Descriptions Of Cross-Serial Dependencies

Hans-Peter Kolb, Uwe Mönnich and Frank Morawietz*
Universität Tübingen
Seminar für Sprachwissenschaft
Wilhelmstr. 113,
72074 Tübingen

Abstract. The main result of this paper is a description of cross-serial dependencies equivalently in terms of (a) regular tree languages (to express the recursive properties) and regular string languages (to establish the intended linguistic relations) and (b) transductions definable in monadic second-order (MSO) logic which are defined on a domain of finite trees that is characterized as the model set of a closed MSO formula. In fact, we shall not deal directly with the two types of regular language families and MSO definable structures, but shall instead consider implementation in the form of finite-state (tree) automata to emphasize the effectivity of our approach.

Keywords: finite state machines, mild context-sensitivity, MSO logic, natural language complexity, tree grammars

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 results of this paper are twofold. The first one 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 other one is a logical description of cross-serial dependencies in terms of Monadic Second Order (MSO) logic. The description says that structures of this kind are specifiable as MSO definable transductions where these are defined on a domain of finite trees that is characterized as the model set of a (closed) MSO formula.

For regular string and tree languages, classical results in the descriptive theory of recognizability have established a tight connection

* Corresponding author is Frank Morawietz (`frank@sfs.nphil.uni-tuebingen.de`).

between logical formalisms and language classes by providing translation procedures that transform logical specifications into finite automata equivalent to the language classes and vice versa. To be a little more specific, Büchi (1960) and Elgot (1961) have shown that regular string languages represented through finite (string) automata can be expressed by sentences in the weak monadic second-order logic with one successor. For tree languages an analogous result is well known to the effect that a tree language is definable in weak monadic second-order logic with multiple successors if and only if it is recognizable by a finite tree automaton (Doner, 1970; Thatcher and Wright, 1968).

It is these earlier characterizations that provide the formal foundations for a renewed interest in logical approaches to grammar specifications. The main open question in this area of research is whether an appropriate extension of the MSO language can be found which is expressive enough to define significant properties of natural languages without becoming too unwieldy from the perspective of complexity theory.

The logical approach to the specification of language classes involves a lot of advantageous properties that have paved the way to its application to linguistic issues. First, the equivalence between automata theoretic operational and logic oriented declarative formalisms leads to a lot of closure properties of the defined language classes that follow immediately from the closure of the specification logics with respect to the traditional operations like negation, conjunction, alternation and (universal and existential) quantification. Second, the transition from strings to finite model-theoretic structures of arbitrary signatures requires no extra conceptual or technical ideas in the logical framework whereas in formal language theory the step from string to tree languages and the concomitant distinction between weak and strong generative capacity constitutes a significant extension of the research agenda. Third, since the logical approach does not depend on an operational process which, starting from some given objects, generates its space of interpretation, but refers directly to an assumed universe of structures, its statements can be phrased in terms of linguistically significant notions that enter into universal principles and language-particular constraints. Finally, those logical languages that capture complexity classes indicate lower bounds on the computing resources a system has to make available that wants to use those structures that fall within the classes correlated with the corresponding logical language.

The structure of the paper is as follows. After some technical preliminaries 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 substitutability 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 of MSO specifications and regular tree grammars/tree automata in constructing a closed formula 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 definable by an MSO formula that is modeled by the regular set of finite strings recognized by a tree-walking automaton which connect the tree nodes standing in those linguistically significant relations. The actual formal definition will take the form of an MSO definable transduction translating the lifted structures into the intended ones.

The basic idea underlying this sequential application of two finite-state devices or two logical specification steps respectively 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).

Similarly, in Courcelle (1992), a certain type of context-free graph languages is characterized as an equational subset of an algebra over a binary glueing operation and a family of definable unary operations. As is well known, a similar result holds for context-free tree languages, the relevant operations being in this case composition and projection. In contrast to the approach favored by Courcelle, we do not evaluate these operations, but interpret the intended linguistic relations directly on the structures given as elements of the free term algebra.

We think that our approach has two 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 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.

2. Preliminaries

The purpose of this section is to fix notations and to present definitions for the basic notions related to tree grammars, the classes of automata and MSO logic 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. Universal Algebra

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*. For such a ranked alphabet Σ, we denote by $T(\Sigma)$ the set of *trees* over Σ. $T(\Sigma)$ is inductively defined with base case $\Sigma_0 \subseteq T(\Sigma)$ and recursive step $f(t_1, \dots, t_n) \in T(\Sigma)$ if $f \in \Sigma_n$ and $t_i \in T(\Sigma)$ for $i = 1, \dots, n$.

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 operations on an algebra.

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.

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: $h(f_{\mathbb{A}}(a_1, \dots, a_n)) = f_{\mathbb{B}}(h(a_1), \dots, h(a_n))$ for every n-tuple $(a_1, \dots, a_n) \in A^n$.

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 $(t_1, \dots, t_n)$ in $T(\Sigma, X)^n$: $f_{\mathbb{T}(\Sigma,X)}(t_1, \dots, t_n) = f(t_1, \dots, t_n)$.

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 right hand side indicates the result of substituting t_i for x_i in t.

2.2. Tree Grammars

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.

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 null-ary 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 1. (Tree Grammar) A *tree grammar* $\Gamma = \langle \Sigma, \mathsf{F}, S, \mathsf{X}, \mathsf{P} \rangle$ is a 5-tuple, where Σ, F are the ranked alphabets of *inoperatives* and

operatives, respectively; $S \in \mathsf{F}$ is the starting symbol; X is a countable set of variables; and P is a set of productions. The $p \in \mathsf{P}$ have the form $F(x_i, \cdots, x_n) \longrightarrow t$, with $F \in \mathsf{F}$, $x_i, \cdots, x_n \in \mathsf{X}$, and t a term (tree) over $\Sigma \cup \mathsf{F} \cup \{x_i, \cdots, x_n\}$.
An application of a rule $F(x_i, \ldots, x_n) \rightarrow t$ "rewrites" a tree rooted in F as the tree t with its respective variables substituted by F's daughters.

Tree grammars with $\mathsf{F}_n = \emptyset, n \neq 0$, 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 $\mathsf{F}_n, n \neq 0$, 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 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. Accordingly, 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 2. Let $\Gamma = \langle \Sigma, \mathsf{F}, S, \mathsf{X}, \mathsf{P} \rangle$ be a context-free tree grammar and let $t, t' \in T(\Sigma \cup \mathsf{F})$. t' is *directly derivable* by an *inside-out step* from t ($t \Longrightarrow t'$) if there is a tree $t_0 \in T(\Sigma \cup \mathsf{F}, X_1)$ containing exactly *one* occurrence of x_1, a corresponding rule $F(x_1, \ldots, x_m) \rightarrow t''$, and trees $t_1, \ldots, t_m \in T(\Sigma)$ such that $t = t_0[F(t_1, \ldots, t_m)]$ and $t' = t_0[t''[t_1, \ldots, t_m]]$. 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 $\overset{*}{\Longrightarrow}$ denotes the transitive-reflexive closure of $\Longrightarrow$.

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

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.

