A Grammar Formalism and Cross-Serial Dependencies*

Tore Burheim
Department of Informatics
University of Bergen
Tore.Burheim@ii.uib.no

Mars 15, 1995

Abstract

First we define a unification grammar formalism. It is based on Lexical Functional Grammar (LFG), but has a strong restriction on the syntax of the equations. We then show that this grammar formalism defines a full abstract family of languages, and that it is capable of describing cross-serial dependencies of the type found in Swiss German.

1 Introduction

Due to their combination of simplicity and flexibility unification grammars have become widely used in computational linguistics the last fifteen yeas. But this flexibility gives us also a very powerful formalism. As a result of this power, the membership problem for unification grammars in their most general form is undecidable. Therefore most of those grammars have restrictions to make them decidable, e.g. the *off-line parsability constraint* in LFG [12]. Nevertheless, the membership problem is NP-complete [1] or harder for most unification grammar formalisms. It is therefore interesting to study restrictions on unification grammars. Most of such studies have been concerned with making the formalism decidable [10, 11]. But there has also been work on formalisms for which the membership problem can be decided in polynomial time. In their work, Keller and Weir [14] defines a grammar formalism with feature structures for which the membership problem can be decided in polynomial time. Here you do not have a common feature structure for the sentence as a hole, but feature structures annotated to the nodes in a phrase structure tree, and limited possibilities to share information. In this paper we will study a formalism that lies somewhere in between the most powerful

*The final preparation of this paper has been supported by grant 100437/410 from the Norwegian Research Council.

formalisms and this limited one. The grammar formalism that we define is based on work by Colban [4].

What we here call unification grammars are also called attribute-value grammars, feature-structure grammars and constraint-based grammars. You may divide them in two major groups, one that is based on a phrase-structure backbone as LFG and PATR, and one that is entirely described in a feature structure framework as HPSG [16]. We will here use a phrase structure backbone and add equations to the nodes in the phrase-structure tree as in LFG. These equations will describe feature structures. With a restriction that we will impose on the equations, the feature structures will be trees that is homomorphic with the phrase structure tree. But before we go into any details, we have to define feature structures and how we describe them with equations.

2 Feature structures

One of the main characteristics of unification grammars is that they are information based. This information is inductively collected from the sentences sub-strings, sub-sub-strings and so on. We will use feature structures to represent this information. There are many ways of viewing, defining and describing feature structures, e.g. as directed acyclic graphs [18], as finite deterministic automata [13], as models for first order logic [19, 20, 9], or as Kripke frames for modal logic [2]. Here we use a slightly modified version of Kasper and Rounds [13] definition of feature structures, and we will later use a subset of the equations schemata used by LFG to describe these feature structures. As a basis we assume two predefined sets, one of attribute symbols and one of value symbols. In a linguistic framework these sets will typical include things like *subject, object, number, case* etc. as attribute symbols, and *singular, plural, dative, accusative* etc. as value symbols.

Definition 1 *A* feature structure *M over the set of attribute symbols A and value symbols V is is a 4-tuple $\langle Q, f_D, \delta_0, \alpha\rangle$ where*

- *Q is a finite set of* nodes*,*
- *$f_D : D \to Q$ is a function, called the* name mapping function*, where D is a finite set of names,*
- *$\delta_0 : Q \times A \to Q$ is a partial function, called the* transition function*,*
- *$\alpha : Q \to V$ is a partial function called the* atomic value function*.*

We extend the transition function δ_0 to be a partial function $\delta : (Q \times A^*) \to Q$ as follows: 1) *For every $q \in Q$, $\delta(q, \varepsilon) = q$,* 2) *If $\delta(q_1, w) = q_2$ and $\delta_0(q_2, a) = q_3$ then $\delta(q_1, wa) = q_3$ for every $q_1, q_2, q_3 \in Q$, $w \in A^*$ and $a \in A$. A feature structure is* well defined *if it is*

- atomic: *For all $q \in Q$, if $\alpha(q)$ is defined, then $\delta_0(q, a)$ is not defined for any $a \in A$.*

- acyclic: *For all* $q \in Q$, $\delta(q, w) = q$ *if and only if* $w = \varepsilon$.
- describable: *For all* $q \in Q$ *there exists an* $x \in D$ *and a* $w \in A^*$ *such that* $\delta(f(x), w) = q$

All feature structures are required to be well defined.

We use equations to describe feature structures, such that a set of equations describes the least feature structure that satisfies all the equations in the set. A feature structure *satisfies* the equation

$$x_1 w_1 = x_2 w_2 \tag{1}$$

if and only if $\delta(f(x_1), w_1) = \delta(f(x_2), w_2)$, and the equation

$$x_3 w_3 = v \tag{2}$$

if and only if $\alpha(\delta(f(x_3), w_3)) = v$, where $x_1, x_2, x_3 \in D$, $w_1, w_2, w_3 \in A^*$ and $v \in V$. We only allow equations on those two forms, $x_1 w_1 = x_2 w_2$ and $x_3 w_3 = v$.

