A linear time extension of deterministic pushdown automata*

Anders Søgaard
Center for Language Technology
University of Copenhagen
Njalsgade 140–142
DK-2300 Copenhagen S
soegaard@hum.ku.dk

Abstract

A linear time extension of deterministic pushdown automata is introduced that recognizes all deterministic context-free languages, but also languages such as $\{a^n b^n c^n \mid n \geq 0\}$ and the MIX language. It is argued that this new class of automata, called λ-acyclic read-first deterministic stack+bag pushdown automata, has applications in natural language processing.

1 Introduction

This article presents a linear time extension of deterministic pushdown automata (DPAs). DPAs have numerous applications in computer science, as many programming languages can be recognized by such automata, but they are not expressive enough for natural language parsing. There are at least two reasons for this; namely, that natural languages are heavily ambiguous, and that natural languages exhibit non-context-free constructions. Deterministic stack+bag pushdown automata introduce a limited form of nondeterminism, since information can be stored in bags. The bag construction also gives us limited context-sensitivity. It is argued that at least for some of the complex constructions in natural languages the degrees of nondeterminism and context-sensitivity are adequate. Our example in Sect. 6 concerns German scrambling.

Thanks to Thomas Hanneforth for pointing out previous work on ϕ-transitions to me. This work was done while the author was a Senior Researcher at the Dpt. of Linguistics, University of Potsdam, supported by the German Research Foundation.

2 Formal preliminaries

A **stack+bag pushdown automaton** (SBPA) is a 6-tuple $P = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ where Q is a finite set of states, Σ the finite alphabet, Γ the finite stack symbols, $q_0 \in Q$ the initial state, $F \subseteq Q$ the final states, and $\delta \subseteq Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \times Q \times (\Gamma \cup \{\lambda\}) \times \{\{\gamma_1, \ldots, \gamma_n\}_M \mid \gamma_1 \ldots \gamma_n \in \Gamma, n \geq 0\}$ a finite set of transitions, where $\{\ldots\}_M$ is a bag or a multiset, i.e. $\{\{\gamma_1, \ldots, \gamma_n\}_M \mid \gamma_1, \ldots, \gamma_n \in \Gamma, n \geq 0\}$ is the set of multisets over elements of Γ.

The elements of δ, e.g. $\delta(q_i, a, A) = (q_j, \lambda, \{A\}_M)$, are transitions between states augmented with instructions to read or process string elements from the alphabet and pop and push stack symbols from the stack and the bag. The transition $\delta(q_i, a, A) = (q_j, \lambda, \{A'\}_M)$ is, for example, an instruction to read a, move from q_i to q_j and pop a stack symbol A from either the stack or the bag and push a symbol A' into the bag. If the transition had been $\delta(q_i, a, A) = (q_j, A', \emptyset_M)$ A' had been pushed onto the stack instead of into the bag.

The notion of an instantaneous description $(q, w, \gamma, \gamma') \in Q \times \Sigma^* \times \Gamma^* \times \{\{\gamma_1, \ldots, \gamma_n\}_M \mid \gamma_1 \ldots \gamma_n \in \Gamma, n \geq 0\}$ is introduced to define the language of a SBPA, where q is the state the SBPA is currently in, w the input string still to be processed, γ the contents of the stack, and γ' the contents of the bag. The derivability relation is the transitive, reflexive closure ($\vdash^*$) of the following binary relation over the class of instantaneous descriptions (ID), $\vdash \subseteq$ ID $\times$ ID, where

- $(q, xw, z\gamma, \gamma') \vdash (q', w, \alpha\gamma, \gamma')$ if $(q', \alpha, \emptyset_M) \in \delta(q, x, z)$, [pop z from stack, push α to stack]
- $(q, xw, z\gamma, \gamma') \vdash (q', w, \gamma, \alpha' \cup \gamma')$ if

Kristiina Jokinen and Eckhard Bick (Eds.)
NODALIDA 2009 Conference Proceedings, pp. 182–189

$(q', \lambda, \alpha') \in \delta(q, x, z)$, [pop z from stack, push α' to bag]

