Underapproximation of Procedure Summaries for Integer Programs

Pierre Ganty, Radu Iosif, Filip Konečný

Underapproximation of Procedure Summaries for Integer Programs

Pierre Ganty[1], Radu Iosif[2], and Filip Konečný[2,3]

[1]IMDEA Software Institute, Madrid, Spain
[2]VERIMAG/CNRS, Grenoble, France
[3]École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Abstract

We show how to underapproximate the procedure summaries of recursive programs over the integers using off-the-shelf analyzers for non-recursive programs. The novelty of our approach is that the non-recursive program we compute may capture unboundedly many behaviors of the original recursive program for which stack usage cannot be bounded. Moreover, we identify a class of recursive programs on which our method terminates and returns the precise summary relations without underapproximation. Doing so, we generalize a similar result for non-recursive programs to the recursive case. Finally, we present experimental results of an implementation of our method applied on a number of examples.

1 Introduction

Formal approaches to reasoning about behaviors of programs usually fall into one of the following two categories: *certification* approaches, that provide proofs of correctness, and *bug-finding* approaches, that explore increasingly larger sets of traces in order to find possible errors. While the methods in the first category are used typically in the development of safety-critical software whose failures may incur dramatic losses in terms of human lives (airplanes, space missions, or nuclear power plants), the methods in the second category have a broad application in industry, outside of the safety-critical market niche. Another difference between the two categories is methodological: certification approaches are based on *over-approximations* of the set of behaviors (if the over-approximation is free of errors, the original system is correct), while bug-finding needs systematic *under-approximation* techniques (if there are errors, the method will eventually discover all of them). Finally, over-approximation methods are guaranteed to terminate, but the answer might be inconclusive (spurious errors are introduced due to the abstraction), whereas under-approximation methods provide precise results (all reported errors are real), but with no guarantee for termination.

Procedure summaries are relations between the input and return values of a procedure, resulting from its terminating executions. Computing summaries is important, as they are a key enabler for the development of modular verification techniques for inter-procedural programs, such as checking safety, termination

or equivalence properties. Summary computation is, however, challenging in the presence of *recursive procedures* with integer parameters, return values, and local variables. While many analysis tools exist for non-recursive programs, only a few ones address the problem of recursion (e.g. InterProc [19]).

In this paper, we propose a novel technique to generate arbitrarily precise *underapproximations* of summary relations. Our technique is based on the following idea. The control flow of procedural programs is captured precisely by the language of a context-free grammar. A k-index underapproximation of this language (where $k \geqslant 1$) is obtained by filtering out those derivations of the grammar that exceed a budget, called *index*, on the number (at most k) of occurrences of nonterminals occurring at each derivation step. As expected, the higher the index, the more complete the coverage of the underapproximation. From there we define the k-index summary relations of a program by considering the k-index underapproximation of its control flow. Our method then reduces the computation of k-index summary relations for a recursive program to the computation of summary relations for a non-recursive program, which is, in general, easier to compute because of the absence of recursion. The reduction was inspired by a decidability proof [4] in the context of Petri nets.

The contributions of this paper are threefold. First, we show that, for a given index, recursive programs can be analyzed using off-the-shelf analyzers designed for non-recursive programs. Second, we identify a class of recursive programs, with possibly unbounded stack usage, on which our technique is complete, i.e. it terminates and returns the precise result. Third, we present experimental results of an implementation of our method applied on a number of examples.

Motivating Example To properly introduce the reader to our result, we describe our source-to-source program transformation through an illustrative example. Consider the recursive program $\mathcal{P} = \{P\}$, consisting of a single recursive procedure P, given in Fig. 1 (a), whose control flow graph is given in Fig. 1 (b). The nodes of this graph represent control locations in the program, with a designated initial location Q_1^{init} and a final location ε. The edges are labeled with relations denoting the program semantics, where primed variables x' and z' denote the values at the next step. For instance, the edge $t_2 : Q_2 \xrightarrow{z'=P(x-1)\wedge x'=x} Q_3$ corresponds to the recursive call on line 3 in the program—the edge labels of the control flow graph explicitly mention the copies of variables not changed by the program action corresponding to the edge, e.g. $x' = x$.

In this paper, we model programs using visibly pushdown grammars (VPG) [3]. The VPG for P is given in Fig. 1 (c). The role of the grammar is to define the set of *interprocedurally valid* paths in the control-flow graph of the program P. Every edge in the control-flow graph matches one or two symbols from the finite alphabet $\{\tau_1, \langle\tau_2, \tau_2\rangle, \tau_3, \tau_4\}$, where $\langle\tau_2$ and $\tau_2\rangle$ denote the call and return, respectively. Each edge in the graph translates to a production rule in the grammar, labeled p_1^b, p_2^c, p_3^a and p_4^a—the superscript a, b and c distinguishes rules with 0, 1 and 2 nonterminals on the right-hand side, respectively. For instance, the call edge t_2 becomes the rule $Q_2 \to \langle\tau_2 Q_1^{init} \tau_2\rangle Q_3$. The language of the grammar of Fig. 1 (c) (with axiom Q_1^{init}) is the set $\{(\tau_1\langle\tau_2)^n \tau_4 (\tau_2\rangle\tau_3)^n \mid n \in \mathbb{N}\}$ of interprocedurally valid paths, where each call symbol $\langle\tau_2$ is matched by a return symbol $\tau_2\rangle$, and the matching relation is well-parenthesized.

The outcome of the program transformation applied to P is the non-recursive

program $\mathcal{Q} = \{query^i\}_{i=0}^{K}$, depicted in Fig. 1 (d), where K is a parameter of our analysis. The main idea is that the executions of the procedure $query^k$, ending with an empty stack, correspond to the derivations of the VPG in Fig. 1 (c), of index at most k—since there is no derivation of index 0, the set of executions of $query^0$ will be empty. The body of a procedure $query^k$ consists of a main loop, starting at the control label *begin_loop* in Fig. 1 (d). Each branch inside the main loop corresponds to the simulation of one of the production rules of the grammar in Fig. 1 (c) and starts with a control label which is the name of that rule ($p_1^b, p_2^c, p_3^a, p_4^a$). Next, we explain the relations labeling the control edges of $query^k$. For each production rule p in the grammar we have a relation $\rho_p(x_I, z_I, x_O, z_O)$, where subscript I and O denote the input and output copies of the program variables of P, respectively. In addition, we consider auxiliary copies x_J, z_J, x_K, z_K and x_L, z_L, defined in a similar way. For instance, the auxiliary variables store intermediate results of the computation of p_2^c as follows: $[x_I, z_I]\ \langle\!\langle \tau_2\ [x_J, z_J]\ Q_1^{init}\ [x_K, z_K]\ \tau_2 \rangle\!\rangle\ [x_L, z_L]\ Q_3\ [x_O, z_O]$. The transition $p_2^c \rightarrow in\_order/out\_of\_order$ can be understood by noticing that $\langle\!\langle \tau_2$ gives rise to the constraint $x_J = x_I - 1$, $\tau_2 \rangle\!\rangle$ to $z_L = z_K$ and $x_I = x_L$ corresponds to the frame condition $x' = x$.

The peculiarity of the resulting program is that a function call is modeled in two possible ways: (i) *in-order* execution of the function body, followed by the continuation of the call, and (ii) *out-of-order* execution of the continuation, followed by the execution of the function body. The two cases correspond to k-index derivations of the VPG in Fig 1 (c) of the form $uQ_1^{init}vQ_3w \Rightarrow^* uv_1vQ_3w \Rightarrow^* uv_1vv_2w$ and $uQ_1^{init}vv_2w \Rightarrow^* uv_1vv_2w \Rightarrow^* uv_1vv_2w$, respectively, where $Q_1^{init} \Rightarrow^* v_1$ and $Q_3 \Rightarrow^* v_2$ are derivations of the VPG. In the first case, the control path simulating the derivation in $query^k$ follows the left branch $in\_order/out\_of\_order \rightarrow begin\_loop$, whereas the second case is simulated by the right branch.

Since the only call of $query^k$ is to $query^{k-1}$, on the edges $in\_order/out\_of\_order \rightarrow begin\_loop$, the whole program is a non-recursive under-approximation of the semantics of the original program P, amenable to analysis using intra-procedural program analysis tools. Indeed, the computation of the pre-condition relation of the program $\mathcal{Q} = \{query^2, query^1, query^0\}$ with the FLATA tool [17] yields the formula $z_O = 2 \cdot x_I$, which matches the summary $z' = 2 \cdot x$ of the program P.

In other words, the analysis of the under-approximation of $\mathcal{P}$ of index at most 2 suffices to infer the complete summary of the program (the analysis for values $K > 2$ will necessarily yield the same result, since the under-approximation method is monotonic in K). This fact matches the completeness result of Section 5, stating that the analysis needs to be carried up to a certain bound (linear in the size of the program's VPG) whenever the language of the VPG is included in the language of the regular expression $w_1^* \dots w_n^*$, for some non-empty words $w_1, \dots, w_n$. In our case, the completeness result applies due to $\{(\tau_1 \langle\!\langle \tau_2)^n\ \tau_4\ (\tau_2 \rangle\!\rangle \tau_3)^n \mid n \in \mathbb{N}\} \subseteq (\tau_1 \langle\!\langle \tau_2)^*\ \tau_4^*\ (\tau_2 \rangle\!\rangle \tau_3)^*$.

Related Work The problem of analyzing recursive programs handling integers (in general, unbounded data domains) has gained significant interest with the seminal work of Sharir and Pnueli [24]. They proposed two orthogonal approaches for interprocedural dataflow analysis. The first one keeps precise values (*call strings*) up to a limited depth of the recursion stack, which bounds the number of executions. In contrast to the methods based on the call strings approach,

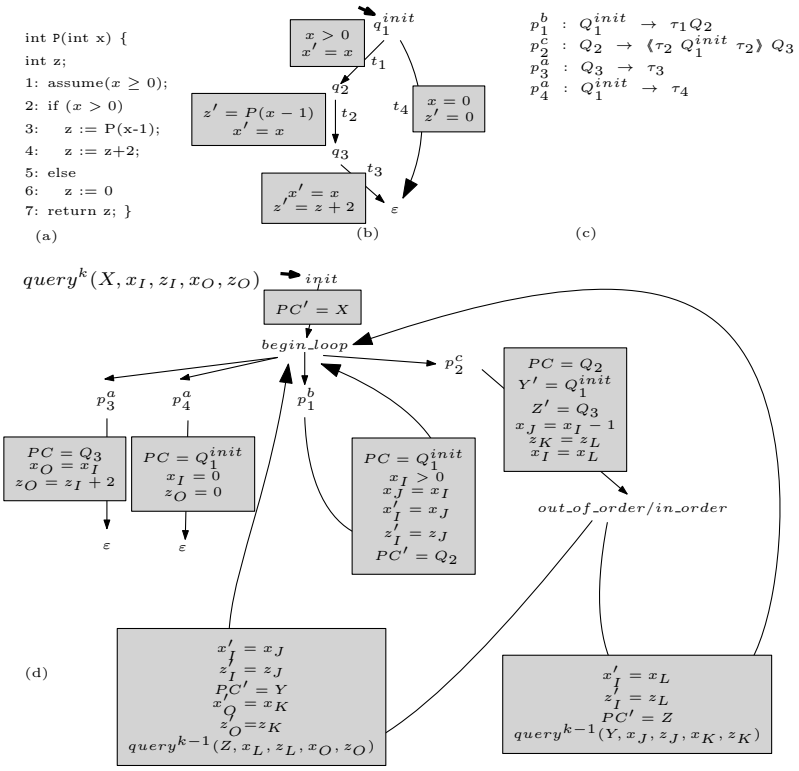

Figure 1: A recursive program returning the parameter value multiplied by two (a), its corresponding control flow graph (b) and visibly pushdown grammar (c), and the non-recursive program $query^k(X, x_I, z_I, x_O, z_O)$ resulting from our index-bounded under-approximation (d).

our method can also analyse precisely certain programs for which the stack is unbounded, allowing for unbounded number of executions to be represented at once.

The second approach of Sharir and Pnueli [24] is based on computing the least fixed point of a system of recursive dataflow equations (the *functional approach*). This approach to interprocedural analysis is based on computing an increasing *Kleene sequence* of abstract summaries. It is to be noticed that abstraction is key to ensuring termination of the Kleene sequence, the result being an over-approximation of the precise summary. Recently [11], a *Newton sequence* defined over the language semiring was shown to converge at least as fast as the Kleene sequence over the same semiring. An iterate of a Newton sequence is the set of control paths in the program that correspond to words produced by a grammar, with bounded number of nonterminals at each step in the derivation. By increasing this bound, we obtain an increasing sequence of languages that converges to the language of behavior of the program. Our contribution can be thus seen as a technique to compute the iterates of the Newton sequence for programs with integer parameters, return values, and local variables, the result being, at each step, an under-approximation of the precise summary.

The complexity of the functional approach was shown to be polynomial in the size of the (finite) abstract domain, in the work of Reps, Horwitz and Sagiv [23]. This result is achieved by computing summary information, in order to reuse previously computed information during the analysis. Following up on this line of work, most existing abstract analyzers, such as InterProc [19], also use relational domains to compute over-approximations of function summaries – typically widening operators are used to ensure termination of fixed point computations. The main difference of our method with respect to static analyses is the use of under-approximation instead of over-approximation. If the final purpose of the analysis is program verification, our method will not return false positives. Moreover, the coverage can be increased by increasing the bound on the derivation index.

Previous works have applied model checking based on abstraction refinement to recursive programs. One such method, known as *nested interpolants* represents programs as nested word automata [3], which have the same expressive power as the visibly pushdown grammars used in our paper. Also based on interpolation is the Whale algorithm [2], which combines partial exploration of the execution paths (underapproximation) with the overapproximation provided by a predicate-based abstract post operator, in order to compute summaries that are sufficient to prove a given safety property. Another technique, similar to Whale, although not handling recursion, is the Smash algorithm [15] which combines may- and must-summaries for compositional verification of safety properties. These approaches are, however, different in spirit from ours, as their goal is proving given safety properties of programs, as opposed to computing the summaries of procedures independently of their calling context, which is our case. We argue that summary computation can be applied beyond safety checking, e.g., to prove termination [5], or program equivalence.

The technique of under-approximation is typically used for bug discovery, rather than certification of correctness. For instance, bug detection based on under-approximation has been developed for non-recursive C programs with arrays [18]. Our approach in orthogonal, as we consider more complex control structures (possibly recursive procedure calls) but simpler data domains (scalar values such as integers).

Paper organization. After introducing the basic definition in Section 2, we present, in Section 3, our model for programs, a semantics based on nested words and another one, equivalent, based on derivations of the underlying grammar. Then, in Section 4, we present our main contribution which is a program transformation underapproximating the semantics of the input program. In Section 5, we define a class of programs for which the underapproximation is complete. Finally, after reporting on experiments in Section 6 we conclude in Section 7.

2 Preliminaries

2.1 Grammars

Let Σ be an *alphabet*, that is a finite non-empty set of symbols. We denote by Σ^* the set of finite words over Σ including ε, the empty word. Given a word $w \in \Sigma^*$, let $|w|$ denote its length and let $(w)_i$, with $1 \leqslant i \leqslant |w|$, be the i-th symbol of w.

By $(w)_{i\ldots j}$, with $1 \leqslant i \leqslant j \leqslant |w|$, we denote the subword $(w)_i \ldots (w)_j$ of w. For a word $w \in \Sigma^*$ and $\Sigma' \subseteq \Sigma$, we denote by $w{\downarrow_{\Sigma'}}$ the result of erasing all symbols of w not in Σ'.

A *context-free grammar* (or simply grammar) is a tuple $G = \langle \Xi, \Sigma, \Delta \rangle$, where Ξ is a finite nonempty set of *nonterminals*, Σ is an alphabet, such that $\Xi \cap \Sigma = \varnothing$, and $\Delta \subseteq \Xi \times (\Sigma \cup \Xi)^*$ is a finite set of *productions*. A production $(X, w) \in \Delta$ is often conveniently noted $X \to w$. Also define $head(X \to w) = X$ and $tail(X \to w) = w$. Given two strings $u, v \in (\Sigma \cup \Xi)^*$, a production $(X, w) \in \Delta$ and $1 \leqslant j \leqslant |u|$, we define a *step* $u \xRightarrow{(X,w)/j}_G v$ if, and only if, $(u)_j = X$ and $v = (u)_1 \cdots (u)_{j-1} \cdot w \cdot (u)_{j+1} \cdots (u)_{|u|}$. We omit (X, w) or j above the arrow when it is not important. In this notation and others, when G is clear from the context, we omit it. *Step sequences* (including the empty sequence) are defined using the reflexive transitive closure of the step relation $\Rightarrow_G$, denoted $\Rightarrow^*_G$. For instance, $X \Rightarrow^*_G w$ means there exists a sequence of steps that produces the word $w \in (\Sigma \cup \Xi)^*$, starting from X. We call any *step sequence* $v \Rightarrow^*_G w$ *a derivation* whenever $v \in \Xi$ and $w \in \Sigma^*$. The language produced by G, starting with a nonterminal X is the set $L_X(G) = \{w \in \Sigma^* \mid X \Rightarrow^*_G w\}$.

By defining a *control word* to be a sequence of productions $\gamma \in \Delta^*$, we can annotate step sequences as expected: $\varepsilon \in \Delta^*$ is the control word for empty step sequences, and given a control word γ of length n we write $u \overset{\gamma}{\Rightarrow}_G v$ whenever there exists $w_0, \ldots, w_n \in (\Xi \cup \Sigma)^*$ such that

$$u = w_0 \xRightarrow{(\gamma)_1}_G w_1 \xRightarrow{(\gamma)_2}_G \ldots w_{n-1} \xRightarrow{(\gamma)_n}_G w_n = v \ .$$

Given a nonterminal $X \in \Xi$ and a set $\Gamma \subseteq \Delta^*$ of control words (a.k.a *control set*), we denote by $\hat{L}_X(\Gamma, G) = \{w \in \Sigma^* \mid \exists \gamma \in \Gamma \colon X \overset{\gamma}{\Rightarrow} w\}$ the language generated by G using only control words in Γ.

2.2 Visibly Pushdown Grammars

To model the control flow of procedural programs we use languages generated by visibly pushdown grammars, a subset of context-free grammars. In this setting, words are defined over a *tagged alphabet* $\widehat{\Sigma} = \Sigma \cup \langle\!\langle \Sigma \cup \Sigma \rangle\!\rangle$, where $\langle\!\langle \Sigma = \{\langle\!\langle a \mid a \in \Sigma\}$ represents procedure *call* sites and $\Sigma \rangle\!\rangle = \{a \rangle\!\rangle \mid a \in \Sigma\}$ represents procedure *return* sites. Formally, a *visibly pushdown grammar* $G = \langle \Xi, \widehat{\Sigma}, \Delta \rangle$ is a grammar that has only productions of the following forms, for some $a, b \in \Sigma$:

$$X \to a \qquad\qquad X \to a\,Y \qquad\qquad X \to \langle\!\langle a\,Y\,b \rangle\!\rangle\,Z \ .$$

It is worth pointing that, for our purposes, we do not need a visibly pushdown grammar to generate the empty string ε. Each tagged word generated by visibly pushdown grammars is associated a *nested word* [3] the definition of which we briefly recall. Given a finite alphabet Σ, a *nested word* over Σ is a pair $(w, \rightsquigarrow)$, where $\rightsquigarrow \subseteq \{1, \ldots, |w|\} \times \{1, \ldots, |w|\}$ is a set of *nesting edges* (or simply edges) where:

1. $i \rightsquigarrow j$ only if $i < j$; edges only go forward;
2. $\|\{j \mid i \rightsquigarrow j\}\| \leqslant 1$ and $\|\{i \mid i \rightsquigarrow j\}\| \leqslant 1$; no two edges share a call/return position;

3. if $i \rightsquigarrow j$ and $k \rightsquigarrow \ell$ then it is not the case that $i < k \leqslant j < \ell$; edges do not cross.

Intuitively, we associate a nested word to a tagged word as follows: there is an edge between tagged symbols $\langle\!\langle a$ and $b\rangle\!\rangle$ if and only if both symbols are produced by the same derivation step. Finally, let w_nw denote the mapping which given a tagged word in the language of a visibly pushdown grammar returns the nested word thereof.

