Theoretical Computer Science ••• (••••) •••–•••

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Toward a theory of input-driven locally parsable languages[☆,☆☆]

Stefano Crespi Reghizzi[a,c], Violetta Lonati[b], Dino Mandrioli[a], Matteo Pradella[a,c,*]

[a] DEIB, Politecnico di Milano, P.zza L. da Vinci, 32, 20133 Milano, Italy
[b] DI, Università degli Studi di Milano, via Comelico 39/41, Milano, Italy
[c] IEIIT, CNR, Milano, Italy

A R T I C L E I N F O

Article history:
Received 16 October 2015
Received in revised form 6 April 2016
Accepted 2 May 2016
Available online xxxx

Keywords:
Operator precedence languages
Input-driven languages
Visibly pushdown languages
Parallel parsing

A B S T R A C T

If a context-free language enjoys the local parsability property then, no matter how the source string is segmented, each segment can be parsed independently, and an efficient parallel parsing algorithm becomes possible. The new class of locally chain parsable languages (LCPLs), included in the deterministic context-free language family, is here defined by means of the chain-driven automaton and characterized by decidable properties of grammar derivations. Such automaton decides whether to reduce or not a substring in a way purely driven by the terminal characters, thus extending the well-known concept of input-driven (ID) alias visibly pushdown machines. The LCPL family extends and improves the practically relevant Floyd's operator-precedence (OP) languages which are known to strictly include the ID languages, and for which a parallel-parser generator exists.

1. Introduction

Syntax analysis or parsing of context-free languages (CFLs) is a mature research area, and good parsing algorithms are available for the whole CFL family and for the deterministic (DCFL) subfamily that is of concern here. Yet the classical parsers are strictly serial and cannot profit from the parallelism of current computers. An exception is the parallel deterministic parser [4,3] based on Floyd's [13] operator-precedence grammars (OPGs) and their languages (OPLs), which are included in the DCFL family. This is a data-parallel algorithm that is based on a theoretical property of OPGs, called *local parsability*: any arbitrary substring of a sentence can be deterministically parsed, returning the unique partial syntax-tree whose frontier matches the input string.

LL(k) and LR(k) grammars do not have this property, and their parsers must scan the input left-to-right to build leftmost derivations (or reversed-rightmost ones). On the contrary, the abstract recognizer of a locally parsable language, called a *local parser*, repeatedly looks in some arbitrary position inside the input string for a rule right-hand side (r.h.s.) and reduces it. The local parsability property ensures the correctness of the syntax tree thus obtained, no matter where and in which order reductions are applied.

☆ Work partially supported by PRIN 2010LYA9RH-006.
☆☆ Preliminary versions of the results of this paper appeared in [8], and were presented at ICTCS'15.
* Corresponding author.
E-mail addresses: stefano.crespireghizzi@polimi.it (S. Crespi Reghizzi), lonati@di.unimi.it (V. Lonati), dino.mandrioli@polimi.it (D. Mandrioli), matteo.pradella@polimi.it (M. Pradella).

http://dx.doi.org/10.1016/j.tcs.2016.05.003
0304-3975/© 2016 Elsevier B.V. All rights reserved.

The informal idea of local parsability is occasionally mentioned in old research on parallel parsing, and has been formalized for OPGs in [4]. Our contribution is the definition of a new and more general class of locally parsable languages: the family of languages to be called *Locally Chain Parsable* (LCPLs), which gains in generative capacity and bypasses some inconveniences of OPGs. We remark that OPLs in turn are a generalization of the well-known family of input-driven (alias visibly pushdown) languages (IDLs) [23,2,9], which are characterized by pushdown machines that choose to perform a push/pop/stay operation depending on the alphabetic class (opening/closing/internal) of the current input character, without a need to check the top of stack symbol.

To understand in what sense our LCPLs are input-driven, we first recall that IDLs generalize parenthesis languages, by taking the opening/closing characters as parentheses to be balanced, while the internal characters are handled by a finite-state automaton. It suffices a little thought to see that IDLs have the local parsability property, which also stems from the fact that IDLs are included in the OPL family. Yet, the rigid alphabetic 3-partition severely reduces their generative capacity. If we allow the parser decision whether to push, pop, or stay, to be based on a *pair* of adjacent terminal characters (more precisely on the precedence relation $\lessdot, \gtrdot, \doteq$ between them), instead of just one as in the IDLs, we obtain the OPL family, which has essentially the same closure and decidability properties [9,18]. Loosely speaking, we may say that the input that drives the automaton for OPLs is a terminal string of length two.

With the LCPL definition, we move further: the automaton bases its decision whether to reduce or not a substring (which may contain nonterminals) on the purely terminal string orderly containing: the preceding terminal, the terminals of the substring, and the following terminal. Such triplet will be called a *chain* and the machine a *chain-driven automaton* (CDA).

