An Operational and Denotational Approach to Non-Context-Free Phenomena

Hans-Peter Kolb   Jens Michaelis   Uwe Mönnich

Frank Morawietz[1]

Universität Tübingen
Seminar für Sprachwissenschaft
Wilhelmstr. 113
72074 Tübingen
Germany

July 14, 2000

[1]Corresponding author is Frank Morawietz: `frank@sfs.nphil.uni-tuebingen.de`

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 (b) both a logical and an operational description (to establish the intended linguistic relations). The logical description is specified in terms of binary monadic second-order (MSO) formulas and the operational description is achieved by means of a linear and non-deleting macro tree transducer. Besides giving a grammatical presentation for the regular tree language we shall also specify an implementation in the form of a finite-state (tree) automaton to emphasize the effectivity of our approach.

Contents

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 Jim Rogers for helpful comments.

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 (RTGs). 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 a special type of tree transformation effected by a linear non-deleting macro tree transducer. Besides giving a grammatical presentation for the regular tree language we shall also create an implementation in the form of a finite-state (tree) automaton to emphasize the effectivity of our approach.

The other result is a logical description of cross-serial dependencies in terms of Monadic Second Order (MSO) logic. The description says that the structures underlying crossing dependencies can be specified as MSO definable relations. These have to be defined on a domain of finite trees which 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 between logical formalisms and language classes. They provide translation procedures that transform logical specifications into finite automata equivalent to the language classes and vice versa. Büchi (1960) and Elgot (1961) have shown that regular string languages represented through finite (string) automata can be expressed by sentences in the weak MSO logic with one successor. For tree languages an analogous result is well known: a tree language is definable in weak MSO 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 reason 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. The properties 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, its statements can be phrased in terms of linguistically significant notions that enter into universal principles and language-particular constraints. This due to the fact that an operational process starts from some given objects and then generates its space of interpretation. The logical approach, on the other hand, refers directly to an assumed universe of structures. Finally, those logical languages that capture complexity classes indicate lower bounds on the computing resources a system has to make available for using those structures that fall within the classes correlated with the corresponding logical language (Immerman; 1987).

In a previous paper (Kolb et al.; 2000) we have shown how to implement the binary MSO formulas mentioned above by finite-state tree-walking automata. In this paper we show that the intended linguistic structures can be regained by a simple macro tree transducer. To be more precise, we construct, in a first step, a regular representation of the cross-serial dependencies using a certain amount of explicit control information. Trees exhibiting this control information are elements of an absolutely free algebra on a signature that results from a well-known derivation process to be explained further below. The macro tree transducer serves to evaluate the tree terms of the free algebra in a different semantic domain. In other words, it constitutes the implementation of the uniquely given homomorphism which sends the trees with the explicit control information into the originally intended structures.

Macro tree transducers integrate the top-down aspect of a tree transducer and the bottom-up aspect of a context-free tree grammar. According to this analysis, a context-free tree grammar appears as a macro tree transducer without any input features. Our macro tree transducer receives its input information from the lifted trees exhibiting the control information and handles its context information via the arguments of its states. These context parameters play the same rôle as the parameters in the original context-free tree grammar. Since the tree-walking automata that served to give an operational account of the intended linguistic relations in the previous paper had to be constructed on a piecemeal basis, the question remained open of how to specify an implementation in a more compact way. We hope that the present result gives a satisfying answer to this question.

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 (CFTGs). This formalism was introduced by Fischer (1968) and constitutes a generalization of RTGs. Recall that the step from regular to context-free string grammars essentially consists in lifting the restriction to initial and final string positions for nodes to be substitutable. In analogy, the rule format which characterizes CFTGs allows for the substitution of inner tree nodes in contrast to RTGs which limit substitutability to terminal nodes. After this first step we LIFT the generated trees by inserting a certain amount of explicit control information. It turns out that the resulting structures can be characterized with RTGs. We exploit the equivalence of RTGs/tree automata and MSO specifications 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 MSO formulas that are defined on the nodes of the lifted trees. The actual formal definition will take the form of an MSO definable transduction translating the lifted structures into the intended ones. We supplement this logical simulation with an operational account of this logic based transduction. As was outlined in the previous paragraph, we provide the necessary details of a macro tree transducer that performs in a clearly operational way the sort of tree transformation that is—statically—specified by the MSO transduction.

The basic idea underlying this sequential application of two logical specification steps 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.

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. In the same way, defined tree operations are the appropriate range for variables in CFTGs. 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. 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. Thereby we 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 RTGs. 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.

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 the logical part of 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 binary MSO formulas are not dependent on the particular regular tree family or the domain defined via the MSO formula. The instruction set of the tree-walking automata behind these formulas and the corresponding definition of the MSO transduction are universal and only serve to reverse the lifting process. In that sense the instructions are nothing else but a restatement of the unique homomorphism which exists between the free algebra and any other algebra of the same signature. The same statement holds for the macro tree transducer. Its rules serve to evaluate the instructions implicit in the composition and projection symbols. These instructions are again independent of the particular tree family specified by the closed MSO formula.

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 $f(t_1, \ldots, t_n) \in T(\Sigma)$ if $f \in \Sigma_n$ and $t_i \in T(\Sigma)$ for $i = 1, \ldots, n$.

