LOGICAL DESCRIPTION OF CROSS-SERIAL DEPENDENCIES
(DRAFT, March 29, 1999)

HANS-PETER KOLB, UWE MÖNNICH, AND FRANK MORAWIETZ

Seminar für Sprachwissenschaft, Universität Tübingen
Wilhelmstrasse 113, D-72074 Tübingen, Germany
{kolb,um,frank}@sfs.nphil.uni-tuebingen.de

ABSTRACT. The main result of this paper 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 in two steps as *MSO* definable relations which are defined on a domain of finite trees that is characterized as the model set of a closed *MSO* formula.

1. INTRODUCTION

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

The main result of this paper is a 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 relations where these relations 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 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 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 which connect the tree nodes standing in those linguistically significant relations.

The basic idea underlying this sequential applications 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.

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 leave 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 (1977/78), 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 Courcelle (1992). In that paper, 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 favoured 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. For this device of characterizing structural relations in terms of a family of previously specified explicit trees we are indebted to Engelfriet and van Oostrom (1996).

We think that our approach has two decisive advantages. First, the operations of the relevant signature appear explicitly in the lifted tree and are not hidden in the final evaluation result. Second, our logical definition of the intended relations is not dependent on the particular tree domain given by the *MSO* specifications of the regular tree language in the first step. The relations established by the logical definition of the second step is universal and only serves to reverse the lifting process. In that sense this logical definition is nothing 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.

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 the following ingredients:

(1) • three disjoint, countable sets of *individual constants* $\mathbf{K}$, *predicate constants* $\mathbf{P}$, and *ranked variables* $\mathbf{X} = \mathbf{X}^0 \cup \mathbf{X}^1$

- four binary predicates: *equality* $\approx$, plus the tree predicates *parent* $\triangleleft$, *dominance* $\triangleleft^\star$, and *left-of* $\prec$
- the usual connectives, quantifiers, brackets

where the $k \in \mathbf{K}$ are be 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 $\mathbf{K}$ and $\mathbf{P}$, respectively. We will write $X(x)$ for 'x is in the set assigned to X'. The syntax is the standard first order predicate logic syntax extended by quantification over monadic predicates: $\forall X \in X^1$, 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 can be explicitly defined. Monadic predicates, however, are always 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 $\triangleleft$, $\triangleleft^\star$, $\prec$ actually have their intuitive meaning, plus interpretations for the additional individual and predicate constants, i.e., just labeled trees (cf. Rogers 1998 for details).

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

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

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

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 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.

(3) **Definition** (LIFT)**.** For each $n \geq 0$, $\Sigma'_n = \{f' | f \in \Sigma_n\}$ is a new set of symbols; $V_k = \{x_i \,|\, 1 \leq i \leq k\}$; for each $n \geq 1$ and each $i, 1 \geq i \geq n$, π^n_i is a new symbol, the ith projection symbol of sort n; for each $n \geq 0, k \geq 0$ the new symbol $c_{n,k}$ is the (n,k)th composition symbol. The set of all $c_{n,k}$ will be denoted by $\mathbf{C}$.

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

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

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

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

(4) **Definition** (Tree-Walking Automaton)**.** A *tree-walking automaton (with MSO tests)* is a finite automaton $\mathfrak{A} = (Q, \Delta, \delta, I, F)$ with states Q, directives Δ, transitions δ and the initial and final states $I \subseteq Q$ and $F \subseteq Q$ 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 $\mathfrak{A}$, which is a variation of the tree-walking automaton introduced in Aho and Ullman (1971), computes a node relation

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

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

3. VERB RAISING: TWO ROUTES AND A DETOUR

As mentioned in the introduction, this exercise in formal coding is made necessary by the fact that natural language sports at least some constructions which lead to (i) non-CF string languages, or (ii) to non-CF, i.e., non-recognizable structures, even though the resulting string languages are formally context-free. Both phenomena show up in the West-Germanic languages: the verbal complex of Züritüütsch is an example of (i), while (ii) is exhibited—for different reasons—by the corresponding constructions of Dutch and Standard German (Huybregts 1976, 1984):

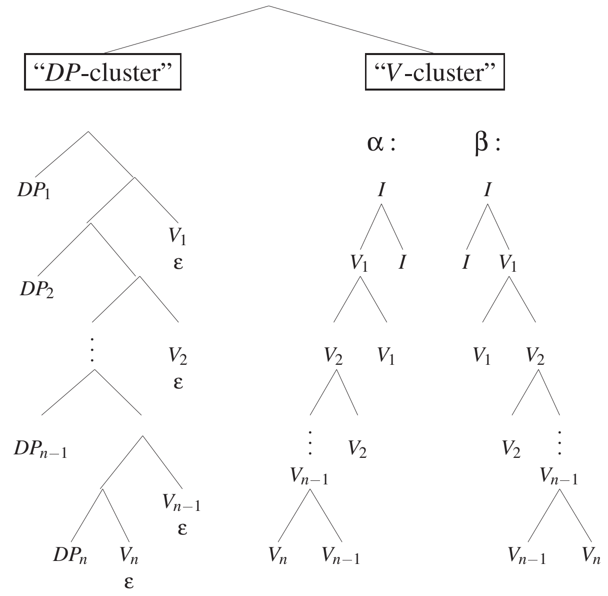
(6) a. *(... weil) der Karl die Maria dem Peter den Hans schwimmen lehren helfen lässt* (German: Palindrome language—CF)

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

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

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

b.



c.

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

FIGURE 1. The structure of Germanic VR

The standard analyses of these constructions involve one 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. 1 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 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-ATTRACT/MOVE machine, however, has little difficulty to combine nearness and separation in the required way. In a nutshell (and suppressing a plethora of important details), this machine is powered by just two operations: MERGE, combining two structures in a rather straightforward way creating a new node by “projecting” one of the input structures and installing both as its sole daughters, and ATTRACT/MOVE, which is dependent on the existence of a pair of attracting/attracted features in the structure so far constructed. It raises an element of the structure carrying the attracted feature and attaches it somewhere higher up in a *check*ing-configuration with the attracting feature, which is deleted in the process. Attracting features come in two flavors, *strong* and *weak*, inducing immediate or delayed movement, respectively. In our case, the moved element will always be adjoined to the element carrying the attracting feature. For details we refer the reader to Chomsky (1995), esp. ch. 4.

