S. Krajči (ed.): ITAT 2018 Proceedings, pp. 10–17
CEUR Workshop Proceedings Vol. 2203, ISSN 1613-0073, © 2018 Martin Plátek, Dana Pardubská, and František Mráz

On (In)Sensitivity by Two-Way Restarting Automata

Martin Plátek[1], Dana Pardubská[2]*, and František Mráz[1]

[1] Charles University, Department of Computer Science
Malostranské nám. 25, 118 00 PRAHA 1, Czech Republic
martin.platek@mff.cuni.cz, frantisek.mraz@mff.cuni.cz

[2] Comenius University in Bratislava, Department of Computer Science
Mlynská Dolina, 84248 Bratislava, Slovakia
pardubska@dcs.fmph.uniba.sk

Abstract: We study *h-lexicalized two-way restarting automaton* (hRLWW(i)) that can rewrite at most i times per cycle, for $i \geq 1$. This model is useful for the study of lexical syntactic disambiguation (a notion from linguistic) through the formal notion of *h-lexicalized syntactic analysis (hLSA)*. The hLSA is composed of a *basic language* and the corresponding *h-proper language* obtained from the basic language by mapping all non-input symbols on input symbols. We compare the properties of input languages, which are the languages traditionally considered in automata theory, to the properties of hLSA, i.e., to the properties of basic and h-proper languages.

The basic and h-proper languages of hRLWW(i)-automata fulfill the so called *reduction correctness preserving property*, but the input languages do not. While basic and h-proper languages are sensitive to the size of the read/write window, the input languages are not. Moreover, the basic and h-proper languages are sensitive to the number of rewrite steps per cycle. All that concerns a subclass of context-sensitive languages containing all context-free languages (and most probably also the class of mildly context-sensitive languages [5]), i.e., a class suitable for studying and classifying syntactic and semantic features of natural languages.

We work here also with the parametrized constraint of monotonicity. While using the monotonicity of degree one we can characterize the class of context-free languages, the monotonicity of higher degrees can model more complex syntactic phenomena of whole natural languages (like cross-serial dependencies [5]).

Finally, we stress the constraint of *weak cyclic form*. It preserves the power of hRLWW(i)-automata, and it allows to extend the complexity results obtained for the classes of infinite languages also into the classes of finite languages (parametrized by the number of performed cycles). It is useful for classifications in computational and corpus linguistics, where all the (syntactic) observation are of the finite nature. The main technical novelty of the paper are the results about the sensitivity and insensivity of finite and infinite hLSA and corresponding languages by hRLWW(i)-automata.

*The research is partially supported by VEGA 2/0165/16

1 Introduction

This paper is a continuation of conference papers [13], [14], and the technical report [15]. Its motivation is to study lexical syntactic disambiguation, which is one of the basic concepts of several linguistic schools, including the schools working with dependency syntax. In order to give a theoretical basis for lexicalized syntax, a model of a restarting automaton that formalizes lexicalization in a similar way as categorial grammars (see, e.g., [1]) – the *h-lexicalized restarting automaton* (hRLWW), was introduced in [13]. This model is obtained from the two-way restarting automaton of [12] by adding a letter-to-letter morphism h that assigns an input symbol to each working symbol. Then the *basic language* $L_C(M)$ of an hRLWW-automaton M consists of all words over the working alphabet of M that are accepted by M, and the *h-proper language* $L_{hP}(M)$ of M is obtained from $L_C(M)$ through the morphism h.

Further, the set of pairs $\{ (h(w), w) \mid w \in L_C(M) \}$, denoted as $L_A(M)$, is called the h-lexicalized syntactic analysis (hLSA) by M. Thus, in this setting the auxiliary symbols themselves play the role of the tagged items. That is, each auxiliary symbol b can be seen as a pair consisting of an input symbol $h(b)$ and some additional syntactico-semantic information (tags).

In contrast to the original hRLWW-automaton that uses exactly one rewrite in a cycle, here we study *h-lexicalized restarting automaton* (hRLWW(i)) allowing $i \geq 1$ rewrites in a cycle. Our effort is to show that this model is suited for a transparent and sensitive modeling of the lexicalized syntactic analysis (lexical disambiguation) of natural languages (compare to [8, 9]).

The lexicalized syntactic analysis based on analysis by reduction is traditionally used to analyze sentences of natural languages with a high degree of word-order freedom like, e.g., Czech, Latin, or German (see, e.g., [8]). Usually, a human reader is supposed to understand the meaning of a given sentence before he starts to analyze it; h-lexicalized syntactic analysis based on the analysis by reduction simulates such a behavior by analysis of sentences, where morphological and syntactical tags have been added to the word-forms and punctuation marks (see, e.g., [9]). We recall some constraints that are typical for restarting automata, and we outline ways for new combinations of con-

straints. We establish infinite (two-dimensional) ascending hierarchies of language classes of h-proper languages for several subtypes of hRLWW(i)-automata with respect to the number of allowed cycles, the number of symbols deleted during each cycle or the length of scanning window of an automaton.

The paper is structured as follows. In Section 2, we introduce our model and its sub-models, we define the h-proper languages, and we state the reduction correctness preserving property and the reduction error preserving property for the basic languages of deterministic h-RLWW-automata. In Section 3, we present the achieved results. The paper concludes with Section 4 in which we summarize our results and state some problems for future work.

2 Definitions

By $\subseteq$ and $\subset$ we denote the subset and the proper subset relation, respectively. Further, we will sometimes use regular expressions instead of the corresponding regular languages. Finally, throughout the paper λ will denote the empty word.

We start with the definition of the two-way restarting automaton as an extension to the original definition from [12]. In contrast to [14], we do not introduce general h-lexicalized two-way restarting list automaton which can rewrite arbitrary many times during each cycle. Instead, we introduce two-way restarting automata which can rewrite at most i times per cycle, for an integer $i \geq 1$.

