Extending Linear Indexed Grammars

Christian Wartena

Universität Potsdam
Institut für Linguistik/Allgemeine Sprachwissenschaft
Postfach 601553, 14415 Potsdam, Germany
wartena@ling.uni-potsdam.de
WWW home page: http://www.ling.uni-potsdam.de/~wartena

Abstract

This paper presents a possibility to extend the formalism of linear indexed grammars. The extension is based on the use of tuples of pushdowns instead of one pushdown to store indices during a derivation. If a restriction on the accessibility of the pushdowns is used, it can be shown that the resulting formalisms give rise to a hierarchy of languages that is equivalent with a hierarchy defined by Weir. For this equivalence, that was already known for a slightly different formalism, this paper gives a new proof. Since all languages of Weir's hierarchy are known to be mildly context sensitive, the proposed extensions of LIGs become comparable with extensions of tree adjoining grammars and head grammars.

1. Introduction

It is well known that tree adjoining grammars (TAGs), head grammars (HGs) and linear indexed grammars (LIGs) are weakly equivalent (Vijay-Shanker & Weir, 1994). Each of these formalisms was developed independently for the description of natural languages. For TAGs and HGs hierarchies of extensions were defined by increasing the number of auxiliary trees that are inserted in one step and by increasing the size of the tuples that are handled, resp. (cf. (Weir, 1988)). The extensions of TAGs, multi–component TAGs (MCTAGs) (Joshi, 1987), were argued to be useful for the description of natural languages by Kroch (1987) and Kroch and Joshi (1987). For LIGs a linguistically motivated extension is defined by Rambow (1994) that is however of a rather different nature than the extensions of HGs and TAGs and does not give rise to a hierarchy of formalisms and language classes. Weir (1988; 1992) defines a hierarchy of linear controlled grammars that are strongly related to LIGs. It is however not immediately apparent what use these formalisms could have for linguistics. In (Wartena, 1998) recently extensions of LIGs, called *context–free linear multi–pushdown grammars (CFL–MPD–Gs)*, were defined that use tuples of pushdowns to store indices instead of a single pushdown. The use of tuples was motivated by linguistic needs. These extensions form a hierarchy of formalisms with an increasing number of pushdowns. If no pushdown is available the grammars are strongly equivalent to context–free grammars. If one pushdown is used we obtain LIGs. The nth element of the hierarchy can be shown to be a subclass of the nth class of Weir's hierarchy of controlled languages.

CFL–MPD–Gs seem to fill up an apparent gap in the square formed by TAGs, HGs and LIGs on the first axis and their extensions on the other axis. In order to formally justify this square we have to show that CFL–MPD–Gs and MCTAGs[1] or the extensions of head grammars are equivalent. (The equivalence between the last two was shown by Weir (1988)). We will go

[1] There are two variants of MCTAGs, the first of which allows only for simultaneous adjunction in one elemen-

the following way to show this equivalence. First we will prove the equivalence between the hierarchy of CFL–MPD–Gs and Weir's hierarchy of linear controlled grammars. Subsequently the equivalence between the latter hierarchy and MCTAGs has to be shown. In this paper we will do the first of the two steps.

2. Grammars with storage

LIGs store their indices in pushdowns. For the description of non–local dependencies in natural languages this organization can be argued to be too restrictive. Thus we might want to define formalisms similar to LIGs but with a more liberal stack structure. We start defining abstract storages, that will form the base of the subsequent extensions.

Definition 1 (storage) A *storage* is a 6–tuple $S = (C, C_0, C_F, P, F, m)$, where C is a set of configurations, $C_0 \subseteq C$ and $C_F \subseteq C$ the sets of initial and final configurations, respectively, P is a set of predicate symbols, F a set of instruction symbols. m is the meaning function, which associates every $p \in P$ with a mapping $m(p) : C \to \{\mathsf{true}, \mathsf{false}\}$ and every $f \in F$ with a partial function $m(f) : C \to C$.

Usually we are interested in properties of classes of storages rather than in properties of individual ones. Classes of storages are often called *storage types*.

Example 1 A *trivial storage* is defined as $S_{\mathrm{triv}} = (\{c\}, \{c\}, \{c\}, \emptyset, \{\mathsf{id}\}, m)$, where c is an arbitrary object and $m(\mathsf{id})(c) = c$. The class of all trivial storages is denoted $\mathfrak{S}_{\mathrm{triv}}$.