If E is a set of equations and M is a well defined feature structure such that M satisfies every equation in E then we say that M *supports* E and we write

$$M \models E \tag{3}$$

If M_1 and M_2 are feature structures, we say that M_1 *subsumes* M_2, written $M_1 \sqsubseteq M_2$, if and only if for every set of equations E, if $M_1 \models E$ then $M_2 \models E$. If M_1 subsumes M_2 then M_2 contains all the information that M_1 contains. Two feature structures M_1 and M_2, are *equivalent* if and only if $M_1 \sqsubseteq M_2$ and $M_2 \sqsubseteq M_1$. This means that they contain the same information. Given a set of equations E, we say that E *describes* a feature structure M if and only if $M \models E$ and for every feature structure M', if $M' \models E$ then $M \sqsubseteq M'$. A given set of equations describes different, but equivalent feature structures. If E describes a feature structure M we write this

$$E \gg M \tag{4}$$

We here describe feature structures without using unification. In our very simple way of defining and describing feature structures this is only a matter of taste and unification is just another approach to the same kind of information collecting. To see this, we define unification in the usual way: Let M_1 and M_2 be two well defined feature structures. Then $(M_1 \sqcup M_2)$ is a feature structure such that $M_1 \sqsubseteq (M_1 \sqcup M_2)$, $M_2 \sqsubseteq (M_1 \sqcup M_2)$ and for every M' such that $M_1 \sqsubseteq M'$ and $M_2 \sqsubseteq M'$, $(M_1 \sqcup M_2) \sqsubseteq M'$. From the definition we then get that if $E_1 \gg M_1$ and $E_2 \gg M_2$ then $(E_1 \cup E_2) \gg M_1 \sqcup M_2$. Instead of using unification directly we collect equations and see if all the equations together describe a feature structure.

A set of equations E is *consistent* if there exists a well defined feature structure that E describes. It is possible that an equation set does not describes any well defined feature structure. We then say that the equation set is *inconsistent*. This happens for instance if the equation set contains both the equations $ea = v$ and $eaa = v$ for a value symbol v. A feature structure that satisfies those two equations cannot be atomic.

3 The grammar formalism

The Tree Homomorphic Feature Structure grammars[1] (THFSG-grammars) are based on LFG [12], but is much simplified. The main difference is that we have a strong restriction on the equation sets, we treat the lexical items in almost the same way as production rules, and we do not have the completeness and coherence constraints or anything like the functional uncertainty mechanism. This grammar formalism is very much like the grammar formalism $GF1$ defined by Colban [4] but there is one main difference; we accept empty right hand sides in the lexicon rules. This gives us the ability to describe a full abstract family of languages which $GF1$ does not [3]. In LFG without functional uncertainty empty right hand sides are used in the analysis of long-distance dependencies. In addition $GF1$ only accepts equations on the format that THFSG-grammars have in their normal form.

Definition 2 *A* Tree Homomorphic Feature Structure Grammar (THFSG) *is a 5-tuple* $\langle \mathcal{K}, \mathcal{S}, \Sigma, \mathcal{P}, \mathcal{L} \rangle$ *over the set of attribute symbols* A *and value symbols* V *where*

- $\mathcal{K}$ *is a finite set of symbols, called categories,*
- $\mathcal{S} \in \mathcal{K}$ *is a symbol, called start symbol,*
- Σ *is a finite set of symbols, called terminals,*
- $\mathcal{P}$ *is a finite set of production rules*

$$\begin{array}{ccccc} K_0 & \rightarrow & K_1 & \dots & K_m \\ & & E_1 & & E_m \end{array} \tag{5}$$

  where $m \geq 1$, $K_0, ..., K_m \in \mathcal{K}$, *and for all* i, $1 \leq i \leq m$, E_i *is a finite set with one and only one equation on the form* $\uparrow a_1...a_n =\downarrow$ *where* $n \geq 0$ *and* $a_1, ..., a_n \in A$, *and a finite number of equations on the form* $\uparrow a_1...a_n = v$ *where* $n \geq 1$, $a_1, ..., a_n \in A$ *and* $v \in V$.

- $\mathcal{L}$ *is a finite set of lexicon rules*

$$\begin{array}{ccc} K & \rightarrow & t \\ & & E \end{array} \tag{6}$$

  where $K \in \mathcal{K}$, $t \in (\Sigma \cup \{\varepsilon\})$, *and* E *is a finite set of equations on the form* $\uparrow a_1...a_n = v$ *where* $n \geq 1$, $a_1, ..., a_n \in A$ *and* $v \in V$.

We see that each equation set in the production rules includes one and only one equation with both up and down arrows. In this equations we only allow attribute

[1] This grammar formalisms is part of a hierarchy of grammar formalisms based on different equation formats and definitions of grammatical strings described in [3]. What we here call THFSG is there named $RS_1\&T_0$. Among the other formalisms is $RS_0\&T_0$, which has an undecidable membership question, and $RS_1\&T_2$ for which we can decide membership in time $\mathcal{O}(n^3)$ and which in fact describes the class of context-free languages.

symbols on the left hand side, and none on the right hand side. As a result of this we will later see that the feature structure will be a tree that is homomorphic with the parsing tree or c-structure as we will call it. But first we shall define which strings that are grammatical with respect to this grammar.

To define the constituent structures we use tree domains: Let $\mathcal{N}_+$ be the set of all integers greater than zero. A *tree domain* D is a set $D \subseteq \mathcal{N}_+^*$ of number strings so that if $x \in D$ then all prefixes of x are also in D, and for all $i \in \mathcal{N}_+$ and $x \in \mathcal{N}_+^*$, if $xi \in D$ then $xj \in D$ for all j, $1 \leq j < i$. The *out degree* $d(x)$ of an element x in a tree domain D is the cardinality of the set $\{i \mid xi \in D, i \in \mathcal{N}_+\}$. The set of terminals of D is $term(D) = \{x \mid x \in D, d(x) = 0\}$. The elements of a tree domain are totally ordered lexicographically as follows: $u \prec x$ if u is a prefix of x, or there exist strings $y, z, z' \in \mathcal{N}_+^*$ and $i, j \in \mathcal{N}_+$ with $i < j$, such that $u = yiz$ and $x = yjz'$.

A tree domain D can be viewed as a tree graph in the following way: The elements of D are the nodes, ε is the root, and for every $x \in D$ the element $xi \in D$ is x's child number i. The terminals of D are then the terminal nodes in the tree.

A tree domain describes the topology of a parsing tree. This representation provides a name for every node in the parsing tree, directly from the definition of a tree domain. A tree domain may be infinite, but we restrict the attention to finite tree domains[2].

