First-order Logic Definability of Free Languages

Violetta Lonati[1], Dino Mandrioli[2], Federica Panella[2], Matteo Pradella[2]

[1] DI - Università degli Studi di Milano
lonati@di.unimi.it
[2] DEIB - Politecnico di Milano
{dino.mandrioli, federica.panella, matteo.pradella}@polimi.it

Abstract. Operator Precedence Grammars (OPGs) define a deterministic class of context-free languages, which extend input-driven languages and still enjoy many properties: they are closed w.r.t. Boolean operations, concatenation and Kleene star; the emptiness problem is decidable; they are recognized by a suitable model of pushdown automaton; they can be characterized in terms of a monadic second-order logic. Also, they admit efficient parallel parsing.
In this paper we introduce a subclass of OPGs, namely Free Grammars (FrGs); we prove some of its basic properties, and that, for each such grammar G, a first-order logic formula ψ can effectively be built so that $L(G)$ is the set of all and only strings satisfying ψ.
FrGs were originally introduced for grammatical inference of programming languages. Our result can naturally boost their applicability; to this end, a tool is made freely available for the semiautomatic construction of FrGs.

1 Introduction

Operator Precedence Grammars (OPGs) and their generated languages, Operator Precedence Languages (OPLs), have been invented by R. Floyd half a century ago with the purpose of building efficient deterministic parsers. Although they are still in use in this peculiar application field, thanks to their simplicity and the efficiency of their parsers [13], their theoretical investigation has been interrupted for a long time and only recently we resumed it in a long term research plan [8]. This led to discover many important properties of this class of languages which can be exploited in different modern applications. In fact, OPLs enable efficient parallel parsing algorithms [4] and are the largest family known to us that is closed under all fundamental operations and is characterized in terms of a monadic second order (MSO) logic, besides of course enjoying decidability of the emptiness problem; in particular, it strictly includes the classes of regular languages, input-driven, alias Visibly Pushdown Languages [3], and other parenthesis-like languages [19]. These properties entitle them to support verification algorithms for many systems modeled either through OPGs or through their corresponding automata, Operator Precedence Automata (OPAs) [17].

Application of MSO logic, however, is in general considered of intractable complexity; thus, the literature exhibits a fairly wide variety of language subclasses that are characterized in terms of simpler logics such as fragments of first-order logics or temporal ones. For instance the equivalence between star-free regular languages and Linear

Temporal Logic (LTL) is proved in [15]; [2] characterizes classes of VPLs by means of various first-order and temporal logics. [16], instead, presents a logical characterization of the class of context-free languages by means of a first-order logic, although extended with a quantification over matchings.

In this paper we move a first step towards accomplishing a similar job with OPLs. We consider free grammars (FrGs) and languages (FrLs), which have been introduced with the main propose of supporting grammar inference [10,9] for programming languages. *Grammatical inference* (or *induction*) is an active and rich field of research, were various kinds of machine learning techniques are employed to infer a formal grammar or a variant of finite state machine from a set of observations, thus constructing a model which accounts for the characteristics of the observed objects. We refer the interested reader to the recent comprehensive works [14,11].

FrGs suffer from large size since their nonterminal alphabet is based on the power set of their terminal one; however they can be easily inferred on the basis of positive samples only, and can be minimized (by losing the property of being free) by applying classical algorithms [19,5]. In this paper we show that they are well suited to describe various language types, not only in the realm of programming languages. Furthermore, they can be used to define a sort of "superlanguage", possibly inferred in the limit from a set of strings of the user's desired language, and that can be further refined by imposing a few restricting properties in terms of first-order formulae.

The main result of this paper is that FrL strings satisfy formulae written in a first-order logic that restricts the MSO one defined for general OPLs; the structure over which such formulae are interpreted is the same as the one defined for general OPLs which required considerable generalization w.r.t. other previous results referring to simpler languages such as regular or input-driven ones [18].

In Section 2 we resume the basic definitions of OPGs and FrGs and languages and prove their basic properties. In Section 3 we provide a few simple examples of FrLs with the purpose of showing their usefulness in describing several types of languages, and we state some of their properties. In Section 4 we focus on their logic characterization. Finally, in Section 5 we envisage further steps in this ongoing research.

2 Preliminaries

A *context-free* (CF) grammar is a 4-tuple $G = (N, \Sigma, P, S)$, where N is the nonterminal alphabet, Σ is the terminal one, P the rule (or production) set, and $S \subseteq N$ the set of axioms[3]. The empty string is denoted ε.

The following naming convention will be adopted, unless otherwise specified: lowercase Latin letters $a, b, \ldots$ denote terminal characters; uppercase Latin letters $A, B, \ldots$ denote nonterminal characters; letters $u, v, \ldots$ denote terminal strings; and Greek letters $\alpha, \beta, \ldots$ denote strings over $\Sigma \cup N$. The strings may be empty, unless stated otherwise.

An *empty rule* has ε as the right hand side (r.h.s.). A *renaming rule* has one nonterminal as r.h.s. A grammar is *reduced* if every rule can be used to generate some string

[3] This less usual but equivalent definition of axioms as a set has been adopted for parenthesis languages [19] and other input-driven languages; we chose it for this paper to simplify some notations and constructions.

in Σ^*. It is *invertible* if no two rules have identical r.h.s. The *direct derivation* relation is denoted by $\Rightarrow$ and its reflexive transitive closure, the *derivation* relation, is denoted by $\overset{*}{\Rightarrow}$. If $\alpha \overset{*}{\Rightarrow} \beta$ in h steps, we write $\alpha \overset{h}{\Rightarrow} \beta$.

A rule is in *operator form* if its r.h.s. has no adjacent nonterminals; an *operator grammar* (OG) contains just such rules. Any CF grammar admits an equivalent OG.

Let G be an OG and α be a string over $(N \cup \Sigma)^*$: its *left and right terminal sets* are

$$\mathcal{L}(\alpha) = \begin{cases} \{a \in \Sigma \mid A \overset{*}{\Rightarrow} Ba\alpha\} & \text{if } \alpha = A \\ \{a\} & \text{if } \alpha = a\beta \\ \mathcal{L}(A) \cup \{a\} & \text{if } \alpha = Aa\beta \end{cases} \qquad \mathcal{R}(\alpha) = \begin{cases} \{a \in \Sigma \mid A \overset{*}{\Rightarrow} \alpha aB\} & \text{if } \alpha = A \\ \{a\} & \text{if } \alpha = \beta a \\ \mathcal{R}(A) \cup \{a\} & \text{if } \alpha = \beta aA \end{cases}$$

where $A \in N$, $B \in N \cup \{\varepsilon\}$, $a \in \Sigma$, $\beta \in (N \cup \Sigma)^*$. For an OG G, let α, β range over $(N \cup \Sigma)^*$ and $a, b \in \Sigma$. Three binary operator precedence (OP) relations are defined:

$$\begin{aligned} \text{equal in precedence: } & a \doteq b \iff \exists\, A \to \alpha aBb\beta, B \in N \cup \{\varepsilon\} \\ \text{takes precedence: } & a \gtrdot b \iff \exists\, A \to \alpha Db\beta, D \in N \text{ and } a \in \mathcal{R}(D) \\ \text{yields precedence: } & a \lessdot b \iff \exists\, A \to \alpha aD\beta, D \in N \text{ and } b \in \mathcal{L}(D) \end{aligned}$$

For an OG G, the *operator precedence matrix* (OPM) $M = OPM(G)$ is a $|\Sigma| \times |\Sigma|$ array that, for each ordered pair (a, b), stores the set M_{ab} of OP relations holding between a and b. If $M_{ab} = \{\circ\}$, with $\circ \in \{\lessdot, \doteq, \gtrdot\}$, we write $a \circ b$.

