Universität Zürich
Institut für Informatik
Computerlinguistik

Optimality Theory and Computational Linguistics

An Overview

Lizentiatsarbeit der Philosophischen Fakultät der Universität Zürich

1. Version, September 2005

Referent:
Prof. Dr. Michael Hess

Betreuer:
Dr. Manfred Klenner

Verfasser:
Johann Fichtner
Buchzelgstr. 110
8053 Zürich
Tel. 01/382 42 70
johann.fichtner@hispeed.ch

Contents

Introduction

In recent years, many researchers have become involved in the Optimality Theoretic approach to the description of the knowledge of language. Ever since its introduction to phonology by Prince and Smolenksy (1993) Optimality Theory (OT) has attracted linguists not only in phonology, but also in syntax, and lately also in semantics and pragmatics. The concept of explaining grammaticality through conflicting violable constraints and language variation solely as the difference in the ranking of these constraints presents a clear and convincing way of expressing linguistic generalizations. Optimality Theory can be seen as a framework that is mounted upon traditional linguistic theories, therefore it is compatible with representations from a variety of linguistic approaches.

Beyond their theoretical application, OT and OT syntax bear a substantial potential for the field of computational linguistics. It is my main intention to provide an overview of research in this area and to discuss the properties of computational approaches to OT, as well as to present some considerations about the possible advantages of bringing OT into computational linguistics.

This study consists of three parts: In chapters 1 and 2, the fundamentals of Optimality Theory and OT syntax are introduced. Chapters 3 through 5 provide a discussion of computational OT, its properties and empirical issues, and its fields of application. Finally, in chapters 6 and 7 an exemplary implementation for OT syntax is presented.

Chapter 1 starts with an introduction of the main concepts of Optimality Theory. The basic explanatory devices of OT are presented with the help of phonological and syntactical examples. In chapter 2, the properties of OT syntax are discussed in greater depth, illuminating also some of the issues that have arisen from the liberty taken by linguists, how OT approaches to syntax can be settled.

In chapter 3, I discuss the aspects of introducing Optimality Theory to computational linguistics. General formalizations of the Optimality-theoretic devices are presented along with considerations about the restrictions that are necessary. Also, possible areas of application for OT syntax in computational linguistics are conceived.

Chapter 4 gives an overview of research in the field of computational OT, ranging from practical work in OT phonology to theoretical studies in the formalization of OT and language learning.

In chapter 5, I discuss a specific attempt at formalizing OT syntax, namely that of Kuhn (1999 – 2003), and its partial application in a large-scale syntactical application.

Chapter 6 introduces the background for the implementation in chapter 7. An analysis of German word order variation by OT means is presented, which postulates a constraint forcing relocation of objects due to certain features.

Chapter 7 finally presents an implementation of this analysis. The application is programmed in Prolog and is capable of generating all possible candidates for a given input and evaluating the optimal solution, as well as parsing a given input sentence and analyzing its (un-)markedness.

I conclude with a summary of findings and some open points that future research needs to address.

1. The Foundations of OT

In this chapter the general ideas of the Optimality-theoretic approach in linguistics will be presented. Optimality Theory (OT) was developed and first applied to phonology by Prince and Smolensky (1993), and quickly attracted many researchers in phonology, but also in morphology and syntax, and, lately, also in semantics and pragmatics. This study focuses on Optimality-theoretic syntax, therefore the first illustration of constraint interaction in section 1.1 comes from syntax. Nevertheless, it is helpful to introduce the basic explanatory devices with phonological examples, as will be done in section 1.2.

1.1 Constraints and Conflicts

This section gives an illustrative example of an analysis in OT syntax. It is taken from Grimshaw's (1997) elegant account of inversion in English, which is set in a syntactic framework working with a representational simulation of movement derivations in the style of *Government & Binding* theory (GB). This paper was probably the most influential early work applying OT methods to syntactic phenomena and has caused many linguists to start working in OT syntax. Here, I will just go through a simplified analysis to introduce the basic concepts of the theory.

The key syntactic constraints needed for the analysis are listed in (1.1) (Grimshaw 1997, 374):

(1.1)	OP-SPEC:	Syntactic operators must be in specifier position.
	OB-HD:	(Obligatory Head) A projection has a head.
	STAY:	Trace is not allowed.

The constraints are formulated as conditions applying to syntactic representations, in this case tree configurations following the GB theory. The concepts of head, projection and specifier are standard terms from *X-bar theory*, referring to particular positions in the *X-bar scheme* (Fig 1.1), which underlies all well formed syntactic trees. The *syntactic operators* (referred to by the OP-SPEC constraint) for the purpose are *wh*-phrases like *who* or *what*.

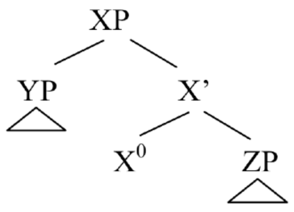


Fig. 1.1 General X-bar scheme

In general, it is not possible to satisfy all OT constraints at the same time since they can be in mutual conflict. For example: following the underlying assumptions about representations, syntactic operators cannot be generated in specifier position. In order to satisfy OP-SPEC, such operators have to be moved to this position, which inevitably will create a trace and violate the STAY constraint. It is the assumption of OT that it is legitimate for a well-formed analysis to violate certain constraints in order to satisfy some other constraints. Thus, contrary to the assumptions of earlier frameworks of generative linguistics, constraints are *violable* in Optimality Theory. By hypothesis, the set of constraints is universal; what differs from language to language is the importance (or *ranking*) of the constraints. In some languages it is more important to satisfy OP-SPEC than STAY, in other languages vice versa. This is achieved by assuming a language-specific hierarchical dominance ranking of the constraints. The dominance relation is written as "≫" ("is more highly ranked than"). For English, the dominance ranking for our three constraints is as follows: OP-SPEC ≫ OB-HD ≫ STAY. The type of relation between constraints assumed in standard OT approaches is one of *strict dominance*, meaning that higher-ranking constraints take absolute priority over lower-ranking constraints: if an analysis satisfies a higher-ranking constraint better than any alternative analysis, it does not matter how many violations of lower-ranking constraints the analysis incurs. The assumption of conflicting violable constraints comes with an inherent need to make comparisons in order to determine what is the well-formed, i.e. grammatical, analysis. This is done by the abstract process of *Harmony Evaluation* – called *(H-)EVAL* – in which a set of candidate analyses is evaluated according to the constraint ranking. In a pair-wise comparison, the most *harmonic* candidate (the "winner") is determined, where Harmony is defined as follows:

(1.2) Candidate C_1 is more harmonic than Candidate C_2 ($C_1 < C_2$) if it contains fewer violations of the highest-ranked constraint distinguishing the marking of C_1 and C_2.

Before we look at an example of EVAL at work, we have to know which candidates enter the competition for the most harmonic candidate. This is a crucial question, since obviously the result of the comparison may vary significantly depending on how "hard" the competition is. Intuitively, only genuine alternatives should be compared, meaning that all candidates should be equivalent in terms of their communicative expressiveness. Most OT approaches are not very explicit in this, "but it is a widespread assumption that all competing candidates share the same semantic content."[1] If, for the sake of clarity at this point, we assume an arbitrary representation of the semantic content, candidate sets can then be defined by a function that maps a content representation to the set of all analyses expressing this content. This function is called *GEN* (for "Generator"), and the content representation that GEN takes as an argument is typically called the *Input*. However, with regard to the present state of the theory, conceptions of the input in OT also are more than unclear and range from the assumption that the input is a fully specified and lexicalized enumeration to the one that there is no input at all. I will discuss the notion of the input in Optimality Theory at a later juncture. Note also that, while being settled in the generative framework, GEN need not be seen as a derivational process. To the contrary, OT analyses and candidates are typically *not* derivations from underlying structures (or *D-structures*) to surface structures as in GB.

In a strictly GB-based model the input should be the logical form (LF) of the sentence. For the purpose of Grimshaw's analysis it suffices to think of the input as the argument structure of the main verb, "plus a specification of the associated tense and aspect" (Grimshaw 1997, 376). Given this information, GEN generates all possible candidates. Note that GEN needs to incorporate certain inviolable principles (constraints) to verify what counts as a valid candidate. In the present example, this comprises the principles of X-bar theory and some principles on chain formation for representing the movement operations.

Thus we can now specify the input for our example in (1.3) (Grimshaw 1997, 378):

[1] Kuhn (2003, 7).

(1.3) read(x, y)
x = Mary
y = what
tense = future

Tableau 1.1 presents the standard table notation for OT analyses – called *Tableau*; the first column shows some sample candidates assigned by *GEN* for the input[2], the following columns show the constraints and their respective violations, marked by a *, with a ! standing for the fatal violation of a constraint by a candidate in pair-wise comparison with another candidate. The left-to-right order of the constraints denotes their importance, the leftmost constraint being the highest-ranked in the hierarchy and the rightmost the lowest-ranked.

Tableau$_{1.1}$

Input: read(*x*,*y*), *x* = Mary; *y* = what; tense = future	OP-SPEC	OB-HD	STAY
a) [$_{IP}$ Mary will [$_{VP}$ read what]]	*!		
b) [$_{CP}$ what$_j$ *e* [$_{IP}$ Mary will [$_{VP}$ read t$_j$]]		*!	*
c) ☞ [$_{CP}$ what$_j$ will$_i$ [$_{IP}$ Mary *e$_i$* [$_{VP}$ read t$_j$]]]			**

Candidate a) does not satisfy the OP-SPEC constraint in, as it has *what* (the *wh*-Operator) in the complement position of the verb. In candidate b), the CP does not contain a head, which leads to a violation of OB-HEAD; also, there is a violation of STAY, since *what* has been moved from the complement position of the verb to the specifier of CP, leaving behind a trace at its original position. Finally, candidate c) avoids the OB-HEAD violation (i.e., the empty C position) by also moving the auxiliary *will* from I to C; however, this incurs an additional STAY violation. We

[2] I follow the standard GB notation here: t marks the trace of the moved *wh*-word *what*; e marks an empty head (of the CP projection).

can see now how every candidate resolves the conflict triggered by the presence of the *wh*-Operator *what* in its own way. So what needs to be done now is to find out which of these ways is the best one.

The candidates are then compared to each other with regard to their constraint profile as marked in Tableau 1.1 by EVAL, taking into account the language-specific constraint hierarchy of the respective constraints, in our example the constraints named in (1.1) with their ranking for English: OP-SPEC ≫ OB-HEAD ≫ STAY. EVAL determines which candidate is the best: the corresponding candidate is denoted the most harmonic, or *optimal,* candidate: by definition, in standard OT this is the one and only grammatical analysis for the given input. The optimal candidate is then highlighted with a ☞ in the Tableau.

In our example, the winner is candidate c), regardless of the two violations of the STAY constraint. As this is the lowest-ranked constraint in the hierarchy (for English, i.e.), the violations of the two other candidates weigh heavier, and thus candidates a) and b) are predicted to be ungrammatical.

To end this section, let me sum up all components needed for an Optimality-theoretic analysis:

- The Input: A representation of the elements that GEN needs to construct all possible analyses for that input (i.e. the underlying representation shared by all candidates generated by GEN).
- The GENerator: A function generating all possible analyses for a given input building on universally valid constraints. Which constraints can be considered part of GEN is the subject of ongoing discussions: there is widespread

agreement that at least the X-Bar-Theory and Binding Principles belong to GEN.

- The Candidate Set: the set of all possible analyses for a given input generated by GEN. The Candidate Set may possibly be infinite; but if the candidate set were infinite, the input would not play any role at all and GEN would simply generate *every* possible sentence of a language (or, according to some approaches[3], principally also every language). To me, this seems inept, since we most surely do not have the lexicon of every language in our knowledge. It probably, also from a computational point of view, makes more sense to think of the Candidate Set as the set of all possible analyses for the same underlying representation (Input) with the same (or very similar, according to definition) semantic properties and meaning.
- The Constraint Set CON: A set of universal constraints shared by all languages. Languages differ not by the constraints themselves but by the ranking of the constraints. This implies that the constraints are hierarchically ordered, and in a relation of strict dominance. Contrary to other generative theories, violation of a constraint need not be fatal, but may be necessary in order to satisfy a higher-ranked constraint.
- The EVALuator: A function filtering the Candidate Set to find the optimal candidate. Falling back upon CON, it assigns a violation mark for every violation of a constraint to each candidate and compares candidates to each other regarding their constraint violations.

[3] This view comes from phonology, where it is reasonable to assume that all languages share the same phonological segments. However, in syntax this seems unlikely, as languages differ in lexicon, morphology and syntax.

We can now define Optimality:

(1.4) A candidate C1 is optimal, if there is no other candidate C2 in the same candidate set that satisfies the constraint ranking CON better than C1.

Note that according to the definition of Optimality, there must be a winner for every competition. If there is not, either the Candidate Set is incomplete or the constraint ranking is imperfect. In standard Optimality Theory this counts also for multiple optimal analyses; according to standard OT, there is only one optimal, thus grammatical, candidate. This view needs to be changed in syntax, as multiple valid analyses are not uncommon here.

We finally also know which elements a formalization of OT needs to comprise: the input representation, the GEN function, the constraints and their language-specific ranking, as well as the function EVAL. We will come back to this in chapters 3, 4, and 5.

The following sections of this chapter are dedicated to presenting the theoretical and methodological assumptions made in the Optimality-theoretic approach to linguistics.

1.2 Predictive Power

One of the central motivations for Optimality Theory is its predictive power regarding the typology of languages. Even if this does not play a central role for the theoretical aspect of this work, it is nevertheless important to discuss one of the strongest substantiations motivating an Optimality-theoretic account of linguistics in the first place. For this, phonological data will be used for, as syntax still is too much a field of ongoing research to present the topic of factorial typology. The example is taken from Kager (1999), following Kuhn (2003) in the argument.

As we have already seen in the previous section, different rankings of constraints give us different winners of one competition. Since by definition every winner of a competition is grammatical, i.e. belongs to a language defined by that particular ranking, different rankings result in different languages. What factorial typology is about, is that by the re-ranking of a set of constraints, we do not get arbitrary languages, but we find certain patterns resulting form the choice of constraints.

Using the example of Kager (1999, section 1.7), let us look at the nasality of vowels in closed syllables. The phonological analysis is supposed to predict under what conditions vowels are pronounced nasally or not nasally. Thus, in the candidate set, both the possibility of keeping the underlying nasality and a change of the underlying nasality have to be provided. Such potential changes of underlying information are controlled by *faithfulness constraints*, which favor candidates without such changes. Typically, faithfulness constraints are in conflict with *markedness constraints*, which favor certain properties of the output form independent of what the underlying form is like. We will come back to these types of constraints in section 2.2.

For the present illustration, we are only interested in whether or not a nasal vowel in the input will also be realized as a nasal in the winning candidate. This potentially depends on the successive consonant, for which it again only matters whether it is a nasal, like [n], or not. Therefore it suffices to look at the following four underlying word forms as representatives for the phenomenon we are interested in: /pan/, /pãn/, /pal/, /pãl/. The input /pan/ can be either realized faithfully as [pan], or, violating faithfulness, as [pãn] (faithfulness violations in the realization of the consonant are left aside here), and /pãn/ could come out as [pãn] or [pan]. The same applies to the other forms. Combinatorially, there are 16 different ways natural languages could behave, as shown in (1.5)

(1.5) a. {/pan/ → [pan], /pãn/ → [pãn], /pal/ → [pal], /pãl/ → [pãl]}
b. {/pan/ → [pãn], /pãn/ → [pãn], /pal/ → [pal], /pãl/ → [pãl]}
c. {/pan/ → [pan], /pãn/ → [pan], /pal/ → [pal], /pãl/ → [pãl]}
d. {/pan/ → [pan], /pãn/ → [pãn], /pal/ → [pãl], /pãl/ → [pãl]}
e. {/pan/ → [pan], /pãn/ → [pãn], /pal/ → [pal], /pãl/ → [pal]}
f. {/pan/ → [pãn], /pãn/ → [pan], /pal/ → [pal], /pãl/ → [pãl]}
g. {/pan/ → [pãn], /pãn/ → [pãn], /pal/ → [pãl], /pãl/ → [pãl]}
h. {/pan/ → [pãn], /pãn/ → [pãn], /pal/ → [pal], /pãl/ → [pal]}
i. {/pan/ → [pan], /pãn/ → [pan], /pal/ → [pãl], /pãl/ → [pãl]}
j. {/pan/ → [pan], /pãn/ → [pan], /pal/ → [pal], /pãl/ → [pal]}
k. {/pan/ → [pan], /pãn/ → [pãn], /pal/ → [pãl], /pãl/ → [pal]}
l. {/pan/ → [pãn], /pãn/ → [pan], /pal/ → [pãl], /pãl/ → [pãl]}
m. {/pan/ → [pãn], /pãn/ → [pan], /pal/ → [pal], /pãl/ → [pal]}
n. {/pan/ → [pãn], /pãn/ → [pãn], /pal/ → [pãl], /pãl/ → [pal]}
o. {/pan/ → [pan], /pãn/ → [pan], /pal/ → [pãl], /pãl/ → [pal]}
p. {/pan/ → [pãn], /pãn/ → [pan], /pal/ → [pãl], /pãl/ → [pal]}

After having listed all possible analyses, let us now turn to the constraints necessary for the analysis. The relevant faithfulness constraint is:

(1.6) IDENT-IO(nasal)

The specification of the feature [nasal] of an input segment must be preserved in its output correspondent.

Furthermore, we have two markedness constraints:

(1.7) $*\mathrm{V}_{\mathrm{NASAL}}$

Vowels must not be nasal.

(1.8) $*\mathrm{V}_{\mathrm{ORAL}}\mathrm{N}$

Before a tautosyllabic nasal, vowels must not be oral.

If the $*\mathrm{V}_{\mathrm{NASAL}}$ constraint is not dominated in the ranking, the resulting language will not feature nasal vowels at all. Such constraints are said to be *context-free.* In contrast to this, constraint (1.8) is called *context-sensitive,* as it refers to a specific following segment. Given these three constraints, six different rankings are possible (in general for a set of *n* constraints, there are *n!* possible rankings, thus the term *factorial* typology):

(1.9) a. IDENT-IO(nasal) ≫ $*\mathrm{V}_{\mathrm{NASAL}}$ ≫ $*\mathrm{V}_{\mathrm{ORAL}}\mathrm{N}$
b. IDENT-IO(nasal) ≫ $*\mathrm{V}_{\mathrm{ORAL}}\mathrm{N}$ ≫ $*\mathrm{V}_{\mathrm{NASAL}}$
c. $*\mathrm{V}_{\mathrm{ORAL}}\mathrm{N}$ ≫ IDENT-IO(nasal) ≫ $*\mathrm{V}_{\mathrm{NASAL}}$
d. $*\mathrm{V}_{\mathrm{ORAL}}\mathrm{N}$ ≫ $*\mathrm{V}_{\mathrm{NASAL}}$ ≫ IDENT-IO(nasal)
e. $*\mathrm{V}_{\mathrm{NASAL}}$ ≫ $*\mathrm{V}_{\mathrm{ORAL}}\mathrm{N}$ ≫ IDENT-IO(nasal)
f. $*\mathrm{V}_{\mathrm{NASAL}}$ ≫ IDENT-IO(nasal) ≫ $*\mathrm{V}_{\mathrm{ORAL}}\mathrm{N}$

Under the ranking in (1.9a), where the faithfulness constraint is not dominated by one of the other constraints, the unfaithful candidate loses in all four competitions. Note that given this set of data, the ranking of the two markedness constraints plays no role, so the results for a competition with ranking b) are identical. It is said that such a ranking is *not crucial*, and in the literature such rankings are commonly left unspecified: The hierarchy IDENT-IO(nasal) ≫ ($*\mathrm{V}_{\mathrm{NASAL}}$, $*\mathrm{V}_{\mathrm{ORAL}}\mathrm{N}$)

specifies all crucial rankings. The Tableaux for the four competitions are shown in Tableaux 1.2 – 1.5; cf. Kuhn (2003, 14ff.).

Tableaux$_{1.2\,–\,1.5}$

(i) Input: /pan/	ID-IO(nas)	*V$_{\text{NASAL}}$	*V$_{\text{ORAL}}$N
a) [pãn]	*!	*	
b) ☞ [pan]			*

(ii) Input: /pãn/	ID-IO(nas)	*V$_{\text{NASAL}}$	*V$_{\text{ORAL}}$N
a) ☞ [pãn]		*	
b) [pan]	*!		*

(iii) Input: /pal/	ID-IO(nas)	*V$_{\text{NASAL}}$	*V$_{\text{ORAL}}$N
a) [pãl]	*!	*	
b) ☞ [pal]			*

(iv) Input: /pãl/	ID-IO(nas)	*V$_{\text{NASAL}}$	*V$_{\text{ORAL}}$N
a) ☞ [pãl]		*	
b) [pal]	*!		*

In ranking (1.9c), the context-sensitive markedness constraint *V$_{\text{ORAL}}$N outranks faithfulness (IDENT-IO(nasal)). This means for the context in which we have a nasal consonant following the vowel, it is more important to satisfy this condition than to be faithful to the input. The respective Tableaux are shown in Tableaux 1.6 – 1.9. In the (i) case, we get an unfaithful output, realizing an underlying oral vowel as a nasal. Cases (iii) and (iv) are unaffected, since faithfulness jumps in to level out the effect *V$_{\text{NASAL}}$ could have. This phenomenon is called *positional neutralization:* In certain positions differences in the underlying forms are neutralized; before /n/, both an underlying /a/ and an /ã/ come out the same.

Tableaux$_{1.6\,–\,1.9}$

(i) Input: /pan/	*V$_{\text{ORAL}}$N	ID-IO(nas)	*V$_{\text{NASAL}}$
a) ☞ [pãn]		*	*
b) [pan]	*!		

(ii) Input: /pãn/	*V$_{\text{ORAL}}$N	ID-IO(nas)	*V$_{\text{NASAL}}$
a) ☞ [pãn]			*
b) [pan]	*!	*	

(iii) Input: /pal/	$*V_{ORAL}N$	ID-IO(nas)	$*V_{NASAL}$
a) [pãl]		*!	*
b) ☞ [pal]			

(iv) Input: /pãl/	$*V_{ORAL}N$	ID-IO(nas)	$*V_{NASAL}$
a) ☞ [pãl]			*
b) [pal]		*!	

Based on the hierarchy in d), evaluation yields Tableaux 1.10 – 1.13 as a result. With the context-sensitive markedness constraint ranking highest, followed by the context-free markedness constraints, and faithfulness ranked lowest, we get the same effect as before for the nasal context ((i) and (ii)), plus we get the effect of $*V_{NASAL}$ for the non-nasal context (cases (iii) and (iv)): we observe *allophonic variation*. In both contexts, neutralization of underlying differences occurs. Two of the four cases, (i) and (iv), show unfaithfulness of the output. This behavior is found in many dialects of English where vowels before nasal consonants (like in *sand, meant*) are nasalized.

Tableaux$_{1.10 – 1.13}$

(i) Input: /pan/	$*V_{ORAL}N$	$*V_{NASAL}$	ID-IO(nas)
a) ☞ [pãn]		*	*
b) [pan]	*!		

(ii) Input: /pãn/	$*V_{ORAL}N$	$*V_{NASAL}$	ID-IO(nas)
a) ☞ [pãn]		*	
b) [pan]	*!		*

(iii) Input: /pal/	$*V_{ORAL}N$	$*V_{NASAL}$	ID-IO(nas)
a) [pãl]		*!	*
b) ☞ [pal]			

(iv) Input: /pãl/	$*V_{ORAL}N$	$*V_{NASAL}$	ID-IO(nas)
a) [pãl]		*!	
b) ☞ [pal]			*

Let us finally look at the ranking in (1.9e), representing also the ranking in f), since here too, only the highest-ranked constraint is decisive. Once more, we get unfaithfulness in two out of four cases ((ii) and (iv)) in the Tableaux 1.15 - 1.17, where the global markedness of nasal vowels suppresses all other factors, be it the context-sensitive markedness of the alternative oral vowel or faithfulness to the underlying input form. In this language vowels are never nasal, there is a *lack of variation* for the nasality feature on vowels. Typologically, this kind of behavior is attested for different features in many languages. Our concrete case here actually holds for the majority of languages in the world. As in the previous case, it is impossible to tell from the output whether the underlying input form had a nasal vowel or not.

