A Note on the Complexity of Optimality Systems

Christian Wartena

Universität Potsdam
Institut für Linguistik/Allgemeine Sprachwissenschaft
Postfach 601553, 14415 Potsdam, Germany
wartena@ling.uni-potsdam.de
WWW home page: http://www.ling.uni-potsdam.de/~wartena

Abstract. Optimality Theory (OT) has a growing importance in various disciplines of linguistics. The way in which correct linguistic expressions are generated according to OT can be captured formally in an Optimality System (OS). An OS defines a relation between an input and an output using a binary relation, called GEN, on the domain, and a set of constraints. Frank and Satta ([1]) have shown that an OS in which GEN is a rational relation on strings and which has only regular constraints defines again a rational relation. This result is of great importance for phonological applications of OT. For syntax, however, it cannot be assumed that GEN is a rational relation on strings. Since OT–syntax is a theory about trees, it seems natural to lift OSs for this purpose to the domain of trees and relations on trees. It is shown in this paper that the result concerning regular string languages can be extended to regular tree languages and it is sketched how such a system might work for natural language syntax.

1 Introduction

Optimality Theory (OT) ([8]) is a generative linguistic theory that is based on the idea that the main characteristics of all languages can be described by a small number of universal constraints. Typically the set of constraints is inconsistent. In the terminology of OT it is said that the constraints are conflicting. Thus a linguistic expression (a morpheme, a word, a sentence) may violate some constraints. Which ones can be violated under which circumstances is determined by a language specific ranking of the constraints. The general method to derive a grammatical sentence is as follows. We start with some underlying representation that is mapped to the *candidate set* by the function *gen*. The procedure *eval* selects the optimal candidate(s) with regard to the constraints and the ranking: A candidate w is considered to be optimal if each constraint that is violated by w is not ranked higher than any other constraint that is violated by some other candidate.

Frank and Satta ([1]) capture this procedure in the definition of *Optimality Systems*. The definition of optimality systems below is slightly different from the original one in order to accommodate the application to tree languages.

Definition 1 (Optimality System). *An* Optimality System (OS) *is a triple* $G = (D, \textsc{gen}, C)$*, where* D *is a set, called the* domain *of* G*,* $\textsc{gen}$ *is a function from* D *to finite subsets of* D *and* $C = \langle c_1, c_2, \ldots, c_p \rangle$*,* $p \geq 0$ *is an ordered sequence of total functions from* D *to* $\{0, 1, \ldots n\}$ *for some* $n \in \mathbb{N}$.

The basic idea of this definition is that $\textsc{gen}$ maps an underlying representation to a candidate set from which the constraints in C select an optimal subset. Each constraint maps a candidate to a natural number, that can be interpreted as the degree of violation of that constraint. For sake of simplicity we will only consider functions from D onto $\{0, 1\}$. In this case 0 means no violation and 1 means that the constraint is violated. For each constraint we are interested in the subset of candidates that violates that constraint to the least possible degree. This subset is defined for each constraint c by the function $\text{argmin}_c : 2^D \rightarrow 2^D$ that is defined as

$$\text{argmin}_c(L) = \{w \mid w \in L, c(w) = \min(\{c(w') \mid w' \in L\})\}$$

In the simple case in which $n = 1$ the function $\text{argmin}_c(L)$ filters out all elements from L that do not satisfy the constraint c if there is at least one element in L that fulfills c. If no element satisfies c then $\text{argmin}_c(L) = L$. The translation induced by an OS now is defined by $\textsc{gen}$ and the application of the function argmin_c for each constraint c, starting with the highest ranked constraint, c_1. Thus c_1 will be satisfied if it can be by fulfilled by any candidate whereas lower ranked constraints might be violated because good candidates are already ruled out by higher ranked constraints. Formally, we define for each OS $G = (D, \textsc{gen}, C)$ a translation function τ_G by setting $\tau_G = \tau_G^p$ which is in turn defined as

$$\begin{aligned} \tau_G^0(w) &= \textsc{gen}(w) \\ \tau_G^i(w) &= \text{argmin}_{c_i}(\tau_G^{i-1}(w)) \end{aligned}$$

Frank and Satta consider optimality systems $G = (D, \textsc{gen}, C)$ with $D = \Sigma^*$ for some finite alphabet Σ, a relation $\textsc{gen}$ that induces a rational relation on $D \times D$, i.e. $\textsc{gen}$ can be realized as a (non–deterministic) finite state transducer and with C a sequence of regular constraints, i.e. constraints that can be realized by finite state automata. They show that these systems define again rational relations. Consequently, the language generated by a regular input and an OS with the given properties is a regular language. This is an important result for phonology.