Definition 1 (Operator precedence grammar and language). *An OG G is an* operator precedence (OPG) *or* Floyd grammar *if, and only if, $M = OPM(G)$ is a conflict-free matrix, i.e., $\forall a, b$, $|M_{ab}| \leq 1$. An* operator precedence language (OPL) *is a language generated by an OPG.*

Definition 2 (Fischer Normal Form [12]). *An OPG is in* Fischer normal form (FNF) *iff it is invertible, has no empty rule except possibly $A \to \varepsilon$, where A is an axiom not used elsewhere, and no renaming rules.*

Previous literature [8,17] assumed that all precedence matrices of OPLs are $\doteq$-cycle free, i.e., they do not contain sequences of relations $a_1 \doteq a_2 \doteq \ldots \doteq a_1$. In the case of OPGs this prevents the risk of r.h.s. of unbounded length [9], but could be replaced by the weaker restriction of production's r.h.s. of bounded length, or could be removed at all by allowing such unbounded forms of grammars –e.g. with regular expressions as r.h.s. In our experience, such assumption helps to simplify notations and some technicalities of proofs; moreover we found that its impact in practical examples is minimal.[4] In this paper we accept a minimal loss of generation power and assume the simplifying assumption of $\doteq$-acyclicity.

Definition 3 (Free Grammar and Language). *Let G be an OPG with no renaming rules and no empty rule except possibly $C \to \varepsilon$, where C is an axiom not used elsewhere; G is a* free grammar *(FrG) iff the two following conditions hold*

[4] An example language that cannot be generated with an $\doteq$-acyclic OPM is the following: $\mathcal{L} = \{a^n(bc)^n \mid n \geq 0\} \cup \{b^n(ca)^n \mid n \geq 0\} \cup \{c^n(ab)^n \mid n \geq 0\}$

– *for every production* $A \to \alpha$, *with* $\alpha \neq \varepsilon$, $\mathcal{L}(A) = \mathcal{L}(\alpha)$ *and* $\mathcal{R}(A) = \mathcal{R}(\alpha)$*;*
– *for every nonterminals A, B,* $\mathcal{L}(A) = \mathcal{L}(B)$ *and* $\mathcal{R}(A) = \mathcal{R}(B)$ *implies* $A = B$*.*

A language generated by a FrG is a free language *(FrL).*

Notice that, by definition, a FrG is in FNF. Also, each nonterminal A is uniquely identified by the pair of sets $\mathcal{L}(A), \mathcal{R}(A)$; thus N is isomorphic to $\wp(\Sigma) \times \wp(\Sigma)$. Indeed, it is customary to use $\wp(\Sigma) \times \wp(\Sigma)$ as the nonterminal alphabet of a free grammar.

FrLs can also be defined in terms of a suitable automata family and extended to ω-languages in a similar way as it has been done for general OPLs [17,21].

Given an OPM M, the *maxgrammar* associated with M is the FrG that contains the productions that induce all and only the relations in M.

Notice that the maxgrammar associated with a complete OPM (i.e., an OPM with no empty case) generates the language Σ^*. The maxgrammar associated with an OPM is unique thanks to the hypothesis of $\doteq$-acyclicity or, in general, if we require that the length of the r.h.s. of the rules is a priori bounded. Also, the set of FrGs with a given OPM is a lattice whose top element is the maxgrammar associated with the matrix [9].

3 Examples and first properties of free languages

In this section we investigate the generative power of free grammars: the following examples, among others not reported here for brevity, show that they are well suited to formalize some typical programming language features and various types of system behavior; we will also show that the class of FrLs is not comparable with other subclasses of OPLs such as, e.g., VPLs.

Furthermore, the examples below show that FrGs are not intended to be built by hand; being driven by the powerset of Σ, both N and P may suffer from combinatorial explosion. However, according to their original motivation to support grammar inference, they are well suited to be easily built by some automatic device: in fact the grammars of the following examples have been automatically generated.[5]

Example 1. The FrG G depicted in Figure 1 with its OPM generates unparenthesized arithmetic expressions with the usual precedences of $\times$ w.r.t. $+$, which instead cannot be expressed as a VPL [8]. This grammar is obtained from the maxgrammar associated with the OPM by taking only those nonterminals that have letter n in both left and right sets. By this way we guarantee that all strings generated by the grammar begin and end with an n, and are thus well formed. All nonterminals of the grammar are axioms too.

Extending the above grammar to generate also parenthesized arithmetic expressions is a conceptually easy exercise since we only need new nonterminals, and corresponding rules, including $\langle\!|$ and $|\!\rangle$ in their left and right terminal sets, respectively. The corresponding FrG has 22 nonterminals and 168 rules, and it can be found among the examples available in the Flup package [1].

[5] The grammars presented here have been produced by the Flup tool (the whole package, which includes various utilities for the general class of OPLs, is available at [1]). In the future we plan to couple Flup with an additional tool that minimizes the original grammar by applying the classical procedure introduced in [19].

$$
\begin{aligned}
\langle\{n\},\{n\}\rangle &\to n\\
\langle\{+,\times,n\},\{+,n\}\rangle &\to \langle\{\times,n\},\{\times,n\}\rangle \ + \ \langle\{n\},\{n\}\rangle\\
\langle\{+,n\},\{+,n\}\rangle &\to \langle\{+,n\},\{+,\times,n\}\rangle \ + \ \langle\{n\},\{n\}\rangle\\
\langle\{+,n\},\{+,\times,n\}\rangle &\to \langle\{+,n\},\{+,n\}\rangle \ + \ \langle\{\times,n\},\{\times,n\}\rangle\\
\langle\{+,\times,n\},\{+,\times,n\}\rangle &\to \langle\{+,\times,n\},\{+,\times,n\}\rangle \ + \ \langle\{\times,n\},\{\times,n\}\rangle\\
\langle\{\times,n\},\{\times,n\}\rangle &\to \langle\{\times,n\},\{\times,n\}\rangle \ \times \ \langle\{n\},\{n\}\rangle\\
\langle\{+,n\},\{+,\times,n\}\rangle &\to \langle\{+,n\},\{+,\times,n\}\rangle \ + \ \langle\{\times,n\},\{\times,n\}\rangle\\
\langle\{+,\times,n\},\{+,n\}\rangle &\to \langle\{+,\times,n\},\{+,n\}\rangle \ + \ \langle\{n\},\{n\}\rangle\\
\langle\{+,\times,n\},\{+,\times,n\}\rangle &\to \langle\{+,\times,n\},\{+,n\}\rangle \ + \ \langle\{\times,n\},\{\times,n\}\rangle\\
\langle\{+,\times,n\},\{+,\times,n\}\rangle &\to \langle\{\times,n\},\{\times,n\}\rangle \ + \ \langle\{\times,n\},\{\times,n\}\rangle\\
\langle\{+,\times,n\},\{+,n\}\rangle &\to \langle\{+,\times,n\},\{+,\times,n\}\rangle \ + \ \langle\{n\},\{n\}\rangle\\
\langle\{+,n\},\{+,n\}\rangle &\to \langle\{n\},\{n\}\rangle \ + \ \langle\{n\},\{n\}\rangle\\
\langle\{+,n\},\{+,\times,n\}\rangle &\to \langle\{n\},\{n\}\rangle \ + \ \langle\{\times,n\},\{\times,n\}\rangle\\
\langle\{\times,n\},\{\times,n\}\rangle &\to \langle\{n\},\{n\}\rangle \ \times \ \langle\{n\},\{n\}\rangle\\
\langle\{+,n\},\{+,n\}\rangle &\to \langle\{+,n\},\{+,n\}\rangle \ + \ \langle\{n\},\{n\}\rangle
\end{aligned}
$$