Furthermore, we fix an indexed set $X = \{x_1, x_2, \ldots\}$ of *variables* and denote by X_n the subset $\{x_1, \ldots, x_n\}$. 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: $h(f_{\mathbb{A}}(a_1, \ldots, a_n)) = f_{\mathbb{B}}(h(a_1), \ldots, h(a_n))$ 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 n-tuple $(a_1, \ldots, a_n) \in A^n$: $t_{\mathbb{A}}(a_1, \ldots, a_n) = \hat{a}(t)$, 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 formally introduce 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. CFTGs constitute an algebraic generalization of macro grammars.

Let us view grammars as a mechanism in which local transformations on trees can be performed. 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. Recall that in string languages a nonterminal auxiliary symbol is rewritten as a string of terminal and nonterminal symbols, independently of the context in which it occurs. As long as the carrier of a tree algebra is made of constant tree terms the process of replacing null-ary variables by trees is analogous. 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 these second-level entities.

Definition 1 (Tree Grammars) *A* context-free tree grammar *(CFTG)* $\Gamma = \langle \Sigma, \mathrm{F}, S, \mathrm{X}, \mathrm{P} \rangle$ *is a 5-tuple, where* Σ, F *are the ranked alphabets of* inoperatives *and* operatives*, respectively;* $S \in \mathrm{F}$ *is the starting symbol;* X *is a countable set of variables; and* P *is a set of productions. The* $p \in \mathrm{P}$ *have the form* $F(x_1, \cdots, x_n) \longrightarrow t$*, with* $F \in \mathrm{F}$*,* $x_1, \cdots, x_n \in \mathrm{X}$*, and* t *a term (tree) over* $\Sigma \cup \mathrm{F} \cup \{x_1, \cdots, x_n\}$*.*
An application of a rule $F(x_1, \ldots, x_n) \to t$ *"rewrites" a tree rooted in* F *as the tree* t *with its respective variables substituted by* F*'s daughters.*

Tree grammars with $\mathrm{F}_n = \emptyset, n \neq 0$, are called *regular* (RTGs). 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 $\mathrm{F}_n, n \neq 0$, is non-empty, that is, if we allow the *operatives* to be parameterized by variables, however, the situation changes. These *context-free* 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 where "rightmost" is to be understood with respect to the linear order of the leaves forming the frontier of a tree in a derivation step.

Definition 2 *Let* $\Gamma = \langle \Sigma, \mathrm{F}, S, \mathrm{X}, \mathrm{P} \rangle$ *be a CFTG and let* $t, t' \in T(\Sigma \cup \mathrm{F})$*.* t' *is* directly derivable *by an* inside-out step *from* t *(*$t \Longrightarrow t'$*) if there is a tree* $t_0 \in T(\Sigma \cup \mathrm{F}, X_1)$

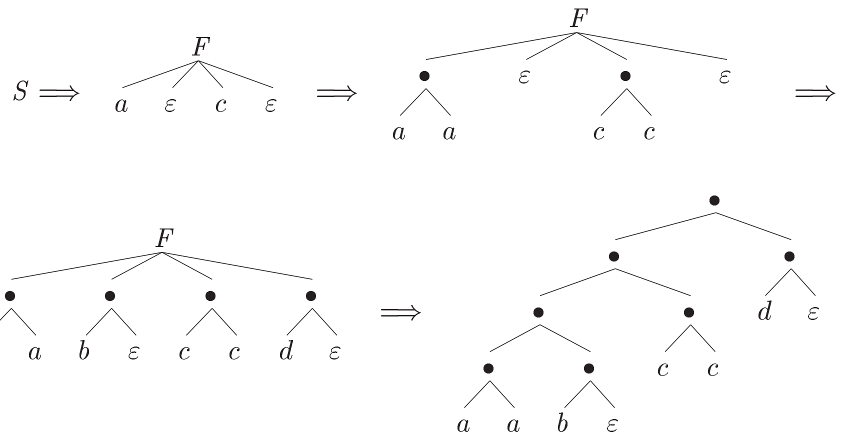


Figure 1: An example derivation of (1): *aabccd*

containing exactly one *occurrence of* x_1*, a corresponding rule* $F(x_1,\ldots,x_m) \to t''$*, 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, i.e., do not contain variables or operatives. As is customary* $\stackrel{*}{\Longrightarrow}$ *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 CFTG. 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.*

In the case of RTGs the analogy with the conventional string theory goes through and inside-out and outside-in derivations yield the same languages.

We will exemplify the gain in generative power of context-free tree grammars compared to RTGs—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 Section 3). The example uses the full power of the second-order substitutions of operators.

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

$$\mathsf{P} = \left\{ \begin{array}{rcl} S & \longrightarrow & \varepsilon \\ S & \longrightarrow & F(a,\varepsilon,c,\varepsilon) \\ S & \longrightarrow & F(\varepsilon,b,\varepsilon,d) \\ F(x_1,x_2,x_3,x_4) & \longrightarrow & F(\bullet(a,x_1),x_2,\bullet(c,x_3),x_4) \\ F(x_1,x_2,x_3,x_4) & \longrightarrow & F(x_1,\bullet(b,x_2),x_3,\bullet(d,x_4)) \\ F(x_1,x_2,x_3,x_4) & \longrightarrow & \bullet(\bullet(\bullet(x_1,x_2),x_3),x_4) \end{array} \right\} \qquad (1)$$

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 $\mathcal{L}' = \{a^n b^m c^n d^m\}$.

