Algebraic Methods in Language Processing AMiLP-3,
Anton Nijholt and Giuseppe Scollo (eds.), 2003.

A Note on the Complexity of Optimality Theory

Stephan Kepser and Uwe Mönnich[*]
Dept. of Linguistics, University of Tübingen, Germany
{kepser,um}@sfs.uni-tuebingen.de

Abstract

Optimality theory (OT), introduced by Prince and Smolensky (1993), is a linguistic framework in which the mapping of one level of linguistic representation to another is based on rules and filters. The rules generate candidate expressions in the target representation, which are subsequently checked against the filters, so that only those candidates remain that survive this filtering process. Germain to OT is the fact that filters or constraints are violable and ranked. Thus a candidate expression may violate a constraint, as long as alternative candidates violate more constraints or higher ranked constraints. An expression is optimal, if it violates the least number of lowest ranked constraints. Frank and Satta (1998) and Wartena (2000) have shown that these principles of OT can be fruitfully modeled using techniques from formal language theory. Both model the generator relation as a rational relation over regular languages and the constraints as regular languages; in (Frank and Satta, 1998) these are string languages, and in (Wartena, 2000) they are tree languages. We show here that generators can be extended to linear frontier-to-root tree transducers on linear context-free tree languages – with constraints being regular tree languages – while the computation of optimal candidates can still be performed using finite state techniques (over trees).

Keywords: Optimality theory, context-free tree grammar, finite state techniques

1 Introduction

Optimality theory (OT henceforth) has been introduced by Prince and Smolensky (1993) originally as a model for generative phonology. In recent years, this approach has been applied successfully to a range of syntactic phenomena, and it is currently gaining popularity in semantics and pragmatics as well. It is based on the idea that a mapping from one level of linguistic representation to another should be described in terms of rules and filters. The novel contribution of OT is that filters – or, synonymously, constraints – are ranked and violable. Thus the result of a rule-based generation process may still be acceptable although it violates certain constraints as long as other results violate more constraints or constraints that are higher ranked.

In other words, the rules generate a set of candidates that are competitors. On this set, the constraints are applied in the order of their ranking starting with the highest ranked constraint. A candidate may violate a constraint more than once. The application of the highest ranked constraint assigns each candidate the number of violations of that constraint. Some of the candidates are now optimal with respect to this constraint in the sense that they violate the constraint the fewest times. These, and only these, are retained for the next round of constraint application. In each round, the current constraint is applied to the set of candidates remaining from the previous rounds. And only those candidates that are optimal with respect to the current constraint make it into the next round. In the end, after applying all constraints, a set of candidates is reached which is optimal with respect to the given ranking of the constraints. The method is therefore comparable to a high jump competition in athletics.

[*]This research was partly funded by a grant of the German Science Foundation (DFG SFB 441-2).

Frank and Satta (1998) show that certain classes of OT-systems can be handled by finite state techniques. Their approach is influenced by ideas from computational phonology, the original field of application for OT. In this view, the generation of candidates is a relation on strings, and this relation is defined by a finite state transducer. In order to also render constraints by finite state automata, two restrictions have to be made. The first one is that constraints have to be binary, that is to say, each constraint assigns each candidate either 0 or 1. The second restriction demands constraints to be *output* constraints. An output constraint is a constraint that assigns a number to a candidate pair purely on the base of its output. Under these restrictions, constraints can be rendered as regular string languages over the output. The aim of the paper by Frank and Satta (1998) is to provide a modularity result for the complexity of an OT-system in the following sense. Suppose that the set of candidates is given by a finite state transducer and all constraints are expressable by regular languages. Then the whole OT-system can be rendered by finite state techniques and is no more complex than its components. The success of the approach by Frank and Satta is based on well-known closure properties of regular string languages.

Wartena (2000), noting that these closure properties extend to regular *tree* languages, demonstrates how the approach of Frank and Satta can be extended from strings to trees. The set of candidates, now, is a binary relation on trees that is defined by means of a linear tree transducer. And binary output constraints are defined by means of tree automata. The use of tree automata as a way to express constraints in syntax was proposed previously by Morawietz and Cornell (1997). Based on these assumptions Wartena achieves the corresponding modularity result that if the components of OT-system are defined by automata on trees then the whole OT-system can be defined by automata on trees and hence shares the complexity of its components.

For the description of natural language syntax, it is trees that are regarded as the underlying data structures by most linguists. Therefore the step from string automata to tree automata is certainly a necessary one. But there are well-known arguments by Shieber (1985) and others that natural languages are not context-free. In particular, certain aspects of the morphology of Bambara and the case agreement in Swiss German are mildly context-sensitive. It is therefore arguable that a simple use of tree automata may not suffice. In a series of papers, Kolb et al. (2003, 2000); Michaelis et al. (2001); Morawietz and Mönnich (2001) provide a systematic way to render mildly context-sensitive phenomena in natural language using purely (tree-) regular means. Based on this approach we show here how to integrate a moderate level of context-sensitivity into an OT-system while still using extended tree automata techniques.

We propose to define the generator of an OT-system to be a relation on linear context-free tree languages. This relation is given as a so-called linear frontier-to-root tree transducer (LF-transducer). Linear context-free tree languages form a proper super class of the regular tree languages, hence standard automata techniques cannot be applied. We therefore employ a two-level mechanism, i.e, we apply a technical process called lifting to the linear context-free tree grammar defining the domain of the generator. And we also lift the LF-transducer. Since the result of lifting the grammar is a regular tree grammar, we can now use automata techniques. Constraints are expressed as monadic second-order (MSO) formulae over candidate output trees. We think this approach is more appealing to linguists than using regular tree grammars because logic allows to specify the constraints in an abstract, grammar-free fashion. If desired, the formulae can be translated into regular tree languages and vice versa using well known formal languages techniques (Gécseg and Steinby, 1997). Constraints will also be lifted to match the level of the lifted generator, and then transformed into tree automata. This allows the computation of optimal candidates on the lifted level using automata techniques. In order to obtain the optimal candidates on the intended, original level, we apply another automaton, a macro tree transducer equivalent to an MSO transduction to be defined below, to undo the translation step introduced in the lifting. We have thus shown that optimal outputs of an OT-system can be computed using finite state techniques over trees even in the case where the generator comprises mildly context-sensitive tree languages.

After reviewing of basic concepts from algebra, logic, grammar and automata theory in the next section we introduce and formalise the concepts of an OT-system in Section 3. Section 4 reports the constructions and results by Frank and Satta (1998) and Wartena (2000) that we build upon. In Section 5 we present our own two-level approach to the formalisation of OT.

