Lexicalized Syntactic Analysis by Restarting Automata

František Mráz[1], Friedrich Otto[1], Dana Pardubská[2⋆], and Martin Plátek[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, otto@ktiml.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 automata* that can rewrite at most i times per cycle for some $i \geq 1$ (hRLWW(i)-automata). This model is considered useful for the study of lexical (syntactic) disambiguation, which is a concept from linguistics. It is based on certain reduction patterns. We study lexical disambiguation through the formal notion of *h-lexicalized syntactic analysis* (hLSA). The hLSA is composed of a *basic language* and the corresponding *h-proper language*, which is obtained from the basic language by mapping all basic symbols to input symbols. We stress the sensitivity of hLSA by hRLWW(i)-automata to the size of their windows, the number of possible rewrites per cycle, and the degree of (non-)monotonicity. We introduce the concepts of *contextually transparent languages* (CTL) and *contextually transparent lexicalized analyses* based on very special reduction patterns, and we present two-dimensional hierarchies of their subclasses based on the size of windows and on the degree of synchronization. The bottoms of these hierarchies correspond to the context-free languages. CTL creates a proper subclass of context-sensitive languages with syntactically natural properties.

1 Introduction

This paper is a continuation of conference paper [11]. The motivation for this paper is to study lexical disambiguation, which is a basic concept of linguistic schools working with lexicalized syntax. Let us note that traditional dependency syntaxes are lexicalized in our sense.

In a lexicalized syntactical analysis of a sentence, at first all input words are replaced by disambiguated word forms, i.e., original words are enhanced with, e.g., morphological and syntactic categories, like the input word 'means' can be extended with a tag that it is a verb or a different tag that it is a noun which is further refined with another tag distinguishing whether it plays the role of subject or object. After such disambiguation, the lexicalized syntactic analysis checks whether the tagged word forms constitute a (grammatically) correct sentence which is correctly tagged.

A model of the restarting automaton that formalizes lexicalized syntactic disambiguation in a similar way as categorial grammars (see, e.g., [1]) – the *h-lexicalized restarting automaton* (hRLWW) – was introduced in [9]. This model is obtained from the two-way restarting automaton of [8] by adding a letter-to-letter morphism h that assigns an input symbol to each working symbol. This morphism models the (pure

⋆ The research is partially supported by VEGA 2/0165/16.
⋆⋆ The research was partially supported by the grant of the Czech Science Foundation No. 19-05704S by the authors stay at Institute of Computer Science, Czech Academy of Sciences.

František Mráz, Friedrich Otto, Dana Pardubská, Martin Plátek: Lexicalized Syntactic Analysis by Restarting Automata, pp. 69–83.
Proceedings of PSC 2019, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-06618-8

non-syntactic) lexical disambiguation. Now the *basic language* $L_{\mathrm{C}}(M)$ 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* $L_{\mathrm{hP}}(M)$ of M is obtained from $L_{\mathrm{C}}(M)$ through the morphism h.

The set of pairs $\{\,(h(w), w) \mid w \in L_{\mathrm{C}}(M)\,\}$, denoted as $L_{\mathrm{A}}(M)$, is called the h-lexicalized syntactic (sentence) 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 or categories).

In contrast to the original hRLWW-automaton that uses exactly one length-reducing rewrite in a cycle, here we study *h-lexicalized restarting automata* that allow up to $i \geq 1$ length-reducing rewrites in a cycle (hRLWW(i)). Our first goal is to show that these models are suited for a transparent and sufficiently flexible modeling of the lexical analysis by analysis by reduction (compare to [6]).

Analysis by reduction is traditionally used to analyze sentences of natural languages with a higher degree of word-order freedom like, e.g., Czech, Latin, or German (see, e.g., [6]). Usually, a human reader is supposed to understand the meaning of a given sentence before he starts to analyze it; h-lexicalized syntactic analysis (hLSA) based on the h-lexicalized analysis by reduction ($\mathrm{AR_h}$) simulates such a behavior by analyzing sentences, where morphological and syntactical tags have been added to the word forms and punctuation marks (see, e.g., [6]). An important property of analysis by reduction is the so-called correctness preserving property. Using hRLWW(i)-automata the linguistic correctness preserving property is simulated by the formal notion of *basic correctness preserving property*.

We stress here newly the constraint of *strong cyclic form*. It preserves the essential part of the power of hRLWW(i)-automata, and, additionally, it allows to extend the complexity results obtained for classes of infinite languages and hLSAs also to classes of finite languages and hLSAs. This is quite useful for the classification and learning of individual phenomena in computational and corpus linguistics, where all the (syntactic) observations are of a finite nature. It is also useful for the formulation of techniques for the localization of syntactic errors (grammar-checking).

Finally, we introduce the concepts of *contextually transparent languages* (CTL) and *contextually transparent lexicalized analyses* (CTLA), which create a formal basis for an environment for lexical analysis and grammar-checking of natural languages. We establish a two level and two-dimensional essential refinement of the Chomsky hierarchy in the area of CTL. We transfer this refinement also to the area of CTLA.

In this paper we do not pay attention to input languages, which are the languages usually studied in the automata theory, as they are not suitable for the modeling of lexicalized sentence disambiguation (see [11]).