	n	$+$	$\times$
n		$\gtrdot$	$\gtrdot$
$+$	$\lessdot$	$\gtrdot$	$\lessdot$
$\times$	$\lessdot$	$\gtrdot$	$\gtrdot$

Fig. 1. A FrG for unparenthesized arithmetic expressions and its OPM.

Example 2. Consider a simplified version of software system that serves requests of operations issued by various users but subject to possible asynchronous interrupts.

We model the behavior of the system by introducing an alphabet with a pair of symbols *call*, *ret*, to describe the request and completion of a user's operation, and symbol *int*, denoting the occurrence of the interrupt. Under normal behavior *call*s and *ret*s must be matched according to the normal LIFO policy; however, if an interrupt occurs when some *call*s are pending, they are reset without waiting for the corresponding *ret*s; possible subsequent *ret*s remain therefore unmatched. Unmatched returns can occur only if previously some interrupt flushed away all unmatched calls.

A FrG that generates sequences of operations and occurrences of interrupts consistent with the above informal description has the OPM displayed in Figure 2 and counts 21 nonterminals and 174 rules. It has been built starting from the maxgrammar associated with the OPM by taking as nonterminals only $\langle\{ret\},\{ret\}\rangle$ and those that do not contain *ret* in their left set. The axioms are all nonterminals $A \in (\wp(\Sigma) \times \wp(\Sigma)) \setminus \{\langle\{ret\},\{ret\}\rangle\}$. Nonterminal $\langle\{ret\},\{ret\}\rangle$ is necessary to generate sequences of unmatched returns; the constraint on the other nonterminals guarantees that a sequence of *ret*s is either matched by corresponding previous *call*s or is unmatched but preceded by an interrupt. This FrG too can be found in the examples in the Flup package.

The resulting grammar can be easily modified to deal with more complex policies, e.g., different levels of interrupt, but with a possible consequent size increase.

	call	*ret*	*int*
call	$\lessdot$	$\doteq$	$\gtrdot$
ret	$\gtrdot$	$\gtrdot$	$\gtrdot$
int	$\gtrdot$	$\lessdot$	$\lessdot$

Fig. 2. The OPM of Example 2.

All the FrGs in the above examples have been built by applying a top-down approach, starting from the maxgrammar associated with the OPM and "pruning" nonterminals and productions that would generate undesired strings. This approach complements the bottom up technique of traditional grammar inference, which builds a FrG generating a desired language by abstracting away from a given sample of language strings (it exploits the distinguishing property of FrGs that they can be inferred in the limit on the basis of a positive sample only [10]).

The typical canonical form of FrGs makes also easy the application of the classical minimization procedure that extends to structure grammars the minimization of finite state machines [19,5].

The above examples also help comparing the generative power of FrLs with other subclasses of OPLs.

Proposition 1. *The class of FrLs is incomparable with the classes of regular languages and VPLs.*

Proof. The language described in Example 2 is a FrL but is not regular, due to the necessity to match corresponding *call* and *ret* symbols, nor a VPL: although, in fact, it retains the rationale of VPLs in that it allows for unmatched "parenthesis-like" symbols (calls and returns in this case), it generalizes this VPLs feature in that such unmatched symbols can occur even inside a matching pair, which is impossible in VPLs. On the other hand, it is known that FrGs generate only non-counting languages [7], whereas regular languages and VPLs, which strictly contain regular ones, can be counting [20]. □

Proposition 2. *FrLs (with a fixed OPM) are closed w.r.t. intersection but not w.r.t. concatenation, complement, union and Kleene *.*

Proof. Closure under intersection, already stated in [7], follows from the fact that, given an OPM, the parsing of any string w is the same for any FrG (all FrG's nonterminal alphabets are pairs of subsets of Σ); it follows that $L(G_1) \cap L(G_2) = L(G_1 \cap G_2)$ where $G_1 \cap G_2$ denotes the grammar whose production set is the intersection of the production sets of G_1 and G_2 (possibly "cleaned up" of the useless productions) and is a FrG.

To prove that FrLs are not closed w.r.t. concatenation, consider language $L = \{a\}$ with $a \lessdot a$. L is a FrL but $L \cdot L$ is not: to generate $\# \lessdot a \lessdot a \gtrdot \#$ a FrG needs the productions $\langle\{a\},\{a\}\rangle \to a$ and $\langle\{a\},\{a\}\rangle \to a\langle\{a\},\{a\}\rangle$ which generate a^+. For the same reason $\neg L = \{a^n \mid n > 1 \vee n = 0\}$ is not a FrL; thus FrLs are not closed w.r.t. complement.

Consider the FrGs G_1 and G_2 below (both grammars have axiom $\langle\{a,b\},\{b\}\rangle$):

G_1 :	G_2 :
$\langle\{a,b\},\{b\}\rangle \to \langle\{a,b\},\{b\}\rangle\ b \mid \langle\{a\},\{a\}\rangle\ b$	$\langle\{a,b\},\{b\}\rangle \to \langle\{a\},\{a\}\rangle\ b$
$\langle\{a\},\{a\}\rangle \to a$	$\langle\{a\},\{a\}\rangle \to a \mid \langle\{a\},\{a\}\rangle\ a$

which generate, respectively, $L_1 = ab^+$ and $L_2 = a^+b$: all productions of G_1 and G_2 are necessary to generate all strings of $L_1 \cup L_2$ but the union of (productions of) G_1 and G_2 generates strings a^+b^+, which do not belong to $L_1 \cup L_2$.