Example 2 A *pushdown* over some finite alphabet Γ can be defined as a storage[2] $S_{\mathrm{pd}}(\Gamma) = (\Gamma^*, \{\epsilon\}, \{\epsilon\}, P, F, m)$ with $P = \{\mathsf{top}(\gamma) | \gamma \in \Gamma\}$, $F = \{\mathsf{push}(\gamma) \mid \gamma \in \Gamma\} \cup \{\mathsf{pop}\} \cup \{\mathsf{id}\}$ and for every $a \in \Gamma$ and $\beta \in \Gamma^*$,

$$m(\mathsf{top}(\gamma))(a\beta) = (a = \gamma) \qquad m(\mathsf{pop})(a\beta) = \beta$$
$$m(\mathsf{push}(\gamma))(\beta) = \gamma\beta \qquad m(\mathsf{id})(\beta) = \beta$$

The class of all pushdowns is denoted $\mathfrak{S}_{\mathrm{pd}}$.

On the base of this notion of storages we can define *context–free linear–S grammars (CFL–S–Gs)* as a generalization of LIGs.

Definition 2 (CF linear S–grammar) If $S = (C, C_0, C_F, P, F, m)$ is a storage then a *context–free linear S–grammar* is a five tuple $G = (N, \Sigma, R, A_{\mathrm{in}}, c_0)$, where N, Σ denote the sets of nonterminal and terminal symbols, respectively. $A_{\mathrm{in}} \in N$ is a distinguished symbol, called the *start symbol*, $c_0 \in C_0$ is the *initial configuration*, and R is a set of *production rules* of the following two forms:

$$A \to \mathsf{if}\ \pi\ \mathsf{then}\ \zeta_1(B, f)\zeta_2$$
$$A \to \mathsf{if}\ \pi\ \mathsf{then}\ w$$

where $A, B \in N$, $\pi \in BE(P))$ and $\zeta_1, \zeta_2 \in (N \cup \Sigma)^*$, $f \in F$, $w \in \Sigma^*$. BE(P) denotes the set of boolean expressions over P.

tary tree, the second one of which allows for adjunction of a tuple in a tuple of elementary trees as well. The first variant is equivalent to (simple) TAGs, the second one gives rise to an hierarchy of languages. In this paper we will only consider these more powerful MCTAGs.

[2]Throughout the paper the following notational conventions are used. The empty string is denoted by ϵ. For each set V the notation V_ϵ is used as an abbreviation for $V \cup \{\epsilon\}$.

A string $\sigma \in O^*$ is said to derive a string $\tau \in O^*$, written $\sigma \underset{G}{\Rightarrow} \tau$, if either (1) or (2).

(1) $\sigma = \alpha(A, c)\beta$
$A \to \text{if } \pi \text{ then } \zeta_1 B f \zeta_2 \in R$
$m(\pi)(c) = \text{true}$[3]
$m(f)$ is defined on c
$\tau = \alpha\zeta_1'(B, c')\zeta_2'\beta$

(2) $\sigma = \alpha(A, c)\beta$
$A \to \text{if } \pi \text{ then } w \in R$
$m(\pi)(c) = \text{true}$
$\tau = \alpha w \beta$

where $A, B \in N$, $\alpha, \beta \in O^*$, $w \in \Sigma^*$, $c \in C$, $c' = m(f)(c)$ and ζ_1', ζ_2' are obtained from ζ_1 and ζ_2, respectively, by replacing every nonterminal D by (D, c_0). The reflexive and transitive closure of $\underset{G}{\Rightarrow}$, denoted by $\underset{G}{\Rightarrow}^*$, is defined as usual. The language generated by G is defined as $L(G) = \{w \in \Sigma^* | (A_{\text{in}}, c_0) \underset{G}{\Rightarrow}^* w\}$. If $\mathfrak{S}$ is a storage type and $S \in \mathfrak{S}$ then a CFL–S–G is called a CFL–$\mathfrak{S}$–G as well. The class of languages generated by CFL–$\mathfrak{S}$–Gs is denoted $\mathfrak{L}_{\text{CFL}(\mathfrak{S})}$.
The way in which the storage is passed on during a derivation is the same as in LIGs. It is easy to check that CFL–$\mathfrak{S}_{\text{pd}}$–Gs are equivalent to LIGs. Now we can easily define extensions of LIGs by choosing other storage types. The generative capacity of variants that are defined in this way crucially depends on the storage type. In order to investigate the typical complexity and generative capacity of a storage type we will use storage–automata.

Definition 3 (S–automaton) If $S = (C, C_0, C_F, P, F, m)$ is a storage, then an *S–automaton* M is a tuple $(Q, \Sigma, \delta, q_0, c_0, Q_F)$, where Q is a finite set of states, Σ is the input alphabet, $q_0 \in Q$ the initial state, $c_0 \in C_0$ the initial configuration, $Q_F \subseteq Q$ the set of final states, and δ, the transition relation, a finite subset of $Q \times \Sigma_\epsilon \times \text{BE}(P) \times Q \times F$.