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

(7)
- VR is a lexical property of a certain class of verbs: *lassen-laten-laa* ("let"), *helfen-helpen-hälffe* ("help") … but not *schwimmen-zwemmen-schwüme* ("swim");
- VR-verbs take VP-complements;
- VR-verbs (optionally) have a strong V-feature, like the one standardly postulated for the inflectional head I;
- PF serializes complex V_0/I_0 as head-last (German) or head-first (Züritüütsch / Dutch).

The minimalist MERGE-ATTRACT/MOVE-CHECK machine achieves the complex result by a sequence of very simple discrete steps and so VR comes as close to being a consequence of formal universals as a basically idiosyncratic phenomenon will ever be able to do.

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

Consider the following macro-tree grammar for the verb-raising construction (6), with the lexicon, the obvious rules for Σ_1, and any structure under Σ_1-nodes omitted for legibility:

(8) Σ_0 (the Lexicon)
$\Sigma_1 = \{ V, V^{\varepsilon}, V_r, V_r^{\varepsilon}, DP, I \}$ $\quad \Sigma_2 = \{ proj, adj \}$
$\mathbf{F}_0 = \{ VR \}$ $\quad \mathbf{F}_2 = \{ +V \}$ $\quad S = VR$ $\quad \mathbf{V} = \{x, y\}$

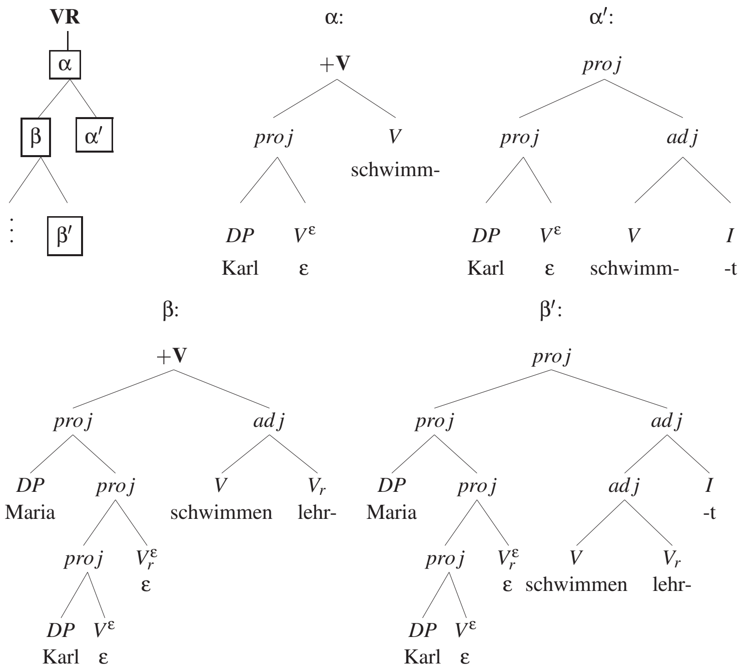


FIGURE 2. A (macro-)tree grammar derivation of VR

$$\mathbf{P} = \left\{ \begin{array}{rcl} VR & \longrightarrow & +V(proj(DP,V^{\varepsilon}),V) \\ +V(x,y) & \longrightarrow & proj(x,adj(y,I)) \\ +V(x,y) & \longrightarrow & +V(proj(DP,proj(x,V_r^{\varepsilon})),adj(y,V_r)) \end{array} \right\}$$

$proj(x,y)$ and $adj(x,y)$ are shorthands for Chomsky's (1995) *head-of-y*(x,y) and $[$*head-of-y*, *head-of-y*$](x,y)$, respectively. We assume that $adj(x,y)$ is "spelled-out" *xy* in German and *yx* in Dutch and Züritüütsch.

The minimalist and the tree grammar generated structures are identical (cf. the sample derivation in Fig. 2).

As indicated in the introduction any *context-free* tree grammar Γ can be LIFTed—in the sense of (3)—into a *regular* tree grammar Γ^L over a new, derived signature, which characterizes a (necessarily recognizable) set of trees encoding the instructions necessary to convert them into the ones the original grammar generates. Γ^L does not use variables at all, all nonterminals are null-ary and we have new alphabet symbols which can be interpreted as composition and projection "operations" by a tree homomorphism. So in a sense, Γ^L makes the compositions and projections involved in constructing the original trees $T(\Gamma)$ explicit.

Again simplifying things to the verge of inaccuracy—in particular, omitting all 0- and 1-place composition symbols, some book-keeping information on the composition symbols (the n,k), as well as a number of rules and symbols for the lexicon and Σ_1^L—, definition (3) yields the following LIFTed grammar for (8):

(9) $\Sigma_0^L = \Sigma_0$ $\quad \Sigma_1^L = \{ V', V^{\varepsilon\prime}, V_r', V_r^{\varepsilon\prime}, DP', I' \}$

$\Sigma_2^L = \{ proj', adj', \pi_1, \pi_2 \}$ $\quad \Sigma_{n,k}^L = \{ c_{n,k} \mid c \in \mathbf{C} \}$

$\mathbf{F}_0 = \{ VR', +V' \}$ $\quad S = VR'$ $\quad \mathbf{V} = \emptyset$

$$\mathbf{P} = \left\{ \begin{array}{rcl} VR' & \Longrightarrow & c(+V', c(proj', DP', V^{\varepsilon\prime}), V') \\ +V' & \Longrightarrow & c(proj', \pi_1, c(adj', \pi_2, I')) \\ +V' & \Longrightarrow & c(+V', c(proj', DP', c(proj', \pi_1, V_r^{\varepsilon\prime})), \\ & & \quad c(adj', \pi_2, V_r')) \end{array} \right\}$$

plus the rules for Σ_1^L

The corresponding structures are given in Fig. 3.

