On Pumping RP-automata Controlled by Complete LR(¢,$)-grammars

Martin Plátek[1], František Mráz[1], Dana Pardubská[2] and Daniel Průša[3]

[1]Charles University, Department of Computer Science, Malostranské nám. 25, 118 00 Praha 1, Czech Republic
[2]Comenius University in Bratislava, Department of Computer Science, Mlynská Dolina, 84248 Bratislava, Slovakia
[3]Czech Technical University, Department of Cybernetics, Karlovo nám. 13, 121 35 Praha 2, Czech Republic

Abstract

We introduce complete LR(0)-grammars with sentinels (called complete LR(¢,$)-grammars) to prepare tools for the study of pumping restarting automata controlled by this type of grammars. A complete LR(¢,$)-grammar generates both a language and the complement of the language with sentinels. Based on a complete LR(¢,$)-grammar, we can construct a deterministic pumping restarting automaton performing pumping analysis by reduction on each word over its input alphabet. A pumping reduction analysis is a method where an input word is stepwise simplified by removing at most two continuous parts of the current word in a way that preserves (in)correctness of the word. Each such simplification corresponds to removing parts of the current word that could be "pumped" in the sense of a pumping lemma for context-free languages. The computation of a pumping restarting automaton ends when the original word is shortened under a given length, and then it is decided about the correctness or incorrectness of the original input word. This means that pumping restarting automata can analyze both correct and incorrect inputs with respect to a deterministic context-free language (DCFL). That gives an excellent formal basis for the error localization and the error recovery of DCFL.

Keywords

restarting automata, LR(0)-grammars, complete grammars, Deterministic Context-Free Languages

1. Introduction

This paper aims to enhance and refine results from papers [1, 2] where some distinguishing restrictions for deterministic monotone restarting pumping automata (det-mon-RP-automata) were introduced and studied.

Some linguistic and non-linguistic motivations for this paper can be found already in [1, 2]. Here we work mainly with a motivation to develop formal tools supporting the characterization and localization of syntactic errors in deterministic context-free languages.

Reduction analysis is a method for checking the correctness of an input word by stepwise rewriting some part of the current form with a shorter one until we obtain a simple word for which we can decide its correctness easily. In general, reduction analysis is nondeterministic, and in one step, we can rewrite a substring of a length limited by a constant with a shorter string. An input word is accepted if there is a sequence of reductions such that the final simple word is from the language. Then, intermediate words obtained during the analysis are also accepted. Each reduction must be *error preserving*, i.e., no word outside the target language can be rewritten into a word from the language.

In this paper, we are interested in a stronger version of reduction analysis called *pumping reduction analysis*. Pumping reduction analysis is a reduction analysis that has several additional properties. In each step of pumping reduction analysis, the current word is not rewritten. Instead, at most two continuous segments of the current word are deleted. Further, pumping reduction analysis works according to a so-called complete grammar.

Informally, a complete grammar (with sentinels ¢ and $) G_C is an extended context-free grammar that has two initial nonterminals S_A and S_R. Such grammar has a finite alphabet Σ of terminals not containing ¢ and $, finite alphabet of nonterminals and a set of rewriting rules of the form $X \to \alpha$, where X is a nonterminal and α is a string of terminals, nonterminals and sentinels ¢, $. The language generated by the grammar is the set L of words w such that the word $\{¢\} \cdot w \cdot \{\$\}$ can be derived from the initial nonterminal S_A and the set of words derived from the second initial nonterminal S_R is exactly $\{¢\} \cdot (\Sigma^* \setminus L) \cdot \{\$\}$.

Pumping reduction analysis corresponds to a complete grammar G_C when for each pair of terminal words u, v such that u can be reduced to v, it holds that there are some terminal words x_1, x_2, x_3, x_4, x_5, and a nonterminal A such that $u = x_1x_2x_3x_4x_5$, $v = x_1x_3x_5$, and $S \Rightarrow^*_{G_C} x_1Ax_5 \Rightarrow^*_{G_C} x_1x_2Ax_4x_5 \Rightarrow^*_{G_C} x_1x_2x_3x_4x_5$, where S equals S_A or S_R. Additionally, there exists a

ITAT'22: Information technologies – Applications and Theory, September 23–27, 2022, Zuberec, Slovakia
martin.platek@mff.cuni.cz (M. Plátek); frantisek.mraz@mff.cuni.cz (F. Mráz); pardubska@dcs.fmph.uniba.sk (D. Pardubská); prusa@fel.cvut.cz (D. Průša)
0000-0003-3147-6442 (M. Plátek); 0000-0001-9869-3340 (F. Mráz); 0000-0001-9383-8117 (D. Pardubská); 0000-0003-4866-5709 (D. Průša)

constant c that depends only on the grammar G_C such that each word of length at least c can be reduced to a shorter word.

The main result of the paper is that for each deterministic context-free language, there exists a complete grammar G_C and a deterministic restarting RP-automaton M that performs pumping reduction analysis on any input word w. The last phase of the computation of M on w will produce a terminal word w' that is not longer than the constant c. If ¢w'\$ is generated from S_A according to G_C, then w' (and thus also w) is accepted by M. Otherwise, if ¢w'\$ is generated from S_R according to G_C, then w' (and thus also w) is rejected by M.

The paper is structured as follows. Section 2 introduces RP-automata and LR(0)-grammars, and presents their basic properties. LR(0)-grammars are used for constructing a complete grammar for any deterministic context-free language.

Section 3 introduces complete grammars and presents a method for constructing a complete grammar for any given deterministic context-free language.

Section 4 presents the main results of this paper. Here, we show that for any complete grammar G_C constructed in Section 3, we can construct a deterministic restarting RP-automaton that performs pumping reduction analysis according to G_C for any input word.

Finally, Section 5 summarizes the results of the paper and gives an outlook for future research.

2. Basic notions

At first, we introduce our base automata model called RP-automata. RP-automata are restarting automata [2] that differ only slightly from the original RW-automata introduced in [3].

An *RP-restarting automaton*, or an RP-*automaton*, $M = (Q, \Sigma, ¢, \$, q_0, k, \delta, Q_\mathrm{A}, Q_\mathrm{R})$ (with k-bounded lookahead) is a device with a finite state control unit with the finite set of states Q containing two disjunctive subsets $Q_\mathrm{A}, Q_\mathrm{R}$ of accepting and rejecting states, respectively. The automaton is equipped with a head moving on a finite linear flexible tape of items (cells). The first item of the tape always contains the left sentinel symbol ¢, the last one the right sentinel symbol \$, and each other item contains a symbol from a finite alphabet Σ (not containing ¢, \$). The head has a flexible read/write window of length at most k (for some $k \geq 1$) – M scans k consecutive items or the rest of the tape when the distance to the right sentinel \$ is less than k. We say that M is of window size k. In the *initial configuration* on an input word $w \in \Sigma^*$, the tape contains the input word delimited by the sentinels ¢ and \$, the control unit is in the initial state q_0, and the window scans the left sentinel ¢ and the first $k-1$ symbols of the input word (or the rest of the tape if the tape contents is shorter than k).

