On Minimalist Attribute Grammars and Macro Tree Transducers

Jens Michaelis and Uwe Mönnich and Frank Morawietz[*]
Seminar für Sprachwissenschaft
Universität Tübingen
Wilhelmstr. 113, 72074 Tübingen
Germany
{michael,um,frank}@sfs.nphil.uni-tuebingen.de

Abstract

In this paper we extend the work by Michaelis (1999) which shows how to encode an arbitrary Minimalist Grammar in the sense of Stabler (1997) into a weakly equivalent multiple context-free grammar (MCFG). By viewing MCFG-rules as terms in a free Lawvere theory we can translate a given MCFG into a regular tree grammar. The latter is characterizable by both a tree automaton and a corresponding formula in monadic second-order (MSO) logic. The trees of the resulting regular tree language are then unpacked into the intended "linguistic" trees both through an MSO transduction based upon tree-walking automata and through a macro tree transduction. This two-step approach gives an operational as well as a logical description of the tree sets involved. As an interlude we show that MCFGs can be regarded as a particularly simple attribute grammar.

1 Introduction

Algebraic, logical and regular characterizations of (tree) languages provide a natural framework for the denotational and operational semantics of grammar formalisms relying on the use of trees for their intended models.

Over the last couple of years, a rich class of mildly context-sensitive grammar formalisms has been proven to be weakly equivalent. Among others, the following families of (string) languages are identical: $STR(HR)$ [languages generated by string generating hyperedge replacement grammars], $OUT(DTWT)$ [output languages of deterministic tree-walking tree-to-string transducers], $yDT_{fc}(REGT)$ [yields of images of regular tree languages under deterministic finite-copying top-down tree transductions], $MCFL$ [languages generated by multiple context-free grammars], $MCTAL$ [languages generated by multi-component tree adjoining grammars], $LCFRL$ [languages generated by linear context-free rewriting systems], $LUSCL$ [languages generated by local unordered scattered context grammars] (more on these equivalences can be found, e.g., in Engelfriet, 1997; Rambow and Satta, 1999; Weir, 1992). It has not been noted before that MCFLs are strongly equivalent with a family of languages that are generated by a particularly simple form of attribute grammars (AGs). AGs are a much more powerful device than MCFGs and can easily accommodate phenomena of natural languages that are beyond the reach of the mildly context-sensitive grammar formalisms just enumerated. Analyzing multiple context-free grammars (MCFGs) as a special type of AG makes for a smooth transition into higher echelons of a refined Chomsky hierarchy without leaving the conceptual framework of AGs that has been successfully applied in such diverse areas as compiler construction and recursive program schemes.

In the present context the combination of algebraic, logical and regular techniques does not only add another characterization of mildly context-sensitive languages to the already long list of

[*]Corresponding author is Frank Morawietz.

weak equivalences. It also makes available the whole body of techniques that have been developed in the tradition of algebraic language theory, logic and automata theory.

In this paper we extend the work by Michaelis (1999) which shows how to encode an arbitrary minimalist grammar (MG) in the sense of Stabler (1997) (and thus in the sense of Stabler (1999a), the chain-based incarnation of the former) into a weakly equivalent linear context-free rewriting system (LCFRS). We present a translation from the formalism of multiple context-free grammars (MCFGs)—a weakly equivalent extension of LCFRSs—into regular tree grammars (RTGs). The idea behind the translation is to "lift" the MCFG-rules to RTG-rules by viewing them as Lawvere terms. Furthermore, we use the equivalence of RTGs, monadic second-order (MSO) logic (on trees) and tree automata to give an algebraic and a logical description of the lifted trees. Since the trees characterized by an RTG contain additional "non-linguistic" information they are then unpacked with a monadic second-order (MSO) transduction thereby giving both an operational and a denotational description of the tree sets involved. The MSO transduction is built upon a tree-walking automaton (with tests).

In addition to the result just mentioned, we present a further implementation of the MSO transduction by a very simple Macro Tree Transducer (MTT). MTTs are a powerful syntax-directed translation device in which the translation of an input tree not only depends on its subtrees but also on the context. The particular MTT we construct below takes the tree language generated by the regular tree grammar referred to above as its input and transduces it into the intended structures that are specified by the logical MSO transduction. It is worth mentioning that the tree transduction is given in terms of the "lifted" terms of the Lawvere algebra. It would have been possible to turn the MCFG—regarded as an AG—directly into an equivalent MTT. We hope that the additional step via the RTG helps to clarify the operational aspect of the implementation by means of the MTT.

We think that our approach has decisive advantages. First, the operations of the relevant signature appear explicitly in the lifted trees and are not hidden in node labels coding instances of rule application. Second, our path component is not dependent on the particular regular tree family or, equivalently, the domain defined via the MSO formula. The instruction set of the tree-walking automaton and the corresponding definition of the MSO transduction are universal and only serve to reverse the lifting process. In that sense the instructions are nothing else but a restatement of the unique homomorphism which exists between the free algebra and any other algebra of the same signature. The same remarks hold for the instructions of the MTT. Thus, the translation from MCFGs to RTGs constitutes a considerable simplification in comparison with other characterizations since it is not built upon derivation trees using productions of the original MCFG as node labels, but rather on the operations of projection, tuple-formation and composition alone.

In the following sections we limit ourselves in the running example to the special case of MCFG-rules with only one nonterminal on the right hand side (RHS). This allows a significant simplification in the presentation since it requires only one level of tupling. The extension to the general case of building tuples out of tuples is considerably more involved and would obfuscate the presentation of the core ideas unnecessarily. The definitions are given for the general case, though.

The structure of the paper is as follows. We start with some basic algebraic, logical and automata-theoretic definitions before sketching some linguistic motivation why the formal coding seems necessary, followed by the formal presentation of MGs. The succeeding sections then sketch how to translate a given MG firstly into an MCFG (and an equivalent AG) and from there into an RTG. Finally, in the last section, we transform the resulting trees back into a format which is intended for linguistic analysis. We conclude with a brief outlook on further optimizations of the presented technique.

2 Background and Basic Definitions

We are going to show how to code the grammar rules of a LCFRS (or better an MCFG) into rules of an RTG. This is done via lifting by viewing MCFG rules as terms in a free Lawvere theory.

Since this coding makes projection, tupling and composition explicit, the resulting trees contain these operations as labeled nodes. Therefore we use on the logical side an MSO transduction—where the regular tree language constitutes the domain—and on the operational side an MTT to transform the "lifted" trees into the intended ones.

In this section we present the corresponding basic algebraic, logical and automata-theoretic definitions before we proceed with the actual translation.

2.1 Basic Algebraic Definitions

Definition 2.1. For a given set of sorts $\mathcal{S}$,[1] a *many-sorted signature Σ (over $\mathcal{S}$)* is an indexed family $\langle \Sigma_{w,s} \mid w \in \mathcal{S}^*, s \in \mathcal{S} \rangle$ of disjoint sets. A symbol $\sigma \in \Sigma_{w,s}$ is an *operator of type $\langle w, s \rangle$*, *arity w*, *sort s* and *rank $|w|$*. The rank of σ is denoted by $rank(\sigma)$.

The *set of trees $T(\Sigma)$ (over Σ)* is built up using the operators in the usual way: If $\sigma \in \Sigma_{\varepsilon,s}$ for some $s \in \mathcal{S}$ then σ is a (trivial) tree of sort s. If, for some $s \in \mathcal{S}$ and $w = s_1 \cdots s_n$ with $s_i \in \mathcal{S}$, $\sigma \in \Sigma_{w,s}$ and $t_1, \ldots, t_n \in T(\Sigma)$ with t_i of sort s_i then $\sigma(t_1, \ldots, t_n)$ is a tree of sort s.

In case $\mathcal{S}$ is a singleton $\{s\}$, i.e., in case Σ is a *single-sorted signature (over sort s)*, we usually write Σ_n to denote the (unique) set of operators of rank $n \in \mathbb{N}$.[2]

The operator symbols of a many-sorted signature Σ over some set of sorts $\mathcal{S}$ induce operations on an algebra with the appropriate structure. A *Σ-algebra* $\mathbb{A}$ consists of an $\mathcal{S}$-indexed family $\langle A^s \mid s \in \mathcal{S} \rangle$ of disjoint sets, the carriers of $\mathbb{A}$, and for each operator $\sigma \in \Sigma_{w,s}$, $\sigma_{\mathbb{A}} : A^w \to A^s$ is a function, where $A^w = A^{s_1} \times \cdots \times A^{s_n}$ and $w = s_1 \cdots s_n$ with $s_i \in \mathcal{S}$. The set $T(\Sigma)$ can be made into a Σ-algebra $\mathbb{T}$ by specifying the operations as follows. For every $\sigma \in \Sigma_{w,s}$, where $s \in \mathcal{S}$ and $w = s_1 \cdots s_n$ with $s_i \in \mathcal{S}$, and every $t_1, \ldots, t_n \in T(\Sigma)$ with t_i of sort s_i we identify $\sigma_{\mathbb{T}}(t_1, \ldots, t_n)$ with $\sigma(t_1, \ldots, t_n)$.

Our main notion is that of an algebraic (Lawvere) theory.

Definition 2.2. Given a set of sorts $\mathcal{S}$, an *algebraic (Lawvere) theory*, as an algebra, is an $\mathcal{S}^* \times \mathcal{S}^*$-sorted algebra $\mathbb{A}$ whose carriers $\langle A^{\langle u,v \rangle} \mid u, v \in \mathcal{S}^* \rangle$ consist of the morphisms of the theory and whose operations are of the following types, where $n \in \mathbb{N}$, $u = u_1 \cdots u_n$ with $u_i \in \mathcal{S}^*$ for $1 \leq i \leq n$ and $v, w \in \mathcal{S}^*$,

$$\begin{aligned}
&\text{projection:} && \pi_i^u \in A^{\langle u, u_i \rangle} \\
&\text{composition:} && c_{(u,v,w)} \in A^{\langle u,v \rangle} \times A^{\langle v,w \rangle} \to A^{\langle u,w \rangle} \\
&\text{target tupling:} && (\quad)_{(v,u)} \in A^{\langle v,u_1 \rangle} \times \cdots \times A^{\langle v,u_n \rangle} \to A^{\langle v,u \rangle}
\end{aligned}$$

The projections and the operations of target tupling are required to satisfy the obvious identities for products. The composition operations must satisfy associativity, i.e.,

$$\begin{aligned}
c_{(v,u,u_i)}\big((\alpha_1, \ldots, \alpha_n)_{(v,u)}, \pi_i^u\big) &= \alpha_i \quad \text{for} \quad \alpha_i \in A^{\langle v,u_i \rangle}, 1 \leq i \leq n \\
\big(c_{(v,u,u_1)}(\beta, \pi_1^u), \ldots, c_{(v,u,u_n)}(\beta, \pi_n^u)\big)_{(v,u)} &= \beta \quad \text{for} \quad \beta \in A^{\langle v,u \rangle} \\
c_{(u,v,z)}\big(\alpha, c_{(v,w,z)}(\beta, \gamma)\big) &= c_{(u,w,z)}\big(c_{(u,v,w)}(\alpha, \beta), \gamma\big) \\
&\qquad \text{for} \quad \alpha \in A^{\langle u,v \rangle}, \beta \in A^{\langle v,w \rangle}, \gamma \in A^{\langle w,z \rangle} \\
c_{(u,u,v)}\big((\pi_1^u, \ldots, \pi_n^u)_{(u,u)}, \alpha\big) &= \alpha \quad \text{for} \quad \alpha \in A^{\langle u,v \rangle}
\end{aligned}$$

where $u = u_1 \cdots u_n$ with $u_i \in \mathcal{S}^*$ for $1 \leq i \leq n$ and $v, w, z \in \mathcal{S}^*$.

[1] Throughout the paper the following conventions apply. $\mathbb{N}$ is the set of all non-negative integers. For any set M, M^* is the Kleene closure of M, i.e., the set of all finite strings over M. For $m \in M^*$, $|m| \in \mathbb{N}$ denotes the length of m. We will use M_ε to denote the set $M \cup \{\varepsilon\}$, where ε is the empty string (over M), i.e., $\varepsilon \in M^*$ with $|\varepsilon| = 0$.

[2] Note that for $\mathcal{S} = \{s\}$ each $\langle w, s \rangle \in \mathcal{S}^* \times \mathcal{S}$ is of the form $\langle s^n, s \rangle$ for some $n \in \mathbb{N}$.

Let Σ be a single-sorted signature and $X = \{x_1, x_2, x_3, \ldots\}$ a countable set of variables. For $k \in \mathbb{N}$ define $X_k \subseteq X$ as $\{x_1, \ldots, x_k\}$. Then, the set of *k-ary trees* $T(\Sigma, X_k)$ *(over Σ)* is the set of trees $T(\Sigma')$ over the single-sorted signature $\Sigma' = \langle \Sigma'_n \mid n \in \mathbb{N} \rangle$, where $\Sigma'_0 = \Sigma_0 \cup X_k$ and $\Sigma'_n = \Sigma_n$ for $n > 0$. Note that $T(\Sigma, X_k) \subseteq T(\Sigma, X_l)$ for $k \leq l$. Let $T(\Sigma, X) = \bigcup_{k \in \mathbb{N}} T(\Sigma, X_k)$.

The power set $\wp(T(\Sigma,X))$ of $T(\Sigma, X)$ constitutes the central example of interest for formal language theory. The carriers $\langle \wp(T(k, m)) \mid k, m \in \mathbb{N} \rangle$ of the corresponding $\mathcal{S}^* \times \mathcal{S}^*$-Lawvere algebra are constituted by the power sets of the sets $T(k, m)$, where each $T(k, m)$ is the set of all m-tuples of k-ary trees, i.e., $T(k, m) = \{(t_1, \ldots, t_m) \mid t_i \in T(\Sigma, X_k)\}$.[3] For $i, k \in \mathbb{N}$ with $1 \leq i \leq k$ the projection constant π_i^k is defined as $\{x_i\}$. Composition is defined as substitution of the projection constants and target tupling is just tupling.

As remarked previously, an arbitrary number of nonterminals on the RHS of an MCFG-rule entails the use of tuples of tuples in the definition of the corresponding mapping. That is to say, each nonterminal on the RHS generates a tuple of terminal strings rather than a single string (cf. Def. 5.1). Therefore we had to define the Lawvere algebra in such a way that each component u_i of any u is from $\mathcal{S}^*$. Since in the running example we use only rules with one nonterminal on the RHS, each u_i we employ there is of length one (i.e., from $\mathcal{S}$) such that we can safely ignore the "outer" tupling.

More on Lawvere theories in general can be found in, e.g., Wagner (1994). More on the connection to linguistics is elaborated in Mönnich (1998).

2.2 Attribute Grammars

In order to define an attribute grammar (AG) we follow Bloem and Engelfriet (1998) who in their turn adopt the variant introduced in Fülöp (1981). Below we are interested only in AGs where the set of inherited attributes is empty, i.e., in AGs which are *only synthesized*. Nevertheless, we present the general definition of an AG in order to explicitly state the difference from the restricted versions needed afterwards.

Definition 2.3. An *attribute grammar (AG)* is a 7-tuple $G = \langle \Sigma, \mathsf{A}_{\mathtt{syn}}, \mathsf{A}_{\mathtt{in}}, \Omega, W, R, \alpha_{mean} \rangle$, where Σ is a many-sorted signature of *operation symbols*, $\mathsf{A}_{\mathtt{syn}}$ is a set of *synthesized attributes*, $\mathsf{A}_{\mathtt{in}}$ is a set of *inherited attributes*, Ω is a finite set of sets, the *semantic domains* of the attributes, W, the *domain assignment*, is a function from A to Ω, $\alpha_{mean} \in \mathsf{A}_{\mathtt{syn}}$ is the *meaning attribute*, $R = \bigcup_{\sigma \in \Sigma \cup \{root\}} R(\sigma)$ is the set of semantic rules, *root* a new symbol not appearing in Σ.