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

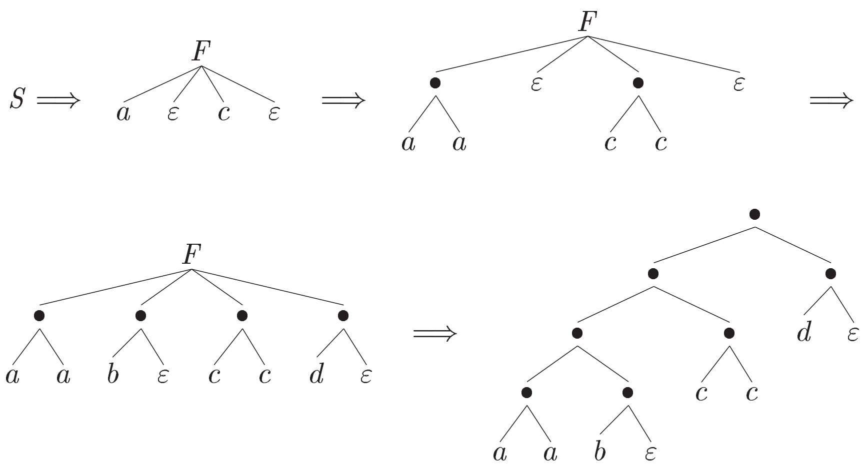


Figure 1. An example derivation of (1): $a^2b^1c^2d^1$

$a^nb^mc^nd^m$ 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.

$$\begin{array}{lll} \Sigma_0 = \{\varepsilon, a, b, c, d\} & & \Sigma_2 = \{\,\bullet\,\} \\ \mathsf{X} = \{x_1, x_2, x_3, x_4\} & \mathsf{F}_0 = \{\ S\ \} & \mathsf{F}_4 = \{\ F\ \} \end{array} \tag{1}$$

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

The tree language generated by the grammar in (1) can intuitively be described as a parallel derivation of a's and c's and b's and d's. 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 $\mathcal{L}' = \{a^nb^mc^nd^m\}$.

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.

Definition 4. (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_i^n 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 C, the π_i by Π.

$$\Sigma_0^L = \Sigma'_0 \qquad \Sigma_n^L = \Sigma'_n \cup \{\pi_i^n | 1 \leq i \leq n\} \text{ for } n \geq 1$$
$$\Sigma_{n,k}^L = \{c_{n,k}\} \text{ for } n, k \geq 0 \qquad \Sigma_i^L = \emptyset \text{ otherwise}$$

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

$$\begin{aligned}
\text{LIFT}_k^\Sigma(x_i) &= \pi_i^k \\
\text{LIFT}_k^\Sigma(f) &= c_{0,k}(f') \text{ for } f \in \Sigma_0 \\
\text{LIFT}_k^\Sigma(f(t_1, \ldots, t_n)) &= c_{n,k}(f', \text{LIFT}_k^\Sigma(t_1), \ldots, \text{LIFT}_k^\Sigma(t_n)) \\
&\quad \text{for } f \in \Sigma_n, n \geq 1
\end{aligned}$$

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

As an example, we present the LIFTed version Γ^L of the context-free tree grammar Γ 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 other composition symbols.

$$\begin{aligned}
\Sigma_0^L &= \{\varepsilon, a', b', c', d', S'\} & \Sigma_2^L &= \{\bullet'\} \\
\Sigma_4^L &= \{F'\} \cup \{\pi_1^4, \pi_2^4, \pi_3^4, \pi_4^4\} & \Sigma_{n,k}^L &= \{c\} \text{ for simplicity} \\
\mathsf{F}_0 &= \{\ S', F'\ \} & \mathsf{X} &= \emptyset
\end{aligned} \tag{2}$$

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

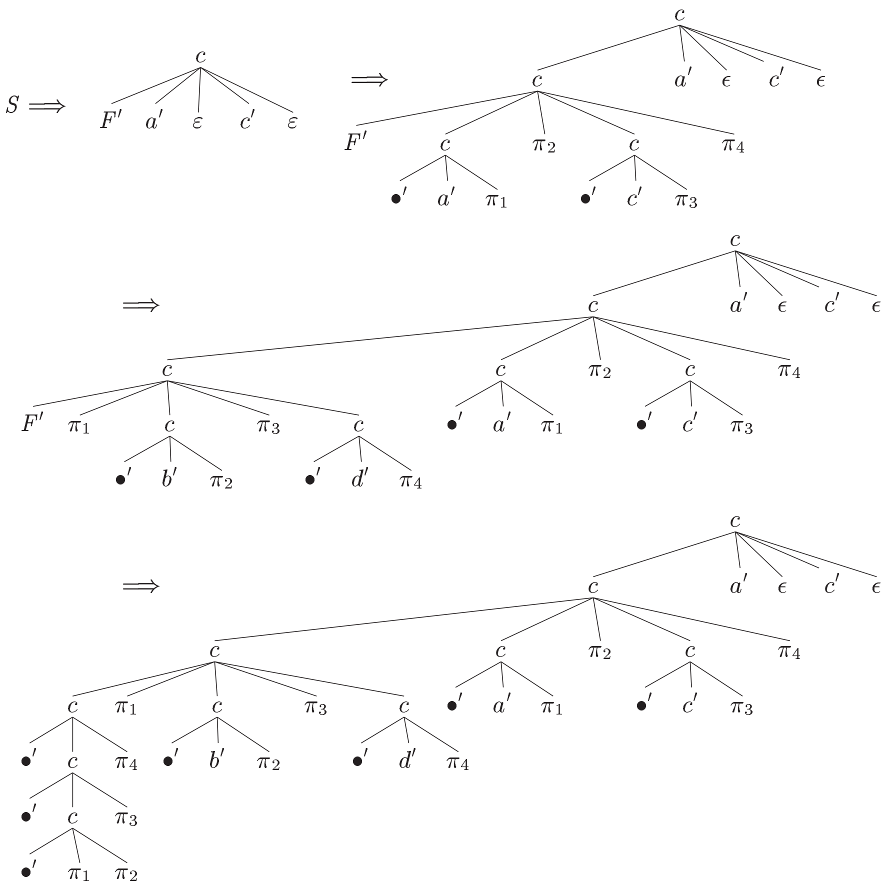


Figure 2. A sample derivation using the LIFTed grammar

We parallel the derivation for *aabccd* shown in Fig. 1 with this lifted grammar in Fig. 2.

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. 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.

2.3. MSO Logic

MSO logic is a straightforward extension of first-order logic to include variables that range over sets (i.e., monadic predicates) and quantifiers over these variables. The particular language we will use is Rogers' variant of monadic second-order predicate logic, $L^2_{K,P}$, with three disjoint, countable sets of *individual constants* $\mathbf{K}$, *monadic predicate constants* $\mathbf{P}$, and *individual and set valued variables* $\mathbf{X} = \mathbf{X}^0 \cup \mathbf{X}^1$; four binary predicates: *equality* $\approx$, plus the tree predicates *parent* $\triangleleft$, *dominance* $\triangleleft^*$, and *left-of* $\prec$, a symbol for set membership $\in$ and the usual connectives, quantifiers, brackets. As usual, the $k \in \mathbf{K}$ are interpreted as nodes, the $p \in \mathbf{P}$ as properties (or labels) of nodes, and the $x \in \mathbf{X}^0$ and $X \in \mathbf{X}^1$ range over nodes and sets of nodes, respectively. The syntax is the standard first order predicate logic syntax extended by quantification over monadic set variables: $\forall X \in \mathbf{X}^1$, if φ is a formula, so are $(\exists X)[\varphi]$ and $(\forall X)[\varphi]$.

