Constructive optimality theoretic syntax

Zeevat, H.

Publication date
2008

Published in
Proceedings of the 5th International Workshop on Constraints and Language Processing (CSLP 2008)

Constructive Optimality Theoretic Syntax

Henk Zeevat

ILLC, Nieuwe Doelenstraat 15, 1012 CP Amsterdam
henk.zeevat@uva.nl

Abstract. This paper documents efforts at developing a constructive optimality theory. Several years ago, two 3rd year AI-students, Chris Regenboog and Jacco van Willegen, came up with an idea to implement optimality theory, in a special case —a system partially describing Dutch word order—. Me and some students have been trying to work out their ideas further in the period since, but many issues still remain open. This paper reports on a reconstruction of Bresnan's Optimal Syntax and a new integrated approach to word order, in three Germanic languages: Dutch, English and German, as a direct illustration of the concept. The idea is the following: interpret every constraint as a procedure that tries to add more structure to an underspecified specification and call the procedures in the order of the strength of the constraints they implement. The main consequence is that the notion of error disappears altogether from optimality theory. Errors are failed attempts to overwrite structure that is already filled in. The process is monotone increase of information and fully deterministic. There should be enough constraints to make sure that the process delivers a fully instantiated ground structure. Two other things that disappear from sight are GEN and various economy constraints: what can be constructed gives an upper bound on what should be in GEN, economy violations are just not generated.

The idea[1] also provides a serious alternative in thinking about the cognitive interpretation of optimality theory. The neural nets from which optimality theory emerged give rise to the idea of harmonic grammar in which the violations of constraints determine an integrated total violation value with respect to which the optimal candidate should win. The empirical discovery that gave rise to optimality theory as we know it is however the discovery that for the purpose of phonological description the possibility of an integrated violation value for several constraints in combination can be ignored: the strongest constraint always takes the decision. While it is possible to formalise optimality theory in harmonic grammar and to implement it in the corresponding nets[9], optimality theory does not arise naturally out of this setup.

The alternative model for OT is to think of a single object that is being built by a group of workers each with a particular task with respect to the structure. They all try to add details to it and conflicts are solved by a strict

[1] Some work on phonology has been omitted from this paper for reasons of space. Exploratory implementations have been produced in C, Prolog and Perl.

power hierarchy between them: the more powerful worker always gets his way, but even the most powerful worker is specialised and only decides some of the details, so that the activities of the lower ranked workers also show up in the finished structure.

A worker can be thought of as some simple map from an input to an output that is an informational increment of the input. While it is possible to conceive of this as a parallel process, in this paper, a serial composition approach will be adopted.

1 Optimality Theory

In optimality theory, structures are characterised as the best candidate for given input given a linearly ordered set of constraints $C_0 \dots C_k$. The set of candidates and the input give an optimisation problem, the constraints determine how it should be solved. In linguistics, the task of selecting a verbal message for a given semantics, selecting a pronunciation for a form, and the task of finding an interpretation for a pronunciation can all be usefully considered to be optimisation problems. But there is no need for a limitation to linguistics.

In standard OT, by definition, $output_1$ is better than $output_2$ iff they are equally good or bad with respect to the first i constraints but $output_2$ violates C_{i+1} more often than $output_1$. An optimisation problem is an input, a constraint system and a set of possible outputs. A solution to the optimisation problem is a candidate that is best: any other candidate is worse or equally good.

The famous applications are in linguistics, particularly in production phonology (mapping an abstract phonological structure to a concrete one) and in production syntax (mapping semantics to a string of words). Less famous are applications to the semantics and pragmatics of natural language.

The application to phonology and syntax also comes with an important typological interpretation: the constraints are the same in all languages and typological variation is just changing the ordering of the contraints.

This offers a way of arguing for or against particular constraints: they should lead to correct typological predictions. But it also imposes a restriction on cognitively plausible implementations: constraint systems and their implementation should allow reordering. The typological interpretation turns OT phonology and syntax into a formal reconstruction of Jakobson's markedness theory [5].

The abstract model of optimality theory allows a direct computational interpretation if one can keep the number of candidates finite. But this would still be a bad strategy, comparable to generate-and-test in classical parsing. The neural basis has also been taken as a starting point and has led to experimental implementations of neural net-based phonology (Smolensky, Oren Schwarz). This is promising, but still experimental.

[4] and [6] have proposed an almost correct implementation for OT phonology using lenient composition over finite state transducers corresponding to the constraints. This is only an approximation since it requires that the number of errors is bounded by a maximum and the OT formalism does not allow such a

maximum number. In addition, this method is limited to phonology. [2] offers a less restricted model of phonology.

[7] proposes OT as a specification mechanism for LFG based processing. This paper agrees more with this strategy than is maybe apparent, but nonetheless as an implementation strategy, it does not qualify as being purely OT. In sum, direct implementations of OT syntax that can be useful in NL processing are missing. This paper hopes to make a contribution to this area, but the main idea is also applicable outside syntax, to phonology and maybe to semantics.