4. From Grammars to Logic

The standard procedure for the conversion of a grammar into an MSO formula given in the literature is via the creation of the corresponding tree automaton whose behaviour is then encoded (Thomas 1997). Instead of taking this indirect route, we propose in this paper to transform the grammar directly into an MSO formula. To our knowledge, this has not been presented elsewhere.

To achieve the coding we use MSO set variables to label nodes in the tree according to the labeling generated by the grammar and individual node variables to refer to the dominance relations introduced by the rules of the grammar. The MSO set variables will also be called features.

Given a lifted context-free tree grammar $\Gamma = \langle \Sigma', \mathbf{F}_0, S, \emptyset, \mathbf{P} \rangle$, we code it in the following way.

First of all, we need a theory of the sets labeling the nodes in our trees to make them behave as labels. All the definitions are relative to a finite, tree shaped set

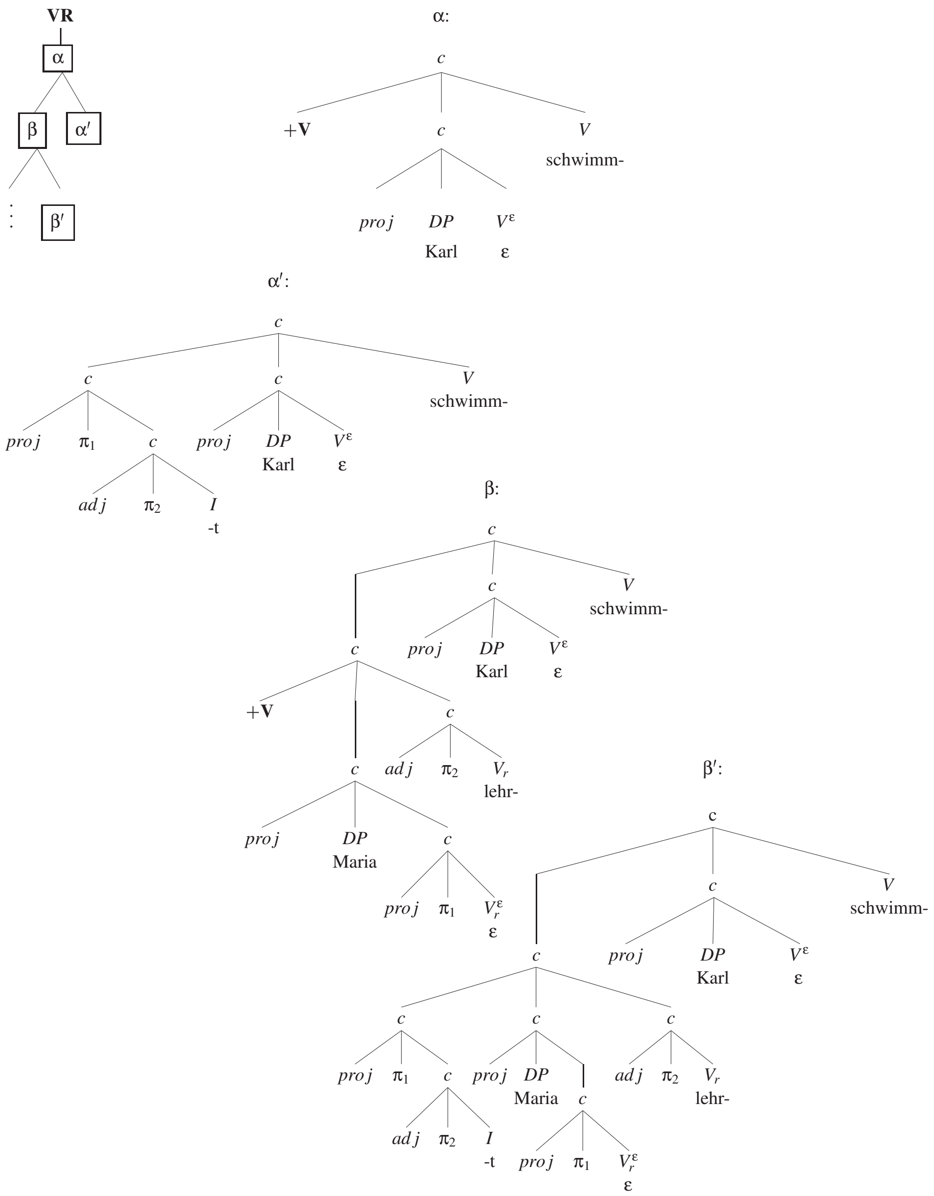


FIGURE 3. A sample derivation using the LIFTed grammar (9)

of nodes T which encodes the parse. The auxiliary predicate $\mathsf{Feature\_Theory}(T)$ encodes the following facts and notations:[1]

- Each lexical feature L_i, $1 \leq i \leq n$, $n = |\Sigma_0^L|$ denotes a lexical element of the signature. We collect them in an auxiliary predicate $\mathsf{L}(X) = \bigvee_{1 \leq i \leq n} X = L_i$.
- $\forall \Sigma_j^L \neq \emptyset$, each terminal feature T_i^j, $1 \leq i \leq n$, $n = |\Sigma_j^L|$, denotes a j-ary branching nonterminal (of the original, non-lifted grammar). Those are collected via the auxiliary predicate $\mathsf{T}(X) = \bigvee_{1 \leq i \leq n} X = T_i^j$.

[1] All auxiliary predicates we assume are given as explicit definitions in the appendix.