The set $\text{ID}(M) = Q \times \Sigma^* \times C$ is called the set of instantaneous descriptions. For each (q_1, xw, c_1), $(q_2, w, c_2) \in \text{ID}(M)$ with $x \in \Sigma_\epsilon$ we write $(q_1, xw, c_1) \vdash_M (q_2, w, c_2)$ if there exists $(q_1, x, \pi, q_2, f) \in \delta$ such that $m(\pi)(c_1) = \text{true}$, $m(f)$ is defined on c_1 and $m(f)(c_1) = c_2$. The transitive and reflexive closure $\vdash_M^*$ of $\vdash_M$ is defined as usual. Sometimes conjunction of function symbols is used. For two function symbols f_1 and f_2 the meaning of the composed function symbol $f_1 \& f_2$ is defined as $m(f_1 \& f_2) = m(f_2) \circ m(f_1)$. The language accepted by M is defined $L(M) = \{w \,|\, (q_0, w, c_0) \vdash_M^* (q, \epsilon, c)$ for some $c \in C$ and $q \in Q_F\}$ if $Q_F \neq \emptyset$ and $L(M) = \{w \,|\, (q_0, w, c_0) \vdash_M (q_1, w', c_1) \vdash_M \ldots (q_n, \epsilon, c_n)$ for some $c_n \in C_F$, $q_n \in Q$, $c_i \in C - C_F$ and $q_i \in Q$ with $1 \leq i < n\}$ otherwise. In the first case we say that M accepts by final state. In the second case M accepts by final configuration. Let $\mathfrak{S}$ be some storage type. If M is an S–automaton and $S \in \mathfrak{S}$ we say as well that M is an $\mathfrak{S}$–automaton. Take e.g. $\mathfrak{S} = \mathfrak{S}_{\text{pd}}$, then we can say as usual that an automaton M is an $\mathfrak{S}_{\text{pd}}$–automaton or a pushdown–automaton without reference to the specific pushdown that is used. Finally we set $L_C(\mathfrak{S}) = \{L(M) \mid M$ is an $\mathfrak{S}$–automaton accepting by final configuration$\}$ and $L_Q(\mathfrak{S}) = \{L(M) \mid M$ is an $\mathfrak{S}$–automaton accepting by final state$\}$. For some important storage types (like pushdowns and concatenations of pushdowns) $L_Q(\mathfrak{S}) = L_C(\mathfrak{S})$. In these cases we drop the subscript. In (Wartena, 2000) these storage types are called *well–behaved*. The reader is referred there for details. A subclass of the well–behaved storage types is constituted by the *concatenating* storages types. In a concatenating storage $C_0 = C_F$ and the cardinality of C_0 is 1.

3. Concatenation of storages

It was argued in (Wartena, 1998) that a tuple of pushdowns would be an adequate storage type to describe non–local dependencies in a number of constructions in various languages. The

[3] In fact only $m(\pi)$ for $\pi \in P$ has been defined so far. It is straightforward to extend the domain of m to $\text{BE}(P)$.

motivation there was that we have to distinguish between different types of non–local dependencies, as proposed in theories like that of relativized minimality (Rizzi, 1990), and that there has to be one stack for each type.

Definition 4 (product of storages) For arbitrary storages $S^1 = (C^1, C_0^1, C_F^1, P^1, F^1, m^1)$ and $S^2 = (C^2, C_0^2, C_F^2, P^2, F^2, m^2)$ the *product* of S^1 and S^2 is defined as $S^1 \circ S^2 = (C^1 \times C^2, C_0^1 \times C_0^2, C_F^1 \times C_F^2, P, F, m)$ where $P = P^1 \cup \{\mathsf{test}(p) \mid p \in P^2\}$, $F = F^1 \cup \{\mathsf{do}(f) \mid f \in F^2\}$ and for every $c^1 \in C^1$ and $c^2 \in C^2$

$$\begin{aligned} m(p)((c^1,c^2)) &= m^1(p)(c^1) \text{ if } p \in P^1 \\ m(f)((c^1,c^2)) &= (m^1(f)(c^1), c^2) \text{ if } f \in F^1 \\ m(\mathsf{test}(p))((c^1,c^2)) &= m^2(p)(c^2) \text{ if } p \in P^2 \\ m(\mathsf{do}(f))((c^1,c^2)) &= (c^1, m^2(f)(c^2)) \text{ if } f \in F^2 \end{aligned}$$

The set of *semi–final* configurations of $S^1 \circ S^2$ is defined as $C_{SF} = (C_0^1 \cup C_F^1) \times C^2$. For two storage types $\mathfrak{S}^1$ and $\mathfrak{S}^2$ the product is defined straightforwardly as $\mathfrak{S}^1 \circ \mathfrak{S}^2 = \{S^1 \circ S^2 \mid S^1 \in \mathfrak{S}^1 \text{ and } S^2 \in \mathfrak{S}^2\}$.