Definition 3 *A* constituent structure *(c-structure) based on a* THFSG-*grammar* $G = \langle \mathcal{K}, \mathcal{S}, \Sigma, \mathcal{P}, \mathcal{L} \rangle$ *is a triple* $c_G = \langle D, K, E \rangle$ *where*

- D *is a finite tree domain,*
- $K : D \rightarrow (\mathcal{K} \cup \Sigma \cup \{\varepsilon\})$ *is a function,*
- $E : (D - \{\varepsilon\}) \rightarrow \Gamma$ *is a function where* Γ *is the set of all equation sets in* G*,*

such that $K(x) \in (\Sigma \cup \{\varepsilon\})$ *for all* $x \in term(D)$*,* $K(\varepsilon) = \mathcal{S}$*, and for all* $x \in (D - term(D))$*, if* $d(x) = m$ *then*

$$\begin{array}{ccccc} K(x) & \rightarrow & K(x1) & \ldots & K(xm) \\ & & E(x1) & & E(xm) \end{array} \tag{7}$$

is a production or lexicon rule in G*.*

The terminal string of a constituent structure is the string $K(x_1)...K(x_n)$ *such that* $\{x_1, ..., x_n\} = term(D)$ *and* $x_i \prec x_{i+1}$ *for all* i*,* $1 \leq i < n$*.*

We see that the terminal string is a string in Σ^* since $K(x) \in (\Sigma \cup \{\varepsilon\})$ for all $x \in term(D)$. To get equations that can be used to describe feature structures we must instantiate the up and down arrows in the equations from the production and lexicon rules. We substitute them with nodes from the c-structure. For this purpose we define the $'$-function such that $E'(xi) = E(xi)(x/\uparrow, xi/\downarrow)$. We see that the value of the function E' is a set of equations that feature structures may support.

[2]See Gallier [5] for more about tree domains.

Definition 4 *The c-structure $c_G = \langle D, K, E\rangle$* generates *the feature structure M if and only if*

$$\bigcup_{x\in(D-\{\varepsilon\})} E'(x) \gg M \tag{8}$$

A c-structure is consistent *if it generates a feature structure.*

The nonterminal part of tree domain will form the name set for feature structures that this union describes. A c-structure is consistent if this union is consistent and a string is grammatical if its c-structure is consistent.

Definition 5 *Let $G = \langle \mathcal{K}, \mathcal{S}, \Sigma, \mathcal{P}, \mathcal{L}\rangle$ be a* THFSG*-grammar. A string $w \in \Sigma^*$ is* grammatical *with respect to G if and only if there exists a consistent c-structure c_G with w as the terminal string.*

The set of all grammatical strings[3] with respect to a grammar G are denoted $L(G)$ and is the language that the grammar G generates. Two grammars G and G' is *equivalent* if $L(G) = L(G')$.

In this grammar formalism we allow one and only one equation with both up and down arrows in each equation set in the production rules. In this equation we only allow attribute symbols on the left hand side, –none on the right hand side. As a result of this the feature structures will be trees and the domination relation in the c-structure is preserved in the feature structure [4]. The domination relation must not be confused with the lexicographical ordering of the nodes in the c-structure, so let us define the domination relation on the c-structure and the feature structure: For all the nodes x in the tree domain D of a c-structure $x' \leq_c x$ for all prefixes x' of x. In the feature structure, let $q' \leq_M q$ for all nodes such that $\delta(q', w) = q$ for a $w \in A^*$. Then for any c-structure $\langle D, K, E\rangle$ which generates a feature structure M

$$x' \leq_c x \;\Rightarrow\; f(x') \leq_M f(x) \tag{9}$$

Then the name function $f : D \to Q$ is a homomorphism between the node sets with the domination relation of those two structures [4].

We close this presentation of THFSG-grammars by defining a *normal form*:

Definition 6 *A* THFSG*-grammar $G = \langle \mathcal{K}, \mathcal{S}, \Sigma, \mathcal{P}, \mathcal{L}\rangle$ is on normal form if each production rule in $\mathcal{P}$ is on the form*

$$\begin{array}{ccc} K_0 & \to & K_1 \quad K_2 \\ & & E_1 \quad E_2 \end{array} \tag{10}$$

where $K_1, K_2, K_3 \in \mathcal{K}$, and each equation set, E_1 and E_2, is a finite set with one and only one equation on the form $\uparrow a =\downarrow$ or $\uparrow=\downarrow$ where $a \in A$, and a finite number of equations on the form $\uparrow a_1 ... a_n = v$ where $n \geq 1$, $a_1, ..., a_n \in A$ and $v \in V$.

[3] We may have different definitions of which strings that are grammatical. If we for grammars on normal form (see below) add the constraint that for any two nodes x and y in the c-structure , if $\delta(f(x), w) = f(y)$ then $f(x) = f(x')$ where x' is the greatest common prefix of x and y, or in other words the closest common predecessor, then we get a grammar formalism that describes the class of context-free languages [3, 4].

We see that a THFSG-grammar is on normal form if every production rule has exactly two elements on the right hand side and the equations with both up and down arrows have no more than one attribute symbol.

Lemma 1 *For every* THFSG*-grammar there exist a equivalent* THFSG*-grammar on normal form.*

Proof: We show how to construct a THFSG-grammar on normal form G' for any THFSG-grammar G such that $L(G) = L(G')$. There are two constraints for grammars on normal form, one on the equations, and one on the format of the production rules. First we show how to get the equations right.