Tableaux$_{1.14 - 1.17}$

(i) Input: /pan/	*V$_{\text{Nasal}}$	*V$_{\text{Oral}}$N	Id-IO(nas)
a) [pãn]	*!		
b) ☞ [pan]		*	

(ii) Input: /pãn/	*V$_{\text{Nasal}}$	*V$_{\text{Oral}}$N	Id-IO(nas)
a) [pãn]	*!		
b) ☞ [pan]		*	*

(iii) Input: /pal/	*V$_{\text{Nasal}}$	*V$_{\text{Oral}}$N	Id-IO(nas)
a) [pãl]	*!		
b) ☞ [pal]			

(iv) Input: /pãl/	*V$_{\text{Nasal}}$	*V$_{\text{Oral}}$N	Id-IO(nas)
a) [pãl]	*!		
b) ☞ [pal]			*

We have seen now that out of the six possible constraint rankings specified in (1.9), only four can be distinguished. Recall that in (1.5), 16 logically possible language behaviors were observed, of which now only four are predicted: the alternatives a., b., h., and j., repeated below.

(1.10) Possible languages:

a. {/pan/ → [pan], /pãn/ → [pãn], /pal/ → [pal], /pãl/ → [pãl]}
b. {/pan/ → [pãn], /pãn/ → [pãn], /pal/ → [pal], /pãl/ → [pãl]}
h. {/pan/ → [pãn], /pãn/ → [pãn], /pal/ → [pal], /pãl/ → [pal]}
j. {/pan/ → [pan], /pãn/ → [pan], /pal/ → [pal], /pãl/ → [pal]}

With the three constraints we specified, no other language may be derived. This is a desirable result: In the languages in the world, so far only these patterns could be found. We can thus formulate the following condition for our Optimality-theoretic system (Kuhn 2003, 18):

(1.11) The typologically attested spectrum of languages should be correctly predicted by the factorial typology of the constraints assumed.

1.3 The Nature of Constraints

What remains to be clarified, is which constraints exist, and what form they have. For the above example we specified three constraints, which were intentionally chosen to reflect the typological patterns of the languages in the world regarding nasality. This setup demonstrated the machinery of Optimality Theory: With a small set of constraints, the space of logically possible formal languages is reduced to a smaller spectrum, which serves as OT's model for the possible natural languages.

What does this mean for the constraints? In phonology, phonetic circumstances can often give a very clear indication of what segments are marked or what combination of segments is marked. Such evidence can originate from properties of either articulation or perception, and it provides *phonetic grounding* of a constraint. Let me illustrate this for the markedness constraint *V_{ORAL}N: it says that an oral vowel is not allowed before a tautosyllabic nasal consonant.

Phonologically this can easily be motivated: it is plausible to assume that pronouncing an oral vowel before a nasal consonant requires more effort than the sequence of a nasal vowel and a nasal consonant. Without going into much articulatory detail here, in the first case, the velum needs to be lowered to allow airflow through the nose (in order to allow for the consonant to be nasal), whereas having both segments nasal requires no repositioning of the velum. In comparison, a hypothetical constraint *V$_{\text{NASAL}}$N (Kuhn 2003, 18) would lack such phonological motivation.

In syntax such motivations are harder to find, since many syntactic rules or constraints are freely formulated constructs, but I will give one example here anyway: A constraint like STAY (Grimshaw 1997, 374), punishing movement ("trace is not allowed"), can be motivated by assuming that it is harder for the (human) parser to analyze a structure where one or more constituents have been moved away from their original position, hence causing a higher computational effort to process the sentence.

We can thus formulate a second condition of our Optimality-theoretic system (cf. Kuhn (2003, 18)):

(1.12)
Constraints used should have an independent motivation.

What we can see from the hypothetical example above is that one should be skeptical about constraints that lack a plausible motivation and/or that cannot be formulated in a straightforward and simple way. From (1.12) we learn that our OT system should feature only the constraints needed (i.e., the fewer constraints, the better the system) and that the constraints should be formulated as simple as possible regarding their expressiveness and their logical structure (i.e. the simpler the constraint, the better). One should try to avoid constraints that fail to discriminate the candidates, i.e. that are either violated or satisfied by all the candidates. Also, unnecessarily complicated constraints are to be avoided; one should, where possible, try to split up "large" constraints into smaller, simpler ones.

1.4 Summary

In this section the basic theoretical and methodological assumptions of Optimality Theory were introduced. The use of interacting violable constraints as central explanatory device was demonstrated by means of of Grimshaw's (1997) analysis of inversion in English, and the predictive power of Factorial Typology was derived using the phonological example of nasality.

Finally, some formal assumptions on the properties of an Optimality system were drawn, finding that a) factorial typology of the assumed constraints should correctly predict the empirically attested spectrum of languages, and b) that the constraints should be independently motivated and as simple as possible.

2. Optimality-theoretic Syntax

After having introduced the basic theoretical and methodological assumptions of Optimality-theoretic linguistics, we can now focus on the fundamentals of OT syntax. As OT has been primarily introduced as a theory of phonology, some differences to the standard theory need to be accentuated. Most prominent among these is the problem of the input in syntax: In phonology, there is a limited set of possible inputs, whereas in syntax it is not clear at all, what is part of the input, in what form it is given, and what may be freely added by GEN. We will discuss this in section 2.1. Second, compared to phonology, there is a much wider range in syntax whereupon a constraint refers to. Even if this is clear, there are still many different ways how a constraint may be phrased (in natural language) and how it could be formalized. We will address the types of constraints and their form in section 2.2. Also, constraint violations and their implications for GEN and candidates depend on the form of the constraints.

2.1 The Problem of the Input

In standard OT, the function GEN maps a given Input *I* to a set of possible competing analyses for that input. These candidate structures are then evaluated with respect to a constraint set CON to find the best, i.e. grammatical, structure. In this system, the input has two functions: First, it defines the candidate set by specifying the input for GEN. In other words, only candidates referring to the same input are possible competitors. Second, faithfulness constraints refer to the input: If they penalize deviation from underlying specifications, they must somehow presuppose the existence of an input.

In standard OT phonology, it is assumed that the input consists of an underlying representation, a lexical item stored in the (mental) lexicon, i.e. phonemes. In syntax, however, it is not yet clear at all what the input should look like.

2.1.1 The Input in Syntax

We have seen at the beginning of this section that there are two tasks an input needs to fulfill in Optimality Theory: It is taken to define candidate sets, and it serves faithfulness constraints as a basis for evaluation.

In OT literature, different proposals have been made as to what the input in an OT syntax system should look like. The proposals can be classified in the following way: *non-structured input, partially structured input* and *highly structured input,* as well as *no input* at all. I address these in the following.

a) Non-structured inputs

This concept of the input resembles very much Chomsky's (1995) concept of „numeration", which constitutes the input in his „Minimalist Program". A non-structured input can be seen as a simple bag of words or lexical items, without any further specification.

If this is the case, the following could in principle be possible (Heck et al. (2002, 355)):

> Two outputs, both grammatical but otherwise radically different, emerge from one and the same input, i.e. from the same candidate set. This scenario is not desirable for two reasons: First, intuitively, two candidates should only compete if they are sufficiently similar, e.g., with respect to their semantic interpretation. Second, if two significantly different candidates compete, it is notoriously difficult to ensure that both can be optimal.

Following Archangeli and Langendoen (1997, 213–215), I will first address the ‚unwanted competition' problem (Heck et al. (2002, 355)). A totally un-structured input like {that, likes, John, Mary} gives rise to two grammatical structures:

(2.1) Input: {that, likes, John, Mary}
 a) that [$_{VP}$ John [$_{V'}$ likes Mary]]
 b) that [$_{VP}$ Mary [$_{V'}$ likes John]]

Clearly, candidates (2.1 a) and b) do not mean the same: According to that, they should not be in the same competition in the first place. Remedy could be given by specifying the input in that nominal lexical items bear case markers: only one of the candidates is optimal, or put differently, if both candidates were optimal, they would emerge from different inputs (with different case markings):

(2.2) {that, likes, John$_{nom}$, Mary$_{acc}$}

The problem seems to be solved, but easily an example is found where the specification by case is not sufficient. Consider the embedded sentence in (2.3a) (Heck et al. (2002), 356):

(2.3) a) Carl$_{nom}$ thinks that John$_{nom}$ likes Mary$_{acc}$.
 b) Carl$_{nom}$ thinks that John$_{nom,\ \textit{embedded}}$ likes Mary$_{acc}$.

The two nominative proper names produce again an ambiguity. The problem again can quickly be resolved, e.g. by marking one nominative as embedded, but as we consider even more complicated sentences, e.g. with double embedding ("Anna$_{nom}$ thinks that Carl$_{nom}$ thinks that Mary$_{nom}$ likes John$_{acc}$"), this is still not enough.

Similar findings for non-structured inputs can be drawn from binding phenomena and from wh-movement. This overall reasoning suggests that the input has to be strongly specified: It needs to be structured.

b) Structured inputs

Taking the conclusion of the last section into account, it is clear that if inputs are to define candidate sets, they necessarily need to be more complex than simple enumerations of lexical items. Literature gives several proposals as to how structured and specified inputs need to be.

Partially structured inputs (Grimshaw 1997): Grimshaw (1997) assumes the input to consist of a predicate-argument structure and some specifications about tense and aspect of the verb. Functional categories (Complementizers, Determiners, Inflection, etc.) are not part of the input but may be freely added by GEN. We recall example (2.1) from above:

(2.4) Input: (likes(John) Mary))
 a) … that [$_{VP}$ Mary [$_{V'}$ likes John]].
 b) *… that [$_{VP}$ John [$_{V'}$ likes Mary]].

Given this input with a specified predicate-argument structure, the two candidates cannot compete; only (2.4a) is a valid candidate. However, this is again not enough to ensure discrimination of candidates in embedded sentences, binding phenomena and wh-movement. Grimshaw (1997, 376) proposes for this reason that competing candidates must have the same meaning: "Competing candidates have non-distinct logical forms, in a sense which must be made precise by further research, but which certainly must entail that they are truth-functionally equivalent." Basically, there are two options how this statement can be understood: First, the logical form (LF) is not part on the input. Then only those candidates that have been generated from the same input and that additionally result in non-distinct logical forms can be in the same candidate set. This seems unattractive, since the concept of the input would be weakened: Candidate sets would have to be defined at least partially with respect to properties that are not part of the input. The second option would entail that the LF is in some way encoded in the input already (Grimshaw 1997, 376): "It may turn out that the input should include a specification of LF-related properties, such as scope." As this approach resembles to what has been proposed by Légendre, Smolensky and Wilson (1998), it will be discussed in the next section.

Highly structured inputs (Légendre, Smolensky and Wilson 1998): This approach assumes that each candidate set is fully determined by the input. For this, the input contains a "target LF", the so-called "Index". Basically this means that the structure contains information about the target scope positions of wh-phrases, entailing that the input is a highly specified structure encoding an interpretation (not necessarily the actual one). Contrarily to Grimshaw's system, in this approach it is possible for the candidates not to have the same LF-output (even though they all emerge from the same "target LF"), because (unfaithful) candidates could deviate from the "target LF" in order to satisfy higher-ranked constraints.

However, such a concept of the input raises a severe problem: Where does this highly structured input come from? One would have to assume it to be generated by another upstream generator, clearly an unattractive assumption.

We are left with the observation that the proposed definitions of input do not offer a satisfactory means to define the notion of candidate sets, because they all face some conceptual problems.

c) No input (Heck et al. 2002)

The conclusion to be drawn from the preceding sections is that the first task of inputs, i.e. to define candidate sets, is undermined by severe difficulties. We turn now to the second legitimation for inputs: The existence of faithfulness constraints. Heck et al. (2002) propose that in syntax, this is not a necessary precondition for inputs; hence, the concept of the input can be abandoned in syntax. Their key motivation for this is as simple as it is linguistically striking: Syntax is information-preserving. While in phonology, if a transformation has applied (e.g. deletion of an element) resulting in an unfaithful candidate, the original input is not (necessarily) recoverable from the output representation, this is not the case in syntax: pieces of information that are attributed to the input and possibly referred to by faithfulness constraints are still accessible in the output and can therefore be referred to by constraints on outputs too. Hence, faithfulness constraints are superfluous in syntax, and thus cannot count as a substantiation for inputs.

Consider movement in syntax: In all generative theories, if an element is moved, it leaves behind a trace at its original position. Thus, this information can be accessed in the output representation. A constraint like STAY can be formulated as "trace is not allowed", clearly a condition on the output, instead of, say, "movement is not allowed", referring to the input structure, and therefore need not be seen as a faithfulness constraint. Heck et al. (2002, 363–371) show that other faithfulness constraints can be treated alike and may be systematically reanalyzed as constraints on output representations. They conclude that "syntactic output representations are richly structured objects that can provide all the information that faithfulness constraints locate in the input [...], that the input is not needed for the definition of candidate sets in syntax [...] and that it is not needed for faithfulness constraints in syntax [...]" (ibid., 371–372). As other empirical motivations for the input are not in sight, their verdict is that the input can be abandoned in Optimality-theoretic syntax.

d) Computation

While the findings of Heck et al. (2002) linguistically are very attractive, they present no solution to the problem of the input in computational linguistic applications. An input is needed here in any case, as no system will be able to generate candidates "out of nothing". We now need to evaluate which of the above proposals (besides the "no input" approach) is suited best for the needs of a computational Optimality-theoretic system.

Conceptually, an input is needed that is sufficiently structured for GEN to create valid candidate sets, but at the same time allows for overgeneration in order to make competition and evaluation possible. Basically, the requirements for the input in computation are the same as above, as are the problems. In any case, the input needs to be enriched with information about case, thus, a simple enumeration is not sufficient. Also, some semantic representation is needed in order to generate only valid sentences. Finally, information about scope (e.g. for matrix vs. embedded sentences, quantifier and wh-movement) is indispensable. Consequently, we are led to a *highly structured input.* Kuhn (2003), who sets his formalization of OT in the LFG framework (see Bresnan 1998), thus assumes the input to be a partially specified f-structure (whereas candidates are possible c-structures for that f-structure). We will return to this issue in chapters 4 and 5.

2.2 Constraints in Syntax

The examples given in chapter 1 have already illustrated the basics of syntactic constraints in OT. However, the examples discussed there point to a potential methodological problem, in particular when we move on from phonology to syntax: above, we were dealing with "toy examples" of three constraints which have a clearly observable effect. But what if there are more constraints and their effect on the observed data is more indirect? More constraints leave us with a larger typology to check, a task quickly outreaching the feasibility of manual work. Here, a computational account clearly would be widely appreciated by linguists. Moreover, and probably worse, the issue of independent, functional evidence for syntactic constraints is even more problematic. Syntactic constraints are based on abstract representations, which are only indirectly related to observable evidence. Thus, the form that constraints take always depends in part on the kind of representations one assumes. This makes it hard to find theory-independent grounding for constraints as can be provided in phonology.

To some degree this is a problem that any theoretical account of syntax faces – the value of the representations assumed can only be judged when looking at the interplay of all aspects of the theory. The problem is even aggravated for OT syntax since the assumed representations are often inherited from another theoretical framework (Government and Binding, Minimalism, Lexical-Functional Grammar, etc.). Obviously, part of what characterizes these other frameworks is replaced by OT's mechanism of constraint interaction. This in turn may influence the form of the representations: the same representation may be suitable in the original framework, but may be unapt for the Optimality-theoretic variant of the framework, or vice versa. Notwithstanding these problems, as has been shown in chapter 1, constraint interaction is seen as the main explanatory device in OT. But the general setup of an OT system – consisting of GEN and EVAL – leaves quite some space for actual implementations: One might choose to assume a fairly restrictive set of inviolable principles (i.e., encode many constraints as part of GEN already) and only leave small space for the actual optimization step (i.e., let only some few constraints be part of CON). Or, one may assume only very few inviolable principles and leave most of the work to evaluation.

Of course, the first option clearly has the advantage that it results in only small candidate sets and making the evaluation/optimization task more perspicuous to the linguist – indeed, most OT studies in the literature focus on just some small set of candidates –, but on the other hand, this weakens the strongest theoretical motivation for OT, namely Factorial Typology. Therefore, a widely assumed methodological principle in OT is (2.5) (Kuhn 2003, 30):

(2.5) Try to explain as much as possible through constraint interaction.

This clearly favors a weak GEN component in the sense of the theory. Constraints are therefore left with the optimization work. Two types of constraints are used in OT: faithfulness and markedness constraints. These shall be introduced in the following, along with standard constraints of each type.

2.2.1 Faithfulness Constraints

These constraints provide that, for a candidate, input and output differ as little as possible. Thus, principally they are in some way connected to the input. However, as shown in section 2.1, this view can be abandoned, and faithfulness constraints may be seen as conditions on the output solely. This clearly is an advantage for a computational account, since the form of the input in syntax – as shown above – still is a matter of discussion. Nevertheless, in the sense of the theory proper, the notion of faithfulness proves useful here.

In the following definitions of constraints, I adopt the "output view" proposed by (Heck et al. 2002).

(2.6) FULL-INT (Full Interpretation)
Expletives are not allowed.

This constraint punishes the insertion of expletives by GEN. In this view, it is a faithfulness constraint, as the expletives are not in the input, thus the output deviates from the input. However, according to the above formulation, it is not a faithfulness constraint in the strict sense anymore, since it does not refer to the input. Nevertheless, its function remains the same; it just doesn't punish the insertion but the existence of the expletive itself.

Another original faithfulness constraint is STAY:

(2.7) STAY
Trace is not allowed.

Originally, this constraint is violated by syntactic movement ("Movement is not allowed"). In the version above however, there is no need to refer to movement (and thus to the input, if we assume it to consist of a predicate-argument structure, of which any deviation is punished), since the original position of the moved element still is visible as a trace in the output representation. The constraint now simply punishes the existence of traces in the output, as does FULL-INT above.

The reason why it is still meaningful to speak of faithfulness is the following: were there only such constraints as above (originally referring to the input, that is), then all derivations and even LF would look the same and the existence of expletives, movement or deletion in natural languages remain unexplained. Therefore it is essential for the theory that there exists a class of constraints having the opposite effect of faithfulness.

2.2.2 Markedness Constraints

These constraints seek to avoid marked structures that could arise from a maximally faithful representation of the input in the output. Thus, they normally ask for the compliance of syntactic shapeliness like the movement of operators to a specific operator position in the sentence. An example of such a constraint is WH-CRIT (Müller 2000, 31):

> (2.8) WH-CRIT (wh-criterion)
> A wh-operator stands in SpecC in s-structure.

If we assume a ranking WH-CRIT >> STAY in German, we deduce that in an interrogative sentence, wh-movement is mandatory:

> (2.9) wh-movement in German
> a) (Ich weiss nicht) [$_{CP}$ wen$_1$ C [$_{IP}$ Fritz t$_1$ getroffen hat]]
> b) *(Ich weiss nicht) [$_{CP}$ *e* C [$_{IP}$ Fritz wen getroffen hat]]

The originating taxonomy of constraints, of course, is very general, and there may be reason to assume a further differentiation of types of constraints on the one (linguistic) side, as well as to bring all types of constraints together in one class (as proposed by Heck et al. 2002), surely attractive for computation.

2.2.3 Consequences for GEN

Identifying something as an effect of constraint interaction as postulated by the *methodological principle of OT* (p. 32) implies that the other component of an OT system, namely GEN, has to leave open all options for this effect. Assuming faithfulness as a violable constraint means that candidate generation has to be insensitive to the preservation of the *input* information in the surface string. If we allow candidates unfaithful to the given input, we quickly may end up with an infinite candidate set, where it is not clear how the input should work. This problem in pure OT is illustrated below: If we assume an input like (2.10), a predicate-argument structure with specifications for tense (cf. Grimshaw 1997, 375), may result in candidate set (2.11):

(2.10) Input: laugh(x), x=Ann, tense = past

(2.11) a. Ann laughed
b. Ann did laugh
c. it laughed Ann
d. laughed
e. Ann
f.
g. she laughed
h. she did
i. Ann yawned
j. John yawned
k. Ann saw him
…

However, as already noted, this is not reasonable to assume. Not only were this computationally intractable, but to me, also in the sense of the theory it is not suggestive to assume candidates like i., j., or k. to be valid outputs for the input (2.10).

2.3 Summary

In this chapter we have discussed the properties distinguishing OT syntax from phonology. Firstly and predominantly, the input is one of the problems of OT syntax. I have presented four different accounts how the input in syntax could be imagined, from *no input* to *highly structured inputs,* and have addressed this question from a computational point of view. Whereas in linguistics the *no input* approach certainly is very attractive, for computation *highly structured inputs* are needed. Definitely, in this area much work remains to be done.

As for the constraints, the same conclusion is valid: phonology deals with a finite amount of features to be checked, accordingly the set of constraints is lucid. In syntax we have a different picture: sentences are potentially infinite structures with many different properties, therefore the constraint set for syntax still is sought for. Also, the form of the constraints is by all means still unclear. However, a step in the right direction was made by the finding that all constraints, i.e. markedness and faithfulness constraints, can be (re-)formulated as conditions on the structure of the output representation. This means that no falling back on any input representation is necessary for the constraints. These findings suggest that OT syntax, like phonology, may be formalized and implemented, if the necessary restrictions are made.

3. Aspects of Computational Optimality Theory

In this chapter, the aspects of combining computational linguistics and Optimality Theory are to be explored. This has two aspects: not only the possibilities of bringing some of the means and methods of Optimality Theory to computational linguistics is of interest here; also, much work in OT could certainly profit from computational approaches.

Optimality-theoretic syntax hopes for a formal and theoretical foundation, as it has already been achieved in OT phonology by the research of Michael Hammond, Lauri Karttunen, and Andrea Heiberg, to name only a few. The work done in computational OT phonology has shown that the basic assumptions and methods of OT can be formalized and implemented, to the benefit of both research areas. OT syntax still lacks this formal backing. Building upon the work of Kuhn, this endeavour aims at showing that OT syntax can be formalized for venturing implementations. Investigations in OT phonology have also shown that OT is computationally tractable. Here, we want to investigate which assumptions can be made so that these findings also hold for OT syntax.

Secondly, with implementations of OT means and methods, larger analyses in the field of syntax become much easier or, in some instances, even possible in the first place. Analyses with many different constraints get complicated and very time-consuming if they have to be done manually. OT analyses by computers will allow more and farther-reaching research in OT syntax and will certainly be a great help for linguists.

On the other side, the straightforward treatment of language variation by constraint re-ranking in OT could allow for the easier implementation of multilingual applications. Also, the EVAL mechanism of the OT grammar is of interest here; for example, one could build a grammaticality checking application with EVAL's methods.

In recent years, much research has been done also on language learning in OT. Various constraint re-ranking algorithms have been proposed (see, for example, the works of Tesar or Boersma). Automatic constraint re-ranking could facilitate work on either side, allowing for the fast generation of constraint sets for different languages given an existing set of constraints. New languages could easily be added to multilingual applications, and linguists would not have to derive new constraint hierarchies manually. We will take a look at research in computational language learning in section 4.3.

This chapter will explore aspects and possibilities of using computational linguistics in OT research in sections 3.1 and 3.2. Vice versa, we will look at the use of Optimality-theoretic methods for computational linguistics in section 3.3. In section 3.4, we will have a look at the computational complexity of Optimality Theory. Section 3.5 will conclude this chapter with a summary of findings.

3.1. Computational Methods for Optimality-theoretic Syntax

There are two main interests for linguists working in the field of Optimality-theoretic syntax in the use of computational methods in their research, the second one actually presupposing the first one. For applications that allow for larger analyses undoubtedly needed in syntactic research (the second), a formalization of the theory (the first) is needed. This poses a series of problems, as many of the issues still are unclarified in the theory. Let us begin by stating what needs to be part of an Optimality System and where there are conceptual problems:

- The Input: Maybe here already lies the most complicated part of a formalization of Optimality-theoretic syntax. Questions concerning the input are: What is part of the input? How specified need it be? We have already addressed this in section 2.1.
- GEN: This should be the lesser problem. Basically, GEN is a function generating all analyses made possible by the underlying Universal Grammar. The Generator thus could be easily built using well-known techniques, or even a pre-existing system could be modified slightly to account for the needs of an Optimality System.
- The Candidate Set: The Candidates probably should have the form of fully specified syntactic trees, which is widely agreed upon. The set itself will be more of a problem: How can we account for a possibly infinite Candidate Set and how can we make it computationally tractable?
- EVAL: This again should be solved easily. The Evaluator is a function that, for every candidate, checks whether the candidate violates the constraints

specified in the Constraint Ranking and assigns a mark to it if it does so. We should be able to implement this function straightforwardly. Also, detecting the optimal candidate can be seen as a function. As we have the constraint profile for every candidate from the first EVAL function, we just need to compare the marks assigned to each candidate starting with the highest-ranked constraint. If one candidate is superior to the others here already, this is the optimal one, otherwise, we move on to the second-highest constraint, considering only those candidates that tied for the highest constraint. We repeat this until a distinction can be made and one candidate comes out as optimal.