2 Preliminaries

2.1 Basic Algebraic Definitions

Let $\mathcal{S}$ be a finite set of *sorts*. An $\mathcal{S}$-signature is a set Σ given with two mappings $\alpha : \Sigma \rightarrow \mathcal{S}^*$ (the *arity* mapping) and $\sigma : \Sigma \rightarrow \mathcal{S}$ (the *sort* mapping). The length of $\alpha(f)$ is called the *rank* of f, and is denoted by $\rho(f)$. The profile of f in Σ is the pair $(\alpha(f), \sigma(f))$. The elements of $\Sigma_{\varepsilon,s}$ are also called constants (of sort s).

A Σ-*algebra* is an pair $\mathcal{A} = \langle (A_s)_{s \in \mathcal{S}}, (f)_{f \in \Sigma} \rangle$ where A_s is a nonempty set for each $s \in \mathcal{S}$, called the domain or universe of sort s of $\mathcal{A}$, and $f : A_{\alpha(f)} \rightarrow A_{\sigma(f)}$ is a total function for each $f \in \Sigma$. (For a sequence $\mu = (s_1, \ldots, s_n)$ in $\mathcal{S}^+$, we let $A_\mu := A_{s_1} \times A_{s_2} \times \cdots \times A_{s_n}$.) A Σ-algebra is *finite* iff the carriers A_s for each $s \in \mathcal{S}$ are finite.

In case $\mathcal{S}$ is a singleton set $\{s\}$, i.e., in case Σ is a *single-sorted or ranked alphabet (over sort s)*, we usually write Σ_n to denote the (unique) set of operators of rank $n \in \mathbb{N}$. In later sections of the paper we will mainly use the single-sorted case of alphabets. We will indicate the need for many-sorted alphabets where necessary.

Let $\mathcal{S}$ be a set of sorts, Σ a signature, and $\mathcal{A}$ and $\mathcal{B}$ two Σ-algebras. A family of functions $h_s : A_s \rightarrow B_s$ (for each $s \in \mathcal{S}$) is called a Σ-*homomorphism* iff for all $f \in \Sigma$ of rank k and all $(a_1, \ldots a_k) \in A_{\alpha(f)}$: $h_{\sigma(f)}(f_{\mathcal{A}}(a_1, \ldots, a_k)) = f_{\mathcal{B}}(h_{s_1}(a_1), \ldots, h_{s_k}(a_k))$.

Of particular interest to us is the algebra $\mathcal{T}_\Sigma$ of *trees* over a single-sorted signature Σ. It is the free algebra of Σ. The carrier T_Σ is defined recursively as follows. Each constant of Σ, i.e., each symbol of rank 0, is a tree. If f is of rank k and $t_1, \ldots, t_k$ are trees, then $f(t_1, \ldots, t_k)$ is a tree. Operations in $\mathcal{T}_\Sigma$ are syntactic, i.e., if $f \in \Sigma_k$ and $t_1, \ldots, t_k \in T_\Sigma$ then $f_{\mathcal{T}_\Sigma}(t_1, \ldots, t_k) := f(t_1, \ldots, t_k)$. $\mathcal{T}_\Sigma$ is the *free* or *initial* algebra in the class of all Σ algebra, i.e., for each algebra $\mathcal{A}$ there exists a unique homomorphism $h_{\mathcal{A}} : \mathcal{T}_\Sigma \rightarrow \mathcal{A}$. This homomorphism is the *evaluation* of a term in $\mathcal{A}$.

A *tree language* $L \subseteq T_\Sigma$ is a subset of T_Σ. With each tree $t \in T_\Sigma$ we can associate a string $s \in \Sigma_0^*$ by reading the leaves of t from left to right. This string is called the *yield* of t, denoted $yd(t)$. The yield of a tree language L is defined straightforwardly as $yd(L) = \{yd(t) \mid t \in L\}$.

2.2 Basic Tree Grammar Definitions

A *context-free tree grammar* (CFTG) is a quintuple $\Gamma = \langle \Sigma, \mathcal{F}, S, X, P \rangle$ where Σ and F are ranked alphabets of *inoperatives* and *operatives*, $S \in \mathcal{F}$ is the start symbol, X is a countable set of variables, and P is a finite set of productions. Each production $p \in P$ is of the form $F(x_1, \ldots, x_n) \rightarrow t$ for some $n \in \mathbb{N}$, where $F \in \mathcal{F}_n, x_1, \ldots, x_n \in X$ and t is a tree over $\Sigma \cup \mathcal{F}, \{x_1, \ldots, x_n\}$. The grammar is *linear* iff each variable occurs at most once in the right hand side of each rule. Intuitively, an application of a rule of the form $F(x_1, \ldots, x_n) \rightarrow t$ "rewrites" a tree rooted in F as the tree t with its respective variables substituted by F's daughters.

A CFTG $\Gamma = \langle \Sigma, \mathsf{F}, S, \mathsf{X}, \mathsf{P} \rangle$ with $\mathsf{F}_n = \emptyset$ for $n > 0$ is called a *regular tree grammar (RTG).* Since RTGs always just substitute some tree for a leaf-node, it is easy to see that they can only generate recognisable sets of trees, *a forteriori* context-free string languages (Mezei and Wright 1967). If F_n is non-empty for some $n \neq 0$, that is, if we allow the *operatives* to be parameterised by variables, however, the situation changes. CFTGs in general are capable of generating sets of structures, the *yields* of which belong to the subclass of context-sensitive languages known as the *indexed* languages.

Let us illustrate the above by means of an example. The following CFTG generates the mildly context-sensitive language $a^n b^n c^n d^n$.

Example 1 Consider the CFTG $\Gamma = \langle\{a, b, c, d, \varepsilon, S_t, S_t^0\}, \{S, S', \overline{S}_1, \overline{S}_2, \overline{a}, \overline{b}, \overline{c}, \overline{d}\}, S', \{x\}, \mathsf{P}\rangle$ with P given as follows

$$\begin{aligned}
S' &\longrightarrow S(\varepsilon) & \overline{a} &\longrightarrow a \\
S(x) &\longrightarrow \overline{S}_1(S(\overline{S}_2(x))) & \overline{b} &\longrightarrow b \\
S(x) &\longrightarrow S_t^0(x) & \overline{c} &\longrightarrow c \\
\overline{S}_1(x) &\longrightarrow S_t(\overline{a}, x, \overline{d}) & \overline{d} &\longrightarrow d \\
\overline{S}_2(x) &\longrightarrow S_t(\overline{b}, x, \overline{c}) & &
\end{aligned}$$

An example of a tree generated by this grammar is shown in Figure 1.