No extra n-place predicates for $n > 1$ are allowed unless they are definable. Definability is a complex issue with MSO logics. We will make use of the fact that all explicitly definable relations and all relations which are definable by tree-walking automata (cf. Bloem and Engelfriet, 1997) are definable. Monadic predicates, however, are possible since they can be added freely to $\mathbf{P}$.

The intended models of $L^2_{K,P}$ are tree domains $\mathbb{T}$ under their their natural interpretation $\mathbb{T}^\natural$, where $\triangleleft$, $\triangleleft^*$, $\prec$ actually have their intuitive meaning, plus interpretations for the additional individual and predicate constants, i.e., just labeled trees (cf. Rogers, 1998, for details).

The following paragraphs go directly back to Courcelle (1997). Representation of objects with relational structures is a standard way to make 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 $r_\mathcal{R} \subseteq D_\mathcal{R}^{\rho(r)}$.

With the classical technique of interpretations where one relational structure is defined within another one we can overcome the limitation that, at first glance, there does not seem to be a convenient machine model for tree transformations. Fortunately, one can use logic directly to define the desired transduction. We will use this concept to transform the LIFTed trees into the intended ones.

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 $(\varphi, \psi, (\theta_q)_{q \in Q})$ consisting of a formula φ defining the domain of the transduction in $\mathcal{R}$, ψ defining the resulting domain of $\mathcal{Q}$, and a family of formulas θ_q defining the new relations 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 context-free and regular power respectively is stated in Courcelle (1997). Viewing the relation of intended dominance defined later by a tree-walking automaton as the cornerstone of an MSO transduction, our description of cross-serial dependencies is an instance of the following theorem:

Theorem 1. The image of an MSO-definable class of structures under a definable transduction is not MSO definable in general.

2.4. 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 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 $\delta : (A^n \times \Sigma_n) \to A$ 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 nondeterministic automaton, but, again as for FSA's, bottom-up tree automata can be made deterministic by a straightforward generalization of the subset construction.[1] 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 6. (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

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

where for all states $q_i, q_j \in Q$ and nodes x, y in t $(x, q_i) \Longrightarrow (y, q_j)$ iff $\exists d \in \Delta : (q_i, d, q_j) \in \delta$ and y is reachable from x in t via d.[2]

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 MSO logic, $\mathfrak{A}$ 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.

3. Linguistic Motivation: Verb Raising

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, 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*

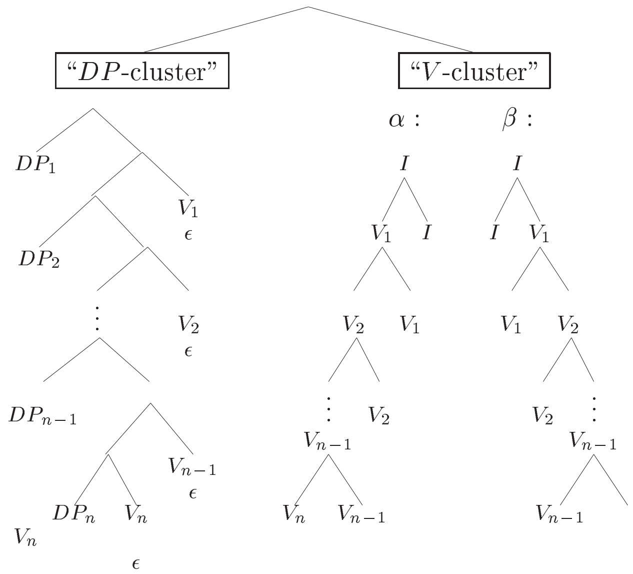
   (Züritüütsch: ww – Non-CF)

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

(... 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. This is observable since 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



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

Figure 3. The structure of Germanic VR

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 number of non-local dependencies the structural separation of the two "clusters" enforces. A relatively simple transformational device as the minimalist MERGE-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 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

$$\mathcal{MG} = \langle V, Cat, Lex, F \rangle$$

$$\begin{array}{rcl} V & = & P \cup I \\ Cat & = & \mathsf{base} \cup \mathsf{select} \cup \mathsf{licensee} \cup \mathsf{licensor} \\ Lex & = & \text{a finite set of expression trees} \\ F & = & \{\textsc{merge}, \textsc{move}\} \end{array}$$

$$\begin{array}{rl} P & \text{are the phonological features: } /p/ \\ I & \text{are the interpreted/semantical features: } |i| \end{array}$$

$$\begin{array}{rcl} \mathsf{base} & = & \{\mathtt{c}, \mathtt{t}, \mathtt{v}, \mathtt{d}, \mathtt{n}, \dots\} \\ \mathsf{select} & = & \{\mathtt{=x}, \mathtt{=X} \mid \mathtt{x} \in \mathsf{base}\} \\ \mathsf{licensee} & = & \{\mathtt{-case}, \mathtt{-wh}, \mathtt{case-}, \mathtt{wh-}, \dots\} \\ \mathsf{licensor} & = & \{\mathtt{+case}, \mathtt{+CASE}, \mathtt{+wh}, \mathtt{+WH}, \dots\} \end{array}$$

$$Label : Leaves \longrightarrow \mathsf{select}^*(\mathsf{licensor})\mathsf{select}^*\mathsf{licensee}^*(\mathsf{base})P^*I^*$$

$$\mathcal{L}_{\mathcal{MG}} = Closure(Lex, F)$$

Figure 4. Minimalist Grammars

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):

- 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-MOVE 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.

To be a little bit more precise, we will show briefly how one could generate the bipartite structures indicated in Fig. 3 with Minimalist Grammars (MG) as they were formalized in Stabler (1997), see Fig. 4. Basically an MG has a lexicon of lists of features which trigger the combination via MERGE and MOVE. The language defined by an MG is then just those trees generated by MERGE and MOVE which have only one category feature left. We omit the full definitions of the MGs, particularly of MERGE and MOVE, relying on the informal account given above. The interested reader is referred to the original literature for a precise definition.

For our example, we only specify the relevant lexicon and leave the sets of features implicit. In fact, the lexicon is fairly simple. We can express it schematically as given in (5). In this minimalist grammar, we do not use actual lexical elements, but rather the main categories as phonological material to facilitate the understanding of the dependencies. We need n different DPs and n different verbs, one of the form $/V_n/$ and $n-1$ of the form $/V_{n-i}/$, $1 \leq i \leq n-1$ and an inflectional element. There are two different entries for the verbs since we need a base case to start the creation of the verbal complex and a "recursive" step which is terminated with the inflectional head.

$$\begin{array}{rlrl} [\texttt{dp}] & /DP_n/ & [\texttt{=dp v v-}] & /V_n/ \\ [\texttt{=v +V i}] & /I/ & [\texttt{=v +V =dp v v-}] & /V_{n-i}/ \end{array} \qquad (5)$$

Note that we allow ourselves `x-` features indicating movement to the right. In Stabler's original system this type of head-movement was done within a special MERGE-step which did movement and merging of structures at the same time, triggered by `X+` features. We think that it is clearer to separate the two operations.