- The Constraint Ranking: Constraints probably pose the second big problem to our formalization, as, in the research done so far, constraints are stated in a very informal way. Consider, e.g., the constraint STAY/*t (Grimshaw (1997), Légendre (1998)), penalizing movement: Shall we explain it as "Don't move!" ("Movement is not allowed") or "Trace is not allowed!"? Depending on this, how can we formalize it giving it the same expressive power? In section 2.3 I have already hinted that constraints focusing on output representations could provide remedy, since it has been shown by Heck et al. (2002) that the expressive power of the constraint remains the same. It does not yet solve the problem of how to formalize the constraint, but clearly conditions on the output structure alone are easier to formalize than those also referring to an input representation.

3.2. Benefits for Linguistic Research

Consider the following example (based upon Müller (2000, 22–24), who follows Grimshaw (1997)):

Tableau$_{3.1}$: Negation and "do"-insertion

Input ¬(leave(*x*)); *x* = Mary		No Lexical Movement	Case	Obligatory Head	Subject	Full Inter-pretation	Stay
a)	[$_{NegP}$ Not [$_{VP}$ Mary left]]				*!		
b)	[$_{NegP}$ Mary$_i$ not [$_{VP}$ t$_i$ left]]		*!				*
c)	[$_{IP}$ Mary$_i$ [$_I$ *e*] [$_{NegP}$ not [$_{VP}$ t$_i$ left]]]		*!	*			**
d)	[$_{IP}$ Mary$_i$ [$_I$ left$_j$] [$_{NegP}$ not [$_{VP}$ t$_i$ t$_j$]]]	*!					***
e) ☞	[$_{IP}$ Mary$_i$ [$_I$ did] [$_{NegP}$ not [$_{VP}$ t$_i$ leave]]]					*	**

The above competition explains the necessity for insertion of an expletive "do" with negation in English. There are six constraints involved in the analysis, shortly explained in the following:

(3.1) No Lexical Movement: Movement of lexical heads is forbidden.
Case: The head of an NP must be assigned case.
Obligatory Head: Every projection has a head.
Subject: Sentences have subjects.
Full Interpretation: Insertion of expletives is forbidden.
Stay: Trace (movement) is not allowed.

The analysis assumes that negation introduces a NegationPosition (NegP) to the sentence structure. Since a sentence requires a subject (according to the Subj constraint), there must be an IP, causing the ungrammaticality of candidate a) (the fatal violation denoted by the exclamation mark). Candidate b) fails on account of a violation of the even higher ranked Case, since "Mary" cannot be assigned case from the verb in this structure. The presence of IP entails the presence of an I^0-Position, which also needs to be filled (Obligatory Head), inducing a mark for candidate c). However, c) fails like b) due to the violation of CASE. Candidate d) finally is ungrammatical because of its violation of NO LEXICAL MOVEMENT, as the verbal head "left" is moved to the empty I^0-Position. This leaves e) as the optimal candidate (denoted by the ☞), despite the two violations of STAY and the insertion of "do", since according to the assumed constraint ranking, "do"-insertion is permitted in order to fulfill the other constraints.

Already this fairly small and clear example requires six constraints and produces at least five candidates (and a Tableau one page wide for its analysis) – for only one property of English syntax! Now imagine we want to analyze a longer, "normal", sentence (not a linguist's toy example) in its entirety, with many different properties. It would request the linguist to split the analysis to probably one Tableau per analyzed property. Every Tableau would feature many constraints, entailing very time-consuming analyses. And, of course, the complexity of an analysis increases with every additional property to be analyzed and the overall length of a sentence.

This situation makes clear how necessary a computational approach to Optimality-theoretic syntax is deemed. Even in phonology, where one is dealing with a restricted inventory of features and a lucid number of constraints, computational approaches have become well appreciated by the scientific community. By facilitating work for linguists and thus allowing for more complex and larger analyses, implementations could allow research in Optimality-theoretic syntax to start off in the first place.

However, such an OT syntax system would require a large number of the necessary constraints to be known and a determined constraint ranking, both still far from realization. Also, the system need be flexible in the sense of allowing for the ranking to be changed and constraints to be added or modified by the user.

3.3 OT Methods in Computational Linguistics

Let us now turn to the possibilities that could arise from bringing OT to computational linguistics. The main point here will not be to criticize work in (computational) syntactic analyses. Instead, I will try to point out some issues that could be ameliorated by an OT treatment.

3.3.1 Word Order Variation

Syntactic Analyses (and not only computational ones) very often do not account for the flexibility of language. This is partly due to the system of phrase structure rules predominantly used for such analyses, which can account for some, but by no means for all of the variability in languages. Through movement, e.g., variations in word order can be derived, but only accompanied by costly processes like generating new phrase structure positions and attaching them to existing positions. Consider, for example, the following sentence:

(3.2) Sie hat die Kinder dieser Gefahr ausgesetzt.
She has the children to this danger exposed.
'She exposed the children to this danger.'

This example is chosen because of the OT analysis of German word order variation (scrambling) presented in chapters 6 and 7. It shows the unmarked word order in the "Mittelfeld" (the area between the finite and infinite verbal parts in a sentence). Now, what if, in a given context, we would like to stress *dieser Gefahr* ('to this danger'), e.g. if one wants to show his indignation or lack of understanding of the facts? The following sentence could be produced then:

(3.3)	Sie hat *dieser Gefahr* die Kinder ausgesetzt!?
	She has *to this danger* the children exposed!?
	'She exposed the children *to this danger*!?'

The dative NP now precedes the accusative NP, deviating from the normal order and resulting in a marked, but still acceptable sentence. (Note, by the way, that there exists no variation in the English sentence. A topicalized phrase in English would be moved out of the "Mittelfeld" into the sentence-initial position.) In a generative approach like Government and Binding Theory (GB; see, e.g., Haegeman 1994), this would probably be analysed in the following way: The dative NP *dieser Gefahr* ('to this danger') is moved from its base position (the NP position right to the left of the infinite verb) to a newly generated position adjoined right before the accusative NP (see figure 3.1).

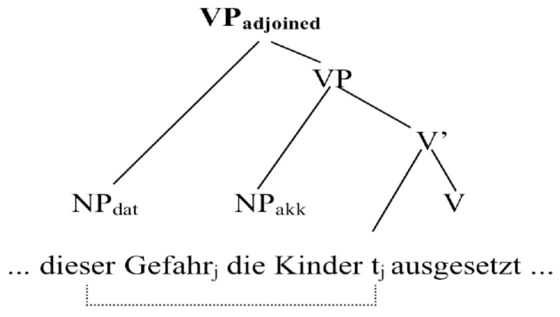


Fig. 3.1 Adjunction of an extra VP position in the Mittelfeld

To me, the generation of a new position does not seem a plausible explanation, for it is unclear where to generate it; furthermore, the process seems costly and thus psychologically questionable. We could, of course, propose an already existing position at that location, like TP (TopicPhrase), FP (FocusPhrase), etc. (see figure 3.2 for an illustration). However, in my opinion this would lead to a similar problem: Is it plausible to assume an unfilled position at every possible location in a freely extensible X-bar structure to which a moved phrase could be attached?

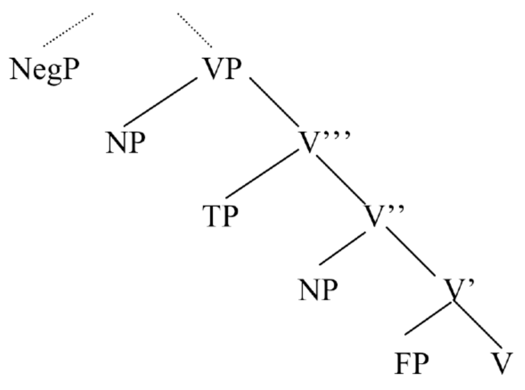


Fig. 3.2 Extended sentence structure

With this example I have demonstrated the difficulty of this from the point of view of a generating system. The problem also exists for a parser, even if in a slightly different way. Suppose a parser is given this sentence as input:

(3.4) Sie hat dieser Gefahr die Kinder ausgesetzt.
She has to this danger the children exposed.
'She exposed the children to this danger.'

The parser, if built on GB-like phrase structure rules, would come up with an analysis like 3.5:

(3.5) [$_{SpecCP}$ Sie$_j$ [$_{CP}$ hat$_i$ [$_{IP}$ t$_j$ [$_{VP/I''}$ dieser Gefahr [$_{V'/I'}$die Kinder [$_{V/I}$ ausgesetzt t$_i$]]]]]]

This is a correct output, of course, but the fact that the dative NP was moved from its base position is missed, and we have no information about why this particular movement happened. Exactly this information, however, is important or even necessary for the semantic understanding of the sentence and could thus be of great use for, e.g., a question-answer system or in machine translation. Please remember that we have seen that contrary to German, no such word order variation exists in English. Also, topic or focus positions may vary between languages, so that, whereas in German a stressed item is moved to the left, it could be also moved to the right in another language. This even aggravates the problems of the analysis as outlined above.

Here, remedy could come from a constraint-based approach like Optimality Theory. Envisage an analysis of the above example with the means of constraints based on features like focus, topic, or case, to name only a few. With such constraints, the word order could simply be analyzed in terms of linear precedence and alignment. The unmarked order would be *accusative* > *dative* and *topic* > *focus* (where "˃" stands for the precedence relation).[1] *The topic* > *focus* rule would then outweigh the *accusative* > *dative* rule, meaning that a dative NP bearing a feature that marks it as topicalized would have to precede an accusative NP. A similar approach could be envisaged in terms of alignment, where one could say that a topicalized NP would have to be as close as possible to the left edge of the "Mittelfeld", whereas a focused phrase would align to its right edge. Such approaches have been proposed in Müller 1998, 1999, 2000, and other work.

A generative system will simply place a phrase (possibly marked as topicalized) in front of any other constituent and follow the unmarked order if not given any specifications. A parser could derive valuable semantic information from the word

[1] This is only a simplified proposal, of course; there are more features involved, e.g., definiteness and animacy. See chapter 6 for a detailed approach.

order of the sentence it is given. If a dative NP is encountered before an accusative NP (in left-to-right parsing direction), the parser will mark it as topicalized, since only under this condition it can precede the accusative.

Now, where could such a system be of some use? I have already mentioned QA systems and machine translation as possible applications, coming to one's mind immediately here. In the following, I will try to sketch what could be envisaged here.

In an answer sentence, new information (i.e., the actual answer to the question) is given at a specific location in the sentence. In German, this would be the focus position, localized at the right edge of the "Mittelfeld":

(3.6) Er hat das Buch dem Mann gestohlen.
He has the book$_{acc}$ the man$_{dat}$ stolen. (Unmarked order)
'He stole the book from the man.'

Q: Wem hat er das Buch gestohlen?
'Whom did he steal the book?'
A: Er hat das Buch *dem Mann* gestohlen.
'He has the book$_{acc}$ the man$_{dat}$ stolen.

Q: Was hat er dem Mann gestohlen?
'What did he steal from the man?'
A: Er hat dem Mann *das Buch* gestohlen.
'He has the man$_{dat}$ the book$_{acc}$ stolen.

Note that in any context and independent of any other features, the information is given at the right edge of the "Mittelfeld". A (German) QA system would easily be able to deviate from the unmarked word order if necessary and/or appropriate by outputting the answer with the "knowledge" of such a constraint- and feature-based approach. It would be able to derive the necessary information for the correct placement of the new information by the word order of the question: Normally, the question pronoun is fronted (like *Wem* or *Was* in the above examples). Then, by simple matching of features, it could place the inquired information at the right edge of the "Mittelfeld", i.e. in other terms, the focus phrase FP. Such a system would already come very close to human language.

Let me clarify: I do not want to say here that existing QA systems would not be able to do this by other means, or that the same could not be done by some other syntactic approach, but the methods of an alike OT analysis strike as very attractive here.

Similarly, such an analysis could be used for deriving the correct word order in translations. Differences in topic and focus positions could be stated as different alignment rules based on other constraint hierarchies as usual in OT, and different word order could be analyzed in terms of distinct precedence relations, likewise derived from differences in the constraint ranking of the respective languages.

3.3.2 Markedness

In this section, I will deviate from the notions of the so-called "Pure OT". "Pure OT", as it has been introduced in chapters 1 and 2, only allows one candidate to be grammatical, as every other candidate inferior regarding his constraint profile is seen as ungrammatical. This again, however, does not account for the variability of natural languages. As is often the case, particularly with scrambling problems, other candidates in the same set are not simply ungrammatical but are dispreferred in comparison with the winner of the competition, or are denoted “more marked”. Pure OT does not account for this, but can be extended easily to allow such a treatment. Constraint violations in a competition allowing more than one winner would invoke some kind of markedness and not ungrammaticality. Markedness and ungrammaticality then could be distinguished in the following way: markedness could arise from competition between constraints in a constraint subset embedded in the actual constraint hierarchy, whereas only a fatal violation in this “master” hierarchy could invoke ungrammaticality, as suggested in Müller 1998, 1999, etc., and explained further in chapter 6.

Another way of treating markedness is with the help of probability values. Various kinds of “Stochastic OT” have been proposed (see Vogel (2005)) and are often used in language learning systems; I will briefly present the basic notions of such a system here. In Stochastic OT, constraints are weighted with probabilistic values. Depending on these, the candidates are assigned a specific value, and the candidate with the best value will be the winner of the respective competition. To account for markedness there are two possibilities. One would be to set a threshold dividing the values into markedness and ungrammaticality values, respectively. A value above that threshold would mark the candidate as a potential winner (with an eventual markedness attribute), whereas a candidate with lower probability (i.e. a value below the threshold) would be denoted as ungrammatical.

The second option would be to forgo such a threshold but just to use the derived probabilistic values as markedness indicators. This option would allow one to even measure the markedness of a specific candidate from its probabilistic value, and thus to denote one candidate as more marked than another, or vice versa.

The most promising way of treating markedness would probably then be to combine these two accounts into one, as the first one lacks a measure for markedness, and the second would not be able to account for real ungrammaticality if applied in the way described above. A means to derive markedness measures and real ungrammaticality would come closest to the human language processing system.

To me, though deviating from "Pure OT", the fundamental strength of Optimality Theory would lie exactly in the methods outlined above to analyze such markedness phenomena: the EVAL mechanism could provide an excellent means for "rating" the markedness of different variation of a sentence.

3.3.3 Making Use of EVAL

Let me turn now to the heart of an OT system, the EVAL mechanism, and its possible uses for computational application. While the function of GEN in an OT system makes the purpose of this device clear from the beginning, this is not the case with EVAL. The reason for that is quite obvious, as such a device has not been applied to linguistic computation so far.

Let us quickly look at what the Evaluator does: EVAL is a function that assigns violation marks to a candidate for every constraint violated by that candidate. Given the constraint hierarchy CON, it thus allows finding the optimal (grammatical) candidate. Now, how can we make this task useful in computational linguistics? The first thing that comes to one's mind originates from sentence generation (and resembles the main Optimality-theoretic method): given a very simple generator – like GEN in OT – that only incorporates basic universal linguistic devices and constraints and thus generates more than one possible candidate, EVAL can be used to find the sought optimal output. If we assume different constraint rankings, EVAL can look for alternatives to that sentence – maybe there is an option that is not optimal (but may still be grammatical) but that allows a better understanding. This second task clearly deviates from the original sense of OT, but recent research has begun to incorporate the "understanding" side into OT analyses. Blutner (2000), building upon research in Optimality-theoretic semantics (see Blutner (2000) for references) has proposed a "bidirectional OT" that combines the generation direction and (going the opposite way) the understanding direction into one account. The key argument is the following: there may be a candidate suboptimal in generation that is optimal in the sense of understanding. Evidence for this comes from the interaction of nouns or proper names and bound pronomina in sentences: the repetition of the base noun may lead to an easier understanding as the grammatically more optimal option of a related pronoun. Consequently, Kuhn (2003) incorporates Blutners account of bidirectionality in his formalization of OT syntax.

Bearing this bidirectional OT in mind, the possible use of EVAL in allowing for alternatives seems striking: we can now generate different possible analyses of one and the same input, each of the alternatives suitable for a certain purpose. This not only could bring natural language systems closer to actual natural language, but it also incorporates (a kind of) semantics in syntactic analyses.

A second application of EVAL directly utilizes this "understanding" direction of bidirectional OT. Assume the output of a OT generation to be the input for a "backward evaluation", then we could make use of EVAL as a "grammar checking" device in the sense that it collects the constraint violations of the output and compares it to those of other possible candidates (that, in principle, could be generated instantly by GEN, given another device that maps the output representation back to the underlying input. Thus, EVAL can be used to check whether a sentence really is the optimal (and therefore grammatical) representation for the underlying input. This could be an alternative to existing automatic "grammar correction" systems.

Also, in combination with the application presented first, EVAL could be used to find out whether there exists a better candidate facilitating understanding for that input.

3.4 Remarks on the Computational Complexity of Optimality Systems

The topic of tractability was already mentioned at some points, mostly when there was need to modify or even restrict some device of the OT system. For example, it is clear that an infinite candidate set is unattractive for computing (even though it could be handled in principle), resulting in the need to modify the input and possibly restrict the capabilities of GEN. From the linguistic point of view, one can blame this to computational accounts. On the other hand, we have to keep in mind here that no consensus has yet been arrived at in research about what the input should look like in OT syntax and what exactly GEN should be capable of doing (cf. sections 2.1 and 2.2). Therefore what seems like a reproach could really be a chance for the theory to prove itself, namely that a precise formalization can be found.

Secondly, we need to see whether we can even compare potential OT applications to "traditional" computational linguistic applications. For it is clear that a OT-based pure generation system probably loses in comparison to a genuine, say, GB-based generator, as it is built up of many more devices and thus is much more complicated. The same probably applies for parsers. As we have seen above, OT systems are not to compete with such traditional applications, but could enhance them in areas were these systems are deficient. Therefore we need to ask ourselves whether it is reasonable to compare OT and traditional systems in the first place, since they probably won't fulfill the same tasks: traditional applications – at least at the moment – won't be replaced but will be the basis whereupon OT systems build up.

Third, it is hard to talk about tractability of OT systems at the moment. At this point, at least to my knowledge, there exists no implemented full-scale OT-Syntax system, so it is not yet known how such a system would do in tests. As noted above, we cannot compare OT and traditional systems. It, thus, is unlikely that the same criteria for evaluation of such systems apply, making a serious statement impossible at the moment. Much research is needed here, in linguistics as well as in computational linguistics.

3.5 Summary

In this chapter we have explored what the mutual benefits of bringing Optimality Theory and its methods into the field of computational linguistics could be. The general conclusion that can be drawn is that, while in phonology computational OT applications are well investigated and widely appreciated devices for research, in syntax too much is yet unclear in the theory for computational approaches to become possible. The vicious circle is that in order for the theory to become more profound, computational aid would be almost necessary, since larger-scale manual analyses are very time-consuming and can get confusing, as I have shown in section 3.2. Nevertheless, the basic functioning of an Optimality System (as presented in section 3.1) is quite clear, and a formalization and implementation of the kernel devices GEN and EVAL as functions should be easily conceivable.

From the viewpoint of computational linguistics, there is a field of application for OT imaginable, wherever traditional approaches get problematic. In the first place, OT could provide a helpful means when dealing with variation and markedness, even if this usage deviates somehow from the basic theory. Also, multilinguality is an area where OT is superior to traditional syntax theories. There is no need anymore for many different grammars in multilingual applications; the set of grammar and constraint rules remains the same for all languages. The only thing necessary is the knowledge of the specific constraint ranking for each language desired.

Finally, some considerations were made regarding the computational complexity and tractability of OT syntax systems. We have found that it is unclear what can be said at the moment, since, as mentioned above, no complete OT syntax system has been examined yet, neither manually nor computationally, since simply none exists. However, as we will see in chapters 4 and 5, with the right presuppositions and restrictions, OT syntax systems are computationally tractable.

4. The Computability of Optimality Theory

In this chapter, I will discuss some approaches to computational Optimality Theory, especially those that prove useful in the computational treatment of OT syntax.

Already shortly after its introduction, several applications in Optimality-theoretic phonology have been developed, which have attracted researchers in both areas ever since. I will not go into any details regarding computational phonology but recapitulate some of the most important findings in section 4.1. In section 4.2, we will have a look at some formal preliminaries for computational OT syntax. Section 4.3 is dedicated to the topic of language learning. Put simply, languages equal to specific constraint rankings in Optimality Theory; consequently, the task of language learning is modeled by (re-)ranking of the universal constraints. This can be a very expendable assignment if done by hand, as rankings in syntax will grow very large. Various algorithms fulfilling this task have already been proposed in the literature and are presented here.

4.1 Overview of Work in Computational OT Phonology

Ellison (1994, 1995) presents a framework for OT phonology computation led by the assumptions that (i) all constraints are binary or can be recast as binary constraints and (ii) that the candidates, i.e. the output of GEN, is a regular set which can be specified by a finite-state automaton (Ellison 1995, 6). On this basis, (finite-state) transducers are defined that are able to compute not only single constraints, but ordered hierarchies of constraints. An algorithm for a product operation on these transducers is formulated, allowing constraints to be applied to candidates. Furthermore, Ellison proposes algorithms for finding the optimal candidate and for culling non-optimal paths from the transducer.

The work of Ellison is refined in Eisner (1997), who introduces a minimalistic variant of OT called primitive Optimality Theory (OTP). By also setting up representations and constraints more efficiently using deterministic finite-state automata and by making use of a more compact ("factored"; Eisner 1997, 320) representation of automata, he is able to reduce the computational complexity of Ellison's algorithms.

Karttunen (1998) presents a novel formalization of Optimality Theory for computation. Unlike the above-mentioned approaches it does not require explicit marking and counting of constraint violations. Instead, Karttunen introduces the notion of "Lenient Composition", defined as the product of *ordinary composition* and *priority union.* To achieve greater efficiency, the GEN relation and all constraints are combined into a single finite-state transducer mapping each underlying form directly to its optimal surface representation. He concludes that, seen this way, OT phonology strongly resembles two older strains of computational phonology: classical rewrite systems and two-level models.

Hammond (1995, 1997) proposes to implement syllabification in OT as a parser. The most striking problem of implementing OT, i.e. the problem of (possibly) infinite candidate sets, is alleviated by hypothesizing that the parser must not feature epenthesis and deletion. This makes sense in so far as the parser is intended solely to model a speaker's ability to judge syllabic wellformedness of (real and nonsense) words. Again, GEN is modeled as a finite-state transducer. The second innovation is that EVAL is applied constraint by constraint, resulting in savings of computation time. The two variants are implemented as Perl (1995) and Prolog (1997) programs, respectively.

Tesar (1994, 1995) addresses the parsing problem in Optimality Theory from a different perspective. He proposes parsing algorithms using dynamic programming techniques to compute the optimal structural description of an input. For grammars employing structures described by regular grammars, he finds parsing time to be linear to the length of the input. For grammars, whose structures are described by a context-free grammar, the algorithm takes time cubic to the length of the input to compute. The most interesting findings for our purpose here, however, may be the following: the prospect of an infinite candidate set generated by GEN need not be feared. A grammar is a function, not an algorithm, and algorithms need not mimic every aspect of the formal description of that grammar in order to be able to compute the function defined by that grammar (Tesar 1995, 111).

