Abstract Interpretation of Indexed Grammars

Marco Campion, Mila Dalla Preda, and Roberto Giacobazzi

Dipartimento di Informatica, University of Verona
{marco.campion, mila.dallapreda, roberto.giacobazzi} @ univr.it

Abstract. Indexed grammars are a generalization of context-free grammars and recognize a proper subset of context-sensitive languages. The class of languages recognized by indexed grammars are called indexed languages and they correspond to the languages recognized by nested stack automata. For example indexed grammars can recognize the language $\{a^n b^n c^n \mid n \geqslant 1\}$ which is not context-free, but they cannot recognize $\{(ab^n)^n \mid n \geqslant 1\}$ which is context-sensitive. Indexed grammars identify a set of languages that are more expressive than context-free languages, while having decidability results that lie in between the ones of context-free and context-sensitive languages. In this work we study indexed grammars in order to formalize the relation between indexed languages and the other classes of languages in the Chomsky hierarchy. To this end, we provide a fixpoint characterization of the languages recognized by an indexed grammar and we study possible ways to abstract, in the abstract interpretation sense, these languages and their grammars into context-free and regular languages.

1 Introduction

Chomsky's hierarchy [6] drove most of the research in theoretical computer science for decades. Its structure, and its inner separation results between formal languages, represent the corner stone to understand the expressive power of symbolic structures. In this paper we show how abstract interpretation can be used for studying formal languages, in particular we consider indexed languages as our concrete semantics. This because of two reasons: (1) they lack, to the best of our knowledge, of a fixpoint semantics and (2) they represent an intermediate family of languages between context-free (CF) and context-sensitive (CS), therefore including CF and regular (REG) languages as subclasses.

Indexed languages have been introduced in [2] as an extension of CF languages in order to include languages such as $\{a^n b^n c^n \mid n \geqslant 1\}$. It is known that indexed languages are strictly less expressive than CS languages, e.g., the language $\{(ab^n)^n \mid n \geqslant 1\}$ is CS but not indexed. This intermediate class between CF and CS has interesting properties, e.g., decidable emptiness test and NP-complete membership check, where the first is undecidable and the latter is PSPACE-complete in CS.

Indexed languages are described by indexed grammars which differ from CF grammars in that each non-terminal is equipped with a stack on which push

and pop instructions can be performed. Moreover, the stack can be copied to all non-terminals on the right side of each production.

Although sporadically used in the literature (we can mention its use in natural language analysis [20] and in logic programming [5]) indexed languages represent an ideal concrete semantics for rebuilding part of Chomsky's hierarchy by abstract interpretation, in particular for the case of regular and CF languages.

Abstract interpretation [10, 11] is a general theory for the approximation of dynamic systems. It generalizes most existing methodologies for static program analysis into a unique sound-by-construction framework which is based on a simple but striking idea: *extracting properties of a system is approximating its semantics* [10]. In this paper we show that abstract interpretation can be used for studying the relation between formal languages in Chomsky's hierarchy.

The first step in our construction is to give a fixpoint semantics to indexed languages (Section 4). The construction follows the one known for CS languages, and derives a system of equations associated with each indexed grammar. We prove that the fixpoint solution of this system of equations corresponds precisely to the language generated by the grammar. This will provide the base fixpoint semantics for making abstract interpretation.

We show in Section 5 that no best abstraction, which in abstract interpretation are represented by Galois Insertions, is possible between indexed languages and respectively CF and regular languages, w.r.t. set inclusion. This means that we need to act at the level of grammar structures (i.e., on the way languages are generated and represented in grammatical form) in order to generate languages as abstract interpretations of an index grammar. It is therefore necessary to weaken the structure of Galois insertion-based abstract interpretation and consider abstractions that do not admit adjoint [12]. This is a quite widespread phenomenon in program analysis, e.g., the polyhedra abstract domain does not form a Galois insertion with the concrete semantics [14]. We introduce several abstractions of grammatical structures in such a way that the abstract language transformer associated with the system of equations of the indexed language generates the desired language. We show that certain simplifications of the productions of indexed grammars can be specified as abstractions, now in the standard Galois insertion based framework, and that the corresponding abstract semantics coincides precisely to classes of languages in Chomsky's hierarchy, in our case the class of CF languages. The main advantage is that known fixpoint characterisation and algorithms for CF languages can be extracted in a *calculational* way by abstract interpretation of the fixpoint semantics of the more concrete indexed grammars. This shows that standard methods for the design of static program analyses and hierarchy of semantics (e.g., see [15, 16, 9]) can be applied to systematically derive fixpoint presentations for families of formal languages and to let abstract interpretation methods to be applicable to Chomsky's hierarchy.

Section 6 concludes the paper with a discussion of related future works.

2 Background

Mathematical Notation We denote with $\mathbf{X} = (X_1, \ldots, X_n)$ a tuple $\mathbf{X}$ of n elements. We define with $\mathtt{proj}_i$ the projection function of the i-th element of a tuple such that $\mathtt{proj}_i(\mathbf{X}) = X_i$ and $\mathtt{proj}_{-1}(\mathbf{X}) = X_n$.

Given two sets S and T, we denote with $\wp(S)$ the powerset of S, with $S \subset T$ strict inclusion and with $S \subseteq T$ inclusion. $\langle P, \leqslant_P \rangle$ denotes a poset P with ordering relation $\leqslant_P$. A function $f : P \to Q$ on poset is additive when for any $Y \subseteq P$: $f(\bigvee_P Y) = \bigvee_Q f(Y)$, and co-additive when for any $Y \subseteq P : f(\bigwedge_P Y) = \bigwedge_Q f(Y)$. A poset $\langle P, \leqslant_P \rangle$ with $P \neq \emptyset$, is a lattice if $\forall x, y \in P$ we have that $x \vee y, x \wedge y \in P$. A lattice is complete if for every $S \subseteq P$ we have that $\bigvee S \in P$ and $\bigwedge S \in P$. A lattice is denoted $\langle C, \leqslant_C, \vee_C, \wedge_C, \top_C, \bot_C \rangle$.

Abstract Domains The Abstract Interpretation (AI) framework is based on the correspondence between a domain of concrete or exact properties and a domain of abstract or approximate properties [11]. The concrete is specified by a set C called the concrete semantic domain and a partial function $F_C : C \to C$ which is the concrete semantic transfer function with fixpoint solution starting from a basic element $\bot_C \in C$ such that $F_C^0(\bot_C) = \bot_C$ and $F_C^{i+1}(\bot_C) = F_C(F_C^i(\bot_C))$. In particular, the concrete iterates may be in increasing order for a partial order $\leqslant_C \in \wp(C \times C)$. This partial order relation may induce a partial order $\langle C, \leqslant_C \rangle$ or even a complete lattice structure on C [11].

The abstract is specified by an abstract semantic domain A which is an approximate version of the concrete semantic domain C. The objective of an abstract interpretation is to find an abstract property $a \in A$, if any, which is a correct approximation of the concrete semantics $c \in C$. Abstract semantics can be specified by transfinite recursion using an abstract basis $\bot_A \in A$ and an abstract semantic function $F_A : A \to A$ such that $F_A^0(\bot_A) = \bot_A$ and $F_A^{i+1}(\bot_A) = F_A(F_A^i(\bot_A))$. The abstract iterates may be in increasing order for a partial order $\leqslant_A \in \wp(A \times A)$ which may induce an order structure $\langle A, \leqslant_A, \bot_A, \cup_A \rangle$ ensuring that the abstract iteration is convergent [11].

The correspondence between the concrete and abstract properties is specified by a soundness relation $\sigma \in \wp(C \times A)$ where $\langle c, a \rangle \in \sigma$ means that the concrete semantics c has the abstract property a. A common assumption is that every concrete property has an abstract approximation: $\forall c \in C : \exists a \in A : \langle c, a \rangle \in \sigma$ [11].

The Galois Insertion approach to AI is based on a Galois Insertion (or equivalently closure operators, Moore families, complete join congruence relations or families of principal ideals [11]) correspondence between concrete and abstract properties. Galois Insertions (GI) are defined between a concrete domain $\langle C, \leqslant_C \rangle$ and an abstract domain $\langle A, \leqslant_A \rangle$ which are assumed to be at least posets [10]. A GI is a tuple (C, α, γ, A) where the abstraction map $\alpha : C \to A$ and the concretization map $\gamma : A \to C$ give rise to an adjunction: $\forall a \in A, c \in C : \alpha(c) \leqslant_A a \Leftrightarrow c \leqslant_C \gamma(a)$. Thus, $\alpha(c) \leqslant_A a$ and, equivalently $c \leqslant_C \gamma(a)$, means that a is a sound approximation of c in A. A tuple

(C, α, γ, A) is a GI iff α is additive iff γ is co-additive [10]. A GI is a Galois Connection (GC) where $\alpha \circ \gamma = \mathrm{id}$. Indeed, GIs ensure that $\alpha(c)$ actually provides the best possible approximation of the concrete value $c \in C$ on A. Whenever we have an additive (resp. co-additive) function f between two domains we can always build a GI by considering the right (resp. left) adjoint map induced by f. In fact, every abstraction map induces a concretization map and viceversa, formally $\gamma(a) = \bigvee_C \{c \mid \alpha(c) \leqslant_A a\}$ and $\alpha(a) = \bigwedge_A \{a \mid \gamma(a) \leqslant_C c\}$ [10].