The implementation proposed in this paper can be given by a metaphor based on unification grammar. Inputs are underspecified feature structures only containing the input. Outputs are fully instantiated ground feature structures. The constraints are also feature structures. Outputs are constructed by default unifications of the constraints applied in the order of their strength to the input structure.

The effect is that constraint violations correspond to vacuous default unification where the partially specified feature structure would get inconsistent information. As a result of the input and the stronger constraints that apply first errors are the cases where information in the constraint conflicts with information in the feature structure.

Perhaps a literal implementation of this metaphor is possible in phonology, by coding the constraints as recursive feature structures.

Less metaphorically, constraints are procedures that add information to underspecified structures if the new information is consistent with the given underspecified structure. Inputs are underspecified structures only containing the input specification. Outputs are complete structures. Outputs are obtained by applying the constraint procedures in order of their strength to the input structure.

An abstract implementation strategy can be based on constraint logic programming. The structure is then a consistent theory satisfied by a class of finite models. Constraints are universal statements and can be represented by the set of their instances (with the constants taken from the objects of the theory so far). Default application is then a question of adding all those instances that are consistent with the theory to it. The output structure is a or the minimal model of the resulting theory. This strategy is be always available, but it is not necessarily efficient.[2]

In the abstract case, the input is a theory, GEN is the set of all its models and optimal elements of GEN are the minimal ground models of the theory after optimisation.

Efficiency can be increased if the theory or class of models can be efficiently represented by a finite structure that can be updated by the constraints. Let's call this "concrete constructive". The rest of the paper presents two instances of concrete constructive OT syntax.

[2] Let T be a finitely witnessed theory and C a universal statement. The incrementation of T by C can be given as $T(C) = T \cup \{C(a) : a \text{ is a witness of } T \text{ and } T \not\models \neg C(a)\}$

GEN is then the set of all ground instances of the input, and the winners those elements of GEN that are ground instances of the optimised input.

2 Bresnan's Optimal Syntax

This following is reconstruction[3] of [1]. The constraints appear in order of strength.

MAX(SPEC,PRED,TENSE,MODE, ...)

specifiers, predicates, tense and modal values are expressed by lexical and morphological means.

HEADS

f-structure heads are c-structure heads

LP COMPLEMENTS

lexical phrase complements are f-structure complements

FP COMPLEMENTS

functional phrase complements are not f-structure complements

PROM

the functional hierarchy of arguments corresponds to c-structure order

CC

the f-structure is complete and coherent

OP-SPEC

the operator appears as the specifier of C

* LEX-F

no lexical heads in functional projections

OB-HEAD

every category has an extended head

AGR

the subject and its predicate agrees

FULL INTERPRETATION

the output f-structure does not add to the content of the input structure

STAY

categories dominate their extended heads

NEG-TO-I

negation adjoins to I

LEX

structural inventory items are phonologically realised

Inputs are abstract f-structures, candidates associations of a c-structure with an f-structure.

Example:

$[_{cp}$*what do* $[_{ip}$*they* $[_{vp}$*read*$]]]$

$[_{cp}$*what read* $[_{ip}$*they*$[_{vp}]]]$ (*LEX-F)

$[_{ip}$*they*$[_{vp}$*read what*$]]$ (OP-SPEC)

[3] Bresnan's presentation is not formal enough for the purposes of this paper, so there are quite a number of decisions hidden both in my version of the constraint system and in the procedural translation. [1] provides empirical and theoretical motivation for the approach

$[_{ip}$*they do* $[_{vp}$*readwhat*$]]]$ (OP-SPEC)
$[_{cp}$*what* $[_{ip}$*they* $[_{vp}$*read*$]]]$ (OB-HEAD(2))
$[_{cp}$*what* $[_{ip}$*they do* $[_{vp}$*read*$]]]$ (OB-HEAD)

The following assigns a procedural meaning to the constraints, in some cases to several constraints at once.

The underspecified data structures are partially specified f-structures with some extra features and lexical items assigned to the nodes. Underspecification show up in missing position features and lexical annotations. Other underspecification are abstract governable functions (gf1, gf2 etc) that can be further specified to be subj, obj, obj1 etc and the possibility of inserting "dummy" features. The non-standard position features express where the node has to be realised in c-structure or in the surface string and the "dummy" feature allows the insertion of dummy elements, like auxiliary "do".

Procedural meaning

The starting point are abstract f-structures as in [1] to position annotated and lexicalised f-structures. (1) should be read as: the root's subject's predicate is *tom* (a semantic predicate), annotated with lexical item "Tom" (a string of phonemes) and the position feature SCP (the specifier of CP).

(1) subj:pred:tom+Tom+SCP

The abstract feature structure that is given as the input is also a representation of the set of all candidates: these are given by any way of assigning position features and lexical and morphological annotations to nodes in the f-structure and adding "dummy branches".

1. CC combined with MAX(SPEC,PRED,MODE,TENSE,...)

This is the lexicalