Pi(\boldsymbol{w})) \ .$$

Proof. First it is important to note that Lemma 2 shows that $f_{\mathcal{P}}$ and $f_{\mathcal{L}}$ are of the same form. Then the proof falls into four parts according to the definition of the differential w.r.t. X.

$f_{\mathcal{L}} \in 2^{\Sigma^*}$ or $f_{\mathcal{L}} \in \mathcal{X} \setminus \{X\}$. In this case, we find that $D_X f_{\mathcal{L}}|_{\boldsymbol{v}}(\boldsymbol{w}) = \emptyset$, hence that $\Pi(D_X f_{\mathcal{L}}|_{\boldsymbol{v}}(\boldsymbol{w})) = \emptyset$. Since $f_{\mathcal{P}}$ is of the above form, we find that $D_X f_{\mathcal{P}}|_{\Pi(\boldsymbol{v})}(\Pi(\boldsymbol{w})) = \emptyset$.

$f_{\mathcal{L}} = X$. So $f_{\mathcal{P}} = X$.

$$\begin{aligned}
\Pi(D_X X|_{\boldsymbol{v}}(\boldsymbol{w})) &= \Pi(\mathrm{d}X(\boldsymbol{w})) && \text{def. of diff} \\
&= \Pi(\boldsymbol{w}(X)) && \text{def. of } \mathrm{d}X \\
&= \Pi(\boldsymbol{w})(X) && \text{def. of } \Pi \\
&= \mathrm{d}X(\Pi(\boldsymbol{w})) && \text{def. of } \mathrm{d}X \\
&= D_X X|_{\Pi(\boldsymbol{v})}(\Pi(\boldsymbol{w})) && \text{def. of diff}
\end{aligned}$$

$f_{\mathcal{L}} = g_{\mathcal{L}} \cdot h_{\mathcal{L}}$ So $f_{\mathcal{P}}$ is of the form $g_{\mathcal{P}} \dot{+} h_{\mathcal{P}}$. The induction hypothesis shows the rest.
$f_{\mathcal{L}} = \bigcup_{i \in I} f_i$ this case is treated similarly.

This result generalizes to the complete differential :

$$\Pi(Df_{\mathcal{L}}|_{\boldsymbol{v}}(\boldsymbol{w})) = Df_{\mathcal{P}}|_{\Pi(\boldsymbol{v})}(\Pi(\boldsymbol{w})) \ .$$

We note that the previous results also generalizes to polynomial transformation in a natural way. In the next subsection, thanks to the previous results, we show that Newton's iteration sequence on the language semiring reaches a stable Parikh image after a finite number of steps. This result is crucial in order to achieve the goal of this section: compute a sublanguage L' of L such that $\Pi(L) = \Pi(L')$.

3.2 Convergence of Newton's Iteration

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 [8,9]. 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.

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 ${\boldsymbol{F}_{\mathcal{P}}}_X = \Pi \circ {\boldsymbol{F}_{\mathcal{L}}}_X \circ \Pi^{-1}$. The next Lemma relates the iterates for $\mu\boldsymbol{F}_{\mathcal{L}}$ and $\mu\boldsymbol{F}_{\mathcal{P}}$ using the Parikh image mapping.

Lemma 5. *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 $\Pi(\boldsymbol{\nu}_i) = \boldsymbol{\kappa}_i$.*

Proof. **base case.** $(i = 0)$ This case is trivially solved using part (2) of Lem. 2.
inductive case. $(i + 1)$

$$
\begin{aligned}
\Pi(\boldsymbol{\nu}_{i+1}) &= \Pi(D\boldsymbol{F}_{\mathcal{L}}|^{*}_{\boldsymbol{\nu}_i}(\boldsymbol{F}_{\mathcal{L}}(\boldsymbol{\nu}_i))) && \\
&= \Pi(\bigcup_{j\in\mathbb{N}} D\boldsymbol{F}_{\mathcal{L}}|^{j}_{\boldsymbol{\nu}_i}(\boldsymbol{F}_{\mathcal{L}}(\boldsymbol{\nu}_i))) && \text{def. of }^{*} \\
&= \bigcup_{j\in\mathbb{N}} \Pi(D\boldsymbol{F}_{\mathcal{L}}|^{j}_{\boldsymbol{\nu}_i}(\boldsymbol{F}_{\mathcal{L}}(\boldsymbol{\nu}_i))) && \text{additivity of } \Pi \\
&= \bigcup_{j\in\mathbb{N}} \Pi\big(D\boldsymbol{F}_{\mathcal{L}}|_{\boldsymbol{\nu}_i}(D\boldsymbol{F}_{\mathcal{L}}|^{j-1}_{\boldsymbol{\nu}_i}(\boldsymbol{F}_{\mathcal{L}}(\boldsymbol{\nu}_i)))\big) && \text{funct. comp.} \\
&= \bigcup_{j\in\mathbb{N}} D\boldsymbol{F}_{\mathcal{P}}|_{\Pi(\boldsymbol{\nu}_i)}\big(\Pi(D\boldsymbol{F}_{\mathcal{L}}|^{j-1}_{\boldsymbol{\nu}_i}(\boldsymbol{F}_{\mathcal{L}}(\boldsymbol{\nu}_i)))\big) && \text{Lem. 4} \\
&= \bigcup_{j\in\mathbb{N}} D\boldsymbol{F}_{\mathcal{P}}|^{j}_{\Pi(\boldsymbol{\nu}_i)}(\Pi(\boldsymbol{F}_{\mathcal{L}}(\boldsymbol{\nu}_i))) && j-1\times \text{ Lem. 4} \\
&= \bigcup_{j\in\mathbb{N}} D\boldsymbol{F}_{\mathcal{P}}|^{j}_{\Pi(\boldsymbol{\nu}_i)}(\boldsymbol{F}_{\mathcal{P}}(\Pi(\boldsymbol{\nu}_i))) && \text{Lem. 2} \\
&= \bigcup_{j\in\mathbb{N}} D\boldsymbol{F}_{\mathcal{P}}|^{j}_{\boldsymbol{\kappa}_i}(\boldsymbol{F}_{\mathcal{P}}(\boldsymbol{\kappa}_i)) && \text{ind. hyp.} \\
&= D\boldsymbol{F}_{\mathcal{P}}|^{*}_{\boldsymbol{\kappa}_i}(\boldsymbol{F}_{\mathcal{P}}(\boldsymbol{\kappa}_i)) && \\
&= \boldsymbol{\kappa}_{i+1} &&
\end{aligned}
$$