An Upper Closure Operator (uco) $\varphi \in C \to C$ on a poset $\langle C, \leqslant \rangle$ is an operator that is monotone, idempotent and extensive (i.e. $\forall c \in C\ c \leqslant \varphi(c)$) [11]. Closures are uniquely determined by the set of their fixpoints $\varphi(C)$. The set of all closures on C is denoted by $\mathrm{uco}(C)$. The lattice of abstract domains of C is therefore isomorphic to $\mathrm{uco}(C)$ [11]. If C is a complete lattice, then $\langle \mathrm{uco}(C), \sqsubseteq, \sqcup, \sqcap, \lambda x.\top, \mathrm{id} \rangle$ is a complete lattice, where $\mathrm{id} = \lambda x.x$ and for every $\rho, \eta \in \mathrm{uco}(C)$, $\rho \sqsubseteq \eta \leftrightarrow \forall y \in C\ \rho(y) \leqslant \eta(y) \leftrightarrow \eta(C) \subseteq \rho(C)$. The glb $\sqcap$ is isomorphic to the so called reduced product, i.e. $\bigsqcap_{i \in I} \rho_i$ is the most abstract common concretization of all ρ_i.

Given $X \subseteq C$, the least abstract domain containing X is the least closure including X as fixpoints, which is the Moore-closure $M(X) = \{\bigwedge S \mid S \subseteq X\}$. Note that $\bigsqcap_{i \in I} \rho_i = M(\bigcup_{i \in I} \rho_i)$. If (C, α, γ, A) is a GI then $\varphi = \gamma \circ \alpha$ is the closure associated with A, such that $\varphi(C)$ is a complete lattice isomorphic to A.

Abstract Interpretation The least fixpoint (lfp) of an operator F on a poset $\langle P, \leqslant \rangle$, when it exists, is denoted by $\mathrm{lfp}^{\leqslant} F$, or by $\mathrm{lfp} F$ when $\leqslant$ is clear. Any continuous operator $F \in C \to C$ on a given complete lattice $\langle C, \leqslant, \sqcup, \sqcap, \top, \bot \rangle$ admits a lfp: $\mathrm{lfp}^{\leqslant}_{\bot} F = \bigvee_{n \in \mathbb{N}} F^i(\bot)$, where for any $i \in \mathbb{N}$ and $x \in C$: $F^0(x) = x$ and $F^{i+1}(x) = F(F^i(x))$. Given an abstract domain $\langle A, \leqslant_A \rangle$ of $\langle C, \leqslant_C \rangle$, $F^\# \in A \to A$ is a correct (sound) approximation of $F \in C \to C$ when $\alpha(\mathrm{lfp}^{\leqslant_C} F) \leqslant_A \mathrm{lfp}^{\leqslant_A} F^\#$. To this end it is enough to have a monotone map $\alpha : C \to A$ such that $\alpha(\bot_C) = \bot_A$ and $\alpha \circ F \leqslant_A F^\# \circ \alpha$ [12]. An abstraction is complete when $\alpha \circ F = F^\# \circ \alpha$. In this case of complete abstractions we have $\alpha(\mathrm{lfp}^{\leqslant_C} F) = \mathrm{lfp}^{\leqslant_A} F^\#$ [11, 21].

3 Indexed Languages

Indexed grammars were introduced by Aho in the late 1960s to model a natural subclass of context-sensitive languages, more expressive than context-free grammars with interesting closure properties [2]. In this paper we use the definition of indexed grammar provided in [1].

Definition 1. *An Indexed Grammar is a 5-tuple* $G = (N, T, I, P, S)$ *such that:*

(1) N, T *and* I *are three mutually disjoint finite sets of symbols: the set* N *of non-terminals, the set* T *of terminals and the set* I *of indices, where* ϵ *is a designated symbol for the empty sequence;*
(2) $S \in N$ *is a distinguished symbol in* N*, namely the start symbol;*
(3) P *is a finite set of productions, each having the form of one of the following:*
(a) $A \to \alpha$ *(Stack copy)*

(b) $A \to B_f$ *(Push)*
(c) $A_f \to \beta$ *(Pop)*
where $A, B \in N$ *are non-terminal symbols,* $f \in I$ *is an index symbol and* $\alpha, \beta \in (N \cup T)^*$.

Observe the similarity to context-free grammars which are only defined by production rules of type (*3a*). The above definition is a finite representation of rules that rewrite pairs of non-terminal and sequences of index symbols that we call stacks. A key feature of indexed grammars is that their productions in P expand non-terminal/stack pairs of the form (A, σ), where $A \in N$ and $\sigma \in I^*$. So each non-terminal symbol $A \in N$ together with its stack $\sigma \in I^*$, can be viewed as a pair (A, σ) and the start symbol S is shorthand for the pair (S, ϵ). Therefore, with a slight abuse of notation, in the rest of the paper we use α to denote a string of terminal symbols and non-terminals symbols with its stack, namely $\alpha \in ((N \times I^*) \cup T)^*$. The string α is often referred to as a *sentential form*. Given any non-empty stack $\sigma \in I^+$, the top symbol is the left-most index. The stack is implicit and is copied, to all non-terminals only, when the production is applied. So, for example, the type (*3a*) production rule $A \to aBC$ is shorthand for $(A, \sigma) \to a(B, \sigma)(C, \sigma)$ with $A, B, C \in N$, $a \in T$ and $\sigma \in I^*$. A production rule of the form (*3b*) implements a push onto the stack while a production rule of the form (*3c*) encodes a pop off of the stack. For example, the production rule $A \to B_f$ applied to (A, σ) expands to $(B, f\sigma)$ where $f\sigma$ is the stack with the index $f \in I$ pushed on. Likewise, $A_f \to \beta$ can only be applied to (A, σ) if the top of the stack string σ is f. The result is β such that any non-terminal $B \in \beta$ is of the form (B, σ'), where σ' is the stack with the top character popped off. Push and Pop productions differ from the original definition given by Aho [2] in which, by Definition 1, at most one index symbol is loaded or unloaded in any production.

Let $G = (N, T, I, P, S)$ be an indexed grammar. A derivation in G is a sequence of strings $\alpha_1, \alpha_2, \ldots, \alpha_n$ with $\alpha_i \in ((N \times I^*) \cup T)^*$, where α_{i+1} is derived from α_i by the application of one production in P, written $\alpha_i \to_G \alpha_{i+1}$. The subscript G is dropped whenever G is clearly understood. Let $\to_G^*$ be the reflexive and transitive closure of $\to_G$ defined as usual: $\alpha \to_G^0 \alpha$ for $n \geqslant 0$, $\alpha \to_G^{n+1} \gamma$ if $\exists \beta : \alpha \to_G^n \beta$ and $\beta \to_G \gamma$; $\alpha \to_G^* \beta$ iff $\alpha \to_G^i \beta$ for some $i \geqslant 0$.

The language $\mathcal{L}(G)$ recognized by an indexed grammar G is

$$\mathcal{L}(G) = \{w \in T^* \mid (S, \epsilon) \to_G^* w\}.$$

We denote with *IL* the set of indexed languages.

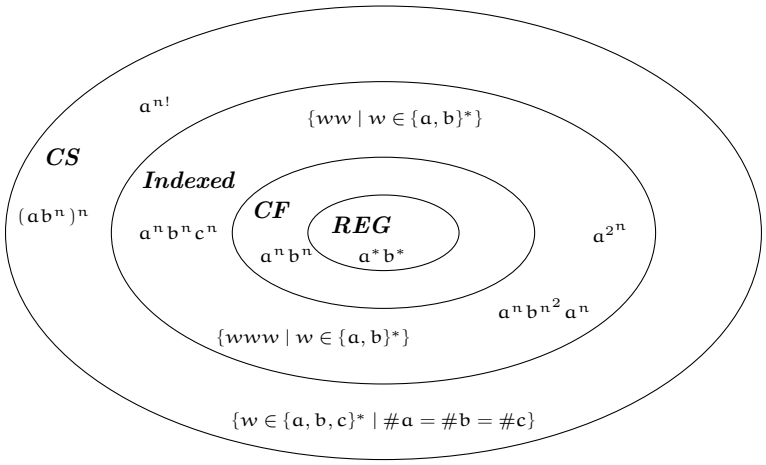
From now on the set of non-terminals N will range on superscript symbols such as $A, B, \ldots$, the set of terminals T on symbols a, b, c, d, e while indices I on f, g.