Definition 1. *Let i be a positive integer. A* two-way restarting automaton*, an* RLWW*(i)-automaton for short, is a machine with a single flexible tape and a finite-state control. It is defined through an 9-tuple $M = (Q, \Sigma, \Gamma, \mathfrak{c}, \$, q_0, k, i, \delta)$, where Q is a finite set of states, Σ is a finite input alphabet, and $\Gamma (\supseteq \Sigma)$ is a finite working alphabet. The symbols from $\Gamma \setminus \Sigma$ are called* auxiliary symbols*. Further, the symbols $\mathfrak{c}, \$ \notin \Gamma$, called* sentinels*, are the markers for the left and right border of the workspace, respectively, $q_0 \in Q$ is the initial state, $k \geq 1$ is the size of the* read/write window*, $i \geq 1$ is the number of allowed rewrites in a cycle (see later), and*

$$\delta : Q \times \mathscr{PC}^{\leq k} \to \mathscr{P}((Q \times \{\mathsf{MVR}, \mathsf{MVL}, \mathsf{W}(y), \mathsf{SL}(v)\}) \cup \{\mathsf{Restart}, \mathsf{Accept}, \mathsf{Reject}\})$$

is the transition relation*. Here $\mathscr{P}(S)$ denotes the powerset of a set S,*

$$\mathscr{PC}^{\leq k} = (\mathfrak{c} \cdot \Gamma^{k-1}) \cup \Gamma^k \cup (\Gamma^{\leq k-1} \cdot \$) \cup (\mathfrak{c} \cdot \Gamma^{\leq k-2} \cdot \$)$$

is the set of possible contents *of the read/write window of M, $v \in \mathscr{PC}^{\leq k-1}$, and $y \in \mathscr{PC}^{\leq k}$.*

Being in a state $q \in Q$ and seeing $u \in \mathscr{PC}^{\leq k}$ in its window, the automaton can perform seven different types of transition steps (or instructions):

1. *A* move-right step $(q,u) \longrightarrow (q', \mathsf{MVR})$ *assumes that $(q', \mathsf{MVR}) \in \delta(q,u)$, where $q' \in Q$ and $u \notin \{\lambda, \mathfrak{c}\} \cdot \Gamma^{\leq k-1} \cdot \$$. This move-right step causes M to shift the window one position to the right and to enter state q'.*

2. *A* move-left step $(q,u) \longrightarrow (q', \mathsf{MVL})$ *assumes that $(q', \mathsf{MVL}) \in \delta(q,u)$, where $q' \in Q$ and $u \notin \mathfrak{c} \cdot \Gamma^* \cdot \{\lambda, \$\}$. It causes M to shift the window one position to the left and to enter state q'.*

3. *An* SL-step $(q,u) \longrightarrow (q', \mathsf{SL}(v))$ *assumes that $(q', \mathsf{SL}(v)) \in \delta(q,u)$, where $q' \in Q$, $v \in \mathscr{PC}^{\leq k-1}$, v is shorter than u, and v contains all the sentinels that occur in u (if any). It causes M to replace u by v, to enter state q', and to shift the window by $|u| - |v|$ items to the left – but at most to the left sentinel $\mathfrak{c}$ (that is, the contents of the window is 'completed' from the left, and so the distance to the left sentinel decreases, if the window was not already at $\mathfrak{c}$).*

4. *A* W-step $(q,u) \longrightarrow (q', \mathsf{W}(v))$ *assumes that $(q', \mathsf{W}(v)) \in \delta(q,u)$, where $q' \in Q$, $v \in \mathscr{PC}^{\leq k}$, $|v| = |u|$, and that v contains all the sentinels that occur in u (if any). It causes M to replace u by v, and to enter state q' without moving its window.*

5. *A* restart step $(q,u) \longrightarrow \mathsf{Restart}$ *assumes that* $\mathsf{Restart} \in \delta(q,u)$*. It causes M to place its window at the left end of its tape, so that the first symbol it sees is the left sentinel $\mathfrak{c}$, and to reenter the initial state q_0.*

6. *An* accept step $(q,u) \longrightarrow \mathsf{Accept}$ *assumes that* $\mathsf{Accept} \in \delta(q,u)$*. It causes M to halt and accept.*

7. *A* reject step $(q,u) \longrightarrow \mathsf{Reject}$ *assumes that* $\mathsf{Reject} \in \delta(q,u)$*. It causes M to halt and reject.*

A *configuration* of an RLWW(i)-automaton M is a word $\alpha q \beta$, where $q \in Q$, and either $\alpha = \lambda$ and $\beta \in \{\mathfrak{c}\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{\mathfrak{c}\} \cdot \Gamma^*$ and $\beta \in \Gamma^* \cdot \{\$\}$; 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 of β if $|\beta| < k$. A *restarting configuration* is of the form $q_0 \mathfrak{c} w \$$, where $w \in \Gamma^*$; if $w \in \Sigma^*$, then $q_0 \mathfrak{c} w \$$ is an *initial configuration*. We see that any initial configuration is also a restarting configuration, and that any restart transfers M into a restarting configuration.

In general, an RLWW(i)-automaton M is *nondeterministic*, that is, there can be two or more steps (instructions) with the same left-hand side (q,u), and thus, there can be more than one computation that start from a given restarting configuration. If this is not the case, the automaton is *deterministic*.

A *computation* of M is a sequence $C = C_0, C_1, \ldots, C_j$ of configurations of M, where C_0 is an initial or a restarting configuration and C_{i+1} is obtained from C_i by a step of M, for all $0 \leq i < j$. In the following we only consider computations of RLWW(i)-automata which are finite and end either by an accept or by a reject step.

Cycles and tails: Any finite computation of an RLWW(i)-automaton M consists of certain phases. A phase, called a *cycle*, starts in a restarting configuration, the window moves along the tape performing non-restarting steps until a restart step is performed and thus a new restarting configuration is reached. If no further restart step is performed, any finite computation necessarily finishes in a halting configuration – such a phase is called a *tail*. It is required that in each cycle RLWW(i)-automaton executes at most i rewrite steps (of type W or SL) but at least one SL-step. Moreover, it must not execute any rewrite step in a tail.

This induces the following relation of *cycle-rewriting* by M: $u \Rightarrow_M^c v$ iff there is a cycle that begins with the restarting configuration $q_0 \¢ u \$$ and ends with the restarting configuration $q_0 \¢ v \$$. The relation $\Rightarrow_M^{c^*}$ is the reflexive and transitive closure of $\Rightarrow_M^c$. We stress that the cycle-rewriting is a very important feature of an RLWW(i)-automaton. As each SL-step is strictly length-reducing, we see that $u \Rightarrow_M^c v$ implies that $|u| > |v|$. Accordingly, $u \Rightarrow_M^c v$ is also called a *reduction* by M.

An *input word* $w \in \Sigma^*$ *is accepted by* M, if there is a computation which starts with the initial configuration $q_0 \¢ w \$$ and ends by executing an accept step. By $L(M)$ we denote the language consisting of all input words accepted by M; we say that M *recognizes (or accepts) the input language* $L(M)$.