2 Definitions

By $\subseteq$ and $\subset$ we denote the subset and the proper subset relation, respectively. 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 [8]. In contrast to [10], we do not consider general h-lexicalized two-way restarting list automata 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* $\mathsf{RLWW}(i)$*-automaton for short, is a machine with a flexible tape and a finite-state control. It is defined through a 9-tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, i, \delta)$, 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 $¢, \$ \notin \Gamma$, called* sentinels*, are the markers for the left and the 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 below), and*

$$\delta : Q \times \mathcal{PC}^{\leq k} \to \mathcal{P}(\,(Q \times (\{\mathsf{MVR}, \mathsf{MVL}\} \cup \{\,\mathsf{SL}(v) \mid v \in \mathcal{PC}^{\leq k-1}\,\})) \cup \{\mathsf{Restart}, \mathsf{Accept}, \mathsf{Reject}\})$$

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

$$\mathcal{PC}^{\leq k} = (\{¢\} \cdot \Gamma^{k-1}) \cup \Gamma^k \cup (\Gamma^{\leq k-1} \cdot \{\$\}) \cup (\{¢\} \cdot \Gamma^{\leq k-2} \cdot \{\$\})$$

is the set of possible contents *of the read/write window of M.*

Being in a state $q \in Q$ and seeing a word $u \in \mathcal{PC}^{\leq k}$ in its window, the automaton can perform six 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 $u \notin \{\lambda, ¢\} \cdot \Gamma^{\leq k-1} \cdot \{\$\}$. 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 $u \notin \{¢\} \cdot \Gamma^* \cdot \{\lambda, \$\}$. 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 $q' \in Q$, $v \in \mathcal{PC}^{\leq k-1}$, 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 ¢ (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 ¢).*
4. *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 ¢, and to reenter the initial state q_0.*
5. *An* accept step $(q, u) \longrightarrow \mathsf{Accept}$ *assumes that $\mathsf{Accept} \in \delta(q, u)$. It causes M to halt and accept.*
6. *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 $\mathsf{RLWW}(i)$-automaton M is a word $\alpha q \beta$, where $q \in Q$, and either $\alpha = \lambda$ and $\beta \in \{¢\} \cdot \Gamma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Gamma^*$ 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 $q_0 ¢ w \$$, where $w \in \Gamma^*$; if $w \in \Sigma^*$, then $q_0 ¢ w \$$ 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 $\mathsf{RLWW}(i)$-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 $C_{\ell+1}$ is obtained from C_ℓ by a step of M, for all $0 \leq \ell < j$. In the following we only consider computations of $\mathsf{RLWW}(i)$-automata which are finite and end either by an accept or by a reject step.

Cycles and tails: Any finite computation of an $\mathsf{RLWW}(i)$-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 an $\mathsf{RLWW}(i)$-automaton executes at least one, but at most i SL-steps. Moreover, it must not execute any SL-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 $q_0\text{¢}u\$$ and ends with the restarting configuration $q_0\text{¢}v\$$. 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 $\mathsf{RLWW}(i)$-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 $q_0\text{¢}w\$$ 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 $q_0\text{¢}w\$$ and ends by executing an accept step. By $L_{\mathrm{C}}(M)$ 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 h-lexicalized $\mathsf{RLWW}(i)$-automaton.

Definition 2. *Let i be a positive integer. An* h-lexicalized $\mathsf{RLWW}(i)$-automaton*, or an* $\mathsf{hRLWW}(i)$-automaton*, is a pair $\widehat{M} = (M, h)$, where $M = (Q, \Sigma, \Gamma, \text{¢}, \$, q_0, k, i, \delta)$ is an $\mathsf{RLWW}(i)$-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 $L_{\mathrm{C}}(\widehat{M})$ *of $\widehat{M}$ is the language $L_{\mathrm{C}}(M)$. Further, we take $L_{\mathrm{hP}}(\widehat{M}) = h(L_{\mathrm{C}}(M))$, and we say that $\widehat{M}$* recognizes (or accepts) the h-proper language $L_{\mathrm{hP}}(\widehat{M})$.

Finally, the set $L_{\mathrm{A}}(\widehat{M}) = \{\, (h(w), w) \mid w \in L_{\mathrm{C}}(M) \,\}$ is called the h-lexicalized syntactic analysis *(shortly hLSA) by $\widehat{M}$.*

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

Evidently, for an $\mathsf{hRLWW}(i)$-automaton $\widehat{M}$, $L(\widehat{M}) \subseteq L_{\mathrm{hP}}(\widehat{M}) = h(L_{\mathrm{C}}(\widehat{M}))$. Let us note that h-syntactic analysis formalizes the linguistic notion of *lexical sentence disambiguation*. Each auxiliary symbol $x \in \Gamma \setminus \Sigma$ of a word from $L_{\mathrm{C}}(\widehat{M})$ can be

[1] Basic languages were also called characteristic languages in [9] and several other papers, therefore, here we preserve the original notation with the subscript C.

considered as a disambiguated form of the input symbol $h(x)$. The following fact ensures the transparency for computations of $\mathsf{hRLWW}(i)$-automata.

Definition 3. (Basic Correctness Preserving Property)
Let M be an $\mathsf{hRLWW}(i)$-automaton. If $u \Rightarrow_M^{c^*} v$ and $u \in L_{\mathrm{C}}(M)$ induce that $v \in L_{\mathrm{C}}(M)$, and therewith $h(v) \in L_{\mathrm{hP}}(M)$ and $(h(v), v) \in L_{\mathrm{A}}(M)$, then we say that M is basically correctness preserving.

Fact 4. *Let M be a deterministic $\mathsf{hRLWW}(i)$-automaton. Then M is basically correctness preserving.*

Notations. For brevity, the prefix det- will be used to denote the property of being deterministic. For any class A of automata, $\mathcal{L}(\mathsf{A})$ will denote the class of input languages that are recognized by automata from A, $\mathcal{L}_{\mathrm{C}}(\mathsf{A})$ will denote the class of basic languages that are recognized by automata from A, $\mathcal{L}_{\mathrm{hP}}(\mathsf{A})$ will denote the class of h-proper languages that are recognized by automata from A, and $\mathcal{L}_{\mathrm{A}}(\mathsf{A})$ will denote the class of hLSA (h-lexicalized syntactic analyses) that are defined by automata from A.

For a natural number $k \geq 1$, $\mathcal{L}(k\text{-}\mathsf{A})$, $\mathcal{L}_{\mathrm{C}}(k\text{-}\mathsf{A})$, $\mathcal{L}_{\mathrm{hP}}(k\text{-}\mathsf{A})$, $\mathcal{L}_{\mathrm{A}}(k\text{-}\mathsf{A})$ will denote the classes of input, basic, h-proper languages, and hLSAs, respectively, that are recognized by those automata from A that use a read/write window of size at most k.