The main results of this paper are presented along the following organization. After the Preliminaries, Section 3 introduces the chain-driven machine as a recognizer for all context-free languages. Section 4 defines local chain parsability for chain-driven automata and for grammars, and proves the two notions to be equivalent. Section 5 extends the definition of chains from embracing a single r.h.s. to representing portions of a whole derivation, and formulates a decidability condition for local chain parsability based on the absence of conflicts between chain sets. Section 6 proves structural properties of LCPLs, the strict inclusion thereof in the DCFL family, and investigates the behavior of the class with respect to classical language operations; precisely, it shows that, under suitable hypotheses of structural compatibility, the application of Boolean operations, but in general not concatenation and Kleene *, to two LCPLs produces a new LCPL; as a corollary, the inclusion problem between structurally compatible LCPLs is decidable, a key property for possible application of model checking techniques. Section 7 establishes the strict inclusion of the OPL (and hence also IDL) family within LCPLs, and claims through practical examples that LCPGs are more suitable than OPG for specifying real programming languages. Section 8 compares our new family of languages with similar families introduced in previous literature. Finally, Section 9 draws some conclusions and outlines several goals for future research.

2. Preliminaries

For terms not defined here, we refer to any textbook on formal languages, e.g. [16]. The *terminal* alphabet is denoted by Σ; it includes the letter # used as start and end of text. Let Δ be an alphabet disjoint from Σ. A string $\beta \in (\Sigma \cup \Delta)^* \, \Sigma \, (\Sigma \cup \Delta)^* \setminus (\Sigma \cup \Delta)^* \, \Delta\Delta \, (\Sigma \cup \Delta)^*$ is in *operator form*; in words, β contains at least one terminal and does not contain adjacent symbols from Δ. OF(Δ) denotes the set of all operator form strings over $\Sigma \cup \Delta$.

The following *naming conventions* are adopted for letters and strings, unless otherwise specified: lowercase Latin letters $a, b, \ldots$ denote terminal characters; uppercase Latin letters $A, B, \ldots$ denote characters in Δ; lowercase Latin letters $x, y, z \ldots$ denote terminal strings; and Greek lowercase letters $\alpha, \ldots, \omega$ denote strings over $\Sigma \cup \Delta$.

Within the preceding convention, symbols in ***bold*** denote strings over an alphabet that includes, as extra symbols, the square brackets, e.g. $\boldsymbol{x} \in (\Sigma \cup \{[,]\})^*$, $\boldsymbol{\alpha} \in (\Sigma \cup \Delta \cup \{[,]\})^*$.

We introduce the following short notation for frequently used operations based on alphabetic projections:

- for erasing all nonterminal symbols in a string $\boldsymbol{\alpha}$, we write $\widehat{\boldsymbol{\alpha}}$;
- for erasing all square brackets, we write $\widetilde{\boldsymbol{\alpha}}$;
- moreover, $\boldsymbol{\alpha} \mathrel{\hat{=}} \boldsymbol{\beta}$ stands for $\widehat{\boldsymbol{\alpha}} = \widehat{\boldsymbol{\beta}}$ and $\boldsymbol{\alpha} \mathrel{\tilde{=}} \boldsymbol{\beta}$ stands for $\widetilde{\boldsymbol{\alpha}} = \widetilde{\boldsymbol{\beta}}$.

A *context-free grammar* is a 4-tuple $G = (V_N, \Sigma, P, S)$, where V_N is the nonterminal alphabet, P the set of rules, and $S \subseteq V_N$ is the set of axioms. V denotes the set $V_N \cup \Sigma$. For a rule $A \to \alpha \in P$, $A \in V_N$ is the left-hand side (l.h.s.) and $\alpha \in V^*$ is the right-hand side (r.h.s.).

Let H be a new symbol, $H \notin V$, and $\sigma : V \to \{H\}$ be the homomorphism that maps every nonterminal to H: for every $X \in V_N, \sigma(X) = H$, otherwise $\sigma(a) = a$. The *stencil* of a rule $A \to \alpha$ is the rule $H \to \sigma(\alpha)$.

The *derivation relation* for a grammar G is denoted as usual by $\Rightarrow_G$ and its reflexive and transitive closure by $\overset{*}{\Rightarrow}_G$. A *sentential form* generated by G is any string $\#\alpha\# \in V^*$ such that $T \overset{*}{\Rightarrow}_G \alpha$ with $T \in S$, and the language generated by G is the set $L(G)$ of strings $x \in \Sigma^*$ such that $\#x\#$ is a sentential form.

A grammar is *invertible* if any two rules differ in their r.h.s. A grammar is an *operator grammar* (OG) if all r.h.s.'s are in the operator form OF(V_N); clearly, every sentential form of an OP grammar is in OF(V_N). Any context-free grammar that does not generate ε admits an equivalent OG (Theorem 4.8.1 of [16]). In this paper we deal only with OG, and assume them to be *reduced*, i.e., such that every rule is used in at least one derivation of a string belonging to its language.