A *basic (or characteristic) word* $w \in \Gamma^*$ is accepted by M if there is a computation which starts with the restarting configuration $q_0 \¢ w \$$ and ends by executing an accept step. By $L_C(M)$ we denote the set of all words from Γ^* that are accepted by M; we say that M *recognizes (or accepts) the basic (or characteristic*[1]*) language* L_C.

Finally, we come to the definition of the central notion of this paper, the h-lexicalized RLWW(i)-automaton.

Definition 2. *Let i be a positive integer. An* h-lexicalized RLWW(i)-automaton*, or an* hRLWW(i)-automaton*, is a pair $\widehat{M} = (M,h)$, where $M = (Q,\Sigma,\Gamma,\¢,\$,q_0,k,i,\delta)$ is an* RLWW(i)*-automaton and $h : \Gamma \to \Sigma$ is a letter-to-letter morphism satisfying $h(a) = a$ for all input letters $a \in \Sigma$. The input language $L(\widehat{M})$ of $\widehat{M}$ is simply the language $L(M)$ and the basic language $L_C(\widehat{M})$ of $\widehat{M}$ is the language $L_C(M)$. Finally, we take $L_{hP}(\widehat{M}) = h(L_C(M))$, and we say that $\widehat{M}$* recognizes (or accepts) the h-proper language *$L_{hP}(\widehat{M})$.*

Finally, the set $L_A(\widehat{M}) = \{ (h(w),w) \mid w \in L_C(M) \}$ is called the h-lexicalized syntactic analysis (shortly hLSA) by $\widehat{M}$.

We say that for $x \in \Sigma^*$, $L_A(\widehat{M},x) = \{ (x,y) \mid y \in L_C(M), h(y) = x \}$ is the h-syntactic analysis (lexicalized syntactic analysis) *for x by $\widehat{M}$. We see that $L_A(\widehat{M},x)$ is non-empty only for x from $L_{hP}(\widehat{M})$.*

[1] Basic languages were called characteristic languages in [10] and several other papers, therefore, here we preserve the original notation with subscript C.

Evidently, for an hRLWW(i)-automaton $\widehat{M}$, we have $L(\widehat{M}) \subseteq L_{hP}(\widehat{M}) = h(L_C(\widehat{M}))$.

Let us note that h-lexicalized syntactic analysis formalizes the linguistic notion of *lexical disambiguation*. Each auxiliary symbol $x \in \Gamma \setminus \Sigma$ of a word from $L_C(\widehat{M})$ can be considered as a disambiguated input symbol $h(x)$.

The following two facts ensure the transparency for computations of hRLWW(i)-automata with respect to their basic and h-proper languages.

Fact 1. (Reduction Error Preserving Property) *Let M be an* hRLWW*(i)-automaton. If $u \Rightarrow_M^{c^*} v$ and $u \notin L_C(M)$, then $v \notin L_C(M)$.*

Fact 2. (Reduction Correctness Preserving Property) *Let M be a deterministic* hRLWW*(i)-automaton. If $u \Rightarrow_M^{c^*} v$ and $u \in L_C(M)$, then $v \in L_C(M)$, and $h(v) \in L_{hP}(M)$.*

Notations. For brevity, the prefix det- will be used to denote the property of being deterministic. For any class A of automata, $\mathscr{L}(\mathsf{A})$ will denote the class of input languages that are recognized by automata from A, $\mathscr{L}_C(\mathsf{A})$ will denote the class of basic languages that are recognized by automata from A, and $\mathscr{L}_{hP}(\mathsf{A})$ will denote the class of h-proper languages that are recognized by automata from A. $\mathscr{L}_A(\mathsf{A})$ will denote the class of hLSA (h-lexicalized syntactic analyses) that are defined by automata from A. For a natural number $k \geq 1$, $\mathscr{L}(k\text{-}\mathsf{A})$, $\mathscr{L}_C(k\text{-}\mathsf{A})$, $\mathscr{L}_{hP}(k\text{-}\mathsf{A})$, and $\mathscr{L}_A(k\text{-}\mathsf{A})$ will denote the class of input, basic, h-proper languages, and hLSA's, respectively, that are recognized by those automata from A that use a read/write window of size k.

2.1 Further Refinements and Constraints on RLWW-Automata (hRLWW-Automata)

Here we introduce some constrained types of rewrite steps whose introduction is motivated by different type of linguistic reductions.

A *delete-left step* $(q,u) \to (q', \mathsf{DL}(v))$ is a special type of an SL-step $(q', \mathsf{SL}(v)) \in \delta(q,u)$, where v is a proper (scattered) subsequence of u, containing all the sentinels from u (if any). It causes M to replace u by v (by deleting excessive symbols), to enter state q', and to shift the window by $|u| - |v|$ symbols to the left, but at most to the left sentinel $\¢$.

A *contextual-left step* $(q,u) \to (q', \mathsf{CL}(v))$ is a special type of DL-step $(q', \mathsf{DL}(v)) \in \delta(q,u)$, where $u = v_1u_1v_2u_2v_3$ and $v = v_1v_2v_3$ such that v contains all the sentinels from u (if any). It causes M to replace u by v (by deleting the factors u_1 and u_2 of u), to enter state q', and to shift the window by $|u| - |v|$ symbols to the left, but at most to the left sentinel $\¢$.

An RLWW(i)-automaton is called RLW(i)-automaton if its working alphabet coincides with its input alphabet, that is, no auxiliary symbols are available for this automaton. Note that in this situation, each restarting configuration is

necessarily an initial configuration. Within an abbreviation for an automata type, R denotes the use of moves to the right, L denotes the use of moves to the left, WW denotes the use of both input and working alphabets, and single W denotes the use of input alphabet only (the working alphabet coincides with the input alphabet).

An RLW(i)-automaton is called RLWD*(i)-automaton* if all its rewrite steps are DL steps, and it is an RLWC*(i)*-automaton if all its rewrite steps are CL-steps. Further, an RLWW(i)-automaton is called RLWWC*(i)-automaton* if all its rewrite steps are CL-steps. Similarly, an RLWW(i)-automaton is called RLWWD*(i)-automaton* if all its rewrite steps are DL-steps. Observe that when concentrating on input languages, DL- and CL-steps ensure that no auxiliary symbols can ever occur on the tape; if, however, we are interested in basic or h-proper languages, then auxiliary symbols can play an important role even though a given RLWW(i)-automaton uses only DL- or CL-steps. Therefore, we distinguish between RLWWC(i)- and RLWC(i)-automata, and between RLWWD(i)- and RLWD(i)-automata.