Finally, consider the FrG G:

$$\begin{aligned}\langle\{a,b\},\{b\}\rangle &\to \langle\{a,b\},\{a\}\rangle\, b\\ \langle\{a,b\},\{a\}\rangle &\to \langle\{a,b\},\{b\}\rangle\, a \mid \langle\{b\},\{b\}\rangle\, a\\ \langle\{b\},\{b\}\rangle &\to b\end{aligned}$$

with axiom $\langle\{a,b\},\{b\}\rangle$, which generates $L = (ba)^+b$ (with $a \gtrdot b$, $b \gtrdot b$ and $b \gtrdot a$). To generate a string in L^* we need to generate two consecutive bs, corresponding respectively to the last and the first character of two consecutive words of L; this can be obtained only by means of a new rule for a nonterminal with right terminal set $\{b\}$, such as $\langle\{a,b\},\{b\}\rangle \to \langle\{a,b\},\{b\}\rangle b$ or the rule $\langle\{b\},\{b\}\rangle \to \langle\{b\},\{b\}\rangle b$, which however imply the generation also of strings containing any number of consecutive b, which do not belong to L^*. □

Ultimately, the above examples show that on the one hand FrGs can model the essential features of various systems but, on the other hand, they exhibit some unexpected limits in generative power which are not suffered even by regular languages. These limits must be ascribed to their distinguishing property of being inferrable in the limit by using only a set of positive strings (in fact the class of FrLs is not closed under complement). Thus they are better suited to define a sort of "skeleton language" to be refined by superimposing further constraints specified by means of some complementary formalism. A natural way to pursue such an approach is, e.g., to "intersect" them with some finite state machine. In this paper, instead, we will exploit the fact that FrLs can be defined in terms of first-order logic sentences, but first-order logic can also be used to define further, even more sophisticated, constraints on these languages.

4 First-order logic definability of free languages

The traditional MSO logic characterization of regular languages has been recently extended to larger classes such as VPLs [3] and OPLs [18]. In case of VPLs the syntax of MSO logic has been extended with a new binary predicate $\rightsquigarrow$, which is interpreted as a relation between positions of characters in the strings, such that $\boldsymbol{x} \rightsquigarrow \boldsymbol{y}$ when at positions $\boldsymbol{x}$ and $\boldsymbol{y}$ two matching parentheses occur (with a minor exception for unmatched open or closed parentheses for which, by convention, if they occur at position $\boldsymbol{x}$, then $\boldsymbol{x} \rightsquigarrow \infty$ or $\infty \rightsquigarrow \boldsymbol{x}$). In case of OPLs a more sophisticated relation has been necessary due to the fact that, as we will see, there is no one-to-one correspondence between open and closed parentheses (calls and returns in VPLs terminology). Then, due to the high complexity of MSO logic, various special cases of language families have been considered with the goal of characterizing them by means of simpler logics [2].

In this section we show that FrLs can be defined in terms of a FO logic rather than a MSO one. The converse property however does not hold: by Proposition 2, in fact, the class of FrLs is not closed under complement; hence, there are languages that can be defined in terms of FO logic but are not FrLs. On the other hand FO formulae can be used to refine FrLs by superimposing further properties.

The key difference between the traditional MSO language formulation and the new FO one is that in the MSO formulation each position in the string (over which the

MSO logic formula is interpreted) may be associated with several states of an automaton recognizing the language defined by the MSO formula, i.e., to several second-order variables denoting subsets of positions according to Büchi's approach; in our FO formulation instead, we associate positions with the left and right terminal sets of the nonterminal of a FrG that is the root of the subtree whose leftmost and rightmost leaves are in the given positions. Thanks to the fact that in FrGs the number of possible nonterminals is a priori bounded and they are univocally identified by their left and right terminal sets, we can express such association by means of first-order formulae, without the need to resort to second-order variables denoting sets of positions.

We first introduce some preliminary notation necessary to define the structure over which FO formulae are interpreted; then, we define the syntax of our FO logic and show how its formulae are interpreted; finally we prove that, for every FrG, a FO sentence can be automatically built that is satisfied by all and only the strings generated by the grammar.

Preliminarily, we introduce a special symbol # not in Σ to mark the beginning and the end of any string. The precedence relations in the OPM are implicitly extended to include #: the initial # can only yield precedence, and other symbols can only take precedence over the ending #.

Definition 4 (Operator precedence alphabet [17]). *An* operator precedence (OP) alphabet *is a pair* (Σ, M) *where* Σ *is an alphabet and* M *is a conflict-free operator precedence matrix, i.e. a* $|\Sigma \cup \{\#\}|^2$ *array that associates at most one of the operator precedence relations:* $\doteq$, $\lessdot$ *or* $\gtrdot$ *with each ordered pair* (a, b).

The operator precedence alphabet determines the "structure" of a string, as formalized by the following notion of chains.

Definition 5 (Chains [17]). *Let* (Σ, M) *be an operator precedence alphabet.*

- *A* simple chain *is a word* $a_0a_1a_2 \dots a_na_{n+1}$*, written as* ${}^{a_0}[a_1a_2 \dots a_n]^{a_{n+1}}$*, such that:* $a_0, a_{n+1} \in \Sigma \cup \{\#\}$*,* $a_i \in \Sigma$ *for every* $i : 1 \le i \le n$*,* $M_{a_0a_{n+1}} \neq \emptyset$*, and* $a_0 \lessdot a_1 \doteq a_2 \doteq \cdots \doteq a_{n-1} \doteq a_n \gtrdot a_{n+1}$.
- *A* composed chain *is a word* $a_0x_0a_1x_1a_2 \dots a_nx_na_{n+1}$*, written* ${}^{a_0}[x_0a_1x_1a_2 \dots a_nx_n]^{a_{n+1}}$*, with* $x_i \in \Sigma^*$*, and where* ${}^{a_0}[a_1a_2 \dots a_n]^{a_{n+1}}$ *is a simple chain, and either* $x_i = \varepsilon$ *or* ${}^{a_i}[x_i]^{a_{i+1}}$ *is a chain (simple or composed), for every* $i : 0 \le i \le n$.
- *The* body *of a chain* ${}^{a}[x]^{b}$*, simple or composed, is the word* x*. The* depth $d(x)$ *of the body* x *is defined recursively:* $d(x) = 1$ *if the chain is simple, whereas* $d(x_0a_1x_1 \dots a_nx_n) = 1 + \max_i d(x_i)$*. The depth of a chain is the depth of its body.*
- *A word* $w \in \Sigma^*$ *is* compatible *with* M *iff the two following conditions hold:*
  - *for each pair of letters* c, d*, consecutive in* w*,* $M_{cd} \neq \emptyset$
  - *for each factor (substring)* x *of* $\#w\#$ *such that* $x = a_0x_0a_1x_1a_2 \dots a_nx_na_{n+1}$ *and* ${}^{a_0}[x_0a_1x_1a_2 \dots a_nx_n]^{a_{n+1}}$ *is a chain (simple or composed), then* $M_{a_0a_{n+1}} \neq \emptyset$.

If an OPG contains the rule $A \to a_1a_2 \dots a_n$ and for some a_0, a_{n+1}, $a_0 \lessdot a_1$, $a_n \gtrdot a_{n+1}$, then ${}^{a_0}[a_1a_2 \dots a_n]^{a_{n+1}}$ is a simple chain. Similarly, if there is a rule $A \to B_0a_1B_1a_2 \dots a_nB_n$ and $B_i \overset{*}{\Rightarrow} x_i$ for every i, $a_0 \lessdot a_1$, $a_n \gtrdot a_{n+1}$ and ${}^{a_0}[x_0]^{a_1}$, ${}^{a_n}[x_n]^{a_{n+1}}$ are chains, then ${}^{a_0}[x_0a_1x_1a_2 \dots a_nx_n]^{a_{n+1}}$ is a composed chain.

Next, we introduce the syntax of our FO logic.

