Regular Description of Cross-Serial Dependencies

Hans-Peter Kolb and **Uwe Mönnich** and **Frank Morawietz**
Universität Tübingen
Seminar für Sprachwissenschaft
Wilhelmstr. 113, 72074 Tübingen
{kolb,um,frank}@sfs.nphil.uni-tuebingen.de

Abstract

The main result of this paper is a description of cross-serial dependencies in terms of regular tree languages (to express the recursive properties) and regular string languages (to establish the intended linguistic relations). In fact, we shall not deal directly with these two types of regular language families, but shall instead consider implementation in the form of finite-state (tree) automata to emphasize the procedural aspect of our approach.

1 Introduction

There are many kinds of structural phenomena in natural languages that cannot be captured by context-free string grammars or regular tree grammars. Among these phenomena cross-serial dependencies have played a prominent rôle in the discussions about the right level of complexity to be assumed for a descriptively adequate linguistic theory.

The main result of this paper is a description of cross-serial dependencies in terms of regular tree languages (to express the recursive properties) and regular string languages (to establish the intended linguistic relations). In fact, we shall not deal directly with these two types of regular language families, but shall instead consider implementation in the form of finite-state automata to emphasize the procedural aspect of our approach.

The structure of the paper is as follows. After a summary of some technical notions from algebraic language theory, we outline a grammatical representation of a particular example of cross-serial dependencies, the verbal complex in Swiss German and Dutch, within the formalism of context-free tree grammars. This formalism was introduced by Fischer (1968) and constitutes a generalization of regular tree grammars. In perfect analogy with the step from regular to context-free string grammars, which, essentially, consists in lifting the restriction to initial and final string positions for nodes to be substitutable, the rule format which characterizes context-free tree grammars allows for the substitution of inner tree nodes in contradiction to the situation in regular tree grammars that limit substitubility to terminal nodes. After this first step we LIFT the generated trees by inserting a certain amount of explicit control information. The resulting structures can be characterized with regular means and we exploit the equivalence between finite bottom-up tree automata and regular tree grammars in constructing an automaton that cuts out of the universal realm of all possible finite trees on the explicit signature just those elements that have counterparts in the original context-free tree family. The final step simulates the structural relations of the original trees. These structural relations turn out to be paths in a regular string language that is characterized by a finite tree-walking automaton.

The basic idea underlying this sequential application of two finite-state devices can be found in the signature free treatment of universal algebra that was developed in the early 60s. According to this treatment, the most important rôle in an algebra is played by the set of all operations definable from the primitive ones by composition and not by the primitive operations themselves. These defined operations then constitute the carrier of a (multi-sorted) algebra whose only non-unary operations are provided by suitably sorted instances of composition.

In the same way that strings (in concatenation algebras) and trees (in term algebras) are the appropriate ranges for the variables in context-free string and regular tree grammars, respectively, defined tree operations are the appropriate range for variables in context-free tree grammars. In the case of strings and trees the process of substituting an element of the carrier of these algebras for a variable involves a simple process of insertion whereas in the case of defined operations the original arguments of the replaced (multi-ary) variable have to be composed with its substitute. Once these two components of the process of higher-order substitution are pulled apart we are free to formally indicate them by means of separate notational ingredients, the variable substituend on the one hand and the composition operator on the other, and thereby arrive at the notion of a lifted free algebra.

Since the variables in the context of a lifted free algebra are separated from their arguments, the process of substituting an appropriate multi-ary defined operation for them takes the form of leaf substitution familiar from regular tree grammars. It was Maibaum (1974) who first adapted the treatment of universal algebra developed in category theory to the context of formal language theory. The topic was taken up again in Engelfriet and Schmidt (1977, 1978), who also give an account of the influence of different modes of derivation, an issue that was not correctly stated in Maibaum's presentation. The fact that derivation steps on the original and the lifted structural level are in one-one correspondence is proved in Mönnich (1999).

A result similar in spirit to the one established in this paper can be found in Engelfriet and van Oostrom (1996). They show that context-free graph languages have a regular path description. Given the context of graph theory the authors use for their first step a regular family of derivation trees and formalize the effect of connection instructions in terms of suitable node labels in the derivation trees. It is an open question whether the information on argument insertion contained in the context-free tree rewriting rules can be codified as a set of connection instructions along the format of context-free graph grammars. On the basis of such a translation it could be shown that a regular path description for context-free tree languages constitutes a particular case of the main result in Engelfriet and van Oostrom (1996).

We think that our approach has two decisive advantages. First, the connection instructions 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. The instruction set of the walking automaton is universal and only serves 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.

2 Basic Notions from Algebraic Language Theory

The purpose of this section is to fix notations and to present definitions for the basic notions related to tree grammars and the classes of automata which will be used in the paper. We have taken pains to give a full list of definitions to make the paper as self-contained as possible.

2.1 Tree Grammars

Recall that a *ranked alphabet* or *signature* Σ is an indexed family $\Sigma_n (n \in \mathbb{N})$ of disjoint sets. A symbol in Σ_n is called an operator of *rank* n. If $n = 0$ then $f \in \Sigma_0$ is also called a *constant*. Then we can define the set of trees over a ranked alphabet as follows.

Definition 1 (Trees). For a ranked alphabet Σ, we denote by $T(\Sigma)$ the set of *trees* over Σ. $T(\Sigma)$ is inductively defined as follows:

(i) $\Sigma_0 \subseteq T(\Sigma)$

(ii) If $f \in \Sigma_n$ and $t_i \in T(\Sigma)$ for $i = 1, \dots, n$
$$f(t_1, \dots, t_n) \in T(\Sigma)$$

Furthermore, we fix an indexed set $X = \{x_1, x_2, \dots\}$ of *variables* and denote by X_n the subset $\{x_1, \dots, x_n\}$. Variables are considered to be constants, i.e., operators of rank 0.

For a ranked alphabet Σ the family $T(\Sigma, X)$ is defined to be $T(\Sigma(X))$, where $\Sigma(X)$ is the ranked alphabet with $\Sigma(X)_0 = \Sigma_0 \cup X$ and $\Sigma(X)_n = \Sigma_n$ for every $n \neq 0$. A subset L of $T(\Sigma)$ is called a *tree language* over Σ.

Having described the tree terms, it remains to specify the central notion of an algebra and to give a precise definition of the way in which the operator symbols induce an operation on an algebra.

Definition 2. Suppose that Σ is a ranked alphabet. A Σ-*algebra* $\mathbb{A}$ is a pair $\mathbb{A} = (A, (f_{\mathbb{A}})_{f \in \Sigma})$ where the set A is the *carrier* of the algebra and for each operator $f \in \Sigma_n, f_{\mathbb{A}} : A^n \to A$ is an operation of arity n on A.

Different algebras, defined over the same operator domain, are related to each other if there exists a mapping between their carriers that is compatible with the basic structural operations.

Definition 3. A Σ-*homomorphism* of Σ-algebras $h : \mathbb{A} \longrightarrow \mathbb{B}$ is a function $h : A \longrightarrow B$, such that for every operator f of rank n

$$h(f_{\mathbb{A}}(a_1, \ldots, a_n)) = f_{\mathbb{B}}(h(a_1), \ldots, h(a_n))$$

for every n-tuple $(a_1, \ldots, a_n) \in A^n$.

The set of trees $T(\Sigma, X)$ can be made into a Σ-algebra by defining the operations in the following way. For every f in Σ_n, for every $(t_1, \ldots, t_n)$ in $T(\Sigma, X)^n$: $f_{\mathbb{T}(\Sigma,X)}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$.

Every variable-free tree $t \in T(\Sigma)$ has a value in every Σ-algebra $\mathbb{A}$. It is the value at t of the unique homomorphism $h : \mathbb{T}(\Sigma) \to \mathbb{A}$.

The existence of a unique homomorphism from the Σ-algebra of trees into an arbitrary Σ-algebra $\mathbb{A}$ provides also the basis for the view that regards the elements of $T(\Sigma, X_n)$ as *derived operations*. Each tree $t \in T(\Sigma, X_n)$ induces an n-ary function $t_{\mathbb{A}} : A^n \to A$.