In the following we will use the corresponding notations also for subclasses of RLWW(i)-automata, and hRLWW(i)-automata.

Evidently, we need not distinguish between hRLW(i)-automata and RLW(i)-automata, since for the RLW(i)-automata the only possible morphism h is the identity.

Fact 3. (Equality of Languages for RLW(i)-automata.) *For any RLW(i)-automaton M, $L(M) = L_{\mathrm{C}}(M) = L_{\mathrm{hP}}(M)$.*

We recall the notion of *monotonicity* (see [3, 4]) as an important constraint for computations of RLWW(i)-automata. Let M be an RLWW(i)-automaton, and let $C = C_k, C_{k+1}, \ldots, C_j$ be a sequence of configurations of M, where $C_{\ell+1}$ is obtained by a single transition step from C_ℓ, $k \leq \ell < j$. We say that C is a *subcomputation* of M. If $C_\ell = \mathrm{¢}\alpha q \beta \$$, then $|\beta\$|$ is the *right distance* of C_ℓ, which is denoted by $D_{\mathrm{r}}(C_\ell)$. We say that a sub-sequence $(C_{\ell_1}, C_{\ell_2}, \ldots, C_{\ell_n})$ of C is *monotone* if $D_{\mathrm{r}}(C_{\ell_1}) \geq D_{\mathrm{r}}(C_{\ell_2}) \geq \cdots \geq D_{\mathrm{r}}(C_{\ell_n})$. A computation of M is called *monotone* if the corresponding sub-sequence of rewrite configurations is monotone. Here a configuration is called a *rewrite configuration* if in this configuration an SL- or W-step is being applied. Finally, M itself is called *monotone* if each of its computations is monotone. We use the prefix mon- to denote monotone types of RLWW(i)-automata. This notion of monotonicity has been considered before in various papers (see [7]) similarly as its following generalization.

Naturally, a mon-RLWW(i)-automaton can be used to simulate bottom-up one-pass parsers. In order to simulate also bottom-up multi-pass parsers, a j-monotonicity for restarting automata was introduced in [12]. For an integer $j \geq 1$, a RLWW(i)- automaton is called j-monotone if, for each of its computations, the corresponding sequence of rewriting configurations can be partitioned into at most j sub-sequences such that each of these sub-sequences is monotone. We use the prefix mon(j)- to denote j-monotone types of RLWW(i)-automata.

Here we transfer some restricted form of restarting automata called *weak cyclic form* (wcf) (see [3]) over to RLWW(i)'s. An RLWW M is said to be in *weak cyclic form* if $|uv| \leq k$ for each accepting configuration $\mathrm{¢}uqv\$$ of M, where k is the size of the read/write window of M. Thus, before M can accept, it must erase sufficiently many letters from its tape. The prefix wcf- will be used to denote restarting automata in the weak cyclic form.

3 Results

3.1 On the Insensitivity of Input Languages

The following characterizations, which are slight extensions of known results, show that with respect to their input languages, (monotone) RLWW(1)-automata are insensitive to the size of their read/write windows. In the following CFL is the class of *context-free languages*.

Theorem 4. *For all $k \geq 3$, the following equalities hold:*

(a) $\mathsf{CFL} = \mathscr{L}(\text{mon-RLWW}(1))$,
(b) $\mathsf{CFL} = \mathscr{L}(k\text{-wcf-mon-RLWW}(1))$,
(c) $\mathscr{L}(\text{RLWW}(1)) = \mathscr{L}(k\text{-wcf-RLWW}(1))$.

Proof. It follows from [7] that $\mathsf{CFL} = \mathscr{L}(k\text{-mon-RLWW}(1))$ for all $k \geq 3$, and the equality $\mathscr{L}(\text{RLWW}) = \mathscr{L}(k\text{-RLWW}(1))$ follows from [16].

It remains to prove that each (monotone) RLWW(1)-automaton with a window of size three can be converted into weak cyclic form.

Transformation into weak cyclic form for input languages. So let M_1 be an RLWW(1)-automaton with window size three. A wcf-RLWW(1)-automaton M_2 can simulate M_1 as follows: M_2 uses a new non-input symbol # to indicate that M_1 halts with accepting. At the beginning of each cycle M_2 checks the symbol in front of the right sentinel \$. If it is # there, M_2 simply deletes the symbol in front of # and restarts, except when # is the only symbol on the tape in which case it accepts. If there is no # on the tape, M_2 simulates M_1 step by step until M_1 is about to halt. If M_1 rejects, then so does M_2. If, however, M_1 accepts, then M_2 either accepts if the tape contents is of length at most three, or, in case of longer tape contents, instead of accepting, M_2 moves its window to the right, rewrites the last two symbols in front of the right sentinel \$ by the special symbol # and restarts.

It is easily seen that M_2 is monotone, if M_1 is, that it has window size three, and that it accepts the same input language as M_1. □

Theorem 5. *For all $k, k', j \geq 1$, $X \in \{\lambda, wcf\}$:*

(a) $\mathscr{L}(k\text{-X-RLWW}(j)) \subseteq \mathscr{L}(k'\text{-X-RLWW}(j'))$, *for all* $j' \geq \lceil \frac{k}{k'} \rceil \cdot j$.

(b) $\mathsf{CFL} \subset \mathscr{L}(2\text{-X-RLWW}(j))$.

Proof. Case (a) follows from an easy observation that a single rewrite operation using a window of size k can be simulated by at most $\lceil \frac{k}{k'} \rceil$ rewrite operations with a window of size k'. To prove (b) we show that 2-wcf-RLWW(1)-automata accept all context-free languages and give a non-context-free language accepted by 2-wcf-RLWW(1)-automaton.

Given any context-free language L, Niemann and Otto in [11] showed how to construct a monotone one-way restarting automaton with window size 3 recognizing L. As the constructed automaton rewrites at most 2 consecutive symbols in a cycle and RLWW(1)-automaton can move in both directions, the constructed automaton can be simulated by a monotone 2-RLWW(1)-automaton. This implies $\mathsf{CFL} \subseteq \mathscr{L}(\text{2-X-RLWW}(j))$. To complete the proof of (b) it remains to show that the inclusion is proper. We will show that the non-context-free language $L = \{a^n b^{2n} c^n \mid n \geq 0\}$ can be accepted by a 2-RLWW(1)-automaton M.