Definition 6 (First-order Logic over (Σ, M)). *Let (Σ, M) be an OP alphabet, and let $\mathcal{V}$ be a countable infinite set of first-order variables (denoted by $\boldsymbol{x}, \boldsymbol{y}, \ldots$). The $FO_{\Sigma,M}$ (*first-order logic *over (Σ, M)) is defined by the following syntax:*

$$\varphi := c(\boldsymbol{x}) \mid \boldsymbol{x} \leq \boldsymbol{y} \mid \boldsymbol{x} \curvearrowright \boldsymbol{y} \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists \boldsymbol{x}.\varphi$$

where $c \in \Sigma \cup \{\#\}$, $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{V}$.

A $\mathrm{FO}_{\Sigma,M}$ formula is interpreted over a string $w \in \Sigma^*$ compatible with M, with respect to assignments $\nu : \mathcal{V} \to \{0, 1, \ldots, |w| + 1\}$ in the following way.

- $w \models c(\boldsymbol{x})$ iff $\#w\# = w_1 c w_2$ and $|w_1| = \nu(\boldsymbol{x})$.
- $w \models \boldsymbol{x} \leq \boldsymbol{y}$ iff $\nu(\boldsymbol{x}) \leq \nu(\boldsymbol{y})$.
- $w \models \boldsymbol{x} \curvearrowright \boldsymbol{y}$ iff $\#w\# = w_1 a w_2 b w_3$, $|w_1| = \nu(\boldsymbol{x})$, $|w_1 a w_2| = \nu(\boldsymbol{y})$, and $a w_2 b$ is a chain ${}^{a}[w_2]^{b}$.
- $w \models \neg\varphi$ iff $w \not\models \varphi$.
- $w \models \varphi_1 \vee \varphi_2$ iff $w \models \varphi_1$ or $w \models \varphi_2$.
- $w \models \exists \boldsymbol{x}.\varphi$ iff $w' \models \varphi$, for some ν' with $\nu'(\boldsymbol{y}) = \nu(\boldsymbol{y})$ for all $\boldsymbol{y} \in \mathcal{V} \setminus \{\boldsymbol{x}\}$.

To improve readability, we use some standard abbreviations in formulae, such as $\boldsymbol{x} + 1, \boldsymbol{x} - 1, \boldsymbol{x} = \boldsymbol{y}, \boldsymbol{x} \neq \boldsymbol{y}, \boldsymbol{x} < \boldsymbol{y}$.

A *sentence* is a formula without free variables. The language of all strings $w \in \Sigma^*$ such that $w \models \varphi$ is denoted by $L(\varphi)$:

$$L(\varphi) = \{w \in \Sigma^* \mid w \models \varphi\}.$$

The distinguishing feature of $\mathrm{FO}_{\Sigma,M}$ w.r.t. the traditional FO logic is given by the introduction of predicate $\curvearrowright$: for each pair of positions $\boldsymbol{x}$ and $\boldsymbol{y}$ in a string, $\boldsymbol{x} \curvearrowright \boldsymbol{y}$ is used to denote that positions $\boldsymbol{x}$ and $\boldsymbol{y}$ "embrace" a chain. The relation formalized by this predicate resembles the $\rightsquigarrow$ defined for VPLs but exhibits two significant differences:

- it is not one-to-one, since a position $\boldsymbol{x}$ can in be in relation $\boldsymbol{x} \curvearrowright \boldsymbol{y}$ with more than one $\boldsymbol{y}$ and vice versa;
- is not defined on the positions where the leftmost and rightmost leaves of a subtree of the syntactic tree of the string occur (which are the positions of calls and returns in VPL terminology) but on the positions of the context of any subtree, i.e. of a chain.

Example 3. Consider the OP alphabet given in Figure 3. In all strings compatible with M, such that ${}^{\#}[w]^{\#}$ is a chain, all parentheses are well-matched.

The sentence in the same figure restricts the set of strings compatible with the OPM to the language where parentheses are used only when they are needed (i.e., to invert the natural precedence between $\times$ and $+$).

We now state our main result.

Theorem 1. *Let $G = \langle N, \Sigma, P, S \rangle$ be a FrG: then a $FO_{\Sigma,M}$ formula ψ can be effectively built such that $w \in L(G)$ iff $w \models \psi$.*

	$+$	$\times$	$(\!\|$	$\|\!)$	n	#
$+$	$\gtrdot$	$\lessdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$\times$	$\gtrdot$	$\gtrdot$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
$(\!\|$	$\lessdot$	$\lessdot$	$\lessdot$	$\doteq$	$\lessdot$	
$\|\!)$	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
n	$\gtrdot$	$\gtrdot$		$\gtrdot$		$\gtrdot$
#	$\lessdot$	$\lessdot$	$\lessdot$		$\lessdot$	$\doteq$

$$\forall \boldsymbol{x}\forall \boldsymbol{y}\left(\begin{array}{l}\boldsymbol{x} \curvearrowright \boldsymbol{y} \wedge \\ (\!|(\boldsymbol{x}+1) \wedge \\ |\!)(\boldsymbol{y}-1)\end{array} \Rightarrow \exists \boldsymbol{z}\left(\begin{array}{c}(\times(\boldsymbol{x}) \vee \times(\boldsymbol{y})) \\ \wedge \\ \boldsymbol{x}+1<\boldsymbol{z}<\boldsymbol{y}-1 \wedge +(\boldsymbol{z}) \wedge \\ \neg\exists \boldsymbol{u}\exists \boldsymbol{v}\left(\begin{array}{c}\boldsymbol{x}+1<\boldsymbol{u}<\boldsymbol{z} \wedge (\!|(\boldsymbol{u}) \wedge \\ \boldsymbol{z}<\boldsymbol{v}<\boldsymbol{y}-1 \wedge |\!)(\boldsymbol{v}) \wedge \\ \boldsymbol{u}-1 \curvearrowright \boldsymbol{v}+1\end{array}\right)\end{array}\right)\right)$$

Fig. 3. An OP alphabet (Σ, M) for arithmetic expressions (left), and a $\mathrm{FO}_{\Sigma,M}$ sentence (right).

Proof. We first introduce some shortcut notation to make formulae more compact and understandable.

When considering a chain ${}^{a}[w]^{b}$ we assume $w = w_0 a_1 w_1 \dots a_\ell w_\ell$, with ${}^{a}[a_1 a_2 \dots a_\ell]^{b}$ being a simple chain (any w_i may be empty). We denote by s_i the position of symbol a_i, for $i = 1, 2, \dots, \ell$ and set $a_0 = a$, $s_0 = 0$, $a_{\ell+1} = b$, and $s_{\ell+1} = |w| + 1$.

