Locally Chain-Parsable Languages

Stefano Crespi Reghizzi[1]

DEIB - Politecnico di Milano

Violetta Lonati[2]

DI - Università degli Sudi di Milano

Dino Mandrioli[1]

DEIB - Politecnico di Milano

Matteo Pradella[1]

DEIB - Politecnico di Milano

Syntax analysis or parsing of context-free (CF) languages is a mature research area, and good parsing algorithms are available for the whole CF family and for the deterministic subfamily (DCFL) 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 [2,3] based on Floyd's [5] operator-precedence grammars (OPG) and their languages (OPL), which are included in DCFL. This is a data-parallel algorithm that is based on a theoretical property of OPG, called *local parsability*: any arbitrary factor of a sentence can be deterministically parsed, returning the unique partial syntax-tree whose frontier is 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 production right-hand side (RHS) and reduces it. The local parsability property ensures the correctness of the syntax tree thus obtained, no matter the position of reduction applications.

[1] {stefano.crespi, dino.mandrioli, matteo.pradella}@polimi.it

[2] lonati@di.unimi.it

[3] This work will be presented at MFCS 2015.

The informal idea of local parsability is occasionally mentioned in old research on parallel parsing, and has been formalized for OPG in [2]. Our first contribution is to propose the definition of a new and more general class of locally parsable languages: the language family to be called *Locally Chain-Parsable* (LCPL), which gains in generative capacity and bypasses some inconveniences of OPG.

The other contribution is towards a generalization of the well-known family of input-driven (alias visibly push-down) languages (IDL) [8,1], which are characterized by push-down 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. Since the attention IDL have recently attracted is due to their rich closure and decidability properties, we hope that the introduction of a larger family of languages with similar properties may be also of interest.

To understand in what sense our LCPL are input-driven, we first recall that IDL 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 IDL have the local parsability property, a fact also stemming from the fact that IDL are included in OPL [4]. 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 IDL, we obtain the OPL family, which has essentially the same closure and decidability properties [4,7]. Loosely speaking, we may say that the input that drives the automaton for OPL 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 factor (which may contain nonterminals) on the purely terminal string orderly containing: the preceding terminal, the terminals of the factor, and the following terminal. Such triplet will be called a *chain* and the machine a chain-driven automaton. For a given CF grammar, the length of chains has an upper bound, which bounds the input portion that drives the choice of a move by the recognizer.

1 Chain-driven automata

The *terminal* alphabet is denoted by Σ; it includes the letter # used as start and end of text. The *nonterminal* alphabet is denoted by V_N. We also assume that in a grammar, $S \subseteq V_N$ is a *set* of initial symbols. Let Δ be an alphabet disjoint from Σ. A string $\beta \in (\Sigma \cup \Delta)^*$ is in *operator form* if it contains one or more terminals and does not contain a factor from Δ^2, i.e., no adjacent symbols from Δ; OF(Δ) denotes the set of all operator form strings over $\Sigma \cup \Delta$.

A CF grammar is an *operator grammar* (OG) if all right hand sides (r.h.s.) are in OF(V_N). Any CF grammar that does not generate ε admits an equivalent OG, which can be assumed to be invertible [6]. Clearly, every sentential form of an OP grammar is in OF(V_N). In this paper we deal only with reduced OG.

The following naming convention is adopted, unless otherwise specified: lowercase Latin letters $a, b, \ldots$ denote terminal characters; uppercase Latin letters $A, B, \ldots$ denote nonter-