Finally, Heiberg (1999) models phonological features in Optimality Theory computationally. For that, she makes three explicit claims on the nature of computational OT: First, a list of GEN operation types is enumerated; second, a set of constraint types and methods for evaluation are defined; third an efficient method for creating and evaluating candidates, as well as finding the most harmonic (i.e., optimal) output is specified. For computation, an algorithm of the form of a "GEN-EVAL-loop" is proposed: For each constraint in the hierarchy (starting with the highest), new candidates are generated by applying "relevant" GEN operations to the existing set of (optimal) candidates. Then, the candidates are evaluated against the constraint hierarchy up to the current constraint, and the candidate set is culled according to that partial evaluation. Finally, any GEN operations that always decrease harmony of a candidate with respect to the constraint are removed from consideration. The algorithm continues this way creating, evaluating, and eliminating candidates until the end of the constraint hierarchy is reached and the optimal candidate is discovered. Three efficiency considerations are made: Consider as few GEN operations as possible; consider as few candidates as possible; do as little evaluation as possible. The performance of the algorithm depends on the nature of the constraint hierarchy: In the worst case, it is exponential in the size of the input representation. This leads to an interesting conclusion: Certain rankings are more efficient than others for the "GEN-EVAL-loop". This suggests a new way of looking at constraint rankings, based on computational efficiency (and not only linguistic tenability). As for the implementation of the system, the representations, constraints, and GEN operations are presented as object-oriented classes in JAVA.

This presents only a small part of the work done in computational OT phonology, of course. The goal was to give a quick overview of what has been achieved so far, possibly bearing consequences for our topic. The reader is referred to the literature cited above for further reading. Also, work in computational OT morphology is left aside here: Trommer (1999) extends work in computational OT phonology to morphology, basically achieving similar results with the same means of finite-state techniques. Again, the reader is referred to the intrinsic literature.

4.2 Formal Properties and Problems of OT Syntax

When one intends to apply the comparison-based definition of grammaticality of Optimality Theory directly to computational syntax, some formal and computational problems are to be faced. The main focus is on a set of problems arising from the complexity of processing tasks (generation and parsing or recognition) when working with an OT grammar. As has been noted before, a massive *Complexity Problem* arises when a large (potentially infinite[1]) set of highly complex structural descriptions (i.e., the candidates) has to be computed and checked for possible constraint violations for the determination of grammaticality.

The complexity problem can be divided in two aspects: (i) What restrictions can be imposed on the formalism to make the processing tasks decidable (i.e., guarantee that algorithms can be formulated that can perform these tasks? (ii) Can further restrictions guarantee that the processing complexity lies in a realistic complexity class? – This work primarily focuses on aspect (i).

Second, a *Conceptual Problem* arises: To account for language-particular ineffability, one needs to assume a candidate unfaithful to the logical form (LF) of the input to win the competition (Légendre, Smolensky, and Wilson 1996). It does not seem

[1] As shown by Tesar (1995, see above), the potential infiniteness of the candidate set itself is not a problem. The dynamic programming techniques developed in his work are able of dealing with such infinite candidate sets. "The challenge for a computational treatment of OT syntax is to come up with algorithms for grammars beyond context-freeness, and without the input being restricted to a strictly ordered sequence of elements." (Kuhn 2000, 1).

implausible (but still rather obscure) that no interpretation at all is assigned to LF-unfaithful winners. Viz., this would require an additional mechanism applying after EVAL that checks the winner's LF against the input LF and nullifies the analysis in case of divergence. This, however, presents a problem regarding the theory of OT syntax itself and shall not be of any concern for our topic.

The third problem is an *Ambiguity Problem:* Production and comprehension do not differ in terms of the algorithm applied for selecting the most harmonic (=optimal) candidate. In production, the candidates share the same underlying representation, in comprehension the same surface form (Kuhn 2000, 2). However, this is incompatible with the phenomenon of ambiguous surface forms, as it is predicted that only one reading (the most harmonic one) should exist. There exist several different concepts of dealing with ambiguity, e.g. the concept of tying constraint (i.e., two constraints have the same strength, or share one location in the hierarchy). I refer to Müller (2000, chapter 5) for further reading on this topic and won't discuss it any further here.

Building on the work of Ellison (1994, 1995) and Karttunen (1998), Frank and Satta (1998) prove in a somewhat more formal way that the GEN relation, i.e. the mapping of input representations to possible output forms, is a rational relation and representable as a finite-state transducer. Furthermore, they propose that constraints can be seen as functions from strings to non-negative integers. Assuming that the reader is familiar with the notion of finite-state automata (FSA), let me briefly recapitulate the concept of finite-state transducers (FST). A FST is a

FSA whose transitions are defined over the cross-product set ($\Sigma \cup \{\varepsilon\} \times (\Delta \cup \{\varepsilon\}$)), where Σ and Δ are (finite) alphabets. If Σ is interpreted as alphabet of input to the machine and Δ as alphabet of the output, "each accepting computation of the transducer can be viewed as defining a mapping between a string in Σ^* and a string in Δ^*. […] Every finite-state transducer can be associated with what is called a rational relation, a relation over $\Sigma^* \times \Delta^*$ containing all possible input–output pairs. A rational relation R can also be regarded as a function $[R]$ from Σ^* to 2^{Δ^*}, by taking $[R](\mathrm{u}) = \{v \mid (u,v) \in R\}$ for each $u \in \Sigma^*$." (Frank and Satta 1998, 3–4).

Formally, an Optimality System (OS) is defined in the following way (Frank and Satta 1998, 4):

> (4.1) Definition: *Optimality System*
> An Optimality System (OS) is a triple G = (Σ, GEN, C), where Σ is a finite alphabet, GEN is a relation over $\Sigma^* \times \Sigma^*$ and C = $\langle c_1, \ldots , c_p \rangle$, $p \geq 1$, is an ordered sequence of total functions from Σ^* to N.

N is denoted the set of non-negative integers. The basic idea underlying this definition is as follows: If w is an underlying representation, [GEN](w) is the non-empty set of all associated surface forms, otherwise [GEN](w) = 0. For a given surface form w, the integer $c(w)$ is the degree of violation that w incurs with respect to the represented constraint. Given a set of candidate S, we are interested in the subset of S which violated c to the least degree, i.e. whose value under the function c is lowest (Frank and Satta 1998, 4).

Additionally, each constraint c is assumed to be regular in that it satisfies the following condition: For each $k \in$ N, the set $\{w \mid w \in \Sigma^*, c(w) = k\}$ (the inverse image of k under c) is a regular language. Put differently, the set of candidates violating a given constraint to any particular level must be regular (Frank and Satta 1998, 5). Most of the constraints proposed in the OT phonology literature are regular in this sense; however, this does not (necessarily) apply to OT syntax constraints. With these settings, an OS as the one outlined above can always be implemented as a FST so long as each constraint of the system satisfies the additional restriction of having a finite co-domain, i.e. "that it distinguishes only a finite set of equivalence classes of candidates" (Frank and Satta 1998, 5).

For every surface form $w \in \Sigma^*$ and for $0 \leq i \leq p$ of the Optimality System G (see the definition above), a function from Σ^* to 2^{Σ^*} is defined. This function is called *Optimality Function* (Frank and Satta 1998, 4) associated with G, and is denoted OT_G. If all necessary preconditions introduced above are met, the following result holds (Frank and Satta 1998, 7):

(4.2) Let $G = (\Sigma, \text{GEN}, C)$ be an OS such that GEN is a rational relation and each constraint in C is regular and has a finite co-domain. Then OT_G is a rational relation.

This result is of significant importance for phonological applications of OT. Yet, as Wartena (2000) points out, it cannot be assumed for syntax that GEN is a rational relation on strings. He proposes to lift OS for syntax to the domain of trees and relations on trees, as OT syntax best is viewed as a theory about trees. He proves the result of Frank and Satta (1998) to hold when extended from regular string languages to regular tree languages. Morawietz and Cornell (1997) have also pointed out that tree automata[2] are a promising formal tool to model syntactic

[2] As this is not of direct concern for our topic, the reader is referred to Wartena (2000) and the literature noted there for formal definitions of trees, tree languages and tree automata.

operations and constraints. Provided that the constraints can be – and according to Morawietz and Cornell (1997) best are – expressed in "weak monadic second order logic" (MSO)[3], it is assured that the domain of finite techniques need not be left for formalization. Building on this work, Jäger (2001) is able to show that for bidirectional Optimality Theory in the sense of Blutner (2000; see section 3.3), the relation of bidirectional optimality between input and output can be modeled by a finite-state transducer, provided that the generator and the constraints can also be modeled by these means.

For the conclusion of this section, let me leave aside such automata-theoretic considerations again and return to linguistic properties. In recent years, a branch of Optimality Theory has been introduced that could prove of promising value for computer linguists as well. *Stochastic Optimality Theory* is a set of specific assumptions about the mechanism of interaction between the universal constraints. The choice of optimal candidates is determined by the rankings of constraint weights, but the rankings may be perturbed by normally distributed "evaluation noise" (Maslova 2004). This model of OT syntax still is in a very hypothetical stage, but a modification of OT in this direction could provide further formal means for an actual implementation of Optimality Systems. From the linguistic viewpoint, such an approach might be of great interest regarding the problems of ambiguity and markedness. Whereas the treatment of markedness is rather clear given the weighting of constraints (and therefore violations of

---

My only interest here was to show that syntactic OT may positively be formalized using finite-state techniques, tree automata at this point obviously being necessary.

[3] A weak second order theory is one in which the set variables are allowed to range only over finite sets (cf. Morawietz and Cornell 1997).

constraints), the relevancy of weighted constraints for treating ambiguity may need some explanation. In Stochastic OT it is assumed that (violations of) constraints may be added up, so the effect of a higher-ranked constraint might be equaled or cancelled, allowing lower-ranked constraints to become important even if overruled by a higher-ranked one (deviating from the notions of traditional OT).

4.3 Automatic Language Learning

One of the greatest strengths of Optimality Theory is to express language variation solely through differences in the constraint rankings. Therefore, one of the main goals of computational OT is the automatic re-ranking of constraints, so applications can be used for analyzing multiple languages. Several algorithms performing this task have been proposed, mostly for the field of phonology, but they can be straightforwardly applied to syntax, too, since the working principle remains the same. Among these proposals are the *Constraint Demotion* algorithm (Tesar and Smolensky 1993, 1996; Tesar 1995), the use of *Genetic Algorithms* (Pulleyblank and Turkel 1996), and the *Gradual Learning* algorithm (Boersma and Hayes 2001). For this section I will concentrate on remarks on *Constraint Demotion* and *Gradual Learning*, as they appear to have had bigger influence on recent research.

4.3.1 Constraint Demotion

It is commonly agreed upon in literature on language learning (e.g., Pinker 1997, Guasti 2004) that the learner needs lexical knowledge to learn a grammar, and grammatical knowledge to learn the lexicon. This suggests that a learner maintains a hypothetical lexicon and a hypothetical grammar during learning. For grammar learning, he then is repeatedly evaluating possible grammars with respect to this hypothetical lexicon and iterates in this fashion until he reaches a stable configuration of the grammar. According to of Tesar and Smolensky (1993, 1996) and Tesar (1995) this suggests that a learner has a yet unranked set of constraints as starting point, before starting with the learning procedure outlined above. We will leave the question of lexicon acquisition aside here.

In Optimality Theory, constraint interaction is pervasive to all levels of linguistic analysis: It is the primary explanatory device and thus entirely unavoidable. Therefore, constraint interaction needs to be faced also in language learning. This entails two major problems: The first one is "that of trying to learn anything from positive data in a framework that employs violable constraints. Given a grammatical description, you may observe that it violates some of the universal constraints. But if grammatical descriptions are allowed to violate constraints, how can anything be learned from those observations?" (Tesar 1995, 74).

The second problem is a complexity issue: The number of total distinct rankings is factorial to the number of constraints, i.e., a set of 10 different constraints has 10! = 3'628'000 distinct rankings. The question here is what amount of data is required to learn the correct ranking: If the data scales along with the number of possible rankings, a grammar with many constraints could require an untractably large amount of data to be learned successfully.

The initial data for the learning procedure are pairs consisting of an input and its optimal description. Since GEN as well as the constraint are assumed to be part of the Universal Grammar, the learner is able to generate multiple candidate descriptions for that input, and is also able to determine which constraint violations are assessed. Thus, each single piece of positive initial data conveys some negative comparative data, rendering every sub-optimal description less harmonic than its optimal counterpart:

(4.3) [sub-optimal description of input *I*] $\prec$ [optimal description of input *I*]

It is such pairs that are presented to the learning algorithm. Each pair carries information about the constraint ranking in the sense that is it known to the learner that the constraint(s) violated by the sub-optimal description must outrank those violated by the optimal description. In order for this to happen, the violations incurred by the sub-optimal candidate must be collectively worse than those of the optimal analyses. According to the theory, both candidates may violate the same constraints (possibly to a different degree), so common violation marks are not decisive. Therefore, the first thing the algorithm does is to cancel out the common marks, rendering the constraint profiles completely distinct. The pairs after cancellation are the data with which the actual *Constraint Demotion* algorithm operates.

The algorithm proceeds recursively, finding first the constraints that may be highest in the ranking according to the data, then eliminating those constraints from consideration and starting over with the lower ranked constraints. Let us, e.g., commence with a constraint set {*A, B, C, D, E*}. After the first iteration, we may get a partial ranking of the form {*A, B*} ≫ {*C, D, E*} (because our data pair only allows for the distinction of these constraint groups). Tesar (1995) calls these partial ranking *stratified hierarchies*. Then the next data pair is taken and this procedure is repeated until a definite ranking (say, {*B* ≫ *A* ≫ *C* ≫ *E* ≫ *D*}) is reached.

The core *Constraint Demotion* algorithm is given in pseudo-code below (Tesar 1995, 83–84):

(4.4) Core *Constraint Demotion* Algorithm

```
Given:     $\mathcal{H}$ = unranked (or stratified) hierarchy of constraints
           Data = a table of pairs of mark lists (constraint
           violations incurred by an optimal (winner-marks) and a
           sub-optimal candidate (loser-marks)),
To Find:   A stratified hierarchy with respect to which each of the
           loser-marks is less harmonic than the corresponding
           marks of the winner-marks.
Algorithm:
I. Mark cancellation:
           For each pair (loser-marks, winner-marks) in Data:
           a. For each occurrence of a mark *C in both loser-marks
              and winner-marks in the same pair, remove that
              occurrence of *C from both.
           b. If, as a result, no winner-marks remain, remove the
              pair from Data.
II. Constraint Demotion
Repeat the following until no more demotions occur:
           For each pair (loser-marks, winner-marks) in Data:
           a. find the mark $*C_{loser}$ in loser-marks which is highest-
              ranked in $\mathcal{H}$.
```

```
b. for each *$C_{winner}$ in winner-marks,
   if *$C_{loser}$ does not dominate *$C_{winner}$ in $\mathcal{H}$
      demote $C_{winner}$ to the stratum immediately below
      $C_{loser}$ (creating such a stratum if necessary)
c. output $\mathcal{H}$
```

Tesar and Smolenky (1998: 257–265) provide proofs that the *Constraint Demotion* algorithm always learns the correct grammar is given suitable training data. This means that OT grammars with totally ordered constraint hierarchies are learnable. Yet the algorithm shares two crucial problems with the standard linguistic theory: Like OT, it has problems representing free variation and ambiguity, as well as markedness phenomena. This is because the constraint hierarchies learned necessarily are totally ordered hierarchies. In addition, because of its formal properties, it cannot deal with erroneous data in the training set (Keller and Asudeh 2002, 2): If there is noisy data in the training set, this can mean the algorithm may not be able to learn the correct grammar.

4.2. The *Gradual Learning* Algorithm

In order to deal with the deficiencies of standard OT and the respective learning algorithm, Boersma and Hayes (2001) propose a probabilistic version of OT along with an associated new learning algorithm, the *Gradual Learning* algorithm (GLA). It is claimed that the GLA can model free variation and ambiguity and that it is able to learn from data containing errors.

The probabilistic OT model stipulates a continuous scale of constraint *strictness* (Keller and Asudeh 2002, 5): The constraints are annotated with numerical strictness values. “If a constraint C_1 has a higher strictness value than a constraint C_2, then C_1 outranks C_2. Boersma and Hayes (2001) assume probabilistic constraint evaluation, which means that at evaluation time, a small amount of random noise is added to the strictness value of a constraint. As a consequence, re-rankings of constraints are possible if the amount of noise added […] exceeds the distance between the constraints on the strictness scale.“ (Keller and Asudeh 2002, 5). The associated GLA basically is a generalization of *Constraint Demotion*: both assume a set of parsed data as training set, and both feature constraint demotion (and promotion). The algorithm starts with a grammar *G* containing arbitrarily ranked constraints, i.e. all constraints have a random strictness value. If a training example *S* is encountered, the GLA computes a corresponding structure *S′* generated by the current grammar *G*. If *S* and *S′* are not identical, learning takes place; the constraint hierarchy is adjusted in the way that it makes *S* optimal instead of *S′*. First, again all common violation marks get cancelled; on the remaining marks, the following is performed: On the constraints violated by *S* (but not *S′*), the strictness value is decreased, and accordingly on the constraints violated only by *S′*, the strictness value is increased. This gradually adjusts the strictness values of all constraints, resulting in the correct hierarchy. In contrast to *Constraint Demotion*, the GLA adjusts constraints gradually, as it only triggers small changes in the strictness values. One training example thus is not sufficient, as possibly not enough has been added or subtracted from the strictness values to reach the final ranking. Therefore the algorithm is robust: incorrect examples only disturb the learning process by a small amount that will be corrected in a later pass. A simplified formalization is given below:

(4.5) *Gradual Learning* Algorithm

```
Given:     $\mathcal{H}$ = A hierarchy of constraints with random strictness
           values.
           Data = A set of fully specified input-output pairs $\langle i,o \rangle$
Algorithm:
I. Generation:
           Generate an output $o'$ for the input $i$ given $\mathcal{H}$.
II. Mark cancellation:
           For each pair $(o, o')$:
           a. If $o = o'$, nothing happens.
           b. For each occurrence of a mark *C in both $o$ and $o'$,
           remove that occurrence of *C from both.
III. Constraint adjustment
           a. Increase the value of all constraints favoring $\langle i,o \rangle$ over
           $\langle i,o' \rangle$ by N.
           b. Decrease the value of all constraints favoring $\langle i,o' \rangle$
           over $\langle i,o \rangle$ by N.
           (where N is some random numerical value)
Repeat steps II. and III. until the constraint values stabilize, then
output $\mathcal{H}$.
```

Despite of its advantages the GLA also has some severe deficiencies, as Keller and Asudeh (2002, 3) point out:

- It has not been tested on unseen data, hence it is unclear whether it is able to generalize
- There are data which cannot be learned
- No proof of correctness is offered

However, in recent literature several modifications of the GLA have been proposed (partly) surpassing these deficiencies, among which the approach of Jäger (2003a) seems very promising for the field of syntax. Jäger takes into account both the production and interpretation direction of optimization and incorporates them into the GLA, hence arriving with a modified version in the sense of bidirectional OT (Blutner 2000), the "Bidirectional GLA" or "BiGLA". Jäger offers formal proofs of his algorithm, showing that the GLA is correct and able to generalize. Yet it still inherits some of the problems of its mother algorithm in the sense that there is still data that probably cannot be learned. Secondly, Jäger (2003b) offers a further adaptation of the GLA by combining it with Maximum Entropy Models (ME). Building upon research by Goldwater and Johnson (2003), who show that a ME approach to probabilistic OT learning for phonology is equivalently well motivated as the GLA and yields the same empirical results too, Jäger is able to show that a combined ME-GLA approach to learning does not suffer from the problems of Keller and Asudeh's (2002) criticism. Whereas the learnability problems of probabilistic OT in the sense of Boersma and Hayes (2001) are still open, there exist provably correct learning algorithms for ME models. On the other hand, the GLA offers a better model of language acquisition than ME algorithms do (cf. Jäger 2003b, 7ff.). A combination would maintain these advantages as well.

4.4 Overview of Available Applications

Before concluding this section, I would like to mention that a range of applications for Optimality Theory is accessible on the World Wide Web, with regard to which I would like to give a short overview.

Raymond and Hogan (1994) introduce the "Optimality Interpreter", a tool that allows the specification of constraints and candidates as well as the valuation of candidates regarding the constraints. Its basic intention lies in giving assistance to researchers in their designs of OT solutions.

Walther (1996) provides a construction-kit to OT implementations programmed in Prolog and making use of standard Unix commands, called "OT SIMPLE". GEN is implemented as a Prolog program interpreting a context-free structural grammar according to a use input. Constraints are specified as finite-state transducers implemented as UNIX 'sed' stream editor scripts mapping ill-formed structures to violation marks. Finally, evaluation reduces to simple UNIX 'sort', which orders candidates on the basis of their violation marks.

With regard to phonology, the most prominent applications are the implementations of Hammond's (1995,1997) Perl and Prolog syllable parser, respectivly, and Heiberg's (1999) JAVA OT phonology system discussed in section 4.1.

As for syntax, most of Kuhn's (1999 – 2003) work has been included in the Xerox XLE system.

Finally, Jäger (2003c) proposes an implementation of his BiGLA algorithm discussed above in a software tool for simulation language evolution, called "evolOT".

4.5 Summary

This chapter has provided an overview of computational accounts of Optimality Theory. Most work in this field has so far been conducted for phonology, where several implementations of OT have been proposed. In section 4.1, the most prominent approaches have been discussed, as well as investigations regarding mathematical properties of OT systems, where it was shown that – given specific restrictions – all parts of an OT system can be implemented using well-known automata techniques. Section 4.2 has pointed out the problems of implementing OT syntax by discussing its formal properties. It was demonstrated how these problems can be faced and which further restrictions and formalizations are necessary for this task. Section 4.3 was dedicated to language learning. Two algorithms were presented for the task of automatic constraint re-ranking, *Constraint Demotion*, set in standard OT, and *Gradual Learning* (GLA), set in a probabilistic OT framework. Both start out with a set of unranked constraints and pairs of inputs and optimal output representations. Then they generate another (pseudo-)optimal output based on the current constraint ranking, whose violation marks are compared with those of the optimal candidate. In a final step, the constraints are rearranged in order for the optimal candidate to win the competition. The GLA overcomes some of the deficiencies of *Constraint Demotion* by making use of the probabilistic features coming along but in turn has its own problems. However, these were shown solvable by extending the algorithm to bidirectional OT and implementing Maximum Entropy models into it. Finally, a short overview of existing OT software was given, ranging from construction-kit approaches over phonology and syntax to language learning.

5. Computational OT Syntax and the Xerox XLE System

After presenting the mathematical bases of computing Optimality Theory in the preceding chapter, I will now proceed to the more linguistic formalization of OT syntax. Whereas the work cited above has achieved thorough *theoretical* knowledge, so far, no *practical* work has been done implementing these findings from a strictly computational perspective. Partly due to the still ample obscurity and even partial inconsistencies of the linguistic background it is more promising at this stage to set out from a strictly linguistic perspective, trying to obtain a clearer idea of the linguistic principles involved first. The pioneer work here was done mainly by Kuhn (e.g., 1999, 2002, 2003). In the following, I will discuss the basics of his work before proceeding to a presentation of the Optimality-theoretical functionality in the *Xerox Language Environment XLE*[1], which is based partly upon the work of Kuhn.

5.1 The Formalization of OT Syntax

One of the questions we asked at the beginning of section 4.2 was what restrictions can be made so the processing task becomes decidable. Kuhn (2003) proposes a formalization of syntax in OT-LFG, a subset of Optimality theory based on Lexical-Functional Grammar (cf. Bresnan 1996). Kuhn sees various reasons for adopting an LFG-based approach to OT syntax, including the fact that "a lot of typological research has been conducted in the LFG framework. For the present purposes, the main advantage of picking LFG as the base formalism is however

[1] www2.parc.com/istl/groups/nltt/xle/.

that its formal and computational properties have undergone thorough research and that there are highly developed systems for processing the formalism." (Kuhn 2003, 55). In the next section, I will give an overview of Kuhn's abstract formal specifications of an OT system, abstaining from an introduction to LFG (instead, I refer the reader to Bresnan (1996, 2000)).

5.1.1 Overall Formalization

- **Candidates** (Kuhn 2003, 63f.): The candidate analyses OT-LFG deals with are tuples of structures known from LFG, i.e. c-structure and f-structure pairs in a correspondence relation. All candidates satisfy certain inviolable principles (i.e. inviolable constraints already part of GEN), which are assumed to be encoded in an LFG grammar G_{inviol}. Thus the set of possible analyses is defined as the language $L(G_{inviol})$ generated by the formal LFG-style grammar G_{inviol}.
- **Input** (Kuhn 2003, 64f.): When candidates are assumed to be fully specified LFG analyses, it suggests itself that the input in the OT sense is a partially specified representation, i.e. a feature representation of the semantic content of an utterance and potentially some further information such as information structure, etc. More generally, the input is defined as being a member of the set of well-formed (partial) f-structures $\mathcal{F}$. It is a property of this definition that both the input and the candidates are thought of as static, contrary to most OT syntactic work where candidates are assumed to be derivations.