Example 1. The language $L = \{a^n b^n c^n \mid n \geqslant 1\}$ is generated by the indexed grammar $G = (\{S, T, A, B, C\}, \{a, b, c\}, \{f\}, P, S)$, with productions:

$$\begin{array}{lll} S \to T & A_f \to aA & B_\epsilon \to b \\ T \to T_f & A_\epsilon \to a & C_f \to cC \\ T \to ABC & B_f \to bB & C_\epsilon \to c \end{array}$$

For example, the word "$aabbcc$" can be generated by the following derivation: $(S, \epsilon) \to (T, \epsilon) \to (T, f) \to (A, f)(B, f)(C, f) \to a(A, \epsilon)(B, f)(C, f) \to aa(B, f)(C, f) \to^* aabbcc$.

Fig. 1. Chomsky hierarchy



Indexed languages are recognized by nested stack automata [3]. A nested stack automaton is a finite automaton that can make use of a stack containing data which can be additional stacks. Like a stack automaton, a nested stack automaton may step up or down in the stack, and read the current symbol; in addition, it may at any place create a new stack, operate on that one, eventually destroy it, and continue operating on the old stack. In this way, stacks can be nested recursively to an arbitrary depth; however, the automaton always operates on the innermost stack only. For more details on nested stack automata see [3].

As argued above, the class of indexed languages properly includes the one of CF languages, while being properly included in the one of CS languages. Fig. 1 represents these different classes and highlights some of the languages that characterize the different classes [24].

Table 1 reports some decidability and computational complexity properties of indexed languages and of the others formal languages in the Chomsky hierarchy. As expected, the decidability results of indexed languages lay in between the ones of CS and CF languages.

4 Fixpoint Characterization of Indexed Languages

In order to study the existence of abstraction functions between context-sensitive, indexed and context-free languages we need to provide a fixpoint characteriza-

Table 1. Decidability and complexity results

Class	Emptiness	Membership	Equivalence
Regular	$\mathbb{P}\ (O(n))$	$\mathbb{P}\ (O(n))$	NL-complete
Context-free	$\mathbb{P}\ (O(n^3))$	$\mathbb{P}\ (O(n^3))$	Undecidable
Indexed	EXP-complete	NP-complete	Undecidable
Context-sensitive	Undecidable	PSPACE-complete	Undecidable

tion of indexed languages. The fixpoint characterizations of CF languages, well known as the ALGOL-like theorem, and CS languages are already constructed and proved [22, 23].

The fixpoint characterization that we present is mainly derived from the one of CS languages [23]. Essentially it consists of two elements: a substitution function that simulates a context-free rule for the pair non-terminal/stack (productions of type (*3a*) and (*3b*)) and a regular expression to verify the context of the stack in case of a pop production (*3c*).

Before showing the theorem, let us give some notations and definitions. Let V be the set of variables of an indexed grammar: an element of V is either a pair of non-terminal/stack or a terminal symbol, namely $V = (N \times I^*) \cup T$. Therefore, if α is a sentential form of an indexed grammar, then $\alpha \in V^*$.

Definition 2. *An indexed state* $\mathbf{X}$ *is a m-tuple* $\mathbf{X} = (X_1, \ldots, X_m)$ *of sets of sentential forms* $X_i \in \wp(V^*)$ *with* $i \in [1, m]$.

Thus, the set of possible m-tuples of indexed states is $\underbrace{\wp(V^*) \times \cdots \times \wp(V^*)}_{m}$.

Definition 3. *A substitution function* h *is a map* $h : \wp(V^*) \to \wp(V^*)$.

Functions h will be defined in the proof of Theorem 1 and will be used to simulate an application of a CF-like production.

Regular sets over V are denoted by regular expressions $R \in \wp(V^*)$ and will be used to verify the top character of the stack for pop productions.

Definition 4. *Given a substitution function* h *and a regular set* R*, we define a pair* $\pi = (h, R)$ *called a* π*-function. A* π*-function is a map*

$$\pi : \underbrace{\wp(V^*) \times \cdots \times \wp(V^*)}_{m} \to \wp(V^*)$$

defined as follows:

$$\pi(\mathbf{X}) = h(\bigcup \mathbf{X} \cap R)$$

where $\bigcup \mathbf{X} = X_1 \cup \cdots \cup X_m$ *is the set corresponding to the union of all components of the indexed state* $\mathbf{X}$*. We denote with the bold symbol*

$$\boldsymbol{\pi} : \underbrace{\wp(V^*) \times \cdots \times \wp(V^*)}_{m} \to \underbrace{\wp(V^*) \times \cdots \times \wp(V^*)}_{m}$$

a vector function of π*-functions* $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_m)$ *such that*

$$\boldsymbol{\pi}(\mathbf{X}) = (\pi_1(\mathbf{X}), \ldots, \pi_m(\mathbf{X})).$$

As we will see in the proof of theorem 1, a π-function allows us to simulate an application of an indexed production.

Theorem 1. *Let* G *be an indexed grammar, then* $\mathcal{L}(G)$ *is a component of the least fixpoint of a system of equations on indexed states:*

$$\mathbf{X}_G^{j+1} = \boldsymbol{\pi}_G(\mathbf{X}_G^j) \tag{1}$$

where $\mathbf{X}_G^0 = (\underbrace{\{(S, \epsilon)\}, \emptyset, \ldots, \emptyset}_{n})$ *is the initial indexed state and the vector function of* π*-functions induced by* G *is* $\boldsymbol{\pi}_G = (\pi_{G,1}, \ldots, \pi_{G,n})$.

Proof. The proof considers an indexed grammar $G = (N, T, I, P, S)$ with m productions in P. Let $[\tau]_i \in P$ be an enumeration of all productions in P with $i \in [1, m]$, where $\tau \in P$ could be in one of the three forms of Definition 1. We build a system of equations having the form (1) where each indexed state $\mathbf{X}_G$ has $n = m + 2$ components: one for each production in P, namely $X_{G,1}, \ldots, X_{G,m}$ and two additional components $X_{G,0}$ and $X_{G,t}$, where t is a variable symbol denoting the "terminals" set and it is always at position $m + 1$ of $\mathbf{X}_G$. So the components of each indexed state are $\mathbf{X}_G = (X_{G,0}, X_{G,1}, \ldots, X_{G,m}, X_{G,t})$. The least fixpoint computation of the so obtained system of equations is calculated by the iterative application of the vector function $\boldsymbol{\pi}_G = (\pi_{G,0}, \pi_{G,1}, \ldots, \pi_{G,m}, \pi_{G,t})$ associated to the grammar G. The sets from $X_{G,1}$ to $X_{G,m}$ are associated to the m productions in P while $X_{G,t}$ contains only terminal symbols and $X_{G,0}$ initialize a sentential form. In particular, given the initial indexed state $\mathbf{X}_G^0$, the language is iteratively built in the last element $X_{G,t}$ of $\mathbf{X}_G$ such that at fixpoint $X_{G,t}^{FIX} = \mathcal{L}(G)$. From now on, the subscript G is dropped whenever it is clearly understood.

We introduce a barred version of the set of non-terminals N: $\bar{N} = \{\bar{A} \mid A \in N\}$ where $\bar{A}$ is the corresponding "marked" non-terminal to A. We also extend the set of variables V in order to contain marked non-terminals: $V = (N \cup \bar{N}, I^*) \cup T$. Marked non-terminals are the only symbols which can be rewritten by an indexed production.