The *computation* of M is controlled by the *transition function*

$$\begin{aligned}\delta : (Q \setminus (Q_\mathrm{A} \cup Q_\mathrm{R})) \times \mathcal{PC}^{(k)} \to \\ \mathcal{P}(Q \times \{MVR, PREPARE\}) \ \cup \\ \mathcal{P}((Q_\mathrm{A} \cup Q_\mathrm{R}) \times \{HALT\}) \ \cup \\ \{RESTART(v) \mid v \in \mathcal{PC}^{\leq(k-1)}\}.\end{aligned}$$

Here $\mathcal{P}(S)$ denotes the powerset of the set S, $\mathcal{PC}^{(k)}$ is the set of *possible contents* of the read/write window of M, where for $i, n \geq 0$

$$\mathcal{PC}^{(i)} = (\{¢\} \cdot \Sigma^{i-1}) \cup \Sigma^i \cup (\Sigma^{\leq i-1} \cdot \{\$\}) \cup (\{¢\} \cdot \Sigma^{\leq i-2} \cdot \{\$\}),$$

$$\Sigma^{\leq n} = \bigcup_{i=0}^{n} \Sigma^i \quad \text{and} \quad \mathcal{PC}^{\leq(k-1)} = \bigcup_{i=0}^{k-1} \mathcal{PC}^{(i)}.$$

The transition function δ represents a finite set of four different types of instructions (transition steps). Let q, q', q_I be states from Q, $u \in \mathcal{PC}^{(k)}$, $w \in \mathcal{PC}^{\leq(k)}$, $v \in \mathcal{PC}^{\leq(k-1)}$ and M be in state q with u being the contents of its read/write window:

(1) A *move-right instruction* of the form $(q, u) \to_\delta (q', MVR)$ is applicable if $u \notin \Sigma^*\$$. It causes M to enter the state q' and to move its read/write head one item to the right.

(2) *A preparing instruction* of the form $(q, u) \to_\delta (q_I, PREPARE)$ changes M's state to a restarting state q_I that determines the next instruction, which must be a restarting instruction.

(3) *A restarting instruction* is of the form $(q_I, w) \to_\delta RESTART(v)$, where $|v| < |w|$ and if w contains any sentinel, v contains the corresponding sentinels, too. This instruction is applicable if w is a prefix of the contents of the read/write window. When executed, M replaces w with v (with this, it shortens its tape) and restarts – i.e. it enters the initial state and places the window at the leftmost position so that the first item in the window contains ¢. Note that the state q_I unambiguously gives the pair (w, v). We can assume that all pairs (w, v) where $|w| > |v|$ and the word w can be replaced with v by some $RESTART$ instruction are ordered and that I is the index of (w, v) in that sequence. Thus, although the RP-automaton is generally nondeterministic, each $RESTART$ instruction of M corresponds unambiguously to one restart state q_I.

(4) *A halting instruction* of the form $(q, u) \to_\delta (q', HALT)$, where $q' \in Q_\mathrm{A}$ or $q' \in Q_\mathrm{R}$, finishes the computation and causes M to accept or reject, respectively, the input word.

Thus, the set of states can be divided into three groups – the *halting states* $Q_\mathrm{A} \cup Q_\mathrm{R}$, the *restarting states* (involved on the left-hand side of restarting instructions), and the rest, called the *transition states.*

A *configuration* of an RP-automaton M is a word $\alpha q \beta$, where $q \in Q$, and either $\alpha = \lambda$, where λ denotes the empty word, and $\beta \in \{¢\} \cdot \Sigma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot$

Σ^* and $\beta \in \Sigma^* \cdot \{\$\}$; here q represents the current state, $\alpha\beta$ is the current contents of the tape, and it is understood that the read/write window contains the first k symbols of β or all symbols of β if $|\beta| < k$. An *initial (restarting) configuration* is $q_0 ¢ w \$$, where $w \in \Sigma^*$. A *rewriting configuration* is of the form $\alpha q_I \beta$, where q_I is a restarting state.

A *computation* of M is a sequence $C = C_0, C_1, \ldots, C_j$ of configurations of M, where C_0 is a restarting configuration, $C_{\ell+1}$ is obtained from C_ℓ by a step of M, for all ℓ, $0 \leq \ell < j$, denoted as $C_\ell \vdash_M C_{\ell+1}$, and $\vdash_M^*$ is the reflexive and transitive closure of the single-step relation $\vdash_M$.

In general, an RP-automaton can be *nondeterministic*, i.e. there can be two or more instructions with the same left-hand side. If that is not the case, the automaton is *deterministic*. In what follows, we are interested in deterministic RP-automata, denoted det-RP.

An input *word w is accepted by* M if there is a computation that starts in the initial configuration with w (bounded by sentinels ¢, \$) on the tape and finishes in an *accepting configuration* where the control unit is in one of the accepting states. $L(M)$ denotes the language consisting of all words accepted by M; we say that M *accepts the language* $L(M)$.

Let M be deterministic. $L_R(M)$ denotes the language consisting of all words rejected by M; we say that M *rejects the language* $L_R(M)$.

Restarting steps divide any computation of an RP-automaton into certain phases that all start in the initial state in restarting configurations with the read/write window in the leftmost position. In a phase called *cycle*, the head moves to the right along the input list (with its read/write window) until a restart occurs – in that case, the computation is resumed in the initial configuration on the new, shorter word. The phase from the last restart to the halting configuration is called *tail*. This immediately implies that any computation of any RP-automaton is finite (ending in a halting state).

The next proposition expresses a certain lucidness of computations of deterministic RP-automata. The notation $u \Rightarrow_M v$ means that there exists a cycle of M starting in the initial configuration with the word u on its tape and finishing in the initial configuration with the word v on its tape; the relation $\Rightarrow_M^*$ is the reflexive and transitive closure of $\Rightarrow_M$. We say that M reduces u to v if $u \Rightarrow_M v$.

The validity of the following proposition is obvious.

Proposition 1. (Correctness preserving property.) *Let M be a deterministic* RP*-automaton and $u \Rightarrow_M^* v$ for some words u, v. Then $u \in L(M)$ iff $v \in L(M)$.*

By a *monotone* RP*-automaton*, we mean an RP-automaton where the following holds for all computations: the number of items to the right from the rightmost item scanned by a restarting instruction in a cycle is not increasing during the whole computation. It means that during any computation of a monotone RP-automaton the rightmost scanned items by restarting operations do not increase their distances from the right sentinel \$.

Considering a deterministic RP-automaton $M = (Q, \Sigma, ¢, \$, q_0, k, \delta, Q_{\mathrm{A}}, Q_{\mathrm{R}})$, it is for us convenient to suppose it to be in the *strong cyclic form*; it means that the words of length less than k, k being the length of its read/write window, are immediately (hence in a tail) accepted or rejected, and that M performs at least one cycle (at least one restarting) on any longer word. For each RP-automaton M, we can construct an RP-automaton M' in strong cyclic form accepting the same language as M but possibly with greater size of the read/write window [4].

We use the following obvious notation. RP denotes the class of all (nondeterministic) RP-automata. Prefix det- denotes the deterministic version, similarly mon- the monotone version. Prefix scf- denotes the version in the strong cyclic form. $\mathcal{L}(A)$, where A is some class of automata, denotes the class of languages accepted by automata from A. E.g., the class of languages accepted by deterministic monotone RP-automata is denoted by $\mathcal{L}(\text{det-mon-RP})$.

Since all computations of RP-automata are finite and the correctness preserving property holds for all deterministic RP-automata, the following proposition is obvious.

Proposition 2. *The classes $\mathcal{L}(\text{det-mon-RP})$ and $\mathcal{L}(\text{det-RP})$ are closed under complement.*

Definition 3. *Let $M = (Q, \Sigma, ¢, \$, q_0, k, \delta, Q_{\mathrm{A}}, Q_{\mathrm{R}})$ be a det-*RP*-automaton and $u \in \Sigma^*$.*

Let $\mathrm{AR}(M, u) = (u, u_1, u_2, \cdots, u_n)$, where $u \Rightarrow_M u_1 \Rightarrow_M u_2 \Rightarrow_M \cdots \Rightarrow_M u_n$, and u_n cannot be reduced by M. We say that $\mathrm{AR}(M, u)$ is the analysis by reduction of u by M.