- $(q, xw, \gamma, \{z\}_M \cup \gamma') \vdash (q', w, \alpha\gamma, \gamma')$ if $(q', \alpha, \emptyset_M) \in \delta(q, x, z)$, and [pop z from bag, push α to stack]
- $(q, xw, z\gamma, \{z\}_M \cup \gamma') \vdash (q', w, \gamma, \alpha' \cup \gamma')$ if $(q', \lambda, \alpha') \in \delta(q, x, z)$, [pop z from bag, push α' to bag],

with $x \in \Sigma \cup \{\lambda\}$, $z \in \Gamma \cup \{\lambda\}$, $\alpha \in \Gamma^*$, and $\alpha' \in \{\{\gamma_1, \ldots, \gamma_n\}_M \mid \gamma_1 \ldots \gamma_n \in \Gamma, n \geq 0\}$. The definition of the language of a SBPA S is now as follows:

$$L(S) = \{w \mid (q_0, w, \lambda, \emptyset_M) \vdash^* (q, \lambda, \lambda, \emptyset_M) \wedge q \in F\}$$

The languages that can be recognized by SBPAs are called stack+bag pushdown languages.

A SBPA S is called **deterministic** if for all possible instantaneous descriptions over S at most one transition in S is applicable. The languages that can be recognized by deterministic SBPAs are called deterministic stack+bag pushdown languages. Note that it can be assumed without loss of generalization that a deterministic SPBA for any state $q \in Q$ contains no λ-transitions or cycles of λ-transitions from q to q.

If a transition that reads an element of the alphabet is always chosen over a transition that reads λ, a read-first strategy is said to have been adopted. A SBPA S is said to be **read-first deterministic** if it is always clear what transition to apply under a read-first strategy, i.e. if for all instantaneous descriptions over S at most one transition of the form $(q, a, A) = \ldots$ where $a \in \Sigma$, and at most one transition of the form $(q, \lambda, A') = \ldots$, is applicable. If an automaton is *not* read-first deterministic it thus means that there are two transitions in δ of the form:

$$\begin{aligned} \delta(q_i, a, A) &\in (q'_i, \ldots, \ldots) \\ \delta(q_i, a, A'') &\in (q''_i, \ldots, \ldots) \end{aligned}$$

or two transitions of the form:

$$\begin{aligned} \delta(q_j, \lambda, A) &\in (q'_j, \ldots, \ldots) \\ \delta(q_j, \lambda, A') &\in (q''_j, \ldots, \ldots) \end{aligned}$$

and it is either *not* the case that A, A' never occur in the same bag, or it is not the case that A can never be the top element with A' in the bag, or vice versa, or both. The languages that can be recognized by read-first deterministic SBPAs running in read-first mode are called read-first deterministic stack+bag pushdown languages.[1] Obviously, the read-first deterministic stack+bag pushdown languages include the deterministic stack+bag pushdown languages.

Finally, we say that a read-first deterministic stack+bag pushdown automaton is λ-**acyclic** if it is impossible to apply a transition

$$\delta(q, \lambda, \ldots) \in \ldots$$

more than once without reading an element from the input string first. The languages that can be recognized by λ-acyclic read-first deterministic SBPAs are called λ-acyclic read-first deterministic stack+bag pushdown languages.

3 Related work

This section compares our work to three rather disparate strands of research, namely (i) work on ϕ-transitions in the automata literature, (ii) deterministic parsing strategies for shift-reduce parsers and (iii) recent work on linguistically motivated extensions of tree-adjoining grammar. The first two comparisons serve to provide a bit of background on the read-first strategy. The third provides a bit of background on our use of bags.

Aho and Corasick (1975) design a class of automata for bibliographic search in which transitions are replaced by a function $g : Q \times \Sigma \rightarrow Q$ that maps pairs of states and input symbols into states or the failure message *fail*. There are no empty transitions, i.e. $\lambda \notin \Sigma$; instead a failure function $f : Q \rightarrow Q$ is consulted whenever g returns *fail*. It is not difficult to see that this is equivalent to a read-first strategy.

The read-first strategy is also related to work on deterministic shift-reduce parsers, e.g. Nivre (2003) for projective dependency grammars. A projective dependency grammar annotates a finite string $w_1 \ldots w_n$ with directed edges E, i.e. governor-dependent relations, such that the string positions, decorated by words, and the edges form an acyclic connected graph $G = \langle \{w_1 \ldots w_n\}, E \rangle$ in which each node has at most one governor and the edges are wellnested. Call such a graph a projective dependency graph. The deterministic shift-reduce parser introduced in Nivre (2003) begins with a 3-tuple $\langle \mathbf{nil}, \lambda, \emptyset \rangle$, in which the first element is the empty stack and the third element is the empty graph, and

[1] Below it is assumed that read-first deterministic SPBAs always run in read-first mode.

terminates when the string has been read, i.e. in $\langle T, w_1 \dots w_n, G \rangle$ where T is a possibly non-empty stack. The string is accepted if G is a projective dependency graph. The algorithm applies four transitions to these states in a prioritized way, i.e. **Left-Arc** first if applicable, otherwise **Right-Arc**, then **Reduce**, and finally, if nothing else works, **Shift**.[2]

- The first transition **Left-Arc** adds an edge to the graph that encodes that the first element on the stack n is a dependent of the initial position n' in the substring still to be processed. The edge is licensed by a grammar rule that relates the two words that decorate the nodes in question in this way, i.e. R is a set of such word-to-word rules. The requirement that n is not governed by anything else is also necessary. The node n is removed from the stack to avoid cycles.