In Figure 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. The LIFT-operation takes a term in $T(\Sigma, \mathrm{X}_k)$ and transforms it into one in $T(\Sigma^L, k)$.[1] Intuitively, the lifting eliminates variables and composes functions with their arguments explicitly, e.g., a term $f(a,b) = f(x_1,x_2) \circ (a,b)$ is lifted to the term $c(c(f,\pi_1,\pi_2),a,b)$.

Definition 4 (LIFT) *Suppose that* Σ *is a ranked alphabet and* $\mathrm{X}_k = \{x_1,\ldots,x_k\}$. *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* C, *the* π_i *by* Π.

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

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

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

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.

A proof that the result of LIFT-ing a CFTG is always a RTG can be found in Mönnich (1999).

As an example, we present the LIFTed version Γ^L of the CFTG 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 right hand sides (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^2_i from Σ^L_2, all the 0- and 1-place composition symbols and the subscripts on all other composition

[1] Since we are working in a single-sorted signature, we can identify S^* with $\mathbb{N}$. This is due to the fact that up to length the elements of S^* are unique strings. So we simplify $T(\Sigma^L, \langle \varepsilon, \langle s^k, s \rangle \rangle)$ to $T(\Sigma^L, k)$

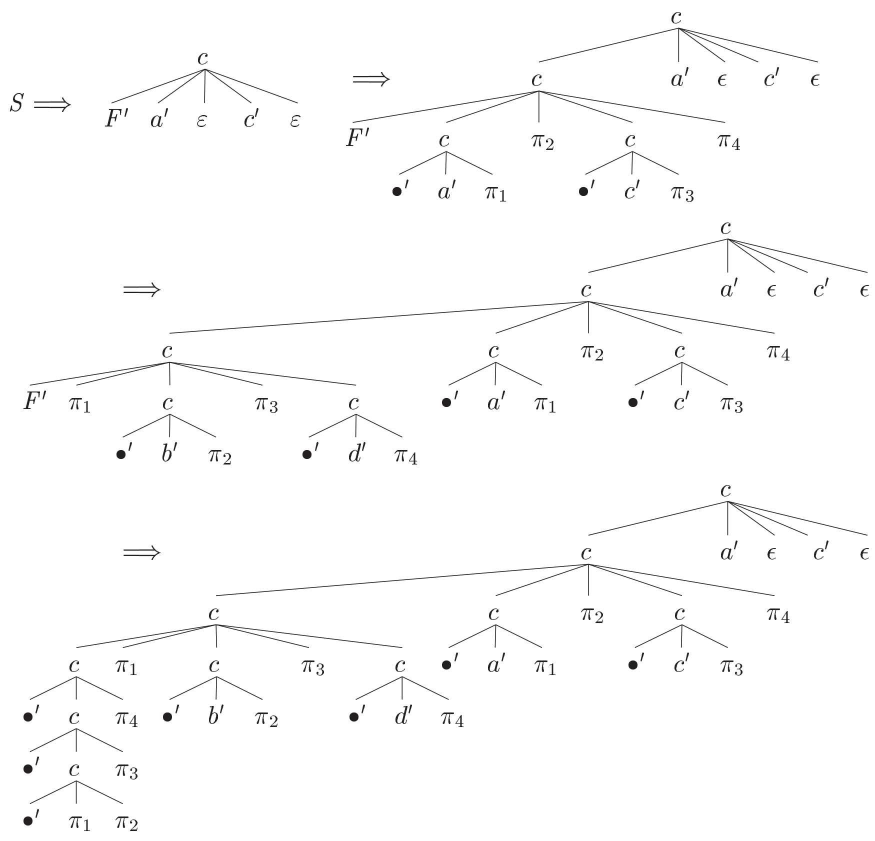


Figure 2: A sample derivation using the LIFTed grammar

symbols.

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

$$
\mathrm{P} = \left\{ \begin{array}{lcl}
S' & \longrightarrow & \varepsilon \\
S' & \longrightarrow & c(F', a', \varepsilon, c', \varepsilon) \\
S' & \longrightarrow & c(F', \varepsilon, b', \varepsilon, d') \\
F' & \longrightarrow & c(F', c(\bullet', a', \pi_1^4), \pi_2^4, c(\bullet', c', \pi_3^4), \pi_4^4) \\
F' & \longrightarrow & c(F', \pi_1^4, c(\bullet', b', \pi_2^4), \pi_3^4, c(\bullet', d', \pi_4^4)) \\
F' & \longrightarrow & c(\bullet', c(\bullet', c(\bullet', \pi_1^4, \pi_2^4), \pi_3^4), \pi_4^4)
\end{array} \right\} \tag{2}
$$

We parallel the derivation for *aabccd* shown in Figure 1 with this lifted grammar in Figure 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 (cf. 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 MSO 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 MSO 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 and 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 within MSO logic is a complex issue. 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. In contrast, addition of monadic predicates is allowed since they can be added freely to $\mathbf{P}$.