The meaning of this function $t_{\mathbb{A}}$ is defined in the following way. For every $(a_1, \ldots, a_n) \in A^n$ $t_{\mathbb{A}}(a_1, \ldots, a_n) = \hat{a}(t)$, where $\hat{a} : \mathbb{T}(\Sigma, X_n) \to \mathbb{A}$ is the unique homomorphism with $\hat{a}(x_i) = a_i$.

In the particular case where $\mathbb{A}$ is the Σ-algebra $\mathbb{T}(\Sigma, X_m)$ of trees over Σ that contain at most variables from $X_m = \{x_1, \ldots, x_m\}$ at their leaves the unique homomorphism extending the assignment of a tree $t_i \in T(\Sigma, X_m)$ to the variable x_i in X_n acts as a substitution $t_{\mathbb{T}(\Sigma,X_m)}(t_1, \ldots, t_n) = t[t_1, \ldots, t_n]$ where the last tree indicates the result of substituting t_i for x_i in t.

We now turn to introducing the notion of a context-free tree grammar. This type of grammar is related to the type of grammars that were defined by (Fischer, 1968) and were called macro grammars by him. In his setting, the use of macro-like productions served the purpose of making simultaneous string copying a primitive operation. Context-free tree grammars constitute an algebraic generalization of macro grammars. since the use of macro-like productions served the purpose of making simultaneous string copying a primitive operation.

Let us view grammars as a mechanism in which local transformations on trees can be performed in a precise way. The central ingredient of a grammar is a finite set of productions, where each production is a pair of trees. Such a set of productions determines a binary relation on trees such that two trees t and t' stand in that relation if t' is the result of removing in t an occurrence of a first component in a production pair and replacing it by the second component of the same pair. The simplest type of such a replacement is defined by a production that specifies the substitution of a single-node tree t_0 by another tree t_1. Two trees t and t' satisfy the relation determined by this simple production if the tree t' differs from the tree t in having a subtree t_1 that is rooted at an occurrence of a leaf node t_0 in t. In slightly different terminology, productions of this kind incorporate instructions to rewrite auxiliary variables as a complex symbol that, autonomously, stands for an element of a tree algebra. As long as the carrier of a tree algebra is made of constant tree terms the process whereby nullary variables are replaced by trees is analogous to what happens in string languages when a nonterminal auxiliary symbol is rewritten as a string of terminal and non-terminal symbols, independently of the context in which it occurs. As we will see, the situation changes dramatically if the carrier of the algebra is made of symbolic counterparts of derived operations and the variables in production rules range over such second-level entities.

Definition 4 (Tree Grammar). A *tree grammar* Γ is a 5-tuple $\langle \Sigma, \mathbf{F}, S, \mathbf{X}, \mathbf{P} \rangle$, where $\Sigma, \mathbf{F}$ are the ranked alphabets of *inoperatives* and *operatives*, respectively; $S \in \mathbf{F}$ is the starting symbol; $\mathbf{X}$ is a countable set of variables; and $\mathbf{P}$ is a set of productions. The $p \in \mathbf{P}$ have the form $F(x_i, \cdots, x_n) \longrightarrow t$, with $F \in \mathbf{F}$, $x_i, \cdots, x_n \in \mathbf{X}$, and t a term (tree) over $\Sigma \cup \mathbf{F} \cup \{x_i, \cdots, x_n\}$.

An application of a rule $F(x_i, \cdots, x_n) \longrightarrow t$ "rewrites" a tree rooted in F as the tree t with its respective variables substituted by F's daughters.

Tree grammars with $\mathbf{F}_n = \emptyset, n \neq 0$, are called *regular*. Since they always just substitute some tree for a leaf-node, it is easy to see that they can only generate recognizable sets of trees, *a forteriori* context-free string languages (Mezei and Wright, 1967). If $\mathbf{F}_n, n \neq 0$, is non-empty, that is, if we allow the *operatives* to be parameterized by variables, however, the situation changes. These *context-free* (or *macro-*)tree grammars are capable of generating sets of structures, the *yields* of which belong to the class of mildly context-sensitive languages known as the *indexed* languages. In fact, they characterize this class exactly (Rounds, 1970).

For reasons having to do with the impossibility of mirroring the process of copying in a grammar with a completely uncontrolled derivation regime we restrict ourselves to one particular mode of derivation. According to this mode a function symbol may be replaced only if all its arguments are trees over the terminal alphabet. In the conventional case this form of replacement mechanism would correspond to a rightmost derivation.

Definition 5 (derivable). Let $\Gamma = \langle \Sigma, \mathbf{F}, S, \mathbf{X}, \mathbf{P} \rangle$ be a context-free tree grammar and let $t, t' \in T(\Sigma \cup \mathbf{F})$. t' is *directly derivable* by an *inside-out step* from t ($t \Longrightarrow t'$) if there is a tree $t_0 \in T(\Sigma \cup \mathbf{F}, X_1)$ containing exactly *one* occurrence of x_1, a corresponding rule $F(x_1, \dots, x_m) \to t''$, and trees $t_1, \dots, t_m \in T(\Sigma)$ such that

$$\begin{aligned} t &= t_0[F(t_1, \dots, t_m)] \\ t' &= t_0[t''[t_1, \dots, t_m]] \end{aligned}$$

In words: t' is obtained from t by replacing an occurrence of a subtree $F(t_1, \dots, t_m)$ by the tree $t''[t_1, \dots, t_m]$. By the inside-out restriction on the derivation scheme it is required that the trees t_1, t_2 through t_n be terminal trees.

As is customary $\stackrel{*}{\Longrightarrow}$ denotes the transitive-reflexive closure of $\Longrightarrow$.

Note that in the following definition of a tree language we now switch back to accepting only trees over the alphabet Σ, i.e., we do not allow operatives to remain in the final trees.

Definition 6 (Inside-Out Tree Language). Let $\Gamma = \langle \Sigma, \mathbf{F}, S, \mathbf{X}, \mathbf{P} \rangle$ be a context-free tree grammar. We call $\mathcal{L}(G, S) = \{t \in T(\Sigma) \mid S \stackrel{*}{\Longrightarrow} t\}$ the *context-free inside-out tree language* generated by G from S.

Note that in the case of regular grammars the analogy with the conventional string theory goes through. There is an equivalence of the unrestricted, the rightmost and the leftmost derivation modes where the terms "rightmost" and "leftmost" are to be understood with respect to the linear order of the leaves forming the frontier of a tree in a derivation step.

Definition 7 (yield). Suppose Σ is a ranked alphabet. We call *yield* the unique homomorphism y that interprets every operator in Σ_n as the n-ary operation of concatenation. More precisely

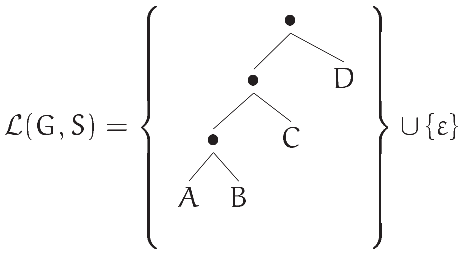
$$\begin{aligned} y(\sigma) &= \sigma && \text{for } \sigma \in \Sigma_0 \\ y(\sigma(t_1, \dots, t_n)) &= y(t_1) \dots y(t_n) && \text{for } \sigma \in \Sigma_n, \\ & && t_i \in T(\Sigma) \end{aligned}$$

We will exemplify the gain in generative power of context-free tree grammars compared to regular tree grammars – or standard context-free grammars – with an artificial construction of the string language $a^n b^m c^n d^m$ which is a subset of the actual non context-free dependencies occurring in Swiss German (see Sec. 3). The example uses the full power of the second-order substitutions of operators.