In detail, given an indexed grammar G, the vector of π-functions induced by G is $\boldsymbol{\pi}_G = (\pi_0, \pi_1, \ldots, \pi_m, \pi_t)$. For $i \in [1, m]$, we associate the π_i-function $\pi_i = (h_i, R_i)$ with the i-th production in the enumeration of P. Each substitution function h_i, with $i \in [1, m]$, is defined inductively as follows where $\alpha, \beta \in V^*$ and $\sigma \in I^*$:

$h_i(\emptyset) = \emptyset$
$h_i(\{\epsilon\}) = \{\epsilon\}$
$h_i(\{a\}) = \{a\}$, if $a \in T$

$$h_i(\{(\bar{A},\sigma)\}) = \begin{cases} \{\alpha\} & \text{if } (\bar{A},\sigma) \in V \text{ and } [\,A \to \alpha\,]_i \in P \text{ } \textit{(Stack copy rule)} \\ \{(B, f\sigma)\} & \text{if } (\bar{A},\sigma) \in V \text{ and } [\,A \to B_f\,]_i \in P \text{ } \textit{(Push rule)} \\ \{\beta\} & \text{if } (\bar{A},\sigma) \in V \text{ and } [\,A_f \to \beta\,]_i \in P \text{ } \textit{(Pop rule)} \\ \{(\bar{A},\sigma)\} & \text{otherwise} \end{cases}$$

$h_i(\{\alpha Y\}) = h_i(\{\alpha\})h_i(\{Y\})$, if $\alpha \in V^+$ and $Y \in V$

$h_i(Q) = \bigcup_{\alpha\in Q} h_i(\{\alpha\})$, if $Q \in \wp(V^*)$, $Q \neq \emptyset$ and $\alpha \notin Q$.

Intuitively, the substitution function h_i will apply the i-th production to the marked non-terminal corresponding to the non-terminal of the associated production, without checking the stack symbols, i.e. in a context-free way. The other non-terminals remain untouched.

Given G, each regular expression R_i associated to the i-th production of P, with $i \in [1, m]$, is defined as follows, with $f \in I$ and $\sigma \in I^*$:

$$R_i = \begin{cases} V^*(\bar{A}, f\sigma)V^* & \text{if } [\,A_f \to \beta\,]_i \in P \text{ } \textit{(Pop rule)} \\ V^* & \text{otherwise.} \end{cases}$$

Intuitively, if the i-th indexed production $A_f \to \beta$ associated to R_i is of type (*3c*), then only the sentential forms containing the signed non-terminal $\bar{A}$ and having f as the top symbol of its stack, will be selected from intersection.

Now an application of $\pi_i = (h_i, R_i)$, with $i \in [1, m]$, to an indexed state corresponds to an application of the i-th indexed production.

We define the π-function π_0: its role is to mark the leftmost non-terminal of each sentential form (this marked non-terminal is the one used in the next iteration). Formally, $\pi_0 = (h_0, R_0)$ is inductively defined as follows for $\alpha \in V^+$:

$h_0(\emptyset) = \emptyset$

$h_0(\{\epsilon\}) = \{\epsilon\}$

$$h_0(\{Y\alpha\}) = \begin{cases} Y\, h_0(\{\alpha\}) & \text{if } Y \in T \\ (\bar{A},\sigma)\, \mathtt{unmark}(\{\alpha\}) & \text{if } Y = (A,\sigma) \text{ and } A \in N \\ (A,\sigma)\, h_0(\{\alpha\}) & \text{if } Y = (\bar{A},\sigma) \text{ and } \bar{A} \in \bar{N} \end{cases}$$

$h_0(Q) = \bigcup_{\alpha\in Q} h_0(\{\alpha\})$, if $Q \in \wp(V^*)$, $Q \neq \emptyset$ and $\alpha \notin Q$.

and function $\mathtt{unmark} : \wp(V^*) \to \wp(V^*)$ differs from h_0 in:

$$\mathtt{unmark}(\{Y\alpha\}) = \begin{cases} (A,\sigma)\, \mathtt{unmark}(\{\alpha\}) & \text{if } Y = (\bar{A},\sigma) \text{ and } \bar{A} \in \bar{N} \\ Y\, \mathtt{unmark}(\{\alpha\}) & \text{otherwise.} \end{cases}$$

The regular expression associated to h_0 is $R_0 = V^*$. Function h_0 marks the leftmost unmarked non-terminal while it unmarks any previously marked ones.

We define the π-function π_t to be applied to the last element of the state $\mathbf{X}$ that collects in X_t all the terminal words. Formally, $\pi_t = (h_t, R_t)$ where h_t is the identity function, namely $h_t = id$, and $R_t = T^*$. This leads us to the following system of equations where $0 \leqslant i \leqslant m$ and $j \geqslant 1$:

$$\begin{cases} \mathbf{X}^0 = (\{(S,\epsilon)\}, \emptyset, \ldots, \emptyset) \\ X_t^{j+1} = \pi_t(\mathbf{X}^j) \\ X_i^{j+1} = \pi_i(\mathbf{X}^j) \end{cases}$$

Let $(S, \epsilon) \rightarrow^n w$, with $w \in T^*$ and $n \geqslant 1$, be a derivation of G after n steps. We can construct a sequence of π-functions that exactly simulate the derivations in G such that the word $w \in X^{2n+1}_{G,t}$. Indeed, an application of the i-th production in P is exactly simulated by an application of two π-functions: $\pi_{G,0}$ to mark the non-terminal used in the production and $\pi_{G,i}$ to apply the i-th index production. After $2n$ steps, an application of the π-function $\pi_{G,t}$ at step $2n+1$ yields $w \in X^{2n+1}_{G,t}$.

Conversely it is straightforward to show by induction on n that, for every $w \in T^*$, if $w \in X^n_{G,t}$, with $n \geqslant 1$, then there exists a derivation in G yielding w after $(n-1)/2$ steps, namely $(S, \epsilon) \rightarrow^{(n-1)/2} w$. This means that $X^{FIX}_{G,t} = \mathcal{L}(G)$. □

Example 2. Consider the indexed language $L = \{a^n b^n c^n \mid n \geqslant 1\}$ and the indexed grammar $G = (\{S, T, A, B, C\}, \{a, b, c\}, \{f\}, P, S)$ generating it presented in Example 1. Let the productions in P be enumerated as follow:

$$\begin{array}{lll} [\, S \to T \,]_1 & [\, A_f \to aA \,]_4 & [\, B_\epsilon \to b \,]_7 \\ [\, T \to T_f \,]_2 & [\, A_\epsilon \to a \,]_5 & [\, C_f \to cC \,]_8 \\ [\, T \to ABC \,]_3 & [\, B_f \to bB \,]_6 & [\, C_\epsilon \to c \,]_9 \end{array}$$

We denote a substitution as a list of replacements, e.g. $\{(\bar{S}, \sigma) \to (T, \sigma)\}$ denotes the substitution h_1 defined by $h_1(\{(\bar{S}, \sigma)\}) = \{(T, \sigma)\}$ and identity otherwise. Following Theorem 1, the fixpoint characterization of the indexed grammar of Example 1 is:

$$\begin{aligned}
X^{j+1}_0 &= \pi_0(h_0, R_0) = h_0(V^* \cap \bigcup \mathbf{X}^j) \\
X^{j+1}_1 &= \pi_1(h_1, R_1) = \{(\bar{S}, \sigma) \to (T, \sigma)\}(V^* \cap \bigcup \mathbf{X}^j) \\
X^{j+1}_2 &= \pi_2(h_2, R_2) = \{(\bar{T}, \sigma) \to (T, f\sigma)\}(V^* \cap \bigcup \mathbf{X}^j) \\
X^{j+1}_3 &= \pi_3(h_3, R_3) = \{(\bar{T}, \sigma) \to (A, \sigma)(B, \sigma)(C, \sigma)\}(V^* \cap \bigcup \mathbf{X}^j) \\
X^{j+1}_4 &= \pi_4(h_4, R_4) = \{(\bar{A}, f\sigma) \to a(A, \sigma)\}(V^*(\bar{A}, f\sigma)V^* \cap \bigcup \mathbf{X}^j) \\
X^{j+1}_5 &= \pi_5(h_5, R_5) = \{(\bar{A}, \epsilon) \to a\}(V^*(\bar{A}, \epsilon)V^* \cap \bigcup \mathbf{X}^j) \\
X^{j+1}_6 &= \pi_6(h_6, R_6) = \{(\bar{B}, f\sigma) \to b(B, \sigma)\}(V^*(\bar{B}, f\sigma)V^* \cap \bigcup \mathbf{X}^j) \\
X^{j+1}_7 &= \pi_7(h_7, R_7) = \{(\bar{B}, \epsilon) \to b\}(V^*(\bar{B}, \epsilon)V^* \cap \bigcup \mathbf{X}^j) \\
X^{j+1}_8 &= \pi_8(h_8, R_8) = \{(\bar{C}, f\sigma) \to c(C, \sigma)\}(V^*(\bar{C}, f\sigma)V^* \cap \bigcup \mathbf{X}^j) \\
X^{j+1}_9 &= \pi_9(h_9, R_9) = \{(\bar{C}, \epsilon) \to c\}(V^*(\bar{C}, \epsilon)V^* \cap \bigcup \mathbf{X}^j) \\
X^{j+1}_t &= \pi_t(h_t, R_t) = (T^* \cap \bigcup \mathbf{X}^j)
\end{aligned}$$

where $\sigma \in I^*$, $\mathbf{X}^0 = (\{(S, \epsilon)\}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$ and the union of all components is given by $\bigcup \mathbf{X}^j = \bigcup \{X^j_y \mid y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, t\}\}$. We have $X^{FIX}_t = \mathcal{L}(G) = \{a^n b^n c^n \mid n \geqslant 1\}$.