- **GEN** (Kuhn 2003, 65): Given the above, GEN can be defined as a function that takes each input f-structure Φ_{in} to a set of candidate analyses, which are contained in G_{inviol}. Specifically, $Gen_{G_{inviol}}$ is a function from the set of f-structures to the power set of candidate analyses (c-structure/f-structure pairs $\langle T, \Phi\rangle$) in $L(G_{inviol})$: $Gen_{G_{inviol}}: \mathcal{F} \rightarrow \wp(L(G_{inviol}))$. Mathematically we express this in the following way:

  (5.1) $Gen_{G_{inviol}}(\Phi_{in}) \subseteq \{\langle T, \Phi'\rangle \mid \langle T, \Phi'\rangle \in G_{inviol}\}$

  Of course, this is just an overall formalization and further restrictions will be needed. One possible restriction certainly is to limit generated candidates by not adding semantic information (where *semantic information* is in need of an exact definition). Then only those of the possible candidate analyses are picked whose f-structure is subsumed by the input f-structure:

  (5.2) $Gen_{G_{inviol}}(\Phi_{in}) \subseteq \{\langle T, \Phi'\rangle \mid \Phi_{in} \sqsubseteq \Phi'\rangle$ where Φ' contains no more semantic information than $\Phi_{in}\}$

- **Constraint marking** (Kuhn 2003, 65f.): Kuhn chooses to represent the number of violations a candidate incurs as a natural number. The constraints $\mathcal{C}$ are given as the sequence $\langle C^1, C^2, \ldots, C^k\rangle$ so the violation counts can be represented as a sequence of natural numbers $\langle n^1, n^2, \ldots, n^k\rangle$, where $n^i \in \mathbb{N}_0$. Thus, the function *marks* is defined as follows:

(5.3) Definition: *marks*

marks is a function from (input) f-structures and LFG analyses to a sequence of k natural numbers (where $k = |C|$, the size of the constraint set).

$$marks\text{: } \mathrm{F} \times L(G_{inviol}) \rightarrow \mathbb{N}_0^k\text{, such that}$$

$marks(\Phi_{in}, \langle T, \Phi' \rangle) =$
$\langle C^1(\Phi_{in}, \langle T, \Phi' \rangle), C^2(\Phi_{in}, \langle T, \Phi' \rangle), \dots , C^k(\Phi_{in}, \langle T, \Phi' \rangle) \rangle$

The exact formalization of OT constraints will be discussed in section 5.1.2.

- **EVAL** (Kuhn 2003, 66f.): As the key concept of optimimization, Evaluation, depends on the violation counts of the candidates (and, of course, the constraint ranking of the language), Eval can again be envisaged as a function that determines the most harmonic (optimal) candidate.

(5.4) Definition: *eval*

Given a set of LFG analyses Γ,

$$eval(\Gamma) \rightarrow \mathbb{N}_0^k\text{, such that}$$

$eval_{(C, >>_{\mathcal{L}})}(\Gamma) =$

$\{ \langle T_j, \Phi_j \rangle \in \Gamma \mid \langle T_j, \Phi_j \rangle$ is maximally harmonic for all analyses in Γ, under $\gg_{\mathcal{L}} \}$
(where $\gg_{\mathcal{L}}$ is the language-specific ranking of the constraint set C)

Note at this point that "pseudo-algorithms" formalizing these components of an OT system were already given in section 3.1.

- **Language generated by an OT(-LFG) system** (Kuhn 2003, 67f.): Recapitulating, we can now finally define an OT(-LFG) system and the language generated by such a system:

(5.5) Definition: *OT(-LFG) system*

An OT(-LFG) system *O* is defined as $\langle G_{inviol},\langle C, \gg_{\mathcal{L}}\rangle\rangle$, where

G_{inviol} is a formal LFG-style grammar definig possible candidate analyses,

C is a sequence of OT constraints, and

$\gg_{\mathcal{L}}$ is a ranking over *C*.

Subsequently, the language generated by an OT(-LFG) system $L(O)$ can be defined as the set of analyses $\langle T_j, \Phi_j\rangle$ for which there exists an input f-structure Φ_{in} such that $\langle T_j, \Phi_j\rangle$ is among the most harmonic candidates for that input:

(5.6) $L(O) = \{\langle T_j, \Phi_j\rangle \in G_{inviol} \mid \exists\Phi_{in} : \langle T_j, \Phi_j\rangle \in eval_{(G>>_{\mathcal{L}})}\ (Gen_{G_{inviol}}(\Phi_{in}))\}$

5.1.2 The Formalization of Constraints

Whereas formalizations of the GEN and EVAL components can be envisaged as rather straightforward, the formalization of the inviolable constraints is the most difficult task in the specification of an OT system. According to the above general formalization, constraint marking basically is a function mapping an analysis to a natural number. More precisely, constraints are representations of conditions over (in our case) syntactic structures. The fundamental question here is: How should the constraints be formulated (in natural language), so that a precise logical formalization becomes possible?

Constraints typically are not formulated as conditions on particular structures but usually take the shape of universally quantified implications: Whenever a constraint applies to a structure *A*, it should also apply to an equivalent structure *B*. Thus a description language is needed that allows us to express universal quantification over structural objects, and that contains negation for expressing implication. Kuhn (2003, 85ff.9) shows that simple logical universal quantification does not suffice for this purpose. The use of general logical formulae of this kind is inappropriate when multiple violations have to be marked. The problem here lies in the choice how the universally quantified constraints are formalized (i.e. $\forall x \exists y$ vs. $\exists x \forall y$). Both options are logically equivalent, yet they differ in that they possibly ascribe different numbers of violations to phrase structures (cf. Kuhn 2003, 87–88). Instead, he finds that expressing the constraints using meta-variables gives us exactly what we need: When the constraints are checked, the variable gets instantiated to one structural element after another (i.e. every constraint is checked at every node). Accordingly, what we get are not the actual constraints but constraint schemata generating classical constraints. Under this assumption the logical equivalence of the options $\forall x \exists y$ vs. $\exists x \forall y$ applies also for the number of violations. Note that, as Kuhn (90) points out, it is also compatible with this assumption to adopt a "scalar" view of constraints, i.e. an alignment constraint like HEADLEFT ("The head is leftmost in its projection"; Kuhn 2003, 90) is violated twice if two elements intervene between the left periphery and the head. The metavariable allows a clear distinction of the scalar and non-scalar (i.e., in contrast to the example above only one violation is marked) formulation of a constraint.

Let us now concludingly look at a sample formalization of a constraint by taking Kuhn's (2003, 91) representation of Bresnan's OBLIGATORY-HEAD constraint:

(5.7) OBLIGATORY-HEAD (OB-HD)

"Every projected category has a lexically filled (Kuhn: extended) head." (Bresnan 2000, 21)

$$(cat(*) \wedge (bar\text{-}level(1) \vee (bar\text{-}level(2))) \rightarrow \exists n.[ext\text{-}hd(n, *)]$$

*If * is an X-bar or X-max category, then there is some node n which is the extended head of *.*

"*" here denotes the meta-variable mentioned above, and *X-max* is a maximal projection of head X (whereas *X-bar* means an intermediate node). During constraint checking, "*" gets instantiated to every node and checks whether there is a head *X* that qualifies as the head of the projection *X-bar* or *X-max*. Similarly, every constraint in the constraint set is applied to each structural element of the candidate, with the meta-variable instantiated to the respective element. When the application of a constraint fails, the count of violations of the candidate gets increased for the respective constraint.

5.2 Optimality-theoretical Functionality in the Xerox Linguistic Environment

The Xerox Linguistic Environment (XLE)[2] is the first large-scale commercial linguistic application to explore Optimality Theory-style functionality for certain fields of application. XLE is not, however, a fully Optimality-based application, nor does it feature a full-scale OT implementation. Rather, the OT functionality in XLE works as an independent add-on that acts upon the LFG grammar of XLE; it is not part of the LFG grammar itself, thus being able to work with any other grammatical framework as well. In theory the OT add-on could therefore be

[2] For further information about XLE see www2.parc.com/istl/groups/nltt/xle.

applied to any existing linguistic system without many changes necessary. This section provides a look at this OT system within XLE and its historical evolution.

Kuhn and Rohrer (1997) explore the use of marked constraints for dealing with syntactic ambiguities. The main objective of their work is not to provide a solution to the ambiguity problem itself – rather, they shift the attention to facilitating the groundwork, i.e. the engineering of large-scale grammars. One of the problems in grammar engineering is the possible commitment to linguistic decisions at an early stage of development. This problem should be avoided, as it may influence an unknown number of other decisions later on. A straightforward way to bypass such commitments is to put off filtering of readings, i.e. to deal with the full number of possible analyses for any sentences during development. This is impractical, for real and syntactic ambiguity may result in a great number of readings, which poses an efficiency problem. In this respect, it is desirable to be able to "switch on and off" the effect of ambiguities irrelevant to the analysis in order to be able to minimize run time of test. What is required for this is "a filtering mechanism that can be easily re-organized, enforced, or switched off altogether".[3] Also, ideally, the linguist should be able to fully understand and control such a system without knowledge of programming techniques. Kuhn and Rohrer propose the use of constraint ranking similar to OT rankings for such a filtering scheme, as the constraint ranking combines most of the advantages of weighting approaches to disambiguation with easy controllability for both grammar writers and linguists. The constraint ranking used in their approach is not a direct implementation of the OT ranking mechanism, but rather an extension

[3] Kuhn and Rohrer (1997, 2).

of it. “In order to be able to rank LFG constraints, Optimality marks have to be introduced in the grammar. This is done by constraints that refer to an *o*-projection”[4] (optimality projection) which is added to the standard *c*- and *f*-structure scheme of LFG. The *o*-structure is viewed as “being a flat multiset of optimality marks percolated from all daughter constituents”.[5] The ranking of the constraints is specified in the header, so the ranking can be changed or constraints can be turned on and off easily by the user.

The standard OT ranking of (dispreference) constraints is extended to a scheme that allows both dispreference and preference constraints. (5.8) shows a sample ranking configuration:

(5.8) Sample ranking configuration

```
OPTIMALITYRANKING   MARK1 MARK2 NEUTRAL MARK3
                    MARK4 NOGOOD MARK5
```

Above, MARK1 is more preferred than MARK2; all unmarked constraints are treated as being NEUTRAL[6]; MARK3 is disprefered, MARK4 even more; MARK5 finally may not show up in any successful analysis since standing to the right of NOGOOD (sc. every reading containing MARK5 is filtered right away).

In an analysis, first of all the most disprefered mark (MARK4) is checked, the analyses with the *smallest* number of marks survive. If more than one remains, the next dispreference mark (3) is checked. In case no unique candidate is found, the preference marks are considered, starting from the most preferred one (1): here,

[4] Kuhn and Rohrer (1997, 2).
[5] Kuhn and Rohrer (1997, 2).

[6] Marks that are introduced in the grammar but don’t appear in the ranking are treated as neutral.

the candidate with the *greatest* number of marks wins. If still no analysis is favoured, the second preference mark (2) is taken into account. The preference marks and their special treatment distinguish this specific XLE-version of OT from standard approaches.

A small example (taken from Kuhn and Rohrer (1997, 3f.) shows how approaching ambiguity with the above-described means could look. The example construction takes care of the subject-object ambiguity found quite often in German:

(5.9) Anna sucht Otto.
Anna seeks Otto.

There is no overt morphological case marking, resulting in an ambiguity of the respective function of the NPs. This may be filtered by stating a preference for subject NPs to be located in the "Vorfeld" (i.e. before the finite verb). Assuming this mark to be ranked as a preference mark, the system will only output the reading in which *Anna* is the subject of the verb.

Even if this is a simple and straightforward approach to syntactic ambiguity, the merits of the reported scheme lie rather in the facilitating of engineering and, in particular, internal evaluation. One major problem in grammar writing is the complex interaction of parts of the grammar in large-scale systems. Problems concerning the interaction, such as finding the exact source of the problem and, even worse, providing a remedy against it, quickly get very difficult. The use of OT marks in a system as the one outlined above support easy testing of variants of a grammar. The marks are used to remove certain parts of the grammar beyond the NOGOOD limit. (A similar effect can be reached be commenting out certain rules of the grammar; however, this brings some disadvantages like errors in commenting out and back in, keeping track of all locations that one may need to vary in testing, or re-trying old tests after some time and changes.) Since the ranking is located in the header of the grammar, systematic (and error-proof) variation can be achieved with little effort – even automatic changing of the ranking using scripts and running tests over slightly different versions of a grammar seems possible in a straightforward fashion. Tests show (see Kuhn and Rohrer 1997, 5ff.) that making use of the outlined system can dramatically decrease the run and test time of a parser.

Another benefit of the addition of OT to the XLE-LFG system is the possible interaction of syntactic constraints with other types of information, such as selectional restrictions on the semantic classes of verb arguments. A further advantage is that the robustness of a grammar can be increased by adding low-ranked "fallback rules" that allow for the parsing of grammatical mistakes (e.g. subject-verb agreement faults or adverb misplacement). This is invaluable in real-world applications where the material is less ideal than in average linguistic examples. And finally, the OT-oriented ranking mechanism facilitates using the same grammar for parsing and generation: while a grammar should be able to accept a wide range of alternative syntactic structures (including even faulty ones), generation should be restricted to a subset of "preferred", i.e. correct, constructions. For the last two applications, an UNGRAMMATICAL mark type is provided, whose marks allow the parser to construe a sentence that can be skipped in generation, where no ungrammatical output is wanted.

Fortmann and Forst (2004) suggest an external usage of these evaluation facilities of XLE-OT in the field of computationally assisted (second) language learning. They make use of the above mentioned methods of grammar checking, like lowly ranked fallback rules, to develop a system that "not only states whether or not the learner's input is grammatical, but also provides crucial information necessary for generating helpful feedback as to what has to be corrected" (Fortmann and Forst 2004, 1). The system covers the most frequent error types of L2 learners of German, including agreement and sentential word order. The grammar must thus be able to recognize ungrammatical (or marked) input (via the fallback rules) and analyze the errors as to give suitable response to the learner (it needs to generate the correct sentence and show the differences). This resembles a possible use of the OT-scrambling approach implemented in chapter 7, where from the OT marks accumulated by a candidate its markedness can be read.

After characterizing the basic functioning, we shall have a more technical look at XLE's OT system in the following.

5.2.1 Mark Types

XLE features a set of special mark types that serve the classification of marks for special purposes besides the two standard mark types, i.e. *preference* and *dispreference marks*, as noted above.

- *Preference marks* come to use when one specific reading out of a set of analyses is preferred. A nice example (from Frank et al. 1998, 5) is the use of such marks to state a preference for multiword terms in technical documentation. With a respective preference mark, the analysis with the multiword expression will always be preferred over all other readings (see 5.10).

(5.10) a) I want [print quality] images.
b) *I want [print] [quality] images.

However, if there is no valid analysis for the multiword expression (as in 5.10b), an analysis using the individual lexicon entries still is possible.

- *Dispreference marks* are set below the (hypothetical) NEUTRAL mark and are generally used for rare constructions that are grammatical and as such parsable, but are unlikely to occur. A dispreference mark ensures that the construction may surface, but only when no other analysis is possible. For example, this can be used to exclude NPs being headed by adjectives from the candidate set. A valid sentence incorporating such a construction is (5.11; Frank et al. 1998, 6):

(5.11) Meistens kauft die grössere Firma die kleinere.
Mostly buys the larger company the smaller.

However, this requires a rule, which allows NPs to consist of an adjective and an optional determiner, that permits the grammar to build such NPs in many implausible places, like in (5.12; Frank et al. 1998, 6):

(5.12) Nachts fallen [$_{\text{NP(nom)}}$ helle] [$_{\text{NP(dat)}}$ Farben] auf.
'At night, bright ones strike colors'.

Of course, the desired reading would be that [$_{\text{NP(nom)}}$ helle Farben] form a single NP (leading to the reading: 'At night, bright colors stand out'), yet the reading from 5.12 is not impossible. Here, a dispreference mark introduced to the grammar brings remedy by constraining the infelicitous reading above.

Despite the pretty clear examples given above, it can be difficult to decide whether to use a preference or dispreference mark in general. There are two main issues at stake: the interaction between the marks, and which analysis is easier to mark (in the grammar, this is). For instance, it is easier to mark a multiword expression with a preference than to mark all of its components with a dispreference.

- *UNGRAMMATICAL marks* have already been introduced above. They are used to increase the robustness of a parsing grammar by allowing for marking error rules that parse ungrammatical constructions. For generation tasks, these UNGRAMMATICAL marks are skipped to prevent erroneous output from surfacing. The difference between dispreference and UNGRAMMATICAL marks is that they only come to action when no grammatical solution at all, whether preferred or disprefered, was found.
- *NOGOOD marks* indicate that the respective analysis is always bad, even if there is no other, valid one. The purpose of such marks is to allow for fine-grained filtering of a grammar. A possible application would be the parsing of section headers (e.g., sentence numbering tags in the *Verbmobil* corpus) that may occur in tagged corpora. It would be desirable to be able to parse such constructions, but the rule clearly is undesirable for other uses of the grammar. If these marks are listed after NOGOOD, then constructions involving the mark are being treated as inconsistent. As such, the rule or lexical item will still appear in the output, but no further structure will be built on top of it. For

further (e.g. statistical) use, nogoods can be converted into INCONSISTENT, INCOHERENT, or INCOMPLETE marks, if the respective mark type is present. However, using these marks can produce enormous numbers of unoptimal solutions present in the output.

- *STOPPOINT* marks can be used for dividing the analysis into multiple stages. If the ranking has one or more of these special marks in it, XLE will process the input in multiple passes, using larger and larger portions of the grammar. Stoppoints are treated from right to left, meaning that any suboptimal solutions involving marks to the left of the stoppoint are not tried. For instance, if the Optimality ranking was as in (5.13):

```
(5.13) OPTIMALITYORDER    … Mark3 STOPPOINT Mark2
                          STOPPOINT Mark1.
```

Then XLE would first try analyses with either no mark at all or only the Mark1 mark. Only if there were no valid solutions found, then it would try analyses including the Mark2 mark, and so on.

This concludes the presentation of the most important general special mark types. The following part deals with the formal implementation of local marks.

5.2.2 Implementation

Optimality marks in an XLE-LFG grammar are constraints added to the appropriate rule of the grammar (5.13):

```
(5.13) a) … Mark1 $ o::* …
       b) … Mark2 $ o::M*
```

5.13a) says that Mark1 is a member of the Optimality projection *o.* In b), the mark is not added to the local node but to its mother M.

(5.14) is an illustrative example of a standard LFG rule including an Optimality mark. The rule describes an elliptical NP, where the missing noun is penalized by the HEADLESS mark, being a natural and clear example for a dispreference mark (based on Kuhn and Rohrer 1997, 4).

(5.14) $$\text{NP} \rightarrow \text{(Det) A} \left\{ \begin{array}{c} \text{N} \\ \text{e} \\ \text{HEADLESS} \in o^* \end{array} \right\}$$

The existence of headless NPs makes it necessary to have such a rule. However, this leads to unwanted ambiguity in sentences like (5.15):

(5.15) Hier fehlen gemütliche Kneipen.
'There are no cozy pubs around here.'

The verb *fehlen* may take an optional dative object and *Kneipen* is morphologically ambiguous between nominative and dative, the sentence gets an additional (if implausible) reading with *gemütliche* as Subject and *Kneipen* as dative object. This case can be excluded by marking it according to the above rule.

In more detail, an (XLE)-LFG rule with Optimality Marks looks as follows:

(5.16) XLE code example

```
NPheadless --> "headless noun phrases; COM{EX RULE NP: the dentist's}"
          "COM{EX RULE NP: mine}"
        e: @NSG
             { @(NULL-PRONOUN ^)
              |@(SUBJ_core pro) "COM{EX RULE ROOT: It is Mary's.}"
               @(PRON-TYPE-NSYN null) };
        { NPposs: (^ SPEC POSS)=! "noun phrase"
                     @(OT-MARK Headless) "only penalize lexical ones;
                     not pronominal"
                     ~(^ SPEC POSS PRON-TYPE)
            |PRONheadless "pronoun; COM{EX RULE NP: mine}"}.[7]
```

In this case, for NPs without a lexical head the Optimality mark HEADLESS (highlighted) is invoked, but not if the NP is a pronominal.

The following pictures illustrate the above: Figure 5.1 shows the tree structure for the sentence "It is the dentist's.", with the corresponding f-structure in Figure 5.2:

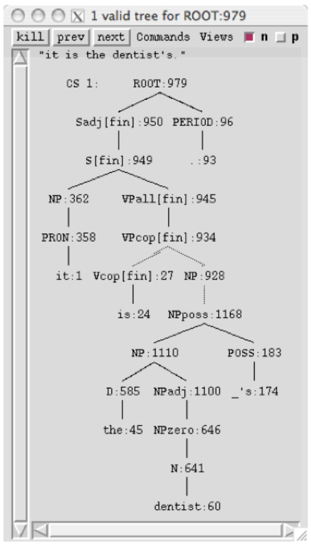
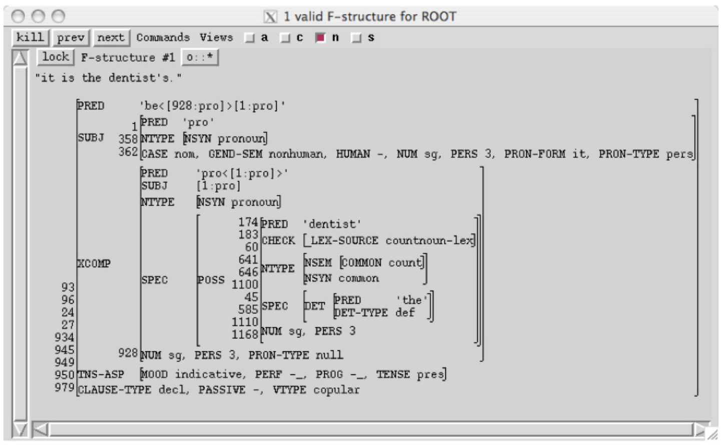


Fig. 5.1 XLE tree structure window



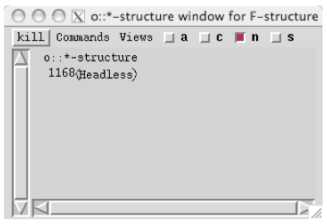
```
"it is the dentist's."

        PRED      'be<[928:pro]>[1:pro]'
                1 PRED   'pro'
        SUBJ  358 NTYPE [NSYN pronoun]
              362 CASE nom, GEND-SEM nonhuman, HUMAN -, NUM sg, PERS 3, PRON-FORM it, PRON-TYPE pers
                  PRED      'pro<[1:pro]>'
                  SUBJ      [1:pro]
                  NTYPE     [NSYN pronoun]
                                     174 PRED    'dentist'
                                     183 CHECK  [LEX-SOURCE countnoun-lex]
                                      60
        XCOMP                        641 NTYPE  [NSEM [COMMON count]
                                     646         NSYN common          ]
                  SPEC      POSS    1100
     93                               45 SPEC   [DET [PRED     'the']]
     96                              585             [DET-TYPE def  ]
     24                             1110
     27                             1168 NUM sg, PERS 3
    934
    945
    949       928 NUM sg, PERS 3, PRON-TYPE null
    950 TNS-ASP   [MOOD indicative, PERF -_, PROG -_, TENSE pres]
    979 CLAUSE-TYPE decl, PASSIVE -, VTYPE copular
```

Fig. 5.2 XLE f-structure window

[7] ^ and ! signify the standard LFG ↑ and ↓ signs in XLE code; e (epsilon) signifies an empty category; ~ a negation; | a disjunction; finally, the @ invokes a so-called *regular macro*, an entity that may expand to arbitrary regular predicates with no intuition that they correspond to singleton nodes.

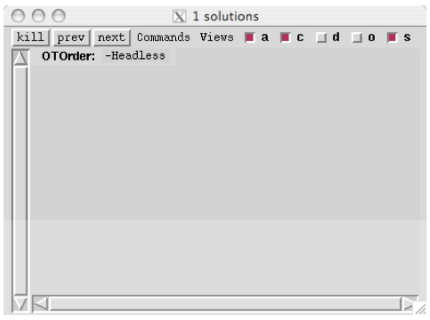
The numbers refer to nodes in the tree structure, facilitating the orientation in the various analysis windows.