□

In [9], 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 6. *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 $k \geq n$, $\Pi(\boldsymbol{\nu}_k) = \Pi(\mu\boldsymbol{F}_{\mathcal{L}})$.*

Proof.

$$
\begin{aligned}
&\boldsymbol{\kappa}_i = \Pi(\boldsymbol{\nu}_i) && \text{for each } i \in \mathbb{N} \text{ by Lem. 5}\\
&\Rightarrow \bigcup_{i\in\mathbb{N}} \boldsymbol{\kappa}_i = \bigcup_{i\in\mathbb{N}} \Pi(\boldsymbol{\nu}_i) && \\
&\Leftrightarrow \mu\boldsymbol{F}_{\mathcal{P}} = \bigcup_{i\in\mathbb{N}} \Pi(\boldsymbol{\nu}_i) && \omega\text{-continuity of } \mathcal{P}\\
&\Leftrightarrow \mu\boldsymbol{F}_{\mathcal{P}} = \Pi(\bigcup_{i\in\mathbb{N}} \boldsymbol{\nu}_i) && \text{additivity of } \Pi\\
&\Leftrightarrow \mu\boldsymbol{F}_{\mathcal{P}} = \Pi(\mu\boldsymbol{F}_{\mathcal{L}}) && \omega\text{-continuity of } \mathcal{L}\\
&\Rightarrow \boldsymbol{\kappa}_k = \Pi(\mu\boldsymbol{F}_{\mathcal{L}}) && \text{for every } k \geq n \text{ by Th. 6 of [9]}
\end{aligned}
$$

Transitivity of the equality shows the remaining result. □

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. 6, 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 $\Pi(\boldsymbol{\nu}_n(X)) = \Pi(L_X(G))$.

3.3 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.

Let us leave for a moment Newton's iteration sequence and turn to our initial problem as stated in Pb. 1. Let L be a context-free language, our goal is to compute a sublanguage L' such that $\Pi(L) = \Pi(L')$ (then we solve Pb. 1 on instance L' instead of L because it is equivalent). Below we give an effective procedure to compute such a L' based on the previously defined iteration sequences and the convergence results.

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 $\boldsymbol{F}$ which is defined using G 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 now interpreted as a monomial on the semiring $\mathcal{L}$.

Example 2. Let $G = (\{X_0, X_1\}, \{a, b\}, \delta)$ be the context-free grammar with the production:

$$X_0 \to aX_1 \mid a$$
$$X_1 \to X_0 b \mid aX_1 bX_0$$

It defines the following polynomial transformation on $\mathcal{L}^{\mathcal{X}}$:

$$\boldsymbol{F} = \begin{pmatrix} aX_1 \cup a \\ X_0 b \cup aX_1 bX_0 \end{pmatrix}$$

It is well known that $L(G) = \mu\boldsymbol{F}$ (see for instance [8]). To evaluate $\mu\boldsymbol{F}$ one can evaluate Newton's iteration sequence $\{\boldsymbol{\nu}_i\}_{i\geq 0}$ for $\boldsymbol{F}$. However, a transfinite number of iterates may be needed before reaching $\mu\boldsymbol{F}$. We now observe that, by the result of Lem. 6, if we consider the iteration sequence $(\boldsymbol{\nu}_k)_{k\leq n}$ up to iterate n where n equals to 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 $\{\boldsymbol{\nu}_i\}_{i\geq 0}$ is an ascending chain we find that: for each variable $X_0 \in \mathcal{X}$, $\boldsymbol{\nu}_n(X_0)$ is a sublanguage of $L_{X_0}(G)$ such that $\Pi(\boldsymbol{\nu}_n(X_0)) = \Pi(L_{X_0}(G))$.

We now explain how to turn this theoretical result into an effective procedure. Our first step is to define an effective representation for the iterates $\{\boldsymbol{\nu}_k\}_{k\leq n}$. Our definition is based on the one that was informally introduced in Example 3.1, part (2) of [8]. To this end, we start by defining how to represent the differential $D\boldsymbol{F}|_{\boldsymbol{v}}^{*}(\boldsymbol{F}(\boldsymbol{v}))$ used in the definition of Newton's iteration sequence as the language generated by a linear grammar.

We define $\boldsymbol{v}$ to be the valuation which maps each variable $X \in \mathcal{X}$ to v_X where v_X is a new symbol w.r.t. Σ. We first observe that $D\boldsymbol{F}|_{\boldsymbol{v}}$ is a polynomial transformation on the set of dual variables $\mathrm{d}\mathcal{X}$ such that the linear form associated to X is a polynomial of the form:

$$(a_1 \cdot \mathrm{d}X_1 \cdot a_1') \cup \cdots \cup (a_k \cdot \mathrm{d}X_k \cdot a_k')$$

where each $a_i, a_i' \in (\Sigma \cup \{v_Y \mid Y \in \mathcal{X}\})^*$ and $X_i \in \mathcal{X}$. Moreover, $\boldsymbol{F}_X$ is a sum of monomials $m_1, \ldots, m_\ell$. Hence, we define the linear grammar $\tilde{G} = (\mathcal{X}, \Sigma \cup \{v_X \mid X \in \mathcal{X}\}, \tilde{\delta})$. For the variable X, the set of productions $\tilde{\delta}$ is:

$$X \to a_1 X_1 a_1' \mid \ldots \mid a_k X_k a_k'$$
$$X \to m_1(\boldsymbol{v}) \mid \ldots \mid m_\ell(\boldsymbol{v})$$

We are able to prove that:

Lemma 7. *Let $\boldsymbol{v}$ be the valuation which maps each variable $X \in \mathcal{X}$ to v_X:*

$$L(\tilde{G}) = D\boldsymbol{F}|_{\boldsymbol{v}}^{*}(\boldsymbol{F}(\boldsymbol{v})) \ .$$

Proof. We show by induction the following equivalence. Let $X \in \mathcal{X}$, $w \in \Sigma \cup \{v_Y \mid Y \in \mathcal{X}\}^*$:

$$X \Rightarrow^{k+1} w \text{ iff } w \in D\boldsymbol{F}|^k_{\boldsymbol{v}}(\boldsymbol{F}(\boldsymbol{v}))(X) \ .$$

Base case. ($k = 0$) In this case, the following equivalence has to be established:

$$\begin{aligned}
&X \Rightarrow w \\
&\text{iff } w \in L_X(X \to m_1(\boldsymbol{v}) \mid \cdots \mid m_\ell(\boldsymbol{v})) \\
&\text{iff } w \in m_1(\boldsymbol{v}) \cup \cdots \cup m_\ell(\boldsymbol{v}) && \text{the monomials for } \boldsymbol{F}_X(\boldsymbol{v}) \\
&\text{iff } w \in \boldsymbol{F}_X(\boldsymbol{v}) \\
&\text{iff } w \in \boldsymbol{F}(\boldsymbol{v})(X)
\end{aligned}$$

Inductive case. ($k + 1$)

$$\begin{aligned}
&w \in D\boldsymbol{F}|^{k+1}_{\boldsymbol{v}}(\boldsymbol{F}(\boldsymbol{v}))(X) \\
&\text{iff } w \in D\boldsymbol{F}|_{\boldsymbol{v}}(D\boldsymbol{F}|^k_{\boldsymbol{v}}(\boldsymbol{F}(\boldsymbol{v})))(X) && \text{funct. comp.} \\
&\text{iff } w \in D\boldsymbol{F}_X|_{\boldsymbol{v}}(D\boldsymbol{F}|^k_{\boldsymbol{v}}(\boldsymbol{F}(\boldsymbol{v}))) \\
&\text{iff } w \in (a_1 \cdot dX_1 \cdot a'_1) \cup \ldots \cup (a_k \cdot dX_k \cdot a'_k)(D\boldsymbol{F}|^k_{\boldsymbol{v}}(\boldsymbol{F}(\boldsymbol{v}))) && \text{def. of diff.} \\
&\text{iff } \exists i \colon w \in (a_i \cdot dX_i \cdot a'_i)(D\boldsymbol{F}|^k_{\boldsymbol{v}}(\boldsymbol{F}(\boldsymbol{v}))) \\
&\text{iff } \exists i \, \exists w' \in D\boldsymbol{F}|^k_{\boldsymbol{v}}(\boldsymbol{F}(\boldsymbol{v}))(X_i) \colon w = a_i \cdot w' \cdot a'_i \\
&\text{iff } \exists i \, \exists w' \colon X \to a_i X_i a'_i \in \tilde{\delta} \wedge X_i \Rightarrow^{k+1} w' \wedge a_i w' a'_i = w \\
&\text{iff } X \Rightarrow^{k+2} w
\end{aligned}$$

Example 3. (cont'd from the previous example) The differential of $\boldsymbol{F}$ is given by:

$$D\boldsymbol{F}|_{\boldsymbol{v}} = \begin{pmatrix} a \, \mathrm{d}X_1 \\ \mathrm{d}X_0 \, b \cup a \, \mathrm{d}X_1 \, b \, \boldsymbol{v}(X_0) \cup a \, \boldsymbol{v}(X_1) \, b \, \mathrm{d}X_0 \end{pmatrix}$$

The grammar $\tilde{G}$ is given by $(\{X_0, X_1\}, \{a, b, v_{X_0}, v_{X_1}\}, \tilde{\delta})$ where $\tilde{\delta}$ is such that:

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

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 $v^k_{\mathcal{X}}$ to be the set of symbols $\left\{v^k_X \mid X \in \mathcal{X}\right\}$. Given a language L on alphabet $\Sigma \cup$

$\{v_X \mid X \in \mathcal{X}\}$, we define $L[v_{\mathcal{X}}^k]$ to be $\sigma_{[v_X/v_X^k]_{X\in\mathcal{X}}}(L)$. This definition naturally extends to valuations.

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. 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 can be characterized as follows: $(\Sigma \cup v_{\mathcal{X}}^k)^* \xrightarrow{\sigma_k} (\Sigma \cup v_{\mathcal{X}}^{k-1})^* \cdots (\Sigma \cup v_{\mathcal{X}}^1)^* \xrightarrow{\sigma_1} (\Sigma \cup v_{\mathcal{X}}^0)^* \xrightarrow{\sigma_0} \Sigma^*$.

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 8 relates k-fold compositions with $(\boldsymbol{\nu}_k)_{k\in\mathbb{N}}$.

Lemma 8. *There exists an effectively computable linear grammar $\tilde{G}$ such that for every $k \geq 0$, every $X \in \mathcal{X}$ we have $\boldsymbol{\nu}_k(X) = \sigma_0^k(v_X^k)$.*

Proof. By induction on k.
Base case. ($k=0$) Definition of the iteration sequence shows that $\boldsymbol{\nu}_0(X) = \boldsymbol{F}(\ddot{0})(X)$ which in turn equals $\sigma_0(v_X^0)$ by definition.
Inductive case. ($k+1$) First, let us define $\sigma_{\boldsymbol{\nu}_k}$ to be the substitution which maps v_X onto $\boldsymbol{\nu}_k(X)$. Hence we have

$$\begin{aligned}\boldsymbol{\nu}_{k+1} &= D\boldsymbol{F}|_{\boldsymbol{\nu}_k}^*(\boldsymbol{F}(\boldsymbol{\nu}_k)) && \text{def. of } \boldsymbol{\nu}_{k+1}\\ &= \sigma_{\boldsymbol{\nu}_k}(L(\tilde{G})) && \text{Lem. 7, def. of } \sigma_{\boldsymbol{\nu}_k}\end{aligned}$$

The above definition shows that $\sigma_{\boldsymbol{\nu}_k}(v_X) = \boldsymbol{\nu}_k(X)$, hence that $\sigma_{\boldsymbol{\nu}_k}(v_X) = \sigma_0^k(v_X^k)$ by induction hypothesis. Hence

$$\begin{aligned}\boldsymbol{\nu}_{k+1}(X) &= \sigma_{\boldsymbol{\nu}_k}(L_X(\tilde{G}))\\ &= \sigma_{\boldsymbol{\nu}_k} \circ \sigma_{[v_Y^k/v_Y]}(\sigma^{k+1}(v_X^{k+1})) && \text{def. of } \sigma^{k+1}\\ &= \sigma_0^k \circ \sigma^{k+1}(v_X^{k+1}) && \text{by above}\\ &= \sigma_0^{k+1}(v_X^{k+1})\end{aligned}$$

□

Lem. 8 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.

Let us conclude this section on a complexity note. Below we show that the linear grammar $\tilde{G}$ given in Lem. 8 is computable in polynomial time in the size of $\boldsymbol{F}$ which is to be defined. To start with we define the size of a monomial which is intuitively the length of the "string" that defines the monomial. Formally, let m be a monomial its size denoted, *sizeof*(m), is given by 0 if m is the empty monomial; 1 if $m \in 2^{\Sigma^*}$ or $m \in \mathcal{X} \cup \mathrm{d}\mathcal{X}$ and by *sizeof*(m_1) + *sizeof*(m_2) if

$m = m_1 \cdot m_2$. The above definition naturally extends to polynomials by summing the sizes of the monomials. The empty polynomial has size zero.

In what follows we show that the derivative of a monomial as a polynomial of some form.

Lemma 9. *Let $m = b_1 \cdots b_k$ be a monomial where each $b_i \in 2^{\Sigma^*} \cup \mathcal{X}$, let $X \in \mathcal{X}$ and $\boldsymbol{v} \in \mathcal{L}^{\mathcal{X}}$. We have $D_X m|_{\boldsymbol{v}}$ coincide with the polynomial given by:*

1. *apply the inductive definition of a derivative on m which is given by $D_X m|_{\boldsymbol{v}} = D_X(b_1 \cdots b_{k-1})|_{\boldsymbol{v}} \cdot \boldsymbol{v}(b_k) \cup (b_1 \cdots b_{k-1}) \cdot a$ where $a = \mathrm{d}X$ if $b_k = X$ and $\emptyset$ otherwise. Above we abusively wrote $\boldsymbol{v}(b_k)$ which in fact denotes $\boldsymbol{v}(b_k)$ if $b_k \in \mathcal{X}$ and b_k otherwise.*
2. *turn the result into a polynomial, that is a finite combination of monomials, by distributing $\cdot$ over $\cup$ (in the inductive part of point (1)).*

In the rest of this section, we identify $D_X m|_{\boldsymbol{v}}$ with the polynomial of Lem. 9.

Lemma 10. *Let $m = b_1 \cdots b_k$, and $D_X m|_{\boldsymbol{v}} = \bigcup_{i \in \{1,\ldots,I\}} m_i$. We have $\mathit{sizeof}(m_i) \leq k$ and $I \leq k$.*

Proof. $k = 1$. $D_X m|_{\boldsymbol{v}} = \begin{cases} \mathrm{d}X & \text{if } m = X \\ \emptyset & \text{else} \end{cases}$ which concludes the case.

$k > 1$. Induction hypothesis shows that $D_X(b_1 \cdots b_{k-1})|_{\boldsymbol{v}} = \bigcup_{j \in \{1,\ldots,J\}} m'_j$ where $\mathit{sizeof}(m'_j) \leq k-1$ and $J \leq k-1$. Hence by Lem. 9, the distributivity of $\cdot$ over $\cup$, the size of $\boldsymbol{v}(b_k)$ bounded by 1 show that $\mathit{sizeof}(m_i) \leq k$ and $I = J + 1 \leq k$. □

Corollary 1. *The size of $D_X(b_1 \cdots b_k)|_{\boldsymbol{v}}$ is bounded by k^2 (where k is the size of the monomial).*

Let us extend this reasoning to polynomials and polynomial transformations. Let $f = \bigcup_{i \in \{1,\ldots,I\}} m_i$. The definition of differential shows that $D_X f|_{\boldsymbol{v}} = \bigcup_{1 \leq i \leq I} D_X m_i|_{\boldsymbol{v}}$ where each $D_X m_i|_{\boldsymbol{v}}$ is a polynomial as shown by Lem. 9. Let $n = \mathit{sizeof}(f)$, we have that $\mathit{sizeof}(D_X f|_{\boldsymbol{v}})$ is bounded by n^3. This result follows from Coro. 1 and the fact that $I \leq n$.

Let us now extend our result to the differential in each variable. The definition of derivative shows that $Df|_{\boldsymbol{v}} = \bigcup_{X \in \mathcal{X}} D_X f|_{\boldsymbol{v}}$ the definition of which is given above. Let $n = \max(|\mathcal{X}|, \mathit{sizeof}(f))$, we find that $\mathit{sizeof}(Df|_{\boldsymbol{v}})$ is bounded by n^4.

Finally we extend the result to polynomial transformation using the equality $(D\boldsymbol{F}|_{\boldsymbol{v}})(X) = D\boldsymbol{F}_X|_{\boldsymbol{v}}$. Let us now characterize the time complexity of the algorithm that computes for $D\boldsymbol{F}|_{\boldsymbol{v}}$.

Corollary 2. *Let $\boldsymbol{F}$ and $\boldsymbol{v}$ be respectively a polynomial transformation and a valuation over $\mathcal{X}$. Define $S = \{\boldsymbol{v}(X)\}_{X \in \mathcal{X}} \cup \{a \in \mathcal{L} \mid \exists X \in \mathcal{X} \colon a \text{ occurrs in } \boldsymbol{F}_X\}$. The size of S is given by the sum of the size of each of its member. The size of $a \in 2^{\Sigma^*}$ is given by the sum of the length of each $w \in a$. If S is of finite size then $D\boldsymbol{F}|_{\boldsymbol{v}}$ is computable in time polynomial in the size of each $\boldsymbol{F}_X$, $\mathcal{X}$ and the size of S.*

Remark that we could generalize and drop the finiteness requirement for S. For example, regular languages or context-free languages would be admissible candidates for each element of S because they come with a finite representation and decision procedure for the tests/operations we need to compute the differential.

We showed above how to compute $\tilde{G}$ from $D\boldsymbol{F}|_{\boldsymbol{v}}$ and $\boldsymbol{F}$. So we conclude that $\tilde{G}$ is computable in time polynomial in the size of each $\boldsymbol{F}_X$, $\mathcal{X}$ and the size of S.

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. Prior to that we recall a result from [16] that is needed in subsequent proofs. However the proof given here is original.

At this point, 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 $A = \left\{c + \sum_{i=1}^{k} \lambda_i p_i \mid \lambda_i \in \mathbb{N}\right\}$: 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: $S = \bigcup_{j=1}^{\ell} A_j$ where each A_j is a linear set. Parikh's theorem (cf. [11]) shows that the Parikh image of every CFL is a semilinear set that is effectively computable.

Lemma 11. *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 $\Pi(L^t) \subseteq \Pi(L^t \cap B')$.

$\boldsymbol{t \leq \ell}$ We conclude from the preservation of Π and the hypothesis $\Pi(L) = \Pi(L \cap B)$ that

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

$\boldsymbol{t > \ell}$ 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 $i \in \{1, \ldots, \ell\}$: $\alpha_i = \begin{cases} \mu_i - 1 & \text{if } \mu_i > 0 \\ 0 & \text{otherwise.} \end{cases}$.

For each $i \in \{1, \ldots, \ell\}$, 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 $w' = u_1^{\alpha_1} \ldots u_\ell^{\alpha_\ell} z_1 \ldots z_\ell$. Clearly, $\Pi(w') = \Pi(w)$ and $w' \in u_1^* \cdots u_\ell^* (L \cup \{\varepsilon\})^\ell$. For each $i \in \{1, \ldots, \ell\}$, $\Pi(L \cap B) = \Pi(L)$ shows that there is $z_i' \in (L \cap B) \cup \{\varepsilon\}$ such that $\Pi(z_i') = \Pi(z_i)$. Let $w'' = u_1^{\alpha_1} \ldots u_\ell^{\alpha_\ell} z_1' \ldots z_\ell'$. 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 [16] (see also [17], 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 elembounded bounded language B is such that $\Pi(L \cap B) = \Pi(L)$.

The inductive case falls naturally into three parts. Let R_1 and R_2 be regular languages, and B_1 and B_2 the inductively constructed elementary bounded languages such that 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. 11 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 12 gives a characterization of linear languages based on regular languages, homomorphism, and some additional structures.

Lemma 12. *(from [13]) For every linear language L over Σ, there exist an alphabet A and its distinct copy $\widetilde{A}$, a 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.*

Proof. Assume the linear language L is given by linear grammar $G = (\mathcal{X}, \Sigma, \delta)$ and a initial variable X_0. We define the alphabet A to be $\{a_p \mid p \in \delta\}$. We define the regular language R as the language accepted by the automaton given by $(\mathcal{X} \cup \{q_f\}, T, X_0, \{q_f\})$ where: $T = \{(X, a_p, Y) \mid p = (X, \alpha Y \beta) \in \delta\} \cup \{(X, a_p, q_f) \mid p = (X, \alpha) \in \delta \wedge \alpha \in \Sigma^*\}$. Next we define the homomorphism, h which, for each $p = (X, \alpha Y \beta) \in \delta$, 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 13. *Let $X, Y \subseteq \Sigma^*$ be two languages and a homomorphism $h : A^* \to \Sigma^*$, 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 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 $\Pi(L \cap B) = \Pi(L)$.*

Proof. Since R is a regular language, we can use the result of Prop. 2 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 $Z\widetilde{A}^* \cap S = \{w\widetilde{w}^r \mid w \in Z\}$.

$\Pi(R') = \Pi(R)$	by above
only if $\Pi(R'\widetilde{A}^* \cap S) = \Pi(R\widetilde{A}^* \cap S)$	by above
only if $\Pi(h(R'\widetilde{A}^* \cap S)) = \Pi(h(R\widetilde{A}^* \cap S))$	Lem. 13
only if $\Pi(h(R'\widetilde{A}^* \cap S)) = \Pi(L)$	def. of L
only if $\Pi(h(R\widetilde{A}^* \cap S) \cap w_1^* \cdots w_m^* \widetilde{w_m^r}^* \cdots \widetilde{w_1^r}^*) = \Pi(L)$	def. of R'
only if $\Pi(h(R\widetilde{A}^* \cap S) \cap h(w_1^* \cdots w_m^* \widetilde{w_m^r}^* \cdots \widetilde{w_1^r}^*)) = \Pi(L)$	
only if $\Pi(L \cap h(w_1^* \cdots w_m^* \widetilde{w_m^r}^* \cdots \widetilde{w_1^r}^*)) = \Pi(L)$	def. of L
only if $\Pi(L \cap h(w_1)^* \cdots h(w_m)^* h(\widetilde{w_m^r})^* \cdots h(\widetilde{w_1^r})^*) = \Pi(L)$	

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

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 $\sigma_j \circ \sigma_{j+1}^k(v_X^k)$.

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 e.b. 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 $\Pi(\sigma(L) \cap B') = \Pi(\sigma(L))$.

Proof. Let $w_1, \ldots, w_k \in \Sigma^*$ be the words such that $B = w_1^* \cdots w_k^*$. Let $L_i = \sigma(w_i)$ for each $i \in \{1, \ldots, k\}$. 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 an elementary bounded language. Next, Lem. 11 where the elementary bounded language is given by $\tau(w_i)$, shows that we can construct an elementary bounded language B_i such that for all $t \in \mathbb{N}$, $\Pi(L_i^t \cap B_i) = \Pi(L_i^t)$. Define $B' = B_1 \ldots B_k$ that is an elementary bounded language. 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 $\Pi(L \cap w_1^* \cdots w_k^*) = \Pi(L)$, there is a word $w' \in \sigma(L \cap w_1^* \cdots w_k^*)$ such that $\Pi(w) = \Pi(w')$. Then we have

$$\begin{aligned}
w' &\in \sigma(L \cap w_1^* \cdots w_k^*) && \\
&\in \sigma(w_1^{t_1} \ldots w_k^{t_k}) && \text{for some } t_1, \ldots, t_k \\
&\in \sigma(w_1^{t_1}) \ldots \sigma(w_k^{t_k}) && \text{property of subst.} \\
&\in \sigma(w_1)^{t_1} \ldots \sigma(w_k)^{t_k} && \text{property of subst.} \\
&\in L_1^{t_1} \ldots L_k^{t_k} && \sigma(w_i) = L_i
\end{aligned}$$

For each $i \in \{1, \ldots, k\}$, we have $\Pi(L_i^{t_i} \cap B_i) = \Pi(L_i^{t_i})$, so we can find $w'' \in (L_1^{t_1} \cap B_1) \ldots (L_k^{t_k} \cap B_k)$ such that $\Pi(w'') = \Pi(w')$. Definition of B' also shows that $w'' \in B'$. Moreover

$$\begin{aligned}
w'' &\in (L_1^{t_1} \cap B_1) \ldots (L_k^{t_k} \cap B_k) && \\
&\in L_1^{t_1} \ldots L_k^{t_k} && \\
&\in \sigma(w_1)^{t_1} \ldots \sigma(w_k)^{t_k} && \sigma(w_i) = L_i \\
&\in \sigma(w_1^{t_1}) \ldots \sigma(w_k^{t_k}) && \text{property of subst.} \\
&\in \sigma(w_1^{t_1} \ldots w_k^{t_k}) && \text{property of subst.} \\
&\in \sigma(L \cap w_1^* \ldots w_k^*) && w_1^{t_1} \ldots w_k^{t_k} \in L \cap w_1^* \ldots w_k^* \\
&\in \sigma(L) &&
\end{aligned}$$

Algorithm 1: Bounded Sequence

Data: $\tilde{G}$ a linear grammar
Data: $\tilde{B}$ a valuation s.t. for every $X \in \mathcal{X}$ $\tilde{B}(X)$ is an elementary bounded language and $\Pi(L_X(\tilde{G})) = \Pi(L_X(\tilde{G}) \cap \tilde{B}(X))$
Data: $n \in \mathbb{N}$
Result: $B \in \mathcal{L}^{\mathcal{X}}$ such that for every $X \in \mathcal{X}$ $B(X)$ is an elementary bounded and $\Pi(B(X) \cap \boldsymbol{\nu}_n(X)) = \Pi(\boldsymbol{\nu}_n(X))$

1 Let B_{n-1} be $\tilde{B}[v_{\mathcal{X}}^{n-1}]$;
for $i = n-2, n-3, \ldots, 0$ **do**
  Let τ_{i+1} be the substitution which maps each v_X^{i+1} on $\tilde{B}[v_{\mathcal{X}}^i]$ and leaves each letter of Σ unchanged;
  foreach $X \in \mathcal{X}$ **do**
2   Let $B_i(X)$ be the language returned by Prop. 4 on the languages $\sigma_{i+2}^n(v_X^n)$ and $B_{i+1}(X)$, and the substitutions σ_{i+1}, τ_{i+1};

Let τ_0 be the substitution which maps each v_X^0 on the elementary bounded language $w_1^* \cdots w_p^*$ where $\{w_1, \ldots, w_p\} = \sigma_0(v_X^0)$ and leaves each letter of Σ unchanged;
foreach $X \in \mathcal{X}$ **do**
3   Let $B(X)$ be the language returned by Prop. 4 on the languages $\sigma_1^n(v_X^n)$ and $B_0(X)$, and the substitutions σ_0, τ_0;
return B

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 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 $\sigma_j^k(v_X^k)$.

4.1 *k*-fold Substitutions

Let us now solve Pb. 1 where the instance is given by a k-fold composition. Given a CFL $L = L_{X_0}(G)$ where $G = (\mathcal{X}, \Sigma, \delta)$ is a grammar and $X_0 \in \mathcal{X}$ an initial variable, we compute the linear grammar $\tilde{G}$ and the k-fold composition $\{\sigma_j\}_{0 \le j \le n}$ as defined in Sec. 3.3. With the result of Prop. 3, we find a valuation $\tilde{B}$ such that for every variable X, (1) $\tilde{B}(X)$ is an elementary bounded language and (2) $\Pi(L_X(\tilde{G})) = \Pi(L_X(\tilde{G}) \cap \tilde{B}(X))$.

The above reasoning is formally explained in Alg. 1.

We now prove the following invariants for Alg. 1.

Lemma 14. *In Alg. 1, for every $X \in \mathcal{X}$,*

- *for every $k \in \{0, \ldots, n-1\}$, $B_k(X)$ is an elementary bounded language on $(\Sigma \cup v_{\mathcal{X}}^k)^*$ such that $\Pi(\sigma_{k+1}^n(v_X^n) \cap B_k(X)) = \Pi(\sigma_{k+1}^n(v_X^n))$;*

– $B(X)$ *is an elementary bounded language on* Σ^* *such that* $\Pi(\boldsymbol{\nu}_n(X) \cap B(X)) = \Pi(\boldsymbol{\nu}_n(X))$.

Proof. – By induction on k:

Base case. ($k = n-1$) Alg. 1 assumes that $\tilde{B}(X)$ is an elementary bounded language, so is B_{n-1} by line 1. It remains to prove that $\Pi(\sigma_n(v_X^n) \cap B_{n-1}(X)) = \Pi(\sigma_n(v_X^n))$, which is equivalent, by definition of σ_n and B_{n-1}, to $\Pi(L_X(\tilde{G})[v_{\mathcal{X}}^{n-1}] \cap \tilde{B}[v_{\mathcal{X}}^{n-1}](X)) = \Pi(L_X(\tilde{G})[v_{\mathcal{X}}^{n-1}])$. By property of the symbol-to-symbol substitution $\sigma_{[v_Y/v_Y^{n-1}]}$, the equality reduces to $\Pi(L_X(\tilde{G}) \cap \tilde{B}(X)) = \Pi(L_X(\tilde{G}))$ which holds by assumption of Alg. 1.

Inductive case. ($0 \leq k \leq n-2$) At line 1, we see that we can apply the result of Prop. 4 because (1) $\sigma_{i+2}^n(v_X^n)$ is a CFL (CFLs are closed by context-free substitutions), (2) $B_{i+1}(X)$ is an elementary bounded language (induction hypothesis), (3) for every variable $Y \in \mathcal{X}$, $\sigma_{i+1}(v_Y^{i+1})$ is a CFL, $\tau_{i+1}(v_Y^{i+1})$ is an elementary bounded language and $\Pi(\sigma_{i+1}(v_Y^{i+1}) \cap \tau_{i+1}(v_Y^{i+1})) = \Pi(\sigma_{i+1}(v_Y^{i+1}))$. Hence, the proposition shows that $B_i(X)$ is an elementary bounded language and $\Pi(\sigma_{i+1}^n(v_X^n) \cap B_i(X)) = \Pi(\sigma_{i+1}^n(v_X^n))$.

– The above invariant for $k = 0$ shows that, for every variable $X \in \mathcal{X}$, (1) $B_0(X)$ is an elementary bounded language, and (2) $\Pi(\sigma_1^n(v_X^n) \cap B_0(X)) = \Pi(\sigma_1^n(v_X^n))$. We conclude from line 1 and Prop. 4 that $\Pi(\sigma_0^n(v_X^n) \cap B(X)) = \Pi(\sigma_0^n(v_X^n))$, and that $\Pi(\boldsymbol{\nu}_n(X) \cap B(X)) = \Pi(\boldsymbol{\nu}_n(X))$ by Lem. 8.

□

Referring to our initial problem, we finally find that:

Corollary 3. *Let* B *be the valuation returned by Alg. 1,* B *is a valuation in* $\mathcal{L}^{\mathcal{X}}$ *such that for every* $X \in \mathcal{X}$: $\Pi(L_X(G) \cap B(X)) = \Pi(L_X(G))$.

In fact, for $X = X_0$, $B(X_0)$ is the solution of Pb. 1 for the instance L. This concludes the proof of Th. 1. In what follows, we show two applications of Th. 1 in software verification. We conclude this section by showing a result related to the notion of progress if the result of Th. 1 is applied repeatedly.