2.1 Further Refinements, and Constraints on hRLWW(i)-Automata

Here we introduce some constrained types of rewrite steps which are motivated by different types 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 $u = v_1u_1v_2u_2v_3$, $u_1, u_2 \in \Gamma^*$, $|u_1u_2| \geq 1$, 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 $\mathsf{RLWW}(i)$-automaton is called an $\mathsf{RLW}(i)$-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 the denotation for types of automata, 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 a single W denotes the use of an input alphabet only (that is, the working alphabet coincides with the input alphabet).

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

Fact 5. (Equalities of Languages for $\mathsf{hRLW}(i)$-automata.)
For any $\mathsf{RLW}(i)$-automaton M, $L(M) = L_{\mathrm{C}}(M) = L_{\mathrm{hP}}(M)$.

An RLW(i)-automaton is called an 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 an RLWWC(i)-*automaton* if all its rewrite steps are CL-steps. Similarly, an RLWW(i)-automaton is called an 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 notation also for subclasses of RLWW(i)- and hRLWW(i)-automata. Additionally, prefix k- for a type X of RLWW(i)-automata and an integer $k \geq 1$ will denote the subclass of X of automata of window size at most k. For example, 3-det-hRLWC(i) denotes the class of deterministic h-lexicalized RLWC(i)-automata with window size at most 3.

We recall the notion of *monotonicity* (see e.g. [11]) 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 subsequence $(C_{\ell_1}, C_{\ell_2}, \ldots, C_{\ell_n})$ 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 subsequence of rewrite configurations is monotone. Here a configuration is called a *rewrite configuration* if in this configuration an SL-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 hRLWW(i)-automata. This notion of monotonicity has already been considered in various papers (see [4]) similarly as the following generalization of it.

A det-mon-hRLWW(i)-automaton can be used to model bottom-up, correctness preserving, context-free parsers. In order to model also bottom-up, correctness preserving, mildly context-sensitive parsers, a notion of j-monotonicity for restarting automata is used here; j-monotonicity was introduced in [8]. For an integer $j \geq 1$, an hRLWW(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 (possibly noncontinuous) subsequences such that each of these subsequences is monotone. We use the prefix mon(j)- to denote j-monotone types of hRLWW(i)-automata.

A restriction of the form of restarting automata called *strong cyclic form* (see [3]) can also be transferred to hRLWW(i)-automata. An hRLWW M is said to be in *strong cyclic form* if $|uv| \leq k$ for each halting configuration $\mathrm{¢}uqv\$$ of M, where k is the size of the read/write window of M. Thus, before M can halt, it must erase sufficiently many letters from its tape. The prefix scf- will be used to denote restarting automata that are in strong cyclic form. The concept of strong cyclic form is useful for techniques of grammar-checking (localization of syntactic errors) by hRLWW(i)-automata.

Lemma 6. *Let $i \geq 1$, and let M be an* RLWW(i)*-automaton. Then there exists a* scf-RLWW(i)*-automaton M_scf such that $L_\mathrm{C}(M) = L_\mathrm{C}(M_\mathrm{scf})$ and, for all words u, v, $u \Rightarrow_M^{c^*} v$ implies $u \Rightarrow_{M_\mathrm{scf}}^{c^*} v$. Moreover, all reductions of M_scf that are not possible for M are in contextual form. If M is deterministic and/or j-monotone for some $j \geq 1$, then M_scf is deterministic and/or j-monotone as well.*

Proof. Let $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, i, \delta)$ be an RLWW(i)-automaton. It is easy to see that the language L_a of words from Γ^* accepted by M in tail computations is a regular sublanguage of $L_C(M)$. Therefore, there exists a deterministic finite automaton A_a such that $L(A_a) = \{ w \in L_a \mid |w| > k \}$. Similarly, the language L_r of words rejected by M in tail computations is regular and there exists a deterministic finite automaton A_r such that $L(A_r) = \{ w \in L_r \mid |w| > k \}$. Assume that the automata A_a and A_r have n_a and n_r states, respectively.

Now we can transform M into an scf-RLWW(i)-automaton $M_{scf} = (Q_{scf}, \Sigma, \Gamma, ¢, \$, q_0, k_{scf}, i, \delta_{scf})$ of window size $k_{scf} = \max\{k, n_a + 1, n_r + 1\}$. The transition relation δ_{scf} contains all transitions of M with the following exception. All accepting steps of M are replaced by MVL-steps into a new state $q_{l,a}$. As M cannot rewrite in tail computations, when M_{scf} enters the state $q_{l,a}$, the contents of its tape has not been changed since the last restart. In this case, M_{scf} moves to the left sentinel and starts to simulate A_a. During this simulation:

- either M_{scf} detects that the current tape contents is of length at most k_{scf} and it accepts,
- or M_{scf} detects that the current tape contents w is of length greater than k_{scf}; in this case, while moving to the right, it continues to simulate A_a until the right sentinel \$ appears in the window. From the pumping lemma for regular languages we know that if $w \in L_a$, then there exists a factorization $w = w'xyz$ such that $|y| > 0$, $|y|+|z| \leq n_a$, $w'xz \in L_a$, and the word $xyz\$$ of length k_{scf} is the contents of the read/write window of M_{scf}. Accordingly, M_{scf} deletes the factor y and restarts. Even if there exist several such factorisations of w, for constructing M_{scf} we select one such factor for any contents of the read/write window $xyz\$$.

Finally, we must ensure that M_{scf} does not halt and reject for any word of length greater than k_{scf}. We can do that by adding new steps to the transition relation δ_{scf}. If $\delta(q, u)$ contains Reject for some state $q \in Q$ and some contents u of the read/write window, then we replace this reject step by a MVL-step into a new state $q_{l,r}$, in which the automaton will move its window to the leftmost position, and then it starts to move to the right while simulating A_r. Similarly as above for A_a, during the simulation of A_r, the automaton either rejects if the current contents of the tape is not longer than k_{scf}, or it shortens the tape by applying the pumping lemma for L_r. Such a simulation of A_r is possible, as when M enters a configuration with state q and u in its read/write window, then it can halt, and hence, the tape contents has not been rewritten since the last restart (as M cannot rewrite in tail computations).