Figure 5.3 now shows the *o*-structure (Optimality structure) window, which lists all marks of the current analysis. In this case, only the HEADLESS mark is present, because the object NP lacks a lexical head (e.g., "the dentist's *computer*"). Finally, Figure 5.4 shows the constraint ranking and marks also that the HEADLESS mark is a dispreference mark, resulting in an eventual back-placement of this reading, if other analyses were found.



```
o::*-structure
 1168⟨Headless⟩
```

Fig. 5.3 XLE *o*-structure window



```
OTOrder:  -Headless
```

Fig. 5.4 XLE solutions window showing the constraint ranking

5.3 Summary

This chapter has introduced a formalization of OT syntax by discussing the work of Kuhn (mostly 1999–2003), who sets up his OT syntax system in an LFG environment, allowing him to make use of the well-known computational properties of Lexical-Functional Grammar. The GEN and EVAL components of his systems can be imagined as rather straightforward functions; GEN as generating all possible outputs for a given input representation using a universal base grammar, and EVAL as a function first marking all candidates with respect to their constraint violations and then determining the optimal candidate. The most difficult task of formalizing OT syntax, namely the formalizations of the constraints, was discussed at length in section 5.1.2. Kuhn (2003) proposes a formalization in terms of logical formulae and using meta-variables for the structural positions where the constraints apply. At every node in the structure, every constraint is applied, incurring a violation mark where it is not satisfied.

The Xerox Linguistic Environment, despite of its establishment upon Kuhn's finding, makes a slightly different approach to OT, as has been presented in section 5.2. Marks are introduced as instances directly into the grammar. I.e., the marks are part of the actual rules, meaning that the mark is added to the Optimality (*o*-)structure, whenever that rule is called. The analyses are then compared with regard to their respective *o*-structure, and are arranged according to the constraint ranking specified in the header file of the grammar. The use of this Optimality-theoretical functionality of XLE is mainly intended for internal use, namely evaluation of the grammar. It allows for easy changing and testing of the grammar by, e.g., switching on and off parts of the grammar; also the run-time of the parser can be reduced by using this functionality to cut down the search space. A further advantage is the increase of the robustness of the grammar by allowing for "fallback" rules marking ungrammaticality but coming to use if no other, valid, analysis is found. Finally, the same grammar can be used for parsing and generation, as such "ungrammatical" rules can be excluded from generation via the constraint ranking. To this instance, XLE features a series of general, overall mark types which permit the partitioning of the constraint ranking. Also, marks can differ between preference and dispreference marks, where a reading with preference mark(s) is generally preferred over other readings, and dispreference marks lead to a dispreference of the actual reading, respectively.

On the other hand, of course, “external” usage of the OT functionality in XLE, like the treatment of ambiguity phenomena, is also conceivable. Yet at the moment, other, conventional approaches still are superior to this OT approach.

6. German Word Order Variation

This chapter presents an analysis of word order variation in German, which will be formalized in the following chapter. There exist several accounts of analyzing word order variation, and many of its properties have been unraveled. Research has accumulated substantial evidence for assuming a "scrambling operation"; however, at least three elementary problems arise for such a scrambling operation for traditional grammar approaches. First, a trigger to enforce scrambling is needed, as "economy of movement", a feature widely accepted in the Generative literature, would block unforced movement. It is not at all clear what such a trigger might be. Second, clause-internal word order variation in languages often exhibits degrees of markedness rather than complete illformedness or wellformedness. The issue of (un-)markedness is not accounted for in traditional approaches. Finally, languages show different reasons or options for changing word order; how can such language-specific variation be explained?

Optimality Theory has brought new possibilities into research to address these problems of scrambling approaches. The system of violable and ranked constraints provides a solution for the first and last of the above problems. The markedness issue remains a problem for pure OT; but proposals were made (cf. Keller 1996) that allow for the treatment of markedness under OT principles.

One of the fundamental strengths of Optimality Theory is its ability to allow for the interaction of different linguistic faculties, e.g. phonology and morphology, or phonology and syntax. For that reason it does not astonish that many analyses of word order variation take into account the interaction of phonology and syntax:

the trigger for clause-internal (forward) movement of constituents is steered by, e.g., accentuation for topicalised elements. This clearly is a striking account for spoken natural language analysis but does not present a valuable framework for computational approaches to syntactic phenomena.

A purely syntactic analysis of German word order variation is proposed in Müller (1998) and will thus be taken as a basis for the formalization. In the following I will present this approach.

6.1 Empirical Evidence for Scrambling and Markedness

We have already seen some examples for word order variation in section 3.3. Let us now have a closer look at such phenomena before dealing with Müllers analysis. Consider the following example (Müller 1998, 9):

(6.1) a)	dass	er	dem	Fritz	die Zeitung	überliess.
	that	he	DET	Fritz$_{dat}$	the newspaper$_{akk}$	left.
b)	dass	er	die Zeitung	dem	Fritz	überliess.
	that	he	the newspaper$_{akk}$	DET	Fritz$_{dat}$	left.

Both sentences are grammatical, though (6.1b) is judged to be more marked by many native speakers of German. A possible reason for this judgement could be the unmarked (D-structure) order of the arguments of the verb: in (6.1a), the indirect object precedes the direct (IO > DO), whereas in b) we find the opposite order. We can now assume IO > DO to be the basic order of arguments, and explain the DO > IO order through scrambling. However, consider (6.2) (Müller 1998, 8–9):

(6.2) a) dass man die Kinder diesem Einfluss entzogen hat.
that one the children$_{akk}$ this influence$_{dat}$ deprived has.

b) dass man diesem Einfluss die Kinder ausgesetzt hat.
that one this influence$_{dat}$ the children$_{akk}$ exposed has.

Here, the DO > IO order seems to be the base-generated one, contradicting example (6.1). Thus, basic argument order cannot serve as an explanation for deriving markedness phenomena. A second and similar explanation, case, can be rejected with the same arguments as DO/IO above: E.g., accusative not always precedes dative nor does dative precede accusative in all cases. Another option immediately comes to mind when examining the above examples: the word order varies with different types of verbs, so we could ascribe also markedness to this. However, how can we describe semantic differences of verbs in syntax? Obviously, this would entail further unwanted problems we should try to avoid.

We can't derive the markedness phenomena from clause structure or morphological features like case, nor can we gain it from semantic properties of the verb. But what about semantic features of the arguments? In (6.1) and (6.2), the unmarked sentence shows precedence of animate objects over inanimate; in the marked variation, inanimate objects precede animate. Thus we can identify animacy as a steering device for markedness (and also as a possible trigger for scrambling): *animate* > *inanimate*. Likewise, definiteness can be identified as a feature steering markedness: *definite* > *indefinite* as in (6.3).

(6.3) dass er dem Mann eine Tasche übergab.
that he the man$_{dat}$ a bag$_{akk}$ gave.

Finally, focus also is such a feature, however the precedence relation here is inverse: focussed arguments align with the right edge of the VP, whereas topicalized (i.e., accentuated or in any other way highlighted) elements align left. Therefore, [-focus] > [+focus].

We have seen that features like definiteness, animacy and focus can influence the word order of arguments in German. In the following these features will be used to construe an OT analysis of word order variation.

6.2 Overview and Basic Assumptions

Müller (1998) suggests an approach for motivating word order variation by postulating a "Scrambling Criterion" (SCR-CRIT) constraint ranked higher than the constraint that blocks movement (STAY/*t). SCR-CRIT does not encode a possible trigger for NP movement, but instead is composed of a constraint (sub)hierachy that incorporates features like definiteness, animacy, focus, etc., driving the scrambling movement. He thus proposes to split up the overall constraint ranking into a matrix hierarchy and a subhierarchy that itself builds a constraint in the matrix hierarchy. This allows for a treatment of word order variation in OT, where principally an inferior candidate is ungrammatical, but in natural language, variation need not necessarily entail ungrammaticality but markedness. For Müller, ungrammaticality arises only with fatal violations of constraints in the matrix hierarchy; the subhierarchy serves for determining markedness. Language-specific parameterization can follow on the one hand from reranking of the matrix hierarchy (i.e. forbidding scrambling by ranking STAY/*t higher than SCR-CRIT) and on the other hand by differences in the scrambling subhierarchy. All of this, of course, presupposes that word order variation does not happen due to differences in the base generation of the arguments of the verb, as is suggested in other analyses (see Müller 1998, 4 for references regarding base generation).

Müller assumes an underlying clause structure of German (simplified), showed in (6.4):

(6.4) $[_{CP}$ – C $[_{TP}$ – $[_{\pi P}$ – $[_{VP}$ Adj $[_{VP}$ SUBJ $[_{V'}$ DO $[_{V'}$ IO $[_{V'}$ OBL V]]]]]] π] T]

SpecC is the landing site for *wh*-movement, SpecT(ense) for subject raising (optional in German). Specπ is the position assumed to be the landing site for weak pronoun movement, i.e. the so-called "Wackernagel" position. The D-structure order of subject (SUBJ), direct object (DO) and indirect object (IO) always (i.e. with all types of verbs!) is assumed to be as in (6.1). Evidence for this comes from the order of weak pronouns in German clauses.[1] Closest to the base position of the verb are oblique arguments realized by NPs bearing lexical case (e.g., genitive) or by PPs. The word order determined by (6.1) can be varied through movement to a specifier position (SpecC, SpecT, Specπ), or by scrambling. Furthermore, Müller assumes that scrambling in German can only be analyzed as adjunction to VP and not to NP, PP, TP etc., and that it is a property of scrambling to be iterable (in contrast to movement to specifier positions).

[1] In the case of three pronomimal arguments, their order is fixed as SUBJ > DO >IO: "dass sie$_{SUBJ}$ es$_{DO}$ ihm$_{IO}$ gegeben hat." Subject pronouns thus obligatorily precede object pronouns, and similarly, a direct object pronoun always prececes a indirect one. (Cf. Müller 1998, 15ff.)

6.3 The Constraints

Let me now turn to the relevant constraints for an Optimality-theoretic approach to scrambling: Müller (1998) distinguished two types of constraints, viz., markedness constraints that trigger movement ("X-Criteria", Müller 1998, 13), and faithfulness constraints that prohibit or minimize the effects of movement. As we have seen according to the basic German clause structure in (6.4) that weak pronouns show up in the "Wackernagel" position πP, Müller (1998) assumes a constraint that forces movement of weak pronouns to the domain of the functional head π at S-structure: the *Pronoun Criterion*, or PRON-CRIT (Müller 1998, 14):

(6.5) Pronoun Criterion (PRON-CRIT):
Weak pronouns must be in the domain of π at S-structure.

The domain of a head X comprises SpecX and XP adjuncts, perhaps also X adjuncts. Müller (1998) assumes that only one pronoun can be substituted in Specπ to fulfill PRON-CRIT; further pronouns adjoin to πP (p. 14).

Next, the *Extended Projection Principle,* or EPP, (cf. Chomsky 1995) requires NPs with nominative case to be in SpecT (the subject position) at S-structure (Müller 1998, 14):

(6.6) Extended Projection Principle (EPP):
NP_{nom} must be in SpecT at S-structure.

These two constraints are the markedness constraints triggering movement of pronouns and subjects. They are counteracted by the faithfulness constraint STAY:

(6.7) STAY:
S-structure movement is not allowed.

Müller presupposes another faithfulness constraint minimizing the effect of syntactic movement, demanding parallel movement of NPs (Müller 1998, 15):

(6.8) Parallel Movement (PAR-MOVE):
If α c-commands β at level L_n, then α c-commands β also at level L_{n+1} (where α, β are arguments).[2]

The ranking of these four constraints for German is as following: PRON-CRIT necessarily dominates STAY (since weak pronouns move to πP) and PAR-MOVE, and EPP and STAY are tied:

(6.9) Constraint ranking for German:
PRON-CRIT ≫ STAY ≪≫ EPP ≫ PAR-MOVE.

A constraint tie means that a candidate may be optimal under any possible ranking of the tied constraints, i.e. either STAY or EPP may dominate the other in a competition while resulting in the same optimal output.

With these constraints we are able to explain weak pronoun movement in German, allowing us to verify the ranking in (6.9). In (6.10a), the two pronouns (the DO and IO) precede the subject NP as expected, whereas the sentence in b) is impossible:

(6.10) a) dass es_1 ihm_2 der Fritz t_1 t_2 gegeben hat.
that it_{acc} him_{dat} DET $Fritz_{nom}$ given has.
"that Fritz gave it to him."
b) *dass ihm_2 es_1 der Fritz t_1 t_2 geben hat.

[2] c-command: A node A c-commands a node B iff (i) A does not dominate B and B does not dominate A; and (ii) the first branching node dominating B also dominates B. (Haegeman 1994, 134).

In OT we can account for this in the following way: the partial ranking PRON-CRIT ≫ STAY implies that weak pronouns must undergo movement to the domain of π at S-structure, either to Specπ ot to a πP-(left-)adjoined position.

The fixed order of the pronouns is ensured by PAR-MOVE. Under the partial ranking PRON-CRIT ≫ PAR-MOVE weak pronouns can cross subject NPs, thereby violating PAR-MOVE in order to satisfy the higher-ranked PRON-CRIT. Judging only by PRON-CRIT, all orders of pronouns were equally well, so this is where PAR-MOVE comes into action. A low-ranked PAR-MOVE predicts that the D-structure order of arguments is preserved at S-structure (if possible), exactly what we need for the analysis. Furthermore, this behavior strengthens the assumption of the underlying clause structure in (6.4), with DO uniformly preceding IO.

With the constraints in (6.5) through (6.8) it is possible to analyze weak pronoun fronting in German, providing evidence for the underlying clause structure in (6.4). But so far, we are not able to account for scrambling, as no motivation for movement of real nouns (i.e., not just pronouns) has yet been presented. Because of STAY and PAR-MOVE, another trigger is needed: Müller (1998) proposes a *Scrambling Criterion* SCR-CRIT, similar in nature to PRON-CRIT.

In contrast to pronoun movement, reordering of Mittelfeld-internal arguments does not necessarily invoke ungrammaticality. Thus, the treatment of such reordering must differ somehow from that of pronoun fronting, and SCR-CRIT therefore needs to differ from PRON-CRIT. For this reason Müller (1998) assumes the *Scrambling Criterion* to consist of several "sub-constraints" which build up SCR-CRIT. This results in two constraint levels, a matrix hierarchy and a subhierarchy. Fatal violations on the matrix hierarchy necessarily induce suboptimality in the sense of pure OT, i.e. strict ungrammaticality. All constraints discussed until now (besides SCR-CRIT) belong to that matrix hierarchy. In contrast, fatal violation on a subhierarchy only leads to markedness. The constraints that trigger scrambling belong to this latter hierarchy, and accordingly we find degrees of markedness with candidates in this domain.

SCR-CRIT consists of several conflicting linearization constraints; to distinguish the subhierarchy from the matrix hierarchy, Müller uses > to indicate ranking on the subhierarchy (in constrast to ≫ on the matrix hierarchy). Thus, SCR-CRIT looks as in (6.11) (Müller 1998, 22):

(6.11) Scrambling Criterion (SCR-CRIT)[3]

In the VP domain,

a) NOM ('Nominative constraint'): [+nom] precedes [-nom] >
b) DEF ('Definiteness constraint'): [+def] precedes [-def] >
c) AN ('Animacy constraint'): [+animate] precedes [-animate] >
d) FOC ('Focus constraint'): [-focus] precedes [+focus] >
e) DAT ('Dative constraint'): [+dat] precedes [-dat] >
f) ADV ('Adverb constraint'): $NP_{[+def]}$ precedes a VP adverb >
g) PER ('Permutation constraint'): If α c-commands β at level L_n, then α does not c-command β at level $_{Ln+1}$.

As for the ranking on the matrix hierarchy, clearly SCR-CRIT must dominate STAY and PAR-MOVE, since scrambling exists in German and may change the word order of NPs in the VP. Furthermore, Pron-Crit must dominate SCR-CRIT because weak pronouns must move to the p domain (where SCR-CRIT does not apply), rather than showing up inside VP in order to satisfy any of the SCR-CRIT linearization sub-constraints. Hence, the ranking will be as in (6.12):

[3] See Müller (1998, 22).

(6.12) Ranking for German

PRON-CRIT ≫ SCR-CRIT ≫ EPP ≪≫ STAY ≫ PAR-MOVE

After having introduced subhierarchies to OT, we need to redifine the notion of opimality, since subhierarchies are not accounted for in traditional OT. As markedness also needs to be defined, a distinction between optimality as unmarkedness and optimality as grammaticality is required.

Basically, the definition of optimality as grammaticality should remain the same: we only need to incorporate subhierarchies. In the case of a constraint that is a subhierarchy, the winning candidate is optimal if the subhierarchy is replaced by a constraint that belongs to the subhierarchy (Müller 1998, 23). In logical terms: the subhierarchy is interpreted via disjunction of its constraints – the relative ranking of these internal constraints is irrelevant.

(6.13) *Grammaticality* (Müller 1998, 23)

A Candidate K_i is grammatical iff, for every candidate K_j in the same candidate set, K_i satisfies the highest-ranking constraint B_k of the matrix hierarchy <B_1, B_2, … B_n> on which K_i and K_j conflict better than K_j, where B_l is replaced by some C_k in <C_1, C_2, … C_n> if B_l is a subhierarchy <C_1, C_2, … C_n>.

In the present case of SCR-CRIT, supposing that SCR-CRIT is the highest-ranking constraint, this means that a candidate will be optimal when SCR-CRIT is replaced by any of the linearization constraints in (6.11). Concretely, a candidate would be grammatical if it is optimal under a ranking where, e.g., SCR-CRIT is replaced by NOM (or DEF, and so on).

Turning now to markedness, we can record that the notion of unmarkedness can be defined similarly:

(6.14) *Unmarkedness* (Müller 1998, 23f.)

A Candidate K_i is grammatical iff, for every candidate K_j in the same candidate set, K_i satisfies the highest-ranking constraint B_k of the matrix hierarchy $<B_1, B_2, \ldots B_n>$ on which K_i and K_j conflict better than K_j, where B_l is replaced by $<C_1, C_2, \ldots C_n>$ if B_l is a subhierarchy $<C_1, C_2, \ldots C_n>$.

The crucial difference is that the unmarked candidate is determined not by substituting any of the constraints of the subhierarchy for SCR-CRIT, but by substituting *all* of the constraints for it. Thus, for the determination of unmarkedness, the distinction of matrix hierarchy and subhierarchy is overridden. Clearly, it follows that an unmarked candidate is grammatical, but not the inversion. What is not yet incorporated is how to account for different degrees of markedness. Here, Müller adopts Keller's (1996) concept of suboptimality in (6.15): "Among the grammatical candidates of a candidate set (determined according to the definition in (6.13)), a candidate K_j is more marked than another candidate K_i if K_j is suboptimal with respect to K_i according to the definition [of unmarkedness in (6.14)] – i.e., the worse the constraint profile of a grammatical candidate is, the more marked it is."(Müller 1998, 24).

(6.15) *Suboptimality* (Keller 1998; cit. Müller 1998, 20)

A structure S_i is suboptimal with respect to a structure S_j if there are subsets R_i and R_j of the reference set such that S_i is optimal for R_i and S_j is optimal for R_j and $R_i \subset R_j$ holds. A structure S_i is less grammatical than a structure S_j if S_i is suboptimal with respect to S_j.

With these theoretical assumptions as background we will now return to empirical evidence to prove the rankings in (6.11) and (6.12), respectively, as well as the constraints involved. I will restrict analyses to only some representative examples, just to be able to principally verify the subhierarchy. For a full proof see Müller (1998, 24–36). First, consider the pair of sentences in (6.16) (Müller 1998, 24):

(6.16) a) dass eine Frau den Fritz geküsst hat.
that a woman$_{nom}$ DET Fritz$_{akk}$ kissed has.

b) ?dass den Fritz eine Frau geküsst hat.
that DET Fritz$_{akk}$ a woman$_{nom}$ kissed has.

There is a conflict here between Nom (which requires the order in a) and Def (which in turn demands the order in b). Both orders are grammatical, but generally it is assumed that b. is more marked than a. Thus we can conclude that in the ranking of the subhierarchy, the "Nominative Constraint" Nom dominates the "Definiteness Constraint" Def:

Tableau$_{6.1}$

Candidates	Scr-Crit							Stay/*t	Par-Move
	Nom	Def	An	Foc	Dat	Adv	Per		
a)		*					*		
b)	*							*	*

Candidate a) is grammatical under substitution of Scr-Crit by Nom (or any of the other linearization constraints but Def and Per[4]) but also unmarked (if the subhierarchy replaces Scr-Crit as a whole, only this candidate emerges as optimal due to the violation of the highest-ranked Nom by candidate b). Candidate b) in turn is grammatical when Scr-Crit is replaced by either Def or Per, which can for this reason be identified as trigger for the scrambling operation, but is therefore necessarily marked.

[4] Substitution of any of the constraints in the subhierarchy on which the two candidates do not differ will pass on the decision two the lower-ranked constraints Stay and Par-Move. Candidate a) does not involve scrambling (whereas candidate b) does) and thus does not violate these constraints. Therefore it has an „inherent advantage" (Müller 1998, 25) over candidate b) and will be optimal.

As a second example, let me illustrate the competition between animacy and focus. This will also serve as the basis of the implementation in chapter 7. Evidence for the ranking AN > FOC can be gained from the data in (6.17) (Müller 1998, 33):

(6.17) a) dass man	die Kinder	diesem	EINFLUSS	entzogen/	
that one	the children$_{acc}$	this	influence$_{dat}$	deprived/	
ausgesetzt/	ausgeliefert	hat.			
exposed/	extradited	has.			

b) ?dass man die KINDER diesem Einfluss entzogen/ausgesetzt/ ausgeliefert hat.

c) ??dass man diesem Einfluss die KINDER entzogen/ausgesetzt/ ausgeliefert hat.

d) ?*dass man diesem EINFLUSS die Kinder entzogen/ausgesetzt/ ausgeliefert hat.

Tableau$_{6.2}$

Candidates	SCR-CRIT							STAY/*t	PAR-MOVE
	NOM	DEF	AN	FOC	DAT	ADV	PER		
a)					*		*		
b)				*	*		*		
c)			*					*	*
d)			*	*				*	*

Candidates a) and b) vary only with respect to the position of the focused element (both violate DAT and DEF): In candidate b), there is a violation of FOC since not the VP-final element is focused. This renders candidate b) more marked then candidate a). Candidates c) and d) both involve scrambling (of the IO) and thus both violate STAY and PAR-MOVE. Additionally, both violate AN (avoiding violations of DAT and PER), hence they lose to candidates a) and b). However, the animacy constraint acts as trigger for scrambling here (hence the violations of STAY and PAR-MOVE), so the violations only lead to (further and stronger) markedness –

all four candidates are (correctly) predicted to be grammatical, even if candidate d) already appears close to the border of ungrammaticality.

I will confine myself to these examples as by now it should be clear how the ranking of the subhierarchy can be deduced.

We have found that it is quite easy for candidates with different VP-internal word order to be grammatical, how markedness arises and how this can be analyzed. What remains to be done is to rule out instances of iterated scrambling: If scrambling does not lead to any improved behaviour regarding one of the linearization constraints of SCR-CRIT, then we want that case to be ungrammatical. Compare the following sentences (Müller 1998, 35):

(6.21) a) dass [$_{VP}$ die Maria$_1$ [$_{V'}$ den Fritz$_2$ geküsst hat]].
that DET Maria$_{nom}$ DET Fritz$_{acc}$ kissed has.
b) *dass [$_{VP}$ die Maria$_1$ [$_{VP}$ den Fritz$_2$ [$_{VP}$ t$_1$ [$_{V'}$ t$_2$ geküsst hat]]]].

So-called *string-vacuous scrambling* is straightforwardly blocked under the discussed assumptions. Candidate b) accumulates a second violation of STAY, which necessarily becomes decisive in this context and rules out candidate b). in favor of the more ecomomic candidate a). Put simply, there is just no motivation for string-vacuous scrambling, as there is always a more ecomomic candidate.