For each $\sigma \in \Sigma$, $R(\sigma)$ is the set of *internal rules (for σ)* such that for each $\alpha_0 \in \mathsf{A}$ there is one $r \in R(\sigma)$ of the form $\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle)$, where either $\alpha_0 \in \mathsf{A}_{\mathtt{syn}}$ and $i_0 = 0$, or $\alpha_0 \in \mathsf{A}_{\mathtt{in}}$ and $1 \leq i_0 \leq rank(\sigma)$. Furthermore we have $k \in \mathbb{N}$, $\alpha_1, \ldots, \alpha_k \in \mathsf{A}_{\mathtt{syn}} \cup \mathsf{A}_{\mathtt{in}}$, $i_1, \ldots, i_k \in \{0, \ldots rank(\sigma)\}$ such that the $\langle \alpha_j, i_j \rangle$'s are pairwise distinct, and f is a function from $W(\alpha_1) \times \cdots \times W(\alpha_k)$ to $W(\alpha_0)$.

$R(root)$ is the set of *root rules* such that for each $\sigma \in \Sigma$ and $\alpha_0 \in \mathsf{A}_{\mathtt{in}}$ there is one $r \in R(\sigma)$ of the form $\langle \alpha_0, 0 \rangle = f(\langle \alpha_1, 0 \rangle, \ldots, \langle \alpha_k, 0 \rangle)$, where $k \in \mathbb{N}$, $\alpha_1, \ldots, \alpha_k \in \mathsf{A}_{\mathtt{syn}} \cup \mathsf{A}_{\mathtt{in}}$, the α_j's are pairwise distinct, and f is a function from $W(\alpha_1) \times \cdots \times W(\alpha_k)$ to $W(\alpha_0)$.

Consider $t \in T(\Sigma)$. We will identify t with its corresponding labeled tree domain $\langle V_t, lab_t \rangle$ here.[4]

$\mathsf{A}(t) = (\mathsf{A}_{\mathtt{syn}} \cup \mathsf{A}_{\mathtt{in}}) \times V_t$ is the set of attributes of t. $R(t)$ is the set of *semantic instructions* of t which is defined as follows: Let $u \in V_t$ with $lab_t(u) = \sigma$. If $\langle \alpha_0, i_0 \rangle = f(\langle \alpha_1, i_1 \rangle, \ldots, \langle \alpha_k, i_k \rangle)$ belongs to $R(\sigma)$ then $\langle \alpha_0, ui_0 \rangle = f(\langle \alpha_1, ui_1 \rangle, \ldots, \langle \alpha_k, ui_k \rangle)$ is in $R(t)$ and called an *internal instruction (of t)*. Furthermore, if $\langle \alpha_0, 0 \rangle = f(\langle \alpha_1, 0 \rangle, \ldots, \langle \alpha_k, 0 \rangle)$ is in $R(root)$ then $\langle \alpha_0, root(t) \rangle = f(\langle \alpha_1, root(t) \rangle, \ldots, \langle \alpha_k, root(t) \rangle)$ is in $R(t)$ and called a *root instruction (of t)*.

[3]Since $\mathcal{S}$ is a singleton, $\mathcal{S}^*$ can be identified with $\mathbb{N}$, because up to length each $w \in \mathcal{S}^*$ is uniquely specified (cf. fn. 2).

[4]Thus, V_t is a unique *prefix closed* and *left closed* subset of $(\mathbb{N} \setminus \{0\})^*$, i.e., $\chi \in V_t$ if $\chi\chi' \in V_t$, and $\chi i \in V_t$ if $\chi j \in V_t$ for $\chi, \chi' \in (\mathbb{N} \setminus \{0\})^*$ and $i, j \in \mathbb{N} \setminus \{0\}$ with $i < j$. lab_t is the labeling function from V_t to Σ which canonically assigns to each node in V_t an element in Σ by "following bottom-up" the inductive definition of t. For reasons of consistency we denote the empty string $\varepsilon \in (\mathbb{N} \setminus \{0\})^*$ by 0 in the context of tree domains.

The *dependency graph (of t)* is the pair $D(t) = \langle \mathsf{A}(t), E \rangle$, where E consists of all edges $\langle \langle \alpha, u \rangle, \langle \alpha', u' \rangle \rangle$ such that there is a $\langle \alpha', u' \rangle = f(\langle \alpha_1, u_1 \rangle, \ldots, \langle \alpha_k, u_k \rangle)$ in $R(t)$ with $\langle \alpha, u \rangle = \langle \alpha_i, u_i \rangle$ for some $1 \leq i \leq k$.

The values of the attributes of t are defined via a function *dec* from $\mathsf{A}(t)$ to $\bigcup \Omega$ such that $dec(\langle \alpha, u \rangle) \in W(\alpha)$. The function *dec* is called a *decoration (of t)* if all semantic instructions are satisfied, i.e., for each $\langle \alpha_0, u_0 \rangle = f(\langle \alpha_1, u_1 \rangle, \ldots, \langle \alpha_k, u_k \rangle) \in R(t)$, $dec(\langle \alpha_0, u_0 \rangle) = f(dec(\langle \alpha_1, u_1 \rangle), \ldots, dec(\langle \alpha_k, u_k \rangle)) \in R(t)$. Note that, if the corresponding dependency graph of $D(t)$ is non-circular, t has a unique decoration.

2.3 Basic Logical Definitions

After these algebraic notions, we briefly present those related to monadic second-order (MSO) logic. MSO logic is the extension of first-order predicate logic with monadic second-order variables and quantification over them. In particular, we are using MSO logic on trees such that individual variables $x, y, \ldots$ stand for nodes in trees and monadic second-order ones $X, Y, \ldots$ for sets of nodes (for more details see, e.g., Rogers, 1998). It is well-known that MSO logic interpreted on trees is decidable via a translation to finite-state (tree) automata (Rabin, 1969; Doner, 1970; Thatcher and Wright, 1968). The decidability proof for MSO on finite trees gives us also a descriptive complexity result: MSO on finite trees yields only recognizable trees which in turn yield context-free string languages. These results are of particular interest, since finite trees are clearly relevant for linguistic purposes, and therefore form the basis for our work.

The following paragraphs go directly back to Courcelle (1997). Recall that the representation of objects within relational structures makes them available for the use of logical description languages. Let R be a finite set of relation symbols with the corresponding arity for each $r \in R$ given by $\rho(r)$. A relational structure $\mathcal{R} = \langle D_{\mathcal{R}}, (r_{\mathcal{R}})_{r \in \mathcal{R}} \rangle$ consists of the domain $D_{\mathcal{R}}$ and the $\rho(r)$-ary relations $r_{\mathcal{R}} \subseteq D_{\mathcal{R}}^{\rho(r)}$. In our case we choose a finite tree as our domain and the relations of immediate, proper and reflexive dominance and precedence.

The classical technique of interpreting a relational structure within another one forms the basis for MSO transductions. Intuitively, the output tree is interpreted on the input tree. E.g., suppose that we want to transduce the input tree t_1 into the output tree t_2. The nodes of the output tree t_2 will be a subset of the nodes from t_1 specified with a unary MSO relation ranging over the nodes of t_1. The daughter relation will be specified with a binary MSO relation with free variables x and y ranging over the nodes from t_1. We will use this concept to transform the lifted trees into the intended ones.

Definition 2.4. A *(non-copying) MSO transduction* of a relational structure $\mathcal{R}$ (with set of relation symbols R) into another one $\mathcal{Q}$ (with set of relation symbols Q) is defined to be a tuple $(\varphi, \psi, (\theta_q)_{q \in Q})$. It consists of the formulas φ defining the domain of the transduction in $\mathcal{R}$ and ψ defining the resulting domain of $\mathcal{Q}$ and a family of formulas θ_q defining the new relations $q \in Q$ (using only definable formulas from the "old" structure $\mathcal{R}$).

The result which gives rise to the fact that we can characterize a non-context-free tree set with two devices which have only regular power is stated in Courcelle (1997). Viewing the relation of intended dominance defined later by a tree-walking automaton as the cornerstone of an MSO definable transduction, our description of non-context-free phenomena with two devices with only regular power is an instance of the theorem that the image of an MSO-definable class of structures under a definable transduction is not MSO definable in general (Courcelle, 1997).

2.4 Basic Automata-Theoretic Definitions

Tree automata are the result of generalizing the transition function of standard finite-state automata from (state-alphabet) symbol pairs to tuples of states. Intuitively, bottom-up tree automata creep up a tree from the leaves to the root by simultaneously taking the states of the daughters and the alphabet symbol of the mother to make a transition to a new state. Since we will not give an example of a tree automaton in this paper, we will not specify further details. More on tree automata can be found in, e.g., Gécseg and Steinby (1984).

Definition 2.5 (Tree Automaton). A *(deterministic) bottom-up tree automaton* $\mathfrak{A}$ is a 5-tuple $\langle A, \Sigma, \delta, a_0, F\rangle$ with A the (finite) set of states, Σ a ranked alphabet, $a_0 \in A$ the initial state, $F \subseteq A$ the final states and $\delta : \bigcup_n (A^n \times \Sigma_n) \to A$ the transition function.

We can extend the transition function inductively to trees by defining $h_\delta(\varepsilon) = a_0$ and $h_\delta(\sigma(t_1, \dots, t_n)) = \delta(h_\delta(t_1), \dots, h_\delta(t_n), \sigma)$, $t_i \in T_\Sigma, 1 \leq i \leq n, \sigma \in \Sigma_n$. An automaton $\mathfrak{A}$ accepts a tree $t \in T_\Sigma$ iff $h_\delta(t) \in F$. The language recognized by $\mathfrak{A}$ is denoted by $T(\mathfrak{A}) = \{t \mid h_\delta(t) \in F\}$.

The sets of trees recognized by bottom-up tree automata are called recognizable, i.e., regular sets of trees, and, as mentioned previously, yield context-free string languages (Gécseg and Steinby, 1984). The recognizable sets are closed under the boolean operations of conjunction, disjunction and negation. The automaton constructions which underlie these closure results are generalizations of the corresponding better-known constructions for finite state automata (FSA). The recognizable sets are also closed under (inverse) projections, and again the construction is essentially that for finite state automata. The projection construction yields a nondeterministic automaton, but, again as for FSA's, bottom-up tree automata can be made deterministic by a straightforward generalization of the subset construction.[5] Finally, tree automata can be minimized by a construction which is, yet again, a straightforward generalization of well known FSA techniques.

We need another type of finite-state machine later in the paper: Macro Tree Transducers (MTTs). Since those are not so well known, we will introduce them via the more accessible standard top-down tree transducers. These are not so different from the bottom-up tree automata introduced above. Instead of working from the leaves towards the root, top-down tree transducers start from the root and work their way downward to the leaves. And, of course, they produce an output tree along the way. In the following paragraphs we will use the notation and presentation introduced in Engelfriet and Vogler (1985). Our presentation is also inspired by Engelfriet and Maneth (2000). A full introduction to tree transductions can be found in Gécseg and Steinby (1997).

Intuitively, top-down tree transducers transform trees over a ranked alphabet Σ into ones over a ranked alphabet Ω. They traverse a tree from the root to the leaves (the input tree) and output on each transition a new tree whose nodes can contain labels from both alphabets, states and variables. More formally, the RHSs of such a production are trees from $T_\Omega(\Sigma(X) \cup Q)$. For this definition we assume that Q is a ranked alphabet containing only unary symbols.

Definition 2.6 (Top-Down Tree Transducer). A top-down tree transducer (TDTT) is a a tuple $T = \langle Q, \Sigma, \Omega, q_0, P\rangle$ with states Q, ranked alphabets Σ and Ω (input and output), initial state q_0 and a finite set of productions P of the form

$$q(\sigma(x_1, \dots, x_n)) \longrightarrow t$$

where $n \geq 0$, $\sigma \in \Sigma_n$ and $t \in T_\Omega(\Sigma(X) \cup Q)$.

The transition relation ($\overset{T}{\Longrightarrow}$) is defined as usual. The transduction realized by a top-down tree transducer T is then defined to be $\{(t_1, t_2) \in T_\Sigma \times T_\Omega \mid q_0(t_1) \overset{T}{\Longrightarrow}^{*} t_2\}$.

Consider as a very simple example the transducer T which maps binary trees whose interior nodes are labeled with a's into ternary trees whose interior nodes are labeled with b's. The leaves are labeled with p and are transduced into q's. Furthermore, new leaves labeled c are introduced at every branching point. Σ consists of one binary symbol a and one constant p, Ω of one ternary symbol b and two constants q and c. The transducer has only one state q_0 and the two productions below:

$$\begin{aligned} q_0(a(x_1, x_2)) &\longrightarrow b(q_0(x_1), c, q_0(x_2)) \\ q_0(p) &\longrightarrow q \end{aligned}$$

Fig. 1 shows one application of the nontrivial rule. The left hand side (LHS) displays the rule in tree notation whereas the RHS displays an actual transition.

[5] Note that *top-down* tree automata do not have this property: deterministic top-down tree automata recognize a strictly narrower family of tree sets.

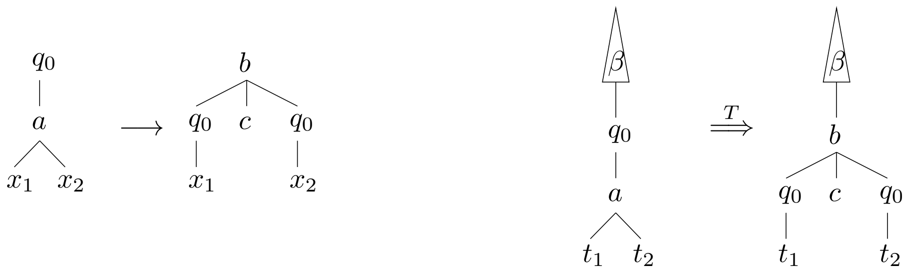


Figure 1: One step of an TDTT derivation

If we have already transduced a subtree β of the input and are in state q_0 and currently working on a node labeled with a with immediate subtrees t_1 and t_2, then we can rewrite it into a tree labeled with b whose leftmost and rightmost daughter are in state q_0 applied to t_1 and t_2 respectively and the middle daughter is labeled with the terminal symbol c.

By generalizing the set of states to a ranked alphabet, we can extend the notion of a top-down tree transducer to a macro tree transducer. This allows to pass parameters—which contain a limited amount of context information from the part of the input tree we have already seen—into the RHSs. We formalize these new RHSs as follows:

Definition 2.7. Let Σ and Ω be ranked alphabets and $n, m \geq 0$. The set of right hand sides $RHS(\Sigma, \Omega, n, m)$ over Σ and Ω with n variables and m parameters is the smallest set $rhs \subseteq T_{\Sigma\cup\Omega}(X_n \cup Y_m)$ such that

1. $Y_m \subseteq rhs$
2. For $\omega \in \Omega_k$ with $k \geq 0$ and $\varphi_1, \ldots, \varphi_k \in rhs$, $\omega(\varphi_1, \ldots, \varphi_k) \in rhs$
3. For $q \in Q_{k+1}$ with $k \geq 0$, $x_i \in X_n$ and $\varphi_1, \ldots, \varphi_k \in rhs$, $q(x_i, \varphi_1, \ldots, \varphi_k) \in rhs$

The productions of MTTs contain one piece of "old" information (a symbol from the input alphabet with the appropriate number of variables) and a number of context parameters.

Definition 2.8 (Macro Tree Transducer). A macro tree transducer (MTT) is a five-tuple $M = \langle Q, \Sigma, \Omega, q_0, P\rangle$ with Q a ranked alphabet of states, ranked alphabets Σ and Ω (input and output), initial state q_0 of rank 1, and a finite set of productions P of the form

$$q(\sigma(x_1, \ldots, x_n), y_1, \ldots, y_m) \longrightarrow t$$

where $n, m \geq 0$, $q \in Q_{m+1}$, $\sigma \in \Sigma_n$ and $t \in RHS(\Sigma, \Omega, n, m)$.