Let $\mathrm{AR}(M) = \{\mathrm{AR}(M, u) | u \in \Sigma^*\}$. We say that $\mathrm{AR}(M)$ is analysis by reduction by M.

Let $\mathrm{AR}(\mathrm{A}, M) = \{\mathrm{AR}(M, u) | u \in L(M)\}$. We say that $\mathrm{AR}(\mathrm{A}, M)$ is accepting analysis by reduction by M.

Let $\mathrm{AR}(\mathrm{R}, M) = \{\mathrm{AR}(M, u) | u \in L_R(M)\}$. We say that $\mathrm{AR}(\mathrm{R}, M)$ is rejecting analysis by reduction by M.

2.1. LR(0) grammars

The proof of our main result is strongly based on the theory of LR(0) grammars. We will recall the definition and properties of LR(0) grammars from Harrison [5]. In contrast to Harrison, we will use the following notation for context-free grammar $G = (N, \Sigma, S, R)$, where N is a set of nonterminals, Σ is a set of terminals, $S \in N$ is the initial symbol and R is a finite set of rules of the form $X \to \alpha$, for $X \in N$ and $\alpha \in (N \cup \Sigma)^*$. We use a common notation $\Rightarrow_R$ for a *right derivation* rewriting step according to G. For two words $w, w' \in (N \cup \Sigma)^*$, $u \Rightarrow_R v$, if there exist words $\alpha, \beta \in (N \cup \Sigma)^*$, $w \in \Sigma^*$ and nonterminal $X \in N$ such that $u = \alpha X w$, $v = \alpha\beta w$ and $X \to \beta$ is a rule from R. The reflexive and transitive closure of the relation $\Rightarrow_R$ we denote as $\Rightarrow_R^*$.

Definition 4 ([5]). *Let $G = (N, \Sigma, S, R)$ be a context-free grammar and $\gamma \in (N \cup \Sigma)^*$. A* handle of *$\gamma$ is an ordered pair (r, i), $r \in R, i \geq 0$ such that there exists $A \in N, \alpha, \beta \in (N \cup \Sigma)^*$ and $w \in \Sigma^*$ such that*

(a) $S \Rightarrow_R^* \alpha A w \Rightarrow_R \alpha\beta w = \gamma$,
(b) $r = A \to \beta$*, and*
(c) $i = |\alpha\beta|$.

In general, the identification of a handle in a string is not uniquely defined, which is not true for LR(0) grammars.

Definition 5. *Let $G = (N, \Sigma, S, R)$ be a reduced context-free grammar such that $S \Rightarrow_R^+ S$ is not possible in G. We say G is an $LR(0)$ grammar if, for each $w, w', x \in \Sigma^*$, $\eta, \alpha, \alpha', \beta, \beta' \in (N \cup \Sigma)^*$, and $A, A' \in N$,*

(a) $S \Rightarrow_R^* \alpha A w \Rightarrow_R \alpha\beta w = \eta w$
(b) $S \Rightarrow_R^* \alpha' A' x \Rightarrow_R \alpha'\beta' x = \eta w'$

implies $(A \to \beta, |\alpha\beta|) = (A' \to \beta', |\alpha'\beta'|)$.

Note that as a consequence of the above definition we have that $A = A'$, $\beta = \beta'$, $\alpha = \alpha'$, $\eta = \alpha\beta = \alpha'\beta'$ and $x = w'$. Thus, if G is an LR(0) grammar, then the rightmost derivation of the word w by G and the left-right analysis is unique (deterministic). In this paper, we consider $LR(0)$ grammars rather as analytical grammars. A language generated by an LR(0) grammar is called an LR(0) language.

In [5], there is shown that every LR(0) language is deterministic context-free, and for each deterministic context-free language $L \subseteq \Sigma^*$ and symbol $\$ \notin \Sigma$, the language $L \cdot \{\$\}$ is LR(0). Further, the monograph describes how to construct an "LR-style parser". Let us sketch how such a parser P works for an LR(0) grammar $G = (N, \Sigma, S, R)$. The parser is actually a pushdown automaton that stores alternately symbols from $N \cup \Sigma$ and certain tables. For a given LR(0) grammar, the set $\mathcal{T}$ of possible tables is finite and there exist two functions

f : $\mathcal{T} \to \{shift, error\} \cup \{reduce\ \pi \mid \pi \in R\}$ is the parsing action function, and
g : $\mathcal{T} \times (N \cup \Sigma) \to \mathcal{T} \cup \{error\}$ is the goto function.

We will omit the details of how the set of tables $\mathcal{T}$ and the functions f and g are constructed. But we will describe how the LR(0) parser for the LR(0) grammar G works on an input word w. At first, an initial table τ_0 is stored at the bottom of the pushdown. Let $z \in \Sigma^*$ denote the unread part of the input, and $\gamma\tau$, where $\gamma \in \mathcal{T}^*$ and $\tau \in \mathcal{T}$, is the contents of the pushdown. Then, the parser performs repeatedly the following actions:

1. If $f(\tau) = shift$, then
   a) if $z = \lambda$, then the parser P outputs an error and rejects the input word,
   b) if $z = az'$, for some $a \in \Sigma$ and $z' \in \Sigma^*$, then
      i. if $g(\tau, a) = error$, then the parser P outputs an error and rejects the input word,
      ii. if $g(\tau, a) \neq error$, then the parser P pushes a and $g(\tau, a)$ onto its pushdown.
2. If $f(\tau) = reduce\ \pi$, where π is a rule $A \to \beta$ from R, then P pops $2|\beta|$ symbols from the pushdown and outputs the rule π. Let τ' be the table that is uncovered at the top of the pushdown.
   a) If $\tau' = \tau_0$, $A = S$, and $z = \lambda$, then the parser accepts. The output is the reversed sequence of rules that, starting from the initial nonterminal S, when applied iteratively on the rightmost nonterminal in the current word from $(N \cup \Sigma)^*$, produces the input w.
   b) If $g(\tau', A) = error$, then the parser P outputs an error and rejects the input word.
   c) Otherwise, P pushes A and $g(\tau', A)$ onto its pushdown.

In what follows, we will refer to an LR(0) analyzer as a pushdown automaton. Based on the way how it is constructed, the pushdown automaton has several properties that are essential for our constructions below:

- The pushdown automaton is deterministic.
- If a word w is accepted by P, then the output of P corresponds to a unique derivation tree.
- Let at some step of the computation of P on input w the contents of its pushdown store be $\tau_0\alpha_1\tau_1\alpha_2 \cdots \tau_{n-1}\alpha_n\tau_n$, for an integer $n \geq 0$, $\tau_0, \tau_1, \ldots, \tau_n \in \mathcal{T}$, $\alpha_1, \ldots, \alpha_n \in (N \cup \Sigma)$, $w = z_r z$, where $z_r \in \Sigma^*$ is the already processed prefix of w and $z \in \Sigma^*$ is the unread part of w.
  – If $f(\tau_n) \neq error$, then there exists a word t such that $S \Rightarrow_R^* \alpha_1 \cdots \alpha_n t \Rightarrow_R^* z_r t$.
  – If $f(\tau_n) = error$, then there is no word t such that $S \Rightarrow_R^* \alpha_1 \cdots \alpha_n z t \Rightarrow_R^* z_r t$. That is, for all words $t \in \Sigma^*$, $z_r t \notin L(G)$.
  – There exist words $z_1, z_2, \ldots, z_n \in \Sigma^*$ such that $z_r = z_1 \cdots z_n$ and $\alpha_i \Rightarrow_R^* z_i$, for $i = 1, \ldots, n$. There exist derivation

sub-trees $T_1, \ldots, T_n$ according to G such that the root of T_i is labeled α_i and the labels of the leaves of T_i concatenated is the word z_i, for $i = 1, \ldots, n$.