(1)
$$\begin{aligned} \Sigma_0 &= \{\varepsilon, a, b, c, d\} \\ \Sigma_2 &= \{\bullet\} \end{aligned}$$
$$\mathbf{F}_0 = \{S\} \qquad \mathbf{F}_4 = \{F\}$$
$$S = S \qquad \mathbf{X} = \{x_1, x_2, x_3, x_4\}$$

$$\mathbf{P} = \left\{ \begin{array}{rcl} S & \longrightarrow & \varepsilon \\ S & \longrightarrow & F(a, \varepsilon, c, \varepsilon) \\ S & \longrightarrow & F(\varepsilon, b, \varepsilon, d) \\ F(x_1, x_2, x_3, x_4) & \longrightarrow & \\ & & F(\bullet(a, x_1), x_2, \bullet(c, x_3), x_4) \\ F(x_1, x_2, x_3, x_4) & \longrightarrow & \\ & & F(x_1, \bullet(b, x_2), x_3, \bullet(d, x_4)) \\ F(x_1, x_2, x_3, x_4) & \longrightarrow & \\ & & \bullet(\bullet(\bullet(x_1, x_2), x_3), x_4) \end{array} \right\}$$

The tree language generated by the grammar in (1) can schematically be displayed as



where

$$\begin{aligned} A &:= \bullet(a, A) \cup a \cup \varepsilon \\ B &:= \bullet(b, B) \cup b \cup \varepsilon \\ D &:= \bullet(d, C) \cup c \cup \varepsilon \\ F &:= \bullet(f, D) \cup d \cup \varepsilon \end{aligned}$$

are recursively defined such that (A and C) and (B and D) are of the same height and always called in parallel, i.e., that whenever we choose a particular disjunct from the definition for A we choose the same disjunct for C, and the same holds for B and D as well. Therefore, the number of occurrences of a's and c's and of b's and d's, respectively, has to be the same. By taking the yield of the tree terms, we get the language $\mathcal{L}' = \{a^n b^m c^n d^m\}$.

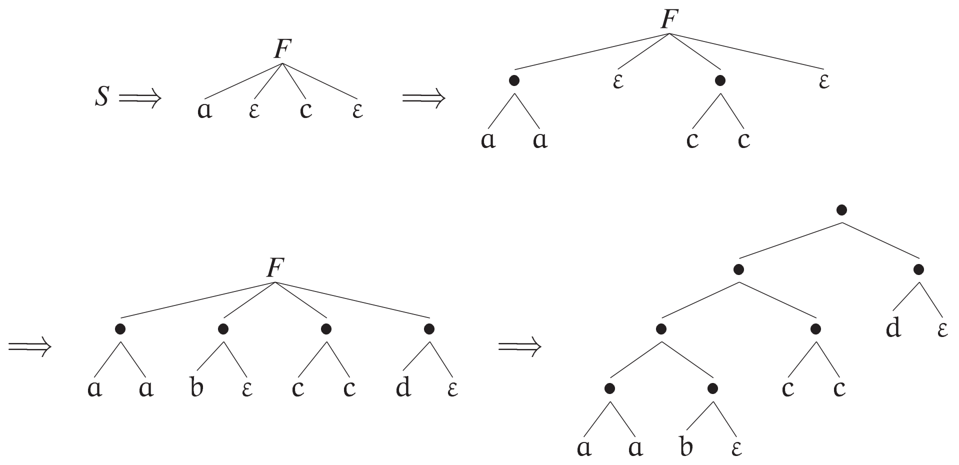


Figure 1: An example derivation of (1): $a^2b^1c^2d^1$

In Fig. 1 we show an example derivation of the string $aabccd$. It uses the second rule for S, followed by successive application of the first, second and third rule for F.

Any *context-free* tree grammar Γ can be transformed into a *regular* tree grammar Γ^L, which characterizes a (necessarily recognizable) set of trees encoding the instructions necessary to convert them by means of a unique homomorphism h into the ones the original grammar generates (Maibaum, 1974). This "LIFTing" is achieved by constructing a new, derived alphabet and translating the terms over the original signature into terms of the derived one via a primitive recursive procedure.

As already mentioned in the introduction, the formation of a derived alphabet can be explained as a nominalization process turning functions into objects. As a result, these latter saturated entities have to rely on an explicit composition operation that provides the necessary glue for combining the objectual counterpart of the original functional entity with its arguments. Since the objectual counterpart now belongs to the type of saturated entities, the substitution process on the symbolic level, when applied to the nominalized expressions resulting from the derivation of the original alphabet, can then take the familiar form of first-order substitution. As should be obvious from the preceding remarks and the intuitive description in the introductory section, the language generated by this first-order substitution procedure is context-free.

There is, however, a negative aspect of the approach using derived alphabets. During the translation of the original alphabet into its nominalized derived counterpart new operators representing functional composition – and, furthermore, new constants standing for projection functions which we have suppressed in our informal account – are introduced that change the configurational shape of the translated expressions.

Definition 8 (LIFT). Suppose that Σ is a ranked alphabet. The *derived* $\mathbb{N}$-sorted alphabet Σ^L is defined as follows: For each $n \geq 0$, $\Sigma'_n = \{f' | f \in \Sigma_n\}$ is a new set of symbols of sort n; for each $n \geq 1$ and each i, $1 \leq i \leq n$, π^n_i is a new symbol, the ith projection symbol of sort n; for each $n \geq 0, k \geq 0$ the new symbol $c_{n,k}$ is the (n,k)th composition symbol. The set of all $c_{n,k}$ will be denoted by $\mathbf{C}$.

$$\begin{aligned}
\Sigma^L_0 &= \Sigma'_0 \\
\Sigma^L_n &= \Sigma'_n \cup \{\pi^n_i | 1 \leq i \leq n\} \text{ for } n \geq 1 \\
\Sigma^L_{n,k} &= \{c_{n,k}\} \text{ for } n, k \geq 0 \\
\Sigma^L_i &= \emptyset \text{ otherwise}
\end{aligned}$$

For $k \geq 0$, $\text{LIFT}^\Sigma_k : T(\Sigma, \mathbf{X}_k) \rightarrow T(\Sigma^L, k)$ is defined as follows:

$$\begin{aligned}
\text{LIFT}^\Sigma_k(x_i) &= \pi^k_i \\
\text{LIFT}^\Sigma_k(f) &= c_{0,k}(f') \text{ for } f \in \Sigma_0 \\
\text{LIFT}^\Sigma_k(f(t_1, \dots, t_n)) &= \\
& c_{n,k}(f', \text{LIFT}^\Sigma_k(t_1), \dots, \text{LIFT}^\Sigma_k(t_n)) \\
& \text{for } f \in \Sigma_n, n \geq 1
\end{aligned}$$

Note, that this very general procedure allows the translation of any term over the original signature. Obviously, a rule of a grammar Γ is just another term over Σ^{Γ}, but so is, e.g., any structure *generated* by Γ.

In Fig. 2 we show an example of a tree and its LIFT-ed counterpart. In the lifted tree, we omitted the k's from the composition symbols $c_{n,k}$ since we can safely ignore the variables for this example.

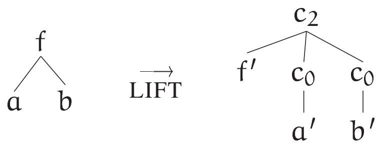


Figure 2: A tree and its lifted counterpart

As a more complicated example, we present the LIFT-ed version of the context-free tree grammar given in (1). The translation process for grammars has at its heart the LIFT-morphism for the translation of the alphabets of the operatives and inoperatives and the RHSs of the production rules. Since the rest of the translation is trivially observable, we dispense with a formal definition. Note that for better readability, we omit the π_i^2 from Σ_2^L, all the 0- and 1-place composition symbols and the subscripts on all the other composition symbols.

(2)
$$\begin{aligned}
\Sigma_0^L &= \{\varepsilon, a', b', c', d', S'\} \\
\Sigma_2^L &= \{\bullet'\} \\
\Sigma_4^L &= \{F'\} \cup \{\pi_1^4, \pi_2^4, \pi_3^4, \pi_4^4\} \\
\Sigma_{n,k}^L &= \{c\} \text{ for simplicity} \\
F_0 = \{\, S', F' \,\} \qquad S &= S' \qquad \mathbf{X} = \emptyset
\end{aligned}$$

$$\mathbf{P} = \left\{ \begin{array}{lll}
S' & \longrightarrow & \varepsilon \\
S' & \longrightarrow & c(F', a', \varepsilon, c', \varepsilon) \\
S' & \longrightarrow & c(F', \varepsilon, b', \varepsilon, d') \\
F' & \longrightarrow & c(F', c(\bullet', a', \pi_1^4), \pi_2^4, \\
 & & \qquad c(\bullet', c', \pi_3^4), \pi_4^4) \\
F' & \longrightarrow & c(F', \pi_1^4, c(\bullet', b', \pi_2^4), \\
 & & \qquad \pi_3^4, c(\bullet', d', \pi_4^4)) \\
F' & \longrightarrow & c(\bullet', c(\bullet', c(\bullet', \\
 & & \qquad \pi_1^4, \pi_2^4), \pi_3^4), \pi_4^4)
\end{array} \right\}$$

It is left as an exercise for the reader to parallel the derivation for $aabccd$ shown in Fig. 1.

