LINEAR LIST AUTOMATA WITH A LOOK-AHEAD WINDOW[1]

Martin Plátek
KTIML MFF, Charles University, Prague,
Malostranské nám. 25, CZ-118 00 Praha 1 - Malá Strana, Czech Republic
e-mail: `platek@ksi.ms.mff.cuni.cz`

and

Petr Jančar
Dept. of Computer Science FEI, Technical University of Ostrava (VSB),
17. listopadu 15, CZ - 708 33 Ostrava - Poruba, Czech Republic
e-mail: `Petr.Jancar@vsb.cz`

ABSTRACT

Linear list automata constitute an interesting class of acceptors which has a remarkable 'explicatory power' w.r.t. the analysis of formal and natural languages. We consider a more general model where each automaton has a (fixed-size) look-ahead window attached to its head. We present a refinement of the Chomsky hierarchy, and display the classes of automata which can support bottom-up analysis. The work is meant as a preparation for a further generalization, which will combine linear list automata and restarting automata.

1. Introduction

We follow some previous studies of list automata (see [1], [5]) and introduce a generalized model – a two-way linear list automaton with a look-ahead window (LLA), and its subclasses defined by sets of allowed operations. The model with the lookahead window is more natural and useful for modelling the (reduction and syntactic) analysis of natural languages (compare [6], [7], [8], [3]), and for developing software tools for linguistic modelling.

Remark. The look-ahead window is also an inherent part of definition of the restarting automata, studied e.g. in [2, 4].

An LLA M is a device with a finite state control and a read/write window of fixed size. This window moves (in both directions) on a tape (i.e., list of items) containing a word delimited by sentinels. M can decide (in general nondeterministically) to replace the contents of the window: it can delete some items from the list (moving the head in one or the other direction), insert some items into the list and/or rewrite some items.

We summarize results for several subclasses of LLA obtained by reducing the set of operations, and we also provide some variants and extensions of the Chomsky hierarchy of languages.

2. Definitions

We start by giving a definition of (generalized) linear-list automata.

[1]Supported by the Grant Agency of the Czech Republic, Grant-No. 201/02/1456

A *linear-list automaton with a lookahead window*, LLA for short, is a one-tape machine given by a tuple $M = (Q, \Sigma, \Gamma, ¢, \$, q_0, k, \delta)$, where Q is a finite set of *states*, Σ is a finite *input alphabet*, Γ is a finite *working alphabet* containing Σ; $¢, \$ \notin \Gamma$ are the markers (*sentinels*) for the left and right border of the work space, respectively, $q_0 \in Q$ is the *initial state*, $k \geq 1$ is the size of the *read/write* (look-ahead) *window*, and δ is a finite set of *instructions* of the type

$$(q, u) \longrightarrow (q', z)$$

where $q, q' \in Q$ and $u \in \mathcal{PC}^{\leq k}$; by $\mathcal{PC}^{\leq k}$ we mean the set $(¢ \cdot \Gamma^{k-1}) \cup \Gamma^k \cup (\Gamma^{\leq k-1} \cdot \$) \cup (¢ \cdot \Gamma^{\leq k-2} \cdot \$)$ of (all) *possible contents* of the read/write window of M. The parameter z in the instructions is one of the special symbols

$$M_r, M_l, W(v), d_R(v), d_L(v), I(v), \mathsf{Accept}, \mathsf{Reject}$$

where the indicated symbols are accompanied by a word v satisfying the constraints described below.

A *configuration* of M is a string $\alpha q \beta$ where $q \in Q$, and either $\alpha = \lambda$ (λ denoting the empty word) 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 head scans the first k symbols of β or all of β when $|\beta| < k$; the list item containing the first symbol of β is viewed as the *position* of the head. An *initial configuration* is of the form $q_0 ¢ w \$$, where $w \in \Sigma^*$.

A *computation* (i.e., a sequence of configurations) is generated by the instruction set δ, as explained further:

1. A *move-right step* $(q, u) \longrightarrow (q', M_r)$ assumes $u \neq \$$. If M is in state q and is scanning u in its read/write window then this step causes M to shift the (position of the) read/write window by one to the right and to enter state q'.
2. A *move-left step* $(q, u) \longrightarrow (q', M_l)$ assumes $u \notin ¢ \cdot \Gamma^*$. It causes M to shift the (position of the) read/write window by one to the left and to enter state q'.
3. A *rewrite step* $(q, u) \longrightarrow (q', W(v))$ assumes that $|v| = |u|$ and the sentinels are at the same positions in u and v (if at all). It causes M to replace the contents u of the read/write window by the string v, and to enter state q'. The head does not change its position.
4. A *delete-right step* $(q, u) \longrightarrow (q', d_R(v))$ assumes that v is a proper subsequence of u, containing all sentinels in u. It causes M to replace u with v (by deleting excessive items-symbols) and to enter state q'; the new position of the head is on the first item of v (the contents of the window is thus 'completed' from the right; the position distance to the right sentinel decreases).
5. A *delete-left step* $(q, u) \longrightarrow (q', d_L(v))$ assumes that v is a proper subsequence of u, containing all sentinels in u. It causes M to replace u with v (by deleting excessive items-symbols), to enter state q' and to shift the head position by $|u| - |v|$ items to the left – but to the left sentinel $¢$ at most. (Thus the contents of the window is 'completed' from the left; the distance to the left sentinel decreases if the head position was not at $¢$ already.)
6. An *insert step* $(q, u) \longrightarrow (q', I(v))$ assumes that u is a proper subsequence of v (keeping the obvious sentinel constraints). It causes M to replace the scanned u by v (by inserting the relevant items), and to enter state q'. The head position is shifted by $|v| - |u|$ to the right (the distance to the left sentinel increases).
7. An *accept step* $(q, u) \longrightarrow (q', \mathsf{Accept})$ causes M to halt and accept. The state q' can provide a type of accepting.
8. A *reject step* $(q, u) \longrightarrow (q', \mathsf{Reject})$ causes M to halt and reject. The state q' can provide a type of rejecting.

We consider only finite computations, which end either by accepting or by rejecting.

In general, automaton M is *nondeterministic*, that is, there can be two or more instructions with the same left-hand side (q, u), and thus there can be more than one computation for an input word. If this is not the case, the automaton is *deterministic*.

An input *word* $w \in \Sigma^*$ *is accepted by* M, if there is a computation which, starting with the initial configuration $q_0¢w\$$, finishes by executing an Accept instruction. By $L(M)$ we denote the language consisting of all input words accepted by M; we say that M *accepts (recognizes)* $L(M)$. By $L(M, q)$, where $q \in Q$ we denote the language consisting of all input words accepted by M in the state q; we say that M *accepts (recognizes)* $L(M, q)$ *by* q.

By restrictions on the set of operations, we can naturally define some subclasses of LLA that are relevant for our study; the subclasses are denoted by the allowed operations (the accepting and rejecting operations are always implicitly assumed). The prefix det- is used to denote the property of being deterministic. For a class A of automata, $\mathcal{L}(\mathsf{A})$ denotes the class of languages recognizable by the automata from A. Thus, for example, $\mathcal{L}(\mathsf{det}\text{−}\{M_r, d_L\})$ means the class of languages recognizable by deterministic LLA which can use only the operations M_r, d_L.

3. Hierarchies of LL-automata

At first we give several characterizations of the classes of languages of the Chomsky hierarchy and of its deterministic variant; a class is characterized by some minimal and maximal sets of operations. We thus extend and generalize the observations from [1].

Theorem 1 *The following classes of languages coincide with the class of regular languages:* $\mathcal{L}(\{M_r\})$, $\mathcal{L}(\{M_r, M_l\})$, $\mathcal{L}(\{M_r, W\})$, $\mathcal{L}(\{d_R\})$, $\mathcal{L}(\{M_r, d_R\})$, $\mathcal{L}(\{M_r, W, d_R\})$.

Theorem 2 *The following classes of languages coincide with CFL (the class of context-free languages):* $\mathcal{L}(\{M_r, W, d_L\})$, $\mathcal{L}(\{M_r, W, d_L, d_R, I\})$.

Theorem 3 *The following classes of languages coincide with CSL (the class of context-sensitive languages):* $\mathcal{L}(\{M_r, M_l, W\})$, $\mathcal{L}(\{M_r, M_l, W, d_R, d_L\})$ *(inserting is not allowed).*

Theorem 4 *The class* $\mathcal{L}(\{M_r, M_l, W, I\})$ *coincides with the class RE (of recursively enumerable languages).*

Note. The deterministic variant of the Chomsky hierarchy is characterized by the deterministic variants of the above mentioned classes (the prefix det- would be used).

3.1. Addition

The next theorem presents a new result (because of the added feature – the lookahead window – we can not derive the result from [5].

Theorem 5 $\mathcal{L}(\mathsf{det}\text{−}\{M_r, d_L\})$ *is a proper subset of* DCFL *(deterministic CFL).*

Proof sketch. Any deterministic LLA M can be simulated by a deterministic pushdown automaton in a straightforward way. On the other hand, the language

$$L = \{\, a^n bwcw^R u \mid u, w \in \{a, b\}^*, |u| = 2n \,\}$$

is obviously in DCFL but can be shown not recognizable by any deterministic LLA using only operations M_r, d_L.

We can note that the above language L can be recognized by a nondeterministic $\{M_r, d_L\}$ − *automaton*; the question if DCFL $\subseteq \mathcal{L}(\{M_r, d_L\})$ remains open.

4. Analyzers

Since pure recognizers (acceptors) do not have a direct use for linguistic modelling, we introduce (informally) the notion of analyzers in order to modell (bottom-up) syntactic and/or reduction analysis (see [3]). In fact, the analyzers should serve both as recognizers and regulated analytical grammars. We consider the analyzers only for languages inside CSL; the practical use in linguistc modelling does not seem to require going outside CSL.