3. LR(¢,$)-grammars

We introduce LR(¢,$)-grammars to obtain grammars that can control RP-automata in such a way that this type of automata will characterize DCFL and regular languages by pumping reductions.

Definition 6. *Let* $¢, \$ \notin (N \cup \Sigma)$ *and* $G = (N, \Sigma \cup \{¢, \$\}, S, R)$ *be an* $LR(0)$ *grammar generating a language of the form* $\{¢\} \cdot L \cdot \{\$\}$*, where* $L \subseteq \Sigma^*$*, and* S *does not occur in the right-hand side of any rule from* R*.*

We say that G *is an LR(¢,$)-grammar. We denote the set of LR(¢,$)-grammars by* LRG(¢, $)*. W.l.o.g., we suppose that an LR(¢,$)-grammar does not contain rewriting rules of the form* $A \to \lambda$ *for any nonterminal* $A \in N$*.*

We say that L *is the* internal language of G *and denote it as* $L_{In}(G)$*.*

Classes of languages. In what follows, $\mathcal{L}(A)$, where A is some (sub)class of grammars or automata, denotes the class of languages generated/accepted by grammars/automata from A. E.g., the class of languages generated by linear LR(¢,$)-grammars is denoted by $\mathcal{L}(lin$-LR(¢,$)). Similarly, for some (sub)class of LR(¢,$)-grammars A we take for internal languages $\mathcal{L}_{In}(A) = \{L \mid \{¢\} \cdot L \cdot \{\$\} \in \mathcal{L}(A)\}$.

Based on the closure properties of DCFL shown, e.g., in [5], internal languages of LR(¢,$)-grammars can be used to represent all deterministic context-free languages.

Proposition 7. $\mathcal{L}_{In}(LRG(¢, \$)) = DCFL$.

Proof. Let $L \subset \Sigma^*$, and L be a DCFL. Let ¢ and $ be not from Σ. We know from [5] that $L \cdot \{\$\}$ is a strict deterministic language, i.e., it is accepted by a deterministic pushdown automaton by empty store. Therefore, $\{¢\} \cdot L \cdot \{\$\}$ is also a strict deterministic language. This implies that there is an LR(¢,$)-grammar G such that $L(G) = \{¢\} \cdot L \cdot \{\$\}$.

On the other hand, if L is the inner language of an LR(¢, $)-grammar, the language $\{¢\} \cdot L \cdot \{\$\}$ is LR(0), and it can be accepted by a deterministic pushdown automaton. Using closure properties of DCFL [5], we can prove that L is also in DCFL. That finishes the proof. $\square$

Note. It is not hard to see that the languages from $\mathcal{L}(LRG(¢, \$))$ are prefix-free and suffix-free languages at the same time.

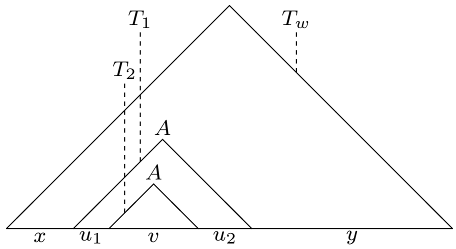

Figure 1: The structure of a derivation tree.

3.1. Pumping notions by LR(¢,$)-Grammars

This section studies the pumping properties of context-free grammars. We start with several definitions and notations.

Pumping reduction. Let $G = (N, \Sigma, S, R)$ be a context-free grammar, x, u_1, v, u_2, y be words over Σ, $|u_1| + |u_2| > 0$ and $A \in N$ be a nonterminal. If

$$S \Rightarrow^* xAy \Rightarrow^* xu_1Au_2y \Rightarrow^* xu_1vu_2y \qquad (1)$$

we say that $xu_1vu_2y \Leftarrow_{P(G)} xvy$ is a *pumping reduction* according to grammar G. Here $\Rightarrow$ denotes the rewriting relation according to a rule of G that need not be a right derivation. Then $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$.

If a word w can be generated by G, then there exists a sequence of words $w_1, \ldots, w_n$ from Σ^*, for some integer $n \geq 1$, such that $w = w_1$, there are pumping reductions $w_i \Leftarrow_{P(G)} w_{i+1}$, for all $i = 1, \ldots, n-1$, and there is no pumping reduction $w_n \Leftarrow_{P(G)} w_{n+1}$, for any $w_{n+1} \in \Sigma^*$.

Let T_w be a derivation tree corresponding to derivation (1), where $w = xu_1vu_2y$. See Fig. 1. The proper sub-trees T_1 and T_2 of T_w are sub-trees whose roots are labelled with the same nonterminal A, thus by replacing T_1 with T_2 properly inside of T_w, we again get a derivation tree, namely the derivation tree $T_{w(0)}$ for the word $w(0) = xvy \in L(G)$.

Analogously, by replacing T_2 with a copy of T_1, we get the derivation tree $T_{w(2)}$ for a longer word $w(2) = xu_1^2vu_2^2y$. If we replace T_2 with T_1 i times, we obtain the derivation tree $T_{w(i+1)}$ for the word $w(i+1) = xu_1^{i+1}vu_2^{i+1}y$.

Pumping tree, prefix, reduction, and their patterns. Let $x, u_1, v, u_2, A, y, T_1$ be as on Fig. 1.

We say that $p_p = xu_1vu_2$ is a *pumping prefix* by G with the pumping pattern (x, u_1, A, v, u_2). We also say that p_p is an (x, u_1, A, v, u_2)-pumping prefix. If $|u_1| > 0$ and $|u_2| > 0$, we say that p_p is a two-sided pumping prefix. Otherwise, we say that p_p is a one-sided pumping prefix by G.

We say that T_1 is the pumping tree of p_p. Let us recall that, for any $t \in \Sigma^*$, it holds that $xu_1vu_2t \in L(G)$ iff $xvt \in L(G)$.

Recall that we suppose that $|u_1u_2| > 0$.

Definition 8. *Let p_p be a (x, u_1, v, u_2)-pumping prefix by G, and $x, u_1, v, u_2, A, y, T_1$ be as on Fig. 1. We say that p_p is an* e-leftmost *(elementary leftmost) pumping prefix by G, and that (x, u_1, A, v, u_2) is an* e-leftmost *pumping pattern if there is no proper prefix of p_p such that it has a pumping pattern different from (x, u_1, A, v, u_2). We say in this case that T_1 is the* e-leftmost *pumping tree.*

Let z be a word from Σ^*. We write $xu_1vu_2z \Leftarrow_{P(G,e)} xvz$, and say that $xu_1vu_2z \Leftarrow_{P(G,e)} xvz$ is an e-leftmost pumping reduction by G, and that (x, u_1, A, v, u_2) is also the pumping pattern of the e-leftmost pumping reduction $xu_1vu_2z \Leftarrow_{P(G,e)} xvz$. We also say that $xu_1vu_2z \Leftarrow_{P(G,e)} xvz$ is an e-leftmost (x, u_1, A, v, u_2)-pumping reduction by G, and that $xu_1vu_2 \Leftarrow_{P(G,e)} xv$ is the smallest e-leftmost (x, u_1, A, v, u_2)-pumping reduction by G.