Notation TreeC is defined as follows ($n > 1$):

$$\mathrm{TreeC}(\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_n, \boldsymbol{x}_{n+1}) := \boldsymbol{x}_0 \curvearrowright \boldsymbol{x}_{n+1} \wedge \bigwedge_{0 \le i \le n}\left(\begin{array}{c}\boldsymbol{x}_i + 1 = \boldsymbol{x}_{i+1} \\ \vee \\ \boldsymbol{x}_i \curvearrowright \boldsymbol{x}_{i+1}\end{array} \wedge \bigwedge_{i+1<j\le n} \neg(\boldsymbol{x}_i \curvearrowright \boldsymbol{x}_j)\right)$$

If $\boldsymbol{x}_0 \curvearrowright \boldsymbol{x}_{n+1}$, there exist (unique) $\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_n, \boldsymbol{x}_{n+1}$ such that $\mathrm{TreeC}(\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_n, \boldsymbol{x}_{n+1})$ holds: in particular, $\boldsymbol{x}_0 \lessdot \boldsymbol{x}_1$, $\boldsymbol{x}_i \doteq \boldsymbol{x}_{i+1}$ for $1 \le i \le n-1$, and $\boldsymbol{x}_n \gtrdot \boldsymbol{x}_{n+1}$.

Let w be a chain body $w = w_0 a_1 w_1 a_2 \dots a_\ell w_\ell$: if every w_i is empty (the chain is simple), then $0 \curvearrowright \ell + 1$ and $\mathrm{TreeC}(0, 1, 2, \dots, \ell, \ell+1)$ holds; if w is the body of a composed chain, then $0 \curvearrowright |w|+1$ and $\mathrm{TreeC}(s_0, s_1, s_2, \dots, s_\ell, s_{\ell+1})$ holds (see Figure 4).

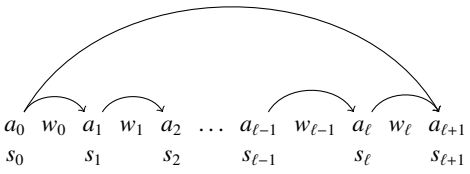

Fig. 4. Chain ${}^{a_0}[w_0 a_1 w_1 a_2 \dots a_\ell w_\ell]^{a_{\ell+1}}$, for which $\mathrm{TreeC}(s_0, s_1, s_2, \dots, s_\ell, s_{\ell+1})$ holds.

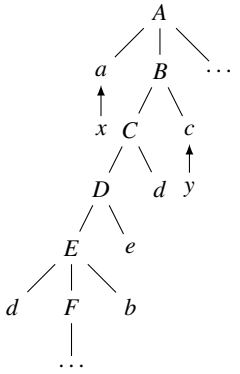

Fig. 5. Pair of positions $\boldsymbol{x}, \boldsymbol{y}$ for which $\mathcal{L}_{\{d,e\}}(\boldsymbol{x}, \boldsymbol{y})$ holds.

The similar notation Tree is instead defined as follows.

$$\mathrm{Tree}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{y}) := \boldsymbol{x} \curvearrowright \boldsymbol{y} \wedge \left(\begin{array}{c}(\boldsymbol{x}+1 = \boldsymbol{u} \ \vee \ \boldsymbol{x} \curvearrowright \boldsymbol{u}) \wedge \neg\exists \boldsymbol{t}(\boldsymbol{u} < \boldsymbol{t} < \boldsymbol{y} \wedge \boldsymbol{x} \curvearrowright \boldsymbol{t}) \\ \wedge \\ (\boldsymbol{v}+1 = \boldsymbol{y} \ \vee \ \boldsymbol{v} \curvearrowright \boldsymbol{y}) \wedge \neg\exists \boldsymbol{t}(\boldsymbol{x} < \boldsymbol{t} < \boldsymbol{v} \wedge \boldsymbol{t} \curvearrowright \boldsymbol{y})\end{array}\right)$$

This notation represents a "projection" of TreeC over positions $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, $\boldsymbol{x}_n$ and $\boldsymbol{x}_{n+1}$ (here corresponding to $\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{y}$), and is used when we do not need to refer to positions $\boldsymbol{x}_2, \ldots, \boldsymbol{x}_{n-1}$ within a chain.

Also, for every $A \subseteq \Sigma$, we define notations:

$$\mathcal{L}_A(\boldsymbol{x}, \boldsymbol{y}) := \left(\begin{array}{c} \forall \boldsymbol{u}, \boldsymbol{v}, z \left(\boldsymbol{u} \leq \boldsymbol{v} < z \leq \boldsymbol{y} \wedge \mathrm{Tree}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}, z) \Rightarrow \bigvee\limits_{a \in A} a(\boldsymbol{u}) \right) \\ \wedge \\ \bigwedge\limits_{a \in A} \exists\, \boldsymbol{u}, \boldsymbol{v}, z \, (\boldsymbol{u} \leq \boldsymbol{v} < z \leq \boldsymbol{y} \wedge \mathrm{Tree}(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}, z) \wedge a(\boldsymbol{u})) \end{array} \right)$$

$$\mathcal{R}_A(\boldsymbol{x}, \boldsymbol{y}) := \left(\begin{array}{c} \forall \boldsymbol{u}, \boldsymbol{v}, z \left(\boldsymbol{x} \leq \boldsymbol{u} < \boldsymbol{v} \leq z \wedge \mathrm{Tree}(\boldsymbol{u}, \boldsymbol{v}, z, \boldsymbol{y}) \Rightarrow \bigvee\limits_{a \in A} a(z) \right) \\ \wedge \\ \bigwedge\limits_{a \in A} \exists\, \boldsymbol{u}, \boldsymbol{v}, z \, (\boldsymbol{x} \leq \boldsymbol{u} < \boldsymbol{v} \leq z \wedge \mathrm{Tree}(\boldsymbol{u}, \boldsymbol{v}, z, \boldsymbol{y}) \wedge a(z)) \end{array} \right)$$

For instance, with reference to Figure 5, for positions $\boldsymbol{x}, \boldsymbol{y}$, $\mathcal{L}_{\{d,e\}}(\boldsymbol{x}, \boldsymbol{y})$ holds. Notice that for each pair of positions $\boldsymbol{x}, \boldsymbol{y}$ there exists a unique pair of sets A, B such that $\mathcal{L}_A(\boldsymbol{x}, \boldsymbol{y})$ and $\mathcal{R}_B(\boldsymbol{x}, \boldsymbol{y})$ hold.

Furthermore, for every $\langle L, R \rangle \in \Gamma$, we add notation $P_{\langle L,R \rangle}(\boldsymbol{x}, \boldsymbol{y})$, which represents the *terminal profile* of the chain, if any, between positions $\boldsymbol{x}$ and $\boldsymbol{y}$:

$$P_{\langle L,R \rangle}(\boldsymbol{x}, \boldsymbol{y}) := \boldsymbol{x} \rightsquigarrow \boldsymbol{y} \wedge \mathcal{L}_L(\boldsymbol{x}, \boldsymbol{y}) \wedge \mathcal{R}_R(\boldsymbol{x}, \boldsymbol{y})$$

Intuitively, $P_{\langle L,R \rangle}(\boldsymbol{x}, \boldsymbol{y})$ holds iff, in the syntax tree, the chain between positions $\boldsymbol{x}$ and $\boldsymbol{y}$ is the frontier of a subtree that has as root nonterminal $\langle L, R \rangle$.