- The second transition **Right-Arc** adds an edge to the graph that encodes that the initial position in the substring to be processed is a dependent of the first element on the stack. The edge is again licensed by the grammar, and it is required that the dependent is not already governed. The dependent node is immediately shifted; again, to prevent cycles.

- The third transition **Reduce** simply pops the first element of the stack. Note that an element can only be popped this way if it is already assigned a governor.

- The fourth transition **Shift** pushes the next position onto the stack.

While this is technically a bit different from the read-first strategy adopted in our proposal, the intuition is the same: The constructive transitions **Left-Arc** and **Right-Arc** are tried out first, and only if no constructive transitions are applicable can the λ-transitions be applied. The underlying if-then-else structure means that the procedure remains deterministic. The three algorithms introduced in Nivre (2003) all terminate in linear time.

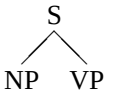
Other related classes of automata include extended pushdown automata (Vijay-Shanker, 1987), weakly equivalent to tree-adjoining grammars, which use nested stacks to provide an additional control layer, and thread automata (Villemonte de La Clergerie, 2002), weakly equivalent to simple range concatenation grammar. These classes are not discussed here, but it should be noted that they were constructed to capture the expressivity of linguistic theories, while the class of automata introduced here "cross-cuts the Chomsky hierarchy" in a non-standard way. It restricts expressivity in some ways (by read-first determinism and λ-acylicity), but adds expressivity in other ways (by introducing a bag).

In the conclusion, once we have established the necessary results, our proposal is also compared to Bertsch and Nederhof (1999). Bertsch and Nederhof (1999) define another linear time extension of deterministic pushdown automata, but their extension remains context-free.

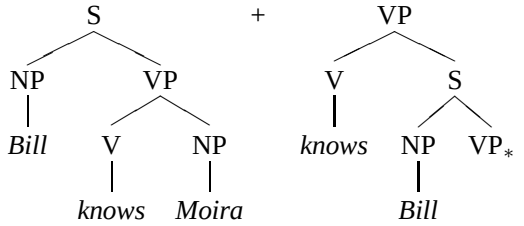
Finally, our use of bags is related to the use of sets of elementary trees in certain linguistically motivated extensions of tree-adjoining grammar, incl. Becker et al. (1991) and Lichte (2007). A very brief summary of tree-adjoining grammar: Tree substitution grammar is a variation over context-free grammar. Instead of production rules of the form

$$S \quad \rightarrow \quad NP\ VP$$

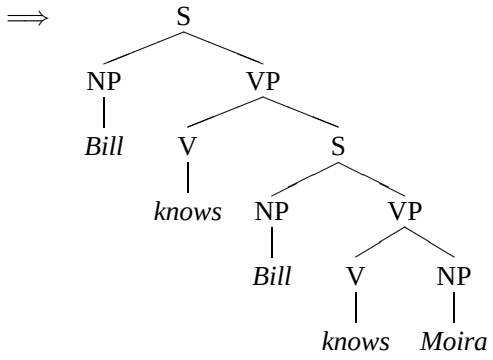
tree fragments of the following form are introduced:



In derivation, trees with root labels A are plugged into trees with leaf nodes labeled by A. If a tree is obtained with root label S (the start symbol) and all leaf nodes are labeled by terminal symbols, the tree is a parse of its yield. Tree-adjoining grammar extends this context-free formalism by an operation on trees called adjunction, e.g.:



[2] In fact this simple set-up is only used to obtain a baseline in Nivre (2003). Two superior parsing algorithms are introduced that complicates this simple scenario by introducing limited lookahead. The details are unimportant for our purposes.



If an auxiliary tree t, with a root node and a leaf node both labeled A, is adjoined at some node n also labeled A in a derived tree t', the subtree s' (of t') rooted at n is replaced by t, and s' is then inserted at the leaf node of t.