- Each nonterminal feature NT_i, $1 \leq i \leq n$, $n = |F_0|$ denotes a nonterminal (of the original, non-lifted grammar). A distinguished one of them corresponds to the start symbol of the grammar and will be referred to as S. The nonterminals are identified via the predicate $\mathsf{NT}(X) = \bigvee_{1 \leq i \leq n} X = NT_i$.
- Simplifying again, C stands for all nonterminals of the lifted grammar, i.e., the composition symbols.
- We will refer to the sets L_i, T_i^j and NT_i as features F_i if we mean any one of them and use the predicate $\mathsf{F}(X) = \mathsf{L}(X) \vee \mathsf{T}(X) \vee \mathsf{NT}(X)$ to talk about them.
- Any two features F_i and F_j, $i \neq j$, are constrained to be pairwise disjoint sets. Furthermore, the lexical and the terminal features are constrained to be pairwise disjoint from C;
- All features are subsets of the tree we are interested in, more specifically all F_i and also C are subsets of a set T standing for our parse tree which is constrained to be a rooted, connected set of nodes.
- All nodes are in T iff they are labeled by either one of the F_is or by C.
- Each T_i^j has exactly one daughter which is labeled with some L_k.
- And finally, each x in T which has no daughter (in T) is indeed labeled with exactly one of the L_i.

Furthermore, we have to encode the rules. Wlog, we may assume that the right hand sides (RHSs) of rules are of depth one. If this is not the case we algorithmically transform the grammar into the desired normal form by introducing extra nonterminals (Brainerd 1968).

Then each rule in $\mathbf{P}$ is of the form $f_0 \to c_{n,k}(f_1, \ldots, f_n)$, $f_0 \in \mathbf{F}_0$, $f_1, \ldots, f_n \in \Sigma_n^L \cup \mathbf{F}_0$. Assuming that the MSO variable NT_i stands for f_0, that for $1 \leq j \leq n$, F_j is the MSO variable encoding symbol f_j, that C denotes $c_{n,k}$ and that T is the parse tree, the formula encoding any such rule is given as follows:[2]

$$
\begin{aligned}
&\mathsf{Rule}_{NT_i}(T,x) \overset{def}{\Longleftrightarrow} \\
&\quad (NT_i(x) \wedge C(x)) \Longleftrightarrow \\
&\quad \bigvee_{f_0 \to c_{n,k}(f_1,\ldots,f_n) \in \mathbf{P}} (\exists x_1, \ldots, x_n \in T)[\bigwedge_{1 \leq j \leq n} (x \lhd x_j \wedge F_j(x_j)) \wedge \\
&\qquad (\neg \exists y \in T)[x \lhd y \wedge \bigwedge_{1 \leq j \leq n} y \neq x_j] \wedge \bigwedge_{1 \leq j,k \leq n, j \neq k} x_j \neq x_k \wedge \\
&\qquad \bigwedge_{1 \leq j,k \leq n, j < k} x_j \prec x_k]
\end{aligned}
$$

The mother node is labeled with a nonterminal iff there exist the appropriate daughters with the appropriate labels. Therefore we have to constrain the daughters to have the number, labeling and order enforced by the grammar rule. Additionally, the outermost disjunction in the formula treats multiple rules with the same LHS. For them, we have to create the disjunction of all the possible RHSs (for an example see the corresponding definition in the example translation of the VR grammar given in App. A.2).

[2]The predicate is indexed with the nonterminal of the LHS such that we can use this later on in a further definition.

All rules are then joined in a disjunction in the auxiliary predicate

$$\mathsf{Rule}(T,x) = \bigvee_{NT_i \in \mathbf{F}_0} \mathsf{Rule}_{NT_i}(T,x)$$

where x refers to the mother node and T to the parse tree.

We need two more auxiliary predicates: $\mathsf{RootedTree}(T,x)$ is defined such that T denotes the parse tree, i.e., a rooted, connected set and x is its root; and $\mathsf{Yield}(T,x)$ is true just in case x does not have any daughters in T.

Now we have all the ingredients to state the formula covering a lifted context-free tree grammar ($k = |\mathbf{F}_0|$):

$$\begin{aligned}(\exists_{1\le i\le k} NT_i)[(\exists r)[&\mathsf{RootedTree}(T,r) \land S(r)]\\ &\land (\forall x \in T)[\mathsf{Rule}(T,x) \lor \mathsf{Yield}(T,x)]\\ &\land \mathsf{Feature\_Theory}(T)]\end{aligned}$$

The formula is true just in case our feature theory is true, each node is either licensed by a rule or is on the frontier – and is labeled with some L_i which is enforced by the feature theory – and the root is labeled with the start symbol of the grammar. Since the nonterminals are not part of the original signature, i.e., we do not want to label the resulting trees with them, we bind them existentially at the top level of the grammar formula. Then the trees allowed by the formula are obviously identical to those generated by the lifted tree grammar.

5. Defining linguistic relations on lifted trees

Unfortunately, the terminal trees in Fig. 3, generated/denoted by (9) or its logical counterpart, don't seem to have much in common with the structures linguists want to talk about, i.e. the ones in Fig. 2.

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

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

To do so, it is helpful to note a few general facts (illustrated in Fig. 4):

(10) a. Our trees feature three families of labels: the "linguistic" symbols, i.e., the lifted in-operatives of the underlying macro-grammar, $\mathbf{L} = \textsc{lift}(\bigcup_{n\ge 0} \Sigma_n)$; the "composition" symbols $\mathbf{C} = \{c_{n,k}\}, n,k \ge 0$; and the "projection" symbols Π. Let $\Pi_i = \{\pi_i^k \mid k \ge 1\}$ and $\Pi = \bigcup_{i\ge 1} \Pi_i$. We add to $\mathbf{P}$ predicates denoting these families, i.e., $\mathbf{L}, \mathbf{C}, \Pi$, as well as their constituent sets, in particular all Π_i.