The intended models of $L^2_{K,P}$ are (finite) 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). Recall that representation of objects with relational structures makes them available for the use of logical description languages. Let R be a finite set of relation symbols with the corresponding arity for each $r \in R$ given by $\rho(r)$. A relational structure $\mathcal{R} = \langle D_\mathcal{R}, (r_\mathcal{R})_{r\in\mathcal{R}} \rangle$ consists of the domain $D_\mathcal{R}$ and the $\rho(r)$-ary relations $r_\mathcal{R} \subseteq D_\mathcal{R}^{\rho(r)}$. There does not seem to be a convenient machine model for tree transformations. Fortunately, one can use logic directly to define the desired transduction. The classical technique of interpreting a relational structures within another one forms the basis for MSO transductions. Intuitively, the output tree is interpreted on the input tree. E.g., suppose that we want to transduce the input tree t_1 into the output tree t_2. The nodes of the output tree t_2 will be a subset of the nodes from t_1 specified with a unary MSO relation ranging over the nodes of t_1. The daughter relation will be specified with a binary MSO relation with free variables x and y ranging over the nodes from t_1. We will use this concept to transform the lifted trees into the intended ones.

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 an MSO formula φ defining the domain of the transduction in $\mathcal{R}$, an MSO formula ψ defining the resulting domain of $\mathcal{Q}$, and a family of MSO formulas θ_q defining the new relations Q using only definable formulas from the "old" structure $\mathcal{R}$.

In this sense, our logical description of non-context-free phenomena with two devices with only regular power is an instance of the theorem that the image of an MSO-definable class of structures under a definable transduction is not MSO definable in general (Courcelle; 1997).

2.4 Automata and Transducers

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 $\delta : \bigcup_n (A^n \times \Sigma_n) \to A$ the transition function.*

We can extend the transition function inductively to trees by defining $h_\delta(\varepsilon) = a_0$ and $h_\delta(\sigma(t_1,\ldots,t_n)) = \delta(h_\delta(t_1),\ldots,h_\delta(t_n),\sigma)$, $t_i \in T_\Sigma, 1 \leq i \leq n, \sigma \in \Sigma_n$. 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. 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.[2] Finally, tree automata can be minimized by a construction which is, yet again, a straightforward generalization of well known FSA techniques.

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

Intuitively, top-down tree transducers transform trees over a ranked alphabet Σ into ones over a ranked alphabet Ω. They traverse a tree from the root to the leaves (the input tree) and output on each transition a new tree whose nodes can contain labels from both alphabets, states and variables. More formally, the right hand sides of such

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

a production are trees from $T_\Omega(\Sigma(X) \cup Q)$. For this definition we assume that Q is a ranked alphabet containing only unary symbols.

Definition 6 (Top-Down Tree Transducer) *A top-down tree transducer (TDTT) is a a tuple* $T = \langle Q, \Sigma, \Omega, q_0, P \rangle$ *with states* Q*, ranked alphabets* Σ *and* Ω *(input and output), initial state* q_0 *and a finite set of productions* P *of the form*

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

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

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

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

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

Figure 3 shows one application of the nontrivial rule. The left hand side displays the rule in tree notation whereas the right hand side displays an actual transition.

Figure 3: One step of an TDTT derivation

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

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

Definition 7 *Let* Σ *and* Ω *be ranked alphabets and* $n, m \geq 0$*. The set of right hand sides* $RHS(\Sigma, \Omega, n, m)$ *over* Σ *and* Ω *with* n *variables and* m *parameters is the smallest set* $rhs \subseteq T_{\Sigma \cup \Omega}(X_n \cup Y_m)$ *such that*

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

The productions of macro tree transducers contain one "old" parameter (an alphabet symbol with the appropriate number of variables) and additionally a number of context parameters.

Definition 8 (Macro Tree Transducer) *A macro tree transducer is a five-tuple* $M = \langle Q, \Sigma, \Omega, q_0, P\rangle$ *with* Q *a ranked alphabet of states, ranked alphabets* Σ *and* Ω *(input and output), initial state* q_0 *of rank* 1*, and a finite set of productions* P *of the form*

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

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

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

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

Generally, just as for CFTGs, a little care has to be taken in the definition of the transition relation with respect to the occuring parameters y_1. Derivations are dependent on the order of tree substitutions. Inside-out (IO) means that trees from T_Ω have to be substituted for the parameters whereas in Outside-in (OI) derivations any subtree must not be rewritten if it is in some context parameter. Again, as for CFTGs, OI-derivations are equivalent to unrestricted ones, whereas IO-derivations produce less. Since we are only dealing with *simple* MTTs in our approach, all three modes are equivalent and can safely be ignored.

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

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

Note that if we disregard the input, MTTs turn into CFTGs.

Consider for example the following rule of an MTT M.

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

Analogous to the presentation in Figure 3, we illustrate the the rule above in Figure 4 without being too concerned about the formal details of specifying a full transducer.

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

3 Linguistic Motivation: Verb Raising

As mentioned in the introduction, the exercise in formal coding is made necessary by the fact that natural language sports at least some constructions which lead (i) to non-context-free string languages, or (ii) to at least non-recognizable tree languages, i.e.

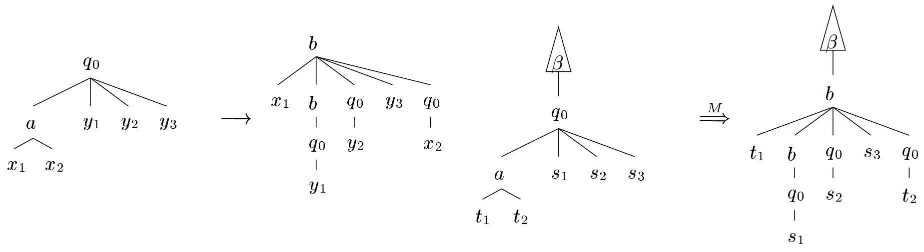


Figure 4: One step of an MTT derivation

tree sets which cannot be generated by any context-free string grammar or regular tree grammar, even though the resulting string languages may formally be context-free.[3] 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):

(3) a. weil der Karl die Maria dem Peter den Hans schwimmen lehren
because Charles Mary$_1$ Peter$_2$ John$_3$ swim$_3$-inf teach$_2$-inf

helfen lässt
help$_1$-inf lets

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

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

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

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

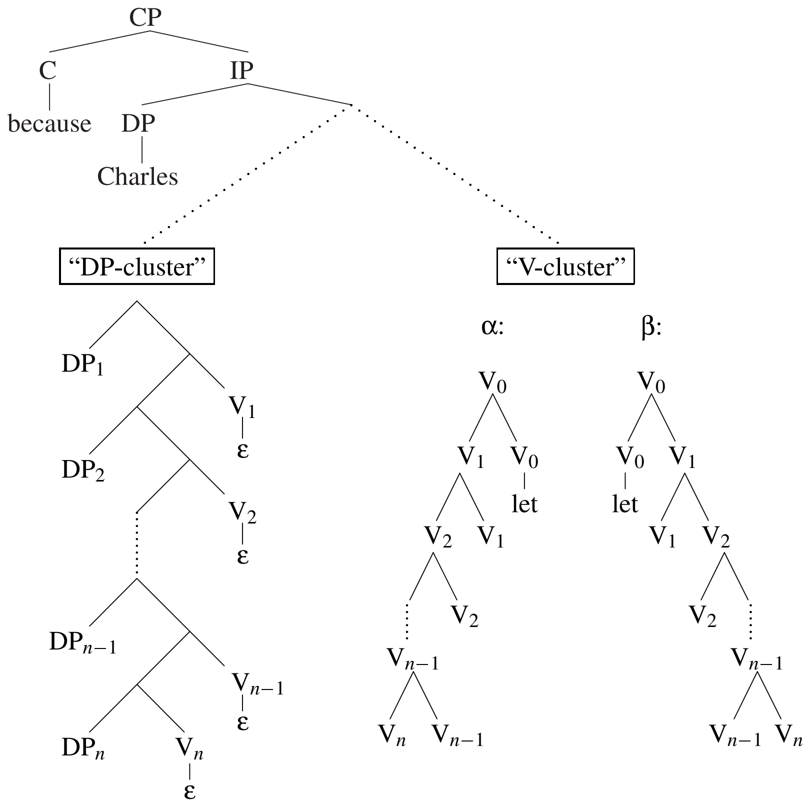
(Züritüütsch fragment as string language: ww—Non-CF)

'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. This is manifested by the case marking of the respective objects. It appears that there are no limits on the length of such constructions in grammatical sentences of Swiss German. This fact alone would not suffice to prove that Swiss German is not a context-free string language. It could still be the case that Swiss German *in toto* is context-free even though it subsumes an isolable context-sensitive fragment. Relying on the closure of context-free languages under intersection with regular languages, Shieber (1985) was able to show that not only the fragment exhibiting the cross-serial dependencies but the whole of the language has to be assumed as non context-free.[4]

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

[4] Huybregts (1984) provides a similar argument for Dutch taking into account a particular fragment: in contrast to Swiss german, Dutch does not show overt case-marking of objects. Huybregts argument



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

Figure 5: The structure of Germanic VR

Abstracting from the details of the particular languages, the standard analyses of these constructions involve the following property which is problematic from the point of view of context-freeness: In all cases they posit roughly a bipartite structure like the one in Figure 5 with basically all DP's on one branch and all the verbs on the other – but with fixed syntactic and semantic relations between the branches, whether visibly marked (as in Züritüütsch, Standard German) or not (Dutch).

As is easily seen, there is no context-free device which could directly handle the unbounded number of non-local dependencies the structural separation of the two "clusters" enforces. Therefore MSO logic alone cannot be sufficient for linguistic reasons. 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.

crucially relies on a given morphologized—and thus syntactical—difference between animate and inanimate pronominals.

4 A Tree Automaton and an MSO formula for Regular 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 $\Gamma^L = \langle \Sigma^L, \mathrm{F}, S, \mathrm{X}, \mathrm{P} \rangle$, i.e., an RTG, is straightforward. Intuitively, since tree automata recognize only local trees in any one transition, we have to use auxiliary transitions for RHSs 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., $Q = \{q_\sigma \mid \sigma \in \Sigma_i^L \cup F_0, 0 \leq i \leq 2\} \cup \{q_0\}$. 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., $q_0^n \times \sigma \to q_\sigma$; 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 nonterminal 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: $q_{d_1} \times \cdots \times q_{d_n} \times c \to q_{NT}$.

In the example, the lifted tree grammar does not have the desired normal form, but it is easy to see how to change it.[5] Accordingly, the tree automaton corresponding to the RTG Γ^L given in (2) looks as given in Figure 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) Σ_1^1-formula $\varphi_{\mathfrak{A}_{\Gamma^L}}$ given there adapted to our signature and for 5-ary tree automata looks as given in (4).[6]

[5] 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.

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

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

$$\delta : Q \times Q \times Q \times Q \times Q \times \Sigma \to Q$$

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

Figure 6: The tree automaton for Γ^L

Intuitively, the sets X_i label the tree where the automaton assumes state i. The first two lines 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.

$$\begin{aligned}
\varphi_{\mathfrak{A}_{\Gamma^L}} \stackrel{def}{\Longleftrightarrow} \quad & (\exists X_0,\ldots,X_m)[\bigwedge_{i\neq j}(\neg\exists y)[y\in X_i \land y \in X_j] \land \\
& \qquad (\forall x)[\mathsf{leaf}(x) \to x \in X_0] \\
& \land (\forall x_1,\ldots,x_5,y)[\bigvee_{\substack{(i_1,\ldots,i_5,\sigma,j)\in\delta \\ 1\leq k\leq 5}} x_k \in X_{i_k} \land y \lhd x_k \land y \in X_j \land y \in P_\sigma] \\
& \qquad \bigvee_{i\in F}(\exists x \forall y)[x \lhd^* y \land x \in X_i]
\end{aligned} \tag{4}$$

5 Reconstructing the Intended Trees

Unfortunately, the terminal trees in Figure 2, generated/recognized by (1) or the tree automaton in Figure 6), don't seem to have much in common with the structures linguists want to talk about, i.e., the ones in Figure 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 suggest, *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 $R^I = \{\blacktriangleleft, \blacktriangleleft^+, \blacktriangleleft^*$ *(dominance)*, *c-command*, $\lessdot$ *(precedence)*$,\ldots\}$ 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 $\blacktriangleleft^* \in R^I$ as a tree order on the set of *"linguistic"* nodes and $\lessdot \in R^I$ as the precedence relation on the resulting structure we have a "new" description language on the intended structures.

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

5.1 The MSO Transduction

We have shown in Kolb et al. (2000) how to give an operational account of an MSO transduction to recover the intended relations via so called tree-walking automata with MSO tests.[7] In this paper, we will present the logical aspect of this transduction without going into the details of how to generate the relevant formulas. The interested reader is referred to the reference given above.

We will use $\mathsf{trans}_{W_\triangleleft}(x,y)$ as the formula denoting immediate dominance ($x \triangleleft y$) on the intended structures. This formula has been constructed recursively from the walking language of a tree-walking automaton linking the appropriate nodes in the lifted tree. Examples of these relations are displayed graphically in Figure 7 which contains another rendering of the last tree of the derivation given in Figure 2. The intended dominance relation marks the endpoints of these tree walks. The only problematic case of finding the appropriate filler for a node labeled with a projection symbol is illustrated with the π-links where we have to recursively traverse the tree (more on the details of how to find these can be found in Kolb et al. (2000)). While we have been sloppy with regard to the subscripts of the compositions so far, we are more precise in this figure and give the exact labeling for reasons which will become clear when we define the macro tree transducer.

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

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

$$R\text{-}\mathsf{closed}(X) \stackrel{def}{\Longleftrightarrow} (\forall x,y)[x \in X \wedge R(x,y) \to y \in X] \tag{5}$$

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

$$\begin{array}{lll} \textit{Reflexive Dominance:} & & \\ x \triangleleft^* y & \stackrel{def}{\Longleftrightarrow} & (\forall X)[\triangleleft\text{-}\mathsf{closed}(X) \wedge x \in X \to y \in X] \\ \textit{Proper dominance:} & & \\ x \triangleleft^+ y & \stackrel{def}{\Longleftrightarrow} & x \triangleleft^* y \wedge x \not\approx y \end{array} \tag{6}$$

Using the defined formula $\mathsf{trans}_{W_\triangleleft}(x,y)$ for $\triangleleft$, the specific MSO transduction we

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

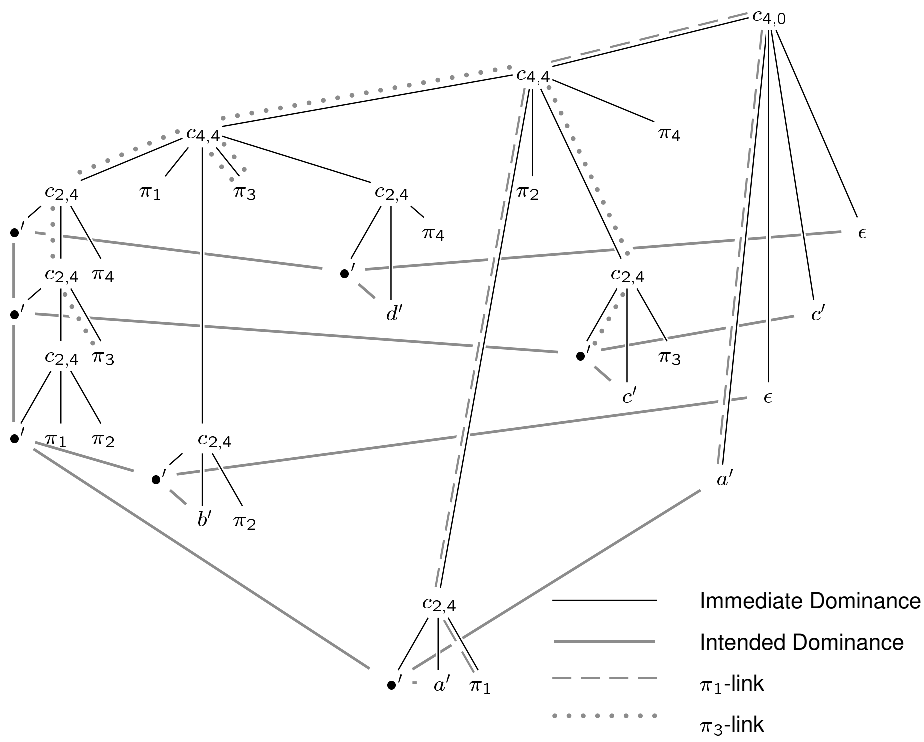


Figure 7: Intended relations on a LIFTed structure

need to transform the LIFTed structures into the intended ones looks as follows:

$$
\begin{aligned}
&(\varphi, \psi, (\theta_q)_{q \in Q}) \\
&Q = \{ \blacktriangleleft, \blacktriangleleft^*, \blacktriangleleft^+, \vartriangleleft, \dots \} \\
\varphi \quad &\equiv \varphi_{\mathfrak{A}_{\Gamma L}} \\
\psi(x) \quad &\equiv (\exists y)[x \blacktriangleleft y \vee y \blacktriangleleft x] \\
\theta_{\blacktriangleleft}(x,y) \quad &\equiv \mathsf{trans}_{W_{\blacktriangleleft}}(x,y) \\
\theta_{\blacktriangleleft^*}(x,y) \quad &\equiv (\forall X)[\blacktriangleleft\text{-}\mathsf{closed}(X) \wedge x \in X \rightarrow y \in X] \\
\theta_{\blacktriangleleft^+}(x,y) \quad &\equiv x \blacktriangleleft^* y \vee x \not\approx y \\
\theta_{\vartriangleleft}(x,y) \quad &\equiv \mathsf{trans}_{W_{\vartriangleleft}}(x,y) \\
\theta_{\mathrm{L} \in \mathrm{Labels}}(x) \quad &\equiv L(x)
\end{aligned}
\tag{7}
$$

As desired, the domain of the transduction is characterized by the MSO formula for the LIFTed trees (see Section 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 the MSO language $L^2_{K,P}$ used to analyze large parts of GB theory in Rogers (1998). For reasons of space, we have to leave the specification of the precedence relation $\mathsf{trans}_{W_{\vartriangleleft}}(x,y)$ open. It is more complicated than dominance, but can be achieved with another tree-walking automaton.[8] Finally, the labeling information for the nodes is just taken over from R.

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

5.2 The Macro Tree Transducer

As stated previously, there is a *unique* morphism h from the "lifted" terms over the derived alphabet Σ^L into the terms over the tree substitution algebra, where $c(t, t_1, \dots, t_k)$ for $t \in T(\Sigma, X_k)$, $t_1, \dots, t_k \in T(\Sigma, X_m)$ denotes the result of *substituting* t_i for x_i in t.

The morphism h is defined inductively as follows:

$$
\begin{aligned}
h(f') &= f(x_1, \dots, x_n) \text{ for } f \in \Sigma_n \\
h(\pi^n_i) &= x_i \\
h(c(t, t_1, \dots, t_n)) &= h(t)[h(t_1), \dots, h(t_n)]
\end{aligned}
\tag{8}
$$

where $t[t_1, \cdots, t_n]$ denotes the result of substituting t_i for x_i in t for $t \in T(\Sigma, X_k)$, $t_i \in T(\Sigma, X_m)$.[9]

[8] 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.

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

This unique morphism h can be performed by a simple macro tree transducer $M = \langle Q, \Sigma^L, \Sigma, q_0, P \rangle$, where $Q = \{q_n \mid n \in$ arity of elements in $\Sigma^L\}$, q_0 is the initial state and P is a finite family of rules.

The MTT which we construct to carry out the transformation effected by the unique homomorphism h combines in a particularly perspicuous way the actions of a top-down finite tree transducer— based upon the syntactic structure of the lifted alphabet Σ^L—and the production aspect of the underlying CFT-grammar via its (i.e. the MTT's) dependence on the local context (parameters): retrieving the "old" arity out of the new sorted constants of the signature Σ^L.

How can we construct the necessary productions to recover the intended trees? We take an intuitive approach to explaining the construction of the needed MTT which is strongly dependent on inspection of the tree in Figure 7.

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

After careful inspection of the tree language generated by the lifted RTG Γ^L, the simplest case is certainly when we are faced with a constant from Σ_0^L. In this case all we have to do is to map it back to the corresponding element from Σ, regardless of the parameters, if there are any. In case we encounter a projection symbol we simply have to return the corresponding parameter. Obviously, this presupposes that we stored the "right" information there.

Furthermore, all rules with a symbol whose "unlifted" version was not a constant will have as many parameters as are needed to compute the corresponding function, e.g., $\bullet$ obviously is binary and therefore needs two parameters (see the last rule) in (9). The resulting rule has, on the right hand side, simply the "executed" function.

For the rules headed by a composition symbol $c_{n,k}$ we need as many parameters as are prescribed by k. This is due to the fact that while generally the relevant information in the lifted trees is on the leftmost branch we nevertheless need the other daughters to be able to unravel the projections. Basically, we follow a depth-first strategy on the leftmost component of the lifted trees while still passing the necessary context (i.e., the evaluation of the computation of the other daughters) down into that computation. Similarly, we also get the necessary states from the arities of the composition symbols. The rules then simply pass the state and the parameters of the left hand side of the rule to the arguments of the alphabet symbol while continuing to work on the first argument. As an example consider an $n+1$ branching fork whose mother is labeled with $c_{n,k}$. Then we have to construct a rule which has on the left hand side state q_k with arity $k+1$. It has as its first argument a term with functor $c_{n,k}$ and arguments $x_1, \ldots, x_{n+1}$. The other arguments are the parameters y_1 to y_k. The right hand side has state q_n of arity $n+1$ with the first argument simply being x_1 and the other arguments being $q_k(x_i, y_1, \ldots, y_k)$, $1 < i \leq n+1$.