It has to be admitted that the use of macro-like productions is not the only device that has been employed for the purpose of providing grammar formalisms with a controlled increase of generative capacity. Alternative systems that were developed for the same purpose are e.g. tree adjoining grammars, head grammars and linear indexed grammars (Vijay-Shanker and Weir, 1994). Although these systems make highly restrictive claims about natural language structure their predictive power is closely tied to the individual strategy they exploit to extend the context-free paradigm. The great advantage of the tree oriented formalism derives from its connection with *descriptive complexity theory*. Tree properties can be classified according to the complexity of logical formulas expressing them. This leads to the most perspicuous and fully *grammar independent* characterization of tree families by monadic second-order logic (Kolb et al., 1999). Although this characterization encompasses only regular tree sets, the lifting process of the preceding section allows us to simulate the effect of macro-like productions with regular rewrite rules.

2.2 Automata

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 then 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.

Definition 9 (Bottom-Up Tree Automaton). A (deterministic) bottom-up tree automaton $\mathfrak{A}$ is a fivetuple $\langle A, \Sigma, \delta, a_0, F\rangle$ 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 $\delta : (A^n \times \Sigma_n) \to A$ the transition function.

We can extend the transition function inductively to trees by defining $h_\delta(\varepsilon) = a_0$ and $h_\delta(\sigma(t_1, \ldots, t_n)) = \delta(h_\delta(t_1), \ldots, h_\delta(t_n), \sigma)$, $t_i \in T_\Sigma, 1 \leq i \leq n, \sigma \in \Sigma_n$. 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, and the automaton constructions which witness 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 non-deterministic automaton, but, again as for FSA's, bottom-up tree automata can be made deterministic by a straightforward generalization of the subset construction. (Note that *top-down* tree automata do not have this property: deterministic top-down tree automata recognize a strictly narrower family of tree sets.) 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 automata: *tree-walking automata with tests*. Intuitively, those automata – which are a variation of the tree-walking automata introduced in Aho and Ullman (1971) – make transitions from nodes in a tree to other nodes along its branches. They can read the labels of the nodes they are on and recognize which child they are of their parent. Each single transition can either test a label of a node, move up in the tree or down to a specific daughter.

Definition 10 (Tree-Walking Automaton).
A *tree-walking automaton (with tests)* over some ranked alphabet Σ is a finite automaton $\mathfrak{A} = (Q, \Delta, \delta, I, F)$ with states Q, directives Δ, transitions $\delta : Q \times \Delta \to Q$ and the initial and final states $I \subseteq Q$ and $F \subseteq Q$ which traverses a tree along connected edges using three kinds of directives: $\uparrow_i$ – "move up to the mother of the current node (if it has one and it is its i-th daughter)", $\downarrow_i$ – "move to the i-th daughter of the current node (if it exists)", and $\varphi(x)$ – "verify that φ holds at the current node".

For any tree t, such an automaton $\mathfrak{A}$ computes a node relation

(3) $$R_t(\mathfrak{A}) = \{(x, y) \mid (x, q_i) \stackrel{*}{\Longrightarrow} (y, q_f) \text{ for some } q_i \in I \text{ and some } q_f \in F\}$$

where for all states $q_i, q_j \in Q$ and nodes x, y in t

$$(x, q_i) \Longrightarrow (y, q_j) \text{ iff } \exists d \in \Delta : (q_i, d, q_j) \in \delta \text{ and } y \text{ is reachable from } x \text{ in } t \text{ via } d.$$

Note that x is reachable from itself if the directive was a (successful) test. It is important not to confuse this relation with the *walking language* recognized by the automaton, i.e., the string of directives needed to come from the initial to the final node in a path.

If all the tests $\varphi(x)$ of $\mathfrak{A}$ are definable in monadic second order logic (MSO), $\mathfrak{A}$ specifies a *regular tree-node relation*. Bloem and Engelfriet (1997), who should be consulted for details, prove that any regular tree-node relation is itself MSO definable and provide a general translation of $\mathfrak{A}$ into an MSO formula. We, however, will not exploit the full power of the definition: a *basic* tree-walking automaton restricting φ to simple tests of node labels (which are trivially MSO definable) is sufficient for our purposes.

2.3 Definability

Recall that the definability in MSO logic is tightly connected to the intertranslatability to some class of automata: MSO logic on trees to tree automata (as shown in Thatcher and Wright (1968) or Doner (1970)) and binary relations on nodes in a tree via tree-walking automata with MSO tests (Bloem and Engelfriet, 1997). So our presentation in terms of regular devices is tightly interwoven with questions of definability within MSO logic.

So we call classes of structures definable if they can be captured using automata: a regular tree language is definable iff it is recognized by a bottom-up tree automaton and a binary node relation is definable iff it can be computed by a tree-walking automaton with MSO tests.

Then the result which gives rise to the fact that we can characterize a non context-free set with two devices which have only context-free and regular power respectively is stated in Courcelle (1997). Viewing the binary relation defined by a tree-walking automaton as a definable transduction, we will make use of the following theorem:

Theorem 1 (Courcelle, Prop. 5.5.8). *The image of an MSO-definable class of structures under a definable transduction is not MSO definable in general.*

3 Verb raising: two routes and a detour

As mentioned in the introduction, this exercise in formal coding is made necessary by the fact that natural language sports at least some constructions which lead to (i) non-CF string languages, or (ii) to non-CF, i.e., non-recognizable structures, even though the resulting string languages are formally context-free. 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; Huybregts, 1984):

(4) a. *(... weil) der Karl die Maria dem Peter den Hans schwimmen lehren helfen lässt*
(German: Palindrome language – CF)

b. *(... omdat) Karel Marie Piet Jan laat helpen leren zwemmen*
(Dutch: $a^n b^n$ – CF)

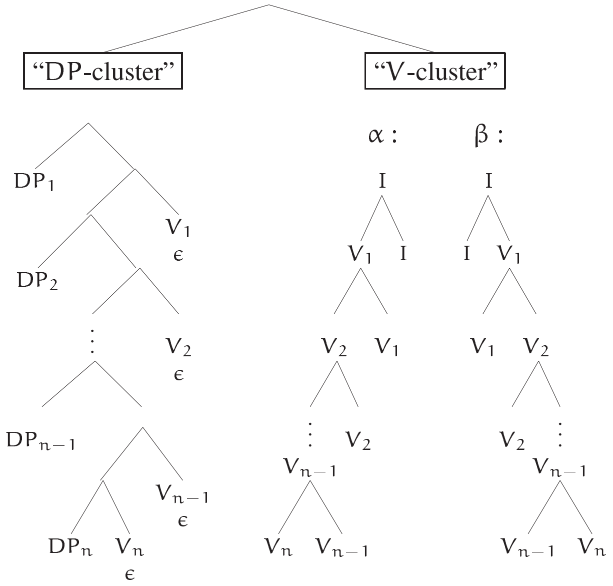
c. *(... wil) de Karl d'Maria em Peter de Hans laat hälffe lärne schwüme*
(Züritüütsch: ww – Non-CF)

'(... because) Charles lets Mary help Peter to teach John to swim'

The structure of the preceding example illustrates what we encountered in the artificial example in (1). 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. *de Karl* is the object of *laat*, *d'Maria* is the object of *hälffe*, *em Peter* is the object of *lärne*, and *de Hans* is the object of *schwüme*. Furthermore the verbs mark their objects for case. 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 (Huybregts, 1984, for Dutch) and (Shieber, 1985, for Swiss German) were able to show that not only the fragment exhibiting the cross-serial dependencies but the whole of the languages has to be assumed as non context-free.