Note that we deliberately do not use Stabler's labels for the intermediate nodes ($<, >$), but rather use $proj(x, y)$ and $adj(x, y)$ as shorthands for Chomsky's $head\text{-}of\text{-}y(x, y)$ and $[head\text{-}of\text{-}y, head\text{-}of\text{-}y](x, y)$, respectively, to allow for the differences in word order. We assume that $adj(x, y)$ is "spelled-out" xy in German and yx in Dutch and Züritüütsch.

We cannot go into much detail here, but as one can see in Fig. 5, we skipped the initial merge of the lowermost verb with the lowermost DP and the next merge with the next highest verb. This tree has both a `v-` and a `+V` feature available which triggers a movement. We indicate the moved category with an appropriately indexed trace. The resulting structure then demands the merging of a DP which makes a `v` feature available which triggers the merging of the next verb. After finitely many steps we choose an inflectional node instead of a verb for merging. It triggers the last move and leads to the final result. Clearly, the result corresponds to the schematic representation choosen for cross-serial dependencies in Fig. 3.

We have shown in this section why MSO logic alone can not be sufficient for linguistic reasons. But in order to concentrate on the relevant details, we will use the artificial example from (1) in the following sections to illustrate our proposal.

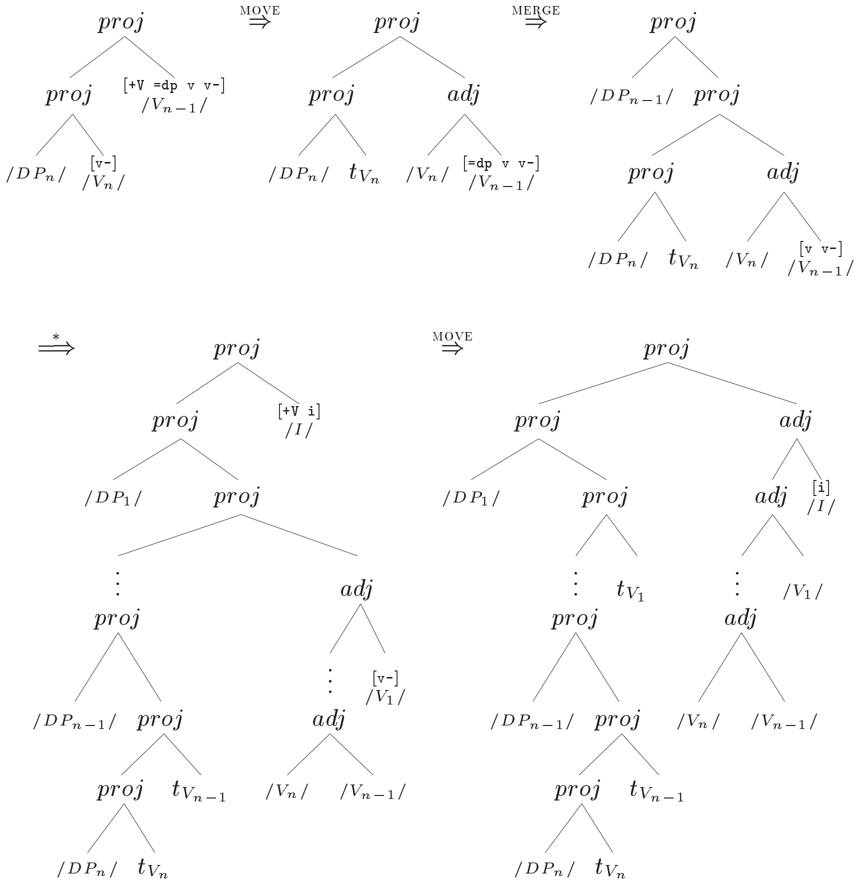


Figure 5. Part of a "minimalist" derivation

4. A Tree Automaton for Non-CF Tree Sets

Since Γ^L in (2) 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 $\Gamma^L = \langle \Sigma^L, \mathsf{F}, S, \mathsf{X}, \mathsf{P} \rangle$, 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 LIFTed 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 LIFTed 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 symbols of the lifted grammar, i.e., $\Sigma = \Sigma_i^L \cup \Sigma_{n,k}^L$. 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., $Q = \{q_\sigma \,|\, \sigma \in \Sigma_i^L \cup F_0,\, 0 \leq i \leq 2\} \cup \{q_0\}$. 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 all the Σ_i from Γ, to the corresponding state; i.e., $q_0^n \times \sigma \rightarrow q_\sigma$; n the maximal branching factor;
- transitions recognizing the internal structure of the local trees appearing in RHSs, i.e., from the states corresponding to the leaves to the non-terminal of the left hand side, i.e., for each lifted tree grammar production of depth one $NT \longrightarrow c(d_1, \ldots, d_n)$ we have to construct a transition in the automaton which looks as follows: $q_{d_1} \times \cdots \times q_{d_n} \times c \rightarrow q_{NT}$.

In the example, the lifted tree grammar does not have the desired normal form, but it is easy to see how to change it.[3] Accordingly, the tree automaton corresponding to the regular tree grammar Γ^L given in (2) looks as given in Fig. 6.

Obviously the automaton recognizes the same set of trees. The reader may have noticed that the transitions of $\mathfrak{A}_{\Gamma^L}$ are uniformly 5-ary whereas in the grammar Γ^L we had 3 to 5 daughters. This is due to the fact that the tree automaton starts in those cases from an initial state which does not correspond to a node in the LIFTed tree.

In (Thomas, 1990) tree automata are converted to formulas in MSO logic by basically encoding their behaviour. Under the assumption that $Q = \{0, \ldots, m\}$ with $q_0 = 0$, the (closed) formula given there adapted to our signature and for 5-ary tree automata looks as given in (6).[4]

Intuitively, the sets X_i label the tree where the 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 the last one says that we need a root node which is in a "final" set.