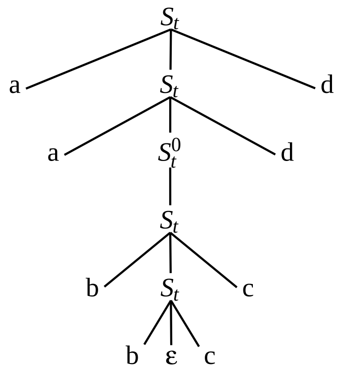

Figure 1: Sample tree

Let Γ be a CFTG. For every tree $t \in L(\Gamma)$, there is a set of sequences of applications of grammar rules that generate t. These sequences may differ in more than just the order of rule application, a tree can have several different derivations.

If Γ is a regular tree grammar, then the language generated by Γ is called a *regular* tree language.

2.3 Basic Automata-Theoretic Definitions

For regular tree languages there exists an automaton model that corresponds to finite state automata for regular string languages. Let Σ be a signature. A *deterministic frontier-to-root tree automaton* is a pair $(\mathcal{A}_\Sigma, F)$ where $\mathcal{A}_\Sigma$ is a *finite* Σ-algebra and $F \subseteq \bigcup_{s \in \mathcal{S}} A_s$ is the set of *final states*. A tree $t \in T_\Sigma$ is recognised by $(\mathcal{A}_\Sigma, F)$ iff $h_\mathcal{A}(t) \in F$, i.e., the evaluation of the term t in the automaton ends in a final state. On an intuitive level, a bottom-up tree automaton labels the nodes in a tree with states starting from the leaves and going to the root. Suppose n is a node in the tree and f is the k-ary function symbol at node n and the k daughters of n are already labelled with states $q_1, \ldots, q_k$, and furthermore $f_\mathcal{A}(q_1, \ldots, q_k) = q$ is true in $\mathcal{A}$ then node n can be labelled with state q. A tree is accepted if the root can be labelled with a final state. Since we will only consider deterministic bottom-up tree automata in this paper, we will henceforce just call them tree automata for brevity.

The language accepted by a tree automaton $(\mathcal{A}, F)$ is the set $\{t \in T_\Sigma \mid h_\mathcal{A}(t) \in F\}$. We will now report some results about the theory of regular tree languages. For more information, consult the work by Gécseg and Steinby (1984, 1997). A tree language L is regular if and only if there is a tree automaton that accepts L. Regular tree languages are closed under union, intersection, and complement. There are corresponding constructions for tree automata.

Tree automata can be generalised to automata that transform one tree into another, so-called tree transducers. The following exposition on tree transduction is taken from (Gécseg and Steinby, 1997). Let Σ and Ω be two (single-sorted) signatures. A binary relation $\tau \subseteq T_\Sigma \times T_\Omega$ is called a *tree transformation.* A pair $(s,t) \in \tau$ is interpreted to mean that τ may transform s into t. We can speak of compositions, inverses, domains, and ranges of tree transformations as those of binary relations. With each tree transformation $\tau \subseteq T_\Sigma \times T_\Omega$ one can associate a translation of the string languages $\{(yd(s), yd(t) \mid (s,t) \in \tau\}$. We will now define frontier-to-root tree transducers.

Definition 2 [F-Transducer] A *frontier-to-root tree transducer* (or F-transducer) consists of a quintuple $\mathcal{A} = (\Sigma, \Omega, Q, P, F)$ where Σ and Ω are signatures; Q is a finite set of *states*, each element of Q is a unary function; $F \subseteq Q$ is the set of *final states*; and P is a finite set of *productions* of the following type:

$$f(q_1(x_1), \ldots, q_m(x_m)) \rightarrow q(t(x_1, \ldots, x_m))$$

where $f \in \Sigma_m$, $q_1, \ldots, q_m, q \in Q$, $t(x_1, \ldots, x_m) \in T_\Omega(x_1, \ldots, x_m)$.

The transformation induced by an F-transducer is defined as follow. We write QT_Ω for the set $\{q(t) \mid q \in Q, t \in T_\Omega\}$ and regard QT_Ω as a signature in which each element $q(t) \in QT_\Omega$ is seen as a constant. Let $s, t \in T_{\Sigma \cup QT_\Omega}$ be two trees. It is said that t can be obtained by a direct derivation from s in $\mathcal{A}$ iff t can be obtained from s by replacing an occurrence of a subtree $f(q_1(t_1), \ldots, q_m(t_m))$ (with $f \in \Sigma_m, q_1, \ldots, q_m \in Q, t_1, \ldots, t_m \in T_\Omega$) in s by $q(t(t_1, \ldots, t_m))$, where $f(q_1(x_1), \ldots, q_m(x_m)) \rightarrow q(t(x_1, \ldots, x_m))$ is a production from P. If s directly derives t in $\mathcal{A}$ then we write $s \Rightarrow_\mathcal{A} t$. The reflexive transitive closure $s \Rightarrow^*_\mathcal{A} t$ is the derivation relation.

Intuitively, an F-transducer traverses a tree s from the leaves to the root rewriting it at the same time. In a single derivation step we consider a node n in s with label f where all the daughter nodes are already transformed into trees of T_Ω and each daughter node is in some state q_i. Then we replace the subtree of node n with the tree t from the production where the place holder variables of t are replaced by the trees of the daughter nodes of n. The root of this subtree is put into state q.

The relation

$$\tau_\mathcal{A} = \{(s,t) \mid s \in T_\Sigma, t \in T_\Omega, s \Rightarrow^*_\mathcal{A} q(t) \text{ for some } q \in F\}$$

is the transformation relation induced by $\mathcal{A}$. A relation $\tau \subseteq T_\Sigma \times T_\Omega$ is an F-*transformation* if there exists an F-transducer $\mathcal{A}$ such that $\tau = \tau_\mathcal{A}$. For a tree language $L \subseteq T_\Sigma$ we define $\mathcal{A}(L) = \{t \in T_\Omega \mid \exists s \in T_\Sigma \text{ with } (s,t) \in \tau_\mathcal{A}\}$.

A production $f(q_1(x_1), \ldots, q_m(x_m)) \rightarrow q(t(x_1, \ldots, x_m))$ is called *linear* if each variable $x_1, \ldots, x_m$ occurs at most once in t. An F-transducer is linear if each production is linear. We denote a linear F-transducer by LF-transducer. There are two important results about LF-transducers (see (Gécseg and Steinby, 1997)): LF-transducers are closed under composition. And if L is a *regular* tree language and $\mathcal{A}$ is an LF-transducer, then $\mathcal{A}(L)$ is again regular. Hence LF-transducers are the counterpart on trees of finite state transducers on strings.

We will now introduce *macro tree transducers.* Their states are complex objects, and they allow to pass parameters – which contain a limited amount of context information from the part of the input tree we have already seen – into the RHSs. We formalise these new RHSs as follows. Let Σ, Ω, and Q be ranked alphabets and $n, m \geq 0$. The set of right hand sides $RHS(\Sigma, \Omega, n, m)$ over Σ and Ω with n variables and m parameters is the smallest set $rhs \subseteq T_{\Sigma \cup \Omega}(X_n \cup Y_m)$ such that

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

The productions of MTTs contain one piece of "old" information (a symbol from the input alphabet with the appropriate number of variables) and a number of context parameters.

Definition 3 [Macro Tree Transducer] A macro tree transducer (MTT) is a quintuple $M = \langle Q, \Sigma, \Omega, q_0, P\rangle$ with Q a ranked alphabet of states, ranked alphabets Σ and Ω (input and output), initial state q_0 of rank 1, and a finite set of productions P of the form

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

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

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

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

Generally, a little care has to be taken in the definition of the transition relation with respect to the occurring parameters y_i. Derivations are dependent on the order of tree substitutions. Inside-out means that trees from T_Ω have to be substituted for the parameters whereas in Outside-in derivations a subtree must not be rewritten if it is in some context parameter. Again, as for context-free tree grammars, neither class of derivations contains the other. Since we are only dealing with *simple* MTTs in our approach, all modes are equivalent and can safely be ignored.

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

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

Note that if we disregard the input, MTTs turn into context-free tree grammars.

2.4 Basic Logical Definitions

After these automata-theoretic notions, we briefly present those related to monadic second-order (MSO) logic. MSO logic is the extension of first-order predicate logic with monadic second-order variables and quantification over them. In particular, we are using MSO logic on trees such that individual variables $x, y, \dots$ stand for nodes in trees and monadic second-order ones $X, Y, \dots$ for sets of nodes (for more details see, e.g., Rogers (1998)). It is well-known that MSO logic interpreted on trees is decidable via a translation to finite-state (tree) automata (Doner, 1970; Thatcher and Wright, 1968). The decidability proof for MSO on finite trees gives us also a descriptive complexity result: MSO on finite trees yields only recognisable trees which in turn yield context-free string languages. These results are of particular interest, since finite trees are clearly relevant for linguistic purposes, and therefore form the basis for our work.

The following paragraphs go directly back to Courcelle (1997). Recall that the representation of objects within relational structures makes them available for the use of logical description languages. Let R be a finite set of relation symbols with the corresponding arity for each $r \in R$ given by $\rho(r)$. A relational structure $\mathcal{R} = \langle D_\mathcal{R}, (r_\mathcal{R})_{r \in \mathcal{R}}\rangle$ consists of the domain $D_\mathcal{R}$ and the $\rho(r)$-ary relations $r_\mathcal{R} \subseteq D_\mathcal{R}^{\rho(r)}$. In our case we choose a finite tree as our domain and the relations of immediate, proper and reflexive dominance and precedence.

The classical technique of interpreting a relational structure within another one forms the basis for MSO transductions. Intuitively, the output tree is interpreted on the input tree. E.g., suppose that we want to transduce the input tree t_1 into the output tree t_2. The nodes of the output tree t_2 will be a subset of the nodes from t_1 specified with a unary MSO relation ranging over the nodes of t_1. The daughter relation will be specified with a binary MSO relation with free variables x and y ranging over the nodes from t_1. We will use this concept to transform the lifted trees into the intended ones.

Definition 4 [MSO transduction] A *(non-copying) MSO transduction* of a relational structure $\mathcal{R}$ (with set of relation symbols R) into another one $\mathcal{Q}$ (with set of relation symbols Q) is defined to be a tuple $(\varphi, \psi, (\theta_q)_{q \in Q})$. It consists of the formulae φ defining the domain of the transduction in $\mathcal{R}$ and ψ defining the resulting domain of $\mathcal{Q}$ and a family of formulae θ_q defining the new relations $q \in Q$ (using only definable formulae from the "old" structure $\mathcal{R}$).

The result which gives rise to the fact that we can characterise a non-context-free tree set with two devices which have only regular power is stated in Courcelle (1997). Viewing the relation of intended dominance defined later by a tree-walking automaton as the cornerstone of an MSO definable transduction, our description of non-context-free phenomena with two devices with only regular power is an instance of the theorem that the image of an MSO-definable class of structures under a definable transduction is not MSO definable in general (Courcelle, 1997).

3 Basic Notions of Optimality Theory

Let us make the notions of optimality theory more precise. In the general case, an OT-system consists of a relation **GEN** and a finite set of constraints that are linearly ordered. Constraints may be violated several times. So a constraint should be construed as a function from **GEN** into the natural numbers. Thus an OT-system assigns each candidate pair from **GEN** a sequence of natural numbers. The ordering of the elements of **GEN** that is induced by the OT-system is the lexicographic ordering of these sequences.

Definition 5 An *OT-system* is a pair $(\mathbf{GEN}, C)$ where **GEN** is a binary relation and $C = \langle c_1, \ldots, c_p \rangle, p \in \mathbb{N}$ is a linearly ordered sequence of functions from **GEN** to $\mathbb{N}$. Let $a, b \in \mathbf{GEN}$. We say a is more economical than b ($a < b$), if there is a $k \leq p$ such that $c_k(a) < c_k(b)$ and for all $j < k : c_j(a) = c_j(b)$.

Intuitively, an output o is optimal for some input i iff **GEN** relates o to i and o is optimal amongst the possible outputs for i. This is expressed by the following definition:

Definition 6 Let $\mathcal{O} = \langle \mathbf{GEN}, C \rangle$ be an OT-system. Then (i, o) is optimal with respect to $\mathcal{O}$ iff

1. $(i, o) \in \mathbf{GEN}$, and
2. there is no o' such that $(i, o') \in \mathbf{GEN}$ and $(i, o') < (i, o)$.

It has frequently been observed that in realistic applications, candidate sets may be infinite. Hence, a brute force complete search algorithm for an optimal output may in general not terminate. Thus the success of the OT research program crucially hinges on the issue whether there are tractable evaluation algorithms.

It is obvious that the complexity of the task of finding the optimal candidates for a given OT-system depends on the complexity of the generator and of the constraints. In the general case, these will provide a lower bound for the complexity of the OT-system as a whole. The crucial question is whether an OT-system as a whole may have a higher complexity than the most complex of its components.