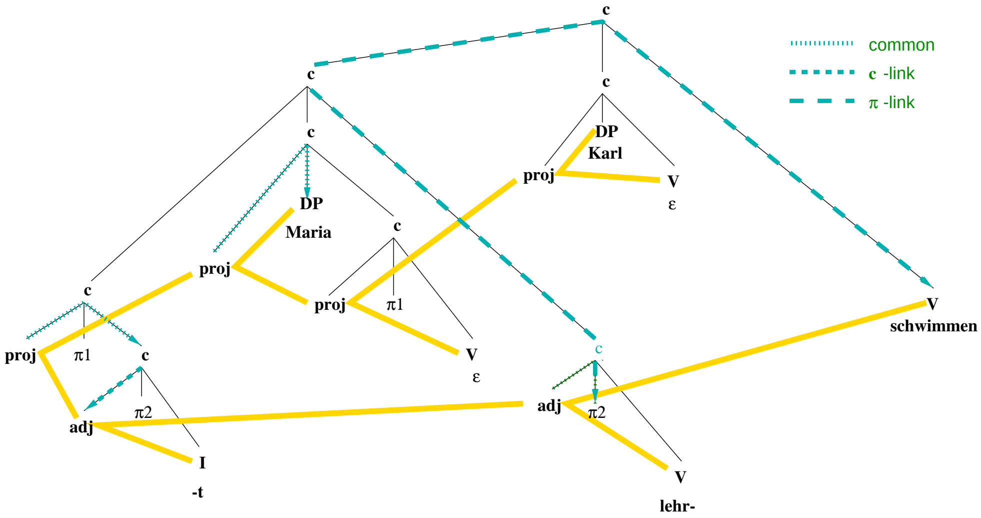


FIGURE 4. Intended relations on a LIFTed structure

b. All non-terminal nodes in T^L are labelled by some $c \in \mathbf{C}$. This is due to the fact that the "composition" symbols are the only non-terminals of a LIFTed grammar. No terminal node is labelled by some c.
c. The terminal nodes in T^L are either labelled by some "linguistic" symbol or by some "projection" symbol $\pi_i \in \Pi$.
d. Any "linguistic" node dominating anything in the intended tree is on some left branch in T^L, i.e., it is the left-most daughter^L of some $c \in \mathbf{C}$. This lies in the nature of composition: $c(x_0, x_1, \cdots, x_n)$ evaluates to $x_0(x_1, \cdots, x_n)$.
e. For any node p labelled with some "projection" symbol $\pi_i \in \Pi$ in T^L there is a unique node n (labelled with some $c \in \mathbf{C}$ by (2.)) which properly dominates p and whose i-th sister will eventually evaluate to the value of π. Moreover, n will be the first node properly dominating p which is on a left branch. This crucial fact is arrived at by easy induction on the construction of Γ^L from Γ.

By (10d) 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 $\triangleleft$. There are three cases to account for:

1. If the node considered carries a "linguistic" label, it evaluates to itself;
2. if it has a "composition" label c, it evaluates to whatever its functor—by (10d) its leftmost daughter—evaluates to;
3. if it carries a "projection" label π_i, it evaluates to whatever the node it "points to"—by (10e) the i^{th} sister of the first $\mathbf{C}$-node on a left branch dominating it—evaluates to.

Note that cases 2 and 3 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 $\blacktriangleleft^\star$.

For the latter case, a standard solution exists, at least in *weak MSO*, the monadic second-order logic on *finite* structures. It is a well-known fact (cf., 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:

(11) $R\text{-}closed(X) \equiv \forall x, y\,(x \in X \wedge R(x,y) \;\rightarrow\; y \in X)$

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

Then, assuming that $\blacktriangleleft$ is *MSO* definable, $\blacktriangleleft^\star$ and $\blacktriangleleft^+$ can be defined as follows:

(12) *Dominance:*

$$x \blacktriangleleft^\star y \equiv \forall X\,(\blacktriangleleft\text{-}closed(X) \wedge x \in X \;\rightarrow\; y \in X)$$

Proper dominance:

$$x \blacktriangleleft^+ y \equiv x \blacktriangleleft^\star y \wedge x \not\approx y$$

To ensure the *MSO* definability of $\blacktriangleleft$, we make use of a tree-walking automaton with *MSO* tests (Bloem and Engelfriet 1997). Actually, we can confine ourselves to a *basic* tree-walking automaton which restricts φ to simple tests of node labels.

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

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

(13)
$$Q = \{q_{in}, e, d, d_1, d_2, d_3, u, u_1, u_2, u_3, q_{fin}\} \qquad I = \{q_{in}\}$$
$$\Delta = \{\downarrow_1, \downarrow_2, \downarrow_3, \uparrow_1, \uparrow_2, \uparrow_3, \mathbf{L}(x), \mathbf{C}(x), \Pi_1(x), \Pi_2(x)\} \qquad F = \{q_{fin}\}$$
$$\begin{array}{ll} \delta = \{ & (q_{in} -\langle\, \mathbf{L}(x)\,\rangle\!\rightarrow u), \\ & (u -\langle \uparrow_1 \rangle\!\rightarrow d), (d -\langle \downarrow_2 \rangle\!\rightarrow e), (d -\langle \downarrow_3 \rangle\!\rightarrow e), \\ \text{case } \mathbf{L}: & (e -\langle\, \mathbf{L}(x)\,\rangle\!\rightarrow q_{fin}), \\ \text{case } \mathbf{C}: & (e -\langle\, \mathbf{C}(x)\,\rangle\!\rightarrow d_1), (d_1 -\langle \downarrow_1 \rangle\!\rightarrow e), \\ \text{case } \Pi_1: & (e -\langle\, \Pi_1(x)\,\rangle\!\rightarrow u_1), (u_1 -\langle \uparrow_2 \rangle\!\rightarrow u_1), (u_1 -\langle \uparrow_3 \rangle\!\rightarrow u_1), \\ & (u_1 -\langle \uparrow_1 \rangle\!\rightarrow d_2), (d_2 -\langle \downarrow_2 \rangle\!\rightarrow e), \\ \text{case } \Pi_2: & (e -\langle\, \Pi_2(x)\,\rangle\!\rightarrow u_2), (u_2 -\langle \uparrow_2 \rangle\!\rightarrow u_2), (u_2 -\langle \uparrow_3 \rangle\!\rightarrow u_2), \\ & (u_2 -\langle \uparrow_1 \rangle\!\rightarrow d_3), (d_3 -\langle \downarrow_3 \rangle\!\rightarrow e) \quad \} \end{array}$$

As indicated in section 2:(5), $\mathfrak{A}_\blacktriangleleft$ specifies, for any tree t, the node relation

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

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

However, there is yet 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 this is the language $W_{\triangleleft} = \mathbf{L}(x) \cdot \uparrow_1 \cdot (\downarrow_2 \cup \downarrow_3) \cdot (W_{\mathbf{C}} \cup W_{\Pi_1} \cup W_{\Pi_2})^* \cdot \mathbf{L}(x)$[3] with

(14)
$$\begin{aligned} W_{\mathbf{C}} &= \mathbf{C}(x) \cdot \downarrow_1 \\ W_{\Pi_1} &= \Pi_1(x) \ \cdot \ (\uparrow_2 \cup \uparrow_3)^* \cdot \uparrow_1 \cdot \downarrow_2 \\ W_{\Pi_1} &= \Pi_2(x) \ \cdot \ (\uparrow_2 \cup \uparrow_3)^* \cdot \uparrow_1 \cdot \downarrow_3 \end{aligned}$$

It is this walking language which is finally translated into a *MSO*-formula $trans_{W_{\triangleleft}}(x,y)$. The rather tedious process, carried through in appendix B, proceeds inductively via the following steps:

(15)
$$\begin{aligned} trans_{\emptyset}(x,y) &= \bot \\ trans_{\downarrow_i}(x,y) &= edge_i(x,y) \\ trans_{\uparrow_i}(x,y) &= edge_i(y,x) \\ trans_{\varphi(x)}(x,y) &= \varphi(x) \wedge x = y \\ trans_{W_1 \cup W_2}(x,y) &= trans_{W_1}(x,y) \vee trans_{W_2}(x,y) \\ trans_{W_1 \cdot W_2}(x,y) &= \exists z\, (trans_{W_1}(x,z) \wedge trans_{W_2}(z,y)) \\ trans_{W^*}(x,y) &= trans^*_W(x,y) \\ trans^*_W(x,y) &= \forall X\, (\forall v,w\, (v \in X \wedge trans_W(v,w) \ \rightarrow \ w \in X) \\ &\qquad \wedge x \in X \ \rightarrow \ y \in X) \end{aligned}$$

The resulting formula uses only the *MSO*-definable tests of the original automaton, the *closed* sets constructed via (11) for the Kleene-*-case, and the $edge_n$ relations (here with $1 \leq n \leq 3$) defined in appendix B. No recursion is involved: $\triangleleft$ is *MSO*-definable:

(16)
$$x \triangleleft y \equiv trans_{W_{\triangleleft}}(x,y)$$

This approach can be extended to the other members of the *dominance* family of relations: $\mathfrak{A}_{\triangleleft^+}$ is just $\mathfrak{A}_{\triangleleft}$ with the transition $(e -\langle\, \mathbf{L}(x) \,\rangle\!\!\rightarrow u)$ added to δ, and $\mathfrak{A}_{\triangleleft^*}$ is $\mathfrak{A}_{\triangleleft^+}$ with $F = \{u\}$, and q_{fin}, as well as any transition referring to it, removed from Q and δ, respectively. The inductive translations of the corresponding walking languages $trans_{W_{\triangleleft^*}}$ and $trans_{W_{\triangleleft^+}}$ into *MSO* provide an (equivalent) alternative to the definitions in (12).

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.

[3]Since Π-nodes are never leftmost daughters, the sublanguage $\mathbf{L}(x) \cdot \uparrow_1 \cdot (\downarrow_2 \cup \downarrow_3) \cdot (W_{\Pi_1} \cup W_{\Pi_2})^* \cdot W_{\mathbf{C}}{}^* \cdot \mathbf{L}(x)$ would be even more accurate for our purposes. The trivial changes to $A_{\triangleleft}$ 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. CONCLUSION

We have shown in the paper how to account for cross-serial dependencies by coupling a logical domain specification with a definition—expressed again in the same logical specification language—of linguistically significant relations on this domain. 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 logical procedure. The original context-free tree language is first translated into its explicit presentation. A corresponding closed *MSO* formula then isolates the explicit tree family within the realm of all possible finite trees on the LIFTed signature. The universal specification by means of an *MSO* string formula 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 logical domain specification followed by a universal description of the linguistically relevant structural relations.

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 logical 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 closed *MSO* expression together with a logical path description. 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.

Model sets specified by expressions of *MSO* on strings or trees possess the useful property that they are characterizable by regular string and tree grammars, respectively. This latter type of characterization has a distinct procedural flavour. The present logical description is completely independent of the grammatical formalism and provides a declarative delineation of the respective language classes along the methodological principles of descriptive complexity theory. A companion to this paper is in preparation (Kolb et al. 1999) in which we show that, based on the indicated equivalence between automata-theoretic and grammatical characterizations of regular languages, a purely procedural description of the phenomenon of cross-serial dependencies can be given as well.

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. Tech. Rep. 97-03, Dept. of Computer Science, Leiden University

Brainerd, W. S. (1968): The minimalization of tree automata. *Information and Control* 13(5):484–491

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*, Elsevier, vol. B, pp. 193–242

Courcelle, B. (1992): The monadic seond-order logics of graphs VII: Graphs as relational structures. *TCS* 102:3–33

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

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

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

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

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. (1999): Macros for Minimalism? Towards weak descriptions for strong structures. In: Kolb and Mönnich (1999), pp. 231–258

Kolb, H.-P. and U. Mönnich (eds.) (1999): *The Mathematics of Syntactic Structure: Trees and Their Logics*. Berlin: Mouton-de Gruyter

Kolb, H.-P., U. Mönnich, and F. Morawietz (1999): Regular description of cross-serial dependencies. Ms., SfS, Tübingen University

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

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

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, Cambridge University Press