Example 2.1. *For the tagged word* $w = \tau_1\langle\!\langle\tau_2\tau_1\langle\!\langle\tau_2\tau_4\tau_2\rangle\!\rangle\tau_3\tau_2\rangle\!\rangle\tau_3$, $w\_nw(w) = (\tau_1\tau_2\tau_1\tau_2\tau_4\tau_2\tau_3\tau_2\tau_3, \{2 \rightsquigarrow 8, 4 \rightsquigarrow 6\})$ *is the associated nested word.* ■

2.3 Integer Relations

Given a set S, let $\|S\|$ denote its cardinality. We denote by $\mathbb{Z}$ the set of integers. Let $\mathbf{x} = \langle x_1, \ldots, x_d\rangle$ be a tuple of variables, for some $d > 0$. We define by $\mathbf{x}'$ the *primed* variables of $\mathbf{x}$ to be the tuple $\langle x'_1, x'_2, \ldots, x'_d\rangle$. We consider implicitly that all variables range over $\mathbb{Z}$. We denote by $|\mathbf{x}| = d$ the length of the tuple $\mathbf{x}$, and for a tuple $\mathbf{y} = \langle y_1, \ldots, y_e\rangle$, we denote by $\mathbf{x} \cdot \mathbf{y} = \langle x_1, \ldots, x_d, y_1, \ldots, y_e\rangle$ their concatenation. For two tuples of variables $\mathbf{t}$ and $\mathbf{s}$ such that $|t| = |s| = k$, we denote by $\mathbf{t} = \mathbf{s}$ the conjunction $\bigwedge_{i=1}^{k} t_i = s_i$.

A *linear term* t is a linear combination of the form $a_0 + \sum_{i=1}^{d} a_i x_i$, where $a_0, \ldots, a_d \in \mathbb{Z}$. An *atomic proposition* is a predicate of the form $t \leqslant 0$, where t is a linear term. We consider formulae in the first-order logic over atomic propositions $t \leqslant 0$, also known as *Presburger arithmetic*. A *valuation* of $\mathbf{x}$ is a function $\nu : \mathbf{x} \to \mathbb{Z}$. The set of all valuations of $\mathbf{x}$ is denoted by $\mathbb{Z}^{\mathbf{x}}$. If $\mathbf{x} = \langle x_1, \ldots, x_d\rangle$ and $\nu \in \mathbb{Z}^{\mathbf{x}}$, then $\nu(\mathbf{x})$ denotes the tuple $\langle \nu(x_1), \ldots, \nu(x_d)\rangle$. An arithmetic formula $\mathcal{R}(\mathbf{x}, \mathbf{y}')$ defining a relation $R \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{y}}$ is evaluated with respect to two valuations $\nu_1 \in \mathbb{Z}^{\mathbf{x}}$ and $\nu_2 \in \mathbb{Z}^{\mathbf{y}}$, by replacing each $x \in \mathbf{x}$ by $\nu_1(x)$ and each $y' \in \mathbf{y}'$ by $\nu_2(y)$ in $\mathcal{R}$. The composition of two relations $R_1 \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{y}}$ and $R_2 \subseteq \mathbb{Z}^{\mathbf{y}} \times \mathbb{Z}^{\mathbf{z}}$ is denoted by $R_1 \circ R_2 = \{\langle \mathbf{u}, \mathbf{v}\rangle \in \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{z}} \mid \exists \mathbf{t} \in \mathbb{Z}^{\mathbf{y}}. \langle \mathbf{u}, \mathbf{t}\rangle \in R_1 \text{ and } \langle \mathbf{t}, \mathbf{v}\rangle \in R_2\}$. We denote $\mathbf{y} \subseteq \mathbf{x}$ if $\mathbf{y} = \langle x_{i_1}, \ldots, x_{i_\ell}\rangle$, for a sequence of indices $1 \leqslant i_1 < \ldots < i_\ell \leqslant d$ of $\mathbf{x}$. For a valuation $\nu \in \mathbb{Z}^{\mathbf{x}}$ and a tuple $\mathbf{y} \subseteq \mathbf{x}$, we denote by $\nu{\downarrow_{\mathbf{y}}} \in \mathbb{Z}^{\mathbf{y}}$ the projection of ν onto variables $\mathbf{y}$, i.e. $\nu{\downarrow_{\langle y_1, \ldots, y_k\rangle}} = \langle \nu(y_1), \ldots, \nu(y_k)\rangle$. Finally, given two valuations $I, O \in \mathbb{Z}^{\mathbf{x}}$, we denote by $I \cdot O$ the valuation $I(\mathbf{x}) \cdot O(\mathbf{x})$, and we define $\mathbb{Z}^{\mathbf{x} \times \mathbf{x}} = \{I \cdot O \mid I, O \in \mathbb{Z}^{\mathbf{x}}\}$.

2.4 Parikh Images

Let $\Theta = \{\theta_1, \ldots, \theta_k\}$ be a linearly ordered subset of the alphabet Σ. For a symbol $a \in \Sigma$ its *Parikh image* is defined as $Pk_\Theta(a) = \mathbf{e}_i$ if $a = \theta_i$, where $\mathbf{e}_i$ is the k-dimensional vector having 1 on the i-th position and 0 everywhere else. Otherwise, if $a \in \Sigma \backslash \Theta$, let $Pk_\Theta(a) = \mathbf{0}$ where $\mathbf{0}$ is the k-dimensional vector with 0 everywhere. For a word $w \in \Sigma^*$ of length n, we define $Pk_\Theta(w) = \sum_{i=1}^{n} Pk_\Theta((w)_i)$.[1] Furthermore, let $Pk_\Theta(L) = \{Pk_\Theta(w) \mid w \in L\}$ for any language $L \subseteq \Sigma^*$.

[1] We adopt the convention that the empty sum evaluates to $\mathbf{0}$.

2.5 Labelled Graphs

In this paper we use of the notion of *labelled graph* $\mathcal{G} = \langle Q, \mathcal{L}, \delta \rangle$, where Q is a finite set of vertices, $\mathcal{L}$ is a set of labels whose elements label edges as defined by the edge relation $\delta \subseteq Q \times S \times Q$. We denote by $q \xrightarrow{\ell} q'$ the fact that $(q, \ell, q') \in \delta$. A *path* π in $\mathcal{G}$ is an alternating sequence of vertices and edges whose endpoints are vertices. Sometimes, π is conveniently written as $q_0 \xrightarrow{\ell_1} q_1 \xrightarrow{\ell_2} \dots q_{n-1} \xrightarrow{\ell_n} q_n$ and further abbreviated $q_0 \xrightarrow{w} q_n$ where $w = \ell_1 \dots \ell_n$.

3 Integer Recursive Programs

We consider in the following that programs are collections of procedures calling each other, possibly according to recursive schemes. Formally, an *integer program* is an indexed tuple $\mathcal{P} = \langle P_1, \dots, P_n \rangle$, where $P_1, \dots, P_n$ are *procedures*. Each procedure is a tuple $P_i = \langle \mathbf{x}_i, \mathbf{x}_i^{in}, \mathbf{x}_i^{out}, S_i, q_i^{init}, F_i, \Delta_i \rangle$, where $\mathbf{x}_i$ are the *local* variables[2] of P_i ($\mathbf{x}_i \cap \mathbf{x}_j = \varnothing$ for all $i \neq j$), $\mathbf{x}_i^{in}, \mathbf{x}_i^{out} \subseteq \mathbf{x}_i$ are the tuples of input and output variables, S_i are the *control states* of P_i ($S_i \cap S_j = \varnothing$, for all $i \neq j$), $q_i^{init} \in S_i \backslash F_i$ is the *initial*, and $F_i \subseteq S_i$ ($F_i \neq \varnothing$) are the *final* states of P_i, and Δ_i is a set of *transitions* of one of the following forms:

- $q \xrightarrow{\mathcal{R}(\mathbf{x}_i, \mathbf{x}_i')} q'$ is an *internal transition*, where $q, q' \in S_i$, and $\mathcal{R}(\mathbf{x}_i, \mathbf{x}_i')$ is a Presburger arithmetic relation involving only the local variables of P_i;
- $q \xrightarrow{\mathbf{z}' = P_j(\mathbf{u})} q'$ is a *call*, where $q, q' \in S_i$, P_j is the callee, $\mathbf{u}$ are linear terms over $\mathbf{x}_i$, $\mathbf{z} \subseteq \mathbf{x}_i$ are variables, such that $|\mathbf{u}| = |\mathbf{x}_j^{in}|$ and $|\mathbf{z}| = |\mathbf{x}_j^{out}|$. The call is said to be *terminal* if $q' \in F_i$. It is well-known that terminal calls can be replaced by internal transitions.

The *call graph* of a program $\mathcal{P} = \langle P_1, \dots, P_n \rangle$ is a directed graph with vertices $P_1, \dots, P_n$ and an edge (P_i, P_j), for each P_i and P_j, such that P_i has a call to P_j. A program is *recursive* if its call graph has at least one cycle, and *non-recursive* if its call graph is a dag.

In the rest of this paper, we denote by $\mathcal{F}(\mathcal{P}) = \bigcup_{i=1}^{n} F_i$ the set of final states of the program $\mathcal{P}$, by $n\mathcal{F}(P_i)$ the set $S_i \backslash F_i$ of non-final states of P_i, and by $n\mathcal{F}(\mathcal{P}) = \bigcup_{i=1}^{n} n\mathcal{F}(\mathcal{P})$ be the set of non-final states of $\mathcal{P}$.

3.1 Simplified syntax

To ease the description of programs defined in this paper, we use a simplified, human readable, imperative language such that each procedure of the program conforms to the following grammar:[3]

$$P ::= \textbf{proc}\ P_i(id^*)\ \textbf{begin var}\ id^*\ S_0;\ S\ \textbf{end}$$
$$S_0 ::= \textbf{assume}\ f \mid \textbf{goto}\ \ell^+ \mid \textbf{havoc}\ id^+ \mid id \leftarrow t$$
$$S ::= S_0 \mid S;\ S \mid id \leftarrow P_i(t^*);\ S_0 \mid P_i(t^*);\ S_0 \mid \textbf{return}\ id$$

[2] Observe that there are no global variables in the definition of integer program. Those can be encoded as input and output variables to each procedure.

[3] Our simplified syntax does not seek to capture the generality of integer programs. Instead, our goal is to give a convenient notation for the programs given in this paper and only those.

The local variables occurring in P are denoted by id, linear terms by t, Presburger formulae by f, and control labels by ℓ. Each procedure consists in local declarations followed by a sequence of statements. Statements may carry a label. Program statements can be either **assume** statements[4], assignments, procedure calls (possibly with a return value), return to the caller (possibly with a value), non-deterministic jumps **goto** ℓ_1 **or** $\ldots$ **or** ℓ_n, and **havoc** $x_1, x_2, \ldots, x_n$ statements[5]. In order to simplify the upcoming technical developments, we forbid empty procedures, procedures starting with a call or a return, i.e. each procedure must start with a statement generated by the S_0 nonterminal. We consider the usual syntactic requirements (used variables must be declared, jumps are well defined, no jumps outside procedures, etc.). We do not define them, it suffices to know that all simplified programs in this paper comply with the requirements. A program using the simplified syntax can be easily translated into the formal syntax (Fig. 1).

Example 3.1. *Figure 1 shows a program in our simplified imperative language and its corresponding integer program $\mathcal{P}$. Formally, $\mathcal{P} = \langle P \rangle$, where P is the only procedure in the program, defined as:*

$$P \quad = \quad \langle \{x, z\}, \{x\}, \{z\}, \{q_1^{init}, q_2, q_3, \varepsilon\}, q_1^{init}, \{\varepsilon\}, \{t_1, t_2, t_3, t_4\} \rangle$$

Since P calls itself once (within the call transition t_2), this program is recursive. ■

3.2 Semantics

We are interested in computing the *summary relation* between the values of the input and output variables of a procedure. To this end, we give the semantics of a program $\mathcal{P} = \langle P_1, \ldots, P_n \rangle$ as a tuple of relations, denoted $[\![q]\!]$ in the following, describing, for each non-final control state $q \in n\mathcal{F}(P_i)$ of a procedure P_i, the effect of the program when started in q upon reaching a state in F_i. The summary of a procedure P_i is the relation corresponding to its unique initial state, i.e. $[\![q_i^{init}]\!]$.

An *interprocedurally valid path* is represented by a tagged word over an alphabet $\widehat{\Theta}$, which maps each internal transition t to a symbol τ, and each call transition t to a pair of symbols $\langle\!\langle \tau, \tau \rangle\!\rangle \in \widehat{\Theta}$. In the sequel, we denote by Q the nonterminal corresponding to the control state q, and by $\tau \in \Theta$ the alphabet symbol corresponding to the transition t of $\mathcal{P}$. Formally, we associate $\mathcal{P}$ a visibly pushdown grammar, denoted in the rest of the paper by $G_{\mathcal{P}} = \langle \Xi, \widehat{\Theta}, \Delta \rangle$, such that $Q \in \Xi$ if and only if $q \in n\mathcal{F}(\mathcal{P})$ and:

(a) $Q \to \tau \in \Delta$ if and only if $t\colon q \xrightarrow{\mathcal{R}} q'$ and $q' \in \mathcal{F}(\mathcal{P})$

(b) $Q \to \tau\ Q' \in \Delta$ if and only if $t\colon q \xrightarrow{\mathcal{R}} q'$ and $q' \in n\mathcal{F}(\mathcal{P})$

(c) $Q \to \langle\!\langle \tau\ Q_j^{init}\ \tau \rangle\!\rangle\ Q' \in \Delta$ if and only if $t\colon q \xrightarrow{\mathbf{z}' = P_j(\mathbf{u})} q'$.

It is easily seen that interprocedurally valid paths in $\mathcal{P}$ and tagged words in $G_{\mathcal{P}}$ are in one-to-one correspondence. In fact, each interprocedurally valid path of $\mathcal{P}$ between state $q \in n\mathcal{F}(P_i)$ and a state of F_i, where $1 \leqslant i \leqslant n$, corresponds exactly to one tagged word of $L_Q(G_{\mathcal{P}})$.

[4] **assume** ϕ is executable if and only if the current values of the variables satisfy the Presburger formula ϕ.

[5] **havoc** assigns non deterministically chosen integers to $x_1, x_2, \ldots, x_n$.

Example 3.2. *(contd. from Ex. 3.1) The visibly pushdown grammar $G_{\mathcal{P}}$ corresponding to $\mathcal{P}$ is given in Fig. 1 (c). In the following, we use superscripts a, b, c to distinguish productions of the form (a) $Q \to \tau$, (b) $Q \to \tau\, Q'$ or (c) $Q \to \langle\!\langle \tau\, Q_j^{init}\, \tau \rangle\!\rangle Q'$, respectively. The language $L_{Q_1^{init}}(G_{\mathcal{P}})$ generated by $G_{\mathcal{P}}$ starting with Q_1^{init} contains the word $w = \tau_1 \langle\!\langle \tau_2 \tau_1 \langle\!\langle \tau_2 \tau_4 \tau_2 \rangle\!\rangle \tau_3 \tau_2 \rangle\!\rangle \tau_3$, of which $w\_nw(w) = (\tau_1\tau_2\tau_1\tau_2\tau_4\tau_2\tau_3\tau_2\tau_3, \{2 \rightsquigarrow 8, 4 \rightsquigarrow 6\})$ is the corresponding nested word. The word w corresponds to an interprocedurally valid path where P calls itself twice. The control words $\gamma_1 = p_1^b p_2^c p_1^b p_2^c p_4^a p_3^a p_3^a$ and $\gamma_2 = p_1^b p_2^c p_3^a p_1^b p_2^c p_4^a p_3^a$ both produce w in this case, i.e. $Q_1^{init} \overset{\gamma_1}{\Longrightarrow} w$ and $Q_1^{init} \overset{\gamma_2}{\Longrightarrow} w$.* ■

The semantics of a program is the union of the semantics of the nested words corresponding to its executions, each of which being a relation over input and output variables. To define the semantics of a nested word, we first associate to each $\tau \in \widehat{\Theta}$ an integer relation ρ_τ, defined as follows:

- for an internal transition $t\colon q \xrightarrow{\mathcal{R}} q' \in \Delta_i$, we define $\rho_\tau \equiv \mathcal{R}(\mathbf{x}_i, \mathbf{x}_i') \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$;
- for a call transition $t : q \xrightarrow{\mathbf{z}' = P_j(\mathbf{u})} q' \in \Delta_i$, we define a *call relation* $\rho_{\langle\tau} \equiv (\mathbf{x}_j^{in\prime} = \mathbf{u}) \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_j}$, a *return relation* $\rho_{\tau\rangle} \equiv (\mathbf{z}' = \mathbf{x}_j^{out}) \subseteq \mathbb{Z}^{\mathbf{x}_j} \times \mathbb{Z}^{\mathbf{x}_i}$ and a *frame relation* $\phi_\tau \equiv \bigwedge_{x \in \mathbf{x}_i \setminus \mathbf{z}} x' = x \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$. Intuitively, the frame relation copies the values of all local variables, that are not involved in the call as return value receivers ($\mathbf{z}$), across the call.

We define the semantics of the program $\mathcal{P} = \langle P_1, \ldots, P_n \rangle$ in a top-down manner. Assuming a fixed ordering of the non-final states in the program, i.e. $n\mathcal{F}(\mathcal{P}) = \langle q_1, \ldots, q_m \rangle$, the semantics of the program $\mathcal{P}$, denoted $[\![\mathcal{P}]\!]$, is the tuple of relations $\langle [\![q_1]\!], \ldots, [\![q_m]\!] \rangle$. For each non-final control state $q \in n\mathcal{F}(P_i)$ where $1 \leqslant i \leqslant n$, we denote by $[\![q]\!] \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$ the relation (over the local variables of procedure P_i) defined as $[\![q]\!] = \bigcup_{\alpha \in L_Q(G_{\mathcal{P}})} [\![\alpha]\!]$.

It remains to define $[\![\alpha]\!]$, the semantics of the tagged word (or equivalently interprocedural valid path) α. Out of convenience, we define the semantics of its corresponding nested word $w\_nw(\alpha) = (\theta, \rightsquigarrow)$ over alphabet Θ, and define $[\![\alpha]\!] = [\![w\_nw(\alpha)]\!]$. For a nesting relation $\rightsquigarrow \subseteq \{1, \ldots, |\theta|\} \times \{1, \ldots, |\theta|\}$, we define $\rightsquigarrow_{i,j} = \{(s - (i{-}1), t - (i{-}1)) \mid (s,t) \in \rightsquigarrow \cap \{i, \ldots, j\} \times \{i, \ldots, j\}\}$, for some $i, j \in \{1, \ldots, \ell\}$, $i < j$. Finally, we define $[\![(\theta, \rightsquigarrow)]\!] \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$ as follows:

$$\begin{cases} \rho_{(\theta)_1} & \text{if } |\theta| = 1 \\ \rho_{(\theta)_1} \circ [\![((\theta)_{2\ldots|\theta|}, \rightsquigarrow_{2,|\theta|})]\!] & \text{if } |\theta| > 1,\ 1 \rightsquigarrow j \text{ for no } j \\ CaRet_\theta^j \circ [\![((\theta)_{j+1\ldots|\theta|}, \rightsquigarrow_{j+1,|\theta|})]\!] & \text{if } |\theta| > 1,\ 1 \rightsquigarrow j \text{ for a } j \end{cases}$$

where, in the last case, which corresponds to call transition $t \in \Delta_i$, we have $(\theta)_1 = (\theta)_j = \tau$ and define $CaRet_\theta^j = \big(\rho_{\langle\tau} \circ [\![(\theta)_{2\ldots j-1}, \rightsquigarrow_{2,j-1})]\!] \circ \rho_{\tau\rangle}\big) \cap \phi_\tau$.