$$\mathfrak{A}_{\mathcal{G}'} = \langle Q, \Sigma, \delta, q_0, \{q_{S'}\}\rangle$$
$$\delta : Q \times Q \times Q \times Q \times Q \times \Sigma \to Q$$

$$\begin{array}{lll@{\qquad}lll}
(q_0,q_0,q_0,q_0,q_0,\epsilon) & \to & q_\epsilon & (q_0, q_0, q_{\bullet'}, q_{a'}, q_{\pi_1}, c) & \to & q_{c_a} \\
(q_0,q_0,q_0,q_0,q_0,a') & \to & q_{a'} & (q_0, q_0, q_{\bullet'}, q_{c'}, q_{\pi_3}, c) & \to & q_{c_c} \\
(q_0,q_0,q_0,q_0,q_0,b') & \to & q_{b'} & (q_0, q_0, q_{\bullet'}, q_{b'}, q_{\pi_2}, c) & \to & q_{c_b} \\
(q_0,q_0,q_0,q_0,q_0,c') & \to & q_{c'} & (q_0, q_0, q_{\bullet'}, q_{d'}, q_{\pi_4}, c) & \to & q_{c_d} \\
(q_0,q_0,q_0,q_0,q_0,d') & \to & q_{d'} & (q_0, q_0, q_{\bullet'}, q_{\pi_1}, q_{\pi_2}, c) & \to & q_{c_2} \\
(q_0,q_0,q_0,q_0,q_0,\pi_1) & \to & q_{\pi_1} & (q_0, q_0, q_{\bullet'}, q_{c_2}, q_{\pi_3}, c) & \to & q_{c_3} \\
(q_0,q_0,q_0,q_0,q_0,\pi_2) & \to & q_{\pi_2} & (q_0, q_0, q_{\bullet'}, q_{c_3}, q_{\pi_4}, c) & \to & q_{F'} \\
(q_0,q_0,q_0,q_0,q_0,\pi_3) & \to & q_{\pi_3} & (q_{F'}, q_{\pi_1}, q_{c_b}, q_{\pi_3}, q_{c_d}, c) & \to & q_{F'} \\
(q_0,q_0,q_0,q_0,q_0,\pi_4) & \to & q_{\pi_4} & (q_{F'}, q_{c_a}, q_{\pi_2}, q_{c_c}, q_{\pi_4}, c) & \to & q_{F'} \\
(q_0,q_0,q_0,q_0,q_0,\bullet') & \to & q_{\bullet'} & (q_{F'}, q_\epsilon, q_b, q_\epsilon, q_d, c) & \to & q_{S'} \\
(q_0,q_0,q_0,q_0,q_0,\epsilon) & \to & q_{S'} & (q_{F'}, q_a, q_\epsilon, q_c, q_\epsilon, c) & \to & q_{S'}
\end{array}$$

Figure 6. The tree automaton for Γ^L

$$\begin{aligned}
(\exists X_0, \ldots, X_m)[\bigwedge_{i \neq j} (\neg\exists y)[y \in X_i \wedge y \in X_j] \wedge (\forall x)[\mathsf{leaf}(x) \to x \in X_0] \\
\wedge (\forall x_1, \ldots, x_5, y)[\bigvee_{\substack{(i_1,\ldots,i_5,\sigma,j)\in\alpha \\ 1 \le k \le 5}} x_k \in X_{i_k} \wedge y \lhd x_k \wedge y \in X_j \wedge y \in P_\sigma] \\
\bigvee_{i \in F} (\exists x \forall y)[x \lhd^* y \wedge x \in X_i] \quad (6)
\end{aligned}$$

Since the actual automaton has 18 states, 11 alphabet symbols and 22 transitions, we refrain from giving its full, but tedious definition.

5. A Tree-Walking Automaton for Linguistic Relations

Unfortunately, the terminal trees in Fig. 2, generated/recognized by (1) or the tree automaton in Fig. 6), don't seem to have much in common with the structures linguists want to talk about, i.e., the ones in Fig. 1.

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 $c \in \mathrm{C}$ 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.

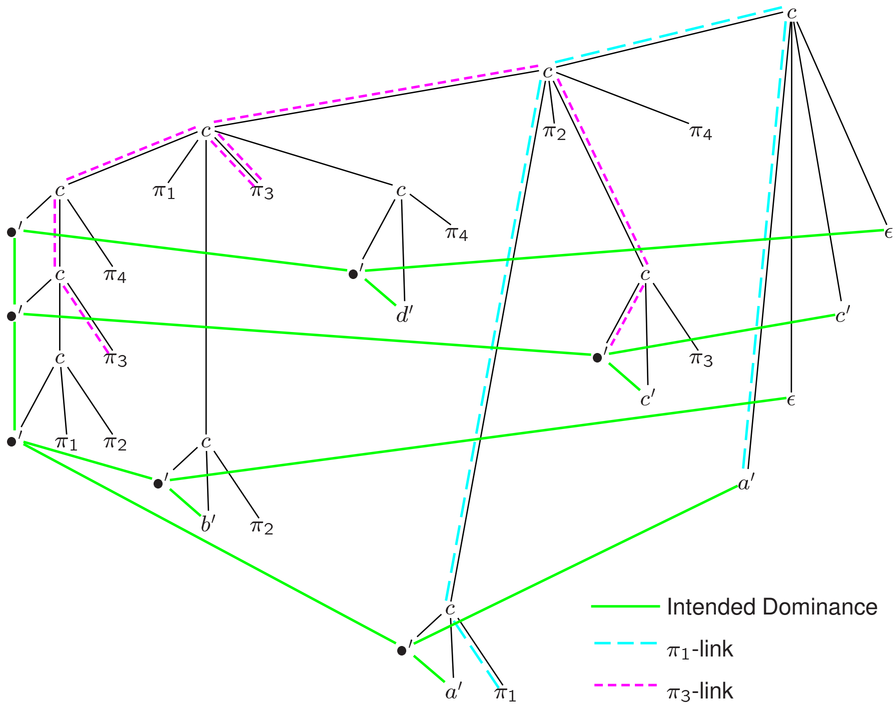


Figure 7. Intended relations on a LIFTed structure

Put differently, it should be possible to define a set of relations $R^I = \{\blacktriangleleft, \blacktriangleleft^+, \blacktriangleleft^\star$ *(dominance)*, *c-command*, $\vartriangleleft$ *(precedence)*, $\dots\}$ holding between the nodes $n \in N^L$ of the explicit or LIFTed 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 $\vartriangleleft \in R^I$ as the precedence relation on the resulting structure we have a "new" description language on the intended structures.

We will use an MSO definable tree transduction as defined in Sec. 2.3 to transform the LIFTed structures into the intended ones. The core of this transduction will be the definition of the new relations. 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. 7 with another rendering of the last tree of the derivation given in Fig. 2):

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, $\mathsf{L} = \text{LIFT}(\bigcup_{n\geq 0} \Sigma_n)$; the "composition" symbols $\mathsf{C} = \{c_{n,k}\}, n, k \geq 0$; and the "projection" symbols Π.

2. All non-terminal nodes in T^L are labeled by some $c \in \mathsf{C}$. 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 of some $c \in \mathsf{C}$. This lies in the nature of composition: $c(x_0, x_1, \cdots, x_n)$ evaluates to $x_0(x_1, \cdots, x_n)$.

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 $c \in \mathsf{C}$ 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.

By (4.) it is not hard to find possible dominees in any T^L. It is the problem of determining the actual "filler" of a candidate-dominee which makes up the complexity of the definition of $\blacktriangleleft$. There are three cases to account for:

6. If the node considered carries a "linguistic" label, it evaluates to itself;

7. if it has a "composition" label c, it evaluates to whatever its functor – by (4.) its leftmost daughter – evaluates to;

8. if it carries a "projection" label π_i, it evaluates to whatever the node it "points to" – by (5.) the i^{th} sister of the first C-node on a left branch dominating it – evaluates to. Two examples for π-links can be found in Fig. 7.

Note that cases (7.) and (8.) are inherently recursive.