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 a bipartite structure like the one in Fig. 3 with basically all DPs 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 conceived, there is no context-free device which could directly handle the unbounded



	overt CASE	V-cluster
German	+	α
Dutch	−	β
Züritüütsch	+	β

Figure 3: The structure of Germanic VR

number of non-local dependencies the structural separation of the two "clusters" enforces. A relatively simple transformational device as the minimalist MERGE-ATTRACT/MOVE machine, however, has little difficulty to combine nearness and separation in the required way. In a nutshell (and suppressing a plethora of important details), this machine is powered by just two operations: MERGE, combining two structures in a rather straightforward way creating a new node by "projecting" one of the input structures and installing both as its sole daughters, and ATTRACT/MOVE, which is dependent on the existence of a pair of attracting/attracted features in the structure so far constructed. It raises an element of the structure carrying the attracted feature and attaches it somewhere higher up in a *check*ing-configuration with the attracting feature, which is deleted in the process. Attracting features come in two flavors, *strong* and *weak*, inducing immediate or delayed movement, respectively. In our case, the moved element will always be adjoined to the element carrying the attracting feature. For details we refer the reader to Chomsky (1995), esp. ch. 4.

The following assumptions suffice for a(n overly simplified) treatment of our example construction, generally known as verb raising (VR):

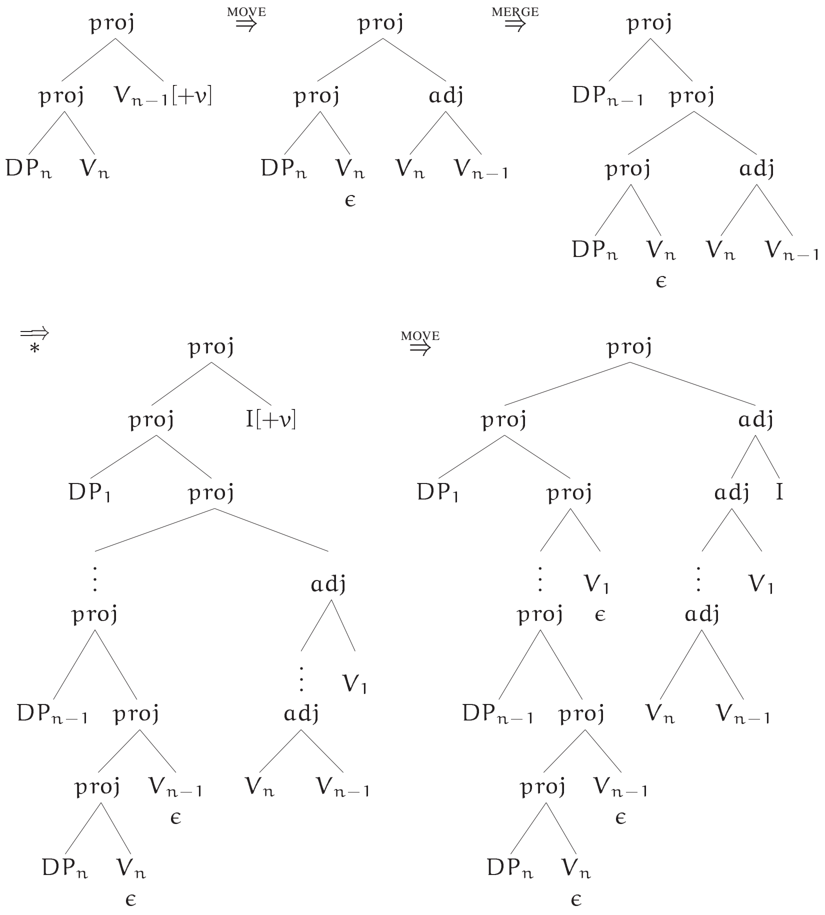


Figure 4: A "minimalist" derivation

- VR is a lexical property of a certain class of verbs: *lassen-laten-laa* ("let"), *helfen-helpen-hälffe* ("help") . . . but not *schwimmen-zwemmen-schwüme* ("swim");
- VR-verbs take VP-complements;
- VR-verbs (optionally) have a strong V-feature, like the one standardly postulated for the inflectional head I;
- PF serializes complex V_0/I_0 as head-last (German) or head-first (Züritüütsch / Dutch).

The minimalist MERGE-ATTRACT/MOVE-CHECK machine achieves the complex result by a sequence of very simple discrete steps and so VR comes as close to being a consequence of formal universals as a basically idiosyncratic phenomenon will ever be able to do. We cannot go into too much detail here, but refer the reader to Fig. 4 which shows a sample derivation. $\mathrm{proj}(x, y)$ and $\mathrm{adj}(x, y)$ are shorthands for Chomsky's (1995) *head-of*-$y(x, y)$ and [*head-of*-y, *head-of*-y](x, y), respectively. We assume that $\mathrm{adj}(x, y)$ is "spelled-out" xy in German and yx in Dutch and Züritüütsch.

While the structures derived by the minimalist machine clearly fall outside the limits of a standard context-free (string-)grammar, they can be captured by a *tree*-grammar.

Consider the following macro-tree grammar for the verb-raising construction (4), with the lexicon, the obvious rules for Σ_1, and any structure under Σ_1-nodes omitted for legibility:

(5) Σ_0 (the Lexicon)

$$\Sigma_1 = \{ V, V^{\epsilon}, V_r, V_r^{\epsilon}, DP, I \}$$
$$\Sigma_2 = \{ \mathrm{proj}, \mathrm{adj} \}$$
$$\mathsf{F}_0 = \{ \mathit{VR} \} \qquad \mathsf{F}_2 = \{ +V \}$$
$$S = \mathit{VR} \qquad \mathbf{X} = \{x, y\}$$

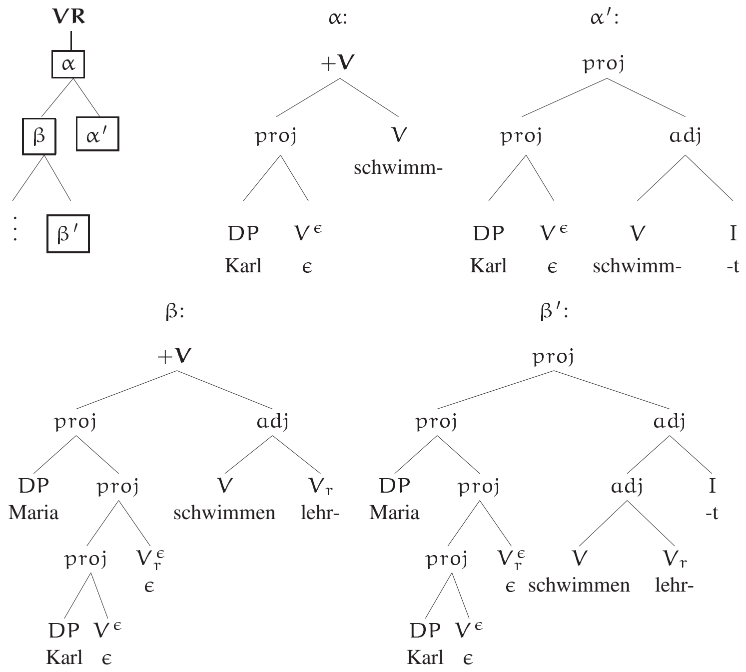


Figure 5: A (macro-)tree grammar derivation of VR

$$\mathbf{P} = \left\{ \begin{array}{rcl} \mathit{VR} & \longrightarrow & +V(\mathtt{proj}(\mathrm{DP}, \mathrm{V}^{\epsilon}), \mathrm{V}) \\ +V(x, y) & \longrightarrow & \mathtt{proj}(x, \mathtt{adj}(y, \mathrm{I})) \\ +V(x, y) & \longrightarrow & +V(\mathtt{proj}(\mathrm{DP}, \\ & & \quad \mathtt{proj}(x, \mathrm{V}^{\epsilon}_{r})), \\ & & \quad \mathtt{adj}(y, \mathrm{V}_{r})) \end{array} \right\}$$

The minimalist and the tree grammar generated structures are identical (cf. the sample derivation in Fig. 5).