The automaton M will use two auxiliary symbols X, Y whose role is to indicate deleted subwords ab and bc, respectively. Within four cycles the automaton deletes one occurrence of a and c and two symbols b. To do so M scans the contents ω of the tape within the sentinels from left to right and with the right sentinel in its window M distinguishes six situations:

1. if $\omega = \lambda$ then M accepts;
2. if $\omega = a^+b^+c^+$, then M rewrites ab with X and restarts;
3. if $\omega = a^*Xb^+c^+$, then M rewrites bc with Y and restarts;
4. if $\omega = a^*Xb^*Yc^*$, then M deletes X and restarts;
5. if $\omega = a^*b^*Yc^*$, then M deletes Y and restarts;
6. finally, in all other situations M rejects.

Obviously, M iteratively repeats a series of four cycles and then accepts/rejects in a tail computation. From the above description, $L = L(M)$ and $L \in \mathscr{L}(\text{2-RLWW}(1)) \setminus \mathsf{CFL}$ follow easily. □

The previous theorem witnesses the insensitivity of input languages of RLWW(i)-automata with respect to the size of look-ahead window.

3.2 hRLWW(i)-Automata and h-Lexicalized Syntactic Disambiguation

In the following we will study basic and h-proper languages of hRLWW(i)-automata, and we will see that with respect to these languages, hRLWW(i)-automata (and their variants) are sensitive to the window size, number of SL-operations in a cycle, etc.

The reformulated basic result from [13] follows.

Theorem 6. *The following equalities hold:*

$$\begin{aligned} \mathsf{CFL} &= \mathscr{L}_{hP}(\text{mon-RLWW}(1)) \\ &= \mathscr{L}_{hP}(\text{det-mon-RLWW}(1)) \\ &= \mathscr{L}_{hP}(\text{det-mon-RLWWD}(1)) \\ &= \mathscr{L}_{hP}(\text{det-mon-RLWWC}(1)). \end{aligned}$$

The previous theorem witnesses that for any context-free language there is a deterministic monotone analyzer which accepts the language when each input word is completely lexically disambiguated.

3.3 Weak Cyclic Form for Basic Languages

Theorem 7. *Let $i \geq 1$. For each* RLWW*(i)-automaton M, there is a* wcf-RLWW*(i)-automaton M_{wcf} such that $L_{\mathrm{C}}(M) = L_{\mathrm{C}}(M_{\mathrm{wcf}})$, and $u \Rightarrow_M^{c^*} v$ implies $u \Rightarrow_{M_{\mathrm{wcf}}}^{c^*} v$, for all words u, v. Moreover, the added reductions are in contextual form. If M is deterministic, j-monotone or simultaneously deterministic and j-monotone, for some $j \geq 1$, then M_{wcf} is deterministic, j-monotone or simultaneously deterministic and j-monotone, respectively.*

Proof. **Transformation into wcf for basic languages.** Let M be an RLWW(i)-automaton. We describe how M can be converted into a wcf-RLWW(i)-automaton M_{wcf} with the basic language $L_{\mathrm{C}}(M_{\mathrm{wcf}}) = L_{\mathrm{C}}(M)$. Let us assume that the size of the window of M is k. It is easy to see that the language L_T accepted by M in tail computations is a regular sub-language of $L_{\mathrm{C}}(M)$. This means that there exists a deterministic finite automaton A_T such that $L(A_T) = L_T$. Assume that A_T has p states. For M_{wcf} we now take a window of size $k_{\mathrm{wcf}} = \max\{k, p+1\}$. M_{wcf} executes all cycles (reductions) of M just as M. However, the accepting tail computations of M are replaced by computations of M_{wcf} that work in the following way:

(1) Any word $w \in L_{\mathrm{C}}(M)$ satisfying $|w| \leq k_{\mathrm{wcf}}$ is immediately accepted.

(2) On any word w satisfying $|w| > k_{\mathrm{wcf}}$, M_{wcf} executes a cycle that works as follows: the window is moved to the right until the right sentinel \$ appears in the window. From the pumping lemma for regular languages we know that if $w \in L_T$, then there exists a factorization $w = xyz$ such that $|y| > 0$, $|y| + |z| \leq p$, and $xz \in L_T$. Accordingly, M_{wcf} deletes the factor y and restarts.

From the construction above we immediately see that $L_{\mathrm{C}}(M_{\mathrm{wcf}}) = L_{\mathrm{C}}(M)$. According to the definition of an RLWW(i)-automaton, the automaton M cannot rewrite during tail computations. Therefore, if M is deterministic then M_{wcf} is deterministic. Additionally, if M is j-monotone, then M_{wcf} is j-monotone, too, as the property of j-monotonicity is not disturbed by the delete operations at the very right end of the tape that are executed at the end of a computation. Moreover, all added reductions are in contextual form. □

This enables to strengthen Theorem 6 by requiring automata in weak cyclic form only.

Corollary 1. *For all $X \in \{\text{RLWW}(1), \text{RLWWD}(1), \text{RLWWC}(1)\}$, the following equalities hold:*

$$\mathsf{CFL} = \mathscr{L}_{hP}(\text{mon-wcf-}X) = \mathscr{L}_{hP}(\text{det-mon-wcf-}X).$$

Hence, we can require that analyzers for context-free languages are not only monotone, but they also should accept without restart short words only.

3.4 On Sensitivity of hLSA by hRLWW(i)-Automata

Here we will recall that for hRLWW(1)-automata in weak cyclic form, there are strict hierarchies of classes of finite and infinite basic and h-proper languages that are based on the window size and on the number of rewrites in one cycle (one reduction) the automata are allowed to execute in accepting computations. As the main new results we will show similar results for hRLWW(j)-automata.

First, however, we need to introduce some additional notions. For a type X of RLWW(j)-automata, we denote the subclass of X-automata which perform at most i reductions in any computation as fin(i)-X-automata, and by fin-X, we denote those X-automata that are of type fin(i)-X for some $i \geq 0$.

For any hRLWW(j)-automaton M, we use $L_{\mathrm{C}}(M,i)$ to denote the subset of $L_{\mathrm{C}}(M)$ that consists of all words that M accepts by computations that contain at most i reductions, and we take $L_{\mathrm{hP}}(M,i) = h(L_{\mathrm{C}}(M,i))$.