For each equation set E_i with an equation $\uparrow a_1 ... a_n =\downarrow$ where $n > 1$ in each production rule

$$\begin{array}{ccccccc} K_0 & \rightarrow & K_1 & ... & K_i & ... & K_m \\ & & E_1 & & E_i & & E_m \end{array} \tag{11}$$

we replace K_i with a unique new category $K'_{i,1}$ and E_i with the set $E'_i = (E_i - \{\uparrow a_1 ... a_n =\downarrow\}) \cup \{\uparrow a_1 =\downarrow\}$, and add new production rules:

$$\begin{array}{ccc} K'_{i,(j-1)} & \rightarrow & K'_{i,j} \\ & & \uparrow a_j =\downarrow \end{array} \tag{12}$$

for all j, $2 \leq j \leq n$ where $K'_{i,2} ... K'_{i,n-1}$ are unique new categories and $K'_{i,n} = K_i$. Now each equation set in each production rule is as required for the normal form.

To get only two elements on the right hand side in each production rule we apply versions of the techniques from the standard proof for Chomsky normal form for context-free grammars [8, page 92-93]. Since we allow the empty string in lexicon rules, this is straightforward.

Now we have a grammar on normal form, and it is easy to see that we have a consistent c-structure for a string based on the original grammar if and only if we have a consistent c-structure for the same string based on the grammar on normal form. This due to the fact that we do not add any new information in the equations. Then $L(G) = L(G')$. □

4 Full abstract family of languages

When studying formal grammars we often want to study the class of languages that a grammar formalism defines. A *class of languages*, $\mathcal{C}_\Gamma$ over a countable set Γ of symbols is a set of languages, such that for each language $L \in \mathcal{C}_\Gamma$ there exist a finite subset Σ of Γ such that $L \subseteq \Sigma^*$. The class $\mathcal{C}_\Gamma(GF)$ of languages that a grammar formalism GF defines is the set of all languages L' over Γ such that there exists a grammar G in GF such that $L(G) = L'$.

For a given countably infinite Γ an uncountable number of different classes of languages exist. Some of them are more natural and well-behaved than others, and of particular interest are the *full abstract families of languages* (full AFL). A full AFL is a class of languages closed under union, concatenation, Kleene closure, intersection with regular languages, string homomorphism and inverse string homomorphism[4]. The class of regular languages and context-free languages are both full AFL, but the class of context-sensitive languages is not. Here we show that the class of languages that the grammar formalism THFSG defines[5], $\mathcal{C}(\text{THFSG})$ is a full abstract family of languages. But first we need a precise definition of full abstract families of languages.

A string homomorphism is a function $h : \Delta^* \to \Sigma^*$ such that for every $w \in \Delta^*$ and $a \in \Delta$ we have

$$\begin{aligned} h(\varepsilon) &= \varepsilon && (13) \\ h(aw) &= h(a)h(w) && (14) \end{aligned}$$

A string homomorphic image of a language $L \subseteq \Delta^*$ for a string homomorphism $h : \Delta^* \to \Sigma^*$ is the language $\{h(w) \mid w \in L\}$. The inverse string homomorphic image of a language $L' \subseteq \Sigma^*$ is the language $\{w \mid h(w) \in L'\}$. The concatenation of two languages L_1 and L_2 is the language $\{w_1 w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2\}$. The Kleene closure of a language L is the language $\{w_1 \ldots w_n \mid n \geq 0 \wedge w_1, \ldots, w_n \in L\}$. Union and intersection are the standard set-theoretic operations.

Lemma 2 $\mathcal{C}(\text{THFSG})$ *is closed under union, concatenation and Kleene-closure.*

The proof of this is straightforward [3] and very much like the proof for the same properties for context-free grammars [15, pages 120–121].

To show that $\mathcal{C}(\text{THFSG})$ is closed under intersection with regular languages, string homomorphism and inverse string homomorphism we show that $\mathcal{C}(\text{THFSG})$ is closed under NFT-mapping. Informally, a *Nondeterministic Finite Transducer* (NFT) is a nondeterministic finite state machine with an additional write tape. In addition to just reading symbols and changing states, an NFT also writes symbols on the write tape. It may write symbols and change states when reading the empty string. Formally, an NFT is a 6-tuple $M = \langle Q, \Delta, \Sigma, \delta_0, q_0, F\rangle$ where Q is a finite set of states, Δ is an input-alphabet, Σ is an output-alphabet, δ_0 is a function from $Q \times (\Delta \cup \{\varepsilon\})$ to finite subsets of $Q \times \Sigma^*$, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is a set of final states.

For every $q_1, q_2, q_3 \in Q$, $a \in (\Delta \cup \{\varepsilon\})$, $w \in \Delta^*$ and $x, y \in \Sigma^*$, the extended transition function δ from $Q \times \Delta^*$ to subsets of $Q \times \Sigma^*$ is defined as the least function satisfying the following

$$(q_1, \varepsilon) \in \delta(q_1, \varepsilon) \qquad (15)$$

$$(q_2, x) \in \delta(q_1, w) \wedge (q_3, y) \in \delta_0(q_2, a) \Rightarrow (q_3, xy) \in \delta(q_1, wa) \qquad (16)$$

For any NFT $M = \langle Q, \Delta, \Sigma, \delta_0, q_0, F\rangle$, the NFT-mapping M of a string $w \in \Delta^*$ and a language $L \subseteq \Delta^*$ is

$$\begin{aligned} M(w) &= \{x \mid \exists q \in F : (q, x) \in \delta(q_0, w)\} && (17) \\ M(L) &= \bigcup_{w \in L} M(w) && (18) \end{aligned}$$

[4]See Ginsburg [7] for more about full abstract families of languages.

[5]We assume here that Γ is the set of all symbols that we use and drop Γ as subscript in $\mathcal{C}(\text{THFSG})$.