Example 3.3. *(contd. from Ex. 3.2) The semantics of a given the nested word $\theta = (\tau_1\tau_2\tau_1\tau_2\tau_4\tau_2\tau_3\tau_2\tau_3, \{2 \rightsquigarrow 8, 4 \rightsquigarrow 6\})$ is a relation between valuations of $\{x, z\}$, given by:*

$$\begin{aligned} [\![\theta]\!] \quad = \quad & \rho_{\tau_1} \circ \big((\rho_{\langle\tau_2} \circ \rho_{\tau_1} \circ \big((\rho_{\langle\tau_2} \circ \rho_{\tau_4} \circ \rho_{\tau_2\rangle}) \cap \phi_{\tau_2}\big) \\ & \circ \rho_{\tau_3} \circ \rho_{\tau_2\rangle}) \cap \phi_{\tau_2}\big) \circ \rho_{\tau_3} \end{aligned}$$

One can verify that $[\![\theta]\!] \equiv x = 2 \wedge z' = 4$, i.e. the result of calling P with input valuation $x = 2$ is an output valuation $z = 4$. ■

Finally, we introduce a few useful notations. An interprocedural valid path α is said to be *feasible* whenever $[\![\alpha]\!] \neq \varnothing$. We denote by $[\![\mathcal{P}]\!]_q$ the component of $[\![\mathcal{P}]\!]$ corresponding to $q \in n\mathcal{F}(\mathcal{P})$. Notice that $[\![\mathcal{P}]\!]_q \in \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$, i.e. is a relation over the valuations of the local variables of the procedure P_i if q is a state of P_i, i.e. $q \in S_i$. Slightly abusing notations, we define $L_{P_i}(G_{\mathcal{P}})$ as $L_{Q_i^{init}}(G_{\mathcal{P}})$ and $[\![\mathcal{P}]\!]_{P_i}$ as $[\![\mathcal{P}]\!]_{q_i^{init}}$. Clearly we have that $[\![\mathcal{P}]\!]_{P_i} \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$.

3.3 A Semantics of Depth-First Derivations

We present an alternative, but equivalent, program semantics, using derivations of visibly pushdown program grammars, instead of the generated (nested) words. This semantics brings us closer to the notion of under-approximation defined in the next section.

We start by defining *depth-first derivations*, that have the following informal property: if X and Y are two nonterminals produced by the application of one rule, then the steps corresponding to a full derivation of the form $X \Rightarrow^* u$ will be applied *without interleaving* with the steps corresponding to a derivation of the form $Y \Rightarrow^* v$. In other words, once the derivation of X has started, it will be finished before the derivation of Y begins.

For an integer tuple $\alpha = \langle \alpha_1, \ldots, \alpha_n \rangle$, we denote by $\|\alpha\|_{\max} = \max_{i=1}^{n} \alpha_i$. For a set of symbols $S \subseteq \Xi \cup \Sigma$, and a set of positive integers $I \subseteq \mathbb{N}$, we define $S^I = \{x^{\langle i \rangle} \mid x \in S, i \in I\}$. Given a word $w \in (\Xi \cup \Sigma)^*$ of length $n \geqslant 0$, and a n-dimensional vector $\alpha = \langle \alpha_1, \ldots, \alpha_n \rangle \in \mathbb{N}^n$, we define w^α as the *birthdate-annotated word* (bd-word) $(w)_1{}^{\langle \alpha_1 \rangle} \ldots (w)_n{}^{\langle \alpha_n \rangle}$ over the alphabet $(\Xi \cup \Sigma)^{\mathbb{N}}$. We denote $w^{\langle\langle c \rangle\rangle} = w^{\mathbf{c}}$, where $c \in \mathbb{N}$ and $\mathbf{c} = \langle c, \ldots, c \rangle \in \mathbb{N}^{|w|}$. For instance, $abc^{\langle 1,2,3 \rangle} = a^{\langle 1 \rangle} b^{\langle 2 \rangle} c^{\langle 3 \rangle}$ and $abc^{\langle\langle 2 \rangle\rangle} = a^{\langle 2 \rangle} b^{\langle 2 \rangle} c^{\langle 2 \rangle}$.

Let $G = \langle \Xi, \Sigma, \Delta \rangle$ be a grammar and $u \xRightarrow{(Z,w)/j} v$ be a step, for some production $(Z, w) \in \Delta$ and $1 \leqslant j \leqslant |u|$. If $\alpha \in \mathbb{N}^{|u|}$ is a vector of birthdates, the corresponding *birthdate-annotated step* (bd-step) is defined as follows: $u^\alpha \xRightarrow{(Z,w)/j} v^\beta$ if and only if $(u^\alpha)_j = Z^{\langle i \rangle}$ and $v^\beta = (u^\alpha)_1 \cdots (u^\alpha)_{j-1} \cdot w^{\langle\langle \|\alpha\|_{\max}+1 \rangle\rangle} \cdot (u^\alpha)_{j+1} \cdots (u^\alpha)_{|u|}$.

Example 3.4. *Consider the grammar* $G = \langle \{X, Y, Z\}, \{a, b\}, \Delta \rangle$ *with rules* $\Delta = \{X \to YZ,\ Y \to aY \mid \varepsilon,\ Z \to Zb \mid \varepsilon\}$. *Then* $X^{\langle 0 \rangle} \xRightarrow{(X,YZ)} Y^{\langle 1 \rangle} Z^{\langle 1 \rangle} \xRightarrow{(Y,aY)} a^{\langle 2 \rangle} Y^{\langle 2 \rangle} Z^{\langle 1 \rangle} \xRightarrow{(Z,Zb)} a^{\langle 2 \rangle} Y^{\langle 2 \rangle} Z^{\langle 3 \rangle} b^{\langle 3 \rangle} \xRightarrow{(Y,\varepsilon)} a^{\langle 2 \rangle} Z^{\langle 3 \rangle} b^{\langle 3 \rangle} \xRightarrow{(Z,\varepsilon)} a^{\langle 2 \rangle} b^{\langle 3 \rangle}$ *and* $X^{\langle 0 \rangle} \xRightarrow{(X,YZ)} Y^{\langle 1 \rangle} Z^{\langle 1 \rangle} \xRightarrow{(Y,aY)} a^{\langle 2 \rangle} Y^{\langle 2 \rangle} Z^{\langle 1 \rangle} \xRightarrow{(Y,\varepsilon)} a^{\langle 2 \rangle} Z^{\langle 1 \rangle} \xRightarrow{(Z,Zb)} a^{\langle 2 \rangle} Z^{\langle 3 \rangle} b^{\langle 3 \rangle} \xRightarrow{(Z,\varepsilon)} a^{\langle 2 \rangle} b^{\langle 3 \rangle}$ *are birthdate-annotated step sequences.* ■

A birthdate annotated step is further said to be *depth-first* whenever, in the above definition of a bd-step, we have, moreover, that i is the most recent birthdate among the nonterminals of u , i.e. $i = \max\{j \mid Pk_{\Xi\{j\}}(u^\alpha) \neq \mathbf{0}\}$. We write this fact as follows $u^\alpha \underset{\mathbf{df}}{\Longrightarrow} v^\beta$. A birthdate annotated step sequence is said to be depth-first if all of its steps are depth-first. Finally, a step sequence $w_0 \xRightarrow{(\gamma)_1/j_1} w_1 \ldots w_{n-1} \xRightarrow{(\gamma)_n/j_n} w_n$ for some control word γ is said to be depth-first, written $w_0 \underset{\mathbf{df}}{\overset{\gamma}{\Longrightarrow}} w_n$, if there exist vectors $\alpha_1 \in \mathbb{N}^{\|Pk_\Xi(w_1)\|}, \ldots, \alpha_n \in \mathbb{N}^{\|Pk_\Xi(w_n)\|}$ such that $w_0^{\langle\langle 0 \rangle\rangle} \underset{\mathbf{df}}{\xRightarrow{(\gamma)_1/j_1}} w_1^{\alpha_1} \ldots w_{n-1}^{\alpha_{n-1}} \underset{\mathbf{df}}{\xRightarrow{(\gamma)_n/j_n}} w_n^{\alpha_n}$ holds.

Example 3.5. *(contd. from Ex. 3.4) Consider the grammar G from Example 3.4. Then $X \xRightarrow{(X,YZ)} YZ \xRightarrow{(Y,aY)} aYZ \xRightarrow{(Z,Zb)} aYZb \xRightarrow{(Y,\epsilon)} aZb \xRightarrow{(Z,\epsilon)} ab$ is not a depth-first derivation, whereas $X \xRightarrow{(X,YZ)} YZ \xRightarrow{(Y,aY)} aYZ \xRightarrow{(Y,\epsilon)} aZ \xRightarrow{(Z,Zb)} aZb \xRightarrow{(Z,\epsilon)} ab$ is a depth-first derivation.* ■

Since we are dealing with visibly pushdown grammars $G_{\mathcal{P}} = \langle \Xi, \widehat{\Theta}, \Delta \rangle$ corresponding to programs $\mathcal{P}$, for every production $Q \to \langle\!\langle \tau Q_j^{init} \tau \rangle\!\rangle Q' \in \Delta$ we have $Q_j^{init} \neq Q'$. Hence, we can assume wlog that for all productions $p \in \Delta$, all nonterminals occurring in $tail(p)$ are distinct (e.g. $X \to Z\,Z$ is not allowed). As we show next, under that assumption, a control word uniquely identifies a depth-first derivation:

Lemma 3.1. *Let $G_{\mathcal{P}} = \langle \Xi, \widehat{\Theta}, \Delta \rangle$ be a visibly pushdown grammar corresponding to a program $\mathcal{P}$, $Q \in \Xi$ be a nonterminal, $Q \overset{\gamma}{\underset{df}{\Rightarrow}} u$ and $Q \overset{\gamma}{\underset{df}{\Rightarrow}} v$ be two depth-first derivations of $G_{\mathcal{P}}$. Then they differ in no step, hence $u = v$.*

Proof. By contradiction, suppose that there exists a step that differs in the two derivations from Q with control word $\gamma \in \Delta^*$. Thus, there exists an integer i, $1 \leqslant i < |\gamma|$, such that $Q = w_0 \xRightarrow{(\gamma)_1} w_1 \cdots w_{i-1} \xRightarrow{(\gamma)_i} w_i$ and w_i contains two occurrences of the nonterminal $head((\gamma)_{i+1})$, that is, there exists $p_1 \neq p_2$ $(w_i)_{p_1} = (w_i)_{p_2} = head((\gamma)_{i+1})$. Two cases arise:

1. $(w_i)_{p_1}$ and $(w_i)_{p_2}$ result from the occurrence of some $(\gamma)_j$ with $j \leqslant i$ which contradicts that all nonterminals occurring in $tail((\gamma)_j)$ are distinct.
2. $(w_i)_{p_1}$ and $(w_i)_{p_2}$ result from the occurrence of $(\gamma)_k$ and $(\gamma)_l$ with $k \neq l$ respectively. Hence in the bd-step sequence thereof, their birthdate necessarily differ. Therefore there is only one occurence of $head((\gamma)_{i+1})$ with the most recent birthdate which contradicts the existence of two distinct depth-first derivations.

□ □

Consequently, in a visibly pushdown grammar corresponding to a program, a control word uniquely determines a step sequence, and, moreover, if this step sequence is a derivation, the control word determines the word produced by it. This remark leads to the definition of an alternative semantics of programs, based on control words, instead of produced words. To this end, for each non-final control location $q \in n\mathcal{F}(P_i)$, of a program $\mathcal{P} = \langle P_1, \ldots, P_n \rangle$, where $1 \leqslant i \leqslant n$, we define the semantics of a control word γ that induces a depth-first derivation $Q \overset{\gamma}{\underset{\mathbf{df}}{\Rightarrow}} w$ of the grammar $G_{\mathcal{P}} = \langle \Xi, \widehat{\Theta}, \Delta \rangle$, as a set $[\![\gamma]\!] \subseteq \mathbb{Z}^{\mathbf{x}} \times \mathbb{Z}^{\mathbf{x}}$, where $\mathbf{x} = \mathbf{x}_1 \cdot \ldots \cdot \mathbf{x}_n$ is the set of variables in $\mathcal{P}$. The definition of $[\![\gamma]\!]$ is by induction on the structure of γ:

(a) if $\gamma = Q \to \tau$ then $[\![\gamma]\!] = \{ I \cdot O \mid \langle I{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}} \rangle \in \rho_\tau \}$, where $Q \in \Xi$ corresponds to $q \in n\mathcal{F}(P_i)$;

(b) if $\gamma = (Q \to \tau Q') \cdot \gamma'$ then

$$[\![\gamma]\!] = \{ I \cdot O \mid \exists J \,.\, \langle I{\downarrow_{\mathbf{x}_i}}, J{\downarrow_{\mathbf{x}_i}} \rangle \in \rho_\tau \text{ and } J \cdot O \in [\![\gamma']\!] \}$$

where $Q, Q' \in \Xi$ correspond to $q, q' \in n\mathcal{F}(P_i)$;

(c) if $\gamma = (Q \to \langle\!\langle \tau Q_j^{init} \tau \rangle\!\rangle Q') \cdot \gamma'$ then $[\![\gamma]\!]$ is given by

$$\{I \cdot O \mid \exists J, K, L \in \mathbb{Z}^{\mathbf{x}}. \langle I{\downarrow_{\mathbf{x}_i}}, J{\downarrow_{\mathbf{x}_j}} \rangle \in \rho_{\langle \tau}, \; J \cdot K \in [\![\gamma_1]\!], \\ \langle K{\downarrow_{\mathbf{x}_j}}, L{\downarrow_{\mathbf{x}_i}} \rangle \in \rho_{\tau\rangle}, \langle I{\downarrow_{\mathbf{x}_i}}, L{\downarrow_{\mathbf{x}_i}} \rangle \in \phi_\tau, \; L \cdot O \in [\![\gamma_2]\!]\} \ ,$$

where $Q_j^{init}, Q' \in \Xi$ correspond to q_j^{init} (the initial control location of P_j), $q' \in n\mathcal{F}(P_i)$, and $Q_j^{init} \underset{\mathbf{df}}{\overset{\gamma_1}{\Longrightarrow}} w_1$, $Q' \underset{\mathbf{df}}{\overset{\gamma_2}{\Longrightarrow}} w_2$, $\gamma' = \gamma_1\gamma_2$, respectively; since γ is the control word of a depth-first derivation, the derivations of Q_j^{init} and Q' are unique, and will not interleave with each other.

The following lemma proves the equivalence of the semantics of a (tagged) word generated by a visibly pushdown grammar and that of a control word that produces it.

Lemma 3.2. *Let $G_{\mathcal{P}} = \langle \Xi, \widehat{\Theta}, \Delta \rangle$ be a visibly pushdown grammar for a program $\mathcal{P} = \langle P_1, \ldots, P_n \rangle$, $\mathbf{x} = \mathbf{x}_1 \cdot \ldots \cdot \mathbf{x}_n$ be the concatenation of all tuples of local variables in $\mathcal{P}$, $Q \in \Xi$ be a nonterminal corresponding to a non-final control location $q \in n\mathcal{F}(P_i)$, and $Q \underset{\boldsymbol{df}}{\overset{\gamma}{\Longrightarrow}} \alpha$ be a depth-first derivation of $G_{\mathcal{P}}$, where $\alpha \in \widehat{\Theta}^*$ and $\gamma \in \Delta^*$. Then, we have:*

$$[\![\gamma]\!] = \left\{ I \cdot O \in \mathbb{Z}^{\mathbf{x} \times \mathbf{x}} \mid \langle I{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}} \rangle \in [\![\alpha]\!] \right\} \ .$$

Proof. By induction on $|\gamma| > 0$. If $|\gamma| = 1$, i.e. $\gamma = Q \to \tau$, we have $\alpha = \tau$, hence $[\![\alpha]\!] = [\![w\_nw(\alpha)]\!] = \rho_\tau$ and the equality follows trivially. If $|\gamma| > 1$, let $\gamma = p \cdot \gamma'$, for some $p \in \Delta$ and some $\gamma' \in \Delta^*$. We distinguish two cases, based on the type of p:

- $p = Q \to \tau\, Q'$: in this case $\alpha = \tau \cdot \beta$ and $Q' \underset{\mathbf{df}}{\overset{\gamma'}{\Longrightarrow}} \beta$ is a depth-first derivation of $G_{\mathcal{P}}$. By the induction hypothesis, since $|\gamma'| < |\gamma|$, we have $[\![\gamma']\!] = \left\{ J \cdot O \mid \langle J{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}} \rangle \in [\![\beta]\!] \right\}$.

$$\begin{array}{rcl} [\![\gamma]\!] & = & \{I \cdot O \mid \exists J \,.\, \langle I{\downarrow_{\mathbf{x}_i}}, J{\downarrow_{\mathbf{x}_i}} \rangle \in \rho_\tau \text{ and } \langle J{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}} \rangle \in [\![\beta]\!]\} \\ & = & \{I \cdot O \mid \langle I{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}} \rangle \in [\![w\_nw(\alpha)]\!]\} \\ & = & \{I \cdot O \mid \langle I{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}} \rangle \in [\![\alpha]\!]\} \end{array}$$

- $p = Q \to \langle\!\langle \tau\, Q_j^{init}\, \tau \rangle\!\rangle Q'$: in this case $\alpha = \langle\!\langle \tau\, \beta_1\, \tau \rangle\!\rangle \beta_2$ and $G_{\mathcal{P}}$ has depth-first derivations $Q_j^{init} \underset{\mathbf{df}}{\overset{\gamma_1}{\Longrightarrow}} \beta_1$ and $Q' \underset{\mathbf{df}}{\overset{\gamma_2}{\Longrightarrow}} \beta_2$. We have two symmetrical cases: either $\gamma' = \gamma_1\gamma_2$ or $\gamma' = \gamma_2\gamma_1$. We consider the first case in the following:

$$\begin{array}{rcll} [\![\gamma]\!] & = & \{I \cdot O \mid & \exists J, K, L \in \mathbb{Z}^{\mathbf{x}} \,.\, \langle I{\downarrow_{\mathbf{x}_i}}, J{\downarrow_{\mathbf{x}_j}} \rangle \in \rho_{\langle \tau}, \\ & & & J \cdot K \in [\![\gamma_1]\!], \; \langle K{\downarrow_{\mathbf{x}_j}}, \; L{\downarrow_{\mathbf{x}_i}} \rangle \in \rho_{\tau\rangle}, \\ & & & \langle I{\downarrow_{\mathbf{x}_i}}, L{\downarrow_{\mathbf{x}_i}} \rangle \in \phi_\tau, \; L \cdot O \in [\![\gamma_2]\!]\} \end{array}$$

We apply the induction hypothesis to γ_1 and γ_2, since $|\gamma_1| < |\gamma|$ and $|\gamma_2| < |\gamma|$, and obtain:

$$\begin{array}{rcll} [\![\gamma]\!] & = & \{I \cdot O \mid & \exists J, K, L \in \mathbb{Z}^{\mathbf{x}} \,.\, \langle I{\downarrow_{\mathbf{x}_i}}, J{\downarrow_{\mathbf{x}_j}} \rangle \in \rho_{\langle \tau}, \\ & & & \langle J{\downarrow_{\mathbf{x}_j}}, K{\downarrow_{\mathbf{x}_j}} \rangle \in [\![\beta_1]\!], \; \langle K{\downarrow_{\mathbf{x}_j}}, \; L{\downarrow_{\mathbf{x}_i}} \rangle \in \rho_{\tau\rangle}, \\ & & & \langle I{\downarrow_{\mathbf{x}_i}}, L{\downarrow_{\mathbf{x}_i}} \rangle \in \phi_\tau, \; \langle L{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}} \rangle \in [\![\beta_2]\!]\} \\ & = & \{I \cdot O \mid & \langle I{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}} \rangle \in [\![w\_nw(\alpha)]\!]\} \\ & = & \{I \cdot O \mid & \langle I{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}} \rangle \in [\![\alpha]\!]\} \end{array}$$

□ □

Consequently, the semantics of a program $\mathcal{P} = \langle P_1, \ldots, P_n \rangle$ can be equivalently defined considering the sets

$$[\![\mathcal{P}]\!]_q = \{\langle I{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}}\rangle \mid I \cdot O \in \bigcup_{Q \underset{\mathbf{df}}{\overset{\gamma}{\Rightarrow}} w} [\![\gamma]\!]\} \ ,$$

for each non-final state $q \in n\mathcal{F}(P_i)$ of the procedure P_i of $\mathcal{P}$.

4 Underapproximating the Program Semantics

In what follows we define context-free language underapproximations by filtering out derivations. In particular, in this section, we define a family of underapproximations of $[\![\mathcal{P}]\!]$, called *bounded-index underapproximations*. Then we show that each k-index underapproximation of the semantics of a (possibly recursive) program $\mathcal{P}$ coincides with the semantics of a non-recursive program computable from $\mathcal{P}$ and k.

4.1 Index-bounded derivations

The central notion of this section are *index-bounded derivations*, i.e. derivations in which each step has a *limited budget* of nonterminals. This notion is the key to our underapproximation method.

For a given integer constant $k > 0$, a word $u \in (\Sigma \cup \Xi)^*$ is said to be of index k, if u contains at most k occurrences of nonterminals (formally, $|u{\downarrow_\Xi}| \leqslant k$). A step $u \Rightarrow v$ is said to be k-indexed, denoted $u \underset{(k)}{\Longrightarrow} v$, if and only if both u and v are of index k. As expected, a step sequence is k-indexed if all its steps are k-indexed. For instance, both derivations from Ex. 3.5 are of index 2.

Lemma 4.1. *For every grammar $G = \langle \Xi, \Sigma, \Delta \rangle$ the following properties hold:*

(1) $\underset{(k)}{\Longrightarrow}^* \subseteq \underset{(k+1)}{\Longrightarrow}^*$ *for all* $k \geqslant 1$

(2) $\Rightarrow \ = \ \bigcup_{k=1}^{\infty} \underset{(k)}{\Longrightarrow}^*$

(3) for all $X, Y \in \Xi$, $XY \underset{(k)}{\Longrightarrow}^* w \in \Sigma^*$ *if and only if there exist* $w_1, w_2 \in \Sigma^*$, *such that* $w = w_1 w_2$ *and either: (i)* $X \underset{(k-1)}{\Longrightarrow}^* w_1$ *and* $Y \underset{(k)}{\Longrightarrow}^* w_2$, *or (ii)* $Y \underset{(k-1)}{\Longrightarrow}^* w_2$ *and* $X \underset{(k)}{\Longrightarrow}^* w_1$.

Proof. The proof of points (1) and (2) follow immediately from the definition of $\underset{(k)}{\Longrightarrow}^*$. Let us now turn to the proof of point (3) (only if). First we define w_1 and w_2. Consider the step sequence $XY \underset{(k)}{\Longrightarrow}^* w$ and look at the last step. It must be of the form $uZv \underset{(k)}{\Longrightarrow}^* uyv = w$, where $u, v, y \in \Sigma^*$, and one of the following must hold: Z has been generated from either X or Y. Suppose that Z stems from Y (the other case is treated similarly). In this case, transitively remove from the step-sequence all the steps transforming the rightmost occurrence of

Y. Hence we obtain a step sequence $XY \underset{(k)}{\Longrightarrow}^* w_1Y$. Then w_2 is the unique word satisfying $w = w_1w_2$. Since $XY \underset{(k)}{\Longrightarrow}^* w_1Y$, by removing the occurrence of Y in rightmost position at every step, we find that $X \underset{(k-1)}{\Longrightarrow}^* w_1$, and we are done. Having Z stemming from X yields $Y \underset{(k-1)}{\Longrightarrow}^* w_2$. For the proof of the other direction (if) assuming (i) (the other case is similar), it is easily seen that $XY \underset{(k)}{\Longrightarrow}^* w_1Y \underset{(k)}{\Longrightarrow}^* w_1w_2$. □ □

The previous definitions extend naturally to bd-steps and bd-step sequences, and we define $\Upsilon^{(k)} = \{w^\beta \in \big((\Xi \cup \Sigma)^{\mathbb{N}}\big)^* \mid |w^\beta{\downarrow_{\Xi^{\mathbb{N}}}}| \leqslant k\}$ the set of bd-words with at most k occurrences of nonterminals. We write the fact that a bd-step sequence $u^\alpha \Rightarrow^* v^\beta$ is both k-indexed and depth-first as $u^\alpha \underset{\mathbf{df}(k)}{\Longrightarrow}^* v^\beta$. For any symbol $X \in \Xi$ and constant $k > 0$, we define the languages:

$$L_X^{(k)}(G) = \{w \in \Sigma^* \mid X \underset{(k)}{\Longrightarrow}^* w\}$$
$$\Gamma^{\mathbf{df}(k)}(G) = \{\gamma \in \Delta^* \mid \exists u^\alpha, v^\beta \in \Upsilon^{(k)} \colon u^\alpha \xrightarrow[\mathbf{df}(k)]{\gamma} v^\beta\} \ .$$

Example 4.1. *(contd. from Ex. 3.2) Inspecting the grammar $G_\mathcal{P}$ from Ex.3.2 reveals that*

$$L_{Q_1^{init}}(G_\mathcal{P}) = \{(\tau_1\langle\!\langle\tau_2)^n\,\tau_4\,(\tau_2\rangle\!\rangle\tau_3)^n \mid n \in \mathbb{N}\} \ .$$

For each value of n we give a 2-index derivation capturing the word: repeat n times the steps

$$Q_1^{init} \xRightarrow{p_1p_2^c} \tau_1\langle\!\langle\tau_2Q_1^{init}\tau_2\rangle\!\rangle Q_3 \xRightarrow{p_3^a} \tau_1\langle\!\langle\tau_2Q_1^{init}\tau_2\rangle\!\rangle\tau_3$$

followed by the step

$$Q_1^{init} \xRightarrow{p_4^a} \tau_4 \ .$$

Therefore the 2-index approximation of $G_\mathcal{P}$ shows that $L_{Q_1^{init}}(G_\mathcal{P}) = L^{(2)}_{Q_1^{init}}(G_\mathcal{P})$. ■

Example 4.2. *(contd. from Ex. 3.5) For the grammar G from Ex. 3.5, we obtain the following control sets:*

$$\begin{array}{rcl} \Gamma^{\mathbf{df}(1)} & = & (Y, aY)^*(Y, \varepsilon) \cup (Z, Zb)^*(Z, \varepsilon) \\ \Gamma^{\mathbf{df}(2)} & = & (X, YZ)(Y, aY)^*(Y, \varepsilon)(Z, Zb)^*(Z, \varepsilon) \cup \\ & & (X, YZ)(Z, Zb)^*(Z, \varepsilon)(Y, aY)^*(Y, \varepsilon) \cup \Gamma^{\mathbf{df}(1)} \ . \blacksquare \end{array}$$

We recall a known result.

Proposition 1 ([20])**.** *For all $k \geqslant 1$, $G = (\Xi, \Sigma, \Delta)$ and $X \in \Xi$, we have $L_X^{(k)}(G) = \hat{L}_X(\Gamma^{\mathbf{df}(k)}, G)$.*

Finally, given $k \geqslant 1$, we define the *k-index semantics* of $\mathcal{P}$ as $[\![\mathcal{P}]\!]^{(k)} = \langle [\![q_1]\!]^{(k)}, \ldots, [\![q_m]\!]^{(k)} \rangle$, where $n\mathcal{F}(\mathcal{P}) = \{q_1, \ldots, q_m\}$ and the k-index semantics of a non-final control state $q \in n\mathcal{F}(P_i)$ of a procedure P_i of the program $\mathcal{P}$ is the relation $[\![q]\!] = [\![\mathcal{P}]\!]_q^{(k)} \subseteq \mathbb{Z}^{\mathbf{x}_i} \times \mathbb{Z}^{\mathbf{x}_i}$, defined as:

$$[\![\mathcal{P}]\!]_q^{(k)} = \{\langle I{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}}\rangle \mid I \cdot O \in \bigcup\nolimits_{Q \xrightarrow[\mathbf{df}(k)]{\gamma} w} [\![\gamma]\!]\} \ .$$

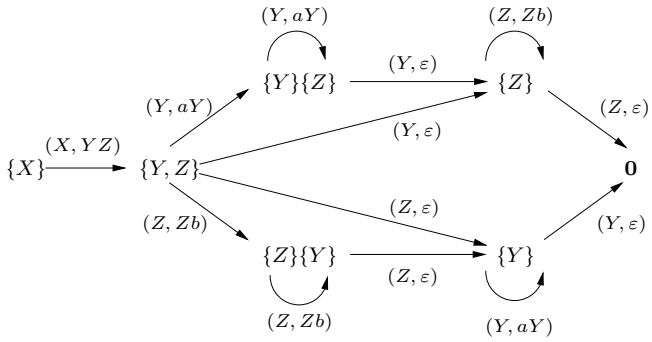

Figure 2: The graph $A^{\mathbf{df}(k)}(G)$ for $k \geqslant 2$ and for the grammar G of Ex. 3.5

4.2 Depth-first index-bounded control sets

For a bd-word w^α, let

$$\left[w^\alpha\right] = Pk_{\Xi\{\|\alpha\|_{\max}\}}(w^\alpha) \cdot Pk_{\Xi\{\|\alpha\|_{\max}-1\}}(w^\alpha) \cdots Pk_{\Xi\{0\}}(w^\alpha) \ .$$

Each symbol in $\left[w^\alpha\right]$ is a $\|\Xi\|$-dimensional vector, that is $\left[w^\alpha\right] \in (\mathbb{N}^{\|\Xi\|})^*$. Therefore with a slight abuse, we can view each of these tuples as a multiset on Ξ. Moreover, each tuple $Pk_{\Xi\{i\}}(w^\alpha)$ in $\left[w^\alpha\right]$ is the multiset of nonterminals that occur in w^α with the same birthdate $0 \leqslant i \leqslant \|\alpha\|_{\max}$, and the elements of $\left[w^\alpha\right]$ are ordered in the reversed order of their birthdates. For instance, the first tuple $Pk_{\Xi\{\|\alpha\|_{\max}\}}(w^\alpha)$ is the multiset of the most recently added nonterminals. Notice that for each bd-word u we have $[u] = \mathbf{0}$ if $u \in (\Sigma^{\mathbb{N}})^*$. Finally, let $\mathbf{0}$ be the identity element for concatenation, i.e. $\left[w^\alpha\right] \cdot \mathbf{0} = \mathbf{0} \cdot \left[w^\alpha\right] = \left[w^\alpha\right]$.

Example 4.3. *(contd. from Ex. 3.5) For the bd-step sequence* $X^{\langle 0\rangle} \Rightarrow Y^{\langle 1\rangle}Z^{\langle 1\rangle} \Rightarrow a^{\langle 2\rangle}Y^{\langle 2\rangle}Z^{\langle 1\rangle} \Rightarrow a^{\langle 2\rangle}Y^{\langle 2\rangle}Z^{\langle 3\rangle}b^{\langle 3\rangle}$ *(Ex. 3.5) we have* $\left[X^{\langle 0\rangle}\right] = \{X\}$, $\left[Y^{\langle 1\rangle}Z^{\langle 1\rangle}\right] = \{Y, Z\}$, $\left[a^{\langle 2\rangle}Y^{\langle 2\rangle}Z^{\langle 1\rangle}\right] = \{Y\} \cdot \{Z\}$ *and* $\left[a^{\langle 2\rangle}Y^{\langle 2\rangle}Z^{\langle 3\rangle}b^{\langle 3\rangle}\right] = \{Z\} \cdot \{Y\}$. ■

The [.] operator is lifted from bd-words to sets of bd-words, i.e. subsets of $\left((\Sigma \cup \Xi)^{\mathbb{N}}\right)^*$. The set $\left[\Upsilon^{(k)}\right]$ is of particular interest in the following developments. Next we define the graph $A^{\mathbf{df}(k)}(G) = \langle \left[\Upsilon^{(k)}\right], (\Delta^*, \cdot), \rightarrow \rangle$, where $\left[\Upsilon^{(k)}\right]$ is the set of vertices, Δ is the set of edge labels and $\rightarrow$ is the edge relation, defined as: $\widetilde{v} \xrightarrow{(Z,w)} \widetilde{w}$ if and only if:

- $\widetilde{v} = (\widetilde{v})_1 \cdot \widetilde{v}_t$, where $(\widetilde{v})_1 \in \mathbb{N}^{\|\Xi\|}$, and $Pk_\Xi(Z) \leqslant (\widetilde{v})_1$, i.e. Z occurs with maximal birthdate $\widetilde{v}$, that is, it occurs in $(\widetilde{v})_1$, and
- $\widetilde{w} = Pk_\Xi(w) \cdot ((\widetilde{v})_1 - Pk_\Xi(Z)) \cdot \widetilde{v}_t$, i.e. Z is removed from its multiset $(\widetilde{v})_1$, and the nonterminals of w are added, with maximal birthdate to obtain $\widetilde{w}$.

Next, define $L(A^{\mathbf{df}(k)}(G)) = \{\gamma \in \Delta^* \mid \widetilde{v} \xrightarrow{\gamma} \widetilde{w} \text{ in } A^{\mathbf{df}(k)}(G)\}$. For example, Fig. 2 shows the $A^{\mathbf{df}(k)}$ graph for the grammar G from Ex. 3.5. The next lemma proves that the paths of $A^{\mathbf{df}(k)}(G)$ represent the control words of the depth-first derivations of G of index k. In the following, we omit the argument G from $\Gamma^{\mathbf{df}(k)}(G)$, or $A^{\mathbf{df}(k)}(G)$, when it is clear from the context.

Lemma 4.2. *Given a grammar $G = \langle \Xi, \Sigma, \Delta \rangle$, and $k > 0$, for each $X \in \Xi$ and $\gamma \in \Delta^*$, there exists a derivation $X \xRightarrow[\mathbf{df}(k)]{\gamma} w$, for some $w \in \Sigma^*$, if and only if $[X] \xrightarrow{\gamma} \mathbf{0}$ in $A^{\mathbf{df}(k)}(G)$.*

Proof. "⇒" We shall prove the following more general statement. Let $u^\alpha \xRightarrow[\mathbf{df}(k)]{\gamma} w^\beta$ be a k-indexed depth-first bd-step sequence. By induction on $|\gamma| \geqslant 0$, we show the existence of a path $[u^\alpha] \xrightarrow{\gamma} \left[w^\beta\right]$ in $A^{\mathbf{df}(k)}$.

For the base case $|\gamma| = 0$, we have $u^\alpha = w^\beta$ which yields $[u^\alpha] = \left[w^\beta\right]$ and since $u^\alpha \in \Upsilon^{(k)}$ by definition of $\Gamma^{\mathbf{df}(k)}$ we have that $[u^\alpha] \in \left[\Upsilon^{(k)}\right]$ and we are done.

For the induction step $|\gamma| > 0$, let $v^\eta \xRightarrow[\mathbf{df}(k)]{(Z,x)} w^\beta$ be the last step of the sequence, for some $(Z, x) \in \Delta$, i.e. $\gamma = \sigma \cdot (Z, x)$ with $\sigma \in \Delta^*$. By the induction hypothesis, $A^{\mathbf{df}(k)}$ has a path $[u^\alpha] \xrightarrow{\sigma} [v^\eta]$. Let $[v^\eta] = \mathbf{v}' \cdot \widetilde{v}_t$, where $\mathbf{v}' = ([v^\eta])_1 \in \mathbb{N}^{|\Xi|}$, and $\widetilde{v}_t \in (\mathbb{N}^{|\Xi|})^*$ is a sequence of multisets of nonterminals. It remains to show that $\left[w^\beta\right] \in \Upsilon^{(k)}$, $Pk_\Xi(Z) \leqslant \mathbf{v}'$ and $\left[w^\beta\right] = Pk_\Xi(x) \cdot (\mathbf{v}' - Pk_\Xi(Z)) \cdot \widetilde{v}_t$ to conclude that $A^{\mathbf{df}(k)}$ has an edge $[v^\eta] \xrightarrow{(Z,x)} \left[w^\beta\right]$, hence a path $[u^\alpha] \xrightarrow{\gamma} \left[w^\beta\right]$.

Since $v^\eta \xRightarrow[\mathbf{df}(k)]{(Z,x)/j} w^\beta$ for some $1 \leqslant j \leqslant |v^\eta|$ we have that $(v^\eta)_j = Z^{\langle i \rangle}$ where $i = \max\{j \mid Pk_{\Xi\{j\}}(v^\eta) \neq \mathbf{0}\}$ and $w^\beta = (v^\eta)_1 \dots (v^\eta)_{j-1} \cdot x^{\langle\langle \|\eta\|_{\max}+1 \rangle\rangle} \cdot (v^\eta)_{j+1} \dots (v^\eta)_{|v^\eta|}$. It is easily seen that $\|\beta\|_{\max} = \|\eta\|_{\max} + 1$. Moreover, since i is the maximal birthdate among the non-terminals of v^η, we have $[v^\eta] = Pk_{\Xi\{i\}}(v^\eta) \dots Pk_{\Xi\{0\}}(v^\eta)$, hence $\mathbf{v}' = Pk_{\Xi\{i\}}(v^\eta)$ and $\widetilde{v}_t = Pk_{\Xi\{i-1\}}(v^\eta) \dots Pk_{\Xi\{0\}}(v^\eta)$. Also we have $Pk_{\Xi\{j\}}(w^\beta) = \mathbf{0}$ for all $j, i < j \leqslant \|\eta\|_{\max}$, $Pk_{\Xi\{i\}}(w^\beta) = Pk_{\Xi\{i\}}(v^\eta) - Pk_{\Xi\{i\}}(Z^{\langle i \rangle})$ and $Pk_{\Xi\{\ell\}}(w^\beta) = Pk_{\Xi\{\ell\}}(v^\eta)$ for all $\ell, 0 \leqslant \ell < i$. Using the foregoing properties of w^β the following equalities are easy to check:

$$
\begin{aligned}
& \left[w^\beta\right] \\
= \;& Pk_{\Xi\{\|\eta\|_{\max}+1\}}(w^\beta) \cdot Pk_{\Xi\{\|\eta\|_{\max}\}}(w^\beta) \dots Pk_{\Xi\{0\}}(w^\beta) \\
= \;& Pk_{\Xi\{\|\eta\|_{\max}+1\}}(w^\beta) \cdot Pk_{\Xi\{i\}}(w^\beta) \cdot Pk_{\Xi\{i-1\}}(w^\beta) \dots Pk_{\Xi\{0\}}(w^\beta) \\
= \;& Pk_\Xi(x) \cdot Pk_{\Xi\{i\}}(w^\beta) \cdot Pk_{\Xi\{i-1\}}(w^\beta) \dots Pk_{\Xi\{0\}}(w^\beta) \\
= \;& Pk_\Xi(x) \cdot (Pk_{\Xi\{i\}}(v^\eta) - Pk_{\Xi\{i\}}(Z^{\langle i \rangle})) \dots Pk_{\Xi\{0\}}(w^\beta) \\
= \;& Pk_\Xi(x) \cdot (\mathbf{v}' - Pk_\Xi(Z)) \cdot Pk_{\Xi\{i-1\}}(w^\beta) \dots Pk_{\Xi\{0\}}(w^\beta) \\
= \;& Pk_\Xi(x) \cdot (\mathbf{v}' - Pk_\Xi(Z)) \cdot Pk_{\Xi\{i-1\}}(v^\eta) \dots Pk_{\Xi\{0\}}(v^\eta) \\
= \;& Pk_\Xi(x) \cdot (\mathbf{v}' - Pk_\Xi(Z)) \cdot \widetilde{v}_t
\end{aligned}
$$

This concludes that $\left[w^\beta\right] = Pk_\Xi(z) \cdot (\mathbf{v}' - Pk_\Xi(Z)) \cdot \widetilde{v}_t$, and since $w^\beta \in \Upsilon^{(k)}$, we obtain that $[v^\eta] \xrightarrow{(Z,x)} \left[w^\beta\right]$ is an edge in $A^{\mathbf{df}(k)}$, and finally that $[u^\alpha] \xrightarrow{\gamma} \left[w^\beta\right]$ is a path in $A^{\mathbf{df}(k)}$.

"⇐" We prove a more general statement. Let $\widetilde{u} \xrightarrow{\gamma} \widetilde{w}$ be a path of $A^{\mathbf{df}(k)}(G)$. We show by induction on $|\gamma|$ that there exist bd-words $u^\alpha, w^\beta \in \Upsilon^{(k)}$, such that $\left[u^\alpha\right] = \widetilde{u}$, $\left[w^\beta\right] = \widetilde{w}$, and $u^\alpha \xRightarrow[\mathbf{df}(k)]{\gamma} w^\alpha$.

The base case $|\gamma| = 0$ is trivial, because $\widetilde{u} = \widetilde{w}$ and since $\widetilde{u} \in \left[\Upsilon^{(k)}\right]$ then there exists $u^\alpha \in \Upsilon^{(k)}$ such that $\left[u^\alpha\right] = \widetilde{u}$, and we are done.

For the induction step $|\gamma| > 0$, let $\gamma = \sigma \cdot (Z, x)$, for some production $(Z, x) \in \Delta$ and $\sigma \in \Delta^*$. By the induction hypothesis, there exist bd-words $u^\alpha, v^\eta \in \Upsilon^{(k)}$

such that $\widetilde{u} = [u^\alpha] \xrightarrow{\sigma} [v^\eta] \xrightarrow{(Z,x)} \widetilde{w}$ is a path in $A^{\mathbf{df}(k)}$, and $u^\alpha \xRightarrow[\mathbf{df}(k)]{\sigma} v^\eta$ is a k-index bd-step sequence. By the definition of the edge relation in $A^{\mathbf{df}(k)}$, it follows that $[v^\eta] = Pk_{\Xi\{i\}}(v^\eta)\cdot\widetilde{v}_t$ where $i = \max\{j \mid Pk_{\Xi\{i\}}(v^\eta) \neq \mathbf{0}\}$. Moreover, there exists j, $1 \leqslant j \leqslant |v^\eta|$ such that $(v^\eta)_j = Z^{\langle i\rangle}$ since $Pk_\Xi(Z) \leqslant Pk_{\Xi\{i\}}(v^\eta)$. Now define $w^\beta = (v^\eta)_1 \dots (v^\eta)_{j-1}\cdot x^{\langle\langle\|\eta\|_{\max}+1\rangle\rangle}\cdot(v^\eta)_{j+1}\dots(v^\eta)_{|v^\eta|}$. It is routine to check $v^\eta \xRightarrow[\mathbf{df}]{(Z,x)/j} w^\beta$ holds. Next we show, $\widetilde{w} = \left[w^\beta\right]$ which concludes the proof.

$$
\begin{aligned}
&\widetilde{w}\\
=&Pk_\Xi(x)\cdot(Pk_{\Xi\{i\}}(v^\eta)-Pk_\Xi(Z))\cdot\widetilde{v}_t\\
=&Pk_{\Xi\{\|\eta\|_{\max}+1\}}(x^{\langle\langle\|\eta\|_{\max}+1\rangle\rangle})\cdot(Pk_{\Xi\{i\}}(v^\eta)-Pk_{\Xi\{i\}}(Z^{\langle i\rangle}))\cdot\widetilde{v}_t\\
=&Pk_{\Xi\{\|\eta\|_{\max}+1\}}(w^\beta)\cdot(Pk_{\Xi\{i\}}(v^\eta)-Pk_{\Xi\{i\}}(Z^{\langle i\rangle}))\cdot\widetilde{v}_t\\
=&Pk_{\Xi\{\|\eta\|_{\max}+1\}}(w^\beta)\cdot(Pk_{\Xi\{i\}}(v^\eta)-Pk_{\Xi\{i\}}(Z^{\langle i\rangle}))\cdot\\
&\qquad Pk_{\Xi\{i-1\}}(v^\eta)\dots Pk_{\Xi\{0\}}(v^\eta)
\end{aligned}
$$

Since $i = \max\{j \mid Pk_{\Xi\{i\}}(v^\eta) \neq \mathbf{0}\}$; $Pk_{\Xi\{\ell\}}(w^\beta) = Pk_{\Xi\{\ell\}}(v^\eta)$ for $0 \leqslant \ell < i$ and $Pk_{\Xi\{i\}}(w^\beta) = Pk_{\Xi\{i\}}(v^\eta)-Pk_{\Xi\{i\}}(Z^{\langle i\rangle})$ show that

$$
\begin{aligned}
=&Pk_{\Xi\{\|\eta\|_{\max}+1\}}(w^\beta)\cdot Pk_{\Xi\{i\}}(w^\beta)Pk_{\Xi\{i-1\}}(w^\beta)\dots Pk_{\Xi\{0\}}(w^\beta)\\
=&Pk_{\Xi\{\|\eta\|_{\max}+1\}}(w^\beta)\cdot Pk_{\Xi\{i\}}(w^\beta)Pk_{\Xi\{i-1\}}(w^\beta)\dots Pk_{\Xi\{0\}}(w^\beta)\\
=&\left[w^\beta\right]
\end{aligned}
$$

□ □

Consequently, we have the following (also proved in [22]):

Corollary 1. *For all $k \geqslant 1$, $G = (\Xi, \Sigma, \Delta)$ and $X \in \Xi$, we have $\Gamma^{\mathbf{df}(k)}$ is regular.*

4.3 Bounded-index Underapproximations of Control Structures

We start describing our program transformation, from a recursive program to a non-recursive program in which all computation traces correspond to words generated by an index-bounded grammar. In the beginning we choose to ignore the data manipulations, and give the non-recursive program only in terms of transitions between control locations and (non-recursive) calls. Then we show that the execution traces of this new program match the depth-first index-bounded derivations of the visibly pushdown grammar of the original program.

Let $\mathcal{P} = \langle P_1, \dots, P_n\rangle$ be a recursive program. For the moment, let us assume that $\mathcal{P}$ has no (local) variables, and thus, all the labels of the internal transitions, as well as all the call, return and frame relations are trivially **true**. As we did previously, we assume a fixed ordering $q_1, \dots, q_m$ on the set $n\mathcal{F}(\mathcal{P})$ of non-final states of $\mathcal{P}$. Let $G_\mathcal{P} = \langle \Xi, \widehat{\Theta}, \Delta\rangle$ be the visibly pushdown grammar associated with $\mathcal{P}$, where each non-final state q of $\mathcal{P}$ is associated a nonterminal $Q \in \Xi$. Then, for a given constant $K > 0$, we define a *non-recursive* program $\mathcal{H}^K$ that captures only the traces of $\mathcal{P}$ corresponding to K-index depth-first derivations of $G_\mathcal{P}$ (Algorithm 1). Formally, we define $\mathcal{H}^K = \langle query^0, query^1, \dots, query^K\rangle$, i.e. the program is structured in $K+1$ procedures, such that:

- $query^0$ consists of a single statement **assume false**, i.e. no execution going through a call of $query^0$ is possible,
- all executions of $query^k$, for each $1 \leqslant k \leqslant K$ correspond to k-index depth-first derivations of $G_{\mathcal{P}}$.

We distinguish between grammar productions of type (a) $Q \to \tau$, (b) $Q \to \tau Q'$ and (c) $Q \to \langle\!\langle \tau, Q_j^{init} \tau \rangle\!\rangle\ Q'$ (see Ex. 3.2) of the visibly pushdown grammar $G = \langle \Xi, \widehat{\Theta}, \Delta \rangle$. Since Ξ and $\widehat{\Theta}$ are finite sets, we associate each nonterminal $Q \in \Xi$ an integer $1 \leqslant \mathcal{I}_Q \leqslant \|\Xi\|$, each alphabet symbol $\tau \in \widehat{\Theta}$ an integer $1 \leqslant \mathcal{I}_\tau \leqslant \|\widehat{\Theta}\|$, and define the productions by the following formulae:

$$\pi_a(x,y) \equiv \bigvee_{(Q \to \tau) \in \Delta} x = \mathcal{I}_Q \wedge y = \mathcal{I}_\tau$$

$$\pi_b(x,y,z) \equiv \bigvee_{(Q \to \tau Q') \in \Delta} x = \mathcal{I}_Q \wedge y = \mathcal{I}_\tau \wedge z = \mathcal{I}_{Q'}$$

$$\pi_c(x,y,z,t,s) \equiv \bigvee_{(Q \to \langle\!\langle \tau Q_j^{init} \tau \rangle\!\rangle Q') \in \Delta} \big(x = \mathcal{I}_Q \wedge y = \mathcal{I}_{\langle\!\langle \tau} \wedge z = \mathcal{I}_{Q_j^{init}} \wedge t = \mathcal{I}_{\tau \rangle\!\rangle} \wedge s = \mathcal{I}_{Q'}\big)$$

It is easy to see that the sizes of the π_a, π_b and π_c formulae are linear in the size of $\mathcal{P}$ (there is one disjunctive clause per production of $G_{\mathcal{P}}$, and each such production corresponds to a transition of $\mathcal{P}$). The translation of $\mathcal{P}$ into $\mathcal{H}$ can hence be implemented as a linear time source-to-source program transformation.

Algorithm 1: proc $query^k$(X) for $1 \leqslant k \leqslant K$

```
    begin
        var PC, Y, Z ;
asgn_0^k:  PC ← X ;
start^k:   goto prod_a^k or prod_b^k or prod_c^k ;
prod_a^k:  assume ∃τ. π_a(PC, τ) ;                          /* Q → τ */
asgn_a^k:  assume true ;
           return;
prod_b^k:  havoc (Y) ;
           assume ∃τ. π_b(PC, τ, Y) ;                       /* Q → τ Q' */
asgn_b^k:  PC ← Y ;
           goto start^k ;
prod_c^k:  havoc (Y,Z) ;
           assume ∃τ, τ'. π_c(PC, τ, Y, τ', Z) ;            /* Q→τ Q_j^init τ' Q' */
ndet^k:    goto swap^k or asgn_c^k ;
swap^k:    swap (Y, Z) ;
asgn_c^k:  PC ← Z ;
           query^{k-1}(Y) ;
           goto start^k;
    end
```

Next, we show a mapping from the paths of $A^{\mathbf{df}(k)}$ onto the feasible interprocedural valid paths of $query^k$. To relate these paths, we need to introduce the

Given $s \in \left[\Upsilon^{(k)}\right] \cup \{sink\}$ and $p \in \Delta$ define $\delta(s,p) = s'$ if $s \xrightarrow{p} s'$ holds in $A^{\mathbf{df}(k)}$ for some s', otherwise ($s \xrightarrow{p} s'$ holds for no s') then $\delta(s,p) = sink$. The output mapping λ is defined as follows:

1. $\lambda(\{X\} \cdot \widetilde{v}, (X, \tau)) = \mathbf{start}^{k-|\widetilde{v}|}\mathbf{prod}_a^{k-|\widetilde{v}|}\mathbf{asgn}_a^{k-|\widetilde{v}|}\mathbf{start}^{k-|\widetilde{v}|+1}$, if $\widetilde{v} \neq \varepsilon$;
2. $\lambda(\{X\}, (X, \tau)) = \mathbf{start}^k\ \mathbf{prod}_a^k\ \mathbf{asgn}_a^k$
3. $\lambda(\{X\} \cdot \widetilde{v}, (X, \tau\, X')) = \mathbf{start}^{k-|\widetilde{v}|}\mathbf{prod}_b^{k-|\widetilde{v}|}\mathbf{asgn}_b^{k-|\widetilde{v}|}\mathbf{start}^{k-|\widetilde{v}|}$
4. $\lambda(\{X\} \cdot \widetilde{v}, (X, \tau\, X_1\, \tau'\, X_2)) = \mathbf{start}^{k-|\widetilde{v}|}\mathbf{prod}_c^{k-|\widetilde{v}|}\mathbf{ndet}^{k-|\widetilde{v}|}$
5. $\lambda(\{Q^{init}, Q'\} \cdot \widetilde{v}, (Q^{init}, \tau\, Q'')) =$
   $\mathbf{asgn}_c^{k-|\widetilde{v}|}\mathbf{asgn}_0^{k-|\widetilde{v}|-1}\mathbf{start}^{k-|\widetilde{v}|-1}\mathbf{prod}_b^{k-|\widetilde{v}|-1}\mathbf{asgn}_b^{k-|\widetilde{v}|-1}\mathbf{start}^{k-|\widetilde{v}|-1}$
6. $\lambda(\{Q^{init}, Q'\} \cdot \widetilde{v}, (Q', \tau)) =$
   $\mathbf{swap}^{k-|\widetilde{v}|}\mathbf{asgn}_c^{k-|\widetilde{v}|}\mathbf{asgn}_0^{k-|\widetilde{v}|-1}\mathbf{start}^{k-|\widetilde{v}|-1}\mathbf{prod}_a^{k-|\widetilde{v}|-1}\mathbf{asgn}_a^{k-|\widetilde{v}|-1}\mathbf{start}^{k-|\widetilde{v}|}$
7. $\lambda(\{Q^{init}, Q'\} \cdot \widetilde{v}, (Q', \tau\, Q'')) =$
   $\mathbf{swap}^{k-|\widetilde{v}|}\mathbf{asgn}_c^{k-|\widetilde{v}|}\mathbf{asgn}_0^{k-|\widetilde{v}|-1}\mathbf{start}^{k-|\widetilde{v}|-1}\mathbf{prod}_b^{k-|\widetilde{v}|-1}\mathbf{asgn}_b^{k-|\widetilde{v}|-1}\mathbf{start}^{k-|\widetilde{v}|-1}$
8. $\lambda(s,p) = \perp$, for all s and p, such that $\delta(s,p) = sink$ holds.

Figure 3: Definition of the mappings δ and λ for SC_Q^k.

notion of gsm mappings.

Definition 1 ([14])**.** *A* generalized sequential machine*, abbreviated gsm, is a* 6*-tuple* $S = \langle K, \Sigma, \Delta, \delta, \lambda, q_1 \rangle$ *where (1)* K *is a finite non-empty set of* states*; (2)* Σ *and* Δ *respectively are* input *and* output *alphabet; (3)* δ *and* λ *are mappings from* $K \times \Sigma$ *into* K *and* Δ^**, respectively; (4)* $q_1 \in K$ *is the* start *state. The functions* δ *and* λ *are extended by induction to* $K \times \Sigma^*$ *by defining for every state* q*,* $x \in \Sigma^*$*, and* $y \in \Sigma$*:*

- $\delta(q, \varepsilon) = q$ *and* $\lambda(q, \varepsilon) = \varepsilon$*.*
- $\delta(q, xy) = \delta(\delta(q, x), y)$ *and* $\lambda(q, xy) = \lambda(q, x)\lambda(\delta(q, x), y)$*.*

The operation defined by $S(x) = \lambda(q_1, x)$ *for each* $x \in \Sigma^*$ *is called a* gsm mapping*.*

We define the gsm $SC_Q^k = \langle \left[\Upsilon^{(k)}\right] \cup \{sink\}, \Delta, \mathcal{L}, \delta, \lambda, [Q] \rangle$ upon $A^{\mathbf{df}(k)}$, where $\mathcal{L}$ denotes the statement labels found in $query^0, \ldots, query^k$; and the mappings δ and λ are given by the rules of Fig. 3.

Lemma 4.3. *For a visibly pushdown grammar* $G = \langle \Xi, \widehat{\Theta}, \Delta \rangle$*, and* $k > 0$*, for each* $Q \in \Xi$ *the set of feasible interprocedural valid paths of* $query^k(Q)$ *coincides with the set* $\{SC_Q^k(\gamma) \mid [Q] \xrightarrow{\gamma} \mathbf{0}$ *in* $A^{\mathbf{df}(k)}\}$*.*

Proof. The feasible interprocedural valid paths of $query^k(Q)$ at Algorithm 1 matches sequences of the form $\sigma_0 \xrightarrow{\delta_0} \sigma_1 \xrightarrow{\delta_1} \ldots \xrightarrow{\delta_{n-1}} \sigma_n$, where each $\sigma_i \in \Xi^*$ is a *stack*, i.e. a possibly empty sequence of frames each containing a snapshot of the value of the local variable PC, $\delta_i \in \Delta$ are productions of G. The sequence of stacks $\sigma_0, \sigma_1, \ldots, \sigma_n$ are snapshots of values of the local variable PC between two consecutive visit to a **start** label or between the last visit to a **start** label and the last **return**. Instances of such consecutive visits are given by $\mathbf{start}^k$, $\mathbf{prod}_a^k$, $\mathbf{asgn}_a^k$; or $\mathbf{start}^k$, $\mathbf{prod}_a^k$, $\mathbf{asgn}_a^k$, **return**, $\mathbf{start}^{k+1}$ (when returning from a previous call); or $\mathbf{start}^k$, $\mathbf{prod}_c^k$, $\mathbf{ndet}^k$, $\mathbf{swap}^k$, $\mathbf{asgn}_c^k$, $\mathbf{start}^{k-1}$ (immediately after entering the call $query^{k-1}$).

When Algorithm 1 is started with a call to $query^k(Q)$, the first stack in the trace is Q. The set of stack sequences are generated by a labelled graph defined

by the following rules, where the stack on both sides of each rule are words $w \in \Xi^*$ such that $|w| \leqslant k$.

(a) $Q \cdot \sigma \xrightarrow{(Q,\tau)} \sigma$

(b) $Q \cdot \sigma \xrightarrow{(Q,\tau Q')} Q' \cdot \sigma$

(c) we have either (i) $Q \cdot \sigma \xrightarrow{(Q,\langle\tau Q'\tau\rangle Q'')} Q' \cdot Q'' \cdot \sigma$, or (ii) $Q \cdot \sigma \xrightarrow{(Q,\langle\tau Q'\tau\rangle Q'')} Q'' \cdot Q' \cdot \sigma$

Following the previous definition, we find that the set of sequences of control labels $\{SC^k_Q(\gamma) \mid Q \xrightarrow{\gamma} \varepsilon\}$ coincides with the feasible interprocedural valid path of $query^k(Q)$.

Next we show that $Q \xrightarrow{\gamma} \varepsilon$ is a valid stack sequence of $query^k(Q)$ if and only if $[Q] \xrightarrow{\gamma} \mathbf{0}$ in $A^{\mathbf{df}(k)}(G)$. For this, consider the following relation between the stacks $\sigma \in \Xi^*$ such that $|\sigma| \leqslant k$ and words $\widetilde{w} \in \left[\Upsilon^{(k)}\right]$: we write $\sigma \lessdot\!\!\lessdot \widetilde{w}$ if and only if exactly one of the following holds:
(1) $|\sigma| = |\widetilde{w}|$ and, for all $1 \leqslant i \leqslant |\widetilde{w}|$: $\{(\sigma)_i\} = (\widetilde{w})_i$, or
(2) $|\sigma| = |\widetilde{w}| + 1$, $(\widetilde{w})_1 = \{(\sigma)_1, (\sigma)_2\}$, and for all $1 < i \leqslant |\widetilde{w}|$: $\{(\sigma)_{i+1}\} = (\widetilde{w})_i$.
The proof goes by induction and shows the following stronger statement relating the reachable stacks and the states of $A^{\mathbf{df}(k)}$ reachable from $[Q]$: for any stack sequence $Q \xrightarrow{\gamma} \sigma$, there exists a path $[Q] \xrightarrow{\gamma} \widetilde{w}$ in $A^{\mathbf{df}(k)}$, such that $\sigma \lessdot\!\!\lessdot \widetilde{w}$, and vice versa.

By putting together the previous result about the feasible interprocedural valid paths of $query^k(Q)$ we find that they coincide with the set $\{SC^k_Q(\gamma) \mid [Q] \xrightarrow{\gamma} \mathbf{0}$ in $A^{\mathbf{df}(k)}\}$. □ □

4.4 Bounded-index Underapproximations of Programs

Algorithm 1 implements the transformation of the control structure of a recursive program $\mathcal{P}$ into a non-recursive program $\mathcal{H}^K = \langle query^0, \ldots, query^K \rangle$, which simulates its K-index derivations (actually, the control words thereof). In this section we extend this construction to programs with integer variables and data manipulations (Algorithm 2), by defining a set of procedures $query^k$, for all $0 \leqslant k \leqslant K$, such that each procedure $query^k$ has five sets of local variables, all of the same cardinality as $\mathbf{x}$: two sets, named $\mathbf{x}_I$ and $\mathbf{x}_O$, are used as input variables, whereas the other three sets, named $\mathbf{x}_J, \mathbf{x}_K$ and $\mathbf{x}_L$ are used locally by $query^k$. Besides, each $query^k$ has local variables called PC, τ, Y, Z and input variable X. There are no output variables in $query^k$. Let $\mathcal{V}^k_{query}$ denote the tuple of local variables of $query^k$, and let $\mathcal{V}^K_{\mathcal{H}} = \mathcal{V}^1_{query} \cdot \ldots \cdot \mathcal{V}^K_{query}$ be the tuple of all variables of $\mathcal{H}^K$.

For two tuples of variables $\mathbf{x}$ and $\mathbf{y}$ of equal length, and a valuation $\nu \in \mathbb{Z}^{\mathbf{x}}$, we denote by $\nu[\mathbf{y}/\mathbf{x}]$ the valuation that maps $(\mathbf{y})_i$ into $(\nu(\mathbf{x}))_i$, for all $1 \leqslant i \leqslant |\mathbf{x}|$. The following lemma is needed in the proof of Thm. 1.

Lemma 4.4. *Let $G_{\mathcal{P}} = \langle \Xi, \widehat{\Theta}, \Delta \rangle$ be a visibly pushdown grammar for a program $\mathcal{P} = \langle P_1, \ldots, P_n \rangle$, let $\mathbf{x} = \mathbf{x}_1 \cdot \ldots \cdot \mathbf{x}_n$ be the tuple of variables in $\mathcal{P}$, and let $\mathcal{H}^K = \langle query^0, \ldots, query^K \rangle$ be the program defined by Algorithm 2. Given a nonterminal $Q \in \Xi$, corresponding to a non-final control state $q \in n\mathcal{F}(\mathcal{P})$, $\gamma \in \Delta^*$,*

Algorithm 2: proc $query^k(\mathrm{X}, \mathbf{x}_I, \mathbf{x}_O)$ for $1 \leqslant k \leqslant K$

begin

var $\mathbf{x}_J, \mathbf{x}_K, \mathbf{x}_L$;
var PC, τ, Y, Z ;
$\mathbf{asgn}_0^k$: PC $\leftarrow X$;
$\mathbf{start}^k$: **goto prod**$_a^k$ **or prod**$_b^k$ **or prod**$_c^k$;
$\mathbf{prod}_a^k$: **havoc** (τ);
assume $\pi_a(\mathrm{PC}, \tau)$; /* $Q \to \tau$ */
$\mathbf{asgn}_a^k$: **assume** $\rho_\tau(\mathbf{x}_I, \mathbf{x}_O)$;
return;
$\mathbf{prod}_b^k$: **havoc** (τ, Y);
assume $\pi_b(\mathrm{PC}, \tau, \mathrm{Y})$; /* $Q \to \tau\, Q'$ */
havoc $(\mathbf{x}_J)$;
assume $\rho_\tau(\mathbf{x}_I, \mathbf{x}_J)$;
$\mathbf{x}_I \leftarrow \mathbf{x}_J$;
$\mathbf{asgn}_b^k$: PC $\leftarrow$ Y;
goto start^k ;
$\mathbf{prod}_c^k$: **havoc** $(\tau, \mathrm{Y}, \mathrm{Z})$;
assume $\pi_c(\mathrm{PC}, \langle\!\langle \tau, \mathrm{Y}, \tau \rangle\!\rangle, \mathrm{Z})$; /* $Q \to \langle\!\langle \tau Q_j^{init} \tau \rangle\!\rangle Q'$ */
havoc $(\mathbf{x}_J, \mathbf{x}_K, \mathbf{x}_L)$;
assume $\rho_{\langle\!\langle \tau}(\mathbf{x}_I, \mathbf{x}_J)$; /* call relation */
assume $\rho_{\tau \rangle\!\rangle}(\mathbf{x}_K, \mathbf{x}_L)$; /* return relation */
assume $\phi_\tau(\mathbf{x}_I, \mathbf{x}_L)$; /* frame relation */
$\mathbf{ndet}^k$: **goto swap**^{k} **or asgn**$_c^k$;
$\mathbf{swap}^k$: swap(Y, Z);
swap($\mathbf{x}_J, \mathbf{x}_L$);
swap($\mathbf{x}_K, \mathbf{x}_O$);
$\mathbf{asgn}_c^k$: $\mathbf{x}_I \leftarrow \mathbf{x}_L$;
PC $\leftarrow$ Z;
$query^{k-1}(\mathrm{Y}, \mathbf{x}_J, \mathbf{x}_K)$;
goto start^k;

end

Example 4.4. *Let us consider an execution of query for the call* $query^2(Q_1^{init}, (\,1\;0\,), (\,1\;2\,))$ *following* $Q_1^{init} \overset{p_1^b p_2^c}{\Longrightarrow} \tau_1\langle\tau_2 Q_1^{init}\tau_2\rangle Q_3 \overset{p_3^a}{\Longrightarrow} \tau_1\langle\tau_2 Q_1^{init}\tau_2\rangle\tau_3 \overset{p_4^a}{\Longrightarrow} \tau_1\langle\tau_2\tau_4\tau_2\rangle\tau_4$. *In the table below, the first row (labelled* PC*) gives the value of local variable* PC *when control hits the labelled statement given at the second row (labelled ip). The third row (labelled* $\mathbf{x}_I/\mathbf{x}_O$*) represents the content of the two arrays.* $\mathbf{x}_I/\mathbf{x}_O = (\,a\;b\,)(\,c\;d\,)$ *says that, in* $\mathbf{x}_I$*,* x *has value* a *and* z *has value* b*; in* $\mathbf{x}_O$*,* x *has value* c *and* z *has value* d*.*

PC	Q_1^{init}	–	Q_2	–	–
ip	$\mathbf{start}^2$	$\mathbf{prod}_b^2\,(p_1^b)$	$\mathbf{start}^2$	$\mathbf{prod}_c^2\,(p_2^c)$	$\mathbf{swap}^2$
$\mathbf{x}_I/\mathbf{x}_O$	$(\,1\;0\,)(\,1\;2\,)$	$(\,1\;0\,)(\,1\;2\,)$	$(\,1\;0\,)(\,1\;2\,)$	$(\,1\;0\,)(\,1\;2\,)$	$(\,1\;0\,)(\,1\;2\,)$
PC	Q_3	–	Q_1^{init}	–	
ip	$\mathbf{start}^1$	$\mathbf{prod}_a^1\,(p_3^a)$	$\mathbf{start}^2$	$\mathbf{prod}_a^2\,(p_4^a)$	
$\mathbf{x}_I/\mathbf{x}_O$	$(\,1\;0\,)(\,1\;2\,)$	$(\,1\;0\,)(\,1\;2\,)$	$(\,0\;0\,)(\,42\;0\,)$	$(\,0\;0\,)(\,42\;0\,)$	

The execution of $query^2(Q_1^{init}, (\,1\;0\,), (\,1\;2\,))$ *starts on row 1, column 1 and proceeds until the call to* $query^1(Q_3, (\,1\;0\,), (\,1\;2\,))$ *at row 2, column 1 (the out of order case). The latter ends at row 2, column 2, where the execution of* $query^2(Q_1^{init}, (\,1\;0\,), (\,1\;2\,))$ *resumes. Since the execution is out of order, and the previous* $\mathbf{havoc}(\mathbf{x}_J, \mathbf{x}_K, \mathbf{x}_L)$ *results into* $\mathbf{x}_J = (\,0\;0\,)$, $\mathbf{x}_K = (\,42\;0\,)$ *and* $\mathbf{x}_L = (\,1\;0\,)$ *(this choice complies with the call relation), the values of* $\mathbf{x}_I/\mathbf{x}_O$ *are updated to* $(\,0\;0\,)/(\,42\;0\,)$. ■

$w \in \widehat{\Theta}^*$*, and* $1 \leqslant k \leqslant K$*, such that* $Q \overset{\gamma}{\underset{\mathbf{df}(k)}{\Longrightarrow}} w$*, we have:*

$$[\![\gamma]\!] = \left\{ \left(I{\downarrow_{\mathbf{x}_I \cdot \mathbf{x}_O}}\right)[\mathbf{x}\cdot\mathbf{x}/\mathbf{x}_I\cdot\mathbf{x}_O] \mid I \cdot O \in [\![SC_Q^k(\gamma)]\!] \right\}$$

where $[\![\gamma]\!] \subseteq \mathbb{Z}^{\mathbf{x}\times\mathbf{x}}$ *and* $[\![SC_Q^k(\gamma)]\!] \subseteq \mathbb{Z}^{\mathcal{V}_{\mathcal{H}}^K}$.

Proof. By induction on $|\gamma| > 0$, applying a case split on the type of the first production in γ. □ □

The following theorem summarizes the first major result in this paper, namely that any K-index underapproximation of the semantics of a recursive program $\mathcal{P}$ can be computed by looking at the semantics of a non-recursive program $\mathcal{H}^K$, obtained from $\mathcal{P}$ by a syntactic source-to-source transformation.

Theorem 1. *Let* $\mathcal{P} = \langle P_1, \ldots, P_n\rangle$ *be a program,* $\mathbf{x} = \mathbf{x}_1 \cdot \ldots \cdot \mathbf{x}_n$ *be the tuple of variables in* $\mathcal{P}$*, and let* $q \in n\mathcal{F}(P_i)$ *be a non-final control state of* $P_i = \langle \mathbf{x}_i, \mathbf{x}_i^{in}, \mathbf{x}_i^{out}, S_i, q_i^{init}, F_i, \Delta_i\rangle$*. Moreover, let* $\mathcal{H}^K = \langle query^0, \ldots, query^K\rangle$ *be the program defined by Algorithm 2. For any* $1 \leqslant k \leqslant K$*, we have:*

$$[\![\mathcal{P}]\!]_q^{(k)} = \{\langle (\widetilde{I}\downarrow_{\mathbf{x}_I}[\mathbf{x}/\mathbf{x}_I])\downarrow_{\mathbf{x}_i}, (\widetilde{I}\downarrow_{\mathbf{x}_O}[\mathbf{x}/\mathbf{x}_O])\downarrow_{\mathbf{x}_i}\rangle \mid \widetilde{I}\cdot\widetilde{O} \in [\![\mathcal{H}^K]\!]_{query^k}, \widetilde{I}(X) = Q\} \ .$$

Proof. Let $G_{\mathcal{P}} = \langle \Xi, \widehat{\Theta}, \Delta\rangle$ be the visibly pushdown grammar corresponding to $\mathcal{P}$. By definition, we have

$$[\![\mathcal{P}]\!]_q^{(k)} = \left\{ \langle I{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}}\rangle \mid I\cdot O \in \bigcup_{Q \overset{\gamma}{\underset{\mathbf{df}(k)}{\Longrightarrow}} w} [\![\gamma]\!] \right\}$$

"$\subseteq$" Let $Q \xRightarrow[\mathbf{df}(k)]{\gamma} w$ be a derivation of $G_{\mathcal{P}}$, and $I \cdot O \in [\![\gamma]\!]$ be a tuple from $\mathbb{Z}^{\mathbf{x}\times\mathbf{x}}$. By Lemma 4.2, $[Q] \xrightarrow{\gamma} \mathbf{0}$ is a path in $A^{\mathbf{df}(k)}(G_{\mathcal{P}})$, and by Lemma 4.3, $SC^k_Q(\gamma)$ is a feasible interprocedurally valid path of $query^k(Q)$. By Lemma 4.4, there exists tuples $\widetilde{I}, \widetilde{O}$ such that $\widetilde{I} \cdot \widetilde{O} \in [\![SC^k_Q(\gamma)]\!]$, and $I \cdot O = \left(\widetilde{I}{\downarrow}_{\mathbf{x}_I \cdot \mathbf{x}_O}\right)[\mathbf{x} \cdot \mathbf{x}/\mathbf{x}_I \cdot \mathbf{x}_O]$. We obtain thus $I = \widetilde{I}\downarrow_{\mathbf{x}_I}[\mathbf{x}/\mathbf{x}_I]$ and $O = \widetilde{I}\downarrow_{\mathbf{x}_O}[\mathbf{x}/\mathbf{x}_O]$.
"$\supseteq$" Let $\widetilde{I}, \widetilde{O} \in \mathbb{Z}^{\mathcal{V}^k_{query}}$, such that $\widetilde{I} \cdot \widetilde{O} \in [\![\mathcal{H}^K]\!]_{query^k}$ and $\widetilde{I}(X) = Q$. Then there exists a feasible interprocedurally valid path π of $query^k(Q)$, such that $\widetilde{I} \cdot \widetilde{O} \in [\![\pi]\!]$. By Lemma 4.3, there exists a control word $\gamma \in \Delta^*$, such that $[Q] \xrightarrow{\gamma} \mathbf{0}$ and $\pi = SC^k_Q(\gamma)$. By Lemma 4.4, $\left(\widetilde{I}{\downarrow}_{\mathbf{x}_I \cdot \mathbf{x}_O}\right)[\mathbf{x} \cdot \mathbf{x}/\mathbf{x}_I \cdot \mathbf{x}_O] \in [\![\gamma]\!]$. By Lemma 4.2, we have that $Q \xRightarrow[\mathbf{df}(k)]{\gamma} w$ is a derivation of $G_{\mathcal{P}}$. We can conclude that $\langle(\widetilde{I}\downarrow_{\mathbf{x}_I}[\mathbf{x}/\mathbf{x}_I])\downarrow_{\mathbf{x}_i}, (\widetilde{I}\downarrow_{\mathbf{x}_O}[\mathbf{x}/\mathbf{x}_O])\downarrow_{\mathbf{x}_i}\rangle \in [\![\mathcal{P}]\!]_q$. □ □

As a last point, we observe that the bounded-index sequence $\{[\![\mathcal{P}]\!]^{(k)}\}_{k=1}^{\infty}$ satisfies several conditions that advocate its use in program analysis, as an underapproximation sequence. The subset order and set union is extended to tuples of relations, point-wise.

$$\begin{array}{lcll} [\![\mathcal{P}]\!]^{(k)} & \subseteq & [\![\mathcal{P}]\!]^{(k+1)} & \text{for all } k \geqslant 1 \quad (A1) \\ [\![\mathcal{P}]\!] & = & \bigcup_{k=1}^{\infty} [\![\mathcal{P}]\!]^{(k)} & (A2) \end{array}$$

Condition $(A1)$ requires that the sequence is monotonically increasing, the limit of this increasing sequence being the actual semantics of the program $(A2)$. These conditions follow however immediately from the two first points of Lemma 4.1. To decide whether the limit $[\![\mathcal{P}]\!]$ has been reached by some iterate $[\![\mathcal{P}]\!]^{(k)}$, it is enough to check that the tuple of relations in $[\![\mathcal{P}]\!]^{(k)}$ is inductive with respect to the statements of $\mathcal{P}$. This can be implemented as an SMT query.

5 Completeness of Index-Bounded Underapproximations for Bounded Programs

In this section we define a class of recursive programs for which the precise summary semantics of each program in that class is effectively computable. We show for each program $\mathcal{P}$ in the class that (a) $[\![\mathcal{P}]\!] = [\![\mathcal{P}]\!]^{(k)}$ for some value $k \geqslant 1$, bounded by a linear function in the total number $\mathrm{loc}(\mathcal{P})$ of control states in $\mathcal{P}$, and moreover (b) the semantics of $\mathcal{H}^k$ is effectively computable (and so is that of $[\![\mathcal{P}]\!]^{(k)}$ by Thm. 1).

Given an integer relation $R \subseteq \mathbb{Z}^n \times \mathbb{Z}^n$, its *transitive closure* $R^+ = \bigcup_{i=1}^{\infty} R^i$, where $R^1 = R$ and $R^{i+1} = R^i \circ R$, for all $i \geqslant 1$. In general, the transitive closure of a relation is not definable within decidable subsets of integer arithmetic, such as Presburger arithmetic. In this section we consider two classes of relations, called *periodic*, for which this is possible, namely octagonal relations, and finite monoid affine relations.

Octagonal relation An *octagonal relation* is defined by a finite conjunction of constraints of the form $\pm x \pm y \leqslant c$, where x and y range over the set $\mathbf{x} \cup \mathbf{x}'$, and c is an integer constant. The transitive closure of any octagonal relation has been shown to be Presburger definable and effectively computable [8].

Linear affine relation A *linear affine relation* is defined by a formula $\mathcal{R}(\mathbf{x}, \mathbf{x}') \equiv C\mathbf{x} \geqslant \mathbf{d} \ \wedge \ \mathbf{x}' = A\mathbf{x} + \mathbf{b}$, where $A \in \mathbb{Z}^{n\times n}$, $C \in \mathbb{Z}^{p\times n}$ are matrices and $\mathbf{b} \in \mathbb{Z}^n$, $\mathbf{d} \in \mathbb{Z}^p$. $\mathcal{R}$ is said to have the *finite monoid property* if and only if the set $\{A^i \mid i \geqslant 0\}$ is finite. It is known that the finite monoid condition is decidable [7], and moreover that the transitive closure of a finite monoid affine relation is Presburger definable and effectively computable [12, 7].

We define a *bounded-expression* $\mathbf{b}$ to be a regular expression of the form $\mathbf{b} = w_1^* \dots w_d^*$, where $d \geqslant 1$ and each w_i is a non-empty word. A language (not necessarily context-free) L over alphabet Σ is said to be *bounded* if and only if L is included in (the language of) a bounded expression $\mathbf{b}$.

Theorem 2 ([21]). *Let $G = (\Xi, \Sigma, \Delta)$ be a grammar, and $X \in \Xi$ be a nonterminal, such that $L_X(G)$ is bounded. Then there exists a linear function $\mathcal{B}\colon \mathbb{N} \to \mathbb{N}$ such that $L_X(G) = L_X^{(k)}(G)$ for some $1 \leqslant k \leqslant \mathcal{B}(\|\Xi\|)$.*

If the grammar in question is $G_{\mathcal{P}}$, for a program $\mathcal{P}$, then clearly $\|\Xi\|$ is bounded by the number of control locations in $\mathcal{P}$, by the definition of $G_{\mathcal{P}}$. The class of programs for which our method is complete is defined below:

Definition 2. *Let $\mathcal{P}$ be a program and $G_{\mathcal{P}} = (\Xi, \widehat{\Theta}, \Delta)$ be its corresponding visibly pushdown grammar. Then $\mathcal{P}$ is said to be* bounded periodic *if and only if:*

1. *$L_X(G_{\mathcal{P}})$ is bounded for each $X \in \Xi$;*
2. *each relation ρ_τ occurring in the program, for some $\tau \in \widehat{\Theta}$, is periodic.*

Example 5.1. *(continued from Ex. 4.1) Recall that $L_{Q_1^{init}}(G_{\mathcal{P}}) = L^{(2)}_{Q_1^{init}}(G_{\mathcal{P}})$ which equals to the set $\{(\tau_1\langle\!\langle\tau_2)^n\tau_4(\tau_2\rangle\!\rangle\tau_3)^n \mid n \geqslant 0\} \subseteq (\tau_1\tau_2\langle\!\langle)^*\tau_4^*(\tau_2\rangle\!\rangle\tau_3)^*$.* ■

Concerning condition 1, it is decidable [14] and previous work [16] defined a class of programs following a recursion scheme which ensures boundedness of the set of interprocedurally valid paths.

This section shows that the underapproximation sequence $\{[\![\mathcal{P}]\!]^{(k)}\}_{k=1}^{\infty}$, defined in Section 4, when applied to any bounded periodic programs $\mathcal{P}$, always yields $[\![\mathcal{P}]\!]$ in at most $\mathcal{B}(\mathrm{loc}(\mathcal{P}))$ steps, and moreover each iterate $[\![\mathcal{P}]\!]^{(k)}$ is computable and Presburger definable. Furthermore the method can be applied *as it is* to bounded periodic programs, without prior knowledge of the bounded expression $\mathbf{b} \supseteq L_Q(G_{\mathcal{P}})$.

The proof goes as follows. Because $\mathcal{P}$ is bounded periodic, Thm. 2 shows that the semantics $[\![\mathcal{P}]\!]$ of $\mathcal{P}$ coincide with its k-index semantics $[\![\mathcal{P}]\!]^{(k)}$ for some $1 \leqslant k \leqslant \mathcal{B}(\mathrm{loc}(\mathcal{P}))$. Hence, the result of Thm. 1 shows that for each $q \in n\mathcal{F}(\mathcal{P})$, the k-index semantics $[\![\mathcal{P}]\!]_q^{(k)} = \{\langle I{\downarrow_{\mathbf{x}_I}}, I{\downarrow_{\mathbf{x}_O}}\rangle \mid I \cdot O \in [\![\mathcal{H}^K]\!]_{query^k}, I(X) = Q\}$, that is, the semantics $[\![\mathcal{P}]\!]_q^{(k)}$ is computed from that of procedure $query^k$ called with $X = Q$. Then, because $\mathcal{P}$ is bounded, we show in Thm. 3 that every procedure $query^k$ of program $\mathcal{H}$ is *flattable* (Def. 3). Moreover, since the only transitions of $\mathcal{H}$ which are not from $\mathcal{P}$ are equalities and **havoc**, all transitions of $\mathcal{H}$ are periodic. Since each procedure $query^k$ is flattable then $[\![\mathcal{P}]\!]$ is computable in finite time by existing tools, such as Fast [6] or Flata [9, 8]. In fact, these tools are guaranteed to terminate provided that (*a*) the input program is flattable; and (*b*) loops are labelled with periodic relations.