minal characters; lowercase Latin letters $x, y, z \ldots$ denote terminal strings; and Greek lowercase letters $\alpha, \ldots, \omega$ denote strings over $\Sigma \cup V_N$.
We use bold symbols to denote strings over an alphabet that includes the square brackets, e.g. $\boldsymbol{x} \in (\Sigma \cup \{[,]\})^*, \boldsymbol{\alpha} \in (\Sigma \cup V_N \cup \{[,]\})^*$. We introduce the following short notation for frequently used operations based on 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 $\widetilde{\boldsymbol{\alpha}} = \widetilde{\boldsymbol{\beta}}$. Also, we use $\boldsymbol{\alpha}_{\text{first}}$ and $\boldsymbol{\alpha}_{\text{last}}$ for taking the first or last symbol in Σ from a string $\boldsymbol{\alpha}$. The same notation is applied when V_N is replaced by the state set of a machine.
For a CF grammar G, the associated *parenthesis grammar*, denoted by $[G]$, is obtained by bracketing with '[' and ']' each r.h.s. of a rule of G. Two grammars G, G' are *structurally equivalent* if $L([G]) = L([G'])$.

Definition 1 *A* chain *is a triple $a\langle y\rangle b$ with $a, b \in \Sigma$ and $y \in \Sigma^+$; (a, b) is the* context *and y the* body *of the chain.*

A chain-driven automaton works by reducing the input string through a sequence of reductions driven by a given set of chains; the automaton finds a given chain within the input string and replaces its body with a state; then the mechanism is applied recursively to the obtained string. Hence during the reduction steps the input string is shortened and simultaneously enriched by the computed states; chains being defined over the input alphabet, the portion of the input factor to be reduced is detected depending on input symbols only; enriching states are used then to (nondeterministically) determine which state will replace the detected factor.

Definition 2 *A* chain-driven automaton *is a tuple $(\Sigma, Q, C, \delta, F)$ where Σ is the input alphabet; Q is a finite set of states; C is a finite set of chains; $\delta : \Sigma \times \mathrm{OF}(Q) \times \Sigma \to \mathcal{P}(Q)$ is the* reduce *function, where $\delta(a, \gamma, b) \neq \emptyset$ implies $a\langle\widehat{\gamma}\rangle b \in C$; $F \subseteq Q$ is the set of final states.*

A *configuration* of the automaton is a string $\gamma \in \mathrm{OF}(Q)$. The *initial* configuration on input $x \in \Sigma^*$ is defined as $\#x\#$; a configuration $\#q\#$ with $q \in F$ is called an *accepting* configuration. The *reduction move* is defined as follows: the automaton may perform the move

$$\alpha\ a\gamma b\ \beta \overset{a\langle y\rangle b}{\longmapsto} \alpha\ aqb\ \beta$$

where $\widehat{\gamma} = y$, and $\delta(a, \gamma, b) \ni q$. Hence, a move of the automaton deletes a factor in $\mathrm{OF}(Q)$ (corresponding to the body of the chain in C, possibly enriched with states in Q) and replaces it with a state.

A *computation* of the automaton is a sequence $K_0 \overset{c_1}{\longmapsto} K_1 \overset{c_2}{\longmapsto} K_2 \overset{c_3}{\longmapsto} \cdots \overset{c_n}{\longmapsto} K_n$ where K_i are configurations, c_i are chains. When not relevant we omit the chain and write simply $K_1 \longmapsto K_2$. We also use $\overset{*}{\longmapsto}$ to denote the reflexive and transitive closure of $\longmapsto$. The language accepted by the automaton is defined as $L(\mathcal{A}) = \{x \in \Sigma^* \mid \#x\# \overset{*}{\longmapsto} \#q\#$ with $q \in F\}$.

Definition 3 *For a chain-driven automaton $\mathcal{A} = \langle \Sigma, Q, C, \delta, F\rangle$, the associated* parenthesis automaton *is the chain-driven automaton $[\mathcal{A}] = \langle \Sigma \cup \{[,]\}, Q, [C], \delta', F\rangle$ where $[C]$ is the set of chains $a\langle[y]\rangle b$ such that $a\langle y\rangle b \in C$, and δ' is defined by setting $\delta'(a, [\gamma], b) = \delta(a, \gamma, b)$ whenever $\delta(a, \gamma, b) \neq \emptyset$.*

A chain-driven automaton $\mathcal{A}$ is structurally ambiguous *if $L([\mathcal{A}])$ contains two strings $\boldsymbol{x_1} \neq \boldsymbol{x_2}$ such that $\boldsymbol{x_1} \,\tilde{=}\, \boldsymbol{x_2}$.*

Definition 4 *The chain $a\langle y\rangle b$ is a* grammatical chain *associated with G if there exists a derivation*

$$\#T\# \overset{*}{\underset{G}{\Longrightarrow}} \alpha\, aAb\, \beta \underset{G}{\Longrightarrow} \alpha\, a\gamma b\, \beta \tag{1}$$

with $\widehat{\gamma} = y$, $T \in S$. The set of grammatical chains associated with G is denoted by C_G.