As indicated in the introduction any *context-free* tree grammar Γ can be LIFTed – in the sense of Def. 8 – into a *regular* tree grammar Γ^{L} over a new, derived signature, which characterizes a (necessarily recognizable) set of trees encoding the instructions necessary to convert them into the ones the original grammar generates. Γ^{L} does not use variables at all, all nonterminals are null-ary and we have new alphabet symbols which can be interpreted as composition and projection "operations" by a tree homomorphism. So in a sense, Γ^{L} makes the compositions and projections involved in constructing the original trees $T(\Gamma)$ explicit.

Again simplifying things to the verge of inaccuracy – in particular, omitting all 0- and 1-place composition symbols, some book-keeping information on the composition symbols (the n, k), as well as rules and symbols for the lexicon and Σ^{L}_{1} – , Def. 8 yields the following LIFTed grammar for (5):

(6)
$$\begin{array}{rl} \Sigma^{L}_{0} & = \Sigma_{0} \\ \Sigma^{L}_{1} & = \{ \mathrm{V}', \mathrm{V}^{\epsilon\prime}, \mathrm{V}'_{r}, \mathrm{V}^{\epsilon_{r}\prime}, \mathrm{DP}', \mathrm{I}' \} \\ \Sigma^{L}_{2} & = \{ \mathtt{proj}', \mathtt{adj}', \pi_{1}, \pi_{2} \} \\ \Sigma^{L}_{n,k} & = \{ c_{n,k} \mid c \in \mathbf{C} \} \end{array}$$

$$F_{0} = \{ \mathit{VR}', +V' \} \qquad S = \mathit{VR}' \qquad \mathbf{X} = \emptyset$$

$$\mathbf{P} = \left\{ \begin{array}{rcl} \mathit{VR}' & \longrightarrow & c(+V', \\ & & \quad c(\mathtt{proj}', \mathrm{DP}', \mathrm{V}^{\epsilon\prime}), \mathrm{V}') \\ +V' & \longrightarrow & c(\mathtt{proj}', \pi_{1}, c(\mathtt{adj}', \pi_{2}, \mathrm{I}')) \\ +V' & \longrightarrow & c(+V', \\ & & \quad c(\mathtt{proj}', \mathrm{DP}', \\ & & \qquad c(\mathtt{proj}', \pi_{1}, \mathrm{V}^{\epsilon_{r}\prime})), \\ & & \quad c(\mathtt{adj}', \pi_{2}, \mathrm{V}'_{r})) \end{array} \right\}$$

plus the rules for Σ^{L}_{1}

The corresponding structures generated by this lifted version of the grammar are given in Fig. 6. These structures constitute different stages of derivations internal to the lifted grammar. Applying the unique homomorphism h to them returns the same value as would be obtained from applying the homomorphism h to the lifted counterparts in Fig. 5.

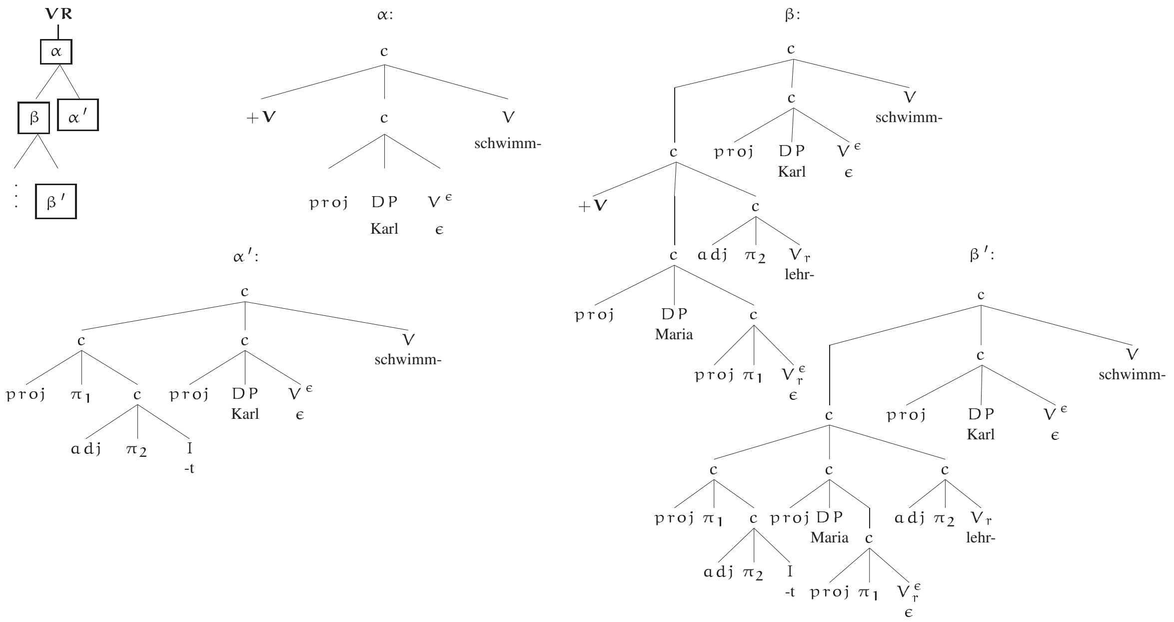


Figure 6: A sample derivation using the LIFTed grammar (6)

4 A Tree Automaton for VR

Since Γ^L in (6) generates a regular set of trees, we can construct a tree automaton to recognize this set.

Construction of a tree automaton from a given lifted context-free tree grammar $\Gamma^L = \langle \Sigma^L, F, S, \mathbf{X}, \mathbf{P} \rangle$, i.e., a regular tree grammar, is straightforward. Intuitively, since tree automata recognize only local trees in any one transition, we have to use auxiliary transitions for right hand sides (RHS) of macro productions with trees of depth greater than one to recognize the right trees incrementally. So what we are doing is to decompose the RHSs into trees of depth one which then can be recognized by a transition. This need is reflected both in the new transitions and in the states we need. In the following, we assume without loss of generality, that the trees of the RHSs of the macro productions are of depth one. If this is not the case they have to be transformed algorithmically by the introduction of extra rules and new nonterminals.

For the alphabet Σ of the automaton, we use the composition symbols of the lifted grammar, i.e., $\Sigma = \Sigma^L_{n,k}$. For the set of states Q, we need distinguishable states for each of the terminals, nonterminals and projection symbols appearing in RHSs of the rules, i.e., $Q = \{q_\sigma \mid \sigma \in \Sigma^L_i \cup F_0,\ 0 \leq i \leq 2\} \cup \{q_0\}$. Furthermore, we need a new initial state q_0. In the automaton, the state which corresponds to the start symbol *S* of the grammar becomes the final state, i.e., $F = \{q_S\}$.

Since our tree automata work bottom up, we have to start the processing at the bottom by having transitions from the new initial state to a new state encoding that we read a particular symbol on the frontier of the tree. So, together with the transitions encoding the productions, we have to construct two kinds of transitions in δ:

- transitions from the initial state on all subtrees reading a terminal symbol σ, i.e., elements of Σ_i, $i \in \{0, 1, 2\}$, to the corresponding state; i.e., $q_0^i \times \sigma \rightarrow q_\sigma$;
- transitions recognizing the internal structure of the local trees appearing in RHSs, i.e., from the states corresponding to the terminal symbols to the non-terminal of the left hand side, i.e., for each lifted tree grammar production of depth one $\mathrm{NT} \longrightarrow c(d_1, \ldots, d_n)$ we need to construct a transition in the automaton which looks as follows: $q_{d_1} \times \cdots \times q_{d_n} \times c \rightarrow q_{\mathrm{NT}}$.