For example, the word "$aabbcc$" can be generated in X_t, after 19 steps, by the following derivation:

$$aabbcc \in \pi_t \pi_9 \pi_0 \pi_8 \pi_0 \pi_7 \pi_0 \pi_6 \pi_0 \pi_5 \pi_0 \pi_4 \pi_0 \pi_3 \pi_0 \pi_2 \pi_0 \pi_1 \pi_0(\mathbf{X}^0)$$

5 Abstract Indexed Grammars

We want to investigate if the relation that exists between regular, CF, indexed and CS languages can be expressed as GIs, namely if less expressive languages can be seen as abstractions of more expressive ones.

Given a finite set of alphabet symbols Σ, we consider the complete lattice of all possible languages on Σ, namely $\langle \wp(\Sigma^*), \subseteq, \cup, \cap, \Sigma^*, \emptyset \rangle$. Suppose that we want to model the relation between Indexed and CF languages as a GI. This means that we want to abstract an indexed language into the best (w.r.t. set inclusion) CF language that includes it. However, this is not possible since CF languages are not closed under intersection, and therefore the abstract domain of CF languages $\langle CF, \subseteq \rangle$ is not a Moore family. The same holds when analyzing the relation between CS and indexed languages, and the one between CF and regular languages: the families of indexed languages and of regular languages do not form a Moore family of $\langle \wp(\Sigma^*), \subseteq \rangle$, as shown in the following three examples.

Example 3. Consider the following family of languages: $\forall i \geqslant 0 : L_i = \overline{\{a^i b^i\}}$. Each set L_i is a regular language since its complement language $\overline{L_i}$ is a finite set and regular languages are closed under complement operation, this means that $\forall i \geqslant 0 : L_i \in REG$. Taking the intersection of all L_i, namely $L = \bigcap_{i=0}^{\infty} L_i$, we get $L = \overline{\{a^n b^n \mid n \geqslant 0\}}$ where the $\overline{w}$ is the complement operation. $L \in CF$ since it can be created from the union of several simpler languages:

$$L = \{a^i b^j \mid i > j\} \cup \{a^i b^j \mid i < j\} \cup (a \cup b)^* b (a \cup b)^* a (a \cup b)^*$$

that is, all strings of as followed by bs in which the number of as and bs differ, joined with all strings not of the form $a^i b^j$. The language $\{a^i b^j \mid i > j\} \in CF$ and a CF grammar generating it is $S \to aSb \mid aS \mid a$ similarly $\{a^i b^j \mid i < j\} \in CF$, while $(a \cup b)^* b (a \cup b)^* a (a \cup b)^* \in REG$ since it is a regular expression. Observe that we have obtained a CF language from an (infinite) intersection of regular languages.

Example 4. Consider the following two CF languages and their corresponding CF grammars:

$$L_1 = \{a^n b^n c^m \mid n, m \geqslant 0\} \qquad L_2 = \{a^n b^m c^m \mid n, m \geqslant 0\}$$
$$S \to AC \qquad S \to AB$$
$$A \to aAb \mid \epsilon \qquad A \to aA \mid \epsilon$$
$$C \to cC \mid \epsilon \qquad B \to bBc \mid \epsilon$$

Note that $L_1 \cap L_2 = L = \{a^n b^n c^n \mid n \geqslant 0\}$ and L is an indexed language but not CF, namely $L \in IL \setminus CF$.

Example 5. Consider the following indexed languages $L_1 = \{w \in \{a, b, c\}^* \mid \#a = \#b\}$ and $L_2 = \{w \in \{a, b, c\}^* \mid \#b = \#c\}$ where $\#a$ means the number of symbols a in a word. L_1 can be generated by the following CF grammar:

$$S \to SS \qquad S \to aSb \qquad S \to bSa \qquad S \to c \qquad S \to \epsilon$$

and similarly for L_2. Consider the language

$$L = L_1 \cap L_2 = \{w \in \{a, b, c\}^* \mid \#a = \#b = \#c\}.$$

$L \in \mathit{CS}$ but $L \notin \mathit{IL}$. A CS grammar generating L is:

$$\begin{array}{llllll} S \to ABC & AB \to BA & BC \to CB & CA \to AC & A \to a & C \to c \\ S \to ABCS & AC \to CA & BA \to AB & CB \to BC & B \to b & \end{array}$$

Observe that we have obtained a CS language from an intersection of two indexed languages.

The examples above show that it is not possible to specify GIs between the domains of languages in the Chiomsky hierarchy. However, this does not exclude the possibility of approximating the fixpoint semantics of indexed grammars by acting on the productions of the grammars. This corresponds to constraining the structures of productions or the way the memory (stack) of the productions of indexed grammars are used, by acting on the indexed states of the equational characterization associated. We provide abstractions of indexed grammars, namely of the mechanism used to generate the indexed languages, with the aim of transforming an indexed language into a more abstract (namely a less expressive) language in Chomsky's hierarchy, such as CF or REG languages. We start in section 5.1 with a simple abstraction, stack elimination, which eliminates completely the stack of all non-terminals. Then, with the purpose of refining the abstraction, we present two other abstractions: stack limitation (section 5.2), which limits the stack capacity, and stack copy limitation (section 5.3), which limits stack copy productions.

5.1 Stack Elimination

Definition 5. *Stack elimination removes the stack of each non-terminal in a sentential form of an indexed grammar. Namely, given a sentential form* α*, each pair* $(A, \sigma) \in \alpha$*, with* $A \in N$ *and* $\sigma \in I^*$*, is replaced by* (A, I^*).

The idea of stack elimination is to abstract away from the stack, namely, to abstract the stack to top in the sentential form. In the concrete domain, the abstract value top of the stack precisely corresponds to the set of all possible stacks. In other words, this corresponds to have a set of sentential forms one for each possible stack value. A major consequence of applying stack elimination is that the three kinds of indexed productions (stack copy, push and pop) in an indexed grammar are turned into a single context-free production.

We want to demonstrate that stack elimination is a sound abstraction of indexed grammars. In the following we denote with Φ_G the domain $\wp(V^*)^m$ of possible indexed states of an indexed grammar $G = (N, T, I, P, S)$ with m productions in P, used in the fixpoint characterization of Theorem 1. It is possible to define a function on the concrete domain $\langle \Phi_G, \sqsubseteq \rangle$ that iteratively computes

the language of an indexed grammar G. The partial order $\sqsubseteq$ over Φ_{G} is defined as follows:

$$\forall \mathbf{X}, \mathbf{Y} \in \Phi_{\mathsf{G}} : \mathbf{X} \sqsubseteq \mathbf{Y} \iff \mathtt{proj}_{-1}(\mathbf{X}) \subseteq \mathtt{proj}_{-1}(\mathbf{Y})$$

The transition relation between two indexed states $\mathbf{X}^{\mathsf{i}}, \mathbf{X}^{\mathsf{i}+1} \in \Phi_{\mathsf{G}}$ corresponds to the application of the vector function $\boldsymbol{\pi}_{\mathsf{G}} : \Phi_{\mathsf{G}} \to \Phi_{\mathsf{G}}$, namely $\mathbf{X}^{\mathsf{i}+1} = \boldsymbol{\pi}_{\mathsf{G}}(\mathbf{X}^{\mathsf{i}})$.

We formalize the abstract domain as a closure $\rho^{\mathsf{E}} : \Phi_{\mathsf{G}} \to \Phi_{\mathsf{G}}$ on $\langle \Phi_{\mathsf{G}}, \sqsubseteq \rangle$, as follows:

$$\rho^{\mathsf{E}}(\mathbf{X}) = (\rho^{\mathsf{E}}(X_1), \ldots, \rho^{\mathsf{E}}(X_m))$$

and, with a slight abuse of notation, $\rho^{\mathsf{E}}(X_{\mathsf{i}}) = \{\rho^{\mathsf{E}}(s_{\mathsf{i}}) \mid s_{\mathsf{i}} \in X_{\mathsf{i}}\}$ where for $s_{\mathsf{i}} = \lambda_{\mathsf{i}1} \ldots \lambda_{\mathsf{i}w}$ with $\lambda_{\mathsf{i}\mathsf{j}} \in V$ we have:

$$\rho^{\mathsf{E}}(\lambda_{\mathsf{i}1})\rho^{\mathsf{E}}(\lambda_{\mathsf{i}2} \ldots \lambda_{\mathsf{i}w}) = \begin{cases} (A, \mathrm{I}^*) \text{ if } \lambda_{\mathsf{i}1} = (A, \sigma) \\ \lambda_{\mathsf{i}1} \qquad \text{otherwise.} \end{cases}$$

Intuitively, the stack of all non-terminal symbols is set to I^*. This means that there is no restrictions on the symbol on the top of the stack when performing a pop operation, turning push and pop productions to stack copy productions.