4 Results by Frank and Satta and by Wartena

The first important complexity result in this spirit was proven by Frank and Satta (1998). They show that certain classes of OT-systems can be handled by finite state techniques. Their approach is influenced by ideas from computational phonology, the original field of application for OT. On this view, **GEN** is a relation on strings, and this relation is defined by a finite state transducer.

In order to also render constraints by finite state automata, certain restrictions have to be made. The first one is that constraints have to be binary, that is to say, each constraint assigns each candidate from **GEN** either 0 or 1. If there exists an upper bound on the number of potential constraint violations for a non-binary constraint, then this non-binary constraint can be translated into a sequence of binary constraints that has the same filtering effect. Therefore this restriction is moderate. The second restriction demands constraints to be *output* constraints. An output constraint is a constraint that assigns a number to a candidate from **GEN** purely on the base of its output, its right hand side element, i.e., if (i, o) and $(i', o) \in \mathbf{GEN}$ then $c((i, o)) = c((i', o))$. Under these restrictions, constraints can be rendered as regular string languages over the output of **GEN**.

The main theorem of the paper by Frank and Satta (1998) provides a modularity result for the complexity of an OT-system in the following sense. Suppose that **GEN** is given by a finite state transducer and all constraints are expressible by regular languages over the output of **GEN**. Then the whole OT-system can be rendered by finite state techniques and is no more complex than its components. The success of the approach by Frank and Satta is based on well-known closure properties of regular string languages.

Wartena (2000), noting that these closure properties extend to regular *tree* languages, demonstrates how the approach of Frank and Satta can be extended from strings to trees. **GEN**, now, is a binary relation on trees that is defined by means of a linear tree transducer. And binary output constraints are defined by means of tree automata. The use of tree automata as a way to express constraints in syntax was proposed previously by Morawietz and Cornell (1997). Based on these assumptions Wartena achieves the corresponding modularity result that if the components of OT-system are defined by automata on trees then the whole OT-system can be defined by automata on trees and hence shares the complexity of its components.

5 A Two-Step Approach to Optimality Theory

The idea of the present approach is twofold. One the one hand, the OT-system shall in parts be represented by languages over linear context-free tree grammars. The generating system **GEN** is given as a linear context-free tree grammar of input trees and an LF-transducer defining the transformation on the input trees. The constraints are given as MSO-sentences over the signature of output trees or equivalently as regular tree languages. On the other hand, we will still use finite state automata to compute optimal pairs.

GEN will be defined as a relation between two mildly context-sensitive languages. Let Σ_I be the (single-sorted) signature of input trees and Σ_O the (single-sorted) signature of output (candidate) trees. We assume that if $f \in \Sigma_I \cap \Sigma_O$ then the rank of f is the same in Σ_I and Σ_O. Let Γ_I be a linear context-free tree grammar over Σ_I and $L(\Gamma_I)$ be the language of input trees. The relation to the output trees will be defined by means of an LF-transducer. Hence let $\mathcal{A}_{\mathbf{GEN}}$ be an LF-transducer over signatures Σ_I and Σ_O. **GEN** is given as the pair $(L(\Gamma_I), \mathcal{A}_{\mathbf{GEN}})$. As a relation on trees it is defined as $\{(s, t) \mid s \in L(\Gamma_I), t \in \mathcal{A}_{\mathbf{GEN}}(L(\Gamma_I))\}$.

A constraint c is defined to be an MSO-sentence over the language of output trees of **GEN**, or equivalently, as a regular tree language over signature Σ_O. An OT-system is hence given by a linear context-free tree grammar and an LF-transducer for the generator, and a sequence of MSO-sentences as constraints.

5.1 Lifting an OT-System

In order to be able to handle an OT-system by automata we have to recode it. The intuition in this recoding is that the basic assumptions about the operations of a tree grammar, namely tree substitution and argument insertion, are made explicit. In the following, we will briefly describe this LIFTing on a more formal level. All technical details, in particular concerning many-sorted signatures, can be found in a paper by Mönnich (1999). Any *context-free* tree grammar Γ for a singleton set of sorts $\mathcal{S}$ can be transformed into a *regular* tree grammar Γ^L for the set of sorts $\mathcal{S}^*$, which characterises a (necessarily regular) set of trees encoding the instructions necessary to convert them by means of a unique homomorphism h into the ones the original grammar generates (Maibaum, 1974). The LIFTing is achieved by constructing for a given single-sorted signature Σ a new, derived alphabet (an $\mathcal{S}^*$-sorted signature) Σ^L, and by translating the terms over the original signature into terms of the derived one via a primitive recursive procedure. The LIFT-operation takes a term in $T_\Sigma(\mathsf{X}_k)$ and transforms it into one in $T_{\Sigma^L}(k)$. Intuitively, the LIFTing eliminates variables and composes functions with their arguments explicitly, e.g., a term $f(a,b) = f(x_1,x_2) \circ (a,b)$ is lifted to the term $c(c(f,\pi_1,\pi_2),a,b)$. The old function symbol f now becomes a constant, the variables are replaced with appropriate projection symbols and the only remaining non-nullary alphabet symbols are the explicit composition symbols c. The trees over the derived LIFTed signature consisting of the old linguistic symbols together with the new projection and composition symbols form the carrier of a free tree algebra T^L.

Definition 7 [LIFT] Let Σ be a ranked alphabet of sort $\mathcal{S}$ and $\mathsf{X}_k = \{x_1,\dots,x_k\}$, $k \in \mathbb{N}$, a finite set of variables. The *derived* many-sorted $\mathcal{S}^*$-sorted alphabet Σ^L is defined as follows: For each $n \geq 0$, $\Sigma'_{\varepsilon,n} = \{f' \mid f \in \Sigma_n\}$ is a new set of symbols of type $\langle \varepsilon, n\rangle$; for each $n \geq 1$ and each $i, 1 \leq i \leq n$, π_i^n is a new symbol, the ith *projection symbol* of type $\langle \varepsilon, n\rangle$; for each $n,k \geq 0$ the new symbol $c_{(n,k)}$ is the (n,k)th *composition symbol* of type $\langle nk_1 \cdots k_n, k\rangle$ with $k_1 = \cdots = k_n = k$.

$$\begin{aligned}
\Sigma^L_{\varepsilon,0} &= \Sigma'_{\varepsilon,0} \\
\Sigma^L_{\varepsilon,n} &= \Sigma'_{\varepsilon,n} \cup \{\pi_i^n \mid 1 \leq i \ \leq n\} \text{ for } n \geq 1 \\
\Sigma^L_{nk_1\cdots k_n,k} &= \{c_{(n,k)}\} \text{ for } n,k \geq 0 \text{ and } k_i = k \text{ for } 1 \leq i \leq k \\
\Sigma^L_{w,s} &= \emptyset \text{ otherwise}
\end{aligned}$$