The adjunction operator buys us limited context-sensitivity. In particular, tree-adjoining grammar is weakly equivalent to linear indexed grammar (Vijay-Shanker and Weir, 1994a) or level 2 control grammars (Weir, 1992). The universal recognition problem and the parsing problem can both be solved in time $\mathcal{O}(|G|n^6)$ (Vijay-Shanker and Weir, 1994b).

The formalism presented in Becker et al. (1991) is called (nonlocal) multicomponent tree-adjoining grammar (MCTAG). In fact, MCTAG comes in a number of varieties, but the intuition behind all of them is to introduce sets of auxiliary trees rather than just singular trees. Scrambling is now obtained when a set of multiple auxiliary trees is used in a relatively unconstrained context. The set must be emptied, i.e. each element must be adjoined, but the adjunctions can in unconstrained contexts result in any possible permutation of the yields of the auxiliary trees. See also Kallmeyer and Yoon (2004) for an analysis of scrambling in Korean in MCTAG.

Lichte (2007) replaces sets of auxiliary trees with 2-tuples $\langle t, \{a_1, \ldots, a_n\}\rangle$ where t can be any kind of tree, and $a_1, \ldots, a_m$ are auxiliary trees. This separation is similar to what is adopted in our analysis of German scrambling below.

4 Weak generative capacity

Lemma 4.1. *The stack+bag pushdown languages strictly include the context-free languages.*

Proof. The languages that can be recognized by pushdown automata, i.e. stack+bag pushdown automata without bags, are exactly the context-free languages (Chomsky, 1962). The languages that can be recognized by pushdown automata can be recognized by stack+bag pushdown automata, by definition, and thus stack+bag pushdown automata recognize context-free languages. It is not difficult to show that the inclusion is strict either. Simply note that the SBPA $S_1 = \langle\{q_0, q_1, q_2, q_3\}, \{a, b, c\}, \{A, B, C\}, \delta, q_0, \{q_3\}\rangle$ with the following transitions δ generates the language $L(S_1) = \{a^n b^n c^n \mid n \geq 0\}$ which is non-context-free by the Bar-Hillel pumping lemma (Aho and Ullman, 1972):

$$\begin{aligned}
\delta(q_0, \lambda, \lambda) &\in \delta(q_0, \lambda, \{A, B, C\}_M)\\
\delta(q_0, \lambda, \lambda) &\in \delta(q_3, \lambda, \emptyset_M)\\
\delta(q_0, a, \lambda) &\in \delta(q_1, \lambda, \emptyset_M)\\
\delta(q_1, b, \lambda) &\in \delta(q_2, \lambda, \emptyset_M)\\
\delta(q_2, c, \lambda) &\in \delta(q_3, \lambda, \emptyset_M)\\
\delta(q_1, a, A) &\in \delta(q_1, \lambda, \emptyset_M)\\
\delta(q_2, b, B) &\in \delta(q_2, \lambda, \emptyset_M)\\
\delta(q_3, c, C) &\in \delta(q_3, \lambda, \emptyset_M)
\end{aligned}$$

The automaton pushes an arbitrary number of A, B, C's into the bag in transitions from state q_0 to state q_0. Since the stack symbols are pushed into the bag simultaneously, it is guaranteed that the bag always contains the same number of A's, B's and C's in state q_0. Unless the automaton recognizes the empty string, in which case it does not push any stack symbols into the bag, but proceeds immediately to the final state q_3, it will first have to remove an A from the bag by moving into state q_1. In fact it has to remove *all* A's, since if it moves to q_2 by removing a B, it is no longer possible to remove the A's that remain, and the input string will not be recognized. Once the A's have been removed, it proceeds to q_2 to remove B's, and so on. Note that the automaton reads an a, resp. b or c, whenever it removes an A, resp. B or C. Since it is guaranteed that the bag always contains the same number of A's, B's and C's in state q_0, the strings that are recognized by this automaton will be of the form $a^n b^n c^n$ for $n \geq 0$. Since the stack+bag pushdown languages include the context-free languages and at least one language that is not context-free, namely $\{a^n b^n c^n \mid n \geq 0\}$, it follows that they strictly include the context-free languages. □

It is not difficult to see how the automaton in the proof of Lemma 4.1 can be modified to recognize the MIX language, i.e. the language that consists of any permutation of a string in $\{a^n b^n c^n \mid n \geq 0\}$. This is of some interest, since

the MIX language is conjectured not to be recognized by any linear indexed grammar (Gazdar, 1988).[3] The context-free languages constitute the first level of the hierarchy of controlled languages (Weir, 1992), the linear indexed languages the second level. Lemma 4.3 below relates the stack+bag pushdown languages to the entire hierarchy and shows that they are not included in the kth level of the hierarchy for any fixed k either.