Finally, for every $\langle L, R \rangle \in \Gamma$, set

$$\psi_{\langle L,R \rangle} := \forall \boldsymbol{x}, \boldsymbol{y} \left(\begin{array}{c} P_{\langle L,R \rangle}(\boldsymbol{x}, \boldsymbol{y}) \\ \Rightarrow \\ \bigvee\limits_{\langle L,R \rangle \to \langle L_0,R_0 \rangle c_1 \langle L_1,R_1 \rangle c_2 \ldots c_k \langle L_k,R_k \rangle} \exists \boldsymbol{x}_1 \ldots \boldsymbol{x}_k \left(\begin{array}{c} \mathrm{TreeC}(\boldsymbol{x}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k, \boldsymbol{y}) \wedge \\ \bigwedge\limits_{1 \leq i \leq k} c_i(\boldsymbol{x}_i) \wedge \\ \bigwedge\limits_{\substack{1 \leq i \leq k-1: \\ \langle L_i,R_i \rangle \neq \varepsilon}} P_{\langle L_i,R_i \rangle}(\boldsymbol{x}_i, \boldsymbol{x}_{i+1}) \wedge \\ \boldsymbol{x} + 1 \neq \boldsymbol{x}_1 \Rightarrow P_{\langle L_0,R_0 \rangle}(\boldsymbol{x}, \boldsymbol{x}_1) \wedge \\ \boldsymbol{x}_k + 1 \neq \boldsymbol{y} \Rightarrow P_{\langle L_k,R_k \rangle}(\boldsymbol{x}_k, \boldsymbol{y}) \end{array} \right) \end{array} \right)$$

where the disjunction is considered over the rules of G:

$$\rho = \langle L, R \rangle \to \langle L_0, R_0 \rangle c_1 \langle L_1, R_1 \rangle c_2 \ldots c_k \langle L_k, R_k \rangle,$$

with $\langle L_i, R_i \rangle \in N \cup \{\varepsilon\}$, $0 \leq i \leq k$, and $L = L_0 \cup \{c_1\}$, $R = R_k \cup \{c_k\}$.

To complete the construction and the proof of Theorem 1 we define:

$$\psi := \bigwedge_{\langle A,B \rangle} \psi_{\langle A,B \rangle} \;\wedge\; \exists \boldsymbol{e} \left(\#(\boldsymbol{e} + 1) \wedge \neg \exists \boldsymbol{y} (\boldsymbol{e} + 1 < \boldsymbol{y}) \wedge \bigvee_{\langle L,R \rangle \in S} P_{\langle L,R \rangle}(0, \boldsymbol{e} + 1) \right)$$

The proof of the theorem is a direct consequence of the following Lemma 1 when $\langle L, R\rangle$ is an axiom of G. □

Lemma 1. *For every $\langle L, R\rangle \in N$ and for every body w of a chain, we have $\langle L, R\rangle \overset{*}{\Rightarrow} w$ iff $w \models P_{\langle L,R\rangle}(0, |w|+1) \wedge \bigwedge_{\langle A,B\rangle} \psi_{\langle A,B\rangle}$.*

Proof. Consider first the direction from left to right of the lemma. The proof is by induction on the length h of a derivation.

If $h = 1$, then $\langle L, R\rangle \overset{*}{\Rightarrow} w$ implies that $\rho = \langle L, R\rangle \to a_1 a_2 \dots a_l$ is a production of G, and $w = a_1 a_2 \dots a_l$ is the body of a simple chain. G being a FrG, it is $L = \{a_1\}$ and $R = \{a_l\}$. Since $0 \curvearrowright l+1$ and $w \models \mathcal{L}_{\{a_1\}}(0, l+1) \wedge \mathcal{R}_{\{a_l\}}(0, l+1)$, then $w \models P_{\langle L,R\rangle}(0, l+1)$.

For every $\langle A, B\rangle \in \Gamma$ and positions $\boldsymbol{x}, \boldsymbol{y}$, $w \models P_{\langle A,B\rangle}(\boldsymbol{x}, \boldsymbol{y})$ holds true only if $\langle A, B\rangle = \langle L, R\rangle$ and $\boldsymbol{x} = 0, \boldsymbol{y} = l+1$. Furthermore, there exist (unique) $\boldsymbol{x}_1 = 1, \boldsymbol{x}_2 = 2, \dots, \boldsymbol{x}_l = l$ such that TreeC$(0, 1, \dots, l, l+1)$ holds, and for every $j = 1, \dots, l$, $a_j(\boldsymbol{x}_j)$ holds true. Thus, $w \models \psi_{\langle A,B\rangle}$ for every $\langle A, B\rangle \in \Gamma$, and $w \models P_{\langle L,R\rangle}(0, |w|+1) \wedge \bigwedge_{\langle A,B\rangle} \psi_{\langle A,B\rangle}$.

Assume that this direction of the lemma holds for every derivation of length $\leq h$. Let $\langle L, R\rangle \overset{h+1}{\Rightarrow} w$, with $\langle L, R\rangle \Rightarrow \langle L_0, R_0\rangle a_1 \langle L_1, R_1\rangle a_2 \dots a_l \langle L_l, R_l\rangle$ and, for each $i = 0, 1, \dots, l$, $\langle L_i, R_i\rangle \overset{h_i}{\Rightarrow} w_i$ such that $h_i \leq h$ and $w = w_0 a_1 w_1 \dots a_l w_l$ is the body of a composed chain ($w_i = \varepsilon$ if $\langle L_i, R_i\rangle = \varepsilon$).

By the inductive hypothesis, for every $i = 0, 1 \dots, l$ such that $w_i \neq \varepsilon$, we have $w_i \models P_{\langle L_i,R_i\rangle}(0, |w_i|+1) \wedge \bigwedge_{\langle A,B\rangle} \psi_{\langle A,B\rangle}$. Let $\rho = \langle L, R\rangle \to \langle L_0, R_0\rangle a_1 \langle L_1, R_1\rangle a_2 \dots a_l \langle L_l, R_l\rangle$: G being a FrG, we have $L = L_0 \cup \{a_1\}$ and $R = R_k \cup \{a_l\}$; thus $w \models \mathcal{L}_L(0, |w|+1) \wedge \mathcal{R}_R(0, |w|+1)$, and $w \models P_{\langle L,R\rangle}(0, |w|+1)$. Furthermore, let $\boldsymbol{x}, \boldsymbol{y}$ be positions such that $w \models P_{\langle A,B\rangle}(\boldsymbol{x}, \boldsymbol{y})$ for some $\langle A, B\rangle \in \Gamma$ and $\boldsymbol{x}, \boldsymbol{y}$ are not both inside the same w_i, and they are not s_i and s_{i+1}; then necessarily $\boldsymbol{x} = 0, \boldsymbol{y} = |w|+1$, and $w \models P_{\langle A,B\rangle}(0, |w|+1)$ only if $\langle A, B\rangle = \langle L, R\rangle$. Also, there exist $\boldsymbol{x}_0 = 0, \boldsymbol{x}_1 = s_1, \dots, \boldsymbol{x}_l = s_l, \boldsymbol{x}_{l+1} = |w|+1$ such that TreeC$(\boldsymbol{x}_0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_l, \boldsymbol{x}_{l+1})$ holds, and for every $j = 1, \dots, l$, $a_j(\boldsymbol{x}_j)$ holds true. Hence, $w \models \bigwedge_{\langle A,B\rangle} \psi_{\langle A,B\rangle}$.

Consider then the direction from right to left of the lemma. The proof is by induction on the depth d of the chain.