The productions $p \in P$ of M are used as term rewriting rules in the usual way. The transition relation of M is denoted by $\overset{M}{\Longrightarrow}$.

The transduction realized by M is the function $\{(t_1, t_2) \in T_\Sigma \times T_\Omega \mid (q_0, t_1) \overset{M}{\Longrightarrow}^* t_2\}$

Generally, a little care has to be taken in the definition of the transition relation with respect to the occuring parameters y_i. Derivations are dependent on the order of tree substitutions. Inside-out (IO) means that trees from T_Ω have to be substituted for the parameters whereas in Outside-in (OI) derivations a subtree must not be rewritten if it is in some context parameter. Again, as for context-free tree grammars, neither class of derivations contains the other. Since we are only dealing with *simple* MTTs in our approach, all three modes are equivalent and can safely be ignored.

An MTT is *deterministic* if for each pair $q \in Q_{m+1}$ and $\sigma \in \Sigma_n$ there is at most one rule in P with $q(\sigma(x_1, \ldots, x_n), y_1, \ldots, y_m)$ on the LHS.

An MTT is called *simple* if it is simple in the input (i.e., for every $q \in Q_{m+1}$ and $\sigma \in \Sigma_n$, each $x \in X_k$ occurs exactly once in $RHS(\Sigma, \Omega, n, m)$) and simple in the parameters (i.e., for every $q \in Q_{m+1}$ and $\sigma \in \Sigma_k$, each $y \in Y_m$ occurs exactly once in $RHS(\Sigma, \Omega, n, m)$). The MTTs discussed in the remainder of the paper will all be simple.

Note that if we disregard the input, MTTs turn into context-free tree grammars.

Consider for example the following rule of an MTT M.

$$q_0(a(x_1, x_2), y_1, y_2, y_3) \longrightarrow b(x_1, b(q_0(y_1)), q_0(y_2), y_3, q_0(x_2))$$

Analogous to the presentation in Fig. 1, we illustrate the the rule above in Fig. 2 without being too concerned about the formal details of specifying a full transducer.

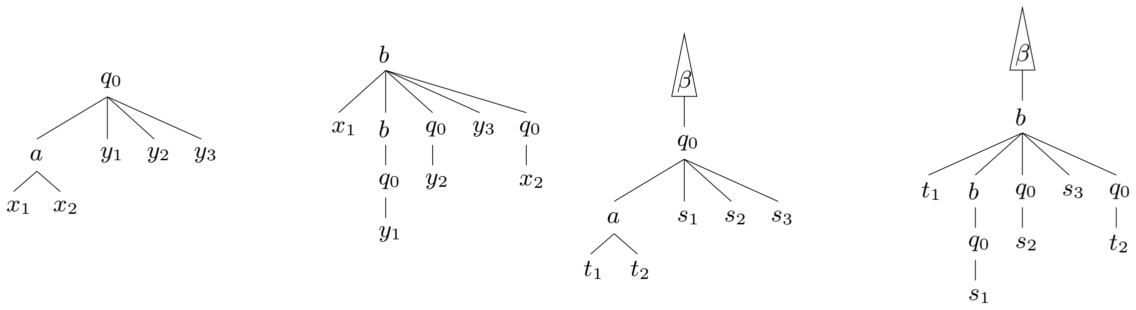


Figure 2: One step of an MTT derivation

The only difference (apart from a totally different transduction) is that we now have parameters which appear as trees s_1 through s_3. Those trees can also be freely used on the RHSs of the MTT productions.

3 Linguistic Motivation: Verb Raising

The exercise in formal coding we present in this paper is made necessary by the fact that natural language sports at least some constructions which lead (i) to non-context-free string languages, or (ii) to at least non-recognizable tree languages (i.e., tree sets which cannot be generated by any context-free string grammar or regular tree grammar), even though in the latter case the resulting string languages may formally be context-free.[6] Both phenomena show up in the West-Germanic languages: the verbal complex of Züritüütsch is an example of (i), while (ii) is exhibited—for different reasons—by the corresponding constructions of Dutch and Standard German (Huybregts, 1976; 1984):

(1) a.

weil	der Karl	die Maria	dem Peter	den Hans	schwimmen	lehren	helfen	läßt
because	Charles	Mary$_1$	Peter$_2$	John$_3$	swim$_3$-inf	teach$_2$-inf	help$_1$-inf	lets

(German fragment as string language: palindrome language—CF)

b.

omdat	Karel	Marie	Piet	Jan	laat	helpen	leren	zwemmen
because	Charles	Mary$_1$	Peter$_2$	John$_3$	lets	help$_1$-inf	teach$_2$-inf	swim$_3$-inf

(Dutch fragment as string language: $a^n b^n$—CF)

c.

wil	de Karl	d'Maria	em Peter	de Hans	laat	hälffe	lärne	schwüme
because	Charles	Mary$_1$	Peter$_2$	John$_3$	lets	help$_1$-inf	teach$_2$-inf	swim$_3$-inf

(Züritüütsch fragment as string language: $a^n b^m c^n d^m$—Non-CF)

'because Charles lets Mary help Peter to teach John to swim'

[6]Let us note here that it is not the goal of this section to attempt a linguistically relevant discussion of cross-serial dependencies. All we want to show is that a formalism for natural languages has to handle non-context-free structures. For a serious introduction of approaches to cross-serial dependencies see Pullum and Gazdar (1982).

The structure of the preceding example illustrates non-context-free phenomena. On a close look at, e.g., the Swiss German example, we note that the DP's and the V's of which the DP's are objects occur in cross-serial order. This is manifested by the case marking of the respective objects. It appears that there are no limits on the length of such constructions in grammatical sentences of Swiss German. This fact alone would not suffice to prove that Swiss German is not a context-free string language. It could still be the case that Swiss German *in toto* is context-free even though it subsumes an isolable context-sensitive fragment. Relying on the closure of context-free languages under intersection with regular languages, Shieber (1985) was able to show that not only the fragment exhibiting the cross-serial dependencies but the whole of the language has to be assumed to be non-context-free.[7]

Abstracting from the details of the particular languages, the standard analyses of these constructions involve the following property which is problematic from the point of view of context-freeness: In all cases they posit roughly a bipartite structure like the one in Fig. 3 with basically all DP's on one branch and all the verbs on the other – but with fixed syntactic and semantic relations between the branches, whether visibly marked (as in Züritüütsch, Standard German) or not (Dutch).

As is easily seen, there is no context-free device which could directly handle the unbounded number of non-local dependencies the structural separation of the two "clusters" enforces. Therefore MSO logic alone cannot be sufficient for linguistic reasons. In order to concentrate on the relevant details, we will use as an artificial example a grammar generating the non-context-free copy language *ww* presented in the following sections to illustrate our proposal.

4 Minimalist Grammars

We first give the definition of a minimalist grammar (MG) along the lines of Stabler (1997). In order to keep the presentation simple, we omit the cases of *strong selection* (triggering *head movement*), and *covert movement*. Thus the definition given here comes close to the one given in Stabler (1999b), where some further restrictions are formulated as to which subtrees of a given tree may move. In fact, the example MG which will be considered below respects both the definition in Stabler (1997) as well as that in Stabler (1999b).

Definition 4.1. For a given set (of features), *Feat*, a five-tuple $\tau = (N_\tau, \lhd^*_\tau, \prec_\tau, <_\tau, \mathit{Label}_\tau)$ fulfilling (E1)–(E3) is called an *expression (over Feat)*.

(E1) $(N_\tau, \lhd^*_\tau, \prec_\tau)$ is a finite, binary ordered tree (domain). N_τ denotes the non-empty set of nodes. $\lhd^*_\tau$ and $\prec_\tau$ denote the usual relations of *dominance* and *precedence* defined on a subset of $N_\tau \times N_\tau$, respectively. I.e., $\lhd^*_\tau$ is the reflexive and transitive closure of $\lhd_\tau$, the relation of *immediate dominance*.[8]

(E2) $<_\tau \subseteq N_\tau \times N_\tau$ denotes the asymmetric relation of *(immediate) projection* which holds for any two siblings in $(N_\tau, \lhd^*_\tau, \prec_\tau)$, i.e., each node different from the root either *(immediately) projects* over its sibling or vice versa.

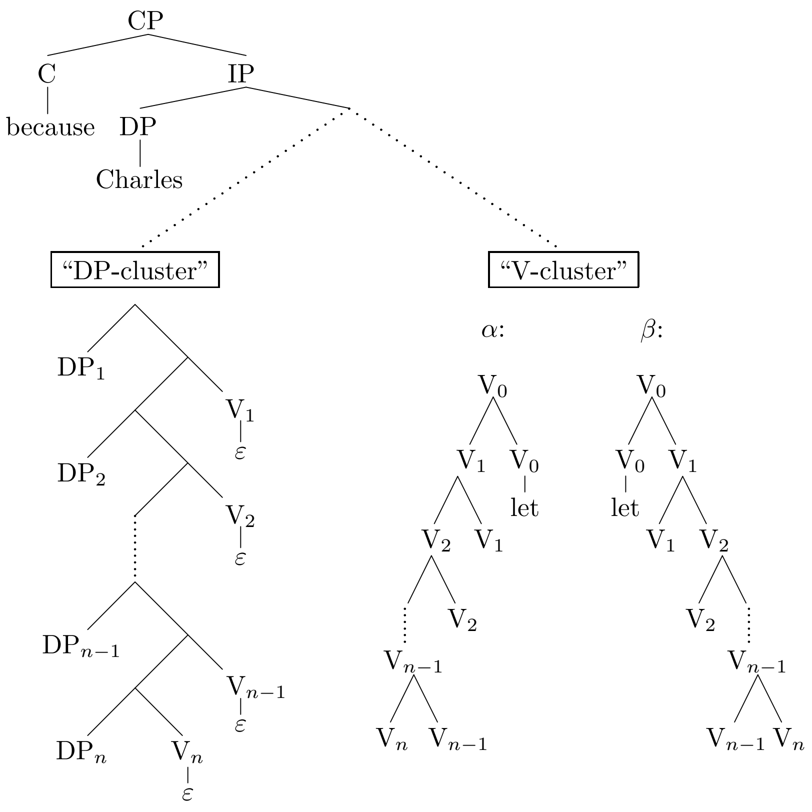
(E3) The function Label_τ assigns a string from Feat^* to every leaf of $(N_\tau, \lhd^*_\tau, \prec_\tau)$, i.e., a leaf-label is a finite sequence of features from *Feat*.

The set of all expressions over *Feat* is denoted by *Exp*(*Feat*).

Let *Feat* be a set of features. Consider $\tau = (N_\tau, \lhd^*_\tau, \prec_\tau, <_\tau, \mathit{Label}_\tau) \in \mathit{Exp}(\mathit{Feat})$.

[7] Huybregts (1984) provides a similar argument for Dutch taking into account a particular fragment: in contrast to Swiss German, Dutch does not show overt case-marking of objects. Huybregts' argument crucially relies on a given morphologized—and thus syntactical—difference between animate and inanimate pronominals.

[8] I.e., $\chi \lhd_\tau \psi$ iff $\psi = \chi i$ for some $i \in \mathbb{N} \setminus \{0\}$, and $\chi \prec_\tau \psi$ iff $\chi = \omega i \chi'$ and $\psi = \omega j \psi'$ for some $\omega, \chi', \psi' \in (\mathbb{N} \setminus \{0\})^*$ and $i, j \in \mathbb{N} \setminus \{0\}$ with $i < j$. Recall that N_τ is a unique *prefix closed* and *left closed* subset of $(\mathbb{N} \setminus \{0\})^*$ (cf. fn. 4).



	overt case	V-cluster
German	+	α
Dutch	$-$	β
Züritüütsch	+	β

Figure 3: The structure of Germanic VR

Each $x \in N_\tau$ has a *head* $h(x) \in N_\tau$, a leaf such that $x \lhd_\tau^* h(x)$, and such that each $y \in N_\tau$ on the path from x to $h(x)$ with $y \neq x$ projects over its sister. The *head of* τ is the head of τ's root.

A subtree υ of τ is a *maximal projection (in* τ*)*, if the root of υ is a node $x \in N_\tau$ such that x is the root of τ or x's sister projects over x. The sister of the head of τ is the *complement (of* τ*)*. Each maximal projection in τ which is not dominated by the mother of the head of τ is a *specifier (of* τ*)*.

τ *has feature* $f \in \mathit{Feat}$ if τ's head-label starts with f. τ is *simple* (a *head*) if it consists of exactly one node, otherwise τ is *complex* (a *non-head*).

Let r_τ be the root of τ. Suppose υ and $\varphi \in \mathit{Exp}(\mathit{Feat})$ to be subtrees of τ with roots r_υ and r_φ, respectively, such that $r_\tau \lhd_\tau r_\upsilon, r_\varphi$. Then we take $[_< \upsilon, \varphi\,]$ ($[_> \varphi, \upsilon\,]$) to denote τ in case that $r_\upsilon <_\tau r_\varphi$ and $r_\upsilon \prec_\tau r_\varphi$ ($r_\varphi \prec_\tau r_\upsilon$).

Definition 4.2 (Stabler, 1997). A 4-tuple $G = \langle \mathit{Non\text{-}Syn}, \mathit{Syn}, \mathit{Lex}, \mathcal{F} \rangle$ that obeys (M1)–(M4) is called a *minimalist grammar (MG)*.

(M1) *Non-Syn* is a finite set of *non-syntactic features* partitioned into a set *Phon* of *phonetic features* and a set *Sem* of *semantic features*.

(M2) *Syn* is a finite set of *syntactic features* partitioned into the sets *Base*, *Select*, *Licensees* and *Licensors* such that for each *(basic) category* $\mathtt{x} \in \mathit{Base}$ the existence of $\mathtt{{}^{=}x} \in \mathit{Select}$ is possible, and for each $\mathtt{-x} \in \mathit{Licensees}$ the existence of $\mathtt{+X} \in \mathit{Licensors}$ is possible. Moreover, the set *Base* contains at least the category $\mathtt{c}$.

(M3) *Lex* is a finite set of expressions over $\mathit{Feat} = \mathit{Non\text{-}Syn} \cup \mathit{Syn}$ such that for each tree $\tau = \langle N_\tau, \lhd_\tau^*, \prec_\tau, <_\tau, \mathit{Label}_\tau \rangle \in \mathit{Lex}$ the function Label_τ assigns a string from $\mathit{Select}^* \mathit{Licensors}_\varepsilon \mathit{Select}^* \mathit{Base}_\varepsilon \mathit{Licensees}^* \mathit{Phon}^* \mathit{Sem}^* \subseteq \mathit{Feat}^*$ to each leaf in $\langle N_\tau, \lhd_\tau^*, \prec_\tau \rangle$.

(M4) The set $\mathcal{F}$ consists of the structure building functions *merge* and *move* as defined in (me) and (mo), respectively.

(me) The function *merge* is a partial mapping from $\mathit{Exp}(\mathit{Feat}) \times \mathit{Exp}(\mathit{Feat})$ to $\mathit{Exp}(\mathit{Feat})$. A pair of expressions $\langle \upsilon, \varphi \rangle$ belongs to $\mathrm{Dom}(\mathit{merge})$ if υ has feature $\mathtt{{}^{=}x}$ and φ has category $\mathtt{x}$ for some $\mathtt{x} \in \mathit{Base}$.[9] Then,

(me.1) $$\mathit{merge}(\upsilon, \varphi) = [_< \upsilon', \varphi'\,] \quad \text{if } \upsilon \text{ is simple and has feature } \mathtt{{}^{=}x},$$

where υ' and φ' are expressions resulting from υ and φ, respectively, by deleting the feature the respective head-label starts with.