Finally, I want to address the topic of cross-linguistic variation. Scrambling not only exists in German, there is also substantial evidence for a scrambling operation, e.g., in Dutch and Icelandic (see Müller 1998, 36 for references). In these languages however, it must be order-preserving: Dutch and Icelandic allow DO and IO to precede or follow adjunct, as long as the D-structure order of the arguments is obeyed. We can account for this if, in the matrix hierarchy, SCR-CRIT dominates STAY but is in turn outranked by PAR-MOVE. All candidates that satisfy SCR-CRIT by changing the order of arguments are filtered out by PAR-MOVE and all that maintain argument order are not. Thus languages of this type (i.e. with the Dutch/Icelandic constraint ranking) show considerably less freedom of word order than German, which, in turn shows less than a language like Russian[5]. In turn, English prohibits all kinds of scrambling and thus exhibits considerably less freedom of word ordering than languages of the Dutch/Icelandic type. The ranking for English differs from that of Dutch/Icelandic by ranking STAY too over SCR-CRIT, thus blurring its effects.

The advantage of the approach of Müller (1998) is that scrambling is not tied to other, independently motivated properties of a language (e.g., morphological case) but solely depends on the relative ranking of the three constraints SCR-CRIT, STAY, and PAR-MOVE.

6.4 Summary

The goal of this chapter was to present the Optimality-theoretical approach of Müller (1998) to scrambling. We have seen that a *Scrambling criterion* SCR-CRIT is proposed that consists of multiple features hierarchically ranked themselves to form a constraint subhierarchy embedded in the overall (matrix) hierarchy. Among the features that make up SCR-CRIT are syntactic attributes like

[5] Russian exhibits more landing sites for scrambling than German. According to (Müller 1998, 5, footnote 3), Russian permits scrambling to TP, CP, and NP.

definiteness, animacy, focus, and others. Formulated as linearization (precedence) constraints, these features can be seen as triggers for scrambling.

The distinction of a matrix hierarchy and a subhierarchy furthermore allows Müller to introduce a notion of markedness into Optimality Theory that is not accounted for in traditional OT. By incorporating subhierarchies into the definition of optimality, the grammaticality of all candidates involving scrambling is assured by replacing the subhierarchy constraint SCR-CRIT with any one of its internal linearization constraints (i.e. a linearization constraint does not play a role for grammaticality). Markedness on the other side is analyzed by substituting the entire subhierarchy for SCR-CRIT, and there the different violations in the subhierarchy become decisive: The worse the violations are, the more marked the candidate is.

The advantage of Müller's (1998) account of scrambling is that it is not dependent of any non-syntactic properties. Also, it manages without any other, independently motivated constraint. Cross-linguistic variation can simply be derived from re-ranking of the three constraints SCR-CRIT, STAY, and PAR-MOVE. The model of Müller (1998) will serve as basis of the computational approach to scrambling in the following chapter.

7. Implementing an OT Approach to Scrambling

Before starting off now with the actual implementation of Müller's (1998) scrambling analysis, a few things need to be mentioned. Above all, I do not intend to argue against other syntactic theories in favor of Optimality-theoretic syntax in general. Also, there is no claim whether OT is a psychologically plausible model of the human speech faculty. The reason for choosing OT as framework for the analysis is that, to me, it offers the best means for this task. Beside this it is the goal of this work and in particular this chapter to show how – against all odds – OT systems can be implemented straightforwardly given the necessary restrictions. It is not the goal, however, to provide a full implementation of an OT syntax system, it will be limited to the analysis of scrambling in German.

Secondly, the choice of Müller's analysis does not mean it were any more plausible than other analyses. Due to the perspective on features it seems more appropriate for a computational approach than such focusing more on the interaction of syntax and phonology, i.e. ascribing scrambling partly to phonological properties such as emphasis. It may even be argued for choosing such approaches as they might be nearer to the actual human speech faculty; yet, building a syntax system upon phonological properties that are of no direct use for a computational account of a strictly syntactical phenomenon (that scrambling is, no matter what explanations for its appearance may be given) does not seem appropriate to me. Clearly this factors out the strength of OT lying in the description of such interaction phenomena, but it is not the goal of this work to build a system of the entire speech faculty.

Accordingly, I will not implement a full-scale OT system but a configuration that corresponds to the needs and assumptions of Müller's analysis. GEN will only generate the X-bar structure that Müller underlies his work. Also, just the constraints used in the analysis will be inlcuded. The only difference from the original set-up will be that a highly specified input is assumed, which is needed for computation.

It is the goal of this section to show that an OT system can be implemented straightforwardly without complicated means, and how this can be done. The generation part of the system will be implemented using Prolog rules. Constraint marking also happens within the Prolog rules: the constraint are included in the rules in such a way that when a special construction is used (like a scrambling operation in the present example), the respective originating mark(s) are appended to the marks list during generation or parsing. Evaluation is used here in the sense of finding the optimal candidate via its marks in the case of generation, and analyzing the mark list of the input sentence in parsing.

7.1 The Input

Recalling the discussion in section 2.1, there is evidence that there is no need for an input (or at least a highly specified one) in Optimality Theory (cf. Heck et al. (2002)). However, for computation, things are different: An input is needed in order for generation to become possible. Thus it is our task to find a compromise between following the theory and making computation possible. Generally, already an input of very low specification would suffice, possibly a "part of speech"-only approach were enough. The generator would then simply create all strings of words possible – in the sense of the theory this were the ideal procedure. Yet, that way we would lack the information necessary for the constraints to apply later. As the constraints will refer to features such as case, definiteness, animacy, focus, etc. (cf. chapter 5), all this information must be present somewhere in the generated strings. If the input were only as weakly specified as suggested above, many further means were necessary for this. E.g., a tagger running over the generated sentences was needed in order to obtain the case information. To get the definiteness, focus and alike features, probably no further mechanism was needed: Those can be analyzed via the constraint ranking and the respective position of the element in the X-bar structure. However, animacy is semantic information, thus an interface to a semantic database like "WordNet" were needful. This would complicate the matter in a pointless way for the present purpose. However, the principal idea should in my opinion be kept in mind for a full-scale OT system built upon pre-existing devices, where such procedures might prove useful.

To avoid unnecessary complication here, I will assume all information to be encoded in the input. This comprises *case, gender, definiteness, animacy,* and *focus* features. A typical input Prolog fact will look as follows:

7.1 General input Prolog fact

```
nn(Case,Gender,Definiteness,Animacy,Focality) --> [word]
```

A specific example is given below:

7.2 Example input Prolog fact

```
nn(dat,m,Def,noanim,nofoc) --> ['Einfluss'].
```

The example stands for the following: An unfocussed, inanimate, indirect (dative) object “Einfluss“ (“influence“); where “nn“ is the part of speech type “noun“. A second example is given in (7.3), denoting a focussed, animate direct (accusative) object “Kind“ (“child“):

7.3 Example input Prolog fact

```
nn(akk,m,Def,anim,foc) --> ['KIND'].
```

The definiteness feature is represented by a Prolog variable, as definiteness is not encoded in the semantic information of a noun (contrary to animacy), but rather is expanded from the determiner to the entire noun phrase. Accordingly, definiteness is encoded in the determiner lexicon entries.

I will stick with these two examples for the moment, as the basic functioning should be clear. The entire code is given in the appendix.

7.2 The Generator

As regards GEN, we encounter a similar problem as with the input: The theory favors a weak GEN as part of the system, yet we need to ensure that its weakness stays within reasonable boundaries for computation. A weak GEN itself of course were not a problem: It would simply generate all possible analyses for the given input and let EVAL do all the work finding the correct output. What is the problem is that generating a large (or possibly infinite) amount of candidate analyses is very time-consuming. An OT system proceeding this way would just offer no serious competition to existing applications, as it would always be less efficient in terms of time and resources. What is sought, thus, is the right compromise between a weak GEN and computational demands. The obvious solution to this problem is to take an already existing generator and make sure it is still overgenerating enough so that an EVAL device applied after generation remains meaningful. To the generator probably only few changes are necessary for this; as most generating systems will produce unwanted outputs anyway, an OT EVAL add-on is the evident solution to remove those. On the other hand, grammars can be easily extracted from corpora, or basic grammars can also be written quickly by hand, and the EVAL add-on can be applied to minimize the problems resulting from such grammars. To me, this suggests that GEN should not be all too much of a concern when developing an OT system. This goes along with the assumptions of the theory – the basic strengths of Optimality Theory lie in the universal constraints and the evaluation part.

I will therefore write a very basic grammar for GEN, which will produce a modified version of the X-bar structures that Müller (1998) assumes to underlie all sentences. I will leave aside the π, πP and Adj positions as the analyses conducted by this systems are not concerned with pronouns and adjectives. Also, I lay the oblique case position OBL aside. The structure is given below, along with a syntax tree representation:

(7.4) [$_{CP}$ – C [$_{TP}$ – [$_{VP}$ SUBJ [$_{V''}$ DO [$_{V'}$ IO V]]]] T]

The grammar is written in Prolog and consists of the following phrase-structure rules:

(7.5) Basic GEN grammar

CP	→ C TP
TP	→ SpecTP T′
T′	→ VP T
VP	→ NP VP
VP	→ SUBJ V′′
V′′	→ DO V′
V′	→ IO V

The scrambling effect is achieved by the VP → NP VP rule: it allows for as many left-adjunctions of NP to VP as needed. That way, all possible allocations of the NPs within the VP domain are generated – exactly what we want to happen. A small constraint will be built into the rules to avoid double occurrences of the same NP.

For the analysis, I will only use finite verbs, which will be accordingly labeled with T. Therefore no actual V's will exist and all candidates share a *t-violation. Also, I assume with Müller that all subjects are in SpecTP, thus the candidates have a second *t-violation in common.

(7.6) shows a grammar rule in Prolog. This particular rule describes a scrambling operation where a dative, inanimated and focussed element is moved to the front of the VP (represented through the fact that the `vp5`-fact introduces a new VP-rule):

(7.6) Prolog GEN grammar rule

```
vp([an,foc,par-move|MarksVP5]) --> np(dat,_,_,noanim,foc),
vp5(MarksVP5).
```

From the `vp5`-rule, via the `MarksVP5` variable all marks possibly picked up earlier in the generation process are brought forward. Also, this rule now adds the animacy, focus, and par-move violations to the mark list.

This gives an overview how the GEN grammar works. The actual GEN mechanism, i.e. the operation that generates all the candidates, is depicted in (7.7).

(7.7) GEN

```
generator :- input,
             phrase(tp(MarkList),Cand,[]),
             writeq(Cand),writeq(MarkList),nl,
             assertz(cand(Cand,MarkList)).

gen_eval :- generator,fail;
            (cand(Cand,[dat,per,'*trace','*trace']);
            cand(Cand,[par-move,'*trace','*trace','*trace'])),
            nl,
            write('This is the optimal candidate: '),writeq(Cand),
            abolish(cand/2).
```

The `gen_eval` predicate, as its name says, first generates all candidates and then evaluates the optimal one. Here, we are only interested in the first step. It calls the `generator` predicate and opens a failure-driven loop, meaning that `generator` fulfills its task over and over until all possible candidates are found. When then, finally, `fail` itself fails, the candidates are handed on to the "Evaluator". The `generator` predicate accommodates the actual Generator: after `input` asks the user which input file is to be used, the `phrase` predicate (which is built in in prolog) calls the GEN grammar to produce a sentence (`tp(MarkList),Cand,[]`). `MarkList` and `Cand` are variables that get instantiated by generation, where `Cand` denotes the candidate sentence and `MarkList` its mark list. Then, a candidate-marklist pair is printed (`writeq(Cand),writeq(MarkList)`), and a Prolog candidate fact is asserted to the database (`assertz(cand(Cand,MarkList))`).

7.3 Constraints and Constraint Marking

In chapter 5 we have seen that there are principally two methods how the task of implementing constraints can be approached: The first option is to build independent constraint rules that apply at every node during the parsing process. This is the method Kuhn (2003) chooses for his approach to OT syntax, and it is the one that clearly lies closer to the theory. The second possibility is the one that is used in XLE, where the marks are built into special rules; whenever that rule is used, a mark is added to the optimality projection. This second approach has the advantage that no additional rules representing the constraints are needed; on the other side, the grammar automatically gets larger and more specified, because every construction involving a mark requires a rule. I will call this the *XLE approach.*

From the implementation perspective, clearly the *XLE approach* seems more feasible. When the marking operation occurs already during the generation operation, actually one step (i.e. constraint marking) can be dropped from the OT analysis procedure; the marks are added to the marks list whenever the rule that introduces a marks is used. Secondly, in the reverse parsing procedure, the same happens automatically: due to Prolog's reversibility, the same grammar is used for parsing, therefore the marks list is built when the input sentence presented to the grammar is recognized. This is also in the sense of bidirectional OT (see chapter 3): A sentence is parsed and all constraint mark information is collected, then this sentence and its marks can be compared to other variants of the sentence and the best version can be determined.

(7.8) again shows a prolog grammar rule with its respective marks, but this time we will regard it from the viewpoint of constraint marking.

(7.8) `v2bar([an,foc,dat,per|MarksV]) --> do3, vbar1(MarksV).`

This rule combines a direct object (`do3`) and a V′ (`vbar1`) to form a V′′-position (`v2bar`). From V′, the earlier accumulated marks (`MarksV`) are handed on to the new mark list; also, there are new marks (`an,foc,dat,per`) introduced, in this case animacy, focus, dative, and permutation, which are all violated by the type of direct object that comes to use here. The direct object of the `do3` type is an accusative, unanimated, and focused NP (`do3 --> np(akk,_,_,noanim,foc).`), which in this case is generated in front of an animated, unfocused dative NP (the indirect object, which is part of the V′ (`vbar1`) rule).

Now, if this particular rule is used during generation of a candidate, the above-mentioned marks get noted on that candidate's mark list. This clearly is a procedural advantage in comparison with Kuhn's approach, since no additional constraint marking operation is necessary. Constraint marking and generation come, as it is, as one integrated procedure. This may not be in the sense of the theory proper, and may also be somehow less lucid than having two separated operations, but certainly seems more effective in an implementation.

In parsing actually the same happens, just in reverse order: if the sentence can only be parsed with the use of this rule, then the marks automatically are written in the mark list of the sentence. One could say that, here, parsing simply has the purpose of accumulating the marks – pretty much the definition of constraint marking in the sense of the theory. The parsing process is implemented as follows:

(7.9) Parsing and Evaluation

```
parse(X) :- consult('parseinput.pl'),
            phrase(tp(MarkList),X,[]),
            ...
            eval_sent(MarkList).
```

After the input file for parsing is consulted, the process is similar to that of generation, the sole difference being that the candidate is known: it is the input sentence specified by the user (denoted by the variable `X`). The GEN grammar (invoked by the predicate `phrase(tp(MarkList),X,[])`) has only the task to accumulate the marks of the sentence, so the mark list can be handed on to evaluation (`eval_sent(MarkList)`).

The process resembles the formalization of constraint marking by Kuhn (see chapter 5): At every node (i.e., that is parsed), it is checked which marks are introduced at that point, and these marks are then added to the mark list. The difference is that not all marks are checked at every node, and those that are violated are added to the list, but the marks are in a sense part of the node being checked.

7.4 Candidate Evaluation

7.4.1 Evaluation at Generation

To simplify the evaluation process for generation a little bit, I have made the restriction that only the optimal candidate is sought. I.e., not all candidates are evaluated. This is, if slightly modified, also in the sense of Optimality Theory, where all candidates are evaluated *to find the optimal one.* So to speak, a shortcut is taken by only looking for the mark list that belongs to the optimal candidate. This is possible in our example, as the optimal mark list is known: the optimal candidate always is the one where no scrambling has happened. (7.9) shows the respective part of the `gen_eval` predicate:

(7.10) Evaluation at Generation

```
 (cand(Cand,[dat,per,'*trace','*trace']),
nl,
write('This is the optimal candidate: '),writeq(Cand),
abolish(cand/2).
```

In the last step the candidates are erased from the database so candidates from different competitions are not mingled. And, of course, if not the optimal but another candidate is sought, this could be achieved in a similar way.

Furthermore, the original evaluation process could also be implemented straightforwardly: Via simple occurrence check, the candidates with the highest constraint mark in their profile are erased from the database. Then, the candidates with the second-highest mark are deleted, and so on, until only one candidate, the optimal solution is left. In a competition with real ungrammaticality (not just markedness, as in the present case) and an unknown winner, this procedure was necessary; but for our example it can be omitted as this solution will suffice.

7.4.2 Evaluation at Parsing

As regards parsing, we encounter a different case: we do not want that just the optimal solution is printed, but we want to know, how the parsed sentence would do in a competition, and if it wasn't the optimal solution, what its deficiencies were. With (second) language learning being a possible field of application, this certainly is the desired feedback. To achieve this, the mark list of the parsed sentence is evaluated in the following way: if it is not the optimal candidate, the type of markedness incurred by the constraint violations is printed. This happens again via occurrence check, as (7.10) illustrates: if a mark can be found on the mark list, the system provides feedback about the type of markedness of the sentence.

(7.11) Evaluation at Parsing

```
eval_sent(Marks) :- member(an,Marks),
                    write('This sentence is marked due to a violation
                           of animacy of the scrambled np.').
```

7.5 Summary

In this chapter I have demonstrated that it is possible to implement a syntactic application based upon Optimality Theory, if the necessary restrictions are made. In our case, this is a system that generates or analyses German sentences that exhibit certain variations in the word order of the “Mittelfeld”, so-called scrambling. The background of the system is given in the OT scrambling analysis in Müller (1998). The basis of the program is a simple phrase structure grammar, the GEN grammar that is capable of generating all variations of word order for a sentence. The input facts are enriched with the features necessary for the analyses, in this instance comprising case, animacy, and focus, but this could be extended to all desired features of Müller’s analysis. The phrase structure rules also accommodate the constraint marks: whenever a rule (i.e., a node in the tree structure) is used in generation or parsing, the marks belonging to that rule are noted on the candidate’s mark list. This simplifies the constraint marking process somewhat, as no additional procedure is necessary; however, I do not follow the theory proper here. Candidate generation is represented by a failure-driven loop that generates all possibilities from the input until no more are found; then, the candidates, along with their marks are handed on to evaluation. This task is facilitated as well, as a shortcut is used to determine the optimal candidate. Like in the case of scrambling, the optimal mark list (i.e., the marks of the un-scrambled sentence) is already known in advance, we only need to search for that mark list and the inherent candidate. A genuine evaluation procedure is easily conceivable, but can be omitted in this example.

With regard to parsing, the GEN grammar returns the mark list of the given input sentence, if a corresponding structure could be found. So to speak, parsing in this sense actually serves as a replacement for the constraint marking procedure. Evaluation here analyses the marks of the parsed sentence and provides feedback about the (un-)markedness of the sentence. This feedback could be then, e.g., used in automatic language learning.

My simple example has shown, that Optimality Theory can be implemented straightforwardly if some restrictions are made, and that it could be of use where other, traditional, linguistic theories fail.

8. Conclusions

This study presented an insight into the field of Optimality Theory and Optimality-theoretic syntax in linguistics and its applicability in computational linguistics in particular. I gave an overview of the research in computational OT that has been made in the past years, yielding that – against all odds that have arisen from theoretical occupation with the matter – assuming the necessary restrictions Optimality Theory can be formalized and implemented. Furthermore, I argued in favor of a field of application conceivable for OT in computational linguistics, where traditional approaches get into difficulties. A small example has underpinned this claim. However, at the present stage, the theory, especially in the field of syntax, still has too many obscurities as for full-scale implementation to become conceivable. It is also unclear whether OT applications present a serious competition to traditional approaches due to the greater complexity of the main devices. The conclusion I draw is that at the moment no final judgment can be made, as too much is yet unknown on both the theoretical and practical side.

In the first two chapters, I presented the OT framework and its underlying assumptions. OT syntax can be seen as a meta-theory that applies upon traditional approaches, replacing some of the devices of that theory. In Optimality Theory, a strong generative device (GEN) is postulated, which can be seen as a very general phrase structure grammar. With this grammar, a set of sentences (the candidates) for a given input is generated. The input still is the matter of ongoing discussions in syntax, with theories ranging from highly specified inputs to having no input at all. After generation, a device applies that checks whether the candidates violate certain restrictions, the so-called constraints. These constraints are in a relation of strict dominance, where the theory allows the violation of lower-ranked constraints in order to fulfill some higher ranked constraint, the number of violations not being of importance. It is a fundamental assumption of OT that the set of constraints is the same in all languages and that languages differ only by the specific ranking of the constraints. Finally, the constraint profiles of the candidates are compared with each other to find the best, i.e. optimal, solution, yielding the one candidate violating less grave constraints than all of the other candidates. However, although these fundamentals of the theory are well understood, there remains much work to be done in the field of OT syntax until a general theory can be formulated.

The second part (chapters 3 through 5) illuminated the formal and computational properties of an Optimality-theoretic approach to syntax. In chapter 3, I discussed the theoretical aspects of Optimality Theory and its application in computational linguistics. I demonstrated how a formalization should be approached and what restrictions are necessary, and what OT's field of application could be. In chapter 4, I gave an overview of research already done in computational OT. Much has been achieved in phonology, where several computational applications exist that are at work and that provide some intuitions with regard to a computational approach to OT syntax. On the more theoretical side, many studies have shown how OT methods could be used in language learning and multilingual applications. Others have suggested various approaches towards a formalization of OT syntax. Chapter 5 then presented Kuhn's formalization of OT syntax, whose work has been the basis of the first larger-scale attempt to make use of OT methods in a syntax system. The Optimality-theoretic functionality of the Xerox Linguistic Environment was accordingly explored in the following. The conclusions that can be gained from the investigations in these chapters show that despite the research that is still necessary, OT syntax can have its place in computational linguistics.

In chapters 6 and 7, I made an attempt to show that Optimality Theory can be implemented straightforwardly. As background for the application, I chose an analysis of scrambling (word order variation) in German, a property of language where traditional syntactic theories prove deficient regarding its treatment. It is assumed that scrambling is triggered by a constraint that forces the relocation of NPs due to features like animacy, case, and focus (chapter 6). Chapter 7 presented the program, which is written in Prolog. GEN is represented as a very basic phrase structure grammar, the input is assumed to be structured in the sense that it contains information about the animacy, focus, etc. of the words. Generating all possible candidates is implemented by a failure-driven loop that forces the grammar to produce all possible trees (sentences) for the given input. Constraint marking is part of the generation process, as the constraints are part of the rules: whenever a rule is used, the marks it incurs are noted on the mark list. In the case of parsing, a input sentence is presented to the GEN grammar, and during recognition, its mark list is accumulated. Evaluation is understood here in the sense that feedback about the (un-)markedness of the parsed sentence is given. In generation, evaluation is reduced to the task of finding the optimal candidate via the optimal mark list, since the best solution (i.e., the un-scrambled sentence) is known in advance. A procedure nearer to the proper sense of evaluation has also been outlined.

One may hope now that research will ultimately yield a full-scale and detailed representation of Optimality-theoretic syntax. Areas of use like automatic language learning, multilingual applications, and finally overcoming the deficiency of traditional approaches provide motivation for further research. Within the present framework, there is a number of points that still have to be addressed, for instance, the issue of the input in syntax, or the number and form of syntactic constraints. With computational research at its beginnings, linguists can hope for a clarification of such fundamentals. On the other side, the improvement of the syntactical framework will strengthen computational approaches.