From the construction above we immediately see that M_{scf} is in strong cyclic form and that $L_C(M_{scf}) = L_C(M)$. Moreover, if M is deterministic, then M_{scf} is deterministic, too. Additionally, if M is j-monotone, then M_{scf} 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. □

3 On the Power and Sensitivity of Lexicalized Constructions

First we introduce the constraint of synchronization. We say that an hRLWW(i)-automaton is *synchronized* if its degree of monotonicity is not higher than the number i of allowed rewrites per cycle. We denote the constraint of synchronization by the

prefix syn-. In this paper we stress the relations of the degree of synchronization to the levels of the Chomsky hierarchy and we show that it can also be used for building a hierarchy within finite languages.

In this section we will study lexicalized constructions of scf-hRLWW(i)-automata. By lexicalized constructions we mean the basic and h-proper languages and hLSAs. We will see that with respect to lexicalized constructions, scf-hRLWW(i)-automata (and their variants) are sensitive to several types of constraints, as, e.g., the window size, the number of rewrites per cycle, and the degree of synchronization. Through these constraints we essentially refine the Chomsky hierarchy, and we will do so in two phases. In phase one we refine the context-sensitive languages by degrees of synchronization. Then, by using the window size, we refine the individual areas of lexicalized constructions that are given by the individual degrees of synchronization.

In order to present our results, we still have to introduce some additional notions. For any type X of RLWW(i)-automaton and any integer $j \geq 0$, we use fin(j)-X to denote the subclass of X-automata that perform at most j reductions in any accepting computation. Thus, for such an automaton, each accepting computation consists of up to j cycles only and a tail. Finally, by fin-X, we denote those X-automata that are of type fin(j)-X for some $j \geq 0$.

3.1 Small Finite Separating Witness Languages

This subsection represents the technical core of this section. It establishes the sensitivity of basic and h-proper languages of scf-hRLWW(i)-automata to the size of their windows, to the number of deletions by a reduction, and to the degree of monotonicity. This is achieved by constructions of small finite languages. In this way it is shown that the sensitivity relies on small syntactic observations.

Proposition 7. *Let $k \geq 2$, let a be a letter, and let $L_1(k) = \{a^k\}$. Then the following statements hold for $L_1(k)$:*

(a) $L_1(k) \in \mathcal{L}_{\mathrm{C}}(k\text{-scf-fin(0)-det-mon-RLWC})$.
(b) $L_1(k) \notin \mathcal{L}_{\mathrm{C}}((k-1)\text{-scf-hRLWW}) \cup \mathcal{L}_{\mathrm{hP}}((k-1)\text{-scf-hRLWW})$.

This proposition shows that for the separation of language classes based on the size of the read/write window it suffices to consider witness languages of cardinality one.

Proof. (a) Let $M_1(k)$ be the deterministic RLWC-automaton with window size k that proceeds as follows given a word $w = a^n$ as input:

1. If $n < k$, then $M_1(k)$ rejects in a tail computation.
2. If $n = k$, then $M_1(k)$ accepts in a tail computation after moving its window to the right sentinel.
3. If $n = i \cdot k$ for some $i \geq 2$, then $M_1(k)$ deletes the last occurrence of the letter a and restarts.
4. If $n = i \cdot k + j$ for some $i \geq 1$ and some $j \in \{1, 2, \ldots, k-1\}$, then $M_1(k)$ deletes the suffix a^k and restarts.

It is now easily seen that $L(M_1(k)) = L_{\mathrm{C}}(M_1(k)) = L_1(k)$, that $M_1(k)$ is in strong cyclic form, and that it is monotone (of degree 1). Further, as each accepting computation of $M_1(k)$ consists of just a tail, $M_1(k)$ is a fin(0)-RLWC-automaton.

(b) For any $(k-1)$-scf-hRLWW-automaton M, the language $L_{\mathrm{C}}(M)$ (and therewith the language $L_{\mathrm{hP}}(M)$) is either empty or it contains at least one word of length at most $k-1$. As $L_1(k)$ only contains a word of length $k > k-1$, we see that $L_1(k)$ is neither the basic language nor the h-proper language of any $(k-1)$-scf-hRLWW-automaton. □

Proposition 8. *Let $k, j \geq 1$, let a be a letter, and let $L_2(j,k) = \{a^{k\cdot(j+1)}, a^k\}$. Then the following statements hold for $L_2(j,k)$:*

(a) $L_2(j,k) \in \mathcal{L}_{\mathrm{C}}(k\text{-scf-fin}(1)\text{-det-mon-RLWC}(j))$.
(b) $L_2(j,k) \notin \mathcal{L}_{\mathrm{C}}(k\text{-scf-hRLWW}(j')) \cup \mathcal{L}_{\mathrm{hP}}(k\text{-scf-hRLWW}(j'))$ *for any* $j' < j$.

This proposition shows that for the separation of language classes based on the number of rewrites that can be executed during a cycle it suffices to consider witness languages of cardinality two.

Proof. (a) Let $M_2(j,k)$ be the deterministic RLWC-automaton with window size k that proceeds as follows given the word a^n as input:

1. If $n < k$, then $M_2(j,k)$ rejects in a tail computation.
2. If $n = k$, then $M_2(j,k)$ accepts in a tail computation.
3. If $n = i \cdot k$ for some $2 \leq i \leq j$, then $M_2(j,k)$ rewrites the word a^n into the empty word and restarts. Each of the rewrite steps deletes the suffix a^k of the current tape contents.
4. If $n = (j+1) \cdot k$, then $M_2(j,k)$ rewrites the word a^n into the word a^k and restarts. Each of the rewrite steps deletes the suffix a^k of the current tape contents.
5. If $n = i \cdot k$ for some $i \geq j+2$, then $M_2(j,k)$ simply deletes the last occurrence of the letter a and restarts.
6. If $n = i \cdot k + \ell$ for some $i \geq 1$ and $\ell \in \{1, 2, \ldots, k-1\}$, then $M_2(j,k)$ simply deletes the suffix a^k and restarts.