For syntax we can neither assume that GEN is a rational relation on strings nor that the constraints can be implemented by finite state automata. Moreover, syntactic constraints are constraints over trees rather than over strings. This paper intends to show that the result of Frank and Satta nevertheless is important for syntax, since we can use regular tree languages instead of regular string languages. If the domain of an optimality system consists of trees, then GEN should be a relation on tree sets that can be realized by a tree transducer. The constraints have to be implemented by finite state tree automata. If the transducer used for GEN is a linear finite state tree transducer and the input is a recognizable (or regular) set of trees the system generates again a recognizable set of trees. Recall that the string language corresponding to a recognizable set is a context free language.

In the next section I will briefly recall the definitions concerning tree languages and I will show how tree transducers and automata can be used to implement GEN and syntactic constraints. Section 3 shows how the proof of Frank and Satta can be transferred to tree languages. In section 4 the assumptions about OT for syntax that had to be made are evaluated and some relations to other systems are investigated.

2 Tree Languages

In this section I will briefly sketch the definitions of trees, tree automata and tree transducers. For a more detailed introduction the reader is referred to [4] and the references cited there.

Trees are defined over a *ranked alphabet*. A ranked alphabet Σ is a (finite) set of operation symbols each of which has a unique nonnegative *arity* (or *rank*). The subset of all m–ary symbols of an alphabet Σ is denoted by Σ_m. Given a ranked alphabet Σ we can think of trees as Σ–terms. Thus, the set T_Σ of all trees Σ–trees is defined as the smallest set T such that

$$f(t_1, \ldots, t_m) \in T \text{ whenever } m \geq 0, f \in \Sigma_m \text{ and } t_1, \ldots, t_m \in T.$$

A set $L \subseteq T_\Sigma$ is called a *tree language*.

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

2.1 Tree automata

Definition 2. *A* deterministic frontier–to–root finite tree recognizer *is a quadruple $M = (Q, \Sigma, \delta, Q_F)$ where Q is a finite set of* states*, Σ a ranked alphabet, δ a function from $\bigcup_{m \geq 1}(Q^m \times \Sigma_m)$ onto Q, and $Q_F \subseteq Q$ the set of* final states*.*

Given a finite tree t, a tree recognizer M tries to assign a state to every node of t. The assignment function δ' is defined by setting for every Σ–tree: $\delta'(f(t_1, \ldots, t_m)) = q$ if $f \in \Sigma_m, t_1, \ldots, t_m \in t_\Sigma, \delta'(t_i) = q_i$ for $1 \leq i \leq m$ and $\delta(q_1, \ldots, q_m, f) = q$. A tree t is said to be *accepted* by M if $\delta'(t) \in Q_F$. The language accepted by M is defined as the set of all recognized trees, as usual. A language accepted by a deterministic frontier–to–root finite tree recognizer is called a *recognizable* or *regular* tree language. The class of all recognizable languages is denoted Rec. The yield of a regular tree language is a context free sting language.

Example 1. Let $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$ with $\Sigma_0 = \{\text{John}, \text{sees}, \text{what}, \textit{trace}\}$ and $\Sigma_1 = \{\text{NP}, \text{V}, \text{C}\}$ and $\Sigma_2 = \{\text{VP}, \text{V'}, \text{C'}, \text{CP}\}$. We can construct a tree recognizer M that accepts all Σ–trees satisfying the following condition: If one ore more *wh*–words occur in T, then one of these *wh*–words has to be in the specifier position of CP (i.e. a direct left daughter of CP). $M = (\{q, r\}, \Sigma, \delta, \{q\})$ where

$$\begin{array}{ll}
\delta(\text{what}) = r & \\
\delta(f) = q \text{ if } f \in \Sigma_0 - \{\text{what}\} & \delta(r, q, \text{CP}) = q \\
\delta(p, f) = p \text{ if } p \in Q \text{ and } f \in \Sigma_1 & \delta(r, r, \text{CP}) = q \\
\delta(p, r, f) = r \text{ if } f \in \Sigma_2 - \{\text{CP}\} & \delta(q, q, \text{CP}) = q \\
\delta(r, q, f) = r \text{ if } f \in \Sigma_2 - \{\text{CP}\} & \delta(q, r, \text{CP}) = r
\end{array}$$

This automaton starts traveling from all leaves (simultaneously) to the root. If a leave is a *wh*–word the automaton enters the state r. All nodes dominating a node with state r get state r as well. The only node that can get the state q if it dominates a node with state r is CP, if its left daughter dominates a *wh*–word as well.