Note that, in the above definition, the word xu_1vu_2z need not be generated by G, but $xu_1vu_2z \Leftarrow_{P(G,e)} xvz$ is still an e-leftmost pumping reduction by G. This is important, as we will use such reduction also when analyzing words not generated by G.

The notion of e-leftmost pumping reduction gives us a basis for a special type of analysis by reduction for $L(G)$, and mainly for analysis by reduction for $L_{In}(G)$. The next notions are the most important notions of this paper.

Core pumping pattern. We say that a pumping pattern (x, u_1, A, v, u_2) by G is a *core pumping pattern* if there is y such that xvy cannot be reduced by any pumping reduction by G. We say that the tuple (u_1, A, v, u_2) is a pumping core by G.

One-sided and two-sided (core) pumping pattern. Let (x, u_1, A, v, u_2) be a (core) pumping pattern by G. We say that (x, u_1, A, v, u_2) is a one-sided (core) pumping pattern if $u_1 = \lambda$, or $u_2 = \lambda$. We say that (x, u_1, A, v, u_2) is a two-sided (core) pumping pattern if $u_1 \neq \lambda$, and $u_2 \neq \lambda$.

Non-pumping accepting trees/words/derivations. Let $z \in \Sigma^*$ and

$$S \Rightarrow_R \alpha_0 \Rightarrow_R \alpha_1 \cdots \alpha_n \Rightarrow_R z \qquad (2)$$

be a right derivation by G. Let T be the derivation tree corresponding to derivation (2). Let no repetition of a nonterminal occurs on any path from the root of T to a leaf of T. We say that T is a *non-pumping accepting tree*, derivation (2) is a *non-pumping accepting derivation*, and z is a *non-pumping accepting word* by G.

Notation. Let $G = (N, \Sigma, S, R)$ be an LR(¢,\$)-grammar, t be the number of nonterminals of G, and k be the maximal length of the right-hand side of the rules from R. If T is a non-pumping accepting tree according to G then it cannot have more than k^t terminal leaves. If T has more than k^t leaves, then there exists a path from a leaf to the root of T containing at least $t+1$ nodes labelled by nonterminals, and T is not a non-pumping tree. Let $K_G = k^t$. We say that K_G is the *grammar number of* G.

Note that any word from $L(G)$ of length greater than K_G must contain a core pumping pattern by G. On the other hand, the length of any non-pumping accepting word by G is at most K_G.

We can see the following obvious proposition that summarizes the leftmost pumping properties of LR(¢, \$)-grammars, which we will use in the following text. It is a direct consequence of the previous definitions and the properties of LR(0)-grammars and their LR(0) analyzers.

Proposition 9. *Let $G = (N, \Sigma \cup \{¢, \$\}, R, S)$ be an LR(¢,\$)-grammar generating (analyzing) the language $\{¢\} \cdot L \cdot \{\$\}$. Let p_p be an e-leftmost pumping prefix by G with the pumping pattern (x, u_1, A, v, u_2), and $xu_1vu_2 \Leftarrow_{P(G,e)} xv$ be the corresponding smallest e-leftmost pumping reduction by G. Then*

(a) Any $w \in \{¢\} \cdot L \cdot \{\$\}$ determines its derivation tree T_w by G unambiguously.

(b) An e-leftmost pumping prefix by G determines its pumping tree unambiguously.

(c) $xu_1^{m+1}v{u_2}^{n+1}z \Leftarrow_{P(G,e)} xu_1^m v{u_2}^n z$ is an e-leftmost pumping reduction by G for any $m, n \geq 0$, and $z \in \Sigma^*$.

(d) $¢xvz\$ \in L(G)$ iff $¢xu_1^m vu_2^m z\$ \in L(G)$ for any $m \geq 0$, and any $z \in \Sigma^*$.

(e) Let $P_r = xu_1vu_2z \Leftarrow_{P(G,e)} xvz$ be an e-leftmost pumping reduction by G. Then P_r is determined unambiguously by the e-leftmost pumping prefix xu_1vu_2.

(f) $¢xu_1^n vu_2^m z\$ \in L(G)$ iff $¢xu_1^{n+k} vu_2^{m+k} z\$ \in L(G)$ for all $m, n, k \geq 0$.

The previous proposition is essential for our further considerations. It shows that, for a non-empty u_1, the distance of the place of pumping from the left end is not limited, and the position of pumping is determined by the pumping prefix of the pumping reduction.

Observation. It is not hard to see that for any e-leftmost pumping pattern $P_l = (x, u_1, A, v, u_2)$ there exists a prefix x_1 of x such that $P_e = (x_1, u_1, A, v, u_2)$ is a core pumping pattern. We say that P_e corresponds to P_l.

Example 1. *Consider the non-regular deterministic context-free language $L = \{¢a^nb^n\$ \mid n \geq 1\}$ with the internal language $\{a^nb^n \mid n \geq 1\}$ that is generated by the LR(¢,\$) grammar $G = (\{S, S_1, a, b\}, \{a, b\}, R, S)$),*

State	Item set	Reg. expression	parsing action function f
0	$\{S \to \cdot ¢S_1\$\}$	λ	shift
1	$\{S \to ¢ \cdot S_1\$, S_1 \to \cdot aS_1b, S_1 \to \cdot ab\}$	$¢$	shift
2	$\{S_1 \to a \cdot S_1b, S_1 \to a \cdot b, S_1 \to \cdot aS_1b, S_1 \to \cdot ab\}$	$¢a^+$	shift
3	$\{S \to ¢S_1 \cdot \$\}$	$¢a^+S_1$	shift
4	$\{S \to ¢S_1\$\cdot\}$	$¢S_1\$$	reduce $S \to ¢S_1\$$
5	$\{S_1 \to ab\cdot\}$	$¢a^+b$	reduce $S_1 \to ab$
6	$\{S_1 \to aS_1 \cdot b\}$	$¢a^+S_1$	shift
7	$\{S_1 \to aS_1b\cdot\}$	$¢a^+S_1b$	reduce $S_1 \to aS_1b$

Table 1
$LR(0)$ automaton states and regular expressions representing words reaching the states from the initial state 0.

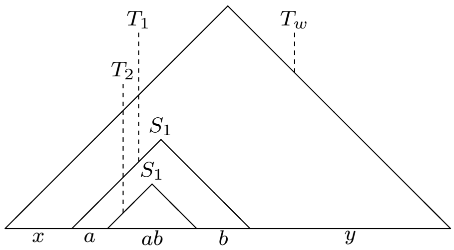

Figure 2: The structure of a derivation tree for $w = xaabby$.

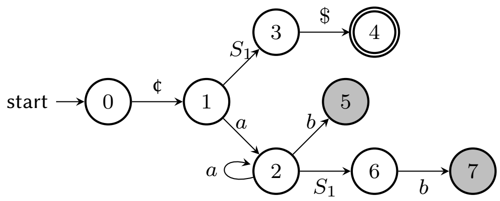

Figure 3: $LR(0)$ automaton.

with the following set of production rules R:

$$\begin{array}{lcl} S & \to & ¢S_1\$ \\ S_1 & \to & aS_1b \mid ab \end{array}$$

The grammar is reduced. Consider the sentence $\gamma = ¢aaabbb\$$.