Tuples or products of pushdowns are from a formal point of view to powerful. Following an idea of Breveglieri et al. (1996) we can reduce this power by restricting the operations that can be applied to the components. A similar idea to restrict the power of tuples of stacks was proposed by Becker (1994). Here we will define concatenation by means of an explicit restriction on a product of two storages. This general definition was suggested by Lothar Budach (p.c.).

Definition 5 (K–product of storages) For arbitrary storages S^1 and S^2 such that $S^1 \circ S^2 = (C, C_0, C_F, P, F, m)$ and for a mapping $K : F^2 \to \{\text{true}, \text{false}\}$ the *K–product* of S^1 and S^2 is defined as $S^1 \circ_K S^2 = (C, C_0, C_F, P, F, m')$ with[4]

$$\begin{aligned} m'(\varphi) &= m(\varphi)|_{C_{SF}} && \text{if } \varphi = \mathsf{do}(\varphi') \text{ and } K(\varphi') = \text{true} \\ m'(\varphi) &= m(\varphi) && \text{otherwise.} \end{aligned}$$

For two storage types $\mathfrak{S}^1$ and $\mathfrak{S}^2$ and any predicate K the K–product is defined as $\mathfrak{S}^1 \circ_K \mathfrak{S}^2 = \{S^1 \circ S^2 \mid_K S^1 \in \mathfrak{S}^1 \text{ and } S^2 \in \mathfrak{S}^2\}$.

Note that $m(\mathsf{do}(f))((c^1,c^2))$ is undefined for $f \in F^2$ if $K(f)$ is true and c^1 is not initial or final. The K–products for two predicates K are of special interest. The predicate r determines what operations are considered as reading operations. For any pushdown let $r(\mathsf{pop}) = \text{true}$ and let $r(\mathsf{push}) = r(\mathsf{id}) = \text{false}$. The r–product of two stores corresponds exactly with the *concatenation with regard to reading* defined in (Wartena, 1998). The counterpart of the predicate r is w which is defined by $w(\mathsf{push}) = \text{true}$ and $w(\mathsf{pop}) = w(\mathsf{id}) = \text{false}$ for any (n–turn) pushdown. The product $\circ_w$ is the same as *concatenation with regard to writing*.

Example 3 S_{pd} denotes a pushdown storage. Consequently, $(S_{\mathrm{pd}} \circ_r S_{\mathrm{pd}}) \circ_r S_{\mathrm{pd}}$ denotes the concatenation w.r.t. reading of three pushdowns. Each component behaves like a pushdown. At each point in the computation elements can be pushed on each of the three pushdowns, popping, however, is only possible from the first non empty one.

[4] For any (partial) function $f : A \to B$ and any $U \subseteq A$ the *restriction of f to U*, denoted $f|_U$, is defined as $f|_U(u) = f(u)$ if $u \in U$ and undefined otherwise.

Using r– and w–products we can recursively define the following hierarchies of storage types and corresponding classes of languages. The hierarchy established in (Breveglieri *et al.*, 1996) can now be defined as $\mathfrak{C}^{\mathfrak{R}}_i = L(\mathfrak{S}_i)$ for each $i \geq 0$ where $\mathfrak{S}_0 = \mathfrak{S}_{\text{triv}}$ and $\mathfrak{S}_i = \mathfrak{S}_{i-1} \circ_r \mathfrak{S}_{\text{pd}}$. ($\mathfrak{C}$ is intended as a mnemonic for concatenation, the superscript $\mathfrak{R}$ indicating that concatenation w.r.t. reading is meant.) For natural language syntax (Wartena, 1998) argues that concatenation w.r.t. writing is more appropriate. Thus the hierarchy, that is defined by setting $\mathfrak{C}^{\mathfrak{W}}_i = L(\mathfrak{S}_i)$ for each $i \geq 0$ where $\mathfrak{S}_0 = \mathfrak{S}_{\text{triv}}$ and $\mathfrak{S}_i = \mathfrak{S}_{i-1} \circ_w \mathfrak{S}_{\text{pd}}$, is of more interest for us. It can however be shown that $\mathfrak{C}^{\mathfrak{R}}_i \subseteq \mathfrak{C}^{\mathfrak{W}}_{i+1}$ and that $\mathfrak{C}^{\mathfrak{W}}_i \subseteq \mathfrak{C}^{\mathfrak{R}}_{i+1}$. Thus both hierarchies are very similar. An interesting fact is that $\mathfrak{C}^{\mathfrak{R}}_2$ and $\mathfrak{C}^{\mathfrak{W}}_2$ are the classes of extended left and extended right linear indexed languages (ELLILs and ERLILs), resp., defined in (Michaelis & Wartena, 1999). ERLILs were proposed as an appropriate restriction of linear indexed languages w.r.t. the paths along which a stack can be inherited.