Proposition 8. *Let $i \geq 0$, $j \geq 1$ and let M be a* wcf-hRLWW(j)*-automaton. Then there exists a* fin(i)-wcf-hRLWW(j)*-automaton M_i such that $L_{\mathrm{C}}(M_i) = L_{\mathrm{C}}(M,i)$, M_i has the same window size as M, and if $u \Rightarrow_M^c v$ and $v \in L_{\mathrm{C}}(M,i-1)$, then $u \Rightarrow_{M_i}^c v$. In addition, if M is deterministic, then so is M_i.*

Proof. Obviously, for an arbitrary $i \geq 0$, $j,k \geq 1$ and k-fin(i)-wcf-hRLWW(j)-automaton M, the language $L_{\mathrm{C}}(M,i)$ is finite. Hence, it is accepted by a finite automaton A_i. Automaton M_i cannot simply accept in tail computations all words from $L_{\mathrm{C}}(M,i)$, as the words can be of length greater than k. Therefore, on input w, automaton M_i first simulates A_i. If A_i accepts, then let $v_1,\ldots,v_n$ be all words from $L_{\mathrm{C}}(M,i-1)$ such that $w \Rightarrow_M^c v_\ell$, for all $\ell, 1 \leq \ell \leq n$. Then M_i nondeterministically selects some ℓ_0 between 1 and n, and executes a cycle that rewrites w into v_{ℓ_0}. Automaton M_i must be able to execute a cycle $w \Rightarrow_{m_i}^c v_\ell$, for all $\ell, 1 \leq \ell \leq n$. This is possible, as both $L_{\mathrm{C}}(M,i)$ and $L_{\mathrm{C}}(M,i-1)$ are finite languages.

If M is deterministic, then $n = 1$ and M_i is deterministic, too. □

For a positive integer k, we will use the prefix de(k)- to denote those hRLWW-automata for which each reduction shortens the word on the tape by at most k symbols.

From the previous ITAT-contribution [14] we have the following hierarchies.

Theorem 9. *For all types $X \in \{$*hRLW(1), hRLWD(1), hRLWC(1), hRLWW(1), hRLWWD(1), hRLWWC(1)*$\}$, all prefixes $pr_1 \in \{\lambda, \mathsf{fin}(i), \mathsf{fin}\}$, where $i \geq 1$, all prefixes $pref_X \in \{$*wcf, mon-wcf, det-wcf, det-mon-wcf*$\}$, and all $k \geq 2$, we have the following proper inclusions:*

(a) $\mathscr{L}_{\mathrm{hP}}(k\text{-}pr_1\text{-}pref_X\text{-}X) \subset \mathscr{L}_{\mathrm{hP}}((k+1)\text{-}pr_1\text{-}pref_X\text{-}X)$,
(b) $\mathscr{L}_{\mathrm{hP}}(\mathsf{de}(k)\text{-}pr_1\text{-}pref_X\text{-}X) \subset \mathscr{L}_{\mathrm{hP}}(\mathsf{de}(k+1)\text{-}pr_1\text{-}pref_X\text{-}X)$,
(c) $\mathscr{L}_{\mathrm{A}}(k\text{-}pr_1\text{-}pref_X\text{-}X) \subset \mathscr{L}_{\mathrm{A}}((k+1)\text{-}pr_1\text{-}pref_X\text{-}X)$,
(d) $\mathscr{L}_{\mathrm{A}}(\mathsf{de}(k)\text{-}pr_1\text{-}pref_X\text{-}X) \subset \mathscr{L}_{\mathrm{A}}(\mathsf{de}(k+1)\text{-}pr_1\text{-}pref_X\text{-}X)$.

3.5 On Sensitivity of LSA by hRLWW(i)-Automata

As a simple consequence of Theorem 6 and the fact that the RLWW(1)-automaton M from the proof of Theorem 5(b) is actually deterministic we obtain the following corollary.

Corollary 2. *For all $i > 1$, $pref \in \{\lambda,$ det, wcf, det-wcf$\}$ and all $X \in \{$*hRLWW(i), hRLWWD(i), hRLWWC(i)*$\}$ the following holds:*

$$\mathsf{CFL} \subset \mathscr{L}_{hP}(pref\text{-}X).$$

The previous corollary and the following results strongly support the idea that hRLWW(i)-automata are strong and fine enough to cover and classify the complexity of the (surface and deep) syntactic features of natural languages such as subordination (dependency), valency, coordination etc.

Lemma 1. *For all $j,k \geq 1$ it holds the following:*

(1) $\mathscr{L}(k\text{-det-mon-fin}(1)\text{-wcf-RLWC}(j+1)) \setminus \mathscr{L}_{hP}(\text{k-wcf-hRLWW}(j)) \neq \emptyset$,

(2) $\mathscr{L}((k+1)\text{-det-mon-fin}(1)\text{-wcf-RLWC}(j)) \setminus \mathscr{L}_{hP}(\text{k-wcf-hRLWW}(j)) \neq \emptyset$.

Proof. To be able to show the lower bound we need a language containing word(s) that are longer than the window size.

For $r \geq 1$, $s \geq 0$ let $L^{(r,s)} = \{b, a^{(r-1)s}b^{s+1}\}$. In order to show $L^{(r,s)} \in \mathscr{L}(r\text{-det-mon-fin}(1)\text{-wcf-RLWC}(s))$ consider the mon-RLWW(s) automaton $M^{(r,s)}$ which on its left-to-right turn distinguishes three situations:

1. if input word is b, then $M^{(r,s)}$ accepts;
2. if input word is not from $L^{(r,s)}$, then $M^{(r,s)}$ rejects;
3. if input word is $a^{(r-1)s}b^{s+1}$, the automaton applies s CL operations each of which deletes $a^{r-1}b$; after that it restarts and accepts b in the next tail computation.

Realize that hRLWW(s) automaton with window size r can delete at most rs symbols in any cycle with at most s rewrite (W- or SL-) operations. Since $|a^{(r-1)s}b^{s+1}| - |b| = rs$, neither wcf-hRLWW($s$) automaton with window size $r-1$ nor wcf-hRLWW($s-1$) automaton with window size r is able to recognize $L^{(r,s)}$ as its basic or h-proper language.

Now, the languages $L^{(k,j+1)}$ and $L^{(k+1,j)}$ can be used to prove (1) and (2), respectively. □

Next we show a similar hierarchy with respect to the degree on monotonicity which is related to the number of rewritings in a cycle. The degree of monotonicity is an important constraint which serves for better approximation of syntax of natural languages, and their individual features.