Lemma 4.2. *The stack+bag pushdown languages include the MIX language.*

Proof. The SBPA $S_2 = \langle\{q_0\}, \{a, b, c\}, \{A, B, C\}, \delta, q_0, \{q_0\}\rangle$ with the following transitions δ generates the MIX language:

$$\begin{array}{rcl}\delta(q_0, \lambda, \lambda) & \in & \delta(q_0, \lambda, \{A, B, C\}_M)\\ \delta(q_0, a, A) & \in & \delta(q_0, \lambda, \emptyset_M)\\ \delta(q_0, b, B) & \in & \delta(q_0, \lambda, \emptyset_M)\\ \delta(q_0, c, C) & \in & \delta(q_0, \lambda, \emptyset_M)\end{array}$$

In the light of our description of the automaton in Lemma 4.1 it should be easy to see how the automaton works. It recognizes the empty string, since the initial state is also a final state, and it recognizes all permutations of strings in $\{a^n b^n c^n \mid n \geq 0\}$, since the transitions that forced us to first remove A's, then B's, and so on, in the above, have been removed. □

Note that none of the two automata S_1, S_2 in the lemmas above are deterministic. Consider, for instance, the instantaneous descriptions $(q_0, aabbcc, \lambda, \emptyset_M)$ when S_1 reads the string $aabbcc$. In this case there are three applicable transitions (the first three on the list).

Note also that the two automata are both read-first deterministic. Another language that is nondeterministic and read-first deterministic is the language of palindromes $\{ww^R \mid w \in \Sigma^*\}$.

Finally, the automaton for the MIX language is λ-acyclic, but the one for $\{a^n b^n c^n \mid n \geq 0\}$ isn't. It is easy to see that there are equivalent stack+bag pushdown automata for $\{a^n b^n c^n \mid n \geq 0\}$ that are λ-acyclic. Consider, for instance, the SBPA $S_3 = \langle\{q_0, q_1, q_2\}, \{a, b, c\}, \{B, C\}, \delta, q_0, \{q_2\}\rangle$ with the following transitions δ:

$$\begin{array}{rcl}\delta(q_0, \lambda, \lambda) & \in & \delta(q_2, \lambda, \emptyset_M)\\ \delta(q_0, a, \lambda) & \in & \delta(q_0, \lambda, \{B, C\}_M)\\ \delta(q_0, b, B) & \in & \delta(q_1, \lambda, \emptyset_M)\\ \delta(q_1, b, B) & \in & \delta(q_1, \lambda, \emptyset_M)\\ \delta(q_1, c, C) & \in & \delta(q_2, \lambda, \emptyset_M)\\ \delta(q_2, c, C) & \in & \delta(q_2, \lambda, \emptyset_M)\end{array}$$

When the automaton reads an a it pushes a B and a C into the bag. The first input b takes the automaton to its second state q_1 in which subsequent bs (if any) are read; the first input c takes the automaton to its final state q_2 in which subsequent cs (if any) are read. Each reading of a b, resp. c, removes a B, resp. C, from the bag. Consequently, for each a there is exactly one b and one c. The transitions between the three states ensure that the as precede the bs, and that the bs precede the cs.

Lemma 4.3. *The stack+bag pushdown languages are not included in the kth level of the hierarchy of control languages (Weir, 1992) for any fixed k.*

Proof. It is known that there exists a k-level control grammar for the language $\{a_1^n \ldots a_{2k}^n \mid n \geq 0\}$, but not for $\{a_1^n \ldots a_{2k+1}^n \mid n \geq 0\}$ (Palis and Shende, 1995). It is easy to see by inspection of the automaton S_1 that we can always build a SBPA that accepts $\{a_1^n \ldots a_{2k+1}^n \mid n \geq 0\}$ for any fixed k. □

It can be seen in the same way by inspection of the automaton S_3 that the same holds for λ-acyclic read-first deterministic stack+bag pushdown languages.