It follows that $L(M_2(j,k)) = L_2(j,k)$, that $M_2(j,k)$ is in strong cyclic form, and monotone (of degree 1). Further, each accepting computation of $M_2(j,k)$ consists of at most a single cycle and a tail, that is, $M_2(j,k)$ is a fin(1)-RLWC-automaton.

(b) Assume that M is a k-scf-hRLWW(j')-automaton such that $L_{\mathrm{C}}(M) = L_2(j,k)$, where $j' < j$. As $a^{k\cdot(j+1)} \in L_2(j,k)$ and $|a^{k\cdot(j+1)}| = k \cdot (j+1) > k$, and as M is in strong cyclic form, each accepting computation of M on input $a^{k\cdot(j+1)}$ begins with a cycle. As a^k is the only other word in $L_2(j,k)$, this cycle must rewrite $a^{k\cdot(j+1)}$ into the word a^k, for which $j \cdot k$ letters must be deleted. However, as M has window size k and can only execute $j' < j$ many rewrites per cycle, we see that it can delete at most $j' \cdot k < j \cdot k$ many letters in a single cycle. This implies that $L_{\mathrm{C}}(M) \neq L_2(j,k)$. Finally, as each word $w \in L_{\mathrm{hP}}(M)$ corresponds to a word of the same length from $L_{\mathrm{C}}(M)$, the argument above also shows that $L_{\mathrm{hP}}(M) \neq L_2(j,k)$. □

Proposition 9. *Let $k, j \geq 2$, $\Sigma = \{a, b, c\}$, and let $u_i = a^k$ for even i and $u_i = b^k$ for odd i. Finally, let*

$$L_3(j,k) = \{\, (u_i u_{i+1} \cdots u_j c^{k\cdot j^2})^j \mid i = 1, 2, \ldots, j \,\} \cup \{\, c^{k\cdot j\cdot r} \mid 0 \leq r \leq j^2 \,\}.$$

Then the following statements hold for $L_3(j,k)$:

(a) $L_3(j,k) \in \mathcal{L}_{\mathrm{C}}(k\text{-scf-fin}(j+j^2)\text{-det-mon}(j)\text{-RLWC}(j))$.

(b) $L_3(j,k) \notin \mathcal{L}_{\mathrm{C}}(k\text{-scf-mon}(j')\text{-hRLWW}(j)) \cup \mathcal{L}_{\mathrm{hP}}(k\text{-scf-mon}(j')\text{-hRLWW}(j))$ *for any* $j' < j$.
(c) $L_3(j,k) \notin \mathcal{L}_{\mathrm{C}}(k\text{-scf-hRLWW}(j')) \cup \mathcal{L}_{\mathrm{hP}}(k\text{-scf-hRLWW}(j'))$ *for any* $j' < j$.

This proposition shows that for the separation of language classes based on the degree of monotonicity it suffices to consider finite witness languages.

Proof. To simplify the notation we introduce, for each $i = 1, 2, \ldots, j$, the word $w_i = (u_i u_{i+1} \cdots u_j c^{k \cdot j^2})^j$ and the word $w_{j+1} = c^{k \cdot j^2 \cdot j}$.

(a) Let $M_3(j,k)$ be the deterministic RLWC-automaton that proceeds as follows given a word w as input. If $|w| \leq k$, then $M_3(j,k)$ performs a tail computation in which it accepts if $w = \lambda$ and rejects otherwise. If $|w| > k$, then $M_3(j,k)$ performs a cycle with the following rewrites completed by a restart step:

1. If $w = w_i$ for some $i \in \{1, 2, \ldots, j\}$, then $M_3(j,k)$ executes j rewrite steps that each delete a factor u_i. In this way w_i is rewritten into w_{i+1}.
2. If $w = c^{k \cdot j \cdot r}$ for some $r \in \{1, 2, \ldots, j^2\}$, then $M_3(j,k)$ deletes the suffix $c^{k \cdot j}$ by executing j rewrite steps that each delete the suffix c^k.
3. When w is not of any of the forms considered above, then $M_3(j,k)$ executes a rewrite step which either
   - deletes the last symbol of w, if $|w| = k \cdot \ell$, for some $\ell > 1$, or
   - deletes a k-letter suffix of w, if the length of w is not divisible by k.

Now it is easily seen that $L(M_3(j,k)) = L_3(j,k)$, that $M_3(j,k)$ is in strong cyclic form, and that each of its accepting computations consists of at most $j + j^2$ cycles and a tail. It remains to show that $M_3(j,k)$ is j-monotone.

If the input word w does not belong to the language $L_3(j,k)$, then in each cycle just a suffix is deleted, that is, the resulting computation is monotone. If $w = c^{k \cdot j \cdot r}$ for some $r \in \{1, 2, \ldots, j^2\}$, then the suffix $c^{k \cdot j}$ is deleted by j rewrite steps that all have right distance $k + 1$. Thus, the resulting accepting computation is monotone.

Hence, it remains to consider the first j cycles for the input $w_1 = (u_1 u_2 \cdots u_j c^{k \cdot j^2})^j$. In the first cycle the j factors u_1 are deleted, in the second cycle the j factors u_2 are deleted, and so forth. Now the leftmost rewrites in all these cycles yield a monotone sequence, the leftmost but one rewrites in all these cycles yield another monotone sequence, and so forth. Thus, we see that $M_3(j,k)$ is indeed j-monotone.