- *The handle of γ (cf. Definition 4) is the pair $(S_1 \to ab, 5)$, as*

  $$S \Rightarrow_R^* ¢aaS_1bb\$ \Rightarrow_R ¢aaabbb\$$$

  and the division of γ into α, β, w is unique:

  $$\gamma = \underbrace{¢aa}_{\alpha}\underbrace{ab}_{\beta}\underbrace{bb\$}_{w}.$$

- *We can see that G is a linear LR(¢, \$)-grammar, as*
  (a) $S \Rightarrow_R^* \alpha Aw \Rightarrow_R \alpha\beta w = \eta w$
  (b) $S \Rightarrow_R^* \alpha' A' x \Rightarrow_R \alpha'\beta' x = \eta w'$

  obviously implies $(A \to \beta, |\alpha\beta|) = (A' \to \beta', |\alpha'\beta'|)$, because $A = S_1$, $\alpha = a^n$, $w = a^n$, $\beta = aS_1b$, for some $n \geq 0$.

The pumping notions can be illustrated in Fig. 2 with a derivation tree for $w = xaabby \in L(G)$, where $x = ¢a^i$, $y = b^i\$$, for any $i \geq 0$.

For $x = ¢a$, $p_p = ¢aaabb$ is an e-leftmost (a, a, S_1, ab, b)-pumping prefix *by G, and T_1 is an e-leftmost pumping tree of p_p. The pumping pattern of p_p by G is (a, a, S_1, ab, b) and $¢aaabb \Leftarrow_{P(G)} ¢aab$ is an e-leftmost (a, a, S_1, ab, b)-pumping reduction by G. Realize that a pumping reduction by G can be applied to any word $¢a^kb^m$, where $k, m > 1$, including the cases when $k \neq m$.*

Moreover, (λ, a, S_1, ab, b) is a core pumping pattern by G.

Table 1 lists the set of tables $\mathcal{T}$ of the $LR(0)$ automaton for the grammar G together with the corresponding parsing action function f. The column with regular expressions summarizes by which strings are the particular states reachable from the initial state.

Table 2 lists the corresponding goto function g of the LR(0) analyzer.

The goto function of the LR(0) automaton can be represented as a finite automaton A with tables as states (see Fig. 3). Note that state 4 is accepting and states 5, 7 are reducing.

Let us interpret all reducing states of the LR(0) automaton A as accepting states of the finite automaton A. What is the regular language accepted by automaton A? The language contains all prefixes $\alpha\beta$ ($\alpha, \beta \in (N \cup \Sigma)^*$) of right sentential forms according to G (obtained from the initial nonterminal using right derivation rewriting steps) such that β is a right-hand side of a production rule of G and there is no proper prefix that can be reduced according to G. Formally:

$$L(A) = \{\alpha\beta \mid \alpha, \beta \in (N \cup \Sigma)^*, \exists w \in \Sigma^*, A \in N : S \Rightarrow_R^* \alpha Aw \Rightarrow_R \alpha\beta w\}.$$

State τ	$g(\tau, ¢)$	$g(\tau, a)$	$g(\tau, b)$	$g(\tau, \$)$	$g(\tau, S)$	$g(\tau, S_1)$
0	1	error	error	error		error
1	error	2	error	error		3
2	error	2	5	error		6
3	error	error	error	4		error
4						
5						
6	error	error	7	error		error
7						

Table 2
Goto function g of the LR(0) analyzer for grammar G. Note that for reduction states 4, 5, and 7 the goto function is not defined. Similarly, the goto function is not defined for the initial nonterminal S.

The automaton A enables to distinguish two types of errors with respect to $L_{in}(G)$.

1) *A correct non-empty prefix of $L(G)$ which is turned incorrect by appending the right sentinel. This corresponds to all strings $\alpha \in (¢ \cdot \{a, b, S\}^*)$ such that, after reading a prefix α of a sentential form, the automaton reaches a state $s \in \{0, 1, 2, 6\}$ with undefined transition for \$. All such strings are represented by the regular expression $R_1 = ¢a^* + ¢a^+S_1$.*
2) *A correct non-empty prefix of $L(G)$ which is turned incorrect by appending one more symbol from $\{a, b, S_1\}$. This corresponds to all sentential forms with a prefix of the form $¢\beta c$, where $\beta \in \{a, b, S_1\}^*$ and $c \in \{a, b, S_1\}$, such that, after reading $¢\beta$, the automaton reaches a state $s \in \{0, 1, 2, 3, 6\}$ with an error transitions for c. These strings are represented by the regular expression $R_2 = ¢b + ¢S_1(a+b+S_1) + ¢a^+S_1(a+S_1)$.*

Now, the $LR(¢, \$)$-grammar generating the language $¢ \cdot \overline{L(G_{in})} \cdot \$$ can be obtained by transforming the regular expression

$$R_1\$ + R_2(a + b)^*\$$$

into an equivalent regular grammar followed by adding productions of the grammar G_{in}. This is the essential observation for constructing complete LR(¢,\$)-grammar below.

3.2. Complete LR(¢,\$)-grammars

In this section, we introduce the complete LR(¢, \$)-grammar that will be used for constructing scf-mon-RP-automaton performing (complete) pumping analysis by reduction on any word over its input alphabet Σ. A complete LR(¢, \$)-grammar is a normalized grammar that analyzes both its internal language and its complement and which, in its analytic mode, returns exactly one derivation tree for each input word of the form $¢w\$$, where $w \in \Sigma^*$. The accepting and rejecting analytic trees are distinguished by the nonterminal under their root.

One-sided LR(¢,\$)-grammar. Let G be an LR(¢,\$)-grammar. We say G is a *one-sided grammar* if all its core pumping patterns are one-sided infixes.

Definition 10. *An LR(¢,\$) grammar $G = (N, \Sigma, S, R)$ is called a complete LR(¢,\$) grammar if*

1. *$L(G) = \{¢\} \cdot \Sigma^* \cdot \{\$\}$.*
2. *$S \to S_A \mid S_R$, where $S_A, S_R \in N$, are the only rules in R containing the initial nonterminal S. No other rule of G contains S_A or S_R in its righthand side.*
3. *The languages $L(S_A)$ and $L(S_R)$ generated by the grammars $G_A = (N, \Sigma, S_A, R)$ and $G_R = (N, \Sigma, S_R, R)$, respectively, are disjoint and complementary with respect to $\{¢\} \cdot \Sigma^* \cdot \{\$\}$. That is, $L(G_A) \cap L(G_R) = \emptyset$ and $L(G) = L(G_A) \cup L(G_R) = \{¢\} \cdot \Sigma^* \cdot \{\$\}$.*

We will denote the grammar as $G = (G_A, G_R)$. Further, we will call G_A and G_R as accepting and rejecting grammar of the complete LR(¢,\$)-grammar G, respectively.

Now we will prove the main theorem.

Theorem 1. *For any LR(¢,\$)-grammar G_A, there exists a complete LR(¢,\$)-grammar $G_C = (G_A, G_R)$.*

Proof. Let $G_A = (N_A, \Sigma \cup \{¢, \$\}, S_A, R_A)$ be an LR(¢,\$)-grammar. We will show how to construct a complete LR(¢,\$)-grammar $G_C = (G_A, G_R) = (N_A \cup N_R \cup \{S\}, \Sigma \cup \{¢, \$\}, S, R_A \cup R_R \cup \{S \to S_A, S \to S_R\})$ such that S and S_R are new nonterminals not contained in N_A, S_R is from the new set of nonterminals N_R. The construction utilizes the fact that for each word w from the complement of $L(G_A)$, LR(0) analyzer of G_A can detect the shortest prefix y of w such that each word of the form yu belongs to the complement of $L(G_A)$, where $w, y, v \in (N_A \cup \Sigma)^*$.

Let $\mathcal{T}$ be the set of tables of the LR(0) analyzer for G_A and $\tau_0 \in \mathcal{T}$ be the initial state (table) of the corresponding LR(0) automaton. Let $N_R = \mathcal{T} \cup \{E\}$, where E is a new nonterminal not contained in $N_A \cup N_R$. The set of rules R_R will contain rules $E \to aE|\$$, for all $a \in \Sigma$, for generating arbitrary suffixes of words from $\Sigma^* \cdot \{\$\}$. Based on the goto function g of the LR(0) analyzer for G_A, we add the following set of rules into R_R

$$\{\tau \to aE \mid \tau \in \mathcal{T}, a \in \Sigma \cup N, g(\tau, a) = error\} \cup \{\tau \to \$ \mid \tau \in \mathcal{T}, g(\tau, \$) = error\}. \tag{3}$$

Now it is easy to see that all words of the form ¢w\$, where $w \in \Sigma^*$, that are rejected by the LR(0) analyzer for G can be generated from the nonterminal τ_0. Hence, we set $S_R = \tau_0$. Note that we did not include rules with the left sentinel ¢ into the set defined in (3), because the complete grammar should generate only words of the form ¢w\$, for $w \in \Sigma^*$.

Additionally, the grammar $G_R = (N_A \cup N_R \cup \{S\}, \Sigma \cup \{¢, \$\}, S_R, R_A \cup R_R \cup \{S \to S_A, S \to S_R\})$ is an LR(¢,\$)-grammar, as the corresponding LR(0) analyzer for G_R can be obtained by modifying the LR(0) analyzer for G_A. □

Observe that the complete grammar constructed according to the above construction has further interesting properties:

1. For each word of the form ¢w\$, where $w \in \Sigma^*$, there is exactly one derivation tree T according to G_C. Under the root of T, there is a node labelled either by S_A or S_R. If it is S_A, the word is generated by the accepting grammar G_A. Otherwise, it is generated by the rejecting grammar G_R.
2. Let T be a derivation tree according to G_C. If a node d from T is labelled by a nonterminal from N_A, then all its descendant nodes are labelled only by symbols from N_A.
3. Let T be a derivation tree according to G_C. If a node d from T is labeled by a nonterminal from N_R, then all nodes on the path from d to the root of T (except the root itself) are labelled only by symbols from N_R.
4. The new rules added in the above construction enable only one-sided pumping patterns. Thus, if G_A is a one-sided LR(¢, \$)-grammar, then G_R and G_C have only one-sided core pumping patterns.

Definition 11. *Let $G = (N, \Sigma \cup \{¢, \$\}, S, R)$ be a complete LR(¢,\$)-grammar, G_A and G_R be its accepting and rejecting grammars. Let $u \in L_{In}(G_A)$, $\mathrm{AR}(G, u) = (u, u_1, u_2, \ldots, u_n)$, where $u \Leftarrow_{P(G,e)} u_1 \Leftarrow_{P(G,e)} u_2 \Leftarrow_{P(G,e)} \cdots \Leftarrow_{P(G,e)} u_n$, and there is not any z such that $u_n \Leftarrow_{P(G,e)} z$. We say that $\mathrm{AR}(G_A, u)$ is a pumping analysis (by reduction) of u by G_A and G as well.*

Let $u \in L_{In}(G_R)$, $\mathrm{AR}(G_R, u) = (u, u_1, \ldots, u_n)$, where $u \Leftarrow_{P(G,e)} u_1 \Leftarrow_{P(G,e)} u_2 \Leftarrow_{P(G,e)} \cdots \Leftarrow_{P(G,e)} u_n$, and there is not any z such that $u_n \Leftarrow_{P(G,e)} z$. We say that $\mathrm{AR}(G_R, u)$ is a pumping analysis by reduction of u by G_R and by G as well.

Let $u \in L_{In}(G_A)$. We take $\mathrm{AR}(G, u) = \mathrm{AR}(G_A, u)$.

Let $u \in L_{In}(G_R)$. We take $\mathrm{AR}(G, u) = \mathrm{AR}(G_R, u)$.

Let $\mathrm{AR}(G) = \{\mathrm{AR}(G, u) \mid u \in \Sigma^*\}$. We say that $\mathrm{AR}(G)$ is pumping analysis by reduction by G.

Let $\mathrm{AR}(\mathrm{A}, G) = \{\mathrm{AR}(G, u) | u \in L(G_A)\}$. We say that $\mathrm{AR}(\mathrm{A}, G)$ is accepting pumping analysis by reduction by G.

Let $\mathrm{AR}(\mathrm{R}, G) = \{\mathrm{AR}(G, u) | u \in L(G_R)\}$. We say that $\mathrm{AR}(\mathrm{R}, G)$ is rejecting pumping analysis by reduction by G.

4. Pumping RP-automata controlled by complete LR(¢,\$)-grammars.

In this section, we show that for any complete LR(¢,\$)-grammar obtained by the construction from the proof of Theorem 1, we can construct an RP-automaton with the same pumping analysis by reduction as G.

Theorem 2. *Let $G_C = (N, \Sigma, S, R)$ be a complete LR(¢,\$)-grammar with an accepting grammar $G_A = (N, \Sigma \cup \{¢, \$\}, S_A, R)$ and a rejecting grammar $G_R = (N, \Sigma \cup \{¢, \$\}, S_R, R)$.*

Then there exists a procedure that constructs an scf-det-mon-RP-automaton $M(G_C) = (Q, \Sigma, ¢, \$, q_0, k, \delta, Q_\mathrm{A}, Q_\mathrm{R})$ such that $\mathrm{AR}(M(G_C)) = \mathrm{AR}(G_C)$, $\mathrm{AR}(\mathrm{A}, M(G_C)) = \mathrm{AR}(\mathrm{A}, G_C)$, and $\mathrm{AR}(\mathrm{R}, M(G_C)) = \mathrm{AR}(\mathrm{R}, G_C)$.

Proof. The construction is based on the same idea as the construction of the det-mon-R-automaton M simulating a syntactic analysis of a deterministic context-free language L in [6]. There, an analysis by reduction of the automaton M simulated a syntactic analysis according to G. Here, we stress that $M(G_C)$ will perform pumping analysis by reduction simulating syntactic analysis by the complete LR(¢,\$) grammar G_C for any word over its input alphabet. More precisely, in the cycles of its computation, $M(G_C)$ performs a limited syntactic analysis by G_A and later possibly by G_R . By this construction, we directly obtain deterministic monotone restarting automaton in the strong cyclic form.

The second difference here is that we use det-mon-RP-automata instead of det-mon-R-automata. Let us note that each det-mon-R-automaton M can be easily converted into a det-mon-RP-automaton M' by splitting each restarting instruction of M into one preparing instruction and one restarting instruction of M'.

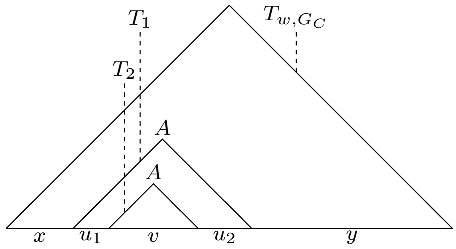

Figure 4: The structure of a derivation tree.

To see that the resulting det-mon-RP-automaton $M(G_C)$ performs pumping analysis by reduction by G_C, we sketch the construction of $M(G_C)$.

By simulating a pumping analysis by reduction by G_C on a word $w \in \{¢\} \cdot \Sigma^* \cdot \{\$\}$, we can construct the derivation tree T_{w,G_C} according to grammar G_C (the inner vertices of which are labeled with nonterminals and leaves correspond to terminal symbols).