Lemma 1. *The function ρ^{E} on domain $\langle \Phi_{\mathsf{G}}, \sqsubseteq \rangle$ is an upper closure operator. Moreover, we have that $\mathsf{lfp}^{\sqsubseteq} \boldsymbol{\pi}_{\mathsf{G}}(\mathbf{X}^0) = \rho^{\mathsf{E}} \circ \mathsf{lfp}^{\sqsubseteq} \boldsymbol{\pi}_{\mathsf{G}}(\mathbf{X}^0)$ while $\mathsf{lfp}^{\sqsubseteq} \boldsymbol{\pi}_{\mathsf{G}}(\mathbf{X}^0) \sqsubseteq \mathsf{lfp}^{\sqsubseteq} \boldsymbol{\pi}_{\mathsf{G}}(\rho^{\mathsf{E}}(\mathbf{X}^0))$.*

Lemma 2. *Let $\widetilde{\mathsf{G}}$ be the indexed grammar obtained by stack elimination from an indexed grammar G, then $\mathsf{lfp}^{\sqsubseteq} \boldsymbol{\pi}_{\widetilde{\mathsf{G}}}(\mathbf{X}^0) = \mathsf{lfp}^{\sqsubseteq} \boldsymbol{\pi}_{\mathsf{G}}(\rho^{\mathsf{E}}(\mathbf{X}^0))$.*

This allows us to prove the soundness of the abstraction showing that any language obtained by stack elimination from an indexed grammar is an over approximation of its original indexed language:

Theorem 2. *Let $\widetilde{\mathsf{G}}$ be the indexed grammar obtained by stack elimination from the indexed grammar G, then $\mathcal{L}(\mathsf{G}) \subseteq \mathcal{L}(\widetilde{\mathsf{G}})$.*

The loss of precision is due to the fact that, when eliminating the stack, an indexed grammar can no longer count or store occurrences of an index symbol, thus it is reduced to a CF grammar. Moreover, it turns out that if the original indexed grammar G is such that $\mathcal{L}(\mathsf{G}) \in \mathit{IL}$ but $\mathcal{L}(\mathsf{G}) \notin \mathit{CF}$ then $\mathcal{L}(\widetilde{\mathsf{G}})$ is a CF language but not indexed.

Corollary 1. *Let $\widetilde{\mathsf{G}}$ be the indexed grammar obtained from G by stack elimination such that $\mathcal{L}(\mathsf{G}) \in \mathit{IL} \setminus \mathit{CF}$, then $\mathcal{L}(\widetilde{\mathsf{G}}) \notin \mathit{IL} \setminus \mathit{CF}$.*

Example 6. The language $\mathsf{L} = \{a^n b^n c^n \mid n \geqslant 1\}$ in Example 1 is an indexed language but not CF, namely $\mathsf{L} \in \mathit{IL} \setminus \mathit{CF}$. If we apply stack elimination as described above, we obtain a new language $\mathcal{L}(\widetilde{\mathsf{G}})$ generated by the new grammar $\widetilde{\mathsf{G}}$ with the following productions:

$$\begin{array}{lll} S \to T & A \to aA & B \to b \\ T \to T & A \to a & C \to cC \\ T \to ABC & B \to bB & C \to c \end{array}$$

The language generated from $\widetilde{G}$ is $\mathcal{L}(\widetilde{G}) = \{a^*b^*c^*\}$ and it is a regular language, $\mathcal{L}(\widetilde{G}) \in \mathsf{REG}$.

Although some examples may be deceiving, in general it is not true that any indexed languages become regular by stack elimination. Indeed, indexed grammars could contain context-free characteristic rules that do not affect stacks and so, after stack elimination, still remain, turning the language to a CF language and not regular. The following is an example of such an indexed grammar.

Example 7. The language $L = \{a^n b^n c^n \mid n \geqslant 1\}$ could be generated also by the following indexed grammar:

$$\begin{array}{ll} S \to aS_f c & T_f \to Tb \\ S \to T & T_\epsilon \to \epsilon \end{array}$$

If we apply stack elimination, we obtain the language $\widetilde{L} = \{a^n b^* c^n \mid n \geqslant 1\}$. Note that $L \subseteq \widetilde{L}$ and $\widetilde{L} \in \mathit{CF}$ but $\widetilde{L} \notin \mathit{REG}$.

It is indeed obvious to observe that stack elimination produces precisely the class of CF languages.

Corollary 2. *For any CF grammar* $\widetilde{G}$ *there exists an indexed grammar* G *such that:* $\mathrm{lfp}^{\sqsubseteq} \boldsymbol{\pi}_{\widetilde{G}}(\mathbf{X}^0) = \mathrm{lfp}^{\sqsubseteq} \boldsymbol{\pi}_G(\rho^E(\mathbf{X}^0))$.

5.2 Stack Limitation

Definition 6. *Stack limitation limits the numbers of symbols on the stack of each non-terminal by a constant* $k \geqslant 0$. *This means that each stack can contain at most* k *symbols and all others* $k+1$ *symbols pushed on to the stack will be discarded.*

We want to demonstrate that stack limitation is a sound abstraction of indexed grammars. As in the previous section, we operate on the concrete domain $\langle \Phi_G, \sqsubseteq \rangle$. We formalize the abstract domain as an upper closure operator ρ^L_k, with $k \geqslant 0$, on the concrete domain $\langle \Phi_G, \sqsubseteq \rangle$. We define $\rho^L_k : \Phi_G \to \Phi_G$:

$$\rho^L_k(\mathbf{X}) = (\rho^L_k(X_1), \ldots, \rho^L_k(X_m))$$

and, with a slight abuse of notation, $\rho^L_k(X_i) = \{\rho^L_k(s_i) \mid s_i \in X_i\}$ where for $s_i = \lambda_{i1} \ldots \lambda_{iw}$ with $\lambda_{ij} \in V$ we have:

$$\rho^L_k(\lambda_{i1})\rho^L_k(\lambda_{i2} \ldots \lambda_{iw}) = \begin{cases} (A, \hat{\sigma}) \text{ if } \lambda_{i1} = (A, \sigma) \text{ and } |\sigma| > k \\ \lambda_{i1} \quad \text{otherwise.} \end{cases}$$

where $|\sigma| = z$ if $\sigma = q_z \ldots q_{k+1} q_k \ldots q_1$ and $|\epsilon| = 0$ with $\sigma \in I^*$, and for $1 \leqslant i \leqslant z$, $q_i \in I$, q_z top symbol and $\hat{\sigma} = q_k \ldots q_1$. Intuitively, by function ρ^L_k, the stack of each non-terminal is limited to k symbols and each additional push of others symbols will be discarded. Observe that this technique corresponds to limiting push productions only.

Lemma 3. *The function* ρ^{L}_{k} *on the domain* $\langle \Phi_G, \sqsubseteq \rangle$ *is an upper closure operator. Moreover,* $\mathtt{lfp}^{\sqsubseteq}\, \boldsymbol{\pi}_G(\mathbf{X}^0) = \rho^{L}_{k} \circ \mathtt{lfp}^{\sqsubseteq}\, \boldsymbol{\pi}_G(\mathbf{X}^0)$.

Lemma 4. *Let* $\widetilde{G}_k$ *be the indexed grammar obtained by stack limitation with the constant* k *from an indexed grammar* G*, then* $\mathtt{lfp}^{\sqsubseteq}\, \boldsymbol{\pi}_{\widetilde{G}_k}(\mathbf{X}^0) = \mathtt{lfp}^{\sqsubseteq}\, \boldsymbol{\pi}_G(\rho^{L}_{k}(\mathbf{X}^0))$.

In the following theorem we prove that stack limitation, as defined by function ρ^{L}_{k}, is not sound, namely, at fixpoint, the language generated is not always an over approximation of the original concrete language.

Theorem 3. $\mathtt{lfp}^{\sqsubseteq}\, \boldsymbol{\pi}_G(\mathbf{X}^0) \not\sqsubseteq \mathtt{lfp}^{\sqsubseteq}\, \boldsymbol{\pi}_G(\rho^{L}_{k}(\mathbf{X}^0))$.