In the example, the lifted tree grammar does not have the desired normal form, but it is easy to see how to change the grammar. For example, the first production $VR' \longrightarrow c(+V', c(\mathrm{proj}', \mathrm{DP}', V^{\epsilon\prime}), V')$ is transformed into the following two new productions: $VR' \longrightarrow c(+V', C1, V')$ and $C1 \longrightarrow c(\mathrm{proj}', \mathrm{DP}', V^{\epsilon\prime})$ The full translation of the other productions into the normal form is left to the reader. Accordingly, the tree automaton corresponding to the regular tree grammar (6) looks as follows:

(7) $$\mathfrak{A}_{\Gamma^L} = \langle Q, \Sigma, \delta, q_0, \{q_{VR}\}\rangle$$
$$\delta : Q \times Q \times Q \times \Sigma \to Q$$

$$\begin{array}{lcl}
(q_0, q_0, q_0, \mathrm{proj}) & \to & q_{\mathrm{proj}} \\
(q_0, q_0, q_0, \mathrm{DP}) & \to & q_{\mathrm{DP}} \\
(q_0, q_0, q_0, V_\epsilon) & \to & q_{V_\epsilon} \\
(q_0, q_0, q_0, V) & \to & q_V \\
(q_0, q_0, q_0, \mathrm{adj}) & \to & q_{\mathrm{adj}} \\
(q_0, q_0, q_0, \pi_1) & \to & q_{\pi_1} \\
(q_0, q_0, q_0, \pi_2) & \to & q_{\pi_2} \\
(q_0, q_0, q_0, I) & \to & q_I \\
(q_0, q_0, q_0, V_r) & \to & q_{V_r} \\
(q_0, q_0, q_0, V_r^\epsilon) & \to & q_{V_r^\epsilon} \\
(q_{\mathrm{proj}}, q_{\mathrm{DP}}, q_{V_\epsilon}, c) & \to & q_{C_1} \\
(q_{+V}, q_{C_1}, q_V, c) & \to & q_{VR} \\
(q_{\mathrm{adj}}, q_{\pi_2}, q_I, c) & \to & q_{C_2} \\
(q_{\mathrm{proj}}, q_{\pi_1}, q_{C_2}, c) & \to & q_{+V} \\
(q_{\mathrm{adj}}, q_{\pi_2}, q_{V_r}, c) & \to & q_{C_3} \\
(q_{\mathrm{proj}}, q_{\pi_1}, q_{V_r^\epsilon}, c) & \to & q_{C_4} \\
(q_{\mathrm{proj}}, q_{\mathrm{DP}}, q_{C_4}, c) & \to & q_{C_5} \\
(q_{+V}, q_{C_5}, q_{C_3}, c) & \to & q_{+V}
\end{array}$$

Obviously the automaton recognizes the same set of trees, in a way exemplified in Fig. 7 by yet another rendering of tree α' from Fig. 6. The nodes of the tree are complex: in the top row we display the symbol from Σ which is read on making a transition and in the bottom row we present the state the automaton is in by making transitions from the frontier towards the root. Note that we omitted the initial transitions in the figure since they do not play a role here. Since the root is labeled with a final state, the tree is accepted by the automaton.

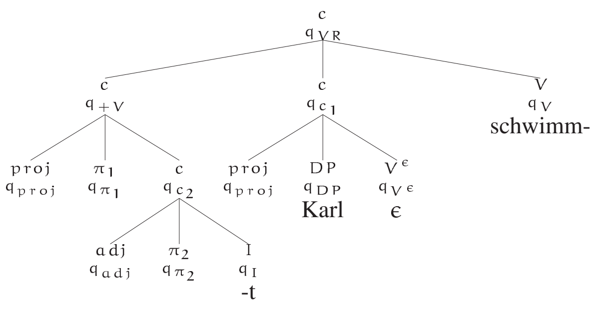


Figure 7: An example run of the tree automaton (7)

Since tree automata can also be used to generate trees, we can now produce candidates for further analysis.

5 A Tree-Walking Automaton for Linguistic Relations

Unfortunately, the terminal trees in Fig. 6, generated/recognized by (6/7), don't seem to have much in common with the structures linguists want to talk about, i.e., the ones in Fig. 5.

However, in some sense to be made operational, the Γ^L structures contain the intended structures. As mentioned before, there is a mapping h from these explicit structures onto structures interpreting the $c \in \mathbf{C}$ and the $\pi \in \Pi$ the way the names we have given them suggests, *viz.* as compositions and projections, respectively, which are, in fact, exactly the intended structures.

Put differently, it should be possible to define a set of relations $R^I = \{\blacktriangleleft, \blacktriangleleft^+, \blacktriangleleft^\star$ *(dominance)*, *c-command*, $\lhd$ *(precedence)*, $\dots\}$ holding between the nodes $n \in N^L$ of the explicit tree T^L which carry a *"linguistic"* label ℓ in such a way, that when interpreting $\blacktriangleleft^\star \in R^I$ as a tree order on the set of *"linguistic"* nodes and $\lhd \in R^I$ as the precedence relation on the resulting structure, $\langle\{n \mid n \in N^L \wedge \ell(n) \in \mathrm{LIFT}(\bigcup_{n\geq 0}\Sigma_n), \blacktriangleleft^\star, \lhd\rangle$ is in fact the intended tree corresponding to T^L. In what follows, we will confine ourselves to the dominance relation from which most linguistically relevant relations can be defined.

To do so, it is helpful to note a few general facts (illustrated in Fig. 8):

1. Our trees – that contain no substitutable elements from F_0 any more – feature three families of labels: the "linguistic" symbols, i.e., the lifted in-operatives of the underlying macro-grammar, $\mathbf{L} = \mathrm{LIFT}(\bigcup_{n\geq 0}\Sigma_n)$; the "composition" symbols $\mathbf{C} = \{c_{n,k}\}, n, k \geq 0$; and the "projection" symbols Π.

2. All non-terminal nodes in T^L are labeled by some $c \in \mathbf{C}$. This is due to the fact that the "composition" symbols are the only non-terminals of a LIFTed grammar. No terminal node is labeled by some c.

3. The terminal nodes in T^L are either labeled by some "linguistic" symbol or by some "projection" symbol $\pi_i \in \Pi$.

4. Any "linguistic" node dominating anything in the intended tree is on some left branch in T^L, i.e., it is the left-most daughter^L of some $c \in \mathbf{C}$. This lies in the nature of composition: $c(x_0, x_1, \cdots, x_n)$ evaluates to $x_0(x_1, \cdots, x_n)$.

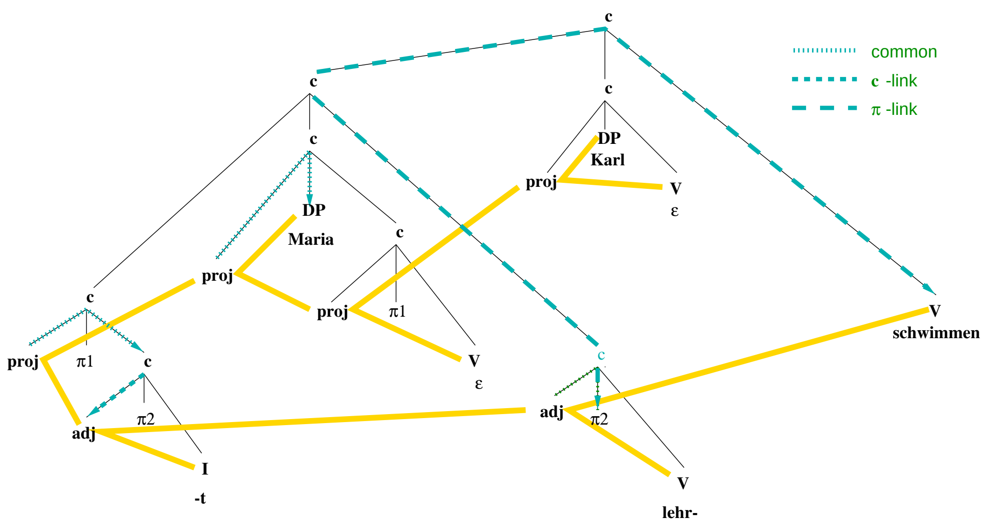


Figure 8: Intended relations on a LIFTed structure

5. For any node p labeled with some "projection" symbol $\pi_i \in \Pi$ in T^L there is a unique node n (labeled with some $c \in \mathbf{C}$ by (2.)) which properly dominates p and whose i-th sister will eventually evaluate to the value of π. Moreover, n will be the first node properly dominating p which is on a left branch. This crucial fact is arrived at by easy induction on the construction of Γ^L from Γ.

Following Bloem and Engelfriet (1997), we will use a *(basic) tree-walking automaton with node-label tests* to specify the *intended* (immediate) dominance relation on LIFTed trees, thus showing in passing that it is a regular tree node relation.