Thus, for any word $w \in \{¢\} \cdot \Sigma^* \cdot \{\$\}$ there is exactly one derivation tree T_{w,G_C} by G_C. Similarly, as in the case of the standard pumping lemma for context-free languages, we can take $p = K_{G_C}$ such that, for any word w of length greater than p, there are (complete) subtrees T_1 and T_2 of T_{w,G_C} such that T_2 is a subtree of T_1 and the roots of both subtrees have the same label (cf. Fig. 4); in addition, T_2 has fewer leaves than T_1, T_1 has at most p leaves, and $|u_1u_2| > 0$.

Obviously, replacing T_1 with T_2, we get the derivation tree $T_{w(0)}$ for a shorter word $w(0)$ (if $w = xu_1vu_2y$ then $w(0) = xvy$).

The key to the construction of $M(G_C)$ is the possibility to identify the leftmost sub-word u_1vu_2 corresponding to sub-trees T_1 and T_2 by G_C, as shown in Fig. 4, when reading from left to right with the help of a constant size memory only. In its constant size memory, $M(G_C)$ stores all maximal sub-trees of the derivation tree(s) with all their leaves in the buffer. This is done by simulating the LR(0) analyzer for G_C. When it identifies the leftmost core pumping sub-tree like T_1 above, $M(G_C)$ deletes u_1 and u_2 by executing a single $RESTART$ operation. As the length of u_1vu_2 is at most p, a read/write window of length $k = 2p$ is sufficient for that.

If no such pumping sub-tree is built over the contents of the read/write window, the automaton $M(G_C)$ forgets the leftmost of these sub-trees with all its $n \geq 1$ leaves, and reads n new symbols to the right end of the buffer (performing MVR-instructions). Then $M(G_C)$ continues constructing the maximal sub-trees with all leaves in the (updated) buffer (again by simulating the LR(0) analyzer for G_C).

Short words of length less than k are accepted/rejected in tail computations.

It is not hard to see from the previous construction that $M(G_C)$ is an scf-det-mon-RP-automaton such that $\mathrm{AR}(M(G_C)) = \mathrm{AR}(G_C)$, $\mathrm{AR}(\mathrm{A}, M(G_C)) = \mathrm{AR}(\mathrm{A}, G_C)$, and $\mathrm{AR}(\mathrm{R}, M(G_C)) = \mathrm{AR}(\mathrm{R}, G_C)$.

The fact that $M(G_C)$ is deterministic and monotone follows from the construction of det-mon-R-automaton in [3]. The strong cyclic form of $M(G_C)$ follows from the fact that all words from $\{¢\} \cdot \Sigma^* \cdot \{\$\}$ are generated by G_C and all accepting computations of the LR(0) analyzer for G_C end with reducing the input into the initial nonterminal S, hence $M(G_C)$ accepts in tail computations with a tape contents of the length at most $K_{G_C} < k$. □

Definition 12. *Let $G_C = (G_A, G_R)$ be a complete LR($¢, \$$)-grammar with the corresponding accepting grammar G_A and rejecting grammar G_R. Let $M(G_C)$ be the scf-det-mon-RP-automaton constructed by the construction described in the proof of Theorem 2.*

We say that $M(G_C)$ is an RP-automaton with pumping analysis by reduction according to G_C, $M(G_C)$ is an RP(LRG(¢,\$))-automaton, and by $\mathcal{L}(RP(LRG(¢, \$)))$ we denote the class of all languages accepted by $\mathrm{RP}(LRG(¢, \$))$-automata. Additionally, we say that $L(G_R)$ is the rejecting language of $M(G_C)$, and we denote it as $L_R(M(G_C))$.

Corollary 1. *For any LR(¢,\$)-grammar G there exists a complete LR(¢,\$)-grammar $G_C = (G, G_R)$ and a deterministic monotone $\mathrm{RP}(LRG(¢, \$))$-automaton with a pumping analysis by reduction according to G_C such that $L_{In}(G) = L(M(G_C))$, $L_{In}(G_R) = L_R(M(G_C))$.*

Lemma 1. $\mathcal{L}(\text{det-mon-RP}) \subseteq \mathrm{DCFL}$.

Proof. As the models of det-mon-R- and det-mon-RP-automata differ only slightly, we can use here a slightly modified proof of Lemma 8 in [6] stating that $\mathcal{L}(\text{det-mon-R}) \subseteq \mathrm{DCFL}$. For given det-mon-RP-automaton M, a method from [6] can be used to construct a deterministic push-down automaton P that accepts the same language as M. □

Theorem 3. $\mathcal{L}(RP(LRG(¢, \$)) = DCFL = \mathcal{L}(\text{det-mon-RP}) = \mathcal{L}(\textit{scf}\text{-det-mon-RP})$

Proof. The theorem is a consequence of the previous lemma and the previous corollary. □

5. Conclusion and Future Work

In this paper, we have introduced complete LR(¢,\$)-grammars and restarting pumping RP(LRG(¢,\$))-automata. We have answered some basic questions concerning this type of automata and grammars. By simulating classical LR(0)-analysis for complete LR(¢,$)-grammars, RP(LRG(¢,$))-automata can perform pumping analysis by reduction for complete LR(¢,$)-grammars.

The constructions and results in this paper should enable to introduce and study regular and non-regular characteristics of two-sided pumping patterns of RP(LRG(¢,$))-automata and LRG(¢,$)-grammars and use such characteristics to prepare tools for localization of syntactic errors of general (and special) deterministic context-free languages. This way, we can extend and refine results from [7].

Acknowledgments

The research has been supported by the grant 1/0601/20 of the Slovak Scientific Grant Agency VEGA (Dana Pardubská) and the grant 19-21198S of the Czech Science Foundation (Daniel Průša).

References

[1] F. Mráz, D. Pardubská, M. Plátek, J. Šíma, Pumping deterministic monotone restarting automata and DCFL, in: M. Holena, T. Horváth, A. Kelemenová, F. Mráz, D. Pardubská, M. Plátek, P. Sosík (Eds.), Proceedings of the 20th Conference Information Technologies – Applications and Theory (ITAT 2020), volume 2718 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020, pp. 51–58. URL: http://ceur-ws.org/Vol-2718/paper13.pdf.

[2] M. Plátek, F. Mráz, D. Pardubská, D. Průša, J. Šíma, On separations of LR(0)-grammars by two types of pumping patterns, in: B. Brejová, L. Ciencialová, M. Holena, F. Mráz, D. Pardubská, M. Plátek, T. Vinar (Eds.), Proceedings of the 21st Conference Information Technologies – Applications and Theory (ITAT 2021), volume 2962 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 140–146. URL: http://ceur-ws.org/Vol-2962/paper05.pdf.

[3] P. Jančar, F. Mráz, M. Plátek, J. Vogel, On monotonic automata with a restart operation, J. Autom. Lang. Comb. 4 (1999) 287–311. doi:10.25596/jalc-1999-287.

[4] M. Plátek, F. Otto, F. Mráz, On h-lexicalized restarting list automata, J. Autom. Lang. Comb. 25 (2020) 201–234. URL: https://doi.org/10.25596/jalc-2020-201. doi:10.25596/jalc-2020-201.

[5] M. A. Harrison, Introduction to Formal Language Theory, Addison-Wesley, USA, 1978.

[6] P. Jančar, F. Mráz, M. Plátek, J. Vogel, Restarting automata, in: H. Reichel (Ed.), Fundamentals of Computation Theory, FCT '95, volume 965 of *Lecture Notes in Computer Science*, Springer, 1995, pp. 283–292. doi:10.1007/3-540-60249-6_60.

[7] M. Procházka, Redukční automaty a syntaktické chyby, Phd-thesis, Faculty of Mathematics and Physics, Charles University, Prague, 2011. In Czech.