Definition 3. *Let $\mathcal{P} = \langle P_1, \dots, P_n\rangle$ be a non-recursive program and $G_{\mathcal{P}} = (\Xi, \widehat{\Theta}, \Delta)$ be its corresponding visibly pushdown grammar. Procedure P_i is said*

to be flattable *if and only if there exists a bounded and regular language R over $\widehat{\Theta}$, such that $[\![\mathcal{P}]\!]_{P_i} = \bigcup_{\alpha \in L_{P_i}(G_\mathcal{P}) \cap R} [\![\alpha]\!]$.*

Notice that a flattable program is not necessarily bounded (Def. 2), but its semantics can be computed by looking only at a bounded subset of interprocedurally valid paths.

The proof that the procedures $query^k$ are flattable relies on grammar based reasoning, and, in particular, on control-sets with relative completeness properties. Let us now turn to our main result, Theorem 3 stated next, whose proof is organized as follows. First, Proposition 2 roughly states that provided $L(G)$ is bounded, then a bounded subset of the k-index depth-first derivations suffices to capture $L^{(k)}(G)$ for some k. The proof of this proposition is split into Theorem 4, Lemma 5.1 and Lemma 5.2. The rest of the proof uses Lemma 4.3 which roughly states that there is a well-behaved mapping from the k-index depth-first derivations of $G_\mathcal{P}$ from Q to the runs of $query^k(Q)$ for every value of k and Q.

Theorem 3. *Let $\mathcal{P} = \langle P_1, \ldots, P_n \rangle$ be a bounded program, then, for any $k \geqslant 1$, procedure $query^k$ of program $\mathcal{H}$ is flattable.*

5.1 Bounded languages with bounded control sets

The following result was proved in [13]:

Theorem 4 (Thm. 1 from [13], also in [20])**.** *For every regular language L over alphabet Σ there exists a bounded expression $\mathbf{b}_\Gamma$ such that $Pk_\Sigma(L \cap \mathbf{b}_\Gamma) = Pk_\Sigma(L)$.*

Next we prove a result characterizing a subset of derivations sufficient to capture a bounded context-free language. But first, given a grammar $G = (\Xi, \Sigma, \Delta)$ and $X \in \Xi$ define

$$\Gamma_X^{\mathbf{df}(k)} = \{\gamma \in \Delta^* \mid [X] \xrightarrow{\gamma} \mathbf{0} \text{ in } A^{\mathbf{df}(k)}\} \ .$$

Observe that $\Gamma_X^{\mathbf{df}(k)}$ is a regular language, because $A^{\mathbf{df}(k)}$ is a finite state automaton.

Lemma 5.1. *Let $G = (\Xi, \Sigma, \Delta)$ be a grammar and $X \in \Xi$ be a nonterminal, such that for all $p \in \Delta$, X does not occur in $tail(p)$. Also $L_X(G) \subseteq (a_1 w_1)^* \ldots (a_d w_d)^*$ where $a_1, \ldots, a_d$ are distinct symbols of Σ none of which occurs in $w_1 \cdots w_d$. Then, for each $k \geqslant 1$ there exists a bounded expression $\mathbf{b}_\Gamma$ over Δ such that $L_X^{(k)}(G) = \hat{L}_X(\mathbf{b}_\Gamma \cap \Gamma_X^{\mathbf{df}(k)}, G)$.*

Proof. We first establish the claim that for each $k \geqslant 1$, there exists a bounded expression $\mathbf{b}_\Gamma$ over Δ such that $Pk_\Delta(\Gamma^{\mathbf{df}(k)} \cap \mathbf{b}_\Gamma) = Pk_\Delta(\Gamma^{\mathbf{df}(k)})$. By Corollary 1, $\Gamma^{\mathbf{df}(k)}$ is a regular language, and by Theorem 4, there exists a bounded expression $\mathbf{b}_\Gamma$ over Δ such that $Pk_\Delta(\Gamma^{\mathbf{df}(k)} \cap \mathbf{b}_\Gamma) = Pk_\Delta(\Gamma^{\mathbf{df}(k)})$ which proves the claim.

Define $\mathcal{A} = \{a_1, \ldots, a_d\}$ and assume Δ is given as a linearly ordered set of m productions $\{p_1, \ldots, p_m\}$. Then for u such that $X \overset{\gamma}{\Rightarrow} u$, we have $Pk_\mathcal{A}(u) = Pk_\Delta(\gamma) \times \Pi$ where Π is the matrix of m rows and d columns where row i is given by $Pk_\mathcal{A}(tail(p_i))$. Next, let γ_1, γ_2 be two control words such that $Pk_\Delta(\gamma_1) = Pk_\Delta(\gamma_2)$ and each γ_i $(i = 1, 2)$ generates a word u_i of $L_X(G)$, that is $X \overset{\gamma_i}{\Longrightarrow} u_i$. We conclude from the above that $Pk_\mathcal{A}(u_1) = Pk_\mathcal{A}(u_2)$. Moreover, the

assumption $L_X(G) \subseteq (a_1 w_1)^* \dots (a_d w_d)^*$ where $a_1, \dots, a_d$ are distinct symbols shows that $u_1{\downarrow_{\mathcal{A}}} = u_2{\downarrow_{\mathcal{A}}}$. Furthermore, because no symbol of $\mathcal{A}$ occurs in $w_1 \cdots w_d$ we find that $u_1 = u_2$.

To show $L_X^{(k)}(G) = \hat{L}_X(\mathbf{b}_\Gamma \cap \Gamma_X^{\mathbf{df}(k)}, G)$ we prove that $L_X^{(k)}(G) \subseteq \hat{L}_X(\mathbf{b}_\Gamma \cap \Gamma_X^{\mathbf{df}(k)}, G)$ the other direction being immediate because of Proposition 1 which says that $L_X^{(k)}(G) = \hat{L}_X(\Gamma^{\mathbf{df}(k)}, G)$ and because only those control words γ such that $head((\gamma)_1) = X$ matters.

So, let $u \in \hat{L}_X(\Gamma_X^{\mathbf{df}(k)}, G)$ be a word, and $X \xRightarrow[\mathbf{df}(k)]{\gamma} u$ be a depth-first derivation of u. Since $Pk_\Delta(\Gamma_X^{\mathbf{df}(k)} \cap \mathbf{b}_\Gamma) = Pk_\Delta(\Gamma_X^{\mathbf{df}(k)})$, there exists a control word $\beta \in \Gamma^{\mathbf{df}(k)} \cap \mathbf{b}_\Gamma$ such that $Pk_\Delta(\beta) = Pk_\Delta(\gamma)$. Also because no production $p \in \Delta$ is such that $tail(p)$ contains an occurrence of X, we find that $(\beta)_1 = (\gamma)_1$. Finally, Lemma 3.1 shows that given $\beta \in \Gamma^{\mathbf{df}(k)}$, there exist a (unique) word u' such that $X \xRightarrow[\mathbf{df}(k)]{\beta} u'$, hence $u' = u$ as shown above. □ □

For the rest of this section, let $G = (\Xi, \Theta, \Delta)$ be a visibly pushdown grammar (we ignore for the time being the distinction between tagged and untagged alphabet symbols), and $X_0 \in \Xi$ be an arbitrarily chosen nonterminal.

Let $\mathbf{b} = w_1^* \cdots w_d^*$ be a bounded expression [6] over alphabet Θ and define the bounded expression $\widetilde{\mathbf{b}} = (a_1 w_1)^* \dots (a_d w_d)^*$ such that $\{a_1, \dots, a_d\}$ and Θ are disjoint. Next, let $\ell_i = |a_i\, w_i|$ for every $1 \leqslant i \leqslant d$ and let $G^{\widetilde{\mathbf{b}}} = (\Xi^{\widetilde{\mathbf{b}}}, \Theta \cup \{a_1, \dots, a_d\}, \delta^{\widetilde{\mathbf{b}}})$ be the regular grammar where

$$\begin{aligned}
\Xi^{\widetilde{\mathbf{b}}} &= \left\{ \mathrm{Q}_r^{(s)} \mid 1 \leqslant s \leqslant d \ \wedge\ 1 \leqslant r \leqslant \ell_s \right\} \\
\delta^{\widetilde{\mathbf{b}}} &= \left\{ \mathrm{Q}_i^{(s)} \to (a_s\, w_s)_i\ \mathrm{Q}_{i+1}^{(s)} \mid 1 \leqslant s \leqslant d \ \wedge\ 1 \leqslant i < \ell_s \right\} \cup \\
&\quad \left\{ \mathrm{Q}_{\ell_s}^{(s)} \to (a_s\, w_s)_{\ell_s}\ \mathrm{Q}_1^{(s')} \mid 1 \leqslant s \leqslant s' \leqslant d \right\} .
\end{aligned}$$

Checking $\{w \mid \mathrm{Q}_1^{(s)} \Rightarrow^* w\, \mathrm{Q}_1^{(x)} \text{ for some } 1 \leqslant s \leqslant x \leqslant d\} = L(\widetilde{\mathbf{b}})$ holds is routine. Next, given G and $G^{\widetilde{\mathbf{b}}}$, define $G^{\bowtie} = (\Xi^{\bowtie}, \Theta \cup \{a_1, \dots, a_d\}, \Delta^{\bowtie})$ such that $L_{X_0^{\bowtie}}(G^{\bowtie}) = L_{X_0}(G) \parallel L(\widetilde{\mathbf{b}})$.[7]

- $\Xi^{\bowtie} = \left\{X_0^{\bowtie}\right\} \cup \left\{ [\mathrm{Q}_r^{(s)} X \mathrm{Q}_y^{(x)}] \mid X \in \Xi,\ \mathrm{Q}_r^{(s)}, \mathrm{Q}_y^{(x)} \in \Xi^{\widetilde{\mathbf{b}}},\ s \leqslant x \right\}$
- $\Delta^{\bowtie}$ is the set containing for every $1 \leqslant s \leqslant x \leqslant d$ a production $X_0^{\bowtie} \to [\mathrm{Q}_1^{(s)} X_0 \mathrm{Q}_1^{(x)}]$, and:
  - for every production $X \to \gamma \in \Delta$, $\Delta^{\bowtie}$ has a production

$$[\mathrm{Q}_r^{(s)} X \mathrm{Q}_y^{(x)}] \to \gamma \qquad \text{if } \mathrm{Q}_r^{(s)} \to \gamma\ \mathrm{Q}_y^{(x)} \in \Delta^{\widetilde{\mathbf{b}}}\ ; \tag{1}$$

[6] Recall that each w_i is a non-empty word.

[7] Given two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ their asynchronous product, denoted $L_1 \parallel L_2$, is the language L over the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$ such that $w \in L$ iff the projections of w to Σ_1 and Σ_2 belong to L_1 and L_2, respectively. Observe that the $L_1 \parallel L_2$ depends on L_1, L_2 **and** also their underlying alphabet Σ_1 and Σ_2.

– for every production $X \to \gamma\, Y \in \Delta$, $\Delta^{\bowtie}$ has a production

$$[\mathrm{Q}_r^{(s)} X \mathrm{Q}_y^{(x)}] \to \gamma\, [\mathrm{Q}_t^{(z)} Y \mathrm{Q}_y^{(x)}] \qquad \text{if } \mathrm{Q}_r^{(s)} \to \gamma\, \mathrm{Q}_t^{(z)} \in \Delta^{\tilde{\mathbf{b}}}; \quad (2)$$

– for every production $X \to \tau\, Z\, \sigma\, Y \in \Delta$, $\Delta^{\bowtie}$ has a production

$$[\mathrm{Q}_r^{(s)} X \mathrm{Q}_y^{(x)}] \to \tau\, [\mathrm{Q}_t^{(z)} Z \mathrm{Q}_v^{(u)}]\, \sigma\, [\mathrm{Q}_k^{(\ell)} Y \mathrm{Q}_y^{(x)}] \qquad \text{if } \mathrm{Q}_r^{(s)} \to \tau\, \mathrm{Q}_t^{(z)} \in \Delta^{\tilde{\mathbf{b}}} \text{ and } \mathrm{Q}_v^{(u)} \to \sigma\, \mathrm{Q}_k^{(\ell)} \in \Delta^{\tilde{\mathbf{b}}}; \quad (3)$$

– for every production $\mathrm{Q}_1^{(s)} \to a_s\, \mathrm{Q}_v^{(u)} \in \delta^{\tilde{\mathbf{b}}}$, $\Delta^{\bowtie}$ has a production

$$[\mathrm{Q}_1^{(s)} X \mathrm{Q}_y^{(x)}] \to a_s\, [\mathrm{Q}_v^{(u)} X \mathrm{Q}_y^{(x)}]\ . \quad (4)$$

$\Delta^{\bowtie}$ has no other production.

Next we define the mapping ξ which maps each nonterminal $[\mathrm{Q}_r^{(s)} X \mathrm{Q}_y^{(x)}] \in \Xi^{\bowtie}$ onto X, $X_0^{\bowtie}$ onto X_0, every a_i, $1 \leqslant i \leqslant d$, onto ε and maps any other terminal (Θ) onto itself. Then ξ is naturally extended to words over $\Theta \cup \{a_1, \ldots, a_d\} \cup \Xi^{\bowtie}$. Next we lift ξ to productions of $\Delta^{\bowtie}$ such that the mapping of a production is defined by the mapping of its head and tail. The lifting of ξ to sequences of productions and sets of sequences of productions is defined in the obvious way.

From the above definition we observe that given a derivation $D^{\bowtie} \equiv X_0^{\bowtie} \Rightarrow [\mathrm{Q}_1^{(s)} X_0 \mathrm{Q}_1^{(x)}] \Longrightarrow^* w$ in $G^{\bowtie}$, ξ maps $D^{\bowtie}$ onto a derivation of G of the form $X_0 \Rightarrow X_0 \Longrightarrow^* w{\downarrow_\Theta}$.

Lemma 5.2. *Let $G = (\Xi, \Theta, \Delta)$ be a visibly pushdown grammar, $X_0 \in \Xi$ be a nonterminal such that $L_{X_0}(G) \subseteq \mathbf{b}$ for a bounded expression $\mathbf{b} = w_1^* \ldots w_d^*$. Let $\{a_1, \ldots, a_d\}$ be a set of d symbols disjoint from Θ. Then for every $k \geqslant 1$, the following hold:*

1. *Let $i_1, \ldots, i_d \in \mathbb{N}$ we have*

$$w_1^{i_1} \ldots w_d^{i_d} \in L_{X_0}^{(k)}(G) \text{ iff } (a_1 w_1)^{i_1} \ldots (a_d w_d)^{i_d} \in L_{X_0^{\bowtie}}^{(k)}(G^{\bowtie})\ ;$$

2. *Given a control set Γ over $\Delta^{\bowtie}$ such that*

$$\hat{L}_{X_0^{\bowtie}}(\Gamma \cap \Gamma^{\mathbf{df}(k)}(G^{\bowtie}), G^{\bowtie}) = L_{X_0^{\bowtie}}^{(k)}(G^{\bowtie})$$

then the control set $\Gamma' = \xi(\Gamma)$ over Δ satisfies

$$\hat{L}_{X_0}(\Gamma' \cap \Gamma^{\mathbf{df}(k)}(G), G) = L_{X_0}^{(k)}(G)\ .$$

Proof. The proof of point 1 is by induction. As customary, we show the following stronger statement: let $k \geqslant 1$ and $w \in (\Theta \cup \{a_1, \ldots, a_d\})^* \cdot \Theta$, we have $[\mathrm{Q}_r^{(s)} X \mathrm{Q}_v^{(u)}] \underset{(k)}{\Longrightarrow}^* w$ iff $\mathrm{Q}_r^{(s)} \Rightarrow^* w\, \mathrm{Q}_v^{(u)}$ and $X \underset{(k)}{\Longrightarrow}^* w{\downarrow_\Theta}$. The proof of the if direction is by induction on the length of $\mathrm{Q}_r^{(s)} \Rightarrow^* w\, \mathrm{Q}_v^{(u)}$.

i = 1. Then $\mathrm{Q}_r^{(s)} \to \tau\, \mathrm{Q}_v^{(u)} \in \Delta^{\tilde{\mathbf{b}}}$. Two cases can occur: (*i*) $\tau \in \Theta$; or (*ii*) $\tau \in \{a_1, \ldots, a_d\}$.

In case (i), we conclude from $X \underset{(k)}{\Longrightarrow}^* w{\downarrow_\Theta}$ that $w{=}w{\downarrow_\Theta}{=}\tau$ and $X \to \tau \in \Delta$, hence that $[\mathrm{Q}_r^{(s)} X \mathrm{Q}_v^{(u)}] \to \tau \in \Delta^{\bowtie}$, and finally that $[\mathrm{Q}_r^{(s)} X \mathrm{Q}_v^{(u)}] \underset{(k)}{\Longrightarrow}^* w$. Case (ii) is not allowed since w must end with a symbol in Θ.

i > 1. Then $\mathrm{Q}_r^{(s)} \Rightarrow \tau\, \mathrm{Q}_{r'}^{(s')} \Rightarrow \circ \Rightarrow^* \overbrace{\tau\, y}^{w}\, \mathrm{Q}_v^{(u)}$. As seen previously, two cases can occur: (i) $\tau \in \{a_1, \ldots, a_d\}$; or (ii) $\tau \in \Theta$. In case (i), because $w = \tau\, y$ and $\tau \notin \Theta$ we find that $X \underset{(k)}{\Longrightarrow}^* w{\downarrow_\Theta} = y{\downarrow_\Theta}$. Hence the induction hypothesis shows that $[\mathrm{Q}_{r'}^{(s')} X \mathrm{Q}_v^{(u)}] \underset{(k)}{\Longrightarrow}^* y$. Finally the definition of $G^{\bowtie}$ shows that $[\mathrm{Q}_r^{(s)} X \mathrm{Q}_v^{(u)}] \to \tau\, [\mathrm{Q}_{r'}^{(s')} X \mathrm{Q}_v^{(u)}] \in \Delta^{\bowtie}$, hence that $[\mathrm{Q}_r^{(s)} X \mathrm{Q}_v^{(u)}]$
$\underset{(k)}{\Longrightarrow}^* \tau\, [\mathrm{Q}_{r'}^{(s')} X \mathrm{Q}_v^{(u)}] \underset{(k)}{\Longrightarrow}^* \tau\, y = w$ and we are done.

For case (ii) ($\tau \in \Theta$), we do a (sub)case analysis according to the first production rule used in the derivation $X \underset{(k)}{\Longrightarrow}^* w{\downarrow_\Theta}$.

- $X \to \tau$. Then $X \underset{(k)}{\Longrightarrow}^* w{\downarrow_\Theta} = \tau$. On the other hand $\mathrm{Q}_r^{(s)} \Rightarrow \tau\, \mathrm{Q}_{r'}^{(s')} \Rightarrow \circ \Rightarrow^* \tau\, y\, \mathrm{Q}_v^{(u)}$ and our assumption on $w = \tau\, y$ shows that y ends with a symbol in Θ. Hence a contradiction since $w{\downarrow_\Theta} = \tau$ does not coincide with the projection of $w = \tau\, y$.

- $X \to \tau\, Y$. Then $X \underset{(k)}{\Longrightarrow} \tau\, Y \underset{(k)}{\Longrightarrow}^* \tau\, y{\downarrow_\Theta} = w{\downarrow_\Theta}$. Also $\mathrm{Q}_r^{(s)} \Rightarrow \tau\, \mathrm{Q}_{r'}^{(s')} \Rightarrow \circ \Rightarrow^* \tau\, y\, \mathrm{Q}_v^{(u)}$. The induction hypothesis applied on $Y \underset{(k)}{\Longrightarrow}^* y{\downarrow_\Theta}$ and $\mathrm{Q}_{r'}^{(s')} \Rightarrow^* y\, \mathrm{Q}_v^{(u)}$ shows that $[\mathrm{Q}_{r'}^{(s')} Y \mathrm{Q}_v^{(u)}] \underset{(k)}{\Longrightarrow}^* y$. Finally, $X \to \tau\, Y \in \Delta$ and $\mathrm{Q}_r^{(s)} \to \tau\, \mathrm{Q}_{r'}^{(s')} \in \Delta^{\tilde{\mathbf{b}}}$ show that $[\mathrm{Q}_r^{(s)} X \mathrm{Q}_v^{(u)}] \to \tau\, [\mathrm{Q}_{r'}^{(s')} Y \mathrm{Q}_v^{(u)}] \in \Delta^{\bowtie}$, hence that $[\mathrm{Q}_r^{(s)} X \mathrm{Q}_v^{(u)}] \underset{(k)}{\Longrightarrow}^* \tau\, [\mathrm{Q}_{r'}^{(s')} Y \mathrm{Q}_v^{(u)}] \underset{(k)}{\Longrightarrow}^* \tau\, y = w$ and we are done.