If $d = 1$, then $w = a_1 a_2 \dots a_l$ is the body of a simple chain. Since $w \models P_{\langle L,R\rangle}(0, |w|+1)$, then there exist $\rho = \langle L, R\rangle \to \langle L_0, R_0\rangle c_1 \langle L_1, R_1\rangle c_2 \dots c_k \langle L_k, R_k\rangle$ and $\boldsymbol{x}_1, \dots, \boldsymbol{x}_k$ such that TreeC$(0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_k, |w|+1)$ and $c_j(\boldsymbol{x}_j)$ for every $j = 1, \dots, k$ hold. By definition of TreeC, we have $\boldsymbol{x}_j = j$ for every $j = 1, \dots, k$ and $k = l$, and $a_j = c_j$ for every j. There is, thus, a production of G: $\rho = \langle L, R\rangle \to a_1 a_2 \dots a_l$, and $\langle L, R\rangle \overset{*}{\Rightarrow} w$ holds.

Let now $d > 1$, then $w = w_0 a_1 w_1 \dots a_l w_l$ is the body of a composed chain and s_j ($1 \leq j \leq l$) are the unique positions such that TreeC$(0, s_1, \dots, s_l, |w|+1)$ holds true. Since $w \models P_{\langle L,R\rangle}(0, |w|+1) \wedge \bigwedge_{\langle A,B\rangle} \psi_{\langle A,B\rangle}$, then there exists a production ρ of G such that $\rho = \langle L, R\rangle \to \langle L_0, R_0\rangle c_1 \langle L_1, R_1\rangle c_2 \dots c_k \langle L_k, R_k\rangle$ and there exist $\boldsymbol{x}_j$ ($1 \leq j \leq k$) with TreeC$(0, \boldsymbol{x}_1, \dots, \boldsymbol{x}_l, |w|+1)$ and $c_j(\boldsymbol{x}_j)$; thus we have $k = l$ and $c_j = a_j$ for each j. Furthermore, let $\boldsymbol{x}_0 = 0, \boldsymbol{x}_{l+1} = |w|+1$: for every $i = 0, 1 \dots, k$ such that $\langle L_i, R_i\rangle \neq \varepsilon$, $w \models P_{\langle L_i,R_i\rangle}(\boldsymbol{x}_i, \boldsymbol{x}_{i+1})$ holds true, and we have $w_i \models P_{\langle L_i,R_i\rangle}(0, |w_i|+1) \wedge \bigwedge_{\langle A,B\rangle} \psi_{\langle A,B\rangle}$. By inductive hypothesis, thus there exists in G a derivation $\langle L_i, R_i\rangle \overset{*}{\Rightarrow} w_i$. Hence, $\langle L, R\rangle \Rightarrow \langle L_0, R_0\rangle a_1 \langle L_1, R_1\rangle a_2 \dots \langle L_{k-1}, R_{k-1}\rangle a_k \langle L_k, R_k\rangle \overset{*}{\Rightarrow} w$. □

5 Conclusions

After having developed a fairly complete theory of the old OPLs, which now includes automata and MSO logic characterization, closure and decidability properties, extensions to the case of ω-languages, with this paper we initiated a new research path aimed at finding suitable subfamilies of OPLs and simpler logics that could enable applications more practical than those based on MSO logic.

In this first step we showed that from any FrG a first-order formula can be automatically derived so that the words generated by the grammars are exactly those that satisfy the formula. FrLs suffer from some generative power limits due to the simplicity of their grammars; however, the same logic that characterizes them can also be applied to refine them by stating additional desired properties.

Several further steps are scheduled for this research within the general theme of finding formalisms that are general enough to define rather sophisticated languages but also allow for relatively "efficient" algorithms for applications. We also plan to further investigate (variants of) our logic; interestingly enough FrLs are non-counting (context-free) languages [6] and previous literature devoted considerable attention to algebraically and logically characterize non-counting or star-free regular languages [20].

References

1. Flup. https://github.com/bzoto/flup
2. Alur, R., Arenas, M., Barceló, P., Etessami, K., Immerman, N., Libkin, L.: First-order and temporal logics for nested words. Logical Methods in Computer Science 4(4) (2008)
3. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journ. ACM 56(3) (2009)
4. Barenghi, A., Crespi Reghizzi, S., Mandrioli, D., Panella, F., Pradella, M.: The PAPAGENO parallel-parser generator. In: Cohen, A. (ed.) Compiler Construction - 23rd International Conference, CC 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8409, pp. 192–196. Springer (2014)
5. Brainerd, W.S.: The minimization of tree automata. Information and Control 13(5) (1968)
6. Crespi Reghizzi, S., Guida, G., Mandrioli, D.: Noncounting Context-Free Languages. Journ. ACM 25, 571–580 (1978)
7. Crespi Reghizzi, S., Mandrioli, D.: A Class of Grammars Generating Non-Counting Languages. Inf. Process. Lett. 7(1), 24–26 (1978)
8. Crespi Reghizzi, S., Mandrioli, D.: Operator Precedence and the Visibly Pushdown Property. Journal of Computer and System Science 78(6), 1837–1867 (2012)
9. Crespi Reghizzi, S., Mandrioli, D., Martin, D.F.: Algebraic Properties of Operator Precedence Languages. Information and Control 37(2), 115–133 (May 1978)
10. Crespi Reghizzi, S., Melkanoff, M.A., Lichten, L.: The Use of Grammatical Inference for Designing Programming Languages. Commununications of the ACM 16(2), 83–90 (1973)
11. D'Ulizia, A., Ferri, F., Grifoni, P.: A survey of grammatical inference methods for natural language learning. Artif. Intell. Rev. 36(1), 1–27 (2011), http://dx.doi.org/10.1007/s10462-010-9199-1
12. Fischer, M.J.: Some properties of precedence languages. In: STOC '69. pp. 181–190 (1969)
13. Grune, D., Jacobs, C.J.: Parsing techniques: a practical guide. Springer, New York (2008)
14. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, New York, NY, USA (2010)

15. Kamp, H.: Tense Logic and the Theory of Linear Order. Ph.D. thesis, University of California, Los Angeles (California) (1968)
16. Lautemann, C., Schwentick, T., Thérien, D.: Logics for context-free languages. In: Selected Papers from the 8th International Workshop on Computer Science Logic. pp. 205–216. CSL '94, Springer-Verlag, London, UK, UK (1995)
17. Lonati, V., Mandrioli, D., Pradella, M.: Precedence Automata and Languages. In: 6th Int. Computer Science Symposium in Russia (CSR), LNCS, vol. 6651, pp. 291–304 (2011)
18. Lonati, V., Mandrioli, D., Pradella, M.: Logic Characterization of Invisibly Structured Languages: the Case of Floyd Languages. In: SOFSEM, LNCS, vol. 7741. Springer (2013)
19. McNaughton, R.: Parenthesis Grammars. Journ. ACM 14(3), 490–500 (1967)
20. McNaughton, R., Papert, S.: Counter-free Automata. MIT Press, Cambridge, USA (1971)
21. Panella, F., Pradella, M., Lonati, V., Mandrioli, D.: Operator precedence ω-languages. In: DLT, LNCS, vol. 7907, pp. 396–408 (2013)