We say that a LLA M *without* the operation *insert* is an *analyzer* iff it can (directly) accept only when the whole tape (list of items) is in the window, and on longer lists it performes the so called *reductions* described further: M computes in certain phases; from the start to the first delete operation, from the first delete operation to the second delete operation and so on. We call such phases *delete-cycles* or simply *cycles*. The change performed on the tape of M during a cycle is called a *reduction*.

Corollary 6 *The following classes of (deterministic) analyzers characterize regular languages by the minimal (maximal) set of allowed operations:* $\{d_R\}$*-analyzers, and* $\{d_R, W\}$*-analyzers.*

Moreover, for any regular language L *there is a* $\{d_R\}$*-analyzer* M *where* $L = L(M)$ *and* M *has the following property: for each* $w \in L(M)$*, each configuration of the computation of* M *contains a word from* $L(M)$ *on the tape.*

Further, for any regular grammar G *there is a* $\{d_R, W\}$*-automaton, which simulates the bottom-up analysis by* G *on the tape.*

Note. We can see that the (use of the non-trivial) look-ahead windows allow to characterize also the regular languages by certain classes of analyzers.

Corollary 7 *The following classes of analyzers characterize CF-languages:* $\{M_r, W, d_L\}$*-analyzers,* $\{M_r, W, d_L, d_R\}$*-analyzers.*

Moreover, for any CF-grammar (or categorial grammar) G *there is a* $\{M_r, d_L, W\}$*-automaton* M*, which simulates the bottom-up analysis by* G *on its tape.*

Corollary 8 *The analyzers which do not use the insert operation characterize CS-languages.*

Moreover, for any CS-grammar G *there is an LLA* M *which simulates the bottom-up analysis by* G *on its tape (and does not use the operation insert).*

Corollary 9 *The following classes of deterministic analyzers characterize deterministic CF-languages: det-*$\{M_r, W, d_L\}$*-analyzers , det-*$\{M_r, W, d_L, d_R\}$*-analyzers.*

Corollary 10 *For any* $L \in DCFL$ *there is a det-*$\{M_r, M_l, d_L\}$*-analyzer* M *where* $L = L(M)$ *and* M *has the following property: for each* $w \in L(M)$*, each configuration of the computation of* M *contains a word from* $L(M)$ *on the tape.*

Note. In linguistic terminology, the previous corollary means that for any deterministic context-free language there is an analyzer which performes analysis by reduction (see [3]).

5. Conclusion

This work is a preparation for a further generalization of LLA, which will create a common framework for linear list automata and restarting automata. We plan to use this framework e.g. for explaining of the principles of an analyzer, which will be build on the basis of the 'Functional Generative Description of Czech' (see [9, 10]). It is also planned to create a software environment for the development of analyzers of natural languages, which is based on such principles. We also believe that the generalized linear list automata create a suitable theoretical tool for the study of the degrees of non-determinism on different levels.

References

[1] M.P. Chytil, M. Plátek, J. Vogel: A Note on the Chomsky hierarchy, Bulletin of the EATCS 27, 1985

[2] P. Jančar, F. Mráz, M. Plátek, J. Vogel: On Monotonic Automata with a Restart Operation. *J. Autom. Lang. Comb.* 4 (1999) 287-311.

[3] Plátek, M., Lopatková, M., Oliva, K. 2003. Restarting Automata: Motivations and Applications. In: Proceedings of the workshop "Petrinetze" and 13. Theorientag 'Formale Sprachen und Automaten',(ed. Holzer, M.), Institut für Informatik, 'Technische Universität München, pp. 90-96.

[4] F. Otto: Restarting automata and their relations to the Chomsky hierarchy. In: Z. Esik, Z. Fülöp (eds.): *Developments in Language Theory, Proceedings of DLT'2003*, LNCS 2710, Springer, Berlin (2003).

[5] M. Plátek, J.Vogel: Deterministic List automata and Erasing Graphs. The Prague Bulletin of Mathematical Linguistics 45, (1986).

[6] M. Plátek, B. Tichá: List automata as Dependency Parsers, PBML 50, 1988, pp. 71–87

[7] K. Oliva, M. Plátek: List Automata with Syntactically Structured Output, COLING'88, Budapest, Hungary, 1988, Vol. 2, pp. 498–500

[8] M. Plátek: A Note on Parsers, The Prague Bulletin of Mathematical Linguistics 54, 1990, pp. 25–36

[9] Sgall, P. 1967. Generativní popis jazyka a česká deklinace. Academia, Praha (in Czech).

[10] Sgall, P., Hajičová, E., Panevová, J. 1986. The Meaning of the Sentence in Its Semantic and Pragmatic Aspects (ed. by J. Mey), Dordrecht:Reidel and Prague:Academia.