In general, recursive definitions in MSO may lead to undecidability and are therefore disallowed. Fortunately, there are certain techniques to ensure that some relation R which would most naturally be defined recursively has a valid MSO definition. Still, special care has to be taken to establish that the relations defined are well-behaved in this respect. In our case this caveat applies to $\blacktriangleleft$ as well as to its reflexive transitive closure $\blacktriangleleft^\star$.

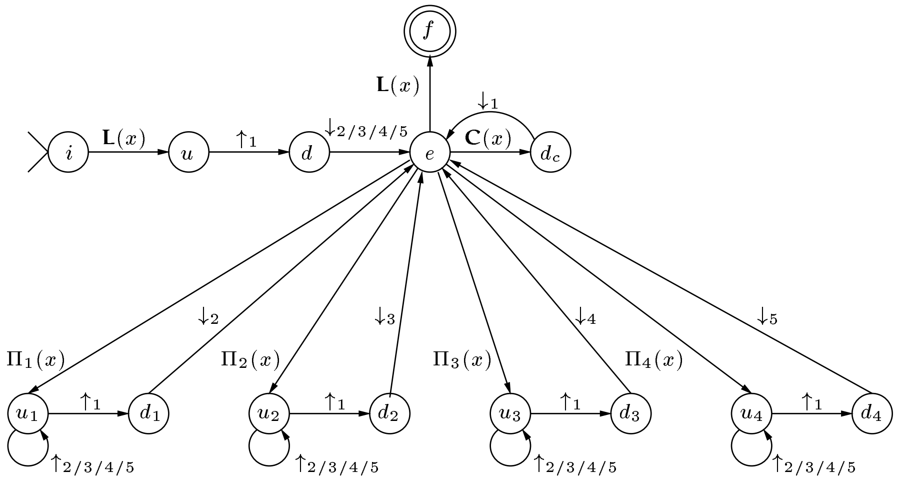


Figure 8. The tree-walking automaton for dominance on intended structures

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 four parameters. The extension to the general case, however, is trivial, as long as branching is bounded by some $n < \omega$.

We define $\mathfrak{A}_{\triangleleft} = (Q, \Delta, \delta, I, F)$ with states Q, directives Δ, transitions δ and the initial and final states $I \subseteq Q$ and $F \subseteq Q$, respectively, as given graphically in Fig. 8.

The automaton works as follows: First of all, if we can read a "linguistic" label on the node we started in and if that node is the first (leftmost) daughter, we go to one of its sisters. Then depending on the case we are in, we can again "read" a linguitic 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, third, forth or fifth sister respectively.

As indicated in Sec. 2, $\mathfrak{A}_{\triangleleft}$ specifies, for any tree t, the node relation

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

However, there is another interpretation of such an automaton. Viewed as an ordinary finite-state automaton over the alphabet Δ, $\mathfrak{A}$ recognizes a regular (string-) language, the *walking language* W.

In our case it is the walking-language $W_{\triangleleft} = \mathsf{L}(x) \cdot \uparrow_1 \cdot (\downarrow_2 \cup \downarrow_3 \cup \downarrow_4 \cup \downarrow_5) \cdot (W_{\mathsf{C}} \cup W_{\Pi_1} \cup W_{\Pi_2} \cup W_{\Pi_3} \cup W_{\Pi_4})^* \cdot \mathsf{L}(x)$ with

$$\begin{aligned} W_{\mathsf{C}} &= \mathsf{C}(x) \cdot \downarrow_1 \\ W_{\Pi_1} &= \Pi_1(x) \cdot (\uparrow_2 \cup \uparrow_3 \cup \uparrow_4 \cup \uparrow_5)^* \cdot \uparrow_1 \cdot \downarrow_2 \\ W_{\Pi_2} &= \Pi_2(x) \cdot (\uparrow_2 \cup \uparrow_3 \cup \uparrow_4 \cup \uparrow_5)^* \cdot \uparrow_1 \cdot \downarrow_3 \\ W_{\Pi_3} &= \Pi_3(x) \cdot (\uparrow_2 \cup \uparrow_3 \cup \uparrow_4 \cup \uparrow_5)^* \cdot \uparrow_1 \cdot \downarrow_4 \\ W_{\Pi_4} &= \Pi_4(x) \cdot (\uparrow_2 \cup \uparrow_3 \cup \uparrow_4 \cup \uparrow_5)^* \cdot \uparrow_1 \cdot \downarrow_5 \end{aligned} \tag{7}$$

which is finally translated into an MSO-formula $\mathsf{trans}_{W_{\triangleleft}}(x,y)$.[5] The rather tedious process, carried through in App. A, proceeds inductively via the following steps:

$$\begin{aligned} \mathsf{trans}_{\emptyset}(x,y) &\equiv \bot \\ \mathsf{trans}_{\downarrow_i}(x,y) &\equiv \mathsf{edg}_i(x,y) \\ \mathsf{trans}_{\uparrow_i}(x,y) &\equiv \mathsf{edg}_i(y,x) \\ \mathsf{trans}_{\varphi(x)}(x,y) &\equiv \varphi(x) \wedge x = y \\ \mathsf{trans}_{W_1 \cup W_2}(x,y) &\equiv \mathsf{trans}_{W_1}(x,y) \vee \mathsf{trans}_{W_2}(x,y) \\ \mathsf{trans}_{W_1 \cdot W_2}(x,y) &\equiv (\exists z)[\mathsf{trans}_{W_1}(x,z) \wedge \mathsf{trans}_{W_2}(z,y)] \\ \mathsf{trans}_{W^*}(x,y) &\equiv \mathsf{trans}^*_W(x,y) \\ \mathsf{trans}^*_W(x,y) &\equiv (\forall X)(\forall v,w)[(v \in X \wedge \mathsf{trans}_W(v,w) \to \\ &\qquad w \in X) \wedge x \in X \to y \in X] \end{aligned} \tag{8}$$

The resulting formula uses only the MSO-definable tests of the original automaton, the closed sets constructed via (10) for the Kleene-*-case, and the edg_n relations (here with $1 \leq n \leq 5$) defined in App. A. No recursion is involved: $\triangleleft$ is MSO- definable:

$$x \triangleleft y \overset{def}{\Longleftrightarrow} \mathsf{trans}_{W_{\triangleleft}}(x,y) \tag{9}$$

For the case of the recursion inherent in reflexive dominance, a standard solution exists, at least in *weak* MSO, the monadic second-order logic on *finite* structures. 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:

$$R\text{-}\mathsf{closed}(X) \overset{def}{\Longleftrightarrow} (\forall x,y)[x \in X \wedge R(x,y) \to y \in X] \tag{10}$$

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; $\blacktriangleleft^\star$ and $\blacktriangleleft^+$ can be defined as follows:

Reflexive Dominance:

$$x \blacktriangleleft^\star y \overset{def}{\Longleftrightarrow} (\forall X)[\blacktriangleleft\text{-}\mathsf{closed}(X) \wedge x \in X \rightarrow y \in X] \tag{11}$$

Proper dominance:

$$x \blacktriangleleft^+ y \overset{def}{\Longleftrightarrow} x \blacktriangleleft^\star y \wedge x \not\approx y$$