Theorem 1 *Chain-driven automata recognize the class of CF languages.*

2 Locally chain-parsable languages

Definition 5 *A* local chain parser *(LCPA) is a chain-driven automaton such that, for every chain $a\langle y\rangle b$, the following condition holds: if $\widehat{\gamma} = y$, then every computation $\alpha\, a\gamma b\, \beta \overset{*}{\vdash\!\!\!-\!\!\!-} \#q_F\#$ with $q_F \in F$ can be decomposed as $\alpha\, a\gamma b\, \beta \overset{*}{\vdash\!\!\!-\!\!\!-} \alpha'\, a\gamma b\, \beta' \vdash\!\!\!-\!\!\!- \alpha'\, aqb\, \beta' \overset{*}{\vdash\!\!\!-\!\!\!-} \#q_F\#$ with suitable α', β', and q.*

Definition 6 *A grammar is* locally chain-parsable *(LCPG) if, for every grammatical chain $a\langle y\rangle b$, the following condition holds: if $\gamma \,\hat{=}\, y$, then each derivation $\#T\# \overset{*}{\Longrightarrow} \alpha a\gamma b\beta$ with $T \in S$ can be decomposed as $\#T\# \overset{*}{\Longrightarrow} \alpha' aAb\, \beta' \Longrightarrow \alpha' a\gamma b\, \beta' \Longrightarrow \alpha\, a\gamma b\, \beta$. A language L is locally chain-parsable (LCPL) if it is generated by a LCPG.*

Theorem 2 *A language is LCPL if and only if it is recognized by a LCPA. An LCPA recognizes any string in linear time.*

Definition 7 Structured strings *are special well-parenthesized strings over $\Sigma \cup \{[,]\}$, defined recursively as follows:*

- *$y \in \Sigma^+$ are* atomic *structured strings;*
- *if $a_i \in \Sigma$ and $\boldsymbol{y_i} = \varepsilon$ or $\boldsymbol{y_i} = [\boldsymbol{v_i}]$ for some structured strings $\boldsymbol{v_i}$, then $\boldsymbol{y_0} a_1 \boldsymbol{y_1} a_2 \ldots a_n \boldsymbol{y_n}$ is a* composed *structured string if at least one $\boldsymbol{y}_i$ is different from ε.*

An extended chain *(briefly* xchain*) is a string $\#[\boldsymbol{y}]\#$ where $\boldsymbol{y}$ is the* body *of the xchain.*

Definition 8 *Let $\mathcal{A}$ be a chain-driven automaton and G a grammar. An xchain $\#[\boldsymbol{y}]\#$ is an $\mathcal{A}$-*xchain *or a G-*xchain*, respectively, if there exist $\boldsymbol{\gamma}$ such that $\widehat{\boldsymbol{\gamma}} = \boldsymbol{y}$ and*

$$\#[\boldsymbol{\gamma}]\# \overset{*}{\underset{[\mathcal{A}]}{\vdash\!\!\!-\!\!\!-}} \#q_F\# \text{ with } q_F \in F \qquad \text{or} \qquad \#T\# \overset{*}{\underset{[G]}{\Longrightarrow}} \#[\boldsymbol{\gamma}]\# \text{ with } T \in S.$$

The sets of $\mathcal{A}$-xchains and G-xchains are denoted respectively by $\mathcal{X}_{\mathcal{A}}$ and $\mathcal{X}_G$.