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

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. (1997): Languages, automata, and logic. In: G. Rozenberg and A. Salomaa (eds.), *Handbook of Formal Languages*, Berlin: Springer, vol. 3: Beyond Words, pp. 389–455

APPENDIX A. *MSO*-VR

A.1. **Auxiliary Predicates.** Given a lifted context-free tree grammar $\Gamma = \langle \Sigma^L, \mathbf{F}_0, S, \emptyset, \mathbf{P} \rangle$ the following auxiliary predicates are defined:

$$
\begin{aligned}
\mathsf{L}(X) &\overset{def}{\iff} \bigvee_{0 \le i \le n} X = L_i \\
\mathsf{T}(X) &\overset{def}{\iff} \bigvee_{0 \le i \le n} X = T_i^j \\
\mathsf{NT}(X) &\overset{def}{\iff} \bigvee_{0 \le i \le n} X = NT_i \\
\mathsf{F}(X) &\overset{def}{\iff} \mathsf{L}(X) \vee \mathsf{T}(X) \vee \mathsf{NT}(X) \\
\mathsf{RootedTree}(T,x) &\overset{def}{\iff} \mathsf{Connected}(T) \wedge x \in T \wedge \mathsf{Root}(T,x) \\
\mathsf{Connected}(T) &\overset{def}{\iff} (\forall x,y,z)[(x \in T \wedge y \in T \wedge x \triangleleft^* z \wedge z \triangleleft^* y) \Rightarrow z \in T] \\
\mathsf{Root}(T,x) &\overset{def}{\iff} x \in T \wedge (\forall y \in T)[x \triangleleft^* y] \\
\mathsf{Yield}(T,x) &\overset{def}{\iff} x \in T \wedge (\neg \exists y \in T)[x \underset{\downarrow}{y}] \\
\mathsf{Disjoint}(X,Y) &\overset{def}{\iff} (\neg \exists x)[X(x) \wedge Y(x)] \\
\mathsf{Feature\_Theory}(T) &\overset{def}{\iff} \\
&\quad C \subseteq T \\
&\quad \wedge (\forall X)[(\mathsf{F}(X) \Rightarrow X \subseteq T] \\
&\quad \wedge (\forall X)[(\mathsf{T}(X) \vee \mathsf{L}(X)) \Rightarrow \mathsf{Disjoint}(X,C)] \\
&\quad \wedge (\forall X,Y)[(\mathsf{F}(X) \wedge \mathsf{F}(Y)) \Rightarrow \mathsf{Disjoint}(X,Y)] \\
&\quad \wedge (\forall x \in T)[(\mathsf{F}(x) \vee C(x))] \\
&\quad \wedge (\forall X)[(\mathsf{T}(X) \wedge x \in X) \Rightarrow \\
&\qquad\qquad (\neg \exists y \in T)[x \triangleleft y \wedge (\exists Y)[(\mathsf{T}(Y) \vee \mathsf{NT}(Y)) \wedge Y(y)]]] \\
&\quad \wedge (\forall x \in T)[\mathsf{Yield}(T,x) \Rightarrow (\exists Y!)[\mathsf{L}(Y) \wedge Y(x)]]]
\end{aligned}
$$

To facilitate the presentation of the translated rules, we will use a further auxiliary predicate $\mathsf{ThreeDaughters}(x,a,b,c,T)$ defined such that x is the mother of exactly three daughters a,b,c, a,b,c appear in the given order and all nodes are in T:

$$
\begin{aligned}
&\mathsf{ThreeDaughters}(x,a,b,c,T) \overset{def}{\iff} \\
&\qquad x \in T \wedge a \in T \wedge b \in T \wedge c \in T \wedge x \triangleleft a \wedge x \triangleleft b \wedge x \triangleleft c \\
&\qquad \wedge (\neg \exists y \in T)[x \triangleleft y \wedge y \neq a \wedge y \neq b \wedge y \neq c] \\
&\qquad \wedge a \neq b \wedge b \neq c \wedge c \neq a \wedge a \prec b \wedge b \prec c
\end{aligned}
$$

A.2. **The formula for the lifted VR grammar.** In this example we did not transform the grammar to a normal form with only RHSs of depth one. Instead, we altered the grammar formula in such a way that we allow nodes which are not themselves licensed by a rule but where either their mother or their grandmother is licensed to account for the rules of depth two and three which occur in the example. This turned out to be simpler than having to deal with more nonterminals. But the presentation of a general translation to an MSO formula gets very confusing if we allow RHSs of arbitrary depths.

$$\begin{aligned}(\exists VR', +V', r)[&\mathsf{RootedTree}(X,r) \wedge VR'(r)\\ &\wedge (\forall x \in X)[\mathsf{rule}(X,x) \vee (\exists y \in X)[y \lhd x \wedge \mathsf{rule}(X,y)]\\ &\qquad \vee (\exists y,z \in X)[y \lhd x \wedge z \lhd y \wedge \mathsf{rule}(X,z)] \vee \mathsf{Yield}(X,x)]\\ &\wedge \mathsf{Feature\_Theory}(X)\end{aligned}$$

We give here only the rules for the nonterminals VR' and $+V'$. The other ones are left as an exercise to the reader. Note that there are two rules with $+V'$ on the LHS such that the disjunct encoding these rules contains itself an disjunction of the two RHSs.