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

$$\textsc{lift}^\Sigma_k(x_i) = \pi_i^k$$
$$\textsc{lift}^\Sigma_k(f) = c_{(0,k)}(f') \text{ for } f \in \Sigma_0$$
$$\textsc{lift}^\Sigma_k(f(t_1,\dots,t_n)) = c_{(n,k)}(f', \textsc{lift}^\Sigma_k(t_1),\dots,\textsc{lift}^\Sigma_k(t_n))$$
$$\text{for } n \geq 1, f \in \Sigma_n \text{ and } t_1,\dots,t_n \in T_\Sigma(X_k)$$

Note that this very general procedure allows the translation of any term over the original signature. The left hand side as well as the right hand side of a rule of a CFTG $\Gamma = \langle \Sigma, \mathsf{F}, \mathsf{X}, S, \mathsf{P}\rangle$ is just a term belonging to $T_{\Sigma\cup\mathsf{F}}(\mathsf{X})$, but so is, e.g., any structure *generated* by Γ. Further remarks on the observation that the result of LIFTing a CFTG is always an RTG can be also found in the paper by Mönnich (1999). To further illustrate the techniques, we present the continuation of Example 1. Note that for better readability, we omit all the 0- and 1-place composition symbols.

Example 8 Let $\Gamma^L = \langle\{a,b,c,d,\varepsilon,S_t,S_t^0\},\{S,S',\overline{S}_1,\overline{S}_2,\overline{a},\overline{b},\overline{c},\overline{d}\},S',\mathsf{P}\rangle$ with P given as follows

$$\begin{aligned}
S' &\longrightarrow c_{(1,0)}(S,\varepsilon) \\
S &\longrightarrow c_{(1,1)}(\overline{S}_1, c_{(1,1)}(S, c_{(1,1)}(\overline{S}_2, \pi_1^1))) \\
S &\longrightarrow c_{(1,1)}(S_t^0, \pi_1^1) \\
\overline{S}_1 &\longrightarrow c_{(3,1)}(S_t, a, \pi_1^1, d) \\
\overline{S}_2 &\longrightarrow c_{(3,1)}(S_t, b, \pi_1^1, c)
\end{aligned}$$

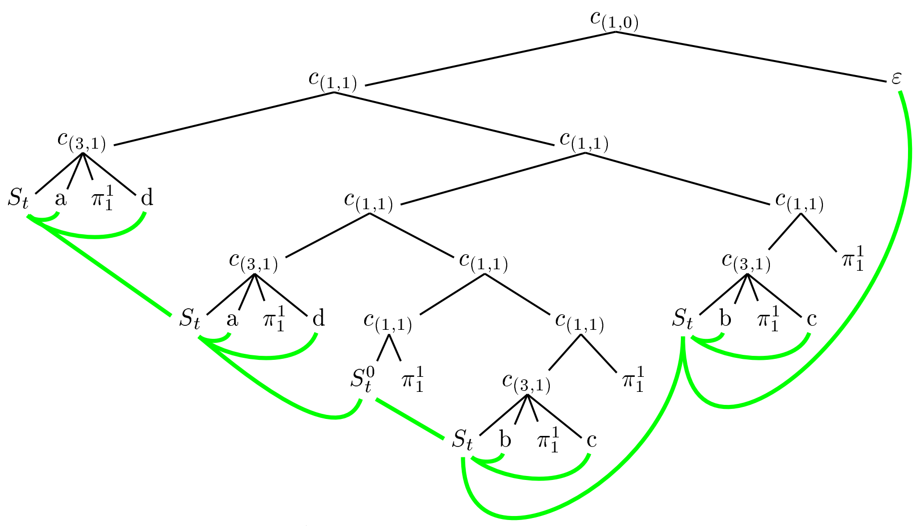

Figure 2: A LIFTed tree with intended relations.

Note that we now have only nullary operatives but extra composition and projection symbols: The linguistic non-terminals have become constants. An example tree generated by this LIFTed grammar is shown in Figure 2. It is the LIFTed tree corresponding to the sample tree of Figure 1. The grey shaded lines show how the intended tree is present in the LIFTed tree.

For every tree $t \in L(\Gamma)$ we define the set LIFT(t) of LIFTed trees of t as follows. For $s \in L(\Gamma^L)$ let $s \in$ LIFT(t) iff s is the result of a derivation sequence of grammar rules in Γ^L such that there is a derivation sequence of t and every rule in the derivation sequence of s is the LIFTed rule of the corresponding rule in t. I.e., we take the derivation sequence of t, LIFT each rule in it, and execute the resulting LIFTed derivation sequence to obtain s. The parsing step that is involved here is at most as complex as n^6 where n is a measure on the size of the tree t (Mehlhorn, 1979).

LIFTing an OT-system includes LIFTing the generator **GEN** and the constraints. **GEN** is given by a linear context-free tree grammar Γ_I and an LF-transducer $\mathcal{A}_{\mathbf{GEN}}$. The grammar Γ_I is LIFTed exactly the way defined above to obtain a regular tree grammar Γ_I^L. LIFTing of the transducer $\mathcal{A}_{\mathbf{GEN}} = (\Sigma_I, \Sigma_O, Q, P, F)$ can be defined on the basis of the insights we gain from LIFTing a grammar. We define $\mathcal{A}^L_{\mathbf{GEN}}$ as the quintuple $(\Sigma_I^L, \Sigma_O^L, Q^L, P^L, F)$ where Σ_I^L and Σ_O^L are the LIFTed signatures as defined above; $Q^L = \bigcup_{k\geq 1} Q^k$. P^L consists of the following productions. For each production $f(q_1(x_1),\ldots,q_m(x_m)) \to q(t(x_1,\ldots,x_m)) \in P$ we have $f' \to (q_1,\ldots,q_m,q)(\text{LIFT}(t)) \in P^L$ where LIFT(t) is the trivial, i.e., grammar-free LIFT of t. For each projection symbol π_i^n and states $q_1,\ldots,q_n \in Q$ there is a production $\pi_i^n \to (q_1,\ldots,q_n,q_i)(\pi_i^n) \in P^L$. For each composition symbol $c_{n,k}$ and all $p, p_1,\ldots,p_n,q_1,\ldots,q_k \in Q$ there is a production $c_{n,k}((p_1,\ldots,p_n,p)(x),(q_1,\ldots,q_k,p_1)(x_1),\ldots,(q_1,\ldots,q_k,p_n)(x_n)) \to (q_1,\ldots,q_k,p)(c_{n,k}(x,x_1,\ldots,x_n)) \in P^L$.