Lemma 15. *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 form $\Pi(L_0 \cap B_0) = \Pi(L_0)$ 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 [13]). □

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 Applications

We now demonstrate two applications of our construction. The first application gives a semi-algorithm for checking reachability of multithreaded procedural programs [19,14,4]. The second application computes an underapproximation of the reachable states of a recursive counter machine.

5.1 Multithreaded Procedural Programs

Multithreaded Reachability. A common programming model consists of multiple recursive threads communicating via shared memory. Formally, we model such systems as pushdown networks [20]. Let n be a positive integer, a *pushdown network* is a triple $\mathcal{N} = (G, \Gamma, (\Delta_i)_{1\leq i\leq n})$ where G is a finite non-empty set of *globals*, Γ is the *stack alphabet*, and for each $1 \leq i \leq n$, 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 $(g, \alpha_1, \ldots, \alpha_n)$, where $g \in G$ and $\alpha_1, \ldots, \alpha_n \in \Gamma^*$ are individual stack content for each thread. Intuitively, the system consists of n threads, each of which have 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: $(g, \alpha_1, \ldots, \alpha_i, \ldots, \alpha_n) \rightarrow (g', \alpha_1, \ldots, \alpha_i', \ldots, \alpha_n)$ iff $(g, \alpha_i) \rightarrow_i (g', \alpha_i')$. By $\rightarrow_i^*$, $\rightarrow^*$, we denote the reflexive and transitive closure of these relations. Moreover, 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 $\rightsquigarrow_{i+1} = \rightsquigarrow_i \circ \rightsquigarrow$. 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 $n = 1$, 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 n threads to a language problem for n 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 $(G, \Gamma, (\Delta_i)_{1\le i \le n})$ with n 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 $\left\{(G', \Gamma, \Sigma, \Delta'_i, c_0^i)\right\}_{1\le i\le n}$, where:

- $G' = G \cup \{\perp\}$, Γ is given as above, and $\Sigma = G \times \{1, \ldots, n\}$,
- 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;
  - $\langle g, \gamma \rangle \overset{(g,j)}{\hookrightarrow} \langle \perp, \gamma \rangle$ for $j \in \{1, \ldots, n\} \setminus \{i\}$, $g \in G$, $\gamma \in \Gamma$;
  - $\langle \perp, \gamma \rangle \overset{(g,j)}{\hookrightarrow} \langle \perp, \gamma \rangle$ for $j \in \{1, \ldots, n\} \setminus \{i\}$, $g \in G$, $\gamma \in \Gamma$;
  - $\langle \perp, \gamma \rangle \overset{(g,i)}{\hookrightarrow} \langle g, \gamma \rangle$ for $g \in G$, $\gamma \in \Gamma$.
- let $C_0 = (g, \alpha_1, \ldots, \alpha_i, \ldots, \alpha_n)$, c_0^i is given by $(\perp, \alpha_i)$ if $i > 1$; (g, α_1) else.

Proposition 5. *Let n be a positive integer, and $(\mathcal{N}, C_0, C)$ be an instance of the reachability problem with n threads, one can effectively construct CFLs $(L_1, \ldots, L_n)$ (as pushdown acceptors) such that $C_0 \to^* C$ iff $L_1 \cap \cdots \cap L_n \neq \emptyset$.*

The converse of the proposition is also true, and since the emptiness problem for intersection of CFLs is undecidable [13], so is the reachability problem. We will now compare two underapproximation techniques. The context-bounded switches for the reachability problem [18] and the bounded languages for the emptiness problem that is given below.

Let $L_1, \ldots, L_k$ be context-free languages, and consider the problem to decide if $\bigcap_{1\le i\le k} L_i \neq \emptyset$. 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 $\bigcap_i^{(B)} L_i = \left(\bigcap_i L_i\right) \cap B$. 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 16. *Given an elementary bounded language $B = w_1^* \cdots w_n^*$ and CFLs $L_1, \ldots, L_k$, it is decidable to check if $\bigcap_{1\le i\le k}^{(B)} L_i \neq \emptyset$.*

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

We conclude from $w \in \bigcap^{(B)}_{1\leq i\leq k} L_i$ 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 $(t_1, \ldots, t_n) = \Pi_A(h^{-1}(w) \cap a_1^* \cdots a_n^*)$, 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 $\bigcap_{1\leq i\leq k} \Pi_A\big(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\big)$.

For the other implication, consider $(t_1, \ldots, t_n)$ a vector of $\bigcap_{1\leq i\leq k} \Pi_A\big(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\big)$ 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 $(t_1, \ldots, t_n) \in \Pi_A\big(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\big)$, there exists a word $w' \in a_1^* \cdots a_n^*$ such that $\Pi_A(w') = (t_1, \ldots, t_n)$ 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 [13]. Moreover, given a CFL, we can compute its Parikh image which is a semilinear set. Finally, we can compute the semilinear sets $\Pi_A\big(h^{-1}(L_i \cap B) \cap a_1^* \cdots a_n^*\big)$ and the emptiness of the intersection of semilinear sets is decidable [11]. □

While Lem. 16 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 $\Pi(L_i \cap B_i) = \Pi(L_i)$. 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. 15 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.

At Alg. 2, we present a 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]. Given a pushdown network, global configurations C_0 and C, and a number $k \geq 1$, the *context-bounded reachability problem* asks whether $C_0 \leadsto_k C$ holds, i.e. if C can be reached from C_0 within at most 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. 16) 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 $C_0 \to^* C$ iff $\cap_i L_i \neq \emptyset$. For each $k \geq 1$, there is an elementary bounded language B_k such that $C_0 \leadsto_k C$ only if $\bigcap_i^{(B_k)} L_i \neq \emptyset$. Also, $\bigcap_i^{(B_k)} L_i \neq \emptyset$ only if $C_0 \to^* C$.*

Algorithm 2: Intersection

Input: 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 elementary bounded languages such that $\Pi(L_1 \cap B_1) = \Pi(L_1)$ and $\Pi(L_2 \cap B_2) = \Pi(L_2)$;
    Compute $B = B_1 \cdot B_2$;
    if $L_1 \cap^{(B)} L_2 \neq \emptyset$ **then**
      return $L_1^0 \cap L_2^0$ *is not empty*
  $L_1 \leftarrow L_1 \cap \overline{B}$, $L_2 \leftarrow L_2 \cap \overline{B}$

Proof. Consider all sequences $C_0 \rightsquigarrow C_1 \cdots C_{k-1} \rightsquigarrow C_k$ of k or fewer switches. Let $S = \bigcup_{j=0}^{k} \Sigma^j$. By the CFL encoding (Prop. 5) each of these sequences corresponds to a word in S. If $C_0 \rightsquigarrow_k C$, then there is a word $w \in \bigcap_i L_i$ and $w \in S$. Define B_k to be $w_1^* \cdots w_m^*$ where $w_1, \ldots, w_m$ is an enumeration of all strings in S. We conclude from $w \in S$ 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 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 $\cap_i L_{ik} \neq \emptyset$.