(me.2) $$\mathit{merge}(\upsilon, \varphi) = [_> \varphi', \upsilon'\,] \quad \text{if } \upsilon \text{ is complex and has feature } \mathtt{{}^{=}x},$$

where υ' and φ' are expressions as in case (me.1).

(mo) The function *move* is a partially defined mapping from $\mathit{Exp}(\mathit{Feat})$ to $\mathit{Exp}(\mathit{Feat})$. An expression υ belongs to $\mathrm{Dom}(\mathit{move})$ in case that υ has feature $\mathtt{+X} \in \mathit{Licensors}$, and υ has exactly one subtree φ that is a maximal projection and has feature $\mathtt{-x} \in \mathit{Licensees}$. Then,

$$\mathit{move}(\upsilon) = [_> \varphi', \upsilon'\,] \quad \text{if } \upsilon \text{ has feature } \mathtt{+X}$$

Here υ' results from υ by deleting the feature $\mathtt{+X}$ from υ's head-label, while the subtree φ is replaced by a single node labeled ε. φ' is the expression resulting from φ just by deleting the licensee feature $\mathtt{-x}$ that φ's head-label starts with.

Note that, by (me.1) and (me.2), a simple tree (head) selects another tree as its complement to the right, whereas a complex tree selects another tree as a specifier to the left.

[9] For each (partial) mapping f from a set M_1 into a set M_2 we take $\mathrm{Dom}(f)$ to denote the *domain of* f, the subset of M_1 for which f is defined.

Example 4.3. Let G_{ww} be the MG with $\mathit{Sem} = \emptyset$, $\mathit{Phon} = \{1,2\}$, $\mathit{base} = \{\mathtt{c}, \mathtt{a}_1, \mathtt{a}_2, \mathtt{b}, \mathtt{c}_1, \mathtt{c}_2, \mathtt{d}\}$, $\mathit{select} = \{{}^{=}\mathtt{a}_1, {}^{=}\mathtt{a}_2, {}^{=}\mathtt{b}, {}^{=}\mathtt{c}_1, {}^{=}\mathtt{c}_2, {}^{=}\mathtt{d}\}$, $\mathit{Licensors} = \{\mathtt{+L}_1, \mathtt{+L}_2\}$, $\mathit{Licensees} = \{\mathtt{-l}_1, \mathtt{-l}_2\}$, while *Lex* consists of the following 10 simple expressions, where $i \in \{1,2\}$,[10]

$$\begin{array}{lll} \alpha_i = \mathtt{a}_i\mathtt{-l}_1 i & \gamma_i = {}^{=}\mathtt{b+L}_1\mathtt{c}_i\mathtt{-l}_1 i & \zeta_1 = {}^{=}\mathtt{b+L}_1\mathtt{d} \\ \beta_i = {}^{=}\mathtt{a}_i\mathtt{b-l}_2 i & \delta_i = {}^{=}\mathtt{c}_i\mathtt{+L}_2\mathtt{b-l}_2 i & \zeta_2 = {}^{=}\mathtt{d+L}_2\mathtt{c} \end{array}$$

Then e.g., for $i, j \in \{1,2\}$, $\mathit{move}(\mathit{merge}(\delta_j, \mathit{move}(\mathit{merge}(\gamma_j, \mathit{merge}(\beta_i, \alpha_i))))) \in \mathit{Exp}(\mathit{Feat})$.

Let $G = (\mathit{Non\text{-}Syn}, \mathit{Syn}, \mathit{Lex}, \mathcal{F})$ be an MG. Then $CL(G) = \bigcup_{k \in \mathbb{N}} CL^k(G)$ is the *closure of Lex (under the functions in $\mathcal{F}$)*. For $k \in \mathbb{N}$ the sets $CL^k(G) \subseteq \mathit{Exp}(\mathit{Feat})$ are inductively defined by

$$\begin{aligned} CL^0(G) &= \mathit{Lex} \\ CL^{k+1}(G) &= CL^k(G) \\ &\quad \cup \{\mathit{merge}(\upsilon, \varphi) \mid (\upsilon, \varphi) \in \mathrm{Dom}(\mathit{merge}) \cap CL^k(G) \times CL^k(G)\} \\ &\quad \cup \{\mathit{move}(\upsilon) \mid \upsilon \in \mathrm{Dom}(\mathit{move}) \cap CL^k(G)\} \end{aligned}$$

Every $\tau \in CL(G)$ is called an *expression in G*. Such a τ is *complete (in G)* if its head-label is in $\{\mathtt{c}\}\mathit{Phon}^*\mathit{Sem}^*$ and each other of its leaf-labels is in $\mathit{Phon}^*\mathit{Sem}^*$. Hence, a complete expression has category $\mathtt{c}$, and this instance of $\mathtt{c}$ is the only instance of a syntactic feature within all leaf-labels.

The *(phonetic) yield* $Y(\tau)$ of an expression $\tau \in \mathit{Exp}(\mathit{Feat})$ is the string created by concatenating τ's leaf-labels "from left to right" and stripping off all non-phonetic features. $L(G) = \{Y(\tau) \mid \tau \in CL(G) \text{ with } \tau \text{ is complete}\}$ is the *(string) language (derivable by G)* and is called a *minimalist language.*

Example 4.3 (continued) For $i \in \{1,2\}$ and $u \in \{1,2\}^+$ the expressions belonging to $CL(G_{ww})$ can recursively be defined by

$$\text{(2a)} \quad \omega_i = \mathit{merge}(\beta_i, \alpha_i)$$

$$\begin{array}{llll} \text{(2b)} & \varphi_{iu} = \mathit{merge}(\gamma_i, \omega_u) & \text{(2b')} & \eta_u = \mathit{merge}(\zeta_1, \omega_u) \\ \text{(2c)} & \chi_{iu} = \mathit{move}(\varphi_{iu}) & \text{(2c')} & \vartheta_u = \mathit{move}(\eta_u) \\ \text{(2d)} & \psi_{iu} = \mathit{merge}(\delta_i, \chi_{iu}) & \text{(2d')} & \kappa_u = \mathit{merge}(\zeta_2, \vartheta_u) \\ \text{(2e)} & \omega_{iu} = \mathit{move}(\psi_{iu}) & \text{(2e')} & \xi_u = \mathit{move}(\kappa_u) \end{array}$$

In more detail, we have

$$\text{(3a)} \quad CL^1(G_{ww}) \setminus CL^0(G_{ww}) = \{\omega_1, \omega_2\}$$

while for $k \in \mathbb{N}$ we have

$$\begin{array}{ll} \text{(3b)} & CL^{4k+2}(G_{ww}) \setminus CL^{4k+1}(G_{ww}) = \{\varphi_{iu}, \eta_u \mid i \in \{1,2\}, u \in \{1,2\}^+ \text{ with } |u| = k\} \\ \text{(3c)} & CL^{4k+3}(G_{ww}) \setminus CL^{4k+2}(G_{ww}) = \{\chi_{iu}, \vartheta_u \mid i \in \{1,2\}, u \in \{1,2\}^+ \text{ with } |u| = k\} \\ \text{(3d)} & CL^{4k+4}(G_{ww}) \setminus CL^{4k+3}(G_{ww}) = \{\psi_{iu}, \kappa_u \mid i \in \{1,2\}, u \in \{1,2\}^+ \text{ with } |u| = k\} \\ \text{(3e)} & CL^{4k+5}(G_{ww}) \setminus CL^{4k+4}(G_{ww}) = \{\omega_{iu}, \xi_u \mid i \in \{1,2\}, u \in \{1,2\}^+ \text{ with } |u| = k\} \end{array}$$

The set of complete expressions in G_{ww} is $\{\xi_u \mid u \in \{1,2\}^+\}$. Each such ξ_u has the phonetic yield $Y(\xi_u) = uu$, i.e., the string language derivable by G_{ww} is $\{uu \mid u \in \{1,2\}^+\}$.

[10]Since all lexical entries are heads, we simply represent them by their respective (unique) labels.

Definition 4.4. For each MG $G = \langle \mathit{Non\text{-}Syn}, \mathit{Syn}, \mathit{Lex}, \mathcal{F} \rangle$, an expression $\tau \in CL(G)$ is called *relevant (in G)* if it has property (R).

(R) For any `-x` $\in$ *Licensees* there is at most one maximal projection $\tau_{\texttt{-x}}$ in τ that has feature `-x`.[11]

For any given MG G, we take $\mathit{Rel}(G)$ to be the set of all relevant expressions $\tau \in CL(G)$. Since each complete $\tau \in CL(G)$ has property (R), we have $L(G) = \{Y(\tau) \,|\, \tau \in \mathit{Rel}(G),\ \tau \text{ is complete}\}$.

Remark 4.5. For G_{ww} as in Example 4.3 we have $\mathit{Rel}(G_{ww}) = CL(G_{ww})$.

5 Translating MGs to MCFGs

In Michaelis (1999) an algorithm is given how to transform an arbitrary MG G into a weakly equivalent MCFG. The core idea is that for $\mathit{Rel}(G)$, i.e., the set of trees appearing as intermediate steps in converging derivations of G, one can define a finite partition. The equivalence classes of this partition are formed by sets of trees where the features triggering movement appear in identical structural positions. Each nonterminal in a corresponding MCFG represents such an equivalence class, i.e., an infinite set of trees. In this paper we will concentrate on the example from above, adopting the methods from Michaelis (1999).

Definition 5.1. A *multiple context-free grammar* (MCFG) is defined as a five-tuple $\mathcal{G} = \langle V_N, V_T, V_F, P, S \rangle$ with V_N, V_T, V_F and P being a finite set of nonterminals, terminals, linear basic morphisms and productions, respectively. $S \in V_N$ is the start symbol. There is a function d from V_N to $\mathbb{N}$ such that $d(S) = 1$. Each $p \in P$ has the form $A \longrightarrow f(A_0, \ldots, A_{n-1})$ for $A, A_0, \ldots, A_{n-1} \in V_N$ and $f \in F$ a function from $(V_T^*)^k$ to $(V_T^*)^{d(A)}$ with arity $k = \sum_{i=0}^{n-1} d(A_i)$ (cf. Seki et al. 1991). Recall that basic morphisms are those which use only variables, constants, concatenation, composition and tupling.

The derivation-relation $\Rightarrow_{\mathcal{G}}$ for $\mathcal{G}$ is defined as follows: If $A \longrightarrow f() \in P$ then $A \Rightarrow_{\mathcal{G}} f()$, where $f() \in (V_T^*)^{d(A)}$ (i.e., f is some constant tuple of terminal strings). If $A \longrightarrow f(A_0, \ldots, A_{n-1}) \in P$ and $A_i \Rightarrow_{\mathcal{G}} t_i$ for some $t_i \in (V_T^*)^{d(A_i)}$ then $A_i \Rightarrow_{\mathcal{G}} f(t_0, \ldots, t_{n-1})$. The language generated by $\mathcal{G}$ is $L(\mathcal{G}) = \{t \in V_T^* \,|\, S \Rightarrow_{\mathcal{G}} t\}$.

Example 4.3 (continued) Let G_{ww} be the MG as given in Example 4.3. In order to define a weakly equivalent MCFG $\mathcal{G}_{ww} = \langle V_N, V_T, V_F, P, S \rangle$, we first let $\mathit{Phon} = \{1, 2\}$ be the set of terminals V_T and proceed by constructing the set of nonterminals V_N.

Each nonterminal will either be the start symbol S or a 3-tuple from $\mathit{Syn}^* \times \mathit{Syn}^* \times \mathit{Syn}^*$. The core idea is the following: Consider $\tau \in CL(G_{ww}) \setminus CL^0(G_{ww})$. For $1 \leq i \leq 2$ take, if it exists, τ_i to be a subtree of τ that is a maximal projection and has licensee `-l`$_i$.[12] Otherwise, take τ_i to be a single node labeled ε. Set $\tau_0 = \tau$. Let μ_i be the prefix of τ_i's head-label consisting of just the syntactic features. Then, $\langle \mu_0, \mu_1, \mu_2 \rangle \in V_N$. The productions (and functions) of the MCFG $\mathcal{G}_{ww}$ will be defined in such a way that for each $\langle p_0, p_1, p_2 \rangle \in \mathit{Phon}^* \times \mathit{Phon}^* \times \mathit{Phon}^*$, $\langle \mu_0, \mu_1, \mu_2 \rangle \Rightarrow_{\mathcal{G}_{ww}} \langle p_0, p_1, p_2 \rangle$ iff (wc) holds.

(wc) For $0 \leq i \leq 2$, p_i is the phonetic yield of τ_i except for each substring that is the phonetic yield of some τ_j, $1 \leq j \leq 3$ and $i \neq j$, being a proper subtree of τ_i.

Although $CL(G_{ww}) \setminus CL^0(G_{ww})$ is an infinite set, V_N is finite. This is due to two reasons emerging from the definition of *merge* and *move*. First, in each $\tau \in CL(G_{ww})$ at most 3 different leaves are present which have syntactic features appearing in their labels.[13] Second, for each MG G the set of all leaf-labels of all $\tau \in CL(G)$ constitutes a finite set, because a leaf-label is always the suffix

[11] In fact, this kind of structure is characteristic of each $\tau \in CL(G)$ involved in creating a complete expression in G. Recall that *move*(τ) is defined for $\tau \in CL(G)$ only in case that there is exactly one maximal subtree of τ that has a particular licensee feature allowing the subtree's "movement into specifier position."

[12] Recall that $\mathit{Rel}(G_{ww}) = CL(G_{ww})$. Therefore, such a τ_i is unique. In order to ensure this in general, we would have to reduce the closure of an MG G to what is defined as the *relevant closure* $RCL(G)$ of G in Michaelis (1999).

[13] Recall again Remark 4.5.

of some leaf-label of some lexical entry, i.e., a suffix of a finite string, and the lexicon of an MG is a finite subset of *Exp*(*Non-Syn* $\cup$ *Syn*). In fact, we have exactly 10 nonterminals different from the start symbol S, each of which corresponds to an infinite set of expressions from $CL(G_{ww})$, namely[14]

(4a) $U = \langle \texttt{b-l}_2, \texttt{-l}_1, - \rangle$ to $\{\omega_i \,|\, i \in \{1,2\}\}$

(4b) $V_i = \langle \texttt{+L}_1\texttt{c}_i\texttt{-l}_1, \texttt{-l}_1, \texttt{-l}_2 \rangle$ to $\{\varphi_{iu} \,|\, u \in \{1,2\}^+\}$, where $i \in \{1,2\}$

(4c) $U_i = \langle \texttt{c}_i\texttt{-l}_1, -, \texttt{-l}_2 \rangle$ to $\{\chi_{iu} \,|\, u \in \{1,2\}^+\}$, where $i \in \{1,2\}$

(4d) $V = \langle \texttt{+L}_2\texttt{b-l}_2, \texttt{-l}_1, \texttt{-l}_2 \rangle$ to $\{\psi_{iu} \,|\, u \in \{1,2\}^+, i \in \{1,2\}\}$

(4e) $U = \langle \texttt{b-l}_2, \texttt{-l}_1, - \rangle$ to $\{\omega_{iu} \,|\, u \in \{1,2\}^+, i \in \{1,2\}\}$

(4b') $W_1 = \langle \texttt{+L}_1\texttt{d}, \texttt{-l}_1, \texttt{-l}_2 \rangle$ to $\{\eta_u \,|\, u \in \{1,2\}^+\}$

(4c') $X_1 = \langle \texttt{d}, -, \texttt{-l}_2 \rangle$ to $\{\vartheta_u \,|\, u \in \{1,2\}^+\}$

(4d') $W_2 = \langle \texttt{+L}_2\texttt{c}, -, \texttt{-l}_2 \rangle$ to $\{\kappa_u \,|\, u \in \{1,2\}^+\}$

(4e') $C = \langle \texttt{c}, -, - \rangle$ to $\{\xi_u \,|\, u \in \{1,2\}^+\}$

Thus, the nonterminals different from S introduce a finite partition of $CL(G_{ww}) \setminus CL^0(G_{ww})$. We now define P and F, the sets of productions and functions in $\mathcal{G}_{ww}$, respectively. Each $p \in P$ in a certain way simulates an application of *merge* and *move* in G_{ww}, but operates w.r.t. the equivalence classes of the induced partition, rather than on single expressions. As indicated in the introduction, we present in this paper only the significantly simpler case of using only MCFG-rules with one nonterminal on the RHS. Although *merge* in principle is a binary operation, i.e., would require two nonterminals, we can in this special case partially evaluate the merge operation. This becomes possible since we only have "simple merges," i.e, we always merge a head with some complex tree, as opposed to merging two complex trees.