Corollary 4.4. *The λ-acyclic read-first deterministic stack+bag pushdown languages are not included in the kth level of the hierarchy of control languages (Weir, 1992) for any fixed k.*

Note also that the λ-acyclic read-first deterministic stack+bag pushdown languages include the deterministic context-free ones, since a deterministic pushdown automaton will never visit a λ-transition more than once without processing a string, since, equivalently, for any state $q \in Q$ it contains no λ-transitions or cycles of λ-transitions from q to q. This observation is stated as a lemma for further reference:

Lemma 4.5. *The λ-acyclic read-first deterministic stack+bag pushdown languages include the deterministic context-free languages.*

5 Complexity

In this section it is shown that the universal recognition problem of λ-acyclic read-first determinis-

[3] Bill Marsh's stronger original conjecture, from an unpublished 1985 ASL paper, is that the MIX language is not even an indexed language.

tic stack+bag pushdown automata can be solved in linear time.

Theorem 5.1. *The universal recognition problem of read-first deterministic stack+bag pushdown automata can be solved in time quadratic in the length of the input string, and in linear time for λ-acyclic ones.*

Proof. Consider the universal recognition problem if for some string $w_1 \dots w_n$ and some read-first stack+bag pushdown automata $P = \langle Q, \Sigma, \Gamma, \delta, q_0, F\rangle$ with start ID $(q_0, w_1 \dots w_n, \lambda, \emptyset_M)$. The string $w_1 \dots w_n$ is recognized by P iff the procedure in Figure 1 returns *true* on the start ID when called recursively.

Under the assumption that the procedure halts and outputs *false* when it reads the same state and string for the nth time,[4] this procedure will loop at most n^2 many times if P is read-first deterministic. If P is also λ-acyclic, the number of loops required is at most $2n$.

Step 2 can be done in time $\mathcal{O}(|F|)$, and **read** and **print** are obviously linear time. The complicated steps are 4 and 7. The reason is of course that $\vdash$ has not been computed, so it must be checked if there is a transition in δ that licenses the relevant derivation, say

$$(q, w_i \dots w_n, \gamma_1, \gamma_2) \vdash (q', w_{i+1} \dots w_n, \gamma_1', \gamma_2')$$

This is linear in $|\delta|$, but on a naïve implementation it may also depend on the size of the bag, which again depends on the length of the input string and the maximum number of stack symbols a transition can push to the bag. Consequently, on such an implementation, the overall runtime would be cubic in the length of the input string for unrestricted read-first SBPAs, and quadratic for λ-acyclic ones. A more efficient option is to keep a table of stack symbols with numerical counters of size $|\Gamma|$. If a stack symbol A is pushed to the bag the value of the counter in column A is increased by one; if A is popped the value decreases. The overall runtime, with such a counter, is in $\mathcal{O}(n^2 \times |\Gamma| \times |\delta| \times |F|)$ for otherwise unrestricted read-first deterministic pushdown automata, and in $\mathcal{O}(n \times |\Gamma| \times |\delta| \times |F|)$ for λ-acyclic ones.[5] □

[4] This move is safe. It is left for the reader to verify this.

[5] One of our reviewers observe that the bit complexity of this algorithm is actually $\mathcal{O}(n \log n \times |\Gamma| \times |\delta \times |F|)$ for λ-acyclic read-first deterministic pushdown automata. The distinction here is comparable to bit complexity vs. word complexity in graph theory.

6 Scrambling in German

This section presents an indication that it is possible to analyze German scrambling phenomena in λ-acyclic read-first deterministic SBPAs in ways similar in spirit to what has been presented in Becker et al. (1991) and Lichte (2007). Unlike these formalisms, both extensions of tree-adjoining grammars, λ-acyclic read-first deterministic SBPAs are computationally efficient. The formalism used in Becker et al. (1991), called non-local MCTAG, recognizes NP-complete languages (Rambow and Satta, 1992).[6]

The phenomenon of scrambling is illustrated by the example in Figure 2:

The point in this case is that all possible permutations of the four NPs are grammatical in German. They can be scrambled in any way. One of the relevant syntactic construction involved in scrambling is of the following form, ignoring the internal syntax of the verb cluster:

$$\text{dass } permute(\text{NP}_1 \dots \text{NP}_{n-1}\ \text{NP'})\ \text{V}_1 \dots \text{V}_n$$

where NP_i is the object complement of V_i for $1 \leq i < n$. The NP' is the subject of the finite verb V_n. This construction is recognized by the SBPA $S_4 = \langle \{q_0, q_1, q_2, q_3\}, \{NP_1, \dots, NP_{n-1}, NP', V_1, \dots V_n\}, \{NP_1, \dots, NP_{n-1}, NP', V_1, \dots V_n\}, \delta, q_0, \{q_3\}\rangle$ with the following transitions δ:[7]