The inverse NFT-mapping M^{-1} of a string $x \in \Sigma^*$ and a language $L' \subseteq \Sigma^*$ is

$$\begin{aligned} M^{-1}(x) &= \{w \mid x \in M(w)\} && (19)\\ M^{-1}(L') &= \bigcup_{w \in L'} M^{-1}(w) && (20) \end{aligned}$$

The definition of NFT is sufficiently general that for any given NFT-mapping, the inverse NFT-mapping is also an NFT-mapping. A finite state machine is a special version of an NFT, which writes every symbol it reads, and does not change state or write anything while reading the empty string. If M is a finite state machine version of an NFT, then $M(L)$ is the intersection of L and the regular language that the finite state machine describes.

A string homomorphism $h : \Delta^* \to \Sigma^*$ can be expressed by an NFT. Let M_h be the NFT $\langle Q, \Delta, \Sigma, \delta_0, q_0, F\rangle$ such that $Q = F = \{q_0\}$ and for all $a \in \Delta$ $\delta(q_0, a) = \{(q_0, h(a)\}$. Then $h(L) = M_h(L)$ for any language $L \subseteq \Delta^*$ and the inverse string homomorphism can also be expressed with an NFT-mapping.

By showing that the class $\mathcal{C}$(THFSG) is closed under NFT-mapping it follows that it is closed under intersection with regular languages, string homomorphism and inverse string homomorphism. We do this by first defining a grammar from a THFSG-grammar on normal form and an NFT, and then show that this grammar generates the NFT-mapping of the language generated by the first grammar.

Definition 7 *Given a* THFSG*-grammar* $G = \langle \mathcal{K}, \mathcal{S}, \Delta, \mathcal{P}, \mathcal{L}\rangle$ *on normal form and a Nondeterministic Finite Transducer* $M = \langle Q, \Delta, \Sigma, \delta_0, q_0, F\rangle$*. Assume that the symbols* $\mathcal{S}_0$ *and* $\tilde{a}$ *for all* $a \in (\Sigma \cup \{\varepsilon\})$ *are not used in* G*. The NFT-mapping* $M(L(G))$*'s grammar,* $G_M = \langle \mathcal{K}', \mathcal{S}_0, \Sigma, \mathcal{P}', \mathcal{L}'\rangle$ *is defined as follows:*
Let $\mathcal{K}'$ *be the set* $(Q \times (\mathcal{K} \cup \Delta \cup \{\varepsilon\}) \times Q) \cup \{\tilde{a} \mid a \in (\Sigma \cup \{\varepsilon\})\} \cup \{\mathcal{S}_0\}$ *and let* $\mathcal{P}'$ *and* $\mathcal{L}'$ *be the least sets such that:*

a) For all $q \in F$*, the following is a rule in* $\mathcal{P}'$*:*

$$\mathcal{S}_0 \;\to\; \begin{array}{c} (q_0, \mathcal{S}, q) \\ \uparrow=\downarrow \end{array} \tag{21}$$

b) For all production rules

$$K_0 \;\to\; \begin{array}{cc} K_1 & K_2 \\ E_1 & E_2 \end{array} \tag{22}$$

in $\mathcal{P}$ *and all* $q_1, q_2, q_3 \in Q$*, the following is a rule in* $\mathcal{P}'$*:*

$$(q_1, K_0, q_3) \;\to\; \begin{array}{cc} (q_1, K_1, q_2) & (q_2, K_2, q_3) \\ E_1 & E_2 \end{array} \tag{23}$$

c) For all lexicon rules

$$K \;\to\; \begin{array}{c} b \\ E \end{array} \tag{24}$$

in $\mathcal{L}$ and all $q_1, q_2 \in Q$, the following is a rule in $\mathcal{P}'$:

$$\begin{array}{ccc} (q_1, K, q_2) & \rightarrow & (q_1, b, q_2) \\ & & E \cup \{\uparrow=\downarrow\} \end{array} \tag{25}$$

d) *For all $q_1, q_2, q_3 \in Q$ and all $b \in (\Delta \cup \{\varepsilon\})$, the following are rules in P'*

$$\begin{array}{cccc} (q_1, b, q_3) & \rightarrow & (q_1, b, q_2) & (q_2, \varepsilon, q_3) \\ & & \uparrow=\downarrow & \uparrow=\downarrow \end{array} \tag{26}$$

$$\begin{array}{cccc} (q_1, b, q_3) & \rightarrow & (q_1, \varepsilon, q_2) & (q_2, b, q_3) \\ & & \uparrow=\downarrow & \uparrow=\downarrow \end{array} \tag{27}$$

e) *For all $q_1, q_2 \in Q, b \in (\Delta \cup \{\varepsilon\})$ and $y \in \Sigma^*$, such that $(q_2, y) \in \delta_0(q_1, b)$ where $y = a_1 ... a_n$ for $|y| = n \geq 1$, or if $y = \varepsilon$ let $\tilde{a}_1 = \tilde{\varepsilon}$ and $n = 1$, the following is a production rule in $\mathcal{P}'$:*

$$\begin{array}{ccccc} (q_1, b, q_2) & \rightarrow & \tilde{a}_1 & \cdots & \tilde{a}_n \\ & & \uparrow=\downarrow & & \uparrow=\downarrow \end{array} \tag{28}$$

f) *For all $a \in (\Sigma \cup \{\varepsilon\})$, the following is a rule in $\mathcal{L}'$:*

$$\begin{array}{ccc} \tilde{a} & \rightarrow & a \\ & & \emptyset \end{array} \tag{29}$$

The main idea in this definition is that if a node in a c-structure based on G with category K is the root of a sub-c-structure with x as terminal string and the NFT accept x as input string in a state q, then there is a corresponding node in a c-structure based on G_M with category (q, K, p). This node is the root of a sub-c-structure with y as terminal string such that $(p, y) \in \delta(q, x)$, or less formally, such that the NFT may write y when reading the string x processing from state q to p (figure 1). This is done such that the new c-structure gives a specification of how the NFT processes the input string, changes states and writes symbols. Downwards in the new c-structure we get more and more details of how the string is processed. In the end the grandmothers of the terminal nodes correspond to each transition step.