Let $i \in \{1,2\}$. P consists of 2 terminating rules,

(5a) $U \rightarrow merge_{\langle \beta_i, \alpha_i \rangle}()$

and 12 nonterminating rules,

(5b) $V_i \rightarrow merge_{\langle \gamma_i, \omega_- \rangle}(U)$

(5c) $U_i \rightarrow move_{\langle \varphi_- \rangle}(V_i)$

(5d) $V \rightarrow merge_{\langle \delta_i, \chi_- \rangle}(U_i)$

(5e) $U \rightarrow move_{\langle \psi_- \rangle}(V)$

(5b') $W_1 \rightarrow merge_{\langle \zeta_1, \omega_- \rangle}(U)$

(5c') $X_1 \rightarrow move_{\langle \eta_- \rangle}(W_1)$

(5d') $W_2 \rightarrow merge_{\langle \zeta_2, \vartheta_- \rangle}(X_1)$

(5e') $C \rightarrow move_{\langle \kappa_- \rangle}(W_2)$

(5f') $S \rightarrow \pi_1^3(C)$ (the initial rule)

The 9 basic functions in F are defined as follows:

[14] Here, we write $-$ instead of ε.

(6a) $merge_{\langle\beta_i,\alpha_i\rangle} : (\{1,2\}^*)^0 \to (\{1,2\}^*)^3$ with $\langle\,\rangle \mapsto \langle i,i,\epsilon\rangle$ for $i \in \{1,2\}$

(6b) $merge_{\langle\gamma_i,\omega\_\rangle} : (\{1,2\}^*)^3 \to (\{1,2\}^*)^3$ with $\langle x_1,x_2,x_3\rangle \mapsto \langle i,x_2,x_1\rangle$ for $i \in \{1,2\}$

(6c) $move_{\varphi\_} : (\{1,2\}^*)^3 \to (\{1,2\}^*)^3$ with $\langle x_1,x_2,x_3\rangle \mapsto \langle x_2x_1,\epsilon,x_3\rangle$

(6d) $merge_{\langle\delta_i,\chi\_\rangle} : (\{1,2\}^*)^3 \to (\{1,2\}^*)^3$ with $\langle x_1,x_2,x_3\rangle \mapsto \langle i,x_1,x_3\rangle$ for $i \in \{1,2\}$

(6e) $move_{\psi\_} : (\{1,2\}^*)^3 \to (\{1,2\}^*)^3$ with $\langle x_1,x_2,x_3\rangle \mapsto \langle x_3x_1,x_2,\epsilon\rangle$

(6b') $merge_{\langle\zeta_1,\omega\_\rangle} : (\{1,2\}^*)^3 \to (\{1,2\}^*)^3$ with $\langle x_1,x_2,x_3\rangle \mapsto \langle \epsilon,x_2,x_1\rangle$

(6c') $move_{\eta\_} : (\{1,2\}^*)^3 \to (\{1,2\}^*)^3$ with $\langle x_1,x_2,x_3\rangle \mapsto \langle x_2x_1,\epsilon,x_3\rangle$

(6d') $merge_{\langle\zeta_2,\vartheta\_\rangle} : (\{1,2\}^*)^3 \to (\{1,2\}^*)^3$ with $\langle x_1,x_2,x_3\rangle \mapsto \langle x_1,\epsilon,x_3\rangle$

(6e') $move_{\kappa\_} : (\{1,2\}^*)^3 \to (\{1,2\}^*)^3$ with $\langle x_1,x_2,x_3\rangle \mapsto \langle x_3x_1,x_2,\epsilon\rangle$

(6f') $\pi^3_1 : (\{1,2\}^*)^3 \to \{1,2\}^*$ with $\langle x_1,x_2,x_3\rangle \mapsto x_1$

Note that $move_{\varphi\_} = move_{\eta\_}$ and $move_{\psi\_} = move_{\kappa\_}$. The 3-tuples of terminal strings derivable from the nonterminals different from S are the following, where $i \in \{1,2\}$ and $u \in \{1,2\}^+$,

(7a) $U \Rightarrow_{\mathcal{G}_{ww}} \langle i,i,\epsilon\rangle$

(7b) $V_i \Rightarrow_{\mathcal{G}_{ww}} \langle i,u,u\rangle$	(7b') $W_1 \Rightarrow_{\mathcal{G}_{ww}} \langle \epsilon,u,u\rangle$
(7c) $U_i \Rightarrow_{\mathcal{G}_{ww}} \langle ui,\epsilon,u\rangle$	(7c') $X_1 \Rightarrow_{\mathcal{G}_{ww}} \langle u,\epsilon,u\rangle$
(7d) $V \Rightarrow_{\mathcal{G}_{ww}} \langle i,ui,u\rangle$	(7d') $W_2 \Rightarrow_{\mathcal{G}_{ww}} \langle u,\epsilon,u\rangle$
(7e) $U \Rightarrow_{\mathcal{G}_{ww}} \langle iu,iu,\epsilon\rangle$	(7e') $C \Rightarrow_{\mathcal{G}_{ww}} \langle uu,\epsilon,\epsilon\rangle$

Thus, we finally have: (7f') $S \Rightarrow_{\mathcal{G}_{ww}} uu$ iff $u \in \{1,2\}^+$.

MCFGs can be interpreted as a restricted type of AGs according to their general definition given in Sec. 2.2. Intuitively, an AG which corresponds to a given MCFG needs just a single synthesized attribute for each nonterminal. The attribute evaluation proceeds via a depth-first left-to-right tree traversal. The semantic domains are the sets of words and tuples of words over the terminal alphabet of the MCFG. The semantic operations are "simple" functions on these domains, namely those functions belonging to the MCFG.

Let $\mathcal{G} = \langle V_N, V_T, V_F, P, S\rangle$ be an MCFG. It is straightforward to formally convert $\mathcal{G}$ into an AG $\Gamma = \langle \Sigma, \mathsf{A_{syn}}, \mathsf{A_{in}}, \Omega, W, R, \alpha_{mean}\rangle$ with empty set of inherited attributes: First, for $d(\mathcal{G}) = \max\{d(A) \,|\, A \in V_N\}$ we define $\mathsf{A_{syn}} = \{1,\dots,d(\mathcal{G})\}$ and $\Omega = \{(V_T^*)^i \,|\, 1 \leq i \leq d(\mathcal{G})\}$, and we set $\alpha_{mean} = 1$.

The function W from $\mathsf{A_{syn}}$ to Ω maps each i to $(V_T^*)^i$. Intuitively, each nonterminal $A \in V_N$ will be "connected" with exactly one synthesized attribute, namely $d(A)$.

Σ is a many-sorted signature over V_N defined in parallel to R as follows: For each production $p = A_0 \longrightarrow f(A_1,\dots,A_k) \in P$ with $A_0,\dots,A_n \in V_N$ and corresponding $f \in V_F$ we let $\sigma_p \in \Sigma_{w,s}$, where $w = A_1 \cdots A_n$ and $s = A_0$, and we let $r_p \in R(\sigma_p)$, where r_p is of the form

$$\langle\alpha_0, 0\rangle = f(\langle\alpha_1, 1\rangle, \cdots, \langle\alpha_k, k\rangle)$$

with $\alpha_i = d(A_i)$ for $0 \leq i \leq k$.

Since the corresponding AG is only synthesized, for each $t \in T(\Sigma)$ the dependency graph is non-circular. Thus, each $t \in T(\Sigma)$ has a unique decoration. The string language generated by the MCFG $\mathcal{G}$ is the set of root-values each of which belonging to some tree in $t \in T(\Sigma)$ of sort $S \in V_N$ such that $t = \sigma_p(t_1, \ldots, t_{|w|})$ for some $t_1, \ldots, t_{|w|} \in T(\Sigma)$, i.e., σ_p arises from some $p \in P$ whose LHS is the start symbol S.

Example 4.3 (continued) As far as our example MCFG $\mathcal{G}_{ww}$ is concerned, the corresponding transformation results in the AG $\Gamma_{ww} = \langle \Sigma, \mathsf{A}_{\mathtt{syn}}, \mathsf{A}_{\mathtt{in}}, \Omega, W, R, \alpha_{mean} \rangle$ with in particular $\mathsf{A}_{\mathtt{syn}} = \{1,2,3\}$ and $\Omega = \{(\{1,2\}^*)^i \mid 1 \leq i \leq 3\}$. For $i \in \{1,2\}$, Σ and R are defined by

(8a)	$\sigma_a^{(i)} \in \Sigma_{\varepsilon,U}$	and $r_a^{(i)} \in R(\sigma_a^{(i)})$ of the form	$\langle 3,0\rangle = merge_{\langle \beta_i, \alpha_i\rangle}()$
(8b)	$\sigma_b^{(i)} \in \Sigma_{V_i,U}$	and $r_b^{(i)} \in R(\sigma_b^{(i)})$ of the form	$\langle 3,0\rangle = merge_{\langle \gamma_i, \omega_{\text{-}}\rangle}(\langle 3,1\rangle)$
(8c)	$\sigma_c^{(i)} \in \Sigma_{U_i,V_i}$	and $r_c^{(i)} \in R(\sigma_c^{(i)})$ of the form	$\langle 3,0\rangle = move_{\langle \varphi_{\text{-}}\rangle}(\langle 3,1\rangle)$
(8d)	$\sigma_d^{(i)} \in \Sigma_{V,U_i}$	and $r_d^{(i)} \in R(\sigma_d^{(i)})$ of the form	$\langle 3,0\rangle = merge_{\langle \delta_i, \chi_{\text{-}}\rangle}(\langle 3,1\rangle)$
(8e)	$\sigma_e \in \Sigma_{U,V}$	and $r_e \in R(\sigma_e)$ of the form	$\langle 3,0\rangle = move_{\langle \psi_{\text{-}}\rangle}(\langle 3,1\rangle)$
(8b')	$\sigma_{b'}^{(i)} \in \Sigma_{W_1,U}$	and $r_{b'}^{(i)} \in R(\sigma_{b'}^{(i)})$ of the form	$\langle 3,0\rangle = merge_{\langle \zeta_1, \omega_{\text{-}}\rangle}(\langle 3,1\rangle)$
(8c')	$\sigma_{c'} \in \Sigma_{X_1,W_1}$	and $r_{c'} \in R(\sigma_{c'})$ of the form	$\langle 3,0\rangle = move_{\langle \eta_{\text{-}}\rangle}(\langle 3,1\rangle)$
(8d')	$\sigma_{d'} \in \Sigma_{W_2,X_1}$	and $r_{d'} \in R(\sigma_{d'})$ of the form	$\langle 3,0\rangle = merge_{\langle \zeta_2, \vartheta_{\text{-}}\rangle}(\langle 3,1\rangle)$
(8e')	$\sigma_{e'} \in \Sigma_{C,W_2}$	and $r_{e'} \in R(\sigma_{e'})$ of the form	$\langle 3,0\rangle = move_{\langle \kappa_{\text{-}}\rangle}(\langle 3,1\rangle)$
(8f')	$\sigma_{f'} \in \Sigma_{S,C}$	and $r_{f'} \in R(\sigma_{f'})$ of the form	$\langle 1,0\rangle = \pi_1^3(\langle 3,1\rangle)$

where the corresponding functions occuring within the semantic rules of Γ_{ww} are defined as in (6a)-(6e) and (6b')-(6f').

As was mentioned in the introduction, AGs constitute a much more powerful device than MCFGs. In particular, Duske et al. (1977) contains the result that the inside-out context-free tree languages are contained in the output of attributed tree transducers. The latter are AGs where all the attribute values are trees and where the operations in the semantic rules are restricted to substitution of trees. Context-free tree grammars may be too strong for an adequate characterization of the complexity of natural languages. But, certain phenomena like the widely discussed cases of *Suffixaufnahme* (multiple case-stacking) seem to indicate that natural languages are not semilinear (cf. Michaelis and Kracht (1997)), and it is well known that the Parikh images of the family of mildly context-sensitive languages listed in the introduction are semilinear. Inside-out context-free tree languages in their turn are sufficiently powerful to handle the known cases of *Suffixaufnahme* as has been shown in Mönnich (1997).

So, there are basically two reasons why one might want to use AGs instead of MCFGs. But since we think that the presentation is more perspicuous using MCFGs with a fine-grained analysis via Lawvere terms and we do not need the additional power of the AGs for our example, we continue to use the more accessible formulation via MCFGs.

6 Translating MCFGs to RTGs

In this section we will show how to translate the rules of a given MCFG into an RTG. We start by giving a formal definition of regular tree grammars.

Definition 6.1. A *regular tree grammar* (RTG) is a 4-tuple $\mathcal{G} = \langle \Sigma, \mathsf{F}_0, S, \mathsf{P} \rangle$, where for some set of sorts $\mathcal{S}$, $\Sigma = \langle \Sigma_{w,s} \,|\, w \in \mathcal{S}^*, s \in \mathcal{S} \rangle$ is a many-sorted signature of *inoperatives* and $\mathsf{F}_0 = \langle \mathsf{F}_{\varepsilon,s} \,|\, s \in \mathcal{S} \rangle$ a (reduced) many-sorted signature of *operatives* of rank 0. Moreover, $\bigcup_{(w,s)\in\mathcal{S}^*\times\mathcal{S}} \Sigma_{w,s}$ and $\bigcup_{s\in\mathcal{S}} \mathcal{F}_{\varepsilon,s}$ are finite. $S \in \mathsf{F}_0$ is the starting symbol and P is a finite set of productions. Each $p \in \mathsf{P}$ has the form $F \longrightarrow t$, where $F \in \mathsf{F}_{\varepsilon,s}$ for some $s \in \mathcal{S}$ and $t \in T(\Sigma \cup \mathsf{F}_0)$, i.e., a term (tree) over $\Sigma \cup \mathsf{F}_0$, such that t is of sort s.

Let $t', t'' \in T(\Sigma \cup \mathsf{F}_0)$ and $p = F \longrightarrow t \in P$. *$t'$ directly derives t'' (by the application of p)*, also denoted by $t' \Rightarrow_{\mathcal{G}} t''$, if t' has a leaf-node F and t'' results from t' by substituting this node F by t. Let $\Rightarrow^*_{\mathcal{G}}$ be the reflexive and transitive closure of $\Rightarrow_{\mathcal{G}}$. The tree-language generated by $\mathcal{G}$ is the set $L_T(\mathcal{G}) = \{t \in T(\Sigma) \,|\, S \Rightarrow^*_{\mathcal{G}} t\}$.

The yield $Y(t)$ of a $t \in T(\Sigma \cup \mathsf{F}_0)$ is the string resulting from concatenating the leaf-nodes of t "from left to right." Thus, $Y(t) \in (\bigcup_{s\in\mathcal{S}} \Sigma_{\varepsilon,s} \cup \bigcup_{s\in\mathcal{S}} \mathcal{F}_{\varepsilon,s})^*$. The string-language generated by $\mathcal{G}$ is the set $L_Y = \{Y(t) \,|\, t \in L_T(\mathcal{G})\} \subseteq (\bigcup_{s\in\mathcal{S}} \Sigma_{\varepsilon,s})^*$.

Since RTG-rules always just substitute some tree for a leaf-node, it is easy to see that they generate recognizable sets of trees, i.e., context-free string languages (Mezei and Wright, 1967).

Now we turn to the actual translation. Each rule of a given MCFG is recursively transformed into a RTG-rule by coding the implicit operations of projection, tupling and composition as nonterminals or terminals. This becomes possible simply by viewing the terms appearing in the rules of the MCFG as elements of a free $\mathbb{N}\times\mathbb{N}$-sorted Lawvere algebra. The resulting RTG then "operates on" this Lawvere algebra. Recall that we are using a simple example MCFG with maximally one nonterminal on the RHSs and therefore do not have to build tuples from tuples.

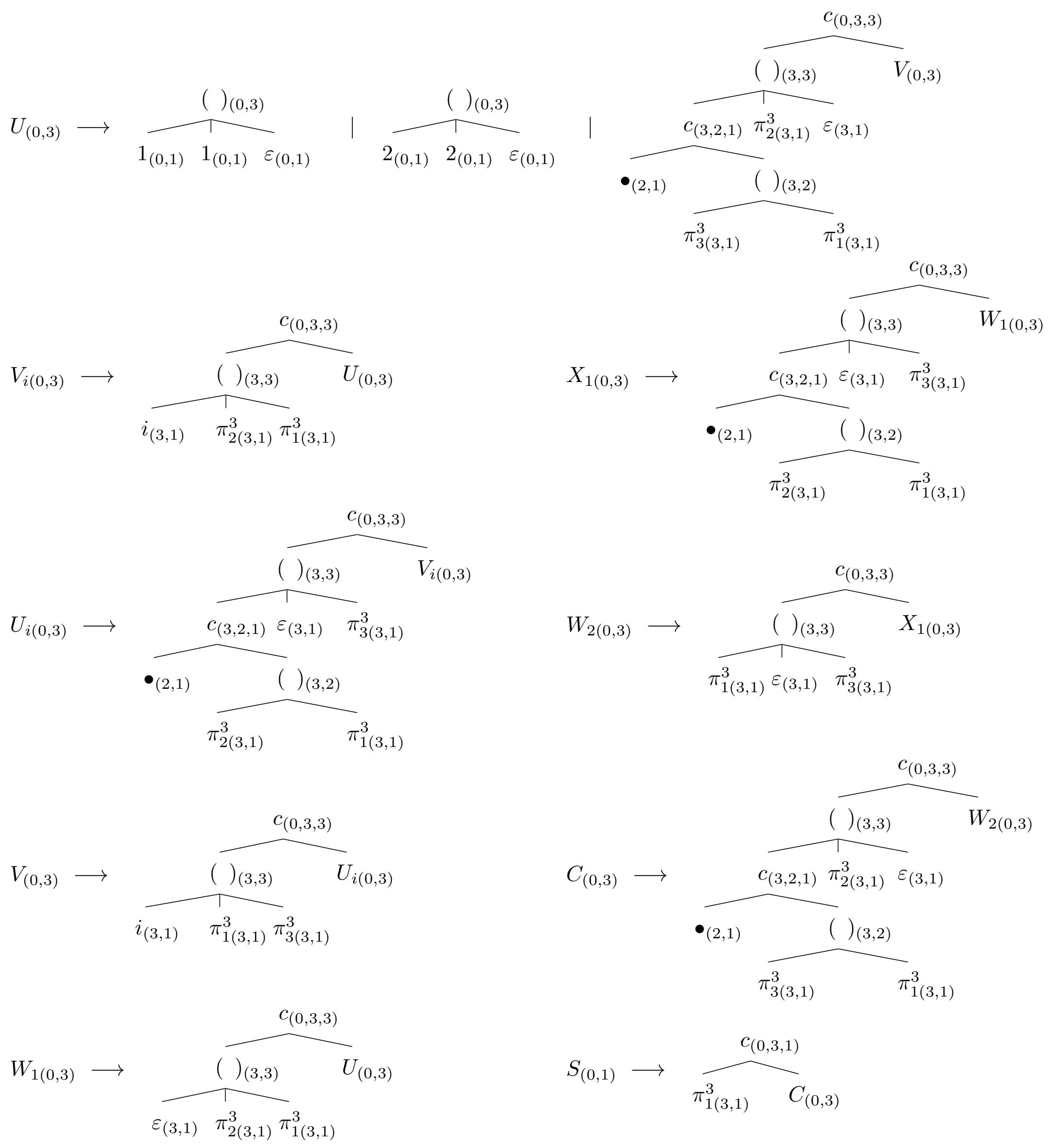
Example 4.3 (continued) As an example, we show how to translate the MCFG $\mathcal{G}_{ww}$ given in Example 4.3 into the weakly equivalent RTG $\mathcal{G}'_{ww}$. Intuitively, we have to make the implicit operations which are "hidden" in the standard presentation of the MCFG-rules explicit. Simply using a tuple, e.g., the pair $\langle a, b \rangle$, means that we need an explicit tupling operator $(\ )$ to combine a and b. In the same spirit, using $x_0 x_1$ means that the values of the two variables are concatenated with an implicit concatenation operator. And finally, applying a function to some arguments is a composition $\circ$ of the function with its arguments. Note that thereby the function becomes a constant, i.e., we reify the function. In this sense, a term such as $f(a, b)$ becomes more complex: $(f \circ ((\ )(x_1, x_2))) \circ (\ )(a, b)$. Now we have to translate these single-sorted terms into the corresponding many-sorted Lawvere algebra.

For $1 \leq i \leq 3$, let π^3_i denote the i-th projection which maps a 3-tuple of strings from V_T^* to its i-th component, i.e., a 1-tuple. Therefore the corresponding Lawvere arity of π^3_1, π^3_2 and π^3_3 is $(3, 1)$. Let $\bullet$ denote the usual binary operation of concatenation defined for strings from V_T^*, i.e., $\bullet$ maps a 2-tuple to a 1-tuple. Thus $\bullet$ is of Lawvere arity $(2, 1)$. Similarly, the corresponding (Lawvere) arity of $S, 1, 2$ and ε is $(0, 1)$ and of $U, V, C, W_1, W_2, X_1, U_i, V_i$ $(0, 3)$.

We use the composition symbols $c_{(i,j,k)}$ introduced in Sec. 2.1 for the composition $\circ$.

Let $\mathcal{G}_{ww} = \langle V_N, V_T, V_F, P, S \rangle$ be the MCFG successively constructed in the preceeding section. Applying the translation $\mathsf{T} : P \longrightarrow \mathsf{F}_0 \times T(\Sigma \cup \mathsf{F}_0)$ given below to the rules of the MCFG $\mathcal{G}_{ww}$ results in the RTG $\mathcal{G}'_{ww} = \langle \Sigma, \mathsf{F}_0, S_{(0,1)}, \mathsf{P} \rangle$ with inoperatives $\Sigma = \langle \Sigma_{w,s} \,|\, w \in (\mathbb{N}\times\mathbb{N})^*, s \in \mathbb{N}\times\mathbb{N} \rangle$, operatives F_0 of rank 0, and productions P which (in tree notation) look as given in Fig. 4, where $i \in \{1, 2\}$.[15] We have $\Sigma_{\varepsilon,(3,0)} = \{(\ )_{(3,0)}\}$, $\Sigma_{\varepsilon,(2,1)} = \{\bullet_{(2,1)}\}$, $\Sigma_{\varepsilon,(0,1)} = \{1_{(0,1)}, 2_{(0,1)}, \varepsilon_{(0,1)}\}$,

[15]Note that we simplified the presentation of the rules at two points: $i_{(3,1)}$ and $\varepsilon_{(3,1)}$ represent trees resulting from attributing to the "real" elements of $\Sigma_{\varepsilon,(0,1)}$ the simplified Lawvere arity $(3, 1)$, i.e., $i_{(3,1)}$ stands for $c_{(3,0,1)}(i_{(0,1)}, (\ )_{(3,0)})$ and $\varepsilon_{(3,1)}$ for $c_{(3,0,1)}(\varepsilon_{(0,1)}, (\ )_{(3,0)})$.



$U_{(0,3)} \longrightarrow$
$(\ )_{(0,3)}$ $1_{(0,1)}$ $1_{(0,1)}$ $\varepsilon_{(0,1)}$
$|$
$(\ )_{(0,3)}$ $2_{(0,1)}$ $2_{(0,1)}$ $\varepsilon_{(0,1)}$
$|$
$c_{(0,3,3)}$ $(\ )_{(3,3)}$ $V_{(0,3)}$ $c_{(3,2,1)}$ $\pi^3_{2(3,1)}$ $\varepsilon_{(3,1)}$ $\bullet_{(2,1)}$ $(\ )_{(3,2)}$ $\pi^3_{3(3,1)}$ $\pi^3_{1(3,1)}$
$V_{i(0,3)} \longrightarrow$
$c_{(0,3,3)}$ $(\ )_{(3,3)}$ $U_{(0,3)}$ $i_{(3,1)}$ $\pi^3_{2(3,1)}$ $\pi^3_{1(3,1)}$
$X_{1(0,3)} \longrightarrow$
$c_{(0,3,3)}$ $(\ )_{(3,3)}$ $W_{1(0,3)}$ $c_{(3,2,1)}$ $\varepsilon_{(3,1)}$ $\pi^3_{3(3,1)}$ $\bullet_{(2,1)}$ $(\ )_{(3,2)}$ $\pi^3_{2(3,1)}$ $\pi^3_{1(3,1)}$
$U_{i(0,3)} \longrightarrow$
$c_{(0,3,3)}$ $(\ )_{(3,3)}$ $V_{i(0,3)}$ $c_{(3,2,1)}$ $\varepsilon_{(3,1)}$ $\pi^3_{3(3,1)}$ $\bullet_{(2,1)}$ $(\ )_{(3,2)}$ $\pi^3_{2(3,1)}$ $\pi^3_{1(3,1)}$
$W_{2(0,3)} \longrightarrow$
$c_{(0,3,3)}$ $(\ )_{(3,3)}$ $X_{1(0,3)}$ $\pi^3_{1(3,1)}$ $\varepsilon_{(3,1)}$ $\pi^3_{3(3,1)}$
$V_{(0,3)} \longrightarrow$
$c_{(0,3,3)}$ $(\ )_{(3,3)}$ $U_{i(0,3)}$ $i_{(3,1)}$ $\pi^3_{1(3,1)}$ $\pi^3_{3(3,1)}$
$C_{(0,3)} \longrightarrow$
$c_{(0,3,3)}$ $(\ )_{(3,3)}$ $W_{2(0,3)}$ $c_{(3,2,1)}$ $\pi^3_{2(3,1)}$ $\varepsilon_{(3,1)}$ $\bullet_{(2,1)}$ $(\ )_{(3,2)}$ $\pi^3_{3(3,1)}$ $\pi^3_{1(3,1)}$
$W_{1(0,3)} \longrightarrow$
$c_{(0,3,3)}$ $(\ )_{(3,3)}$ $U_{(0,3)}$ $\varepsilon_{(3,1)}$ $\pi^3_{2(3,1)}$ $\pi^3_{1(3,1)}$
$S_{(0,1)} \longrightarrow$
$c_{(0,3,1)}$ $\pi^3_{1(3,1)}$ $C_{(0,3)}$

Figure 4: The translated example grammar $\mathcal{G}'_{ww}$

$\Sigma_{\varepsilon,(3,1)} = \{\pi^3_{1\,(3,1)}, \pi^3_{2\,(3,1)}, \pi^3_{3\,(3,1)}\}$,

$$\begin{array}{rclrcl}
\Sigma_{(0,3)(3,3),(0,3)} & = & \{c_{(0,3,3)}\} & \Sigma_{(3,1)(3,1),(3,2)} & = & \{(\ )_{(3,2)}\} \\
\Sigma_{(0,3)(3,1),(0,1)} & = & \{c_{(0,3,1)}\} & \Sigma_{(0,1)(0,1)(0,1),(0,3)} & = & \{(\ )_{(0,3)}\} \\
\Sigma_{(3,2)(2,1),(3,1)} & = & \{c_{(3,2,1)}\} & \Sigma_{(3,1)(3,1)(3,1),(3,3)} & = & \{(\ )_{(3,3)}\} \\
\Sigma_{(3,0)(0,1),(3,1)} & = & \{c_{(3,0,1)}\} & & &
\end{array}$$

and $\mathsf{F}_0 = F_{\varepsilon,(0,1)} \cup F_{\varepsilon,(0,3)}$ with $F_{\varepsilon,(0,1)} = \{S_{(0,1)}\}$ and $F_{\varepsilon,(0,3)} = \{U_{(0,3)}, U_{i(0,3)}, V_{(0,3)}, V_{i(0,3)}, W_{1(0,3)}, W_{2(0,3)}, X_{1(0,3)}, C_{(0,3)}\}$.[16]

As one can see in Fig. 4, the basic functions have been realized as terms with their respective implicit operations as nonterminal (composition and tupling) or terminal (projection and empty tupling) nodes. In the following paragraphs, we sketch the translation T from nonterminal rules of the example MCFG to RTG-rules. T takes each rule $X \longrightarrow f(Y)$, where $X, Y \in V_N$ and $f \in V_F$, of the MCFG including the corresponding definition of the mapping $f(x_1, \ldots, x_k)$ with $k \geq 0$ and transforms it into a RTG-rule as follows. We create a mother node labeled with the appropriate binary composition $c_{(j,k,l)}$ such that the left daughter contains the "lifted" version of $f(x_1, \ldots, x_k)$ under T and the right daughter the translation of the nonterminal Y. Both nonterminals X and Y are used "unchanged", but annotated with the corresponding Lawvere arity resulting in the following schematic presentation of the translation: $X_{(j,l)} \longrightarrow c_{(j,k,l)}(\mathsf{T}(f(x_1, \ldots, x_k)), Y_{(j,k)})$, where f is a mapping from k-tuples to l-tuples of terminal strings.

Note that having more than one nonterminal on the RHS of an MCFG-rule leads to an RTG-rule which requires an additional tupling node above the corresponding nonterminals in the second argument of the composition. This tupling node has to contain the information how to compose the tuples resulting from the computation of the nonterminals, e.g., if the tuple is of length four and dominates two nonterminals, the first nonterminal could contribute one, two or three components and correspondingly the second nonterminal three, two or one component. Suppose we indicate this splitting with a superscript. So, there is no unique tupling node of a certain type anymore, but a family of tupling operators of each type which can be differentiated via their superscripts. We omit this—in our case trivial—tupling node in the example for better readability.

The easiest case of translating a mapping $f \in F$ from our example via T is constituted by the terminal U-rules (5a). We simply view the mapping as a Lawvere term. The function $merge_{\langle \beta_i, \alpha_i \rangle}$ just returns a triple built from i, i and ε. The corresponding tree has a mother node labeled with a ternary tupling symbol and the three unary arguments of the mapping as daughters.[17] The nonterminating U-rule (5e) is more complicated: the function $move_{\langle \psi_{\_} \rangle}$ takes three arguments, concatenates the last and first one in the first argument slot, leaves the second slot untouched and inserts ε into the last one. The definition of the function can be written explicitly as the Lawvere term $(\ )_{(3,3)}(c_{(3,2,1)}(\bullet_{(2,1)}, (\ )_{(3,2)}(\pi^3_{3(3,1)}, \pi^3_{1(3,1)})), \pi^3_{2(3,1)}, \varepsilon_{(3,1)})$. Note again that the implicit binary concatenation $\bullet$ in $move_{\langle \psi_{\_} \rangle}$ now becomes the constant $\bullet_{(2,1)}$. The variables are simply replaced by the projections and are concatenated. The resulting term is then applied via composition to the operative $V_{(0,3)}$ such that we get the RHS displayed in the last disjunct of the $U_{(0,3)}$-rule in Fig. 4. A comparable procedure is followed with respect to the other MCFG-rules.