$$\begin{aligned}&\mathsf{Rule}(T,x) \overset{def}{\Longleftrightarrow}\\ &\quad (C(x) \wedge VR'(x) \Longleftrightarrow\\ &\quad (\exists u,v,w)[\mathsf{ThreeDaughters}(x,u,v,w,T) \wedge +V'(u) \wedge C(v) \wedge V'(w) \wedge\\ &\qquad (\exists r,s,t \in T)[\mathsf{ThreeDaughters}(v,r,s,t,T) \wedge Proj'(r) \wedge DP'(s) \wedge V^{\varepsilon\prime}(t)]])\\ &\vee (C(x) \wedge +V'(x) \Longleftrightarrow\\ &\quad (\exists u,v,w)[\mathsf{ThreeDaughters}(x,u,v,w,T) \wedge Proj'(u) \wedge \Pi_1(v) \wedge C(w) \wedge\\ &\qquad (\exists r,s,t)[\mathsf{ThreeDaughters}(w,r,s,t,T) \wedge Adj'(r) \wedge \Pi_2(s) \wedge I'(t)]]\\ &\quad \vee\\ &\quad (\exists u,v,w \in T)[\mathsf{ThreeDaughters}(x,u,v,w,T) \wedge +V'(u) \wedge C(v) \wedge C(w) \wedge\\ &\qquad (\exists o,p,q)[\mathsf{ThreeDaughters}(v,o,p,q,T) \wedge Proj'(o) \wedge DP'(p) \wedge C(q) \wedge\\ &\qquad\quad (\exists a,b,c)[\mathsf{ThreeDaughters}(q,a,b,c,T) \wedge Proj'(a) \wedge \Pi_1(b) \wedge V_r^{\varepsilon\prime}(c)]] \wedge\\ &\qquad (\exists r,s,t)[\mathsf{ThreeDaughters}(w,r,s,t,T) \wedge Adj'(r) \wedge \Pi_2'(s) \wedge V_r'(t)]])\\ &\vee \dots\end{aligned}$$

What remains is only to define the particular lexical entries, terminals and nonterminals.

$$\begin{aligned}&\mathsf{L}(X) \overset{def}{\Longleftrightarrow} (X = Karl) \vee (X = Maria) \vee (X = Peter) \vee (X = Hans) \vee\\ &\qquad (X = schwimmen) \vee (X = lehren) \vee (X = helfen) \vee (X = l\ddot{a}sst)\\ &\mathsf{T}(X) \overset{def}{\Longleftrightarrow} (X = Proj') \vee (X = Adj') \vee (X = \Pi_1') \vee (X = \Pi_2') \vee\\ &\qquad (X = V') \vee (X = V^{\varepsilon\prime}) \vee (X = V_r') \vee (X = V_r^{\varepsilon\prime}) \vee (X = DP') \vee (X = I')\\ &\mathsf{NT}(X) \overset{def}{\Longleftrightarrow} (X = VR') \vee (X = +V')\end{aligned}$$

APPENDIX B. *MSO*-$\blacktriangleleft$

The translation via (15) of the walking language $W_{\blacktriangleleft}$ derived from the tree-walking automaton $\mathfrak{A}_{\blacktriangleleft}$ (13) yields the following *MSO*-definition of $\blacktriangleleft_W$ for binary branching intended trees and underlying grammars with macros with at most 2 parameters:

(17)
$$\begin{aligned} x \blacktriangleleft_W y \equiv\ & trans_{W_{\blacktriangleleft}}(x,y) \\ \equiv\ & lexical(x) \wedge \exists z(edge_1(z,x) \wedge \exists u((edge_2(z,u) \vee edge_3(z,u)) \wedge \\ & \forall X(\forall v,w((v \in X \wedge (c\text{-}link(v,w) \vee (\pi_1\text{-}link(v,w) \vee \pi_2\text{-}link(v,w))) \\ & \to w \in X) \wedge (u \in X \to y \in X))) \wedge lexical(y))). \end{aligned}$$

(18)
$$c\text{-}link(x,y) \equiv C(x) \wedge edge_1(x,y).$$

(19)
$$\begin{aligned} \pi_1\text{-}link(x,y) \equiv\ & \Pi_1(x) \wedge \exists z(\forall X(\forall u,v((u \in X \wedge (edge_2(v,u) \vee edge_3(v,u)) \to v \in X) \\ & \wedge (x \in X \to z \in X))) \wedge \exists w(edge_1(w,z) \wedge edge_2(w,y))). \\ \pi_2\text{-}link(x,y) \equiv\ & \Pi_2(x) \wedge \exists z(\forall X(\forall [u,v]((u \in X \wedge (edge_2(v,u) \vee edge_3(v,u)) \to v \in X) \\ & \wedge (x \in X \to z \in X))) \wedge \exists w(edge_1(w,z) \wedge edge_3(w,y))). \end{aligned}$$

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

(20)
$$\begin{aligned} edge_n(x,y) \equiv\ & \exists x_1,\cdots,x_{n-1}(x \lhd_{\mathrm{L}} x_1 \wedge \cdots \wedge x \lhd_{\mathrm{L}} x_{n-1} \wedge x \lhd_{\mathrm{L}} y \\ & \wedge x_1 \prec_{\mathrm{L}} x_2 \wedge \cdots \wedge x_{n-1} \prec_{\mathrm{L}} y \wedge\ \forall w(x \lhd_{\mathrm{L}} w \\ & \wedge w \not\approx x_1 \wedge \cdots \wedge w \not\approx x_{n-1} \wedge w \not\approx y\ .\to .\ y \prec_{\mathrm{L}} w)) \end{aligned}$$

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 (17) and (19), again introducing a definition scheme in the case of (19'):

(17')
$$\begin{aligned} x \blacktriangleleft y \equiv\ & lexical(x) \wedge \exists z(edge_1(z,x) \wedge \exists u((\bigvee_{1<n<\omega} edge_n(z,u)) \wedge \\ & \forall X(\forall v,w((v \in X \wedge (c\text{-}link(v,w) \vee (\bigvee_{1<i<\omega} \pi_i\text{-}link(v,w))) \\ & \to w \in X) \wedge (u \in X \to y \in X))) \wedge lexical(y))). \end{aligned}$$

(19')
$$\begin{aligned} \pi_i\text{-}link(x,y) \equiv\ & \Pi_i(x) \wedge \exists z(\forall X(\forall u,v((u \in X \wedge (\bigvee_{1<n<\omega} edge_n(v,u)) \to v \in X) \\ & \wedge (x \in X \to z \in X))) \wedge \exists w(edge_1(w,z) \wedge edge_{i+1}(w,y))). \end{aligned}$$