Definition 9 *An xchain* conflicts *with a chain $a\langle y\rangle b$ iff it can be decomposed as $\boldsymbol{xaybz}$ where $\widetilde{\boldsymbol{y}} = y$ and $\boldsymbol{y} \notin [^+y]^+$. A set $\mathcal{X}$ of xchains and a set C of chains are* conflictual *iff there is an xchain in $\mathcal{X}$ that conflicts with some chain in C.*

Theorem 3 *The fact that $\mathcal{X}_G$ and C_G are nonconflictual is decidable for every grammar G; the fact that $\mathcal{X}_{\mathcal{A}}$ and C are nonconflictual is decidable for every automaton $\mathcal{A}$ driven by the set of chains C.*

Theorem 4 *A chain-driven automaton $\mathcal{A}$ is a local parser if and only if $\mathcal{X}_{\mathcal{A}}$ and the set C of chains driving $\mathcal{A}$ are not conflictual. A grammar G is locally parsable if and only if $\mathcal{X}_G$ and C_G are not conflictual.*

Corollary 1 *The fact that a grammar is LCPG is decidable; the fact that a chain-driven automaton is LCPA is decidable.*

Definition 10 *Two LCPA $\mathcal{A}_1 = (\Sigma, Q_1, C_1, \delta_1, F_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, C_2, \delta_2, F_2)$ are* compatible *if $\mathcal{X}_{\mathcal{A}_1} \cup \mathcal{X}_{\mathcal{A}_2}$ and $C_1 \cup C_2$ are not conflictual. Two LCPL L_1 and L_2 are* compatible *if they are recognized by compatible LCPA.*

Theorem 5 *Let $\mathcal{A}_1 = (\Sigma, Q_1, C_1, \delta_1, F_1)$ and $\mathcal{A}_2 = (\Sigma, Q_2, C_2, \delta_2, F_2)$ be compatible chain-driven automata recognizing respectively L_1 and L_2. Then $L_1 \cup L_2$, $L_1 \cap L_2$, and $L_1 \setminus L_2$ are LCPL.*

Theorem 6 *The OPL family is strictly contained within the LCPL family. Moreover every OPG is locally chain-parsable.*

LCPG grammars permit to define relevant syntactic constructs beyond the capacity of OPG. As an argument we mention a well-known practical construct: arithmetic expressions containing both unary and binary minus signs, which notoriously introduce precedence conflicts.

To our knowledge, our class of automata and grammars differs from all existing, somewhat related, models, either, or both, with respect to the local parsability property and to the input-driven aspects.

This class properly extends the known input-driven classes, preserving important closure properties, and maintains the decidability of the containment problem. This increased generative power can be exploited to define practical languages and to obtain efficient parallel parsers, thus extending the algorithm presented in [3] for OPL and replicating the obtained benefits in terms of time complexity and speed up. Further closure properties, in particular concatenation and Kleene's star, are still to be investigated.

References

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

[2] Barenghi, A., S. Crespi Reghizzi, D. Mandrioli and M. Pradella, *Parallel parsing of operator precedence grammars*, IPL (2013).

[3] Barenghi, A., S. Crespi Reghizzi, D. Mandrioli and M. Pradella, *Parallel parsing made practical*, SCP (2015), under revision.

[4] Crespi Reghizzi, S. and D. Mandrioli, *Operator Precedence and the Visibly Pushdown Property*, JCSS **78** (2012), pp. 1837–1867.

[5] Floyd, R. W., *Syntactic Analysis and Operator Precedence*, JACM **10** (1963), pp. 316–333.

[6] Harrison, M. A., "Introduction to Formal Language Theory," Addison Wesley, 1978.

[7] Lonati, V., D. Mandrioli, F. Panella and M. Pradella, *Operator precedence languages: Their automata-theoretic and logic characterization*, SICOMP (2015), to appear.

[8] Mehlhorn, K., *Pebbling mountain ranges and its application of DCFL-recognition*, in: *Automata, languages and programming (ICALP-80)*, LNCS **85**, 1980, pp. 422–435.