(b) Let M be a k-scf-hRLWW(j)-automaton such that $L_{\mathrm{C}}(M) = L_3(j,k)$. We claim that M is not j'-monotone for any $j' < j$. Assume to the contrary that M is j'-monotone for some $j' < j$. As $w_1 \in L_3(j,k)$, M has an accepting computation for w_1 that is j'-monotone. In addition, as M is in strong cyclic form, this accepting computation must rewrite w_1 into a word of length at most k before it accepts. From the definition of $L_3(j,k)$ we see that this computation must start with the following sequence of cycles, as in each cycle M can delete at most $k \cdot j$ letters:

$$w_1 \Rightarrow_M^c w_2 \Rightarrow_M^c \cdots \Rightarrow_M^c w_j \Rightarrow_M^c w_{j+1}.$$

Thus, the rewrite (delete) steps that are executed in this sequence can be displayed as follows:

cycle no.	1-st rewrite	2-nd rewrite	$\cdots$	$(j-1)$-st rewrite	j-th rewrite
1	u_1	u_1	$\cdots$	u_1	u_1
2	u_2	u_2	$\cdots$	u_2	u_2
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$j-1$	u_{j-1}	u_{j-1}	$\cdots$	u_{j-1}	u_{j-1}
j	u_j	u_j	$\cdots$	u_j	u_j

The next table shows the corresponding right distances, where we disregard the right sentinel:

cycle no.	1-st rewrite	2-nd rewrite	$\cdots$	j-th rewrite
1	$j \cdot k \cdot (j^2+j)$	$(j-1) \cdot k \cdot (j^2+j)$	$\cdots$	$1 \cdot k \cdot (j^2+j)$
2	$j \cdot k \cdot (j^2+j-1)$	$(j-1) \cdot k \cdot (j^2+j-1)$	$\cdots$	$1 \cdot k \cdot (j^2+j-1)$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$j-1$	$j \cdot k \cdot (j^2+2)$	$(j-1) \cdot k \cdot (j^2+2)$	$\cdots$	$1 \cdot k \cdot (j^2+2)$
j	$j \cdot k \cdot (j^2+1)$	$(j-1) \cdot k \cdot (j^2+1)$	$\cdots$	$1 \cdot k \cdot (j^2+1)$

Now it can be checked easily that, for each $i \in \{1, 2, \ldots, j-1\}$, the right distance of the i-th rewrite in cycle j, which is $d_1 = (j+1-i) \cdot k \cdot (j^2+1)$, is larger than the right distance of the $(i+1)$-st rewrite in cycle 1, which is $d_2 = (j-i) \cdot k \cdot (j^2+j)$, since

$$d_1 = (j+1-i) \cdot k \cdot (j^2+1) = (j-i) \cdot k \cdot (j^2+1) + k \cdot (j^2+1),$$

while

$$d_2 = (j-i) \cdot k \cdot (j^2+j) = (j-i) \cdot k \cdot (j^2+1) + (j-i) \cdot k \cdot (j-1).$$

Hence,

$$\begin{aligned} \tfrac{1}{k} \cdot (d_1 - d_2) &= j^2 + 1 - (j-i) \cdot (j-1) = j^2 + 1 - (j^2 - j - i \cdot j + i) \\ &= 1 + j + i \cdot j - i = 1 + j \cdot (1+i) - i > 0. \end{aligned}$$

Hence, in order to arrange the j^2-many rewrite steps in the above computation into monotone subsequences, we need at least j such subsequences. This implies that the above computation is not j'-monotone for any $j' < j$. Thus, $L_3(j,k)$ is not the basic language of a k-scf-mon(j')-hRLWW(j)-automaton for any $j' < j$. The same argument also shows that $L_3(j,k)$ is not the h-proper language of such an automaton.

(c) Let M be a k-scf-hRLWW(j')-automaton for some $j' < j$. We will first show that $L_{\mathrm{C}}(M)$ cannot be the language $L_3(j,k)$. Assume to the contrary that $L_{\mathrm{C}}(M) = L_3(j,k)$. As $w_1 \in L_3(j,k)$, M has an accepting computation for w_1. In addition, as M is in strong cyclic form, this accepting computation must reduce w_1 into a word $v \in L_3(j,k)$. But this is impossible with window size k and less than j rewrites in a cycle. The same argument also shows that $L_3(j,k)$ is not the h-proper language of such an automaton. □

3.2 Sensitivity of scf-hRLWW(i)-Automata

Now we focus on results that are related to the sensitivity of scf-hRLWW(i)-automata. In particular, we show the sensitivity of these automata to the size of the window, to the number of rewrites in a cycle, and to the degree of monotonicity.

Corollary 10. *For all $j, k \geq 1$, the following hold:*

(1) $\mathcal{L}_{\mathrm{C}}(k\text{-scf-fin}(1)\text{-det-syn-RLWC}(j+1)) \setminus \mathcal{L}_{\mathrm{hP}}(k\text{-scf-hRLWW}(j)) \neq \emptyset$.
(2) $\mathcal{L}_{\mathrm{C}}((k+1)\text{-scf-fin}(0)\text{-det-syn-RLWC}(j)) \setminus \mathcal{L}_{\mathrm{hP}}(k\text{-scf-hRLWW}(j)) \neq \emptyset$.

Proof. The statement in (1) follows from Proposition 8, where $L_2(j,k)$ was shown to belong to $\mathcal{L}_{\mathrm{C}}(k\text{-scf-fin}(1)\text{-det-mon-RLWC}(j))$, but not to $\mathcal{L}_{\mathrm{C}}(k\text{-scf-hRLWW}(j')) \cup \mathcal{L}_{\mathrm{hP}}(k\text{-scf-hRLWW}(j')))$ for any $j' < j$. Just observe that the automaton $M_2(j,k)$ for the language $L_2(j,k)$ is synchronized and that each of its accepting computations consists of at most one cycle and a tail.

The statement in (2) follows from Proposition 7, where $L_1(k) = \{a^k\}$ was shown to belong to $\mathcal{L}_{\mathrm{C}}(k\text{-scf-fin}(0)\text{-det-mon-RLWC})$ and not to $\mathcal{L}_{\mathrm{C}}((k-1)\text{-scf-hRLWW}) \cup \mathcal{L}_{\mathrm{hP}}((k-1)\text{-scf-hRLWW})$. Recall that the automaton $M_1(k)$ for $L_1(k)$ is synchronized and that its only accepting computation consists just of a tail computation. □

Next we show a similar hierarchy with respect to the degree of monotonicity which is related to the number of rewrites in a cycle.