[6] Its set-local variant (Weir, 1988), which may not suffice for analyses of scrambling (Rambow et al., 1992), though see Xia and Bleam (2000) for discussion, is weakly equivalent to simple range concatenation grammar whose universal recognition problem can be solved in deterministic time $\mathcal{O}(|G|n^{6k})$, where k, intuitively, is the number of (possibly scrambled) complements a verb may take. The complexity is to be precise $\mathcal{O}(|G|n^{2k(l+1)})$ where l is the maximum number of RHS nonterminals/predicates. See Boullier (1998) for an example of a parsing algorithm, applicable via the conversion described in Weir (1988). Set-local MCTAG is more succinct than simple range concatenation grammar, however, and its universal recognition problem can be shown to be NP-complete (Søgaard et al., 2007). The formalism used in Lichte (2007) has also been shown to be NP-complete (Søgaard et al., 2007).

[7] The indeces here should not lead the reader to think that we are not accounting for an unbounded number of dependencies. If the NPs in the above example are all the same, say *John*, and $n-1$ of the verbs are *let*, except the transitive, most embedded one, our automaton only needs two transitisions for reading NPs (no matter how long the sentence is).

1. **read** $(q, w_i \dots w_n, \gamma_1, \gamma_2)$
2. **if** $q \in F, w_i \dots w_n = \lambda, \gamma_1 = \lambda, \gamma_2 = \emptyset_M$
3. **print** *true*
4. **elsif** $(q, w_i \dots w_n, \gamma_1, \gamma_2), (q', w_{i+1} \dots w_n, \gamma_1', \gamma_2') \in \vdash$
5. **print** $(q', w_{i+1} \dots w_n, \gamma_1', \gamma_2')$
6. **return**
7. **elsif** $(q, w_i \dots w_n, \gamma_1, \gamma_2), (q', w_i \dots w_n, \gamma_1', \gamma_2') \in \vdash$
8. **print** $(q', w_i \dots w_n, \gamma_1', \gamma_2')$
9. **return**
10. **else**
11. **print** *false*

Figure 1: Recognition procedure for read-first deterministic stack+bag pushdown automata.

dass	der	Dedektiv	dem	Klienten	den	Verdächtigen	des	Verbrechens
that	the	detective.NOM	the	client.DAT	the	suspect.ACC	the	crime.GEN
zu	überführen	versprochen	hat					
to	indict	promised	has					

'that the detective has promised the client to indict the suspect of the crime.'

Figure 2: Example from Becker et al. (1991).

$$
\begin{aligned}
\delta(q_0, \lambda, \lambda) &\in \delta(q_1, \lambda, \{NP', V_n\}_M) \\
\delta(q_1, NP_1, \lambda) &\in \delta(q_1, \lambda, \{V_1\}_M) \\
&\vdots \\
\delta(q_1, NP_{n-1}, \lambda) &\in \delta(q_1, \lambda, \{V_{n-1}\}_M) \\
\delta(q_1, NP', \lambda) &\in \delta(q_2, \lambda, \emptyset_M) \\
\delta(q_2, V_1, V_1) &\in \delta(q_2, \lambda, \emptyset_M) \\
&\vdots \\
\delta(q_2, V_{n-1}, V_{n-1}) &\in \delta(q_2, \lambda, \emptyset_M) \\
\delta(q_2, V_n, V_n) &\in \delta(q_3, \lambda, \emptyset_M)
\end{aligned}
$$

In the transition from q_0 to q_1 a requirement that there is a main verb that has a subject, intuitively, is pushed into the bag. In the cyclic transitions in q_1, the NPs, incl. the subject of the finite verb V_n, are read, and when NP_i for $1 \leq i < n$ is read the stack symbol for the corresponding embedded verb V_i is pushed into the bag. The verbs are read in the cyclic transitions in q_2. Finally, the finite verb V_n is read.

7 Conclusion

This article presents a class of extended pushdown automata, i.e. λ-acyclic read-first deterministic stack+bag pushdown automata, that recognize a class of languages that strictly includes the deterministic context-free languages (Lemma 4.5), but also languages conjectured not to be indexed languages (by the observation that the automaton in Lemma 4.2 is λ-acyclic and read-first deterministic). In fact, the λ-acyclic read-first deterministic stack+bag pushdown languages are not included in the kth level of the hierarchy of control languages for any fixed k (Corollary 4.4). It was shown that the universal recognition problem for this class of pushdown automata can be solved in linear time (Theorem 5.1).