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

$$
\begin{aligned}
q_0(c_{4,0}(x_1,x_2,x_3,x_4,x_5)) &\longrightarrow \\
&\quad q_4(x_1,q_0(x_2),q_0(x_3),q_0(x_4),q_0(x_5)) \\
q_0(\sigma') &\longrightarrow \sigma \qquad\qquad \text{for } \sigma \in \{a,b,c,d,\varepsilon\} \\
q_4(c_{4,4}(x_1,x_2,x_3,x_4,x_5),y_1,y_2,y_3,y_4) &\longrightarrow \\
&\quad q_4(x_1,q_4(x_2,y_1,y_2,y_3,y_4),q_4(x_3,y_1,y_2,y_3,y_4), \\
&\quad\quad q_4(x_4,y_1,y_2,y_3,y_4),q_4(x_5,y_1,y_2,y_3,y_4)) \qquad (9) \\
q_4(P_i,y_1,y_2,y_3,y_4) &\longrightarrow y_i \qquad\qquad \text{for } P_i = \pi_i \\
q_4(\sigma',y_1,y_2,y_3,y_4) &\longrightarrow \sigma \qquad\qquad \text{for } \sigma \in \{a,b,c,d,\varepsilon\} \\
q_4(c_{2,4}(x_1,x_2,x_3),y_1,y_2,y_3,y_4) &\longrightarrow \\
&\quad q_2(x_1,q_4(x_2,y_1,y_2,y_3,y_4),q_4(x_3,y_1,y_2,y_3,y_4)) \\
q_2(\bullet',y_1,y_2) &\longrightarrow \bullet(y_1,y_2)
\end{aligned}
$$

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

Applying this MTT to the tree in Figure 7 yields the final tree from the derivation displayed in Figure 1. To get the reader started, let us consider the first rule in (9) starting the transduction on the root of the tree displayed in Figure 7. We start in state q_0 and out root is indeed labeled with $c_{4,0}$. Then we continue in state q_4 with its leftmost daughter and pass as parameters the results of computing q_0 of the other daughters. Since in this case they are elements from Σ_0^L, we can simply use the appropriate constants from Σ in further computations. The rest of the computation leading to the final result is straightforward and left as an exercise.

6 Conclusion

We have shown in the paper how to account for cross-serial dependencies by coupling (a) a logical domain specification followed by a logically definable transduction and (b) a bottom-up finite-state tree automaton with a tree transformation induced by a macro tree transducer. The result is, of course, not restricted to cross-serial dependencies. Any type of structural relationship that is amenable to a formal analysis by means of CFTGs can be described according to the same operational 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 MTT/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 CFTGs are amenable to a regular analysis followed by a fixed universal transduction.

Comparing this statement of the result of the paper with the characterization of context-free graph languages by Engelfriet and van Oostrom (1996) 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 macro tree transducer/MSO transduction. For a recent result on the equivalence between regular tree languages followed by an MSO definable tree transduction and the tree languages generated by context-free graph grammars see Engelfriet and Maneth (1999).

One drawback of the approach, namely that there is no principled connection between CFTGs and a linguistic formalism which handles the cross-serial dependencies is addressed in another paper. Using the result from Michaelis (1999), a similar technique has been applied to code Minimalist Grammars (MGs) in the sense of Stabler (1997):[10] First, MGs are translated into multiple context-free grammars. The latter can be viewed as Lawvere theories such that we get, again, lifted structures which can be recognized with RTGs/tree automata/MSO formulas. Finally, those can be transduced into the intended structures with a variant of the MSO/MTT technique used here. Thus we get another two step approach showing a direct connection between MGs and the proposal here (Michaelis et al.; 2000).

References

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

Büchi, J. R. (1960). Weak second-order arithmetic and finite automata, *Z. Math. Logik Grundlag. Math.* **6**: 66–92.

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

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, chapter 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.

[10] MGs provide an attempt of a rigorous algebraic formalization of the new linguistic perspective which reflects the change within the linguistic framework of transformational grammar from GB-theory to minimalism.

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

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

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

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

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

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

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

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

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

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

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*, Foris, Dordrecht, pp. 81–99.

Immerman, N. (1987). Languages that capture complexity classes, *SIAM J. of Computing* **16**(4): 760–778.

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

Maibaum, T. S. E. (1974). A generalized approach to formal languages, *J. Comput. System Sci.* **88**: 409–439.

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

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

Michaelis, J., Mönnich, U. and Morawietz, F. (2000). Algebraic descriptions of derivational minimalism, *Algebraic Methods in Language Processing: Second AMAST Workshop on Language Processing*, number 16 in *TWLT*, Parlevink, Univerity of Twente, University of Iowa. To Appear.

Mönnich, U. (1999). On cloning contextfreeness, *in* H.-P. Kolb and U. Mönnich (eds), *The Mathematics of Syntactic Structure*, number 44 in *Studies in Generative Grammar*, Mouton de Gruyter, pp. 195–229.

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

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

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, *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*, Springer, Berlin, pp. 68–95. LNAI 1328.

Thatcher, J. W. and Wright, J. B. (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., chapter 4, pp. 133–191.

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