Corollary 11. *For all $j, k \geq 1$, the following hold:*

$$\mathcal{L}_{\mathrm{C}}(k\text{-scf-fin-det-syn-RLWC}(j+1)) \setminus \mathcal{L}_{\mathrm{hP}}(k\text{-scf-mon}(j)\text{-hRLWW}(j+1)) \neq \emptyset.$$

Proof. This result follows from Proposition 9. □

3.3 On Characterizations of Context-Free Constructions

In what follows we use LRR to denote the class of left-to-right regular languages. Several characterizations of LRR in terms of restarting automata can be found in [7].

Theorem 12. *Let $X \in \{\mathsf{hRLWW}(1), \mathsf{hRLWWD}(1), \mathsf{hRLWWC}(1)\}$. Then* $\mathsf{LRR} = \mathcal{L}_{\mathrm{C}}(\text{scf-det-syn-}X)$ *and* $\mathsf{CFL} = \mathcal{L}_{\mathrm{hP}}(\text{scf-det-syn-}X)$.

Proof. An hRLWW(1)-automaton is synchronized if and only if it is monotone. It is known that the basic languages of monotone hRLWW(1)-automata are context-free [10]. As the class of context-free languages is closed under the application of morphisms, it follows that $\mathcal{L}_{\mathrm{hP}}(\text{syn-hRLWW(1)})$ only contains context-free languages.

On the other hand, it is shown in [10] that the class CFL coincides with the class of h-proper languages of det-mon-hRLWWC(1)-automata. Now we can use Lemma 6 to complete the proof. □

Remark. This theorem presents the robustness of the constraint of synchronization for several subclasses of hRLWW(1)-automata with respect to basic and h-proper languages. It enhances the results about the robustness of context-free and LRR-languages.

Notation. By CFLA we denote the class $\mathcal{L}_{\mathrm{A}}(\text{scf-det-syn-hRLWW}(1))$. With this notion we enhance the concept of context-freeness from formal languages to lexicalized syntactic analyses.

Corollary 13. *For all $X \in \{\mathsf{hRLWW}(1), \mathsf{hRLWWD}(1), \mathsf{hRLWWC}(1)\}$,*

$$\mathsf{CFLA} = \mathcal{L}_{\mathrm{A}}(\text{scf-det-syn-}X).$$

Proof. Let us first recall the definition of $\mathcal{L}_{\mathrm{A}}(X)$. For a class of restarting automata X, a set of pairs L belongs to the class $\mathcal{L}_{\mathrm{A}}(X)$ if there are a restarting automaton $M \in X$ and a letter-to-letter homomorphism h such that $L = \{ (h(w), w) \mid w \in L_{\mathrm{C}}(M) \}$. The proof then follows from the Theorem 12. □

Remark. This corollary presents the robustness of the constraint of synchronization for several subclasses of $\mathsf{hRLWW}(1)$-automata with respect to the lexicalized syntactic analysis.

4 On Contextually Transparent Constructions

In this section we introduce and study classes of contextually transparent (lexicalized language) constructions (CTC) which are composed from infinitely many subclasses given by degrees of synchronization.

Notations. For $i \geq 1$, we denote by $\mathsf{CTL}(i)$ the class $\mathcal{L}_{\mathrm{hP}}(\mathsf{scf\text{-}det\text{-}syn\text{-}hRLWWC}(i))$ and by $\mathsf{CTLA}(i)$ the class $\mathcal{L}_{\mathrm{A}}(\mathsf{scf\text{-}det\text{-}syn\text{-}hRLWWC}(i))$. Taking the union over all positive integers we obtain the classes $\mathsf{CTL} = \bigcup_{i\geq 1} \mathsf{CTL}(i)$ and $\mathsf{CTLA} = \bigcup_{i\geq 1} \mathsf{CTLA}(i)$. We say that CTL is the set of *contextually transparent languages* and that CTLA is the set of *contextually transparent lexicalized analyses*.

Corollary 14. *For all $i \geq 1$, we have the following relations:*

(1) $\mathsf{CFL} = \mathsf{CTL}(1)$, $\mathsf{CFLA} = \mathsf{CTLA}(1)$,
(2) $\mathsf{CTL}(i) \subset \mathsf{CTL}(i+1) \subset \mathsf{CTL}$, $\mathsf{CTLA}(i) \subset \mathsf{CTLA}(i+1) \subset \mathsf{CTLA}$,
(3) $\mathsf{CTL} \subset \mathsf{CSL}$.

Proof. We just give outlines of the proofs. With the definitions in mind, claim (1) follows from Theorem 12 and Corollary 13, and claim (2) follows from Corollary 11.

Finally, the separation in (3) can be shown by using the context-sensitive language $L_e = \{ a^{2^n} | n \geq 1 \}$. It is easily seen that the languages in CTL have the constant growth property, which is defined as follows (cf. [5]). Let X be an alphabet, let $L \subseteq X^*$. The language L is said to have the *constant growth property* if there are a constant $c_0 > 0$ and a finite set of positive integers C such that, for all $w \in L$ with $|w| > c_0$, there is a $w' \in L$ with $|w| = |w'| + c$ for some $c \in C$. Obviously, the language L_e does not have the constant growth property, and hence, it does not belong to the class CTL. □

The sensitivity of $\mathsf{hRLWWC}(i)$-automata to the size of their windows can be utilized to essentially refine the hierarchies of CTLA. These refined hierarchies yield a fine classification of syntactic phenomena in lexicalized syntaxes of natural languages.

Recall that the prefix k- indicates the window size. So, k-$\mathsf{CTL}(i)$ is the class $\mathcal{L}_{\mathrm{hP}}(k\text{-}\mathsf{scf\text{-}det\text{-}syn\text{-}hRLWWC}(i))$; analogously for k-$\mathsf{CTLA}(i)$, k-CTLA, and k-CTL. We say that k-$\mathsf{CTL}(i)$ is the set of *k-transparent context-sensitive languages* of degree i, k-$\mathsf{CTLA}(i)$ is the set of *k-transparent context-sensitive lexicalized (sentence) analyses* of degree i, k-CTL is the set of *k-contextually transparent languages*, and k-CTLA is the set of *k-contextually transparent lexicalized analyses*. The next corollary easily follows from Corollary 10.