LIFTing of the constraints is done in a different fashion that we can explain only after having shown how to "undo" a LIFTing.

5.2 Reconstructing the Intended Trees

As Figure 2 shows, the LIFTed trees do not seem to have much in common with the trees we originally started with. Since the candidate output trees generated by $\mathcal{A}^L_{\mathbf{GEN}}$ applied to $L(\Gamma_I^L)$ are LIFTed trees, we need a way to "find" the intended tree in the LIFTed trees. In some sense to be made operational, the LIFTed structures contain the intended structures. There is a mapping h from these explicit structures onto structures interpreting the compositions (the c's) and the projections (the π's) the way the names we have given them suggest, *viz.* as compositions and projections, respectively, which are, in fact, exactly the intended structures.

On the denotational side, we can implement the mapping h with an MSO definable tree transduction, and on the operational side with a Macro Tree Transducer to transform the LIFTed structures into the intended ones. We cannot mention all the technical details here. Rather the interested reader is referred to the papers by Michaelis et al. (2001) and Morawietz and Mönnich (2001).

Let us restate our goal then: Rogers (1998) has shown the suitability of an MSO description language for linguistics which is based upon the primitive relations of immediate ($\triangleleft$), proper ($\triangleleft^+$) and reflexive ($\triangleleft^*$) dominance and proper precedence ($\prec$). We will indicate how to define these relations with an MSO transduction built upon finite-state tree-walking automata thereby implementing the unique homomorphism mapping the terms into elements of the corresponding context-free tree language, i.e., the trees linguists want to talk about.

Put differently, it should be possible to define a set of relations $R^I = \{\blacktriangleleft, \blacktriangleleft^+, \blacktriangleleft^*$ *(dominance)*, *c-command*, $\lessdot$ *(precedence)*, $\ldots\}$ holding between the nodes $n \in N^L$ of the LIFTed tree T^L which carry a *"linguistic"* label L in such a way, that when interpreting $\blacktriangleleft^* \in R^I$ as a tree order on the set of *"linguistic"* nodes and $\lessdot \in R^I$ as the precedence relation on the resulting structure, we have a "new" description language on the intended structures.

As mentioned before, we will use an MSO definable tree transduction to transform the LIFTed structures into the intended ones. The core of this transduction will be the definition of the new relations via tree-walking automata.

To do so, it is helpful to note a few general facts (illustrated in Figure 2):

1. Our trees feature three families of labels: the "linguistic" symbols, i.e., the lifted inoperatives of the underlying macro-grammar, $\mathsf{L} = \text{LIFT}(\bigcup_{n\geq 0} \Sigma_n)$; the "composition" symbols $\mathsf{C} = \{c_{(n,k)}\}, n, k \geq 0$; and the "projection" symbols Π.
2. All non-terminal nodes in T^L are labeled by some $c_{(n,k)} \in \mathsf{C}$. This is due to the fact that the "composition" symbols are the only non-terminals of a LIFTed grammar. No terminal node is labeled by some $c_{(n,k)}$.
3. The terminal nodes in T^L are either labeled by some "linguistic" symbol or by some "projection" symbol $\pi_i \in \Pi$.
4. Any "linguistic" node dominating anything in the intended tree is on a leftmost branch in T^L, i.e., it is the left-most daughter of some $c_{(n,k)} \in \mathsf{C}$. This lies in the nature of composition: $c_{(n,k)}(x_0, x_1, \ldots, x_n)$ evaluates to $x_0(x_1, \ldots, x_n)$.
5. For any node p labeled with some "projection" symbol $\pi_i \in \Pi$ in T^L there is a unique node μ (labeled with some $c_{(n,k)} \in \mathsf{C}$ by (2.)) which properly dominates p and whose i-th sister will eventually evaluate to the value of π. Moreover, μ will be the first node properly dominating p which is on a left branch. This crucial fact is arrived at by an easy induction on the construction of Γ^L from Γ.

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

6. If the node considered carries a "linguistic" label, it evaluates to itself;
7. if it has a "composition" label $c_{(n,k)}$, it evaluates to whatever its function symbol – by (4.) its leftmost daughter – evaluates to;
8. if it carries a "projection" label π_i, it evaluates to whatever the node it "points to" – by (5.) the i^{th} sister of the first C-node on a left branch dominating it – evaluates to.

Note that cases (7.) and (8.) are inherently recursive such that a simple MSO definition will not suffice.

The precise definition of the MSO-transduction from the LIFTed trees to the intended trees can be found in the papers (Michaelis et al., 2001; Morawietz and Mönnich, 2001). Let us note that the definition of the relations in the MSO-transduction can be reversed. Using the inverse of the MSO-transduction we can "lift" the MSO-sentences defining the constraints of an OT-system. What has to be done in this "lifting" is just a way of defining the dominance and precedence relations of the intended trees (which are the ones that the MSO-sentences speak about) into the dominance and precedence, composition and projection relations of the LIFTed trees. So, all we have to do is take their definitions from the MSO-transduction. The result is an MSO-sentence in the signature for the LIFTed candidate output trees. Such a sentence can be transformed into a tree automaton over signature Σ_O (see, e.g., (Gécseg and Steinby, 1997)). For a sequence of constraints $(c_1, \ldots, c_p)$ of an OT-system we obtain a sequence of tree automata $(\mathcal{A}_1, \ldots, A_p)$ over the signature of LIFTed output trees.

As stated before, there exists an operational model for the MSO-transduction, namely a macro tree transducer. The details of how to define the particular MTT we need here can be found in (Michaelis et al., 2001). In this paper, we take this MTT as given (and denote it by $\mathcal{MTT}$) and use it as the automaton that, given a LIFTed tree, actually computes the intended tree.

5.3 Finding an Optimal Pair

The idea behind the constructions and automata introduced above is now that an optimal pair $(i, o) \in \mathbf{GEN}$ can be computed in the following way. We LIFT the input tree, on the set of LIFTed input trees we run the LIFTed LF-transducer $\mathcal{A}^L_{\mathbf{GEN}}$. The result is a regular set of LIFTed output trees. These will be filtered using the automata representing the lifted constraints of the OT-system. Finally, after applying the constraints, we use the macro tree transducer $\mathcal{MTT}$ to obtain the intended output trees, which are then optimal for the given input.