Since RTGs can only generate recognizable (tree) languages, we can characterize them with both MSO logic on trees and tree automata. The tree automaton $\mathfrak{A}_{\mathcal{G}'_{ww}}$ is constructed by transforming the grammar into a normal form such that each RHS is of depth one by introducing auxiliary operatives. Then we can easily construct appropriate transitions by basically reversing the arrow: the nonterminals become state names and the mother node will be read as alphabet symbol. It is know from Thomas (1990) how to transform this tree automaton into a Σ^1_1-formula $\varphi_{\mathfrak{A}_{\mathcal{G}'_{ww}}}$ by encoding its behaviour. In short, assuming an enumeration of $\mathfrak{A}$'s states $\{0, \ldots, m\}$ such that the initial state is represented by 0, the formula uses sets X_i to label the nodes where the

[16] For simplicity and readability we will sometimes drop the subscript notion (k, m) from the inoperatives and operatives of rank 0, and sometimes even from the composition symbol $c_{(k,l,m)}$.

[17] Note that we do not need to use a further composition symbol dominating $\mathsf{T}(f)$ in case there is no nonterminal on the RHS of the rule of the MCFG.

automaton assumes state i. The first line of the formula says that we cannot have a node which is in two states and that X_0 is our "initial" set; the second one licenses the distribution of the sets according to the transitions and says that we need a root node which is in a "final" set. P_a stands for the predicate labeling a node with the symbol a. For n-ary tree automata the formula $\varphi_{\mathfrak{A}}$ looks as follows:

$$(\exists X_0,\dots,X_m)[\bigwedge_{i\neq j}(\neg\exists y)[y\in X_i \wedge y \in X_j] \wedge (\forall x)[(\neg\exists y)[x \lhd y] \rightarrow x \in X_0] \wedge$$
$$(\forall x_1,\dots,x_n,y)[\bigvee_{\substack{(i_1,\dots,i_n,\sigma,j)\in\alpha \\ 1\leq k\leq n}} x_k \in X_{i_k} \wedge y \lhd x_k \wedge y \in X_j \wedge y \in P_\sigma] \bigvee_{i\in F}(\exists x \forall y)[x \lhd^* y \wedge x \in X_i]$$

A more detailed description how to construct both the tree automaton and the corresponding MSO formula can be found in Kolb et al. (2000).

7 Reconstructing the Intended Trees

Unfortunately, the terminal tree in Fig. 5 generated/recognized by $\mathcal{G}'_{ww}$ given in Fig. 4, does not seem to have much in common with the structures linguists want to talk about.

However, in some sense to be made operational, the $\mathcal{G}'_{ww}$ structures contain the intended structures. As mentioned before, there is a mapping h from these explicit structures onto structures interpreting the $c_{(u,v,w)}$, $(\ )_{(v,u)}$ and the π^u_i the way the names we have given them suggest, *viz.* as compositions, tuplings and projections, respectively, which are, in fact, exactly the intended structures.

On the denotational side, we will use an MSO definable tree transduction (as defined in Sec. 2.3) and operationally we will use a Macro Tree Transducer (see Sec. 2.4) to transform the "lifted" structures into the intended ones.

7.1 The MSO Transduction

Rogers (1998) has shown the suitability of an MSO description language $L^2_{K,P}$ for linguistics which is based upon the primitive relations of immediate ($\lhd$), proper ($\lhd^+$) and reflexive ($\lhd^*$) dominance and proper precedence ($\prec$). We will show how to define these relations with an MSO transduction thereby implementing the unique homomorphism mapping the terms into elements of the corresponding multiple context-free tree language, i.e., the trees linguists want to talk about.

Put differently, it should be possible to define a set of relations $R^I = \{\lhd, \lhd^+, \lhd^*, \prec, \dots\}$ holding between the nodes of the explicit or "lifted" trees which carry a *"linguistic"* label in such a way, that when interpreting $\lhd^* \in R^I$ as a tree order on the set of *"linguistic"* nodes and $\prec \in R^I$ as the precedence relation on the resulting structure we have a "new" description language on the intended structures.

We have shown in Michaelis et al. (2000a) how to give an operational account of an MSO transduction to recover the intended relations via so called tree-walking automata with MSO tests.[18] In this paper, we will present the logical aspect of this transduction without going into the details of how to generate the relevant formulas. The interested reader is referred to the reference given above.

We will use $\mathsf{trans}_{W_\lhd}(x,y)$ as the formula denoting immediate dominance $(x \lhd y)$ on the intended structures. This formula was constructed recursively from the walking language of a tree-walking automaton linking the appropriate nodes in the lifted tree. An example of these relations is displayed graphically in Fig. 5 which contains a rendering of a simple tree generated by the RTG $\mathcal{G}'_{ww}$. The intended dominance relation marks the endpoints of these tree walks.

Suppose we want to find the two daughters of the leftmost concatenation symbol (•). What we have to do is find its sister (a tupling node) and use the daughters of that node as the immediate

[18] A tree-walking automaton with MSO tests is a finite state automaton which can navigate through a tree by following simple directives or by testing properties of nodes via MSO formulas. Bloem and Engelfriet (1997) show that the relations between two nodes recognized by their walks is constructively MSO definable.

daughters of the concatenation. These are indicated in the figure with the curved arrows. If these nodes are labeled with a "linguistic" label, we are done, if not we have to find the appropriate nodes recursively.[19] The case of finding the appropriate filler for a node labeled with a projection symbol is illustrated with the $\pi^3_{3(3,1)}$-link where we have to recursively traverse the tree (more on the details of how to find these can be found in Michaelis et al. (2000a,b); Kolb et al. (2000)). The resulting immediate dominance links are indicated by the grey lines.

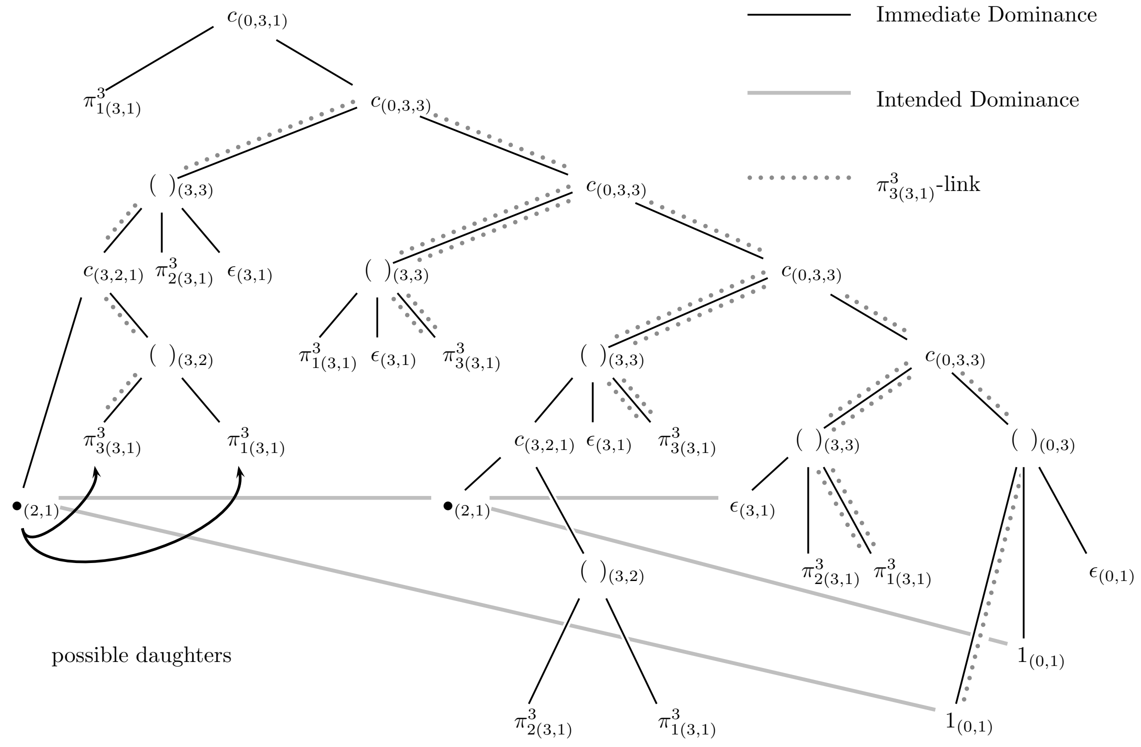


Figure 5: Intended relations on a lifted structure

Presupposing this definition of immediate dominance, we can define the other relations we need for the MSO transduction as follows.

For the case of the recursion inherent in reflexive dominance a standard solution exists in MSO logic on finite trees. It is a well-known fact (e.g., Courcelle 1990) that the reflexive transitive closure R^* of a binary relation R on nodes is (weakly) MSO-definable, if R itself is. This is done via a second-order property which holds of the sets of nodes which are closed under R:

(9) $$R\text{-}\mathsf{closed}(X) \overset{def}{\Longleftrightarrow} (\forall x, y)[x \in X \wedge R(x, y) \rightarrow y \in X]$$

Now, for any node n, the intersection of all such sets which contain n is exactly the set of m, such that $R^*(n, m)$. Since we are dealing with the (necessarily finite) trees generated by a context-free grammar, this construction can be safely exploited for our purposes; $\triangleleft^*$ and $\triangleleft^+$ can be defined as follows:

(10) *Reflexive Dominance:*

$$x \triangleleft^* y \quad \overset{def}{\Longleftrightarrow} \quad (\forall X)[\triangleleft\text{-}\mathsf{closed}(X) \wedge x \in X \rightarrow y \in X]$$

Proper dominance:

$$x \triangleleft^+ y \quad \overset{def}{\Longleftrightarrow} \quad x \triangleleft^* y \wedge x \not\approx y$$

[19]This recursion makes the use of the tree-walking automata imperative since in general MSO relations cannot be defined recursively.

Using the defined formula $\mathsf{trans}_{W_\triangleleft}(x,y)$ for $\triangleleft$, the specific MSO transduction we need to transform the "lifted" structures into the intended ones looks as follows:

$$(\varphi, \psi, (\theta_q)_{q \in Q})$$

$$Q = \{\triangleleft, \triangleleft^*, \triangleleft^+, \prec, \dots\}$$

(11)
$$\begin{array}{rcl}
\varphi & \equiv & \varphi_{\mathfrak{A}_{G'_{ww}}} \\
\psi(x) & \equiv & (\exists y)[x \triangleleft y \vee y \triangleleft x] \\
\theta_\triangleleft(x,y) & \equiv & \mathsf{trans}_{W_\triangleleft}(x,y) \\
\theta_{\triangleleft^*}(x,y) & \equiv & (\forall X)[\triangleleft\text{-}\mathsf{closed}(X) \wedge x \in X \rightarrow y \in X] \\
\theta_{\triangleleft^+}(x,y) & \equiv & x \triangleleft^* y \vee x \not\approx y \\
\theta_\prec(x,y) & \equiv & \mathsf{trans}_{W_\prec}(x,y) \\
\theta_{\mathrm{L} \in \mathrm{Labels}}(x) & \equiv & L(x)
\end{array}$$

As desired, the domain of the transduction is characterized by the MSO formula for the "lifted" trees (see the end of Sec. 6). The domain, i.e., the set of nodes, of the intended tree is characterized by the formula ψ which identifies the nodes with a "linguistic" label which stand indeed in the new dominance relation to some other node. Building on it, we define the other primitives of our description language analogous to the MSO language $L^2_{K,P}$ used to analyze large parts of GB theory in Rogers (1998). For reasons of space, we have to leave the specification of the precedence relation $\mathsf{trans}_{W_\prec}(x,y)$ open. It is more complicated than dominance, but can be achieved with another tree-walking automaton. Finally, the labeling information for the nodes is just taken over from R.

Note also that while standardly "linguistic" relations like *c-command* or *government* would be defined in terms of *dominance*, our approach allows the alternative route of taking, in the spirit of Frank and Vijay-Shanker (1998), *c-command* as the primitive relation of linguistic structure by defining, in a similar, though—since Chomsky's (1985) distinction between *segments* and *categories* has to be accommodated—somewhat more complicated fashion, an automaton $\mathfrak{A}_{c\text{-}command}$, which computes the intended *c-command* relation directly, without recourse to dominance.

7.2 The Macro Tree Transducer

As stated previously, there is a *unique* morphism h from the "lifted" terms over the derived alphabet Σ into the terms over the tree substitution algebra. The morphism h is defined inductively as follows:

(12)
$$\begin{array}{rcl}
h(f') & = & f(x_1, \dots, x_n) \text{ for } f \in \Sigma_{w,s}, |w| = n \\
h(\pi^u_i) & = & x_i \\
h((\ )_{(u,v)}(t_1, \dots, t_u)) & = & (h(t_1), \dots, h(t_u)) \\
h(c_{(u,v,w)}(t_1, t_2)) & = & h(t_1)[h(t_2)]
\end{array}$$

where $t[t_1, \cdots, t_k]$ denotes the result of substituting t_i for x_i in t for $t \in T(\Sigma, X_k)$, $t_i \in T(\Sigma, X_m)$.[20]

This unique morphism h can be performed by a simple macro tree transducer $M = \langle Q, \Sigma, V_T \cup \{\bullet\}, q_0, P \rangle$, where $Q = \{q_u \,|\, \text{for all } c_{(u,v,w)} \in \Sigma$ where the leftmost daughter is not a tupling node $\} \cup \{q^i_u \,|$ for all $c_{(u,v,w)} \in \Sigma$ where the leftmost daughter is a tupling node $(\ )_{(r,s)} \in \Sigma, i \in \{1, \dots, s\}\} \cup \{q^i_v \,|$ for all $(\ )_{(v,u)} \in \Sigma, i \in \{1, \dots, u\}\}$,[21] q_0 is the initial state and P is a finite family of rules. Recall, that we use only a simple example with just one nonterminal on the RHS of the MCFG-rules. This simplifies the "lifted" trees as well as the MTT we have to define.

The MTT which we construct to carry out the transformation effected by the unique homomorphism h combines in a particularly perspicuous way the actions of a top-down finite tree

[20]It is immediately obvious how one can translate this recursive definition into a simple Prolog program. But since we can simulate a Turing machine with Prolog, nothing much is gained on the formal side. In case one considers implementing the approach, it makes sense to use this simple homomorphism directly.

[21]Note that this informal motivation for the needed states does not mean that the resulting transitions will refer to information about particular daughters.

transducer—based upon the syntactic structure of the "lifted" Lawvere alphabet Σ—and the production aspect of the underlying MCFG via its (i.e., the MTT's) dependence on the local context (parameters): retrieving the "old" arity out of the new sorted constants of the signature Σ.

How can we construct the necessary productions to recover the intended trees? We take an intuitive approach to explaining the construction of the needed MTT which is strongly dependent on inspection of the tree in Fig. 5 and the given homomorphism h.

Before we proceed, we briefly review some notation for the states introduced above. For convenience, the states have both sub- and superscripts. The subscripts convey information about the number of parameters and the superscripts hint at computations which will return the particular arguments of a tupling operation. Furthermore, we will also use variables with both super- and subscripts. In this case, the superscripts indicate the type of the mother node of the tree which has to be substituted, whereas the subscripts just give us new variables of the same type. This typing of the variables is strictly speaking not necessary since one can either assume that the input trees of the MTT have been generated with an appropriate RTG and therefore are well-typed; or, alternatively, that we could change the definition of the MTT to an MTT with regular lookahead. Then the lookahead—implemented with a tree automaton—ensures the well-typing of the input.