4. Linear controlled grammars

The hierarchy of Weir can be expressed most easily in terms of linear control. Linear control of context–free grammars (CFGs) is defined in (Weir, 1992).

Definition 6 (linear controlled grammar) A *linear distinguished grammar (LDG)* is a quadruple $G = (N, \Sigma, R, A_{\text{in}})$, where N and Σ are finite sets of non–terminal and terminal symbols, respectively, $A_{\text{in}} \in N$ is the start symbol and where R is a finite set of production rules of the form: $A \to \beta_1 X ! \beta_2$ with $A \in N$, $X \in N \cup \Sigma$, called the *distinguished symbol*, $\beta_1, \beta_2 \in (N \cup \Sigma)^*$, and ! a special symbol not in $(N \cup \Sigma)$. A *linear controlled grammar (LCG)* is a pair $K = (G, H)$, where G is an LDG and H is a language over R, called the control language.

The set of (nonterminal and terminal) objects in K is defined as $O(K) = (N \cup \Sigma) \times R^*$. A string $\sigma \in O(K)^*$ is said to derive a string $\tau \in O(K)^*$, written $\sigma \underset{G}{\Rightarrow} \tau$, if

$$\begin{gathered} \sigma = \gamma(A, \omega)\delta \\ r = A \to \beta_1 X ! \beta_2 \in R \\ \tau = \gamma\beta_1'(X, \omega r)\beta_2'\delta \end{gathered}$$

where $A \in N$, $X \in N \cup \Sigma$, $\beta_1, \beta_2 \in (N \cup \Sigma)^*$, $\gamma, \delta \in O(K)^*$, $\omega \in R^*$, and β_1' and β_2' are obtained from β_1 and β_2, resp. by replacing every symbol $Y \in N \cup \Sigma$ by (Y, ϵ). In this case $(X, \omega r)$ is called the *distinguished child* of (A, ω). The reflexive and transitive closure of $\underset{G}{\Rightarrow}$, denoted $\underset{G}{\Rightarrow}^*$, is defined as usual. The language generated by K is defined as $L(K) = \{a_1 \dots a_n | (S, \epsilon) \underset{G}{\Rightarrow}^* (a_1, \omega_1) \dots (a_n, \omega_n)$ and $a_i \in \Sigma, \omega_i \in H$ for $1 \leq i \leq n\}$. The class of all LDGs is denoted by $\mathfrak{G}_{\text{LD}}$. Furthermore, for any class of grammars $\mathfrak{G}$ for which control is defined let $\mathfrak{G}/\mathfrak{L} = \{(G, H) \mid G \in \mathfrak{G}$ and $H \in \mathfrak{L}\}$ and for any class of grammars $\mathfrak{G}$ let $L(\mathfrak{G}) = \{L(G) \mid G \in \mathfrak{G}\}$. The obvious relation between linear controlled grammars and CFL–$\mathfrak{S}$–grammars was shown in (Wartena, 1998).

Proposition 7 $L(\mathfrak{G}_{\text{LD}}/L_Q(\mathfrak{S})) = \mathfrak{L}_{\text{CFL}(\mathfrak{S})}$ □

In order to refer to objects in a derivation it is sometimes assumed that the objects have addresses in $\mathbb{N}^*$.[5] In the following we will use two different address assignments, *leftmost* and *inside–out* address assignment. Suppose a string $\sigma = \alpha X \beta \in O(K)^*$ derives a string τ rewriting the object X with address ξ into new objects $Y_0 Y_1 \dots Y_i \dots Y_n$ with Y_i the distinguished child of X. If the address assignment is leftmost then the address of each Y_k is ξk with $0 \leq k \leq n$. In the case of inside–out assignment the address of Y_k is $\xi(i-k)$ for $0 \leq k \leq i$ and ξk for

[5] $\mathbb{N}$ denotes the set of all non–negative integers.