The usage of tree automata for the representation of constraints is e.g. investigated in [6]. Here the reader can find more examples of the representation of syntactic constraints by tree automata.

2.2 Tree transducers

Definition 3. *A* frontier-to-root tree transducer (F–transducer) *is a quintuple $M = (Q, \Sigma, \Omega, P, Q_F)$ where Q is a ranked set of unary operators,* *the* states *of M, Σ, Ω are ranked alphabets, the input and output alphabet, respectively, and $Q_F \subseteq Q$ is the set of* final states. *P is a finite set of productions of the form $f(q_1(\xi_1)\dots q_m(\xi_m)) \rightarrow q(t)$ where $f \in \Sigma_m$, $q, q_1, \dots q_m \in Q$, $t \in T_{\Omega\cup\Xi}$ and $\Xi = \{\xi_1, \dots \xi_m\}$ is a set of symbols disjunct from Σ and Ω.*

An F–transducer $M = (Q, \Sigma, \Omega, P, Q_F)$ defines a derives relation on trees over $\Sigma \cup Q(T_\Omega)$, where $Q(T_\Omega) = \{q(t) \mid q \in Q \text{ and } t \in T_\Omega\}$. The set $Q(T_\Omega)$ is treated as a set of 0-ary symbols. A tree s is said to *directly derive* a tree t, written $s \Rightarrow_M t$ by replacing an occurrence of a subtree $f(q_1(t_1), \dots, q_m(t_m))$ (with $f \in \Sigma_m, q_1, \dots, q_m \in Q$, $t_1, \dots, t_m \in T_\Omega$) in s by $q(t')$ if there is a production $f(q_1(\xi_1), \dots, q_m(\xi_m)) \rightarrow q(t)$. t' is obtained from t by replacing for each $1 \leq i \leq m$ every occurrence of ξ_i in t by t_i. The reflexive and transitive closure $\Rightarrow_M^*$ of $\Rightarrow_M$ is defined as usual.

Finally, we can define the translation $\tau_M : T_\Sigma \rightarrow 2^{T_\Omega}$ induced by M:

$$\tau_M(s) = \{t \mid s \Rightarrow_M^* q(t) \text{ for some } q \in Q_F\}$$

For two ranked alphabets Σ and Ω a transformation $\tau : T_\Sigma \rightarrow 2^{T_\Omega}$ is called an *F–transformation* if there exists some F–transducer M such that $\tau = \tau_M$. The class of all F–transformations denoted by $\mathcal{F}$.

Example 2. Consider how an F–transducer can define derivations from an underlying representation to a set of surface structures that. Let $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$ with $\Sigma_0 = \{\text{John}, \text{see}, \text{does}, \text{what}\}$ and $\Sigma_1 = \{\text{NP}, \text{V}\}$ and $\Sigma_2 = \{\text{VP}, \text{V'}\}$. Define a second ranked alphabet $\Omega = \Omega_0 \cup \Omega_1 \cup \Omega_2$ with $\Omega_0 = \Sigma_0 \cup \{\mathit{trace}, \text{does}\}$ and $\Omega_1 = \Sigma_1 \cup \{C\}$ and $\Omega_2 = \Sigma_2 \cup \{\text{CP}, C'\}$ and a set of state $Q = \{q_\epsilon, q_x \mid x \in \Sigma_0\}$. Finally, we define the F–transducer $M = (Q, \Sigma, \Omega, P, \{q_\epsilon\})$ by setting

$$\begin{aligned}
P = \{&f \rightarrow q_\epsilon(f) \mid f \in \Sigma_0\} \\
\cup \{&f \rightarrow q_f(\mathit{trace}) \mid f \in \Sigma_0\} \\
\cup \{&f(q(\xi_1)) \rightarrow q(f(\xi_1)) \mid f \in \Sigma_1, q \in Q\} \\
\cup \{&f(q_x(\xi_1), q_\epsilon(\xi_2)) \rightarrow q_x(f(\xi_1, \xi_2)) \mid f \in \Sigma_2, q_x \in Q\} \\
\cup \{&f(q_\epsilon(\xi_1), q_x(\xi_2)) \rightarrow q_x(f(\xi_1, \xi_2)) \mid f \in \Sigma_2, q_x \in Q\} \\
\cup \{&\mathit{VP}(q_1(\xi_1), q_2(\xi_2)) \rightarrow q_\epsilon(\mathit{CP}(\mathit{NP}(x), C'(C(\mathit{does}, \mathit{VP}(\xi_1, \xi_2))))) \mid \\
&\quad q_1 = q_x \text{ or } q_2 = q_x, x \in \Sigma_0\}
\end{aligned}$$