Lemma 2. *For all $j,k \geq 1$ it holds the following:*

(1) $\mathscr{L}(k\text{-det-mon}(j+1)\text{-wcf-RLWWC}(j+1)) \setminus \mathscr{L}_{\mathrm{hP}}(k\text{-wcf-RLWW}(j)) \neq \emptyset$;

(2) $\mathscr{L}(k\text{-det-mon}(j+1)\text{-wcf-RLWWC}(j+1)) \setminus \mathscr{L}_{\mathrm{hP}}(k\text{-mon}(j)\text{-wcf-RLWW}(j+1)) \neq \emptyset$.

Proof. We provide a parametrized sequence of finite languages $\left\{L_{\mathrm{m}}^{(k,j)}\right\}_{j,k=1}^{\infty}$, which satisfy

(a) $L_{\mathrm{m}}^{(k,j+1)} \in \mathscr{L}(k\text{-det-mon}(j+1)\text{-wcf-RLWC}(j+1))$,
(b) $L_{\mathrm{m}}^{(k,j+1)} \notin \mathscr{L}_{\mathrm{hP}}(k\text{-wcf-hRLWW}(j))$, and
(c) $L_{\mathrm{m}}^{(k,j+1)} \notin \mathscr{L}_{\mathrm{hP}}(k\text{-mon}(j)\text{-wcf-RLWW}(j+1))$.

Let $L_{\mathrm{m}}^{(k,j)} = \{(c^k d^k e^{k(j+1)})^j, (d^k e^{k(j+1)})^j\} \cup \{e^{krj} \mid 0 \leq r \leq j+1\}$. We can construct a k-det-mon(j)-RLWC(j)-automaton $M_{\mathrm{m}}^{(k,j)}$ accepting $L_{\mathrm{m}}^{(k,j)}$ as its input (and basic) language. The automaton scans the word on its tape and distinquishes the following cases:

1. if the word is of the form $(c^k d^k e^{k(j+1)})^j$, then $M_{\mathrm{m}}^{(k,j)}$ deletes j blocks of c^k and restarts;
2. if the word is of the form $(d^k e^{k(j+1)})^j$, then $M_{\mathrm{m}}^{(k,j)}$ deletes j blocks of d^k and restarts;
3. if the word is of the form e^{krj}, for some $r, 1 \leq r \leq j+1$, then $M_{\mathrm{m}}^{(k,j)}$ deletes j blocks of e^k and restarts;
4. if the word is empty, then $M_{\mathrm{m}}^{(k,j)}$ accepts, and
5. otherwise, $M_{\mathrm{m}}^{(k,j)}$ rejects.

It is not hard to partition the right distances of the rewriting configurations from an accepting computation of $M_{\mathrm{m}}^{(k,j)}$ into at most j monotone sequences. The highest degree j of monotonicity is necessary for accepting the input word $(c^k d^k e^{k(j+1)})^j$, which is accepted by $M_{\mathrm{m}}^{(k,j)}$ after performing $j+3$ reductions.

For reasons similar to that ones used in the proof of Lemma 1, the language $L_{\mathrm{m}}^{(k,j)}$ cannot be accepted neither as an h-proper language of a k-wcf-RLWW$(j'-1)$-automaton for any $j' < j$ nor as an h-proper language of a k-mon(j')-wcf-RLWW(j)-automaton, for any $j' < j$. □

The sample languages from the proofs of Lemma 1 and Lemma 2 have disjunctive alphabets and, therefore, they can be combined in order to obtain hierarchies with combined constraints. E.g., the language $L^{(k,j)} \cup L_{\mathrm{m}}^{(k',j')}$ is a language which can be accepted as an h-proper language of a $\hat{k}$-det-mon(j')-wcf-RLWC$(\hat{j})$-automaton, where $\hat{k} = max(k,k')$ and $\hat{j} = max(j,j')$, but not as an h-proper language of neither a k-wcf-RLWW$(j-1)$-automaton, nor a k'-mon$(j'-1)$-wcf-RLWWC(j').

In a similar way, we obtain the following consequences. We present here only such consequences which can be interpreted as linguistically relevant with respect to the complete lexical disambiguation.

Corollary 3. *For all $j,k \geq 1$, all prefixes $pref_X, pref_Y \in \{$det-wcf, det-fin(i)-wcf$\}$ and all $X,Y \in \{$hRLWW, hRLWWD, hRLWWC$\}$, the following holds:*

(a) $\mathscr{L}_A(k\text{-}pref_X\text{-}X(j)) \subset \mathscr{L}_A(k\text{-}pref_X\text{-}X(j+1))$,
(b) $\mathscr{L}_A(k\text{-}pref_X\text{-}X(j+1)) \setminus \mathscr{L}_A(k\text{-}pref_Y\text{-}Y(j)) \neq \emptyset$,
(c) $\mathscr{L}_A(k\text{-}pref_X\text{-}X(j)) \subset \mathscr{L}_A((k+1)\text{-}pref_X\text{-}X(j))$,
(d) $\mathscr{L}_A((k+1)\text{-}pref_X\text{-}X(j)) \setminus \mathscr{L}_A(k\text{-}pref_Y\text{-}Y(j)) \neq \emptyset$.

If additionally $i \geq j+3$, the following holds:

(aa) $\mathscr{L}_A(k\text{-mon}(j)\text{-}pref_X\text{-}X(j)) \subset \mathscr{L}_A(k\text{-mon}(j+1)\text{-}pref_X\text{-}X(j+1))$,
(bb) $\mathscr{L}_A(k\text{-mon}(j+1)\text{-}pref_X\text{-}X(j+1)) \setminus \mathscr{L}_A(k\text{-}pref_Y\text{-}Y(j)) \neq \emptyset$,
(cc) $\mathscr{L}_A(k\text{-mon}(j)\text{-}pref_X\text{-}X(j)) \subset \mathscr{L}_A((k+1)\text{-mon}(j)\text{-}pref_X\text{-}X(j))$,
(dd) $\mathscr{L}_A((k+1)\text{-mon}(j)\text{-}pref_X\text{-}X(j)) \setminus \mathscr{L}_A(k\text{-}pref_Y\text{-}Y(j)) \neq \emptyset$.

We can see that the h-lexicalized syntactic analyses of det-wcf-hRLWW(j)-automata are sensitive to the maximal number of rewrite (SL- and W-) operations in a cycle and to the size of the window. The syntax of these languages is given directly by individual reductions, i.e., by individual instructions of the hRLWW(j)-automata. Namely the reductions of RLWWC(j)-automata describe (define) the (discontinuous) syntactic constituents of the analyzed words (sentences). Note that the monotonicity of degree one means a synonymy for context-freeness of the accepted languages by restarting automata. The monotonicity of higher degree means a degree of non-context-freeness of accepted languages. In the previous corollary we have transferred this concept from infinite to finite languages. That can be very useful for classification of individual syntactic features.