Similar classes of linear time recognizable languages have been identified in the literature. Bertsch and Nederhof (1999) also identify a class of linear time recognizable pushdown languages, namely the class of all languages that are in the regular closure of the class of deterministic pushdown languages. This class includes a number of ambiguous context-free languages, incl. $\{a^m b^m c^n\} \cup \{a^m b^n c^n\}$ which is probably not a read-first deterministic stack+bag pushdown language, but no non-context-free languages. It follows, if so, that this class and the class of λ-acyclic read-first deterministic stack+bag pushdown languages are strict extensions of their intersection.

Since the paper was first submitted, a parser has been implemented in Python. The parser hardwires a read-first strategy and warns the user about nondeterminism and λ-cycles. It is of course difficult to test if the degree of nondeterminism given to us by bags is adequate for natural language processing, but a toy automaton has been constructed that parses attachment ambiguities, verbs

with different subcategorization frames, and recursive modifiers.

References

Alfred Aho and Margaret Corasick. 1975. Efficient string matching: an aid to bibliographic search. *Communications of the Association for Computing Machinery*, 18(6):333–340.

Alfred Aho and Jeffrey Ullman. 1972. *The theory of parsing, translation and compiling*. Prentice-Hall, London, England.

Tilman Becker, Aravind Joshi, and Owen Rambow. 1991. Long-distance scrambling and tree adjoining grammars. In *Proceedings of the 5th Conference of the European Chapter of the Association for Computational Linguistics*, pages 21–26, Berlin, Germany.

Eberhard Bertsch and Mark-Jan Nederhof. 1999. Regular closure of deterministic languages. *SIAM Journal on Computing*, 29(1):81–102.

Pierre Boullier. 1998. Proposal for a natural language processing syntactic backbone. Technical report, INRIA, Le Chesnay, France.

Noam Chomsky. 1962. Context-free grammars and pushdown storage. Quarterly Progress Report 65, Research Laboratory of Electronics, Massachusetts Institute of Technology, Boston, Massachusetts.

Gerald Gazdar. 1988. Applicability of indexed grammars to natural languages. In Uwe Reyle and Christian Rohrer, editors, *Natural language parsing and linguistic theories*, pages 69–94. Reidel, Dordrecht, the Netherlands.

Laura Kallmeyer and Sin-Won Yoon. 2004. Tree-local MCTAG with shared nodes. In *Proceedings of the Traitement Automatique des Langues Naturelles*, Fes, Marocco.

Timm Lichte. 2007. An MCTAG with tuples for coherent constructions in German. In *Proceedings of the 12th Conference on Formal Grammar*, Dublin, Ireland.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 149–160, Nancy, France.

Michael Palis and Sunil Shende. 1995. Pumping lemmas for the control language hierarchy. *Mathematical Systems Theory*, 28:199–213.

Owen Rambow and Giorgio Satta. 1992. Formal properties of nonlocality. In *Proceedings of the 2nd International Workshop on Tree Adjoining Grammars*, Philadelphia, Pennsylvania.

Owen Rambow, Tilman Becker, and Michael Niv. 1992. Scrambling is beyond LCFRS. Manuscript, University of Pennsylvania.

Anders Søgaard, Timm Lichte, and Wolfgang Maier. 2007. On the complexity of linguistically motivated extensions of tree-adjoining grammar. In *Proceedings of Recent Advances in Natural Language Processing 2007*, Borovets, Bulgaria.

K. Vijay-Shanker and David Weir. 1994a. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27:511–546.

K. Vijay-Shanker and David Weir. 1994b. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636.

K. Vijay-Shanker. 1987. *A study of tree-adjoining grammar*. Ph.D. thesis, University of Pennsylvania, Philadelphia, Pennsylvania.

Éric Villemonte de La Clergerie. 2002. Parsing mildly context-sensitive languages with thread automata. In *Proceedings of the 19th International Conference on Computational Linguistics*, pages 1–7, Taipei, Taiwan.

David Weir. 1988. *Characterizing mildly context-sensitive grammar formalisms*. Ph.D. thesis, University of Pennsylvania, Philadelphia, Pennsylvania.

David Weir. 1992. A geometric hierarchy beyond context-free languages. *Theoretical Computer Science*, 104:235–261.

Fei Xia and Tonia Bleam. 2000. A corpus-based evaluation of syntactic locality in TAGs. In *Proceedings of the 5th International Workshop on Tree Adjoining Grammars and Related Formalisms*, Paris, France.

ISSN 1736-6305 Vol. 4
http://hdl.handle.net/10062/9206