Computer Science Review 27 (2018) 61–87

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Review article

Generalizing input-driven languages: Theoretical and practical benefits

Dino Mandrioli [a], Matteo Pradella [a,b,*]

[a] DEIB, Politecnico di Milano, via Ponzio 34/5, 20133 Milano, Italy
[b] IEIIT, Consiglio Nazionale delle Ricerche, via Ponzio 34/5, 20133 Milano, Italy

ARTICLE INFO

Article history:
Received 16 March 2017
Received in revised form 25 September 2017
Accepted 4 December 2017

Keywords:
Regular languages
Context-free languages
Input-driven languages
Visibly pushdown languages
Operator-precedence languages
Monadic second order logic
Closure properties
Decidability and automatic verification

ABSTRACT

Regular languages (RL) are the simplest family in Chomsky's hierarchy. Thanks to their simplicity they enjoy various nice algebraic and logic properties that have been successfully exploited in many application fields. Practically all of their related problems are decidable, so that they support automatic verification algorithms. Also, they can be recognized in real-time.

Context-free languages (CFL) are another major family well-suited to formalize programming, natural, and many other classes of languages; their increased generative power w.r.t. RL, however, causes the loss of several closure properties and of the decidability of important problems; furthermore they need complex parsing algorithms. Thus, various subclasses thereof have been defined with different goals, spanning from efficient, deterministic parsing to closure properties, logic characterization and automatic verification techniques.

Among CFL subclasses, so-called *structured* ones, i.e., those where the typical tree-structure is visible in the sentences, exhibit many of the algebraic and logic properties of RL, whereas deterministic CFL have been thoroughly exploited in compiler construction and other application fields.

After surveying and comparing the main properties of those various language families, we go back to *operator precedence languages* (OPL), an old family through which R. Floyd pioneered deterministic parsing, and we show that they offer unexpected properties in two fields so far investigated in totally independent ways: they enable parsing parallelization in a more effective way than traditional sequential parsers, and exhibit the same algebraic and logic properties so far obtained only for less expressive language families.

Contents

* Corresponding author at: DEIB, Politecnico di Milano, via Ponzio 34/5, 20133 Milano, Italy.
E-mail addresses: dino.mandrioli@polimi.it (D. Mandrioli), matteo.pradella@polimi.it (M. Pradella).

https://doi.org/10.1016/j.cosrev.2017.12.001
1574-0137/

1. Introduction

Regular (RL) and context-free languages (CFL) are by far the most widely studied families of formal languages in the rich literature of the field. In Chomsky's hierarchy, they are, respectively, in positions 2 and 3, 0 and 1 being recursively enumerable and context-sensitive languages.

Thanks to their simplicity, RL enjoy practically all positive properties that have been defined and studied for formal language families: they are closed under most algebraic operations, and most of their properties of interest (emptiness, finiteness, containment) are decidable. Thus, they found fundamental applications in many fields of computer and system science: HW circuit design and minimization, specification and design languages (equipped with powerful supporting tools), automatic verification of SW properties, etc. One of their most relevant applications is now model-checking which exploits the decidability of the containment problem and important characterizations in terms of mathematical logics [1,2].

On the other hand, the typical linear structure of RL sentences makes them unsuitable or only partially suitable for application in fields where the data structure is more complex, e.g., is tree-like. For instance, in the field of compilation they are well-suited to drive lexical analysis but not to manage the typical nesting of programming and natural language features. The classical language family adopted for this type of modeling and analysis is the context-free one. The increased expressive power of CFL allows to formalize many syntactic aspects of programming, natural, and various other categories of languages. Suitable algorithms have been developed on their basis to parse their sentences, i.e., to build the structure of sentences as syntax-trees.