An example of a derivation licensed by M is given in Figure 1. Besides this more or less useful derivation, M can make a number of other derivations starting with the same input tree, reordering nothing at all or moving some other leaf into the specifier position of CP. In an OS all possible derivations starting with the same tree would define a set of candidates, the optimal one of which is determined by the constraints.

This over simplified and small example is of course inadequate from a linguistic point of view, but it might give an impression of the possibilities to compute syntactic derivations with an F–transducer.

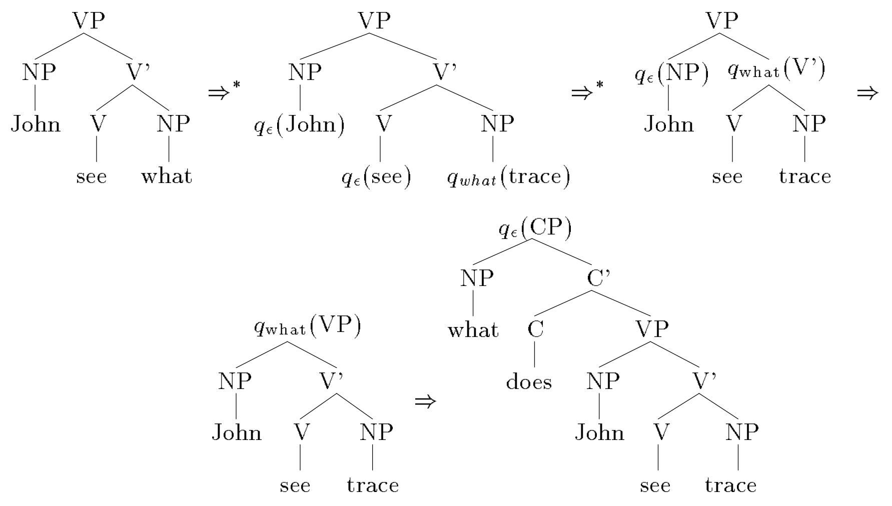

Fig. 1. Example of a derivation induced by a tree transducer

A production $f(q_1(\xi_1)\dots q_m(\xi_m)) \rightarrow q(t)$ is said to be *linear* if each ξ_i $(0 \le i \le m)$ occurs at most once in t. An F–transducer is called linear if all of it productions are linear. The class of all transformations induced by linear F–transducers is denoted $\mathcal{LF}$. A linear F–transducer cannot copy arbitrary large subtrees and for that reason preserves recognizability of a regular input. Thus linear F–transducers can be seen as the counterpart of finite state transducers at the level of trees. In order to make these statements a little bit more precise, we use the notion of *surface language*. A surface language is the image of a recognizable tree language under a tree transformation. Given a tree transformation $\tau : T_\Sigma \rightarrow 2^{T_\Omega}$ we extend τ to languages by setting (for any $T \subseteq T_\Sigma$): $\tau(T) = \bigcup_{t \in T} \tau(t)$. Given a class of tree transformations $\mathcal{K}$ the class of $\mathcal{K}$–surface languages is defined as $\text{Surf}(\mathcal{K}) = \{\tau(R) \mid \tau \in \mathcal{K} \text{ and } R \in \text{Rec}\}$.

Proposition 1 ([4]). $\text{Surf}(\mathcal{LF}) = \text{Rec}$. □

Another important property of the class $\mathcal{LF}$ is that it is closed under composition. For two classes of tree transformations $\mathcal{K}_1$ and $\mathcal{K}_2$ their composition is simply defined as $\mathcal{K}_1 \circ \mathcal{K}_2 = \{\tau_1 \circ \tau_2 \mid \tau_1 \in \mathcal{K}_1 \text{ and } \tau_2 \in \mathcal{K}_2\}$.

Proposition 2 ([4]). *$\mathcal{LF} \circ \mathcal{LF} = \mathcal{LF}$.* □

There exists an obvious relation between tree recognizers and tree transducers and it is not difficult to construct an F–transducer that does in fact nothing else as the state assignment that is done by the function δ' associated with a frontier–to–root finite tree recognizer.