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`.

5.2 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 [21]) and to programs manipulating complex data structures like lists (see for instance [3]). 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 $\mathbb{N}^{2n}$.

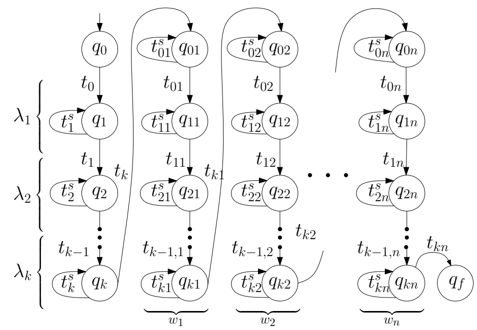
A *M-configuration* (q, x) consists of a location $q \in Q$ and a vector $x \in \mathbb{N}^n$; we define C_M as the set of M-configurations. For each transition $t \in T$, its semantics is given by the reachability relation $R_M(t)$ over C_M defined as $(q, x)R_M(t)(q', x')$ iff $q = \alpha(t)$, $q' = \beta(t)$, 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 $\{1, \ldots, |Q|\}$.

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 $B = w_1^* \cdots w_n^*$.

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

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

$G_{t_i^s} = G^+_{\lambda_{i1}} \circ \ldots \circ G^+_{\lambda_{in}}$,
$G_{t_{ij}^s} = G(w_j^{p_{ij}}) \circ G^-_{\lambda_{ij}}$, 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 $M' = (Q', T', \alpha', \beta', \{G_t\}_{t\in T'})$.

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 symbols $a_1, \ldots, a_n$ to the words $w_1, \ldots, w_n$, respectively. We compute the language $L'_A = h^{-1}(L \cap B) \cap a_1^* \cdots a_n^*$. Let $S = \Pi_{\{a_1,\ldots,a_n\}}(L'_A)$, and note that S is a semilinear set. For clarity, we first consider a linear set H where $p_0 = (p_{01}, \ldots, p_{0n})$ denotes the constant and $\{p_i = (p_{i1}, \ldots, p_{in})\}_{i\in I\setminus\{0\}}$ the set of periods of H and $I = \{0, \ldots, k\}$. Let $J = \{1, \ldots, n\}$. In the following, for every pair of vectors $x = (x_1, \ldots, x_r)$ and $y = (y_1, \ldots, y_s)$, we denote by (x, y) the vector $(x_1, \ldots, x_r, y_1, \ldots, y_s)$. 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 $w_1^{p_{01}+\lambda_1 p_{11}+\cdots+\lambda_k p_{k1}} \ldots w_n^{p_{0n}+\lambda_1 p_{1n}+\cdots+\lambda_k p_{kn}}$ 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 $\big\{(q_0, (x, v)) \mid (q_f, x) \in D \wedge v = 0^{(k+1)n}\big\}$.

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 [5,10]. 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 $\text{post}[R_{M'}(T'^*)](D')$ of reachable configurations of (M', D') is Presburger definable, effectively computable and relates to $\text{post}[R_M(L')](D)$ for the bounded language L' as follows.

Lemma 17. *Let* $(q_f, x) \in C_M$,
$(q_f, x) \in \text{post}[R_M(L')](D)$ *iff* $\exists v \in \mathbb{N}^{(k+1)n} \colon (q_f, (x, v)) \in \text{post}[R_{M'}(T'^*)](D')$.

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 composant of q_f equal to one and the others equal to zero. Lem. 15 shows that by iterating the construction we obtain a semi-algorithm for a context-free language.

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

References

1. S. Bardin, A. Finkel, J. Leroux and Ph. Schnoebelen. Flat Acceleration in Symbolic Model Checking. In *ATVA '05*, LNCS 3707, 474–488. Springer, 2005.
2. M. Blattner and M. Latteux. Parikh-bounded languages. In *ICALP '81*, LNCS 115, 316–323. Springer, 1981.
3. A. Bouajjani, M. Bozga, P. Habermehl, R. Iosif, P. Moro, and T. Vojnar. Programs with Lists are Counter Automata. In *CAV'06*, LNCS 4144, 517–531. Springer, 2006.
4. 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.
5. M. Bozga, C. Gîrlea, and R. Iosif. Iterating octagons. In *TACAS '09*, LNCS 5505, 337–351. Springer, 2009.
6. N. Chomsky and M.P. Schützenberger. The algebraic theory of context-free languages. *Comp. Programming and Formal Systems*, 118–161. North-Holland, 1963.
7. P. Cousot. *Méthodes Itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes (in French).* Thèse d'état ès sciences mathématiques, Université scientifique et médicale de Grenoble, March 1978.
8. 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.

9. J. Esparza, S. Kiefer, and M. Luttenberger. On fixed point equations over commutative semirings. In *STACS '07*, LNCS 4393, 296–307. Springer, 2007.
10. A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *FSTTCS '02*, LNCS 2556, 145–156. Springer, 2002.
11. S. Ginsburg. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, NY, USA, 1966.
12. S. Ginsburg and E. Spanier. Bounded ALGOL-like languages. *Trans. Amer. Math. Soc.*, 113:333–368, 1964.
13. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation (1st Edition).* Addison Wesley, 1979.
14. V. Kahlon. Tractable analysis for concurrent programs via bounded languages. Unpublished.
15. 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.
16. M. Latteux and J. Leguy. Une propriete de la famille GRE. In *Fundamentals of Computation Theory*, 255–261, 1979. Akademie-Verlag.
17. J. Leroux and G. Sutre. On flatness for 2-dimensional vector addition systems with states. In *CONCUR '04*, LNCS 3170, 402–416. Springer, 2004.
18. S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS '05*, LNCS 3440, 93–107. Springer, 2005.
19. G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM TOPLAS*, 22(2):416–430, 2000.
20. D. Suwimonteerabuth, J. Esparza and S. Schwoon. Symbolic Context-Bounded Analysis of Multithreaded Java Programs. In *SPIN '08*, LNCS 5156, 270–287. Springer, 2008.
21. L. Van Begin. Efficient Verification of Counting Abstractions for Parametric Systems. *Ph.D Thesis.* Université Libre de Bruxelles. 2003.