Appendix A – Program Code

A.1 OptiScramble.pl

```
%                                                                %
% This is OptiScramble, a tool that generates and analyses      %
% German sentences with Mittelfeld variation and marks all      %
% constraint violations for determinig the optimal solution.    %
%                                                                %
% Author: JF                                                     %
% Date:   7/20/05                                                %
% Version: First usable ;-)                                      %
%                                                                %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



% End of header
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Program Rules
%%%%%%%%%%%%%%%

dynamic(cand/2).

input :- write('Please specify an input file: '), % Specify the input file for candidate
         read(USERINPUT),                         % generation.
         consult(USERINPUT).


%% Generate and evaluate
%%%%%%%%%%%%%%%%%%%%%%%%

generate :- input,                                         % Help predicate for grammar
            findall(Cand-MarkList,phrase(tp(MarkList),Cand,[]),Candidates),  % debugging:
            write('Candidates:'), nl,nl,                      % generates all candidate-
            writeq(Candidates), nl,nl,                        % marklist pairs.
            write('no more candidates were found.').

output :-   input,
            findall((Cand,MarkList),phrase(tp(MarkList),Cand,[]),Candidates), % Output all
            tell('candidates.txt'),                                           % candidate-
            writeq(Candidates), nl,nl,                                        % marklist
            told.pairs                                                        % to a file

generator :- input,                                                  % Generates candidate-
             phrase(tp(MarkList),Cand,[]),                           % marklist pairs for
             writeq(Cand),writeq(MarkList),nl,                       % immediate evaluation
             assertz(cand(Cand,MarkList)).

gen_eval :- generator,fail;                                         % Takes the generated
            (cand(Cand,[dat,per,'*trace','*trace']);                % candidate-marklist
             cand(Cand,[par-move,'*trace','*trace','*trace'])),     % pairs and determines
            nl,                                                     % the optimal solution
            write('This is the optimal candidate: '),writeq(Cand),
            abolish(cand/2).


%% Parse and evaluate
%%%%%%%%%%%%%%%%%%%%%

parse(X) :- consult('parseinput.pl'),                           % Parses an user-specified
            phrase(tp(MarkList),X,[]),nl,                       % input sentence and
            write('Candidate: '), writeq(X),                    % evaluates its marklist
            write(', Marks: '),writeq(MarkList),nl,             % (see preds below)
            eval_sent(MarkList),nl.

eval_sent(Marks) :- member(an,Marks),member(foc,Marks),
                    write('This sentence is marked due to a violation of animacy and focus
                           of the scrambled np.').
```

```
eval_sent(Marks) :- member(an,Marks),
                    write('This sentence is marked due to a violation of animacy of the
                           scrambled np.').

eval_sent(Marks) :- member(foc,Marks),
                    write('This sentence is marked due to a violation of focus of the
                           scrambled np.').

eval_sent(Marks) :- Marks = [dat,per,'*trace','*trace'];
                    Marks = [par-move,'*trace','*trace','*trace'],
                    write('This sentence is optimal.').


%% Evaluate given input
%%%%%%%%%%%%%%%%%%%%%%%

find_optimal :- consult('candidates.pl'),                     % Finds the optimal candidate
                (c(Cand,[dat,per,'*trace','*trace']);         % from a specified candidate
                 c(Cand,[par-move,'*trace','*trace','*trace'])),          % file
                write('This is the optimal candidate: '),write(Cand).


%% General prolog predicates
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

append([],L,L).
append([H|T],L,[H|NT]) :-
           append(T,L,NT).

member(X, [X|Y]).
member(X, [H|L]) :- member(X, L).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GEN Grammar                                                                                  %
%%%%%%%%%%%%%%                                                                                 %
%                                                                                              %
% generates the sentence structure according to Müller's 1998 OT scrambling analysis (the %
% Animacy-Focus-Dative competition). Marks are accumulated during generation/parsing.     %
% The various VP rules are used for the different scrambling operations, taking into      %
% account the np-features form the input.                                                 %
% No ungrammatical structures are generated.                                              %
%                                                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

tp(MarkList) --> spectp, tbar(MarkList).

tbar(MarkList) --> t, vp(MarkList).
tbar(MarkList) --> t, vp2(MarkList).

vp([par-move|MarksVP3]) --> np(dat,_,_,anim,nofoc), vp3(MarksVP3).
vp([foc,par-move|MarksVP4]) --> np(dat,_,_,anim,foc), vp4(MarksVP4).
vp([an,foc,par-move|MarksVP5]) --> np(dat,_,_,noanim,foc), vp5(MarksVP5).
vp([an,par-move|MarksVP6]) --> np(dat,_,_,noanim,nofoc), vp6(MarksVP6).
vp([par-move|MarksVP7]) --> np(dat,_,_,noanim,nofoc), vp7(MarksVP7).
vp([foc,par-move|MarksVP8]) --> np(dat,_,_,noanim,foc), vp8(MarksVP8).

vp2(MarkList) --> subj(MarksSubj), v2bar(MarksV2bar),
                  {append(MarksV2bar,MarksSubj,MarkList)}.

vp3(MarksVP3) --> subj(MarksSubj), v2bar2(MarksVbar1),
                  {append(MarksVbar1,MarksSubj,MarksVP3)}.
vp4(MarksVP4) --> subj(MarksSubj), v2bar3(MarksVbar1),
                  {append(MarksVbar1,MarksSubj,MarksVP4)}.
vp3(MarksVP4) --> subj(MarksSubj), v2bar4(MarksVbar1),
                  {append(MarksVbar1,MarksSubj,MarksVP4)}.
vp4(MarksVP3) --> subj(MarksSubj), v2bar5(MarksVbar1),
                  {append(MarksVbar1,MarksSubj,MarksVP3)}.
vp5(MarksVP5) --> subj(MarksSubj), v2bar3(MarksVbar1),
                  {append(MarksVbar1,MarksSubj,MarksVP5)}.
vp6(MarksVP6) --> subj(MarksSubj), v2bar2(MarksVbar1),
                  {append(MarksVbar1,MarksSubj,MarksVP6)}.
```

```
vp7(MarksVP7) --> subj(MarksSubj), v2bar4(MarksVbar1),
                  {append(MarksVbar1,MarksSubj,MarksVP7)}.
vp8(MarksVP8) --> subj(MarksSubj), v2bar5(MarksVbar1),
                  {append(MarksVbar1,MarksSubj,MarksVP8)}.
v2bar([foc,dat,per|MarksV]) --> do, vbar1(MarksV).
v2bar([dat,per|MarksV]) --> do1, vbar(MarksV).
v2bar([an,foc,dat,per|MarksV]) --> do3, vbar1(MarksV).
v2bar([an,dat,per|MarksV]) --> do2, vbar(MarksV).
v2bar2(MarksVbar1) --> do, vbar2(MarksVbar1).
v2bar3(MarksVbar1) --> do1, vbar2(MarksVbar1).
v2bar4(MarksVbar1) --> do3, vbar2(MarksVbar1).
v2bar5(MarksVbar1) --> do2, vbar2(MarksVbar1).

vbar(MarksV) --> io, v(MarksV).
vbar(MarksV) --> io2, v(MarksV).
vbar1(MarksV) --> io1, v(MarksV).
vbar1(MarksV) --> io3, v(MarksV).
vbar2(['*trace'|MarksV]) --> trace, v(MarksV).

spectp --> np(nom,_,_,_,_).
t --> vv.
subj(['*trace']) --> trace.
do --> np(akk,_,_,anim,foc).
do1 --> np(akk,_,_,anim,nofoc).
do2 --> np(akk,_,_,noanim,nofoc).
do3 --> np(akk,_,_,noanim,foc).
io --> np(dat,_,_,anim,foc).
io1 --> np(dat,_,_,anim,nofoc).
io2 --> np(dat,_,_,noanim,foc).
io3 --> np(dat,_,_,noanim,nofoc).
v(['*trace']) --> trace.

np(Cas,Gen,Def,Anim,Foc)--> det(Cas,Gen,Def,Anim,Foc), nn(Cas,Gen,Def,Anim,Foc).
```

A.2 Input Files for Generation (input{1,2,3}.pl)

```
% Input for AN > FOC
%%%%%%%%%%%%%%%%%%%%

trace --> [t].
vv --> [entzog].
det(nom,m,def,Anim,Foc) --> [der].
det(akk,n,def,Anim,Foc) --> [das].
det(dat,m,def,Anim,Foc) --> [diesem].
nn(nom,m,Def,anim,foc) --> ['Mann'].
nn(akk,n,Def,anim,nofoc) --> ['Kind'].
nn(akk,n,Def,anim,foc) --> ['KIND'].
nn(dat,m,Def,noanim,nofoc) --> ['Einfluss'].
nn(dat,m,Def,noanim,foc) --> ['EINFLUSS'].

trace --> [t].
%vv --> [entzog].
vv --> [ueberliess].
det(nom,m,def,Anim,Foc) --> [der].
det(akk,n,def,Anim,Foc) --> [das].
det(dat,f,def,Anim,Foc) --> [dieser].
nn(nom,m,Def,anim,foc) --> ['Mann'].
nn(akk,n,Def,anim,nofoc) --> ['Kind'].
nn(akk,n,Def,anim,foc) --> ['KIND'].
nn(dat,f,Def,anim,nofoc) --> ['Frau'].
nn(dat,f,Def,anim,foc) --> ['FRAU'].


trace --> [t].
%vv --> [entzog].
vv --> [ueberliess].
det(nom,m,def,Anim,Foc) --> [der].
det(akk,m,def,Anim,Foc) --> [den].
det(dat,f,def,Anim,Foc) --> [dieser].
nn(nom,m,Def,anim,foc) --> ['Mann'].
nn(akk,m,Def,noanim,nofoc) --> ['Brief'].
nn(akk,m,Def,noanim,foc) --> ['BRIEF'].
nn(dat,f,Def,anim,nofoc) --> ['Frau'].
nn(dat,f,Def,anim,foc) --> ['FRAU'].
```

A.3 Input File for Parsing (parseinput.pl)

```
trace --> [t].
vv --> [entzog].
vv --> [ueberliess].
det(nom,m,def,Anim,Foc) --> [der].
det(akk,m,def,Anim,Foc) --> [den].
det(akk,n,def,Anim,Foc) --> [das].
det(dat,m,def,Anim,Foc) --> [diesem].
det(dat,f,def,Anim,Foc) --> [dieser].
nn(nom,m,Def,anim,foc) --> ['Mann'].
nn(akk,n,Def,anim,nofoc) --> ['Kind'].
nn(akk,n,Def,anim,foc) --> ['KIND'].
nn(akk,m,Def,noanim,nofoc) --> ['Brief'].
nn(akk,m,Def,noanim,foc) --> ['BRIEF'].
nn(dat,m,Def,noanim,nofoc) --> ['Einfluss'].
nn(dat,m,Def,noanim,foc) --> ['EINFLUSS'].
nn(dat,f,Def,anim,nofoc) --> ['Frau'].
nn(dat,f,Def,anim,foc) --> ['FRAU'].
```

A.4 Sample Candidates File for Direct Evaluation (candidates.pl)

```
c([der,'Mann',entzog,diesem,'EINFLUSS',t,das,'Kind',t,t],
  [an,foc,par-move,'*trace','*trace','*trace']).
c([der,'Mann',entzog,diesem,'Einfluss',t,das,'KIND',t,t],
  [an,par-move,'*trace','*trace','*trace']).
c([der,'Mann',entzog,t,das,'KIND',diesem,'Einfluss',t],
  [foc,dat,per,'*trace','*trace']).
c([der,'Mann',entzog,t,das,'Kind',diesem,'EINFLUSS',t],[dat,per,'*trace','*trace']).
```

A.5 Perl Script for Creating Prolog Facts from Text Candidates

```
#!usr/bin/perl

while(<>) {

        s/\),\(/\)\.\nc\(/g;
        s/\[\(/c\(/g;
        s/\)\]/\)\./g;
        print;
}
```

References

Archangeli, Diana, and Terence D. Langendoen, eds. (1997): *Optimality Theory: An Overview.* Oxford, Blackwell.

Barbosa, Pilar, Danny Fox, et al., eds. (1998): *Is the Best Good Enough? Optimality and Competition in Syntax*. Cambridge etc., MIT Press.

Benz, Anton (2001): *Towards a Framework for Bidirectional Optimality Theory in Dynamic Contexts*. Rutgers Optimality Archive[1], ROA-465-0901.

Blutner, Reinhard (2000): *Some Aspects of Optimality in Natural Language Interpretation*. Rutgers Optimality Archive, ROA-389-0400.

Borsley, Robert D. (1997): *Syntax-Theorie: ein zusammengefasster Zugang*. Tübingen, Niemeyer.

Bresnan, Joan (2000): *Optimal Syntax.* In: Dekkers et al. 2000, pp. 334–385.

Büring, Daniel (2000): *Let's Phrase It! Focus, Word Order, and Prosodic Phrasing in German Double Object Constructions.* In: Müller & Sternefeld 2000, pp. 69–105.

Choi, Hye-Won (2001): *Phrase Structure, Information Structure, and Resolution of Mismatch.* In: Sells 2001, pp. 17–63.

Covington, Michael A. (1994): *Natural language processing for Prolog programmers*. Upper Saddle River, Prentice-Hall.

Dekkers, Joost, Frank van der Leeuw, et al., eds. (2000): *Optimality Theory: Phonology, Syntax, and Acquisition.* Oxford etc., Oxford University Press.

Deransart, Pierre, AbdelAli Ed-Dbali, et al. (1996): *Prolog: The Standard*. Berlin etc., Springer.

Eisner, Jason (1997a): *Efficient Generation in Primitive Optimality Theory.* In: Proceedings of the ACL 1997, Madrid, Spain, no pages.

Eisner, Jason (1997b): *What Constraints Should OT Allow?* Rutgers Optimality Archive, ROA-204-0797.

Eisner, Jason (2000): *Directional Constraint Evaluation in Optimality Theory.* In: Proceedings of the COLING 2000, Sarrebrucke, Germany, no pages.

Ellison, T. Mark (1994a): *Constraint, Exceptions and Representations.* In: Proceedings of the ACL 1994, Las Cruces, NM, no pages.

Ellison, T. Mark (1994b): *Phonological Derivation in Optimality Theory*. Rutgers Optimality Archive, ROA-75-0000.

Fanselow, Gisbert, Matthias Schlesewsky, et al., eds. (2005): *Gradience in Grammar*. Oxford etc., Oxford University Press.

Fanselow, Gisbert, Matthias Schlesewsky, et al. (1999): *Optimal Parsing: Syntactic Parsing Preferences and Optimality Theory*. Rutgers Optimality Archive, ROA-367-1299.

Fortmann, Christian, and Martin Forst (2004): *An LFG Grammar Checker for CALL.* In: Proceedings of the InSTIL/ICALL, Venice, 17–19 June, no pages.

Fosler, J. Eric (1996): *On Reversing the Generation Process in Optimality Theory.* In: Proceedings of the ACL 1996, Santa Cruz, CA, no pages.

Frank, Annette, Tracy Holloway King, et al. (2001): *Optimality Theory Style Constraint Ranking in Large-Scale LFG Grammars.* In: Sells 2001, pp. 367–399.

Frank, Robert, and Giorgio Satta (1997): *Optimality Theory and the Generative Complexity of Constraint Violability*. Rutgers Optimality Archive, ROA-228-1197.

Goldwater, Sharon, and Mark Johnson (2003): *Learning OT Constraint Rankings Using a Maximum Entropy Model.* In: Proceedings of the Stockholm Workshop on Variation within Optimality Theory, Stockholm, pp. 111–120.

Grimshaw, Jane (1997): *Projection, Heads, and Optimality*. Rutgers Optimality Archive, ROA-68-0000.

[1] http://roa.rutgers.edu.

Grimshaw, Jane (2001): *Economy of Structure in OT*. Rutgers Optimality Archive, ROA-434-0601.

Guasti, Maria Theresa (2004): *Language Acquisition. The Growth of Grammar*. Cambridge etc., MIT Press.

Haegeman, Liliane (1994): *Introduction to Government & Binding Theory*. Blackwell Textbooks in Linguistics. Oxford etc., Blackwell.

Hammond, Michael (1995): *Syllable Parsing in English and French*. Rutgers Optimality Archive, ROA-58-0000.

Hammond, Michael (1997): *Parsing in OT*. University of Arizona.

Heck, Fabian (2000): *Tiefenoptimierung*. In: Linguistische Berichte, 184 (2000), pp. 441–468.

Heck, Fabian, Gereon Müller, et al. (2002): *On the nature of the input in optimality theory*. In: The Linguistic Review, 19 (2002), pp. 345–376.

Heiberg, Andrea Janine (1999): *Features in Optimality Theory*. PhD Thesis, University of Arizona.

Holloway King, Tracy, Stefanie Dipper, et al. (2000): *Ambiguity Management in Grammar Writing*. In: Proceedings of the ESSLLI'2000 Workshop on Linguistic Theory and Grammar Implementation, Tübingen, August 14–18, no pages.

Jäger, Gerhard (2000): *Some Notes on the Formal Properties of Bidirectional Optimality Theory*. Rutgers Optimality Archive, ROA-414-0900.

Jäger, Gerhard (2003a): *evolOT – Software for simulating language evolution using Stochastic Optimality Theory.*

Jäger, Gerhard (2003b): *Learning constraint sub-hierarchies. The Bidirectional Gradual Learning Algorithm.* University of Potsdam, Germany.

Jäger, Gerhard (2004): *Maximum Entropy Models and Stochastic Optimality Theory*. Rutgers Optimality Archive, ROA-625-1003.

Kager, René (1999): *Optimality Theory*. Cambridge Textbooks in Linguistics. Cambridge etc., Cambridge University Press.

Karttunen, Lauri (1998): *The Proper Treatment of Optimality in Computational Phonology*. Rutgers Optimality Archive, ROA-258-0498.

Keller, Frank (2000): *Evaluating Competition-based Models of Word Order*. In: Proceedings of the 22th Annual Conference of the Cognitive Science Society, Manawah, NJ, pp. 747–752.

Keller, Frank, and Ash Asudeh (2001): *Probabilistic Learning Algorithms and Optimality Theory*. Rutgers Optimality Archive, ROA-675-0804.

Kuhn, Jonas (1999): *Resolving some apparent formal problems of OT Syntax*. In: Proceedings of the NELS 30, Rutgers, NJ, no pages.

Kuhn, Jonas (2000): *Processing Optimality-theoretic Syntax by Interleaved Chart Parsing and Generation*. In: Proceedings of the ACL 2000, Hongkong, China, pp. 360–367.

Kuhn, Jonas (2001a): *Computational Optimality-theoretic Syntax: a Chart-based Approach to Parsing and Generation*. In: Rohrer et al. 2001a, pp. 353–387.

Kuhn, Jonas (2001b): *Generation and Parsing in Optimality Theoretic Syntax: Issues in the Formalization of OT-LFG*. In: Sells 2001b, pp. 313–367.

Kuhn, Jonas (2001c): *Formal and Computational Aspects of Optimality-theoretic Syntax*. PhD Thesis, University of Stuttgart, Germany.

Kuhn, Jonas (2002): *OT Syntax: Decidability of Generation-based Optimization*. In: Proceedings of the ACL 2002, Philadelphia, pp. 48–55.

Kuhn, Jonas (2003): *Optimality-Theoretic Syntax – A Declarative Approach*. Butt, Miriam, Andreas Kathol, et al., eds.: Studies in Constraint-Based Lexicalism. Stanford, CSLI Publications.

Kuhn, Jonas, and Christian Rohrer (1997): *Approaching the ambiguity in real-life sentences: the application of an Optimality Theory-inspired constraint ranking in a large-scale LFG grammar*. 6. Fachtagung der Sektion Computerlinguistik DGfS-CL, Heidelberg, October 1997.

Kuhn, Jonas, Judith Eckle-Kohler, et al. (1998): *Lexicon Acquisition with and for Symbolic NLP-Systems – a Bootstrapping Approach.* In: Proceedings of the First International Conference on Language Resources and Evaluation (LREC98), Granada, pp. 89–94.

Lee, Hanjung (2001): *Markedness and Word Order Freezing.* In: Sells 2001, pp. 63–129.

Legendre, Géraldine, Jane Grimshaw, et al., eds. (2001): *Optimality-Theoretic Syntax*. Cambridge etc., MIT Press.

Legendre, Géraldine, Paul Smolensky, et al. (1998): *When is less more? Faithfulness and Minimal Links in WH-Chains.* In: Barbosa et al. 1998, pp.

Maslova, Elena (2004): *Stochastic OT as a model of constraint interaction*. Rutgers Optimality Archive, ROA-694-1104.

Morawietz, Frank, and Tom Cornell (1997): *Representing Constraints with Automata.* In: Proceedings of the ACL 1997, Madrid, Spain, no pages.

Müller, Gereon (1998): *German Word Order and Optimality Theory.* Arbeitspapiere des Sonderforschungsbereichs 340, University of Dusseldorf, Germany.

Müller, Gereon (1999): *Optimality, Markedness, and Word Order in German*. In: Linguistics, 37 (1999), pp. 777–818.

Müller, Gereon (2000): *Elemente der optimalitätstheoretischen Syntax*. Tübingen, Stauffenburg.

Müller, Gereon (2001): *Order Preservation, Parallel Movement, and the Emergence of the Unmarked.* In: Legendre et al. 2001, pp. 279–315.

Müller, Gereon (2002): *Constraints in Syntax (6): Optimality-Theoretic Syntax.* Dusseldorf LSA/DGfS Summerschool 2002.

Müller, Gereon, and Wolfgang Sternefeld, eds. (2000): *Competition in Syntax*. Berlin etc., Mouton de Gruyter.

Pinker, Steven (1997): *Language Learnability and Language Development.* Cambridge etc., Harvard University Press.

Prince, Alan, and Paul Smolensky (1993): *Optimality Theory. Constraint Interaction in Generative Grammar*. Rutgers Optimality Archive, ROA-537-0802.

Pulleyblank, Douglas, and William J. Turkel (1998): *The Logical Problem of Language Acquisition on Optimality Theory.* In: Barbosa et al. 1998, pp. 399–421.

Pulleyblank, Douglas, and William J. Turkel (2000): *Learning Phonology: Genetic Algorithms and Yoruba Tongue-Root Harmony*. In: Dekkers et al. 2000, pp. 554–591.

Raymond, William, and Apollo Hogan (1994): *A Users Guide to the Optimality Interpreter: A Software Tool for Optimality Theoretic Analysis*. Rutgers Optimality Archive, ROA-130-0000.

Richter, Michael (2002): *Der Fokus im Mittelfeld*. In: Linguistik online, 12 (2002), pp. 71–85.

Rohrer, Christian, Antje Rossdeutscher, et al., eds. (2001): *Linguistic Form and its Computation*. Stanford, CSLI Publications.

Sells, Peter, ed., (2001): *Formal and Empirical Issues in Optimality Theoretic Syntax*. Stanford, CSLI Publications.

Tesar, Bruce (1994): *Parsing in Optimality Theory: A Dynamic Programming Approach.* University of Colorado.

Tesar, Bruce (1995a): *Computing Optimal Forms in Optimality Theory: Basic Syllabification*. Rutgers Optimality Archive, ROA-52-0295.

Tesar, Bruce (1995b): *Computational Optimality Theory.* PhD Thesis, University of Boulder, Colorado.

Tesar, Bruce (1996): *Computing Optimal Descriptions for Optimality Theory Grammars with Context-Free Position Structures.* In: Proceedings of the ACL 1996, Santa Cruz, CA, pp. 101–107.

Tesar, Bruce (2000): *On the Roles of Optimality and Strict Domination in Language Learning.* In: Dekkers et al. 2000, pp. 592–620.

Tesar, Bruce B. (1998): *Error-Driven Learning in Optimality Theory via the Efficient Computation of Optimal Forms.* In: Barbosa et al. 1998, pp. 421–437.

Tesar, Bruce, and Paul Smolensky (1993): *The Learnability of Optimality Theory: An Algorithm and Some Basic Complexity Results*. Rutgers Optimality Archive, ROA-2-1093.

Tesar, Bruce, and Paul Smolensky (1996): *Learnability in Optimality Theory*. Rutgers Optimality Archive, ROA-156-1196.

Trommer, Jochen (1999): *Generation and Parsing in OT-based Morphology.* In: Proceedings of the Formal Grammar 1999, Utrecht, The Netherlands, no pages.

Vogel, Ralf (2004): *Remarks on the Architecture of OT Syntax Grammars*. Rutgers Optimality Archive, ROA-660-0504.

Vogel, Ralf (2005): *Degraded Acceptability and Markedness in Syntax, and the Stochastic Interpretation of Optimality Theory.* In: Fanselow et al. 2005, no pages.

Walther, Markus (1996): *OT SIMPLE – A construction-kit approach to Optimality Theory implementation.* Arbeitspapiere des Sonderforschungsbereichs 282, University of Dusseldorf, Germany.

Wartena, Christian (2000): *A Note on the Complexity of Optimality Systems.* Rutgers Optimality Archive, ROA-385-0300.

Wilson, Colin (2001): *Bidirectional Optimization and the Theory of Anaphora.* In: Legendre et al. 2001, pp. 465–509.

Curriculum Vitae

Johann Fichtner

27. 08. 1979	geboren in Summit, New Jersey, USA
1986 – 1992	Besuch der Primarschule Aesch (Kanton Zürich)
1992 – 1999	Besuch der Kantonsschule Hohe Promenade, Zürich; Matur Typ A
1999 – 2006	Studium der Germanistik, Computerlinguistik und Musikwissenschaft an der Universität Zürich