General CFL, however, lose various of the nice mathematical properties of RL: they are closed only under some of the algebraic operations, and several decision problems, typically the inclusion problem, are undecidable; thus, the automatic analysis and synthesis techniques enabled for RL are hardly generalized to CFL. Furthermore, parsing CFL may become considerably less efficient than recognizing RL: the present most efficient parsing algorithms of practical use for general CFL have an $O(n^3)$ time complexity.

The fundamental subclass of deterministic CFL (DCFL) has been introduced, and applied to the formalization of programming language syntax, to exploit the fact that in this case parsing is in $O(n)$. DCFL, however, do not enjoy enough algebraic and logic properties to extend to this class the successful applications developed for RL: e.g., although their equivalence is decidable, containment is not; they are closed under complement but not under union, intersection, concatenation and Kleene *.

From this point of view, *structured CFL* are somewhat in between RL and general CFL. Intuitively, by structured CFL we mean languages where the structure of the syntax-tree associated with a given sentence is immediately apparent in the sentence. *Parenthesis languages (PL)* introduced in a pioneering paper by McNaughton [3] are the first historical example of such languages. McNaughton showed that they enjoy closure under Boolean operations (which, together with the decidability of the emptiness problem, implies decidability of the containment problem) and their generating grammars can be minimized in a similar way as *finite state automata* (FSA) are minimized (in fact an equivalent formalism for parenthesis languages are *tree automata* [4,5]).

Starting from PL various extensions of this family have been proposed in the literature, with the main goal of preserving most of the nice properties of RL and PL, yet increasing their generative power; among them *input-driven languages (IDL)* [6,7], later renamed *visibly pushdown languages (VPL)* [8] have been quite successful: the attribute *Input-driven* is explained by the property that their recognizing pushdown automata can decide whether to apply a push operation, or a pop one to their pushdown store or leaving it unaffected exclusively on the basis of the current input symbol; the attribute *visible*, instead, refers to the fact that their tree-like structure is immediately visible in their sentences.

IDL, alias VPL, are closed under all traditional language operations (and therefore enjoy the consequent decidability properties). Also, they are characterized in terms of a *monadic second order* (MSO) logic by means of a natural extension of the classic characterization for RL originally and independently developed by Büchi, Elgot, and Trakhtenbrot [9–11]. For these reasons they are a natural candidate for extending model checking techniques from RL. To achieve such a goal in practice, however, MSO logic is not yet tractable due to the complexity of its decidability problems; thus, some research is going on to "pair" IDL with specification languages inspired by temporal logic as it has been done for RL [12].

Structured languages do not need a real *parsing*, since the syntax-tree associated with their sentences is already "embedded" therein; thus, their recognizing automata only have to decide whether an input string is accepted or not, whereas full parsers for general CFL must build the structure(s) associated with any input string which naturally supports its semantics (think, e.g., to the parsing of unparenthesized arithmetic expressions where the traditional precedence of multiplicative operators over the additive ones is "hidden" in the syntax of the language.) This property, however, severely restricts their application field as the above example of arithmetic expressions immediately shows.

Rather recently, we resumed the study of an old class of languages which was interrupted a long time ago, namely *operator precedence languages (OPL)*. OPL and their generating grammars (OPG) have been introduced by Floyd [13] to build efficient deterministic parsers; indeed they generate a large and meaningful subclass of DCFL. We can intuitively describe OPL as "*input driven but not visible*": they can be claimed as *input-driven* since the parsing actions on their words – whether to push or pop – depend exclusively on the input alphabet and on the relation defined thereon, but their structure is *not visible* in their words: e.g., they can include unparenthesized expressions.

In the past their algebraic properties, typically closure under Boolean operations [14], have been investigated with the main goal of designing inference algorithms for their languages [15]. After that, their theoretical investigation has been abandoned because of the advent of more powerful grammars, mainly LR ones [16,17], that generate all DCFL (although some deterministic parsers based on OPL's simple syntax have been continuously implemented at least for suitable subsets of programming languages [18]).

The renewed interest in OPG and OPL has been ignited by two seemingly unrelated remarks: on the one hand we realized