Using the defined formula for $\blacktriangleleft$ the specific MSO transduction we need to transform the LIFTed structures into the intended ones looks as follows:

$$\begin{aligned}
&(\varphi, \psi, (\theta_q)_{q \in Q}) \\
&Q = \{\blacktriangleleft, \blacktriangleleft^\star, \blacktriangleleft^+, \lhd, \dots\} \\
\varphi &\equiv \text{ the formula coding } \mathfrak{A}_{\Gamma L} \\
\psi &\equiv (\exists y)[x \blacktriangleleft y \vee y \blacktriangleleft x] \\
\theta_{\blacktriangleleft}(x,y) &\equiv \mathsf{trans}_{W_{\blacktriangleleft}}(x,y) \\
\theta_{\blacktriangleleft^\star}(x,y) &\equiv (\forall X)[\blacktriangleleft\text{-}\mathsf{closed}(X) \wedge x \in X \rightarrow y \in X] \\
\theta_{\blacktriangleleft^+}(x,y) &\equiv x \blacktriangleleft^\star y \vee x \not\approx y \\
\theta_{\lhd}(x,y) &\equiv \text{another tree-walking automaton} \\
\theta_{\text{labels}} &\equiv \text{taken over from } R
\end{aligned} \tag{12}$$

As desired, the domain of the transduction is characterized by the MSO formula for the LIFTed trees (see Sec. 4). 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 $L^2_{K,P}$. For reasons of space, we have to leave the specification of the precedence relation open. It is more complicated than dominance, but can be achieved with another tree-walking automaton.[6]

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.

6. Conclusion

We have shown in the paper how to account for cross-serial dependencies by coupling (a) a bottom-up finite-state tree automaton with a universal tree-walking automaton and (b) a logical domain specification followed by a logically definable transduction. 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 or logical procedure. The original context-free tree language is first translated into its explicit presentation. A corresponding tree automaton/closed MSO formula then isolates the explicit tree family within the realm of all possible finite trees on the LIFTed signature. The universal tree-walking automaton/MSO transduction 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/closed MSO formula and a tree-walking automaton/MSO transduction. As should be obvious, the class of languages obtained by varying the domain component together with the path component is much larger than the context-free tree languages.

One obviuos drawback of the approach, namely that there is no principled connection beteen context-free tree grammars and MGs is addressed in another paper. Using the result from Michaelis (1999), a similar technique has been applied to code minimalist grammars. First, MGs are translated into multiple context-free grammars. These can be viewed as Lawvere theories such that we get, again, lifted structures which can be recognized with regular tree grammars/tree automata/MSO formulas. Those can be transduced into the intended structures with a variant of the tree-walking automaton used here. We get another two step approach showing a direct connection between MGs and the proposal here (Michaelis et al., 2000).

Acknowledgements

The research presented in this paper was supported by the *Deutsche Forschungsgemeinschaft* within the *Sonderforschungsbereich* 340, TP A8. The authors wish to thank Tom Cornell and Jens Michaelis for discussion and advice.

Appendix

A. The MSO Definition of Intended Dominance

The translation via (8) of the walking language $W_{\blacktriangleleft}$ derived from the tree-walking automaton $\mathfrak{A}_{\blacktriangleleft}$ in Fig. 8 yields the following MSO-definition of $\blacktriangleleft_W$ for binary branching intended trees and underlying grammars with macros with at most four parameters. We allow ourselves a predicate $\mathsf{lexical}(x)$ which identifies all elements with a "lexical" label, i.e., in the example those are $\bullet'$, ε, a', b', and c'.

$$\begin{aligned}
x \blacktriangleleft_W y &\overset{def}{\iff} \mathsf{trans}_{W_{\blacktriangleleft}}(x,y) \\
&\overset{def}{\iff} \mathsf{lexical}(x) \wedge (\exists z)[\mathsf{edg}_1(z,x) \wedge \\
&\quad (\exists u)[(\mathsf{edg}_2(z,u) \vee \mathsf{edg}_3(z,u) \vee \mathsf{edg}_4(z,u) \vee \mathsf{edg}_5(z,u)) \wedge \\
&\quad\quad (\forall X)(\forall v,w)[(v \in X \wedge (c\text{-}\mathsf{link}(v,w) \vee \\
&\quad\quad\quad (\pi_1\text{-}\mathsf{link}(v,w) \vee \pi_2\text{-}\mathsf{link}(v,w) \vee \\
&\quad\quad\quad \pi_3\text{-}\mathsf{link}(v,w) \vee \pi_4\text{-}\mathsf{link}(v,w))) \rightarrow \\
&\quad\quad\quad w \in X) \wedge (u \in X \rightarrow y \in X)] \wedge \\
&\quad\quad\quad\quad \mathsf{lexical}(y)]].
\end{aligned} \tag{13}$$

$$c\text{-}\mathsf{link}(x,y) \overset{def}{\iff} C(x) \wedge \mathsf{edg}_1(x,y) \tag{14}$$

$$\pi_1\text{-}\mathsf{link}(x,y) \overset{def}{\Longleftrightarrow} \Pi_1(x) \wedge (\exists z)(\forall X)(\forall u,v)[(u \in X \wedge (\mathsf{edg}_2(v,u) \vee \mathsf{edg}_3(v,u) \vee \mathsf{edg}_4(v,u) \vee \mathsf{edg}_5(v,u)) \rightarrow v \in X) \wedge (x \in X \rightarrow z \in X) \wedge (\exists w)[\mathsf{edg}_1(w,z) \wedge \mathsf{edg}_2(w,y)]]. \tag{15}$$

$$\pi_2\text{-}\mathsf{link}(x,y) \overset{def}{\Longleftrightarrow} \Pi_2(x) \wedge (\exists z)(\forall X)(\forall u,v)[(u \in X \wedge (\mathsf{edg}_2(v,u) \vee \mathsf{edg}_3(v,u) \vee \mathsf{edg}_4(v,u) \vee \mathsf{edg}_5(v,u)) \rightarrow v \in X) \wedge (x \in X \rightarrow z \in X) \wedge (\exists w)[\mathsf{edg}_1(w,z) \wedge \mathsf{edg}_3(w,y)]]. \tag{16}$$

We omit the obvious definitions for $\pi_3\text{-}\mathsf{link}(x,y)$ and $\pi_4\text{-}\mathsf{link}(x,y)$.