4 Conclusion

We have seen that monotone RLWW(i)-automata are not sensitive to the number of deletions and to the size of their window with respect to their input languages and that these languages do in general not yield reduction correctness preserving computations of RLWW(i)-automata. On the other hand, hRLWW(i)-automata satisfy the reduction correctness preserving property with respect to their basic and h-proper languages, and consequently also with respect to their lexicalized syntactic analysis. The reduction

correctness preserving property enforces the sensitivity to the number of rewritings in a reduction and to the size of the window.

We believe that the class of h-proper languages of det-mon(2)-wcf-RLWWC(2)-automata is strong enough to model lexicalized (surface) syntax of natural languages, that is, to model their reduction correctness preserving lexicalized syntactic analysis. Namely, we strongly believe that the class of h-proper languages of det-mon(2)-wcf-RLWWC(2)-automata is a superclass of the class of mildly context-sensitive languages [5, 6]. In the future we will try to characterize the class of mildly context-sensitive languages by h-proper languages of RLWW-automata with some constraints.

Our long term goal is to propose and support a formal (and possibly also software) environment for a further study and development of Functional Generative Description (FGD) of Czech (see [9]). We strongly believe that the lexical disambiguation of FGD can be fully described by (a certain refinement of) det-mon(4)-wcf-RLWWD(4)-automata.

We stress that our current efforts cover an important gap in theoretical tools supporting computational and corpus linguistics. Chomsky grammars and the corresponding types of automata do not support lexicalized syntactic analysis, as these grammars work with categories bound to individual constituents with respect to constituent syntactic analysis. They do not support syntactic analysis with any kind of correctness preserving property, but they do support several types of insensitivity to the form of individual grammar rules (see several normal forms for context-free grammars, like Chomsky and Greibach normal form [2]), and, finally, they do not support the classification of finite phenomena of (natural) languages.

On the other hand, in corpus linguistics, only finite language phenomena can be observed. Now the basic and h-proper languages of hRLWW(i)-automata in weak cyclic form allow common classifications of finite phenomena as well as classifications of their infinite generalizations to the corresponding parts of the Chomsky hierarchy. All these classifications are based on the reduction correctness preserving property and the weak cyclic form. Let us recall for restarting and list automata the monotonicity means a synonymy for context-freeness. Here we are able to distinguish between finite monotone and finite non-monotone languages (syntactic phenomena), too.

Finally, note that many practical problems in computational and corpus linguistic became decidable if we consider only parametrized finite languages.

References

[1] Yehoshua Bar-Hillel: A quasi-arithmetical notation for syntactic description. *Language* 29: 47–58 (1953)

[2] John E. Hopcroft, Jeffrey D. Ullman: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, M.A. (1979)

[3] Petr Jančar, František Mráz, Martin Plátek, Jörg Vogel: Restarting automata. In: Horst Reichel (ed.): FCT'95, Proc., pages 283–292, LNCS 965, Springer, Berlin (1995)

[4] Petr Jančar, František Mráz, Martin Plátek, Jörg Vogel: On monotonic automata with a restart operation. *J. Autom. Lang. Comb.* 4: 287–311 (1999)

[5] Aravind K. Joshi, K. Vijay-Shanker, David Weir: The convergence of mildly context-sensitive grammatical formalisms. In: Peter Sells, Stuart Shieber, Tom Wasow (eds.), *Foundational Issues in Natural Language Processing*, pages 31–82, MIT Press, Cambridge MA (1991)

[6] Aravind K. Joshi, Yves Schabes: Tree-adjoining grammars. In: Grzegorz Rozenberg, Arto Salomaa (eds.), *Handbook of Formal Languages*, vol. 3, pages 69–123, Springer, Berlin, New York (1997)

[7] Tomasz Jurdziński, František Mráz, Friedrich Otto, Martin Plátek: Degrees of non-monotonicity for restarting automata. *Theor. Comp. Sci.* 369: 1–34 (2006)

[8] Markéta Lopatková, Martin Plátek, Vladislav Kuboň: Modeling syntax of free word-order languages: Dependency analysis by reduction. In: Václav Matoušek, Pavel Mautner, Tomáš Pavelka (eds.), TSD 2005, Proc., pages 140–147, LNCS 3658, Springer, Berlin (2005)

[9] Markéta Lopatková, Martin Plátek, Petr Sgall: Towards a formal model for functional generative description: Analysis by reduction and restarting automata. *Prague Bull. Math. Linguistics* 87: 7–26 (2007)

[10] František Mráz, Martin Plátek, Friedrich Otto: A Measure for The Degree of Nondeterminism of Context-free Languages. In: Jan Holub and Jan Žďárek (eds.), *Proceedings of CIAA 2007*, pages 192–20S, LNCS 4783, Springer, Berlin (2007)

[11] Niemann, G. Otto, F.: Restarting automata, Church-Rosser languages, and representations of re languages. In: Developments In Language Theory: Foundations, Applications, and Perspectives, World Scientific, 2000, 103–114

[12] Martin Plátek: Two-way restarting automata and j-monotonicity. In: Leszek Pacholski, Peter Ružička (eds.): SOFSEM'01, Proc., pages 316–325, LNCS 2234, Springer, Berlin (2001)

[13] Martin Plátek, Friedrich Otto: On h-lexicalized restarting automata. In: Erzsébet Csuhaj-Varjú, Pál Dömösi, György Vaszil (eds.), *AFL 2017, Proc.*, Open Publishing Association, EPTCS 252: 219–233 (2017), DOI:10.4204/EPTCS.252.21

[14] Martin Plátek, Friedrich Otto, František Mráz: On h-lexicalized automata and h-syntactic analysis. In: *ITAT 2017, Proc.*, CEUR Workshop Proceedings Vol. 1885, pp. 40–47 (2017)

[15] Martin Plátek, Friedrich Otto, František Mráz: On h-Lexicalized Restarting List Automata. *Technical Report*, www.theory.informatik.uni-kassel.de/projekte/RL2016v6.4.pdf, Kassel (2017)

[16] Natalie Schluter: Restarting automata with auxiliary symbols restricted by lookahead size. *Intern. J. Comput. Math.* 92: 908–938 (2015)