- $X \to \tau\, X_1\, \sigma\, X_2$. Then $X \underset{(k)}{\Longrightarrow} \tau\, X_1 \sigma\, X_2 \underset{(k)}{\Longrightarrow}^* \tau\, w_1{\downarrow_\Theta}\, \sigma\, w_2{\downarrow_\Theta} = w{\downarrow_\Theta}$. Moreover, since $\mathrm{Q}_s^{(r)} \Rightarrow^* w\, \mathrm{Q}_v^{(u)}$ and $\tau, \sigma \in \Theta$ we find that there exist $\mathrm{Q}_s^{(r)} \Rightarrow \tau\, \mathrm{Q}_a^{(b)} \Rightarrow^* \tau\, w_1\, \mathrm{Q}_{a'}^{(b')} \Rightarrow \tau\, w_1\, \sigma\, \mathrm{Q}_c^{(d)} \Rightarrow^* \tau\, w_1\, \sigma\, w_2 \mathrm{Q}_v^{(u)}$. Hence, the definition of $G^{\bowtie}$ shows that

$$[\mathrm{Q}_s^{(r)} X \mathrm{Q}_v^{(u)}] \to \tau\, [\mathrm{Q}_a^{(b)} X_1 \mathrm{Q}_{a'}^{(b')}]\, \sigma\, [\mathrm{Q}_c^{(d)} X_2 \mathrm{Q}_v^{(u)}]\ .$$

  On the other hand, since $X_1 X_2 \underset{(k)}{\Longrightarrow}^* w_1 {\downarrow_\Theta}\ w_2{\downarrow_\Theta}$ (simply delete τ and σ), Lemma 4.1 shows that either $X_1 \underset{(k-1)}{\Longrightarrow}^* w_1{\downarrow_\Theta}$ and $X_2 \underset{(k)}{\Longrightarrow}^* w_2{\downarrow_\Theta}$; or $X_1 \underset{(k)}{\Longrightarrow}^* w_1{\downarrow_\Theta}$ and $X_2 \underset{(k-1)}{\Longrightarrow}^* w_2{\downarrow_\Theta}$. Let us assume the latter holds (the other being treated similarly). Applying the induction hypothesis, we find that $[\mathrm{Q}_a^{(b)} X_1 \mathrm{Q}_{a'}^{(b')}] \underset{(k)}{\Longrightarrow}^* w_1$ and $[\mathrm{Q}_c^{(d)} X_2 \mathrm{Q}_v^{(u)}] \underset{(k-1)}{\Longrightarrow}^* w_2$, hence we conclude the case with the k-index derivation $[\mathrm{Q}_s^{(r)} X \mathrm{Q}_v^{(u)}] \underset{(k)}{\Longrightarrow}^* \tau\, [\mathrm{Q}_a^{(b)} X_1 \mathrm{Q}_{a'}^{(b')}]\, \sigma\, [\mathrm{Q}_c^{(d)} X_2 \mathrm{Q}_v^{(u)}] \underset{(k)}{\Longrightarrow}^* \tau\, [\mathrm{Q}_a^{(b)} X_1 \mathrm{Q}_{a'}^{(b')}]\, \sigma\, w_2 \underset{(k)}{\Longrightarrow}^* \tau\, w_1\, \sigma\, w_2$.

The "only if" direction is proved similarly, this time by induction on the length of the derivation $[\mathrm{Q}_r^{(s)} X \mathrm{Q}_v^{(u)}] \underset{(k)}{\Longrightarrow}^* w$.

For the proof of point 2 the "$\subseteq$" direction is obvious by definition of depth-first derivations. For the reverse direction "$\supseteq$" point 1 combined with the assumption shows that for every $i_1, \ldots, i_d \in \mathbb{N}$ the following equivalence holds:

$$\begin{gathered} w_1^{i_1} \ldots w_d^{i_d} \in L_{X_0}^{(k)}(G) \\ \text{iff} \\ (a_1 w_1)^{i_1} \ldots (a_d w_d)^{i_d} \in \hat{L}_{X_0^{\bowtie}}(\Gamma \cap \Gamma^{\mathbf{df}(k)}, G^{\bowtie}) \ . \end{gathered}$$

So let $D \equiv X_0^{\bowtie} \underset{(k)}{\Longrightarrow}^* w$ be a depth-first k-index derivation of $G^{\bowtie}$ with control word conforming to Γ. Now consider $\xi(D)$, it defines again a depth-first k-index derivation except that this time the control word conforms to $\xi(\Gamma)$. Further, the definition of ξ shows that the word generated by $\xi(D)$ results from deleting the symbols $\{a_1, \ldots, a_d\}$ from $w = (a_1 w_1)^{i_1} \cdots (a_d w_d)^{i_d}$. To conclude, observe that $w_1^{i_1} \cdots w_d^{i_d} \in L_{X_0}^{(k)}(G)$ and we are done. □ □

The following proposition shows that $L_Q^{(k)}(G_{\mathcal{P}})$ is captured by a subset of depth-first derivations whose control words belong to some bounded expression.

Proposition 2. *Let $G = (\Xi, \widehat{\Theta}, \Delta)$ be a visibly pushdown grammar, $X_0 \in \Xi$ be a nonterminal such that $L_{X_0}(G)$ is bounded. Then for each $k \geqslant 1$ there exists a bounded expression $\mathbf{b}_\Gamma$ over Δ such that $\hat{L}_{X_0}(\mathbf{b}_\Gamma \cap \Gamma^{\mathbf{df}(k)}, G) = L_{X_0}^{(k)}(G)$.*

Proof. Since $L_{X_0}(G)$ is bounded there exists a bounded expression $\mathbf{b} = w_1^* \ldots w_d^*$ such that $L_{X_0}(G) \subseteq \mathbf{b}$.

Next, define $\{a_1, \ldots, a_d\}$ be an alphabet disjoint from Θ. Lemma 5.2 shows that for every $i_1, \ldots, i_d \in \mathbb{N}$ the equivalence $w_1^{i_1} \ldots w_d^{i_d} \in L_{X_0}^{(k)}(G)$ iff $(a_1 w_1)^{i_1} \ldots (a_d w_d)^{i_d} \in L_{X_0^{\bowtie}}^{(k)}(G^{\bowtie})$ holds. Next, applying Lemma 5.1 on $L_{X_0^{\bowtie}}^{(k)}(G^{\bowtie})$ (whose assumptions holds by definition of $G^{\bowtie}$) we obtain a bounded expression $\mathbf{b}_{\Gamma^{\bowtie}}$ over $\Delta^{\bowtie}$ such that $\hat{L}_{X_0^{\bowtie}}(\mathbf{b}_{\Gamma^{\bowtie}} \cap \Gamma^{\mathbf{df}(k)}, G^{\bowtie}) = L_{X_0^{\bowtie}}^{(k)}(G^{\bowtie})$. Our next step is to apply the results of Lemma 5.2 (second point) to obtain that $L_{X_0}^{(k)}(G) = \hat{L}_{X_0}(\xi(\mathbf{b}_{\Gamma^{\bowtie}}) \cap \Gamma^{\mathbf{df}(k)}), G)$. Finally, since $\mathbf{b}_{\Gamma^{\bowtie}}$ is a bounded expression, and ξ is an homomorphism we have that $\xi(\mathbf{b}_{\Gamma^{\bowtie}})$ is bounded (see Lem. 5.3), hence included in a bounded expression and we are done by setting $\mathbf{b}_\Gamma$ to $\xi(\mathbf{b}_{\Gamma^{\bowtie}})$. □ □

5.2 Proof of Theorem 3

We recall two results from Ginsburg [14].

Theorem 5 (Theorem 3.3.2, [14])**.** *Each gsm mapping preserves regular sets.*

Lemma 5.3 (Lemma 5.5.3, [14])**.** *$S(w_1^* \ldots w_n^*)$ is bounded for each gsm S and all words $w_1, \ldots, w_n$.*

And finally, the proof that *query*^{k} is flattable.

	$k=2$			$k=3$			$k=4$		
	#	t	fp	#	t	fp	#	t	fp
`identity`	210	0.10	no	330	0.22	yes		-	
`leq`	152	0.12	no	240	0.27	no	328	0.41	yes
`parity`	384	0.14	no	606	0.54	no	828	1.31	yes
`plus`	462	0.53	no	728	2.54	no	994	9.20	yes
`times2`	210	0.14	no	330	0.35	yes		-	

Table 1: Experiments with recursive implementations of basic arithmetic functions and predicates [1].

of Theorem. 3. Since $\mathcal{P}$ is bounded periodic we can apply Proposition 2 showing the existence of a bounded expression $\mathbf{b}_\Gamma$ over Δ such that $\hat{L}_Q(\mathbf{b}_\Gamma \cap \Gamma^{\mathbf{df}(k)}, G_\mathcal{P}) = L_Q^{(k)}(G_\mathcal{P})$. Hence we find that $[\![\mathcal{P}]\!]_q^{(k)}$ coincides with $\bigcup_{\alpha \in L_Q^{(k)}(G_\mathcal{P})} [\![\alpha]\!]$ which in turn is equal to $\bigcup_{\alpha \in \hat{L}_Q(\mathbf{b}_\Gamma \cap \Gamma^{\mathbf{df}(k)}, G_\mathcal{P})} [\![\alpha]\!]$.

Lemma 3.2 shows that for all control word $\gamma \in \Delta^*$ such that $Q \overset{\gamma}{\underset{\mathbf{df}}{\Longrightarrow}} \alpha$ we have that $[\![\gamma]\!] = \{I \cdot O \mid \langle I{\downarrow_{\mathbf{x}_i}}, O{\downarrow_{\mathbf{x}_i}} \rangle \in [\![\alpha]\!]\}$. This enables the use of Lemma 4.3 showing that such control word γ is such that $[\![\gamma]\!] = [\![SC_Q^k(\gamma)]\!]$. This is saying the semantics of γ in $\mathcal{P}$ can be obtained by computing that of $SC_Q^k(\gamma)$ in $query^k$.

We then conclude from Lem. 5.3 and Thm. 5, that $\mathrm{SC}_Q^k(\mathbf{b}_\Gamma)$ is a bounded and regular language. Back to $[\![\mathcal{H}]\!]_{query^k}$, we find that

$$[\![\mathcal{H}]\!]_{query^k} = \bigcup_{\alpha \in L_{query^k}(G_\mathcal{H})} [\![\alpha]\!] = \bigcup_{\alpha \in L_{query^k}(G_\mathcal{H}) \cap \mathrm{SC}_Q^k(\mathbf{b}_\Gamma)} [\![\alpha]\!]$$

and that $[\![\mathcal{H}]\!]_{query^k}$ is flattable since $\mathrm{SC}_Q^k(\mathbf{b}_\Gamma)$ is a bounded regular set. □ □

6 Experiments

We have implemented the proposed method in the Flata verifier [17] and experimented with several benchmarks. The Flata tool is publicly available[8] and the benchmarks used in this section are given in the repository. First, we have considered several programs from external sources [1], that compute arithmetic functions or predicates in a recursive way such as `identity` (identity), `plus` (addition), `times2` (multiplication by two), `leq` (comparison), and `parity` (parity checking). It is worth noting that all of these programs have bounded index visibly pushdown grammars, i.e. $L(G^P)$ is of bounded index, for each program $P \in \{\texttt{identity}, \texttt{plus}, \texttt{times2}, \texttt{leq}, \texttt{parity}\}$, the stabilization of the under-approximation sequence is thus guaranteed. For all our benchmarks, the condition that the tuple of relation $[\![\mathcal{P}]\!]^{(k)}$ is inductive with respect to the statements of $\mathcal{P}$ is met for $k \leqslant 3$. Table 1 shows the results, giving the size (#) of each under-approximation $query^k$ (the number of transitions) and the time (t) needed to compute its summary (in seconds). The column fp indicates whether the fixpoint check was successful. The platform used for all experiments is MacBookPro with Intel Core i7 2,3 GHz with 16 GB of RAM.

Next, we have considered two generalizations of the McCarthy 91 function [10], a well-known verification benchmark that has long been a challenge. We have automatically computed precise summaries of its generalizations F_a (Table

[8] `https://github.com/filipkonecny/flata`

	$k=2$			$k=3$			$k=4$		
	#	t	fp	#	t	fp	#	t	fp
F_1	32	0.05	no	50	0.07	no	68	0.09	yes
F_2	72	0.06	no	114	0.74	no	156	1.55	yes
F_3	128	0.06	no	204	0.30	no	280	1.59	yes
F_4	200	0.06	no	320	0.44	no	440	4.02	yes
F_5	288	0.07	no	462	0.63	no	636	5.97	yes
F_6	392	0.07	no	630	0.82	no	868	7.54	yes
F_7	512	0.08	no	824	0.86	no	1136	14.23	yes
F_8	648	0.08	no	1044	1.09	no	1440	12.87	yes

$$F_a(x) = \begin{cases} x - 10 & \text{if } x \geqslant 101 \\ (F_a)^a(x + 10 \cdot a - 9) & \text{if } x \leqslant 100 \end{cases}$$

Table 2: Generalized McCarthy F_a Experiments. The function F_2 is the original McCarthy91 function.

	$k=2$			$k=3$			$k=4$		
	#	t	fp	#	t	fp	#	t	fp
G_{11}	72	0.06	no	114	0.74	no	156	1.55	yes
G_{12}	72	0.08	no	114	1.53	no	156	n/a	?
G_{13}	72	0.08	no	114	5.07	no	156	n/a	?
G_{14}	72	0.08	no	114	7.07	no	156	n/a	?

$$G_b(x) = \begin{cases} x - 10 & \text{if } x \geqslant 101 \\ G(G(x + b)) & \text{if } x \leqslant 100 \end{cases}$$

Table 3: Generalized McCarthy G_b Experiments. The function G_{11} is the original McCarthy91 function.

$G_{11}(x)$	91	if $x \leqslant 100$
	$x-10$	if $x \geqslant 101$
$G_{12}(x)$	91	if $x \leqslant 100$ and $2 \mid x+1$
	92	if $x \leqslant 100$ and $2 \mid x$
	$x-10$	if $x \geqslant 101$
$G_{13}(x)$	91	if $x \leqslant 100$ and $3 \mid x+1$
	92	if $x \leqslant 100$ and $3 \mid x$
	93	if $x \leqslant 100$ and $3 \mid x+2$
	$x-10$	if $x \geqslant 101$
$G_{14}(x)$	91	if $x \leqslant 100$ and $4 \mid x+3$
	92	if $x \leqslant 100$ and $4 \mid x+2$
	93	if $x \leqslant 100$ and $4 \mid x+1$
	94	$x \leqslant 100$ and $4 \mid x$
	$x-10$	if $x \geqslant 101$

Table 4: Automatically computed summaries for the generalized McCarthy G_b functions (for index $k = 3$).

2) and G_b (Table 3) above for $a = 2, \ldots, 8$ and $b = 12, 13, 14$. For the F_a functions, the computed summaries are given by:

$$F_a(x) = \begin{cases} 91 & \text{if } x \leqslant 100 \\ x-10 & \text{if } x \geqslant 101 \end{cases} \quad \text{for all } a = 1, \ldots, 8 \ .$$

The computed summaries for the G_b functions are given in Table 4.

The visibly pushdown grammars corresponding to the recursive programs implementing the F_a, G_b functions are not bounded. In the case of the F_a function, the under-approximation sequence reaches a fixpoint after 4 iterations. In the case of G_b, for $b = 12, 13, 14$, the summary of $query^3$ is the expected result. However, due to the limitations of the FLATA tool, which is based on an acceleration procedure without abstraction, we could not compute the summary of $query^4$, and we could not verify automatically that the fixpoint has been reached.

7 Conclusions

We have presented an underapproximation method for computing summaries of recursive programs operating on integers. The underapproximation is driven by bounding the index of derivations that produce the execution traces of the program, and computing the summary, for each index, by analyzing a non-recursive program. We also present a class of programs on which our method is complete. Finally, we report on an implementation and experimental evaluation of our technique.

Acknowledgements. Pierre Ganty is supported by the EU FP7 2007–2013 program under agreement 610686 POLCA, by the Madrid Regional Government under CM project S2013/ICE-2731 (N-Greens) and RISCO: RIgorous analysis of Sophisticated COncurrent and distributed systems, funded by the Spanish Ministry of Economy and Competitiveness No. TIN2015-71819-P (2016–2018). Pierre thanks Thomas Reps for pointing out inconsistencies in the examples.

References

[1] Termination Competition 2011. `http://termcomp.uibk.ac.at/termcomp/home.seam`.

[2] A. Albarghouthi, A. Gurfinkel, and M. Chechik. Whale: An interpolation-based algorithm for inter-procedural verification. In *VMCAI '12*, volume 7148 of *LNCS*, pages 39–55. Springer, 2012.

[3] R. Alur and P. Madhusudan. Adding nesting structure to words. *JACM*, 56(3):16, 2009.

[4] M. F. Atig and P. Ganty. Approximating petri net reachability along context-free traces. In *FSTTCS '11*, volume 13 of *LIPIcs*, pages 152–163. Schloss Dagstuhl, 2011.

[5] A. P. B. Cook and A. Rybalchenko. Summarization for termination: no return! *Formal Methods in System Design*, 35:369–387, 2009.

[6] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. FAST: Fast acceleration of symbolic transition systems. In *CAV '03*, volume 2725 of *LNCS*, pages 118–121. Springer, 2003.

[7] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD thesis, University of Liège, 1998.

[8] M. Bozga, R. Iosif, and F. Konečný. Fast acceleration of ultimately periodic relations. In *CAV '10*, volume 6174 of *LNCS*, pages 227–242. Springer, 2010.

[9] M. Bozga, R. Iosif, and Y. Lakhnech. Flat parametric counter automata. *Fundamenta Informaticae*, 91(2):275–303, 2009.

[10] J. Cowles. Knuth's generalization of mccarthy's 91 function. In *Computer-Aided reasoning: ACL2 case studies*, pages 283–299. Kluwer Academic Publishers, 2000.

[11] J. Esparza, S. Kiefer, and M. Luttenberger. Newtonian program analysis. *JACM*, 57(6):33:1–33:47, 2010.

[12] A. Finkel and J. Leroux. How to compose presburger-accelerations: Applications to broadcast protocols. In *FSTTCS '02*, volume 2556 of *LNCS*, pages 145–156. Springer, 2002.

[13] P. Ganty, R. Majumdar, and B. Monmege. Bounded underapproximations. *Formal Methods in System Design*, 40(2):206–231, 2012.

[14] S. Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, Inc., New York, NY, USA, 1966.

[15] P. Godefroid, A. V. Nori, S. K. Rajamani, and S. Tetali. Compositional may-must program analysis: unleashing the power of alternation. In *POPL '10*, pages 43–56. ACM, 2010.

[16] G. Godoy and A. Tiwari. Invariant checking for programs with procedure calls. In *SAS '09*, volume 5673 of *LNCS*, pages 326–342. Springer, 2009.

[17] H. Hojjat, F. Konečný, F. Garnier, R. Iosif, V. Kuncak, and P. Rümmer. A verification toolkit for numerical transition systems - tool paper. In *FM*, pages 247–251, 2012.

[18] D. Kroening, M. Lewis, and G. Weissenbacher. Under-approximating loops in C programs for fast counterexample detection. In *CAV '13: Proc. 23rd Int. Conf. on Computer Aided Verification*, LNCS, pages 381–396. Springer, 2013.

[19] G. Lalire, M. Argoud, and B. Jeannet. Interproc. `http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc/index.html`.

[20] M. Latteux. Mots infinis et langages commutatifs. *Informatique Théorique et Applications*, 12(3), 1978.

[21] M. Luker. A family of languages having only finite-index grammars. *Information and Control*, 39(1):14–18, 1978.

[22] M. Luker. Control sets on grammars using depth-first derivations. *Mathematical Systems Theory*, 13:349–359, 1980.

[23] T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *POPL '95*, pages 49–61. ACM, 1995.

[24] M. Sharir and A. Pnueli. Two approaches to interprocedural data flow analysis. In *Program Flow Analysis: Theory and Applications*, chapter 7, pages 189–233. Prentice-Hall, Inc., 1981.