The edg-relations used above are instantiations of the following definition scheme suitable for the general, n-branching tree:

$$\mathsf{edg}_n(x,y) \overset{def}{\Longleftrightarrow} (\exists x_1,\cdots,x_{n-1})[x \lhd x_1 \wedge \cdots \wedge x \lhd x_{n-1} \wedge x \lhd y \wedge x_1 \prec x_2 \wedge \cdots \wedge x_{n-1} \prec y \wedge (\forall w)[x \lhd w \wedge w \not\approx x_1 \wedge \cdots \wedge w \not\approx x_{n-1} \wedge w \not\approx y \rightarrow y \prec w]] \tag{17}$$

In a similar vein, generalization of $\blacktriangleleft_W$ to the unrestricted case $\blacktriangleleft$ of the n-branching tree and macros of arity i, $n,i < \omega$ is trivial. The only changes required concern (13) and (15/16), again introducing a definition scheme in the case of (15'):

$$x \blacktriangleleft y \overset{def}{\Longleftrightarrow} \mathsf{lexical}(x) \wedge (\exists z)[\mathsf{edg}_1(z,x) \wedge (\exists u)[(\bigvee_{1<n<\omega} \mathsf{edg}_n(z,u)) \wedge (\forall X)(\forall v,w)[(v \in X \wedge (c\text{-}\mathsf{link}(v,w) \vee (\bigvee_{1<i<\omega} \pi_i\text{-}\mathsf{link}(v,w))) \rightarrow w \in X) \wedge (u \in X \rightarrow y \in X)]] \wedge \mathsf{lexical}(y)]. \tag{13'}$$

$$\pi_i\text{-}\mathsf{link}(x,y) \overset{def}{\Longleftrightarrow} \Pi_i(x) \wedge (\exists z)(\forall X)(\forall u,v)[(u \in X \wedge (\bigvee_{1<n<\omega} \mathsf{edg}_n(v,u)) \rightarrow v \in X) \wedge (x \in X \rightarrow z \in X) \wedge (\exists w)[\mathsf{edg}_1(w,z) \wedge \mathsf{edg}_{i+1}(w,y)]]. \tag{15'}$$

Notes

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

[2] Note that x is reachable from itself if the directive was a (successful) test.

[3] For example, the production for $F' \longrightarrow c(F', c(\bullet', a', \pi_1^4), \pi_2^4, c(\bullet', c', \pi_3^4), \pi_4^4)$ is transformed into the following two new productions: $F' \longrightarrow c(F', C1, \pi_2^4, C2, \pi_4^4)$, $C1 \longrightarrow c(\bullet', a', \pi_1^4)$ and $C2 \longrightarrow c(\bullet', c', \pi_3^4)$ The full translation of the other productions into the normal form is left to the reader.

[4] P_a stands for the predicate labeling a node with the symbol a and $\mathsf{leaf}(x) \overset{def}{\Longleftrightarrow} (\neg\exists y)[x \lhd y]$.

[5] Since Π-nodes are never leftmost daughters, the sublanguage $\mathsf{L}(x) \cdot \uparrow_1 \cdot (\downarrow_2 \cup \downarrow_3 \cup \downarrow_4 \cup \downarrow_5) \cdot (W_{\Pi_1} \cup W_{\Pi_2} \cup W_{\Pi_3} \cup W_{\Pi_4})^* \cdot W_{\mathbf{C}}{}^* \cdot \mathsf{L}(x)$ would be even more accurate for our purposes. The trivial changes to $\mathfrak{A}_{\lhd}$ required to compute exactly this language (adding an extra state and duplicating several transitions) just enlarge the automaton, decreasing its perspicuity without providing any new insights.

[6] This is certainly true for LIFTed structures resulting from *linear* context-free tree grammars. For non-linear ones, we have to take each individual run of the automaton into consideration which complicates matters considerably.

References

Aho, A. V. and J. D. Ullman: 1971, 'Translations on a Context-Free Grammar'. *Information and Control* **19**, 439–475.

Bloem, R. and J. 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.

Büchi, J. R.: 1960, 'Weak Second-Order Arithmetic and Finite Automata'. *Z. Math. Logik Grundlag. Math.* **6**, 66–92.

Chomsky, N.: 1985, *Barriers*. Cambridge, MA: MIT Press.

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

Courcelle, B.: 1990, 'Graph Rewriting: An Algebraic and Logic Approach'. In: J. van Leeuwen (ed.): *Handbook of Theoretical Computer Science*, Vol. B. Elsevier, pp. 193–242.

Courcelle, B.: 1992, 'The Monadic Seond-Order Logics of Graphs VII: Graphs as relational structures'. *TCS* **102**, 3–33.

Courcelle, B.: 1997, 'The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic'. In: G. Rozenberg (ed.): *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations.* World Scientific, Chapt. 5, pp. 313–400.

Doner, J. E.: 1970, 'Tree Acceptors and some of their Applications'. *J. Comput. System Sci.* **4**, 406–451.

Elgot, C. C.: 1961, 'Decision Problems of Finite Automata Design and Related Arithmetics'. *Trans. Amer. Math. Soc.* **98**, 21–51.

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

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

Engelfriet, J. and V. van Oostrom: 1996, 'Regular Description of Context-free Graph Languages'. *Journal Comp. & Syst. Sci.* **53**(3), 556–574.
Fischer, M. J.: 1968, 'Grammars with Macro-like Productions'. In: *Proceedings of the 9th Annual Symposium on Switching and Automata Theory*. pp. 131–142.
Frank, R. and K. Vijay-Shanker: 1998, 'Primitive C-Command'. Ms., Johns Hopkins University & Universtiy of Delaware.
Gécseg, F. and M. Steinby: 1984, *Tree Automata*. Budapest: Akadémiai Kiadó.
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: G. J. de Haan, M. Trommelen, and W. Zonneveld (eds.): *Van periferie naar kern*. Dordrecht: Foris, pp. 81–99.
Kolb, H.-P. and U. Mönnich (eds.): 1999, *The Mathematics of Syntactic Structure: Trees and Their Logics*. Berlin: Mouton-de Gruyter.
Maibaum, T. S. E.: 1974, 'A Generalized Approach to Formal Languages'. *J. Comput. System Sci.* **88**, 409–439.
Mezei, J. and J. B. Wright: 1967, 'Algebraic Automata and Contextfree Sets'. *Inform. and Control* **11**, 3–29.
Michaelis, J.: 1999, 'Derivational Minimalism is Mildly Context-Sensitive'. In: M. Moortgat (ed.): *LACL '98*, LNAI. Springer. To appear.
Michaelis, J., U. Mönnich, and F. Morawietz: 2000, 'Algebraic descriptions of derivational minimalism'. In: *Algebraic Methods in Language Processing: Second AMAST Workshop on Language Processing*. University of Iowa. To Appear.
Mönnich, U.: 1999, 'On Cloning Contextfreeness'. in (Kolb and Mönnich, 1999), pp. 195–229.
Peters, P. S. and R. W. Ritchie: 1973, 'On the generative power of transformational grammars'. *Information Sciences* **6**, 49–83.
Rogers, J.: 1998, *A Descriptive Approach to Language-Theoretic Complexity*, Studies in Logic, Language and Information. CSLI and FOLLI.
Rounds, W. C.: 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*. pp. 109–116.
Shieber, S. M.: 1985, 'Evidence against the Context-Freeness of Natural Language'. *Linguistics & Philosophy* **8**(3), 333–344.
Stabler, E.: 1997, 'Derivational Minimalism'. In: C. Retoré (ed.): *Logical Aspects of Computational Linguistics*. Berlin, pp. 68–95. LNAI 1328.
Thatcher, J. W. 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.
Thomas, W.: 1990, 'Automata on Infinite Objects'. In: J. van Leeuwen (ed.): *Handbook of Theoretical Computer Science*. Elsevier Science Publishers B. V., Chapt. 4, pp. 133–191.
Vijay-Shanker, K. and D. J. Weir: 1994, 'The Equivalence of Four Extensions of Context-Free Grammars'. *Mathematical Systems Theory* **27**(6), 511–546.