Proof. The proof is made by providing a counterexample. The language $L = \{a^n b^n c^n \mid n \geqslant 1\}$ in Example 1 has only one push production: $T \to T_f$. Therefore, after stack limitation, each stack can contain at most k index symbols of f and the following family of languages is generated:

$$\forall k \geqslant 0\ ,\ L_k = \{a^n b^n c^n \mid 1 \leqslant n \leqslant k+1\}$$

For all $k \geqslant 0$, the language L_k is a regular language since each family contains a finite number of words: for $k = 0$ then $\{abc\}$, $k = 1$ then $\{abc, aabbcc\}$, ... Moreover, each L_k is not an over approximation of the original language since $L \not\subseteq L_k$, this leads to $\mathtt{lfp}^{\sqsubseteq}\, \boldsymbol{\pi}_G(\mathbf{X}^0) \not\sqsubseteq \mathtt{lfp}^{\sqsubseteq}\, \boldsymbol{\pi}_G(\rho^{L}_{k}(\mathbf{X}^0))$.

□

Observe that the infinite intersection of all family of languages L_k obtained by stack limitation from a language L, corresponds to L_0, namely $\bigcap_{k=0}^{\infty} L_k = L_0$, while the infinite union of all L_k is a superset of the original language L, namely $L \subseteq \bigcup_{k=0}^{\infty} L_k$.

At first glance, the family of languages generated from a language non-stack-limited is regular, as the previous example showed but, in general, this is not always true: the next example shows a counterexample, similar to Example 7:

Example 8. The language $L = \{a^n b^n c^n \mid n \geqslant 1\}$ could be generated also by the following indexed grammar:

$$\begin{array}{ll} S \to aS_f c & T_f \to Tb \\ S \to T & T_\epsilon \to \epsilon \end{array}$$

If we apply stack limitation we get the following family of languages:

$$\forall k \geqslant 1\ ,\ L_k = \{a^n b^m c^n \mid n \geqslant 1\ \wedge\ m \leqslant k\}$$

Note that $\forall k \geqslant 1\ L_k \in \mathit{CF}$ while for $k = 0\ \ L_0 = \{\epsilon\}$ the empty word ϵ is not accepted by L.

We can force the soundness of stack limitation abstraction by modifying the upper closure operator ρ^{L}_{k} as follows, obtaining the new $\mathtt{uco}\ \rho^{LE}_{k}$:

$$\rho^{LE}_{k}(\mathbf{X}) = \cdots = \rho^{LE}_{k}(\lambda_{ij}) = \begin{cases} (A, I^*) & \text{if } \lambda_{i1} = (A, \sigma) \text{ and } |\sigma| > k \text{ or } k = 0 \\ \lambda_{i1} & \text{otherwise.} \end{cases}$$

Intuitively, ρ^{LE}_{k} eliminates completely the stack of a non-terminal if and only if $k = 0$ or the size of the stack exceeds the parameter k.

Lemma 5. $\mathsf{lfp}^{\sqsubseteq}\,\boldsymbol{\pi}_{\mathsf{G}}(\mathbf{X}^0) \sqsubseteq \mathsf{lfp}^{\sqsubseteq}\,\boldsymbol{\pi}_{\mathsf{G}}(\rho_{\mathsf{k}}^{\mathsf{LE}}(\mathbf{X}^0))$.

By Lemma 5 and by substituting $\rho_{\mathsf{k}}^{\mathsf{L}}$ with the new $\mathfrak{uco}$ $\rho_{\mathsf{k}}^{\mathsf{LE}}$ in Lemma 3 and Lemma 4, we can prove the soundness of this new abstraction.

Theorem 4. *Let* $\widetilde{\mathsf{G}}_{\mathsf{k}}$ *be the indexed grammar obtained by stack limitation with the constant* k *from an indexed grammar* G*, then* $\mathcal{L}(\mathsf{G}) \subseteq \mathcal{L}(\widetilde{\mathsf{G}}_{\mathsf{k}})$.

5.3 Stack Copy Limitation

Definition 7. *Stack copy limitation limits the copy of the stack, from the right-side of a production, to a finite number of non-terminals symbols in a given set* H*. The contents of the other stacks are set to* I^* *meaning that you can do push and pop operations with no limits as in section 5.1.*

As before, we want to demonstrate that stack copy limitation is a sound abstraction of indexed grammars. As in the previous sections, we formalize the abstract domain as an upper closure operator $\rho_{\mathsf{H}}^{\mathsf{C}}$ on the concrete domain $\langle \Phi_{\mathsf{G}}, \sqsubseteq \rangle$ with $\mathsf{H} \subseteq \mathsf{N}$ where N is the set of non-terminals of the indexed grammar. We define $\rho_{\mathsf{H}}^{\mathsf{C}} : \Phi_{\mathsf{G}} \to \Phi_{\mathsf{G}}$ as:

$$\rho_{\mathsf{H}}^{\mathsf{C}}(\mathbf{X}) = (\rho_{\mathsf{H}}^{\mathsf{C}}(\mathsf{X}_1), \ldots, \rho_{\mathsf{H}}^{\mathsf{C}}(\mathsf{X}_{\mathrm{m}}))$$

and, with a slight abuse of notation, $\rho_{\mathsf{H}}^{\mathsf{C}}(\mathsf{X}_{\mathrm{i}}) = \{\rho_{\mathsf{H}}^{\mathsf{C}}(\mathsf{s}_{\mathrm{i}}) \mid \mathsf{s}_{\mathrm{i}} \in \mathsf{X}_{\mathrm{i}}\}$ where for $\mathsf{s}_{\mathrm{i}} = \lambda_{\mathrm{i}1} \ldots \lambda_{\mathrm{i}w}$ with $\lambda_{\mathrm{ij}} \in \mathsf{V}$ we have:

$$\rho_{\mathsf{H}}^{\mathsf{C}}(\lambda_{\mathrm{i}1})\rho_{\mathsf{H}}^{\mathsf{C}}(\lambda_{\mathrm{i}2} \ldots \lambda_{\mathrm{i}w}) = \begin{cases} (\mathsf{A}, \mathrm{I}^*) \text{ if } \lambda_{\mathrm{i}1} = (\mathsf{A}, \sigma) \text{ and } \mathsf{A} \notin \mathsf{H} \\ \lambda_{\mathrm{i}1} \qquad \text{otherwise.} \end{cases}$$

Intuitively, the function $\rho_{\mathsf{H}}^{\mathsf{C}}$ eliminates the stack of only a restricted set of non-terminals, namely those not in the set H, while for all non-terminals in H the stack will be copied and so all the indices symbols on it still remain. Observe that if $\mathsf{H} = \mathsf{N}$ then $\rho_{\mathsf{H}}^{\mathsf{C}} = id$ where id is the identity function, while if $\mathsf{H} = \emptyset$ then $\rho_{\mathsf{H}}^{\mathsf{C}} = \rho^{\mathsf{E}}$ where ρ^{E} is the stack elimination technique presented in 5.1.

Lemma 6. *The function* $\rho_{\mathsf{H}}^{\mathsf{C}}$ *on the domain* $\langle \Phi_{\mathsf{G}}, \sqsubseteq \rangle$ *is an uco. Moreover, we have* $\mathsf{lfp}^{\sqsubseteq}\,\boldsymbol{\pi}_{\mathsf{G}}(\mathbf{X}^0) = \rho_{\mathsf{H}}^{\mathsf{C}} \circ \mathsf{lfp}^{\sqsubseteq}\,\boldsymbol{\pi}_{\mathsf{G}}(\mathbf{X}^0)$ *while* $\mathsf{lfp}^{\sqsubseteq}\,\boldsymbol{\pi}_{\mathsf{G}}(\mathbf{X}^0) \sqsubseteq \mathsf{lfp}^{\sqsubseteq}\,\boldsymbol{\pi}_{\mathsf{G}}(\rho_{\mathsf{H}}^{\mathsf{C}}(\mathbf{X}^0))$.

Lemma 7. *Let* $\widetilde{\mathsf{G}}_{\mathsf{H}}$ *be the indexed grammar obtained by stack copy limitation from an indexed grammar* G*, with* $\mathsf{H} \subseteq \mathsf{N}$ *where* N *is the set of non-terminals of* G*, then* $\mathsf{lfp}^{\sqsubseteq}\,\boldsymbol{\pi}_{\widetilde{\mathsf{G}}_{\mathsf{H}}}(\mathbf{X}^0) = \mathsf{lfp}^{\sqsubseteq}\,\boldsymbol{\pi}_{\mathsf{G}}(\rho_{\mathsf{H}}^{\mathsf{C}}(\mathbf{X}^0))$.