In general, in the first argument of a rule from P we will have a tree during a transduction. So in the rules, we have to take care of all symbols which can appear as mothers of (possibly trivial) trees with the number of variables x_i corresponding to their arities in the first argument of any LHS.

After careful inspection of the tree language generated by the lifted RTG $\mathcal{G}'_{ww}$, the simplest case is certainly when we are faced with a constant, i.e., with an element from Σ with rank 0. In this case all we have to do is to map it back to the corresponding element from $V_T \cup \{\bullet\}$, regardless of the parameters, if there are any. In case we encounter a projection symbol we simply have to return the corresponding parameter. Obviously, this presupposes that we stored the "right" information there.

Furthermore, all rules with a constant symbol whose "unlifted" version was not a constant—in this case only the concatenation symbol $\bullet$—need as many parameters as are needed to compute the corresponding function, e.g., $\bullet$ obviously is binary and therefore needs two parameters (see the third rule in (13)). The resulting rule has, on the RHS, simply the "executed" function.

The more complicated cases involve branching nodes in the tree. Those are labeled with either a tupling or a composition symbol introduced by the lifting process. Let us turn to the easier case of tupling first.

We have to construct the rules for nodes labeled with a tupling symbol by inspecting the sort information in the subscript. Furthermore, the state on the LHS is marked with a superscript indicating along which branch of the tuple we have to descend, i.e., which argument/daughter of the tupling node we are currently evaluating. Given a symbol $(\ )_{(v,u)}$, we construct a transition with state q^i_v of arity $v+1$ which has as arguments a term labeled with $(\ )_{(v,u)}$ which has an appropriate number u of daughters x_j, and v parameters on the LHS. On the RHS, we start a "fresh" transducer (state q_v) on argument x_i of the term and the parameters, therefore q_v is also of arity $v+1$.

For the rules headed by a composition symbol $c_{(u,v,w)}$ we need as many parameters on the LHS as are prescribed by u and as many parameters on the RHS as prescribed by v. This is due to the fact that while generally the relevant information in the lifted trees is on the leftmost branch, we nevertheless need the other daughters to be able to unravel the projections. Basically, we follow a depth-first strategy on the leftmost component of the lifted trees while still passing the necessary context (i.e., the evaluation of the computation of the other daughters) down into that computation. Similarly, we also get the necessary states from the arities of the composition symbols. The rules then simply pass the state and the parameters of the LHS of the rule to the arguments of the alphabet symbol while continuing to work on the first argument. As an example consider a fork whose mother is labeled with $c_{(u,v,w)}$. Then we have to construct a rule which has on the LHS state q_u with arity $u+1$. It has as its first argument a term with functor $c_{(u,v,w)}$ and two appropriately typed arguments $x^{(v,w)}$ and $x^{(u,v)}$. The other arguments are the parameters y_1

to y_u. The RHS has state q_v of arity $v+1$ with the first argument simply being the first daughter of the composition symbol, i.e., $x^{(v,w)}$, and the other arguments being $q_u^{i-1}(x^{(u,v)}, y_1, \ldots, y_u)$, $(ilequ)$, with the interpretation that we have no parameters if $u < 1$. Furthermore, we have to supplement the state on the LHS with superscripts according to the typing information if the leftmost, i.e., the head-daughter carries a tupling symbol. More concretely, if the head-daughter of $c_{(u,v,w)}$ is $(\ )_{(s,t)}$ we need t transitions with superscripts ranging from 1 to t, i.e., q_u^i where $i \in \{1, \ldots, t\}$. And then, if the state on the LHS had a superscript, i.e., we are working on the computation of a value of an element of a tuple, we simply pass this superscript on to the state on the RHS.

For our concrete example, the set of rules P of the MTT M look as given below:[22]

$$
\begin{array}{rll}
q_0(c_{(0,3,1)}(x_1^{(3,1)}, x_1^{(0,3)})) & \longrightarrow q_3(x_1^{(3,1)}, q_0^1(x_1^{(0,3)}), q_0^2(x_1^{(0,3)}), q_0^3(x_1^{(0,3)})) & \\
q_0(\sigma) & \longrightarrow \sigma & \text{for } \sigma \in \{1_{(0,1)}, 2_{(0,1)}, \varepsilon_{(0,1)}\} \\
q_2(\bullet_{(2,1)}, y_1, y_2) & \longrightarrow \bullet_{(2,1)}\ (y_1, y_2) & \\
q_3(P_i, y_1, y_2, y_3) & \longrightarrow y_i & \text{for } P_i = \pi_i \\
q_3(\sigma, y_1, y_2, y_3) & \longrightarrow \sigma & \text{for } \sigma \in \{1_{(3,1)}, 2_{(3,1)}, \varepsilon_{(3,1)}\} \\
q_3(c_{(3,2,1)}(x_1^{(2,1)}, x_1^{(3,2)}), y_1, y_2, y_3) & \longrightarrow q_2(x_1^{(2,1)}, q_3^1(x_1^{(3,2)}, y_1, y_2, y_3), & \\
 & \qquad q_3^2(x_1^{(3,2)}, y_1, y_2, y_3)) & \\
q_0^i(c_{(0,3,3)}(x_1^{(3,3)}, x_1^{(0,3)})) & \longrightarrow q_3^i(x_1^{(3,3)}, q_0^1(x_1^{(0,3)}), q_0^2(x_1^{(0,3)}), q_0^3(x_1^{(0,3)})) & \\
 & & \text{for } i \in \{1,2,3\} \\
q_0^i((\ )_{(0,3)}(x_1^{(0,1)}, x_2^{(0,1)}, x_3^{(0,1)})) & \longrightarrow q_0(x_i^{(0,1)}) & \text{for } i \in \{1,2,3\} \\
q_3^i((\ )_{(3,3)}(x_1^{(3,1)}, x_2^{(3,1)}, x_3^{(3,1)}), y_1, y_2, y_3) & \longrightarrow q_3(x_i^{(3,1)}, y_1, y_2, y_3) & \text{for } i \in \{1,2,3\} \\
q_3^i((\ )_{(3,2)}(x_1^{(3,1)}, x_2^{(3,1)}), y_1, y_2, y_3) & \longrightarrow q_3(x_i^{(3,1)}, y_1, y_2, y_3) & \text{for } i \in \{1,2\}
\end{array}
\tag{13}
$$

As one can see, the only remaining tree forming symbol which remains on the RHSs is the concatenation $\bullet$. So, we are indeed back with our "old" alphabet. The parameters serve just as memory slots to pass the necessary information for undoing the projections and explicit compositions further down the tree.

Applying this MTT to the tree in Fig. 5 yields a final tree for a derivation of the MCFG displayed in Example 4.3. Namely the one indicated by the grey lines in Fig. 5. To get the reader started, let us consider the first rule in (13) beginning the transduction on the root of the tree displayed in Fig. 5. We start in state q_0 and our root is indeed labeled with $c_{(0,3,1)}$. Then we continue in state q_3 with its leftmost daughter and pass as parameter y_i the result of computing q_0^i of the second daughter (in this case with $i \in \{1, 2, 3\}$). These transductions have to be computed separately and yield the input to the "final" projection π_1^3. The rest of the computation leading to the final result is straightforward, if tedious, and left as an exercise.

8 Conclusion

Taking the result of Michaelis' translation of MGs as the input we have shown how to define a RTG by lifting the corresponding MCFG-rules into terms of a free Lawvere theory. This gives us both a regular (via tree automata and macro tree transducers) and a logical description (via MSO logic and an MSO definable transduction) of the intended syntactic trees. Equivalently, we provide both an operational and a denotational account of Stabler's version of Minimalism without having to go via derivation trees.

In the wake of the celebrated result of Peters and Ritchie (1973) on the generative strength of Transformational Grammars a great number of research activities were inspired by the so-called

[22] Note that the transitions leading to constant symbols of type $(3,1)$ are artifacts of the simplification we alluded to in fn. 15. Note furthermore, that the use of more than one nonterminal on the RHS of an MCFG-rule entails, as outlined in Sec. 6, extra tupling nodes which naturally have to be accommodated by the MTT. In particular, it has to make use of the additional superscripts to determine the appropriate daughter to find the correct value.

universal base hypothesis. One version of this hypothesis can be paraphrased as claiming that there exists a fixed grammar G that plays the rôle of the base component of a Transformational Grammar of any natural language. Adapting this methodological point to our result it can be stated as follows: Empirical linguistic phenomena that can be accommodated within the framework of MGs are amenable to a regular analysis followed by a fixed universal transduction.

Comparing this statement of the result of the paper with the characterization of context-free graph languages by Engelfriet and van Oostrom (1996), we want to stress the point that our regular description of MCFG languages does not provide a characterization of this language family in the technical understanding of an equivalence between MCFG languages and languages defined by a regular tree language/closed MSO formula and a macro tree transducer/MSO transduction. For a recent result on the equivalence between regular tree languages followed by an MSO definable tree transduction and the tree languages generated by context-free graph grammars see Engelfriet and Maneth (1999).

Acknowledgments

The research presented in this paper was supported by the *Deutsche Forschungsgemeinschaft* within the *Sonderforschungsbereich* 340, TP A8. The authors wish to thank Kai-Uwe Kühnberger and Stephan Kepser for helpful discussions.

References

Bloem, R. and Engelfriet, J. (1997). Characterization of properties and relations defined in Monadic Second Order logic on the nodes of trees. Technical Report 97-03, Dept. of Computer Science, Leiden University.

Bloem, R. and Engelfriet, J. (1998). A comparison of tree transductions defined by monadic second order logic and by attribute grammars. Technical Report 98-02, Leiden University Technical Report.

Chomsky, N. (1985). *Barriers.* MIT Press, Cambridge, MA.

Courcelle, B. (1990). Graph rewriting: An algebraic and logic approach. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science*, volume B, pages 193–242. Elsevier.

Courcelle, B. (1997). The expression of graph properties and graph transformations in monadic second-order logic. In Rozenberg, G., editor, *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*, chapter 5, pages 313–400. World Scientific.

Doner, J. E. (1970). Tree acceptors and some of their applications. *J. Comput. System Sci.*, 4:406–451.

Duske, J., Parchmann, R., Sedello, M., and Specht, J. (1977). IO-macrolanguages and attributed translations. *Information and Control*, 35(2):87–105.

Engelfriet, J. (1997). Context-free graph grammars. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages. Vol. III: Beyond Words*, chapter 3, pages 125–213. Springer.

Engelfriet, J. and Maneth, S. (1999). Macro tree transducers, attribute grammars, and MSO definable tree translations. *Information and Computation*, 154:34–91.

Engelfriet, J. and Maneth, S. (2000). Tree languages generated by context-free graph grammars. In H. Ehrig, G. Engels, H.-J. K. and Rozenberg, G., editors, *Proceedings of Theory and Applications of Graph Transformations - TAGT'98*, number 1764 in LNCS, pages 15–29.

Engelfriet, J. and van Oostrom, V. (1996). Regular description of context-free graph languages. *Journal Comp. & Syst. Sci.*, 53(3):556–574.

Engelfriet, J. and Vogler, H. (1985). Macro tree transducers. *Journal of Computer and System Sciences*, 31(1):71–146.

Frank, R. and Vijay-Shanker, K. (1998). Primitive c-command. Ms., Johns Hopkins University & Universtiy of Delaware.

Fülöp, Z. (1981). On attributed tree transducers. *Acta Cybernetica*, 5:261–279.

Gécseg, F. and Steinby, M. (1984). *Tree Automata*. Akadémiai Kiadó, Budapest.

Gécseg, F. and Steinby, M. (1997). Tree languages. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages: Beyond Words*, volume 3, Berlin. Springer.

Huybregts, M. A. C. (1976). Overlapping dependencies in dutch. *Utrecht Working Papers in Linguistics*, 1:24–65.

Huybregts, M. A. C. (1984). The weak adequacy of context-free phrase structure grammar. In de Haan, G. J., Trommelen, M., and Zonneveld, W., editors, *Van periferie naar kern*, pages 81–99. Foris, Dordrecht.

Kolb, H.-P., Mönnich, U., and Morawietz, F. (2000). Descriptions of cross-serial dependencies. To appear in a special issue of *Grammars*. Draft available under `http://tcl.sfs.nphil.uni-tuebingen.de/~frank/`.

Mezei, J. and Wright, J. (1967). Algebraic automata and contextfree sets. *Information and Control*, 11:3–29.

Michaelis, J. (1999). Derivational minimalism is mildly context-sensitive. *Linguistics in Potsdam (LiP)* 5, Institut für Linguistik, Universität Potsdam. Available under `http://www.ling.uni-potsdam.de/~michael/`.

Michaelis, J. and Kracht, M. (1997). Semilinearity as a syntactic invariant. In Retoré, C., editor, *Proceedings of the 1st International Conference on Logical Aspects of Computational Linguistics (LACL-96)*, volume 1328 of *LNAI*, pages 329–345, Berlin. Springer.

Michaelis, J., Mönnich, U., and Morawietz, F. (2000a). Algebraic descriptions of derivational minimalism. In *Algebraic Methods in Language Processing: Second AMAST Workshop on Language Processing*, TWLT, University of Iowa. Parlevink.

Michaelis, J., Mönnich, U., and Morawietz, F. (2000b). Derivational minimalism in two regular and logical steps. In *Proceedings of the Tag+5 Conference*, Paris.

Mönnich, U. (1997). Lexikalisch kontrollierte Kreuzabhängigkeiten und Kongruenzmorphologien. Talk at the Innovationskolleg Potsdam. Available under `http://tcl.sfs.nphil.uni-tuebingen.de/~tcl/uwe/uwe_moennich.html`.

Mönnich, U. (1998). TAGs M-constructed. In *TAG+ 4th Workshop, Philadelphia*.

Peters, P. S. and Ritchie, R. W. (1973). On the generative power of transformational grammars. *Information Sciences*, 6:49–83.

Pullum, G. and Gazdar, G. (1982). Natural languages and context-free languages. *Linguistics and Philosophy*, 4(4):471–504.

Rabin, M. O. (1969). Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35.

Rambow, O. and Satta, G. (1999). Independent parallelism in finite copying parallel rewriting systems. *Theoretical Computer Science*, 223(1–2):87–120.

Rogers, J. (1998). *A Descriptive Approach to Language-Theoretic Complexity.* Studies in Logic, Language, and Information. CSLI Publications and FoLLI.

Seki, H., Matsumura, T., Fujii, M., and Kasami, T. (1991). On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Shieber, S. M. (1985). Evidence against the context-freeness of natural language. *Linguistics & Philosophy*, 8(3):333–344.

Stabler, E. (1997). Derivational minimalism. In Retoré, C., editor, *Logical Aspects of Computational Linguistics*, pages 68–95, Berlin. Springer. LNAI 1328.

Stabler, E. (1999a). Minimalist grammars and recognition. Presented at the Workshop *Linguistic Form and its Computation* of the SFB 340 in Bad Teinach. Draft for the same volume.

Stabler, E. (1999b). Remnant movement and complexity. In Bouma, G., Kruijff, G.-J. M., Hinrichs, E., and Oehrle, R. T., editors, *Constraints and Resources in Natural Language Syntax and Semantics*, volume II of *Studies in Constrained Based Lexicalism*, pages 299–326. CSLI.

Thatcher, J. W. and Wright, J. B. (1968). Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81.

Thomas, W. (1990). Automata on infinite objects. In van Leeuwen, J., editor, *Handbook of Theoretical Computer Science*, chapter 4, pages 133–191. Elsevier Science Publishers B. V.

Wagner, E. G. (1994). Algebraic semantics. In Abramsky, S., Gabbay, D. M., and Maibaum, T. S. E., editors, *Semantic Structures*, volume 3 of *Handbook of Logic in Computer Science*, pages 323–393. Oxford University Press.

Weir, D. J. (1992). Linear context-free rewriting systems and deterministic tree-walk transducers. In *30th Meeting of the Association for Computational Linguistics (ACL'92).*