In the definition, part *a*), *b*) and *c*) give us for any c-structure based on G with $w = b_1 ... b_n$ as terminal string, the upper part of a new c-structure based on G_M, where the upper part is isomorphic with the first c-structure except that it will have an additional root node on the top, *a*). The main point here is that the terminal nodes in the first c-structure will have corresponding nodes with categories $(q_0, b_1, q_1), (q_1, b_2, q_2), ...,$ (q_{n-1}, b_n, q_n) in the new one, for any possible sequence of states, $q_0, q_1, ..., q_n$ where q_0 is the initial state, and q_n is a final state. This is done such that if a node has (exactly) two daughters labeled (q, K_1, q') and (q'', K_2, q'''), q' must be equal to q'' and the mother node must be labeled (q, K_0, q''') where K_0, K_1 and K_3 are the categories

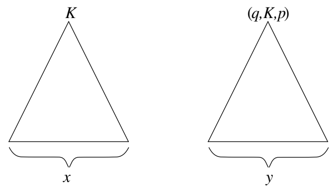

Figure 1: *Transformation to the NFT-mappings grammar for* $(p, y) \in \delta(q, x)$.

labeling the corresponding nodes in the first c-structure. Part *d*) in the definition allows the NFT to write symbols and change states while reading the empty string. In part *e*) we limit the previous parts of the definition such that all c-structures must correspond to the transition function in the NFT. This is achieved by requiring that for any symbol $b \in (\Delta \cup \{\varepsilon\})$, the triple category (q_1, b, q_2) only can label the grandmother nodes of the terminal nodes in a c-structure if in fact there exists a one step transition from state q_1 to q_2 while reading b. The daughters of this node have nonterminal categories representing the output symbols of this one step transition. The last part of the definition *f*) is just the lexical complement of part *e*).

With respect to the equation sets we take those with us to the new c-structure such that we get the same constraints on the new c-structures as on the c-structure based on the original ones.

Lemma 3 $\mathcal{C}$(THFSG) *is closed under NFT-mapping.*

Proof: Given Definition 7 we have to show that for all strings in $u \in \Sigma^*$, $u \in L(G_M)$ if and only if there exist a string w in $L(G)$ and a a final state $q \in F$ such that $(q, u) \in \delta(q_0, w)$.

($\Rightarrow$) Assume that we have a consistent c-structure based on G_M with u as terminal string such that $u \in L(G_M)$. 1) By induction on the height of the nodes we have that if a node with category (q, b, q') is a root of a sub-c-structure with y as terminal string, where $b \in (\Delta \cup \{\varepsilon\})$, then $(q', y) \in \delta(q, b)$. 2) By an induction top down in the c-structure we have that for any horizontal node-cut of nodes labeled with triple categories $(q_1, \beta_1, q_1'), ..., (q_n, \beta_n, q_n')$ where $\beta_1, ..., \beta_n \in (\mathcal{K} \cup \Delta \cup \{\varepsilon\})$, that $q_i' = q_{i+1}$ for all i, $1 \leq i \leq (i-1)$, q_1 is the initial state and q_n' is a final state. 3) There exists a sequence of the topmost nodes with triple categories where each β_i is in $(\Delta \cup \{\varepsilon\})$ and each node has a mother node with a category (q_i', K_i, q_i) for $K_i \in \mathcal{K}$. This sequence forms a node cut and if $(q_0, b_1, q_1), (q_1, b_2, q_2), ..., (q_{n-1}, b_n, q_n)$ are the categories labeling these nodes in lexicographical order, this sequence give us a string $w = b_1...b_n$ in Δ^*. The concatenation of the terminal strings $y_1, ..., y_n$ of the sub-c-structures where these nodes are the roots is u. ¿From the definition of the extended transition function and the induction in the first part we have that $(q_n, u) \in \delta(q_0, w)$. 4) By reversing Definition 7 *b*) and *c*) it is straightforward to construct a c-structure for w based on G, and if the c-structure for u generates a feature structure so must the one for w. Then $w \in L(G)$.

($\Leftarrow$) Assume that we have a $w \in L(G)$ and a final state q such that $(q, u) \in \delta(q_0, w)$ for a string $u \in \Sigma^*$. Since $(q, u) \in \delta(q_0, w)$ there must be a processing of w of the

NFT with u as output. Following the discussion of Definition 7 it is straightforward to construct a c-structure for u based on G_M. If the c-structure for w generates a feature structure so must the new one also, since we do not add any substantial new equations. Then we have that $u \in L(G_M)$. $\square$

¿From Lemma 2 and Lemma 3 we have the main result in this section.

Theorem 1 $\mathcal{C}(\text{THFSG})$ *is a full abstract family of languages.*

5 Cross-Serial Dependencies

During the last decade the discussion whether or not natural languages can be described by context-free grammars has been revived [6]. In this discussion it has been distinguished between a grammars capacity to describe a language *strongly*, i.e., to describe the language as a structured set, or *weakly*, i.e., to describe the language as a set of strings. Cross-serial dependencies are one of the main characteristics used to show that context-free grammars are not capable of describing natural language.

Cross-serial dependencies occur in languages like $\{xx \mid x \in \Sigma^*\}$[6] and $\{wa^m b^n x c^m d^n y \mid w, x, y, \in \Sigma^*, m, n \geq 1, a, b, c, d \in \Sigma\}$, but not in languages like $\{xx^R \mid x \in \Sigma^*\}$[7] where we have nested dependencies.

Shieber [17] has shown that Swiss German has cross-serial dependencies on the syntax level, and therefore in a weak description of the language. This is due to two facts about Swiss German:

> *"First, Swiss German uses case-marking (dative and accusative) on objects, just as standard German does; different verbs subcategorize for objects of different case. Second, Swiss German, like Dutch, allows cross-serial order for the structure of subordinate clauses. Of critical importance is the fact that Swiss German requires appropriate case-marking to hold even within the cross-serial construction."* Shieber [17] (page 334).