Theorem 9 *Let $\mathcal{O} = \langle \mathbf{GEN}, (c_1, \ldots, c_p) \rangle$ be an OT-system. Then the pair $(i, o) \in \mathbf{GEN}$ is optimal iff $o = \mathcal{MTT}(\mathcal{A}_p \circ \cdots \circ \mathcal{A}_1 \circ \mathcal{A}^L_{\mathbf{GEN}}(\text{LIFT}(i)))$.*

Proof. Recall that a pair $(i, o) \in \mathbf{GEN}$ is optimal iff every MSO-sentence c_j is true for o ($j = 1, \ldots, p$). Note further that $\mathcal{MTT}(\text{LIFT}(o)) = o$, as was shown by Michaelis et al. (2001). The proof is based on the insight that the LF-transducer on the intended level and the LIFTed LF-transducer on the lifted level lead to the same results. I.e., let $i \in L(\Gamma_I)$. Then $\mathcal{A}_{\mathbf{GEN}}(i) = \mathcal{MTT}(\mathcal{A}^L_{\mathbf{GEN}}(\text{LIFT}(i)))$.

So, let every MSO-constraint sentence c_j be true for o. Now, $\mathcal{A}^L_{\mathbf{GEN}}(\text{LIFT}(i)) = \text{LIFT}(o)$ by construction of $\mathcal{A}^L_{\mathbf{GEN}}$. Since c_j is true for o, $o^L \in \mathcal{A}_j(\text{LIFT}(o))$ for every $o^L \in \text{LIFT}(o)$ and $j = 1, \ldots, p$. Hence $\text{LIFT}(o) = \mathcal{A}_p \circ \cdots \circ \mathcal{A}_1 \circ \mathcal{A}^L_{\mathbf{GEN}}(\text{LIFT}(i))$. By the above remark, $o = \mathcal{MTT}(\mathcal{A}_p \circ \cdots \circ \mathcal{A}_1 \circ \mathcal{A}^L_{\mathbf{GEN}}(\text{LIFT}(i)))$.

Now let $o = \mathcal{MTT}(\mathcal{A}_p \circ \cdots \circ \mathcal{A}_1 \circ \mathcal{A}^L_{\mathbf{GEN}}(\text{LIFT}(i)))$. Then $o^L \in \mathcal{A}_j(\text{LIFT}(o))$ for every $o^L \in \text{LIFT}(o)$ and $j = 1, \ldots, p$. Hence, MSO-sentence c_j is true for o for every $j = 1, \ldots, p$. That that means (i, o) is optimal. ■

We therefore obtain the following modularity result. Let $(\mathbf{GEN}, C)$ be an OT-system where $\mathbf{GEN}$ is defined by a linear context-free tree grammar and an LF-transducer, and all constraints of C are defined by MSO-sentences. Then the complexity of the OT-system is equal to the complexity of its most complex component.

6 Conclusion

The notion of optimality that we used in this paper is that of *unidirectional* optimality. We are interested in the optimal output for a given input. This view is apparently generation driven. Blutner (2000) points out that in particular in semantics and pragmatics unidirectional optimality may not suffice. The optimal interpretation of an utterance is obtained by an interplay between the generation process on the speaker side and the parsing process on the hearer side. Blutner therefore introduces the notion of *bidirectional* optimality theory. Formal properties of bidirectional OT are studied by Jäger (2002, 2003). He shows that the modularity result of Frank and Satta extends to bidirectional OT-systems on strings. Jäger (2003) also shows that for bidirectional OT-systems the restriction to binary constraints is essential to gain the modularity result.

An interesting question, that we would like to persue, is to see if the results presented here can be extended to the bidirectional case. In other words, is there a modularity result for bidirectional OT-systems over mildly context-sensitive languages of trees.

References

Blutner, R. (2000). Some aspects of optimality in natural language interpretation. *Journal of Semantics*, 17:189–216.

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

Doner, J. (1970). Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451.

Frank, R. and Satta, G. (1998). Optimality theory and the generative complexity of constraint violability. *Computational Linguistics*, 24:307–315.

Gécseg, F. and Steinby, M. (1984). *Tree Automata*. Akademiai Kiado, Budapest.

Gécseg, F. and Steinby, M. (1997). Tree languages. In Rozenberg, G. and Salomaa, A., editors, *Handbook of Formal Languages, Vol 3: Beyond Words*, pages 1–68. Springer-Verlag.

Jäger, G. (2002). Some notes on the formal properties of bidirectional optimality theory. *Journal of Logic, Language, and Information*, 11:427–451.

Jäger, G. (2003). Recursion by optimization: On the complexity of bidirectional optimality theory. *Natural Language Engineering*, 9(1):21–38.

Kolb, H.-P., Michaelis, J., Mönnich, U., and Morawietz, F. (2003). An operational and denotational approach to non-context-freeness. *Theoretical Computer Science*, 293:261–289.

Kolb, H.-P., Mönnich, U., and Morawietz, F. (2000). Descriptions of cross-serial dependencies. *Grammars*, 3(2/3):189–216.

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

Mehlhorn, K. (1979). Parsing macro grammars top down. *Information and Control*, 40(2):123–143.

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

Michaelis, J., Mönnich, U., and Morawietz, F. (2001). On minimalist attribute grammars and macro tree transducers. In Rohrer, C., Roßdeutscher, A., and Kamp, H., editors, *Linguistic Form and its Computation*, pages 287–326. CSLI.

Mönnich, U. (1999). On cloning contextfreeness. In Kolb, H.-P. and Mönnich, U., editors, *The Mathematics of Syntactic Structure*, pages 195–229. Mouton de Gruyter.

Morawietz, F. and Cornell, T. (1997). Representing constraints with automata. In Cohen, P. and Wahlster, W., editors, *Proceedings ACL 1997*, pages 468–475. ACL.

Morawietz, F. and Mönnich, U. (2001). A model-theoretic description of tree adjoining grammars. In Kruiff, G.-J., Moss, L., and Oehrle, R., editors, *Proceedings FG-MOL 2001*, volume 53 of *ENTCS*. Kluwer.

Prince, A. and Smolensky, P. (1993). Optimality theory: Constraint interaction in generative grammar. Technical Report RuCCTS-TR 2, Rutgers University.

Rogers, J. (1998). *A Descriptive Approach to Language-Theoretic Complexity*. CSLI Publications.

Shieber, S. (1985). Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343.

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

Wartena, C. (2000). A note on the complexity of optimality systems. In Blutner, R. and Jäger, G., editors, *Studies in Optimality Theory*, pages 64–72. University of Potsdam. Also available at Rutgers Optimality Archive as ROA 385-03100.