Corollary 15. *For all $i, k \geq 1$, the following relations hold:*

(1) k-$\mathsf{CTL}(i) \subset k$-$\mathsf{CTL}(i+1) \subset k$-$\mathsf{CTL}$, k-$\mathsf{CTLA}(i) \subset k$-$\mathsf{CTLA}(i+1) \subset k$-$\mathsf{CTLA}$,
(2) k-$\mathsf{CTL}(i) \subset (k+1)$-$\mathsf{CTL}(i)$, k-$\mathsf{CTLA}(i) \subset (k+1)$-$\mathsf{CTLA}(i)$.

For $i, k \geq 1$, we denote the class $\mathcal{L}_{\mathrm{hP}}(k\text{-scf-fin-det-syn-hRLWWC}(i))$ by k-CTL(i)FIN and the class $\mathcal{L}_{\mathrm{A}}(k\text{-scf-fin-det-syn-hRLWWC}(i))$ by k-CTLA(i)FIN. Further, by k-CTLFIN we denote the union $\bigcup_{i\geq 1} k$-CTL(i)FIN and by k-CTLAFIN we denote the union $\bigcup_{i\geq 1} k$-CTLA(i)FIN.

Corollary 16. *For all $i, k \geq 1$, the following relations hold:*

(1) k-CTL(i)FIN $\subset$ k-CTL($i+1$)FIN $\subset$ k-CTLFIN,
k-CTLA(i)FIN $\subset$ k-CTLA($i+1$)FIN $\subset$ k-CTLAFIN,
(2) k-CTL(i)FIN $\subset$ ($k+1$)-CTL(i)FIN *and* k-CTLA(i)FIN $\subset$ ($k+1$)-CTLA(i)FIN.

Proof. These results also follow from Corollary 10. □

5 Conclusion

The 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 and analysis by reduction. The basic correctness preserving property enforces the sensitivity to the degree of synchronization, number of rewrites in a cycle, and to the size of the window.

Thanks to the long time study of the PDT (Prague Dependency Treebank) we believe that class 12-CTLA(2) defined above is strong enough to model the lexicalized surface syntax of Czech, that is, to model the lexicalized sentence analysis based on PDT.

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 [6]). We believe that the lexicalized syntactic analysis of full (four level) FGD can be described by tools very close to 24-CTLA(4).

We stress that our current efforts cover an important gap in theoretical tools supporting computational and corpus linguistics. Chomsky's and other types of phrase-structure grammars and the corresponding types of automata do not support lexical disambiguation, 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, they do not support any type of sensitivity to the size of individual grammar (automata) rules (see several normal forms for context-free grammars, like Chomsky normal form [2]), and, finally, they do not support any kind of classification of finite syntactic constructions of (natural) languages.

On the other hand, in traditional and corpus linguistics, only finite language phenomena can be directly observed. Now the basic and h-proper languages of hRLWWC(i)-automata in strong cyclic form with constraints on the window size allow common classifications of finite phenomena as well as classifications of their infinite relaxations. All these classifications are based on the reduction correctness preserving property and the strong cyclic form. Let us recall that for restarting and list automata the monotonicity means a synonymy for context-freeness. Here we are able to distinguish degrees of non-monotonicity of finite languages (syntactic phenomena), too.

Finally, note that many practical problems in computational and corpus linguistic become decidable when we only consider languages parametrized by the size of the windows, or even easier when they are parametrized by a finite number of reductions.

References

1. Y. Bar-Hillel: *A quasi-arithmetical notation for syntactic description*. Language, 29(1) 1953, pp. 47–58.
2. J. E. Hopcroft and J. D. Ullman: *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1980.
3. P. Jančar, F. Mráz, M. Plátek, and J. Vogel: *Restarting automata*, in FCT'95, H. Reichel, ed., vol. 965 of LNCS, Dresden, Germany, August 1995, Springer, pp. 283–292.
4. T. Jurdziński, F. Mráz, F. Otto, and M. Plátek: *Degrees of non-monotonicity for restarting automata*. Theoretical Computer Science, 369(1–3) 2006, pp. 1–34.
5. L. Kallmeyer: *Parsing Beyond Context-Free Grammars*, Cognitive Technologies, Springer, 2010.
6. M. Lopatková, M. Plátek, and P. Sgall: *Towards a formal model for functional generative description: Analysis by reduction and restarting automata*. The Prague Bulletin of Mathematical Linguistics, 87 2007, pp. 7–26.
7. F. Mráz, F. Otto, and M. Plátek: *Characterizations of LRR-languages by correctness-preserving computations*, in NCMA 2018, Proc., R. Freund, M. Hospodár, G. Jirásková, and G. Pighizzini, eds., Vienna, 2018, Österreichische Computer Gesellschaft, pp. 149–164.
8. M. Plátek: *Two-way restarting automata and j-monotonicity*, in SOFSEM 2001: Theory and Practice of Informatics, 28th Conference on Current Trends in Theory and Practice of Informatics, L. Pacholski and P. Ružička, eds., vol. 2234 of LNCS, Berlin, 2001, Springer, pp. 316–325.
9. M. Plátek and F. Otto: *On h-lexicalized restarting automata*, in AFL 2017, Proc., E. Csuhaj-Varjú, P. Dömösi, and G. Vaszil, eds., vol. 252 of EPTCS, 2017, pp. 219–233.
10. M. Plátek, F. Otto, and F. Mráz: *On h-lexicalized automata and h-syntactic analysis*, in ITAT 2017, Proc., J. Hlaváčová, ed., vol. 1885 of CEUR Workshop Proceedings, 2017, pp. 40–47.
11. M. Plátek, D. Pardubská, and F. Mráz: *On (in)sensitivity by two-way restarting automata*, in ITAT 2018, Proc., S. Krajči, ed., vol. 2203 of CEUR Workshop Proceedings, 2018, pp. 10–17.