This occurs e.g. in the following subordinate clauses preceded by *"Jan säit das"* ("Jan says that"):

...mer	em Hans	es huus	hälfed	aastriiche	
...we	Hans(DAT)	the house(ACC)	helped	paint	(30)
...we helped Hans paint the house.					

Here the verb *hälfed* subcategorizes for an object in dative; *em Hans*, and the verb *aastriiche* subcategorizes for an object in accusative; *es huus*. Shieber shows that this dependency is robust and that it holds in quite complex clauses, as seen in this example:

...mer	d'chind	em Hans	es huus
...we	the children(ACC)	Hans(DAT)	the house(ACC)

[6]We assume that Σ has more than one symbol

[7]If $x = a_1 \ldots a_n$ then $x^R = a_n \ldots a_1$.

haend	wele	laa	hälfe	aastriiche	
have	wanted	let	help	paint	(31)

...we have wanted to let the children help Hans paint the house.

If we change the cases of the objects then the strings become ungrammatical. Shieber (p. 336) specifies 4 claims that this construction in Swiss German satisfies:

1. *"Swiss-German subordinate clauses can have a structure in which all the Vs follow all the NPs."*
2. *"Among such sentences, those with all dative NP's preceding all accusative NPs, and all dative-subcategorizing Vs preceding all accusative-subcategorizing Vs are acceptable."*
3. *"The number of Vs requiring dative objects (e.g.,* hälfe*) must equal the number of dative NPs (e.g.,* em Hans*) and similarly for accusatives (*laa *and* chind*)."*
4. *"An arbitrary number of Vs can occur in a subordinate clause of this type (subject, of course, to performance constraints)."*

Shieber then shows that any language that satisfies these claims cannot be context-free, since such languages allow constructions on the form $wa^m b^n x c^m d^n y$. Here we study the language L which contains strings on the form

$$\textit{Jan säit das mer } N_1 \ldots N_n \textit{ es huus haend wele } V_1 \ldots V_n \textit{ aastriiche} \tag{32}$$

where $n \geq 1$ and $N_i \in \{\textit{em Hans, es Hans, d'chind}\}$[8] and $V_i \in \{\textit{hälfe, laa}\}$ for all i, $1 \geq i \geq n$, and such that $V_i = \textit{hälfe}$ if and only if $N_i = \textit{em Hans}$.

We see that this is a subset of Swiss German with the right case marking and subcategorizing and that it satisfies Shiebers claims. Hence it cannot be context-free. To make it easier to study we use the following homomorphism[9]:

$$\begin{array}{lcll} h(\textit{Jan säit das mer}) & = & x & \\ h(\textit{es huus haend wele}) & = & y & \\ h(\textit{aastriiche}) & = & z & \\ h(s) & = & \mathrm{s} & \text{for all } s \in (N_{all} \cup V_{all}) \end{array} \tag{33}$$

where N_{all} is the set $\{\textit{em Hans, es Hans, d'chind}\}$ and V_{all} is the set $\{\textit{hälfe, laa}\}$. We then have that $h(L)$ is the following language:

$$\begin{aligned} h(L) \;=\; & \{xN_1 \ldots N_n y V_1 \ldots V_n z \mid \\ & n \geq 1 \wedge \\ & \forall i\, 1 \leq i \leq n\, [N_i \in N_{all} \wedge V_i \in V_{all} \wedge (V_i = \textit{hälfe} \iff N_i = \textit{em Hans})]\} \end{aligned} \tag{34}$$

We construct the following THFSG-grammar $G = \langle \mathcal{K}, \mathcal{S}, \Sigma, \mathcal{P}, \mathcal{L} \rangle$ for the language $h(L)$: Let

[8] For simplicity we define the constructions *em Hans, es Hans* and *d'chind* as atomic symbols.

[9] We can do this since our grammar formalism is closed under string homomorphism and inverse string homomorphism

$$\Sigma = \{\textit{em Hans, es Hans, d`chind, hälfe, laa}, x, y, z\}, \text{ and}$$
$$\mathcal{K} = \{S, VP, V, NP, N, X, Y, Z\}$$

We have the following production rules in $\mathcal{P}$:

$$S \rightarrow \begin{array}{ccccc} X & NP & Y & VP & Z \\ \uparrow=\downarrow & \uparrow=\downarrow & \uparrow=\downarrow & \uparrow=\downarrow & \uparrow=\downarrow \end{array}$$

$$NP \rightarrow \begin{array}{cc} N & NP \\ \uparrow obj =\downarrow & \uparrow vcomp =\downarrow \end{array} \qquad NP \rightarrow \begin{array}{c} N \\ \uparrow obj =\downarrow \\ \uparrow vcomp = null \end{array}$$

$$VP \rightarrow \begin{array}{cc} V & VP \\ \uparrow obj =\downarrow & \uparrow vcomp =\downarrow \end{array} \qquad VP \rightarrow \begin{array}{c} V \\ \uparrow obj =\downarrow \\ \uparrow vcomp = null \end{array}$$

We have the following lexicon rules in $\mathcal{P}$:

$$N \rightarrow \begin{array}{c} \textit{em Hans} \\ \uparrow case = DAT \end{array} \qquad N \rightarrow \begin{array}{c} \textit{es Hans} \\ \uparrow case = ACC \end{array} \qquad N \rightarrow \begin{array}{c} \textit{d'chind} \\ \uparrow case = ACC \end{array}$$

$$V \rightarrow \begin{array}{c} \textit{la} \\ \uparrow obj\ case = ACC \end{array} \qquad V \rightarrow \begin{array}{c} \textit{hälfe} \\ \uparrow obj\ case = DAT \end{array}$$

$$X \rightarrow \begin{array}{c} x \\ \emptyset \end{array} \qquad Y \rightarrow \begin{array}{c} y \\ \emptyset \end{array} \qquad Z \rightarrow \begin{array}{c} z \\ \emptyset \end{array}$$

¿From this grammar we get that strings like *"x em Hans d'chind y hälfe laa z"* are grammatical, while a string like *"x es Hans d'chind y hälfe laa z"* is ungrammatical, because of an inconsistency in the equation set. In figure 2 we show the c-structure and feature structure for the string

$$\textit{“x d'chind em Hans y laa hälfe z”} \tag{35}$$

This is not an adequate linguistic analysis, but an example of how we may collect cross-serial information with a THFSG-grammar.