Lemma 1. *For each frontier–to–root finite tree recognizer M there exists an F–transducer M' such that $\tau_{M'}(t) = \{t\}$ if $t \in L(M)$ and $\tau_{M'}(t) = \emptyset$ otherwise.* □

3 Optimality systems

Given a domain of strings or trees D we can think of a constraint c as the set of all strings or trees in D satisfying c. This set is denoted $L(c)$. Now we can say what we mean if we say that an automaton M implements a constraint c, namely $L(M) = L(c)$. From now on we will call a constraint that defines a regular set (of strings or trees) a *regular (string or tree) constraint*. Thus, given some set L of string or trees, the subset of L that satisfies c is the intersection $L \cap L(c)$. We can now reformulate the translation function associated with some OS using this intersection. If constraint c can be defined by finite state automata the set $L(c)$ is an regular set of strings (or trees). Since class of regular sets is closed under intersection a subset of D satisfying an arbitrary number of constraints is still a regular language. This observation forms the base of the proof given by Frank and Satta, that can be adopted straightforwardly for tree languages.

Given an OS $G = (D, \textsc{gen}, \langle c_1, \ldots c_p\rangle)$, we can reformulate the definition of τ_G^i for $0 \le i \le p$ as

$$\begin{array}{ll} \tau_G^0(w) = \textsc{gen}(w) & \\ \tau_G^i(w) = \tau_G^{i-1}(w) \cap L(c_i) & \text{if}\, \tau_G^{i-1}(w) \cap L(c_i) \neq \emptyset \\ \tau_G^i(w) = \tau_G^{i-1}(w) & \text{if}\, \tau_G^{i-1}(w) \cap L(c_i) = \emptyset \end{array}$$

Using Lemma 1 we can associate with each regular constraint c as well a linear F–transducer and a translation $\tau_c \in \mathcal{LF}$ such that $\tau_c(t) = \{t\}$ if $t \in L(c)$ and $\tau_c(t) = \emptyset$ otherwise. For any set T of trees $\tau_c(T)$ denotes the subset of T satisfying c. Thus we can reformulate the definition of τ_G^i for $0 \leq i \leq p$ as

$$\begin{array}{ll} \tau_G^0(w) = \textsc{gen}(w) & \\ \tau_G^i(w) = \tau_{c_i} \circ \tau_G^{i-1}(w) & \text{if} \tau_G^{i-1}(w) \cap L(c_i) \neq \emptyset \\ \tau_G^i(w) = \tau_G^{i-1}(w) & \text{if} \tau_G^{i-1}(w) \cap L(c_i) = \emptyset \end{array}$$

Proposition 3. *Let $G = (T_\Sigma, \textsc{gen}, C)$ be an OS with* GEN $\in \mathcal{LF}$ *and C a sequence of regular tree constraints, then $\tau_G \in \mathcal{LF}$.*

Proof. Let $G = (T_\Sigma, \textsc{gen}, C)$ be an optimality system with GEN $\in \mathcal{LF}$ and $C = \langle c_1, \ldots, c_p \rangle$ a sequence of regular constraints. For $i = 0$ we have $\tau_G^i =$ GEN $\in \mathcal{LF}$. Suppose the assertion is true for some i with $0 \leq i \leq p$. Now we partition D into two sets D_1 and D_2 such that $t \in D_1$ iff $\tau_G^i(t) \cap L(c_{i+1}) \neq \emptyset$. Evidently D_1 and D_2 are recognizable sets and there is some τ_D such that $\tau_D(t) = \{t\}$ if $t \in D_2$ and $\tau_D(t) = \emptyset$ otherwise. Now we define the functions $\tau_1 = \tau_{c_{i+1}} \circ \tau_G^i$ and $\tau_2 = \tau_D \circ \tau_G^i$. By Proposition 2 $\tau_1, \tau_2 \in \mathcal{LF}$. Finally we construct τ_G^{i+1} such that $\tau_G^{i+1}(t) = \tau_1(t) \cup \tau_2(t)$ for each $t \in D$. It is easy to verify that $\tau_G^{i+1} \in \mathcal{LF}$ as well. □

Corollary 1. *Let $G = (T_\Sigma, \textsc{gen}, C)$ be an OS with* GEN $\in \mathcal{LF}$ *and C a sequence of regular tree constraints, and let $L \in$* Rec *then $\tau_G(L) \in$* Rec. □

This last result says that an OS with regular tree constraints and with GEN a linear F–transformation that starts with a recognizable set of trees generates again a recognizable tree set or, looking at the yield, a context free string language. Up to now we have considered only constraints that represent functions from D onto $\{0, 1\}$. For linguistic purposes other degrees of violation might be necessary as well. Especially, it makes in many cases a difference whether a constraint is violated n or $n + 1$ times. The extension to violations of a fixed number of degrees can be taken over straightforwardly from Frank and Satta.