The following theorem asserts the soundness of the abstraction by showing that any language obtained by stack copy limitation from an indexed grammar is an over approximation of its original indexed language:

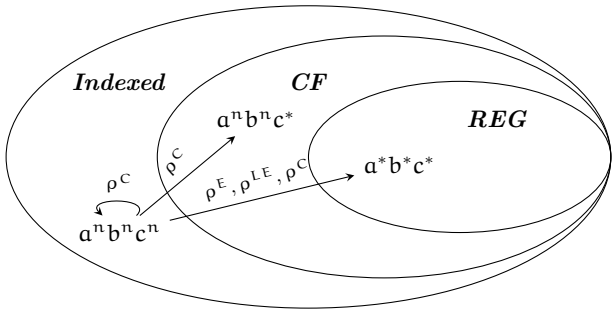
Theorem 5. *Let* $\widetilde{\mathsf{G}}_{\mathsf{H}}$ *be the indexed grammar obtained by stack copy limitation of a subset of non-terminals* H *from an indexed grammar* G*, then* $\mathcal{L}(\mathsf{G}) \subseteq \mathcal{L}(\widetilde{\mathsf{G}}_{\mathsf{H}})$.

As expected, the quality of this abstraction, in terms of classification in the Chomsky hierarchy, may be better then stack elimination, depending on which non-terminals form the set $H \subseteq N$.

Example 9. Consider the language $L = \{a^n b^n c^n \mid n \geqslant 1\}$ in Example 1 and let $H = \{A\}$, then by stack copy limitation we obtain: $\widetilde{L} = \{a^n b^* c^* \mid n \geqslant 1\}$. Observe that $\widetilde{L} \in \mathit{REG}$ and $L \subseteq \widetilde{L}$, indeed, if H contains one of the three non-terminals then stack copy limitation is equivalent to stack elimination. However, if $H = \{A, B\}$, then by stack copy limitation we obtain $\widetilde{L}' = \{a^n b^n c^* \mid n \geqslant 1\}$. Note that $L \subseteq \widetilde{L}' \subseteq \widetilde{L}$, $\widetilde{L}' \in \mathit{CF}$ and $\widetilde{L}' \notin \mathit{REG}$.

We conclude by showing in Fig. 5.3 the three sound abstractions presented in this section applied to Example 6.

Fig. 2. Sound abstractions of indexed grammars presented in section 5 and applied to Example 6



6 Related and Future Works

The approximation of grammar structures by abstract interpretation is not new. In [13] and [15] the authors introduced the idea of abstracting formal languages by abstract interpretation for the design of static analysers that manipulate symbolic structures. This provided both the source for new symbolic abstract domains for program analysis and the possibility of formalising known algorithms, such as parsers, as abstract interpreters. Abstractions into regular languages have been used in formal verification (e.g., see [7]). In program analysis non-regular approximations of formal languages have been used in aliasing analysis [19]. The idea of grammar abstraction as a relation between CF grammars has been also used for relating concrete and abstract syntax in [4]. We exploit this latter line of research by establishing a relation, formalised here by abstract interpretation, between indexed grammars with the aim of relating languages in Chomsky's hierarchy [6].

None of the above mentioned approaches considered the more general problem of correlating languages in Chomsky's hierarchy by the theory of fixpoint abstraction by abstract interpretation. We believe that a systematic reconstruction of Chomsky's hierarchy by fixpoint abstract interpretation may provide both new insights into a fundamental field of computer science and new algorithms and methods for approximating structures described by grammars. Indeed, the current work originated from the desire of finding suitable abstract domains for expressing the invariant properties among obfuscated malware variants [17, 18]. We reformulated the Chomsky's hierarchy by using the standard abstract interpretation methods: we provided a fixpoint semantics for indexed languages and we characterised classes of less expressive languages in terms of fixpoint abstractions of this semantics. In our case, the approximation of indexed languages shows how it is possible to systematically and constructively derive all fixpoint descriptions for CF languages as abstract interpretations. In particular, we proved that a *calculational design*, in the style of [8], of these fixpoint representation for CF languages is possible, and how new families of languages can be derived in this form. As future work we plan to generalize known separation results between classes of languages, e.g., the Pumping Lemmata, as instances of incompleteness of language abstractions. The idea is that, if a family of languages corresponds to a suitable abstraction of the fixpoint semantics of a more concrete family of languages–in our case indexed languages, then languages not expressible in one family should correspond to witnesses of the incompleteness of this abstraction [21]. The interest in this perspective over Chomsky's hierarchy is in the fact that we can reformulate most of this hierarchy, including separation results, in terms of abstract interpretation, providing powerful tools for comparing symbolic abstract domains with respect to their expressive power.

References

1. Adams, J., Freden, E., Mishna, M.: From indexed grammars to generating functions. RAIRO-Theoretical Informatics and Applications 47(4), 325–350 (2013)
2. Aho, A.V.: Indexed grammars – an extension of context-free grammars. Journal of the ACM 15(4), 647–671 (1968)
3. Aho, A.V.: Nested stack automata. Journal of the ACM 16(3), 383–406 (Jul 1969)
4. Ballance, R.A., Butcher, J., Graham, S.L.: Grammatical abstraction and incremental syntax analysis in a language-based editor. In: Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation. pp. 185–198. PLDI '88, ACM, New York, NY, USA (1988)
5. Bertsch, E.: On the relationship between indexed grammars and logic programs. The Journal of Logic Programming 18(1), 81–98 (1994)
6. Chomsky, N.: On certain formal properties of grammars. Information and Control 2(2), 137–167 (June 1959)
7. Clarke, E.M., Grumberg, O., Jha, S.: Verifying parameterized networks. ACM Trans. Program. Lang. Syst. 19(5), 726–750 (1997)
8. Cousot, P.: The calculational design of a generic abstract interpreter. In: Broy, M., Steinbrüggen, R. (eds.) Calculational System Design, vol. 173, pp. 421–505. NATO

Science Series, Series F: Computer and Systems Sciences. IOS Press, Amsterdam (1999)
9. Cousot, P.: Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. Theor. Comput. Sci. 277(1-2), 47–103 (2002)
10. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conference Record of the 4th ACM Symposium on Principles of Programming Languages (*POPL '77*). pp. 238–252. ACM Press (1977)
11. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Conference Record of the 6th ACM Symposium on Principles of Programming Languages (*POPL '79*). pp. 269–282. ACM Press (1979)
12. Cousot, P., Cousot, R.: Abstract interpretation frameworks. J. Logic and Comput. 2(4), 511–547 (1992)
13. Cousot, P., Cousot, R.: Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game-theoretic form (Invited Paper). In: Wolper, P. (ed.) Proc. of the 7th Internat. Conf. on Computer Aided Verification (*CAV '95*). Lecture Notes in Computer Science, vol. 939, pp. 293–308. Springer-Verlag (1995)
14. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: POPL '78: Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages. pp. 84–96. ACM Press (1978)
15. Cousot, P., Cousot, R.: Grammar semantics, analysis and parsing by abstract interpretation. Theor. Comput. Sci. 412(44), 6135–6192 (2011)
16. Cousot, P., Cousot, R.: Abstract interpretation: past, present and future. In: Henzinger, T.A., Miller, D. (eds.) Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014. pp. 2:1–2:10. ACM (2014)
17. Dalla Preda, M., Giacobazzi, R., Debray, S.K., Coogan, K., Townsend, G.M.: Modelling metamorphism by abstract interpretation. In: Proc. of the 19th Int. Static Analysis Symp. (*SAS '10*). Lecture Notes in Computer Science, vol. 6337, pp. 218–235. Springer-Verlag, Berlin (2010)
18. Dalla Preda, M., Giacobazzi, R., Debray, S.K.: Unveiling metamorphism by abstract interpretation of code properties. Theor. Comput. Sci. 577, 74–97 (2015)
19. Deutsch, A.: Interprocedural may-alias analysis for pointers: Beyond k-limiting. SIGPLAN Not. 29(6), 230–241 (Jun 1994)
20. Gazdar, G.: Applicability of indexed grammars to natural languages. In: Natural language parsing and linguistic theories, pp. 69–94. Springer (1988)
21. Giacobazzi, R., Ranzato, F., Scozzari., F.: Making abstract interpretation complete. Journal of the ACM 47(2), 361–416 (March 2000)
22. Ginsburg, S.: The Mathematical Theory of Context Free Languages. McGraw-Hill Book Company (1966)
23. Istrail, S.: Generalization of the Ginsburg-Rice Schützenberger fixed-point theorem for context-sensitive and recursive-enumerable languages. Theoretical Computer Science 18(3), 333–341 (1982)
24. Partee, B.B., ter Meulen, A.G., Wall, R.: Mathematical methods in linguistics, vol. 30. Springer Science & Business Media (2012)