6 Summary and remarks

We have defined a grammar formalism that describes a full abstract family of languages and showed that it can describe weakly a small subset of Swiss German with cross-serial dependencies. It is not surprising that the membership problem for this grammar formalism is also NP-hard [4]. This due to the feature structures capacity to collect and distribute information across the sentence. This gives us the possibility to distribute truth-assignments uniformly for boolean expressions.

The main conflicting interest is concerned with *linguistic flexibility*. This implies the search for a formalism that is so flexible that we can express linguistic phenomena, as precisely as possible, in the way linguists would wish to state them. In the grammar formalism presented here we have a simple way of describing feature structures. This results that the feature structures will be trees. It may be argued that this is too limited compared to the much richer formalisms used in HPSG and LFG. On the other hand,

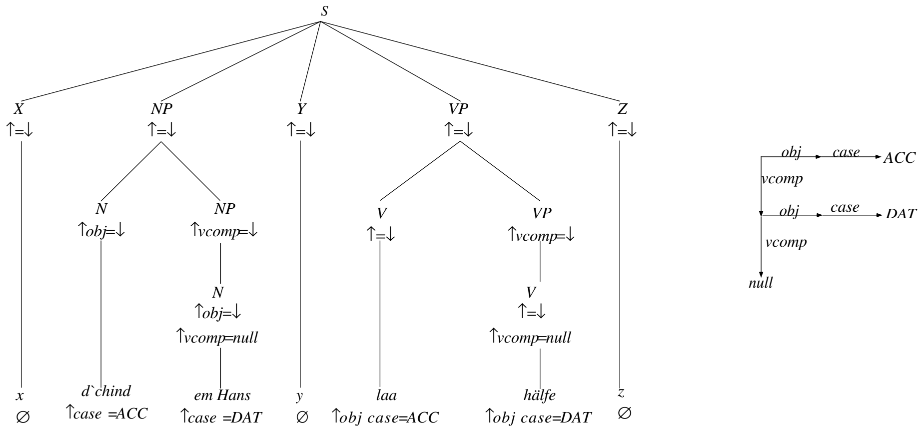

Figure 2: *c-structure and feature structure for cross-serial dependencies*

on the string level of natural languages, cross-serial dependencies are to my knowledge, the only constructions that are outside the context-free domain. We will not draw any strong conclusions regarding the linguistic adequacy of this grammar formalism, but leave it as an open question.

References

[1] Edward G. Barton, Robert C. Berwick, and Eric S. Ristad. *Computational Complexity and Natural Language*. MIT-Press, 1987.

[2] Patrick Blackburn. Structures, languages and translations: the structural approach to feature logic. In C. J. Rupp, M. A. Rosner, and R. L. Johnson, editors, *Constraints, Language and Computation*, pages 1–28. Academic Press, London., 1994.

[3] Tore Burheim. Regelstrukturer og tolkningsdefinisjoner. Master's thesis, Department of Informatics, University of Bergen, 1992. In Norwegian.

[4] Erik A. Colban. *Three Studies in Computational Semantics*. PhD thesis, University of Oslo, 1991.

[5] Jean H. Gallier. *Logic for Computer Science*. Harper & Row, Publishers, New York, 1986.

[6] Gerald Gazdar and Geoffrey Pullum. Natural languages and context-free languages. *Linguistics and Philosophy*, 4:471–504, 1982.

[7] Seymour Ginsburg. *Algebraic and Automata-Theoretic Properties of Formal Languages*. North-Holland Publishing Company, 1975.

[8] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[9] Mark Johnson. *Attribute-Value Logic and the Theory of Grammar*. Number 14 in CSLI Lecture Notes. Center of the Study of Language and Information., Stanford University, California, USA, 1988.

[10] Mark Johnson. Features and formulae. *Computational Linguistics*, 17(2):131–151, 1991.

[11] Mark Johnson. Computing with features as formulae. *Computational Linguistics*, 20(1):1–25, 1994.

[12] Ronald M. Kaplan and Joan Bresnan. Lexical functional grammar: A formal system of grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT-Press, 1982.

[13] Robert T Kasper and William T. Rounds. The logic of unification in grammar. *Linguistics and Philosophy*, 13(1):35–58, 1990.

[14] Bill Keller and David Weir. A tractable extension of linear indexed grammars. In *Proceedings of the 7th EACL*. Association for Computational Linguistic, 1995.

[15] Harry R. Lewis and Christos H. Papadimitriou. *Elements of The Theory of Computation*. Prentice-Hall, Inc, 1981.

[16] Carl Pollard and Ivan Sag. *Head-Driven Phrase Structure Grammar*. CSLI, Stanford University and University of Chicago Press, Chicago, 1994.

[17] Stuart M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343, 1985.

[18] Stuart M. Shieber. *An Introduction to Unification-based Approaches to Grammar*. Number 4 in CSLI Lecture Note Series. Chicago University Press, 1986.

[19] Gert Smolka. A feature logic with subsorts. LILOG-report 33, IBM-Deutschland, Stuttgart, May 1988.

[20] Gert Smolka. Feature constraint logics for unification grammars. *Journal of Logic Programming*, 12:51–87, 1992.