To keep the automaton small and perspicuous, we consider the probably only linguistically relevant case with ternary (binary) branching in the explicit (intended) trees and macros with at most two parameters. The extension to the general case, however, is trivial, as long as branching is bounded by some $n < \omega$.

We define $\mathfrak{A}_{\blacktriangleleft} = (Q, \Delta, \delta, I, F)$ with states Q, directives Δ, transitions δ and the initial and final states $I \subseteq Q$ and $F \subseteq Q$, respectively, as follows:

(8)
$$Q = \{q_i, e, d, d_1, d_2, d_3, u, u_1, u_2, u_3, q_f\}$$
$$I = \{q_i\} \qquad F = \{q_f\}$$
$$\Delta = \{\downarrow_1, \downarrow_2, \downarrow_3, \uparrow_1, \uparrow_2, \uparrow_3, \mathbf{L}(x), \mathbf{C}(x), \Pi_1(x), \Pi_2(x)\}$$

$$\begin{array}{rl}
\delta = \{ & (q_i, \mathbf{L}(x), u), (u, \uparrow_1, d), \\
 & (d, \downarrow_2, e), (d, \downarrow_3, e), \\
\text{case } \mathbf{L}: & (e, \mathbf{L}(x), q_f), \\
\text{case } \mathbf{C}: & (e, \mathbf{C}(x), d_1), (d_1, \downarrow_1, e), \\
\text{case } \Pi_1: & (e, \Pi_1(x), u_1), (u_1, \uparrow_2, u_1), \\
 & (u_1, \uparrow_3, u_1), (u_1, \uparrow_1, d_2), \\
 & (d_2, \downarrow_2, e), \\
\text{case } \Pi_2: & (e, \Pi_2(x), u_2), (u_2, \uparrow_2, u_2), \\
 & (u_2, \uparrow_3, u_2), (u_2, \uparrow_1, d_3), \\
 & (d_3, \downarrow_3, e) \quad \}
\end{array}$$

The automaton works as follows: First of all, if we can read a label on the node we started in and if that node is the first (leftmost) daughter, we go to either one of its sisters. Then depending on the case we are in, we can again "read" the label and halt, or read a composition symbol and find the leftmost daughter, or read a projection symbol and find the corresponding node by walking upward until we are on a leftmost daughter and then finding the second or third sister respectively.

As indicated in Sec. 2:(3), $\mathfrak{A}_{\blacktriangleleft}$ specifies, for any tree t, the node relation

$$\begin{aligned}
\blacktriangleleft_t = \; & R_t(\mathfrak{A}_{\blacktriangleleft}) = \\
& \{(x, y) \mid \text{there is a successful run of } \mathfrak{A} \text{ on} \\
& t \text{ starting in an initial state at node} \\
& x \text{ and ending at node } y \text{ in a final one}\}.
\end{aligned}$$

If t is an explicit tree T^L recognized by $\mathfrak{A}_{\Gamma^L}$, $\blacktriangleleft_t$ is a representation of the intended tree T corresponding to T^L.

Obviously, this approach can be extended to the other members of the *dominance* family of relations: $\mathfrak{A}_{\blacktriangleleft^+}$ is just $\mathfrak{A}_{\blacktriangleleft}$ with the transition $(e, \mathbf{L}(x), u)$ added to δ, and $\mathfrak{A}_{\blacktriangleleft^\star}$ is $\mathfrak{A}_{\blacktriangleleft^+}$ with $F = \{u\}$, and q_{fin}, as well as any transition referring to it, removed from Q and δ, respectively.

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.

6 Conclusion

We have shown in the paper how to account for cross-serial dependencies by coupling a bottom-up finite-state tree automaton with a universal tree-walking automaton. The result is, of course, not restricted to cross-serial dependencies. Any type of structural relationship that is amenable to a formal analysis with the means of context-free tree grammars can be described according to the same regular procedure. The original context-free tree language is first translated into its explicit presentation. A corresponding tree automaton then isolates the explicit tree family within the realm of all possible finite trees on the LIFTed signature. The universal tree-walking automaton finally serves to reestablish the intended structural relations.

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 *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 data that can be accommodated within the framework of context-free tree grammars are amenable to a regular analysis followed by a fixed universal path description.

Comparing this statement of the result of the paper with the characterization of context-free graph languages by Engelfriet and van Oostrom (1996) mentioned in the introduction, we want to stress the point that our regular description of context-free tree languages does not provide a characterization of this language family in the technical understanding of an equivalence between context-free tree languages and languages defined by a regular tree language and a tree-walking automaton. As should be obvious, the class of languages obtained by varying the automaton component together with the regular language component is much larger than the context-free tree languages.

Regular string and tree languages possess the useful property that they are characterizable as model sets that satisfy specifications expressed by means of monadic second-order logic on strings and trees, respectively. This type of logic is a highly flexible medium in which configurational facts of trees and strings can be stated in a fairly perspicuous manner. As was mentioned toward the end of Sec. 2.1, this logical characterization is completely independent of the grammatical formalism and provides a declarative delineation of the respective language classes along the methodological principles of descriptive complexity theory. A sequel to this paper is in preparation in which we show that, based on the indicated equivalence between logical and grammatical characterizations of regular languages, a purely logical description of the phenomenon of cross-serial dependencies holds can be given along a similar two stage approach.

References

A. V. Aho and J. D. Ullman. 1971. Translations on a context-free grammar. *Information and Control*, 19:439–475.

Roderick Bloem and Joost Engelfriet. 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.

Noam Chomsky. 1985. *Barriers*. MIT Press, Cambridge, MA.

Noam Chomsky. 1995. *The Minimalist Program*. MIT Press, Cambridge, MA.

B. Courcelle. 1997. The expression of graph properties and graph transformations in monadic second-order logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, pages 313–400. World Scientific, Singapore.

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

Joost Engelfriet and E. M. Schmidt. 1977. IO and OI, part I. *J. Comput. System Sci.*, 15:328–353.

Joost Engelfriet and E. M. Schmidt. 1978. IO and OI, part II. *J. Comput. System Sci.*, 16:67–99.

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

Michael J. Fischer. 1968. Grammars with macro-like productions. In *Proceedings of the 9th Annual Symposium on Switching and Automata Theory*, pages 131–142. IEEE.

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

Ferenc Gécseg and Magnus Steinby. 1984. *Tree Automata*. Akadémiai Kiadó, Budapest.

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

Marinus A. C. Huybregts. 1984. The weak adequacy of context-free phrase structure grammar. In Ger J. de Haan, Mieke Trommelen, and Wim Zonneveld, editors, *Van periferie naar kern*, pages 81–99. Foris, Dordrecht.

Hans-Peter Kolb and Uwe Mönnich, editors. 1999. *The Mathematics of Syntactic Structure: Trees and Their Logics*. Mouton-de Gruyter, Berlin.

Hans-Peter Kolb, Uwe Mönnich, and Frank Morawietz. 1999. Logical description of cross-serial dependencies. Ms., SfS, Tübingen University, accepted for presentation at the workshop *Finite Model Theory and its Applications* at FLoC'99.

Hans-Peter Kolb. 1999. Macros for Minimalism? Towards weak descriptions for strong structures. In Kolb and Mönnich (Kolb and Mönnich, 1999), pages 231–258.

T. S. E. Maibaum. 1974. A generalized approach to formal languages. *J. Comput. System Sci.*, 8:409–439.

J. Mezei and J. B. Wright. 1967. Algebraic automata and contextfree sets. *Inform. and Control*, 11:3–29.

Uwe Mönnich. 1999. On cloning contextfreeness. In Kolb and Mönnich (Kolb and Mönnich, 1999), pages 195–229.

P. Stanley Peters and Robert W. Ritchie. 1973. On the generative power of transformational grammars. *Information Sciences*, 6:49–83.

William C. Rounds. 1970. Tree-oriented proofs of some theorems on context-free and indexed languages. In *Proceedings of the 2nd Annual ACM Symposium on Theory of Computing*, pages 109–116.

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

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

K. Vijay-Shanker and D. J. Weir. 1994. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27(6):511–546.