$i < k \leq n$. For each object in α and β the address in σ and τ is the same. A sequence of strings of objects $\delta = \sigma_0 \ldots \sigma_n$ such that $\sigma_i \Rightarrow \sigma_{i+1}$ is called a derivation If in each step the nonterminal object with the lexicographic[6] smallest address is rewritten then the derivation is called leftmost in case the address assignment is leftmost and inside–out in case the address assignment is inside–out.

5. The hierarchy of Weir

The classes of languages constituting Weir's hierarchy can be defined by setting $\mathfrak{W}_0 = \mathfrak{L}_{\text{reg}}$ and $\mathfrak{W}_i = L(\mathfrak{G}_{\text{LD}}/\mathfrak{W}_{i-1})$ for each $i > 0$. The following proposition was already shown in (Wartena, 1998).

Proposition 8 $\mathfrak{C}^{\mathfrak{R}}{}_i \subseteq \mathfrak{W}_i$ □

The languages of Weir's hierarchy of controlled grammars are accepted by concatenated pushdowns as well. Below we will show that the derivation of an LDG controlled by some $\mathfrak{S}$–automaton can be executed by an $(\mathfrak{S} \circ_r (\mathfrak{S}_{\text{pd}} \circ_r \mathfrak{S}_{\text{pd}}))$–automaton. The idea is that the automaton follows one spine using its finite control to store the element actually being expanded and using the first component to compute the control word. Everything that is generated to the right of the spine is written on the third pushdown, terminal and nonterminal symbols generated to the left of the spine are written on the second pushdown. If the foot of the spine is reached the second pushdown contains the left part of the derived sentential form in reversed order. The automaton now continues expanding the nonterminals on that pushdown, starting with the nonterminal directly to the left of the foot of the spine that just is reached. The automaton can read that symbol, since the first component is empty, just having accepted a control word. Thus the automaton simulates an inside–out derivation.

Lemma 9 *Let S be a concatenating storage. Then the following holds.*

$$L(\mathfrak{G}_{\text{LD}}/L(\mathfrak{S})) \subseteq L(\mathfrak{S} \circ_r (\mathfrak{S}_{pd} \circ_r \mathfrak{S}_{pd}))$$

Proof. Let $\mathfrak{S}$ be a concatenating storage type, let $S = (C, C_0, C_F, P, F, m) \in \mathfrak{S}$ and let $K = (G, L(M))$ be an LCG with $G = (N, \Sigma, R, A_{\text{in}})$ an LDG and $M = (Q, R, \delta, c_0, \emptyset)$ an S–automaton. Assume w.l.o.g. that each production of G is of the form $A \to B_1 B_2 ! B_3$ or $A \to a$ with $A, B_1, B_2, B_3 \in N$ and $a \in \Sigma_\epsilon$. Construct an $(S \circ_r (S_{pd}(\Gamma) \circ_r S_{pd}(\Gamma))$–automaton $M' = (Q \times N_\epsilon, \Sigma, \delta', (q_0, A_{\text{in}}), (c_0, \epsilon, \epsilon), \emptyset)$ with $\Gamma = N \cup \Sigma$, by setting

$$\begin{aligned}
\delta' &= \{((q_1, A), \epsilon, \pi, (q_2, B_2), f\&\mathsf{do}(\mathsf{push}(B_1)\&\mathsf{do}(\mathsf{do}(\mathsf{push}(B_2)))) \mid \\
&\qquad r = A \to B_1 B_2 ! B_3 \in R \text{ and } (q_1, r, \pi, q_2, f) \in \delta\} && (1a)\\
&\cup \{((q_1, A), \epsilon, \pi, (q_2, \epsilon), f\&\mathsf{do}(\mathsf{do}(\mathsf{push}(a))) \mid \\
&\qquad r = A \to a \in R \text{ and } (q_1, r, \pi, q_2, f) \in \delta\} && (1b)\\
&\cup \{((q_1, A), \epsilon, \pi, (q_2, \epsilon), f) \mid (q_1, \epsilon, \pi, q_2, f) \in \delta\} && (2)\\
&\cup \{((q, \epsilon), \epsilon, \mathsf{test}(\mathsf{top}(A)), (q, A), \mathsf{do}(\mathsf{pop})) \mid A \in N\} && (3a)\\
&\cup \{((q, \epsilon), \epsilon, \mathsf{test}(\mathsf{test}(\mathsf{top}(A))), (q_0, A), \mathsf{do}(\mathsf{do}(\mathsf{pop}))) \mid A \in N\} && (3b)\\
&\cup \{((q, \epsilon), a, \mathsf{test}(\mathsf{test}(\mathsf{top}(a))), (q, \epsilon), \mathsf{do}(\mathsf{do}(\mathsf{pop}))) \mid a \in \Sigma\} && (4)
\end{aligned}$$