4 Discussion

The last corollary is relevant for OT–syntax if we accept the following three assumptions: 1. The input is a recognizable tree set; 2. GEN is a linear F–transformation; and 3. all constraints can be formulated as regular tree constraints. Unfortunately, most syntacticians are nor very explicit about the assumptions they make with regard to formal properties of the system. Thus we have to look whether our assumptions are compatible with the current praxis. As to the input most linguists seem to assume that it does not consist of trees but of a more abstract argument structure. This is nevertheless be compatible with our approach: since an argument structure is represented by terms over an alphabet of predicates and constants, by our definition of trees an argument structure is a tree!

Example 2 suggests that linear F–transducers provide quite good possibilities to do realistic derivations. Interestingly, the information that is required on the nodes between a base position and the target of a constituent that is moved is coded in a *state* of the automaton, while the labels of the nodes remain unchanged. In this respect this approach differs from the slash–feature mechanism ([2]) that enables the computation of non–local dependencies in a weakly context free formalism too. The limits of both approaches are however similar: it is not possible to move an unbounded number of elements out of one constituent. This might however not be true for natural languages. The problem could be overcome if we would use tree transducers with an additional pushdown that can be passed on in a linear way like in a linear indexed grammar ([3]). Though, to our knowledge, tree transducers have not yet been used to study formal properties of linguistic transformational derivations, the basic ideas are nevertheless present in the literature. Tree adjoining grammars ([5]) represent an important field of research using tree languages and tree grammars for natural language syntax. The idea to describe transformations with transducers was e.g. investigated by [7], who used however string transducers.

As I have already mentioned before, the idea to represent constraints by tree automata was e.g. already used by [6]. It is however not crucial that we use tree automata. A well know result of formal language theory states that a tree language is recognizable if and only if it is definable in *Monadic Second Order Logic (MSO)*. Thus, we can alternatively formulate the constraints in MSO without changing the result. The possibilities to define linguistic constraints (including nonlocal constraints) were investigated in detail by [9]. The limits on constraints definable in MSO or by frontier–to–root finite tree recognizers are the same as those mentioned for linear F–transducers. Here, we have as well the possibility to shift to more powerful automata.

This paper has shown how a result about optimality systems in the domain of regular string languages can be transferred to regular tree languages and thereby becomes relevant for optimality theoretic syntax. This result besides stresses the importance of tree automata for the study of natural language syntax.

References

1. Robert Frank and Giorgio Satta. Optimality theory and the generative complexity of constraint violability. *Computational Linguistics*, 24(2):277–299, 1998.
2. Gerald Gazdar. Unbounded dependencies and coordinate structure. *Linguistic Inquiry*, 12:155–184, 1981.
3. Gerald Gazdar. Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. D. Reidel, Dordrecht, 1988.
4. Ferenc Gécseg and Magnus Steinby. Tree languages. In Rozenberg and Salomaa [10], pages 1–68.
5. Aravind K. Joshi and Yves Schabes. Tree–adjoining grammars. In Rozenberg and Salomaa [10], pages 69–123.
6. Frank Morawietz and Tom Cornell. Representing constraints with automata. In *35th Annual Meeting of the ACL*, Madrid, Spain, 1997. ACL.
7. Martin Plátek and Petr Sgall. A scale of context sensitive languages: Applications to natural language. *Information and Control*, 38(1):1–20, 1978.
8. Alan Prince and Paul Smolensky. Optimality theory: Constraint interaction in generative grammar. ms. Rutgers University and University of Colorado, Boulder, 1993.
9. James Rogers. *A Descriptive Approach to Language–Theoretic Complexity.* Studies in Logic Language and Information. CSLI Publications, Stanford, CA, 1998.
10. Gregorz Rozenberg and Arto Salomaa, editors. *Handbook of Formal Languages*, volume 3, Beyond Words. Springer, Berlin/Heidelberg, 1997.