It can be shown by induction on the number of steps in a computation and in a derivation, respectively, that

$$((q_1, A), w, (c_1, (\epsilon, \epsilon))) \vdash_{M'}^* ((q_2, B), \epsilon, (c_2, (\alpha^R, \beta)))$$
iff
$$(A, \epsilon) \overset{*}{\underset{G}{\Rightarrow}} w\alpha'(B, \omega)\beta' \text{ (inside–out) and } (q_1, \omega, c_1) \vdash_M^* (q_2, \epsilon, c_2)$$

[6] The lexicographic ordering relation $<_{\text{lex}}$ on $\mathbb{N}^*$ is defined by: $\chi <_{\text{lex}} \chi j \omega$ and $\chi i \psi <_{\text{lex}} \chi j \omega$ if $i < j$ for all $\chi, \psi, \omega \in \mathbb{N}^*$ and $i, j \in \mathbb{N}$.

where α', β' are obtained from α and β respectively by replacing every nonterminal $A \in N$ by (A, ϵ). In case $A = A_{\text{in}}$ and $\alpha = \beta = B = \epsilon$ we see that $L(M') = L(K)$. □

Proposition 10 *for each* $i > 1$: $\mathfrak{W}_i \subset \mathfrak{C}^{\mathfrak{R}}{}_{2i-1}$

Proof. First we show by induction that $\mathfrak{W}_i \subseteq \mathfrak{C}^{\mathfrak{R}}{}_{2i-1}$ for $i \geq 1$. For $i = 1$ the proposition is trivially true, since $\mathfrak{W}_1 = \mathfrak{C}^{\mathfrak{R}}{}_1 = \mathfrak{L}_{\text{CF}}$, the class of all context–free languages. Suppose that the assertion is true for some $i \in \mathbb{N}$. Let $\mathfrak{S}_i$ denote the concatenation w.r.t. reading of i–pushdowns, for some $i \in \mathbb{N}$. thus $L(\mathfrak{S}_i) = \mathfrak{C}^{\mathfrak{R}}{}_i$. For $i + 1$ we find

$$\begin{array}{rcll} \mathfrak{W}_{i+1} & = & L(\mathfrak{G}_{\text{LD}}/\mathfrak{W}_i) & \text{(by definition)} \\ & \subseteq & L(\mathfrak{G}_{\text{LD}}/\mathfrak{C}^{\mathfrak{R}}{}_{2i-1}) & \text{(by induction)} \\ & \subseteq & L(\mathfrak{S}_i \circ_r (\mathfrak{S}_{pd} \circ_r \mathfrak{S}_{pd})) & \text{(by Lemma 9)} \\ & = & \mathfrak{C}^{\mathfrak{R}}{}_{2i+1} = \mathfrak{C}^{\mathfrak{R}}{}_{2(i+1)-1} & \text{(by definition)} \end{array}$$

For $i > 1$ the inclusion is proper since it is known that both $\mathfrak{C}^{\mathfrak{R}}{}_i$ and $\mathfrak{W}_i$ contain the language $\{a_1^n \ldots a_{2i}^n \mid n \in \mathbb{N}\}$ but not the language $\{a_1^n \ldots a_{2i+1}^n \mid n \in \mathbb{N}\}$. □

This result combined with Proposition 8 implies that the languages from the multi–pushdown hierarchy are the same as those in Weir's hierarchy.

Corollary 11 $\bigcup_{i=0}^{\infty} \mathfrak{W}_i = \bigcup_{i=0}^{\infty} \mathfrak{C}^{\mathfrak{R}}{}_i$ □

A similar result was found by Cherubini and San Pietro (1999a; 1999b), using different proofs. Finally, let us return to the context–free linear $\mathfrak{S}$–grammars. The extensions of LIGs we are interested in are CFL–$\mathfrak{S}$–Gs with $\mathfrak{S}$ a concatenation of pushdowns. Calling each storage type formed by concatenation of pushdowns a *multiple pushdown (MPD)* we can refer to these grammars as CFL–MPD–grammars. It is straightforward to check that the languages generated by CFL–MPD–Gs are included in the hierarchy of Weir as well. Let $\mathfrak{S}_i$ be the storage that arises from concatenation w.r.t. reading from i–pushdowns, for some $i \in \mathbb{N}$. Then we find

$$\begin{array}{rcll} \mathfrak{L}_{CFL(\mathfrak{S}_i)} & = & L(\mathfrak{G}_{\text{LD}}/\mathfrak{C}^{\mathfrak{R}}{}_i) & \text{(by Proposition 7)} \\ & \subseteq & L(\mathfrak{G}_{\text{LD}}/\mathfrak{W}_i) & \text{(by Proposition 8)} \\ & \subseteq & \mathfrak{W}_{i+1} & \text{(by definition)} \end{array}$$

The inclusion of the classes Weir's hierarchy in the classes of languages generated by CFL–MPD–Gs is even simpler, since it can be shown that $\mathfrak{L}_{CFL(\mathfrak{S})} \supset L(\mathfrak{S})$.

6. Conclusion

In this paper two hierarchies of storage types were presented that are based on tuples of pushdowns with restrictions on the accessibility of the components. These storage types can be used to make new and linguistically interesting extensions of LIGs and besides for the construction of automata. It can be shown that automata based on a concatenation of two pushdowns accept only a subset of the linear indexed languages (LILs). Automata based on a triple of pushdowns accept already languages that cannot be generated by any LIG. A storage type corresponding to LIGs, the *nested pushdown*, was defined by Weir (1994). Though this storage type is rather different from ours, in a nested pushdown there are as well various possibilities for writing but only one for popping symbols. Becker (1994) defined automata, as well accepting LILs, that use two nested stacks with an explicit restriction on popping symbols from the second one, similar to the restrictions defined above. Reading from the second nested stack is possible if the

top of the first one is a bottom of stack symbol of an embedded stack. Thus reading from the second component is restricted but not only to situations in which the first component is empty. Storage types based on tuples with restricted possibilities for writing to our knowledge were not considered in any form up to now.

The main result of this paper is a new proof for the equivalence of the hierarchies of concatenated pushdowns and a hierarchy of controlled languages established by Weir (1988; 1992). By this equivalence we know that the languages generated by the extensions of LIGs presented here are mildly context sensitive and therefore are comparable with extensions of TAGs and head grammars. Whether the languages studied in this paper and the languages generated by MCTAGs coincide, is a question that remains for future research.

References

BECKER T. (1994). A new automaton model for TAGs. *Computational Intelligence*, **10** (4), p. 422–430.

BREVEGLIERI L., CHERUBINI A., CITRINI C. & CRESPI REGHIZZI S. (1996). Multi-push-down languages and grammars. *International Journal of Foundations of Computer Science*, **7** (3), p. 253–291.

CHERUBINI A. & SAN PIETRO P. (1999)a. On the relation between multi-depth grammars and tree-adjoining grammars. *Publ. Math. Debrecen*, **54** (Supplement), p. 625–640.

CHERUBINI A. & SAN PIETRO P. (1999)b. On the relations between multi-depth grammars and label-distinguished control grammars. In C. L. NEHANIV & M. ITO, Eds., *Algebraic Engineering*, Singapore: World Scientific.

JOSHI A. K. (1987). An introduction to tree adjoining grammars. In A. MANASTER-RAMER, Ed., *Mathematics of Language*, p. 87–113. Amsterdam/Philadelphia: John Benjamins.

KROCH A. S. (1987). Unbounded dependencies and subjacency in a tree adjoining grammar. In A. MANASTER-RAMER, Ed., *Mathematics of Language*, p. 143–172. Amsterdam/Philadelphia: John Benjamins.

KROCH A. S. & JOSHI A. K. (1987). Analyzing extraposition in a tree adjoining grammar. In G. J. HUCK & A. E. OJEDA, Eds., *Discontinuous Constituency*, volume 20 of *Syntax and Semantics*, p. 107–149. Academic Press.

MICHAELIS J. & WARTENA C. (1999). LIGs with reduced derivation sets. In G. BOUMA, G.-J. M. KRUIJFF, E. HINRICHS & R. T. OEHRLE, Eds., *Constraints and Resources in Natural Language Syntax and Semantics*, volume II of *Studies in Constrained Based Lexicalism*, p. 263–279. Stanford, CA: CSLI Publications.

RAMBOW O. (1994). *Formal and Computational Aspects of Natural Language Syntax*. PhD thesis, University of Pennsylvania.

RIZZI L. (1990). *Relativized Minimality*. Cambridge, MA: MIT Press.

VIJAY-SHANKER K. & WEIR D. J. (1994). The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, **27** (6), p. 511–546.

WARTENA C. (1998). Grammars with composite storages. In *Proceedings of the Conference on Logical Aspects of Computational Linguistics (LACL '98)*, p. 11–14, Grenoble.

WARTENA C. (2000). Operations on storage types. Submitted.

WEIR D. J. (1988). *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, University of Pennsylvania.

WEIR D. J. (1992). A geometric hierarchy beyond context-free languages. *Theoretical Computer Science*, **104** (4), p. 235–261.

WEIR D. J. (1994). Linear iterated pushdowns. *Computational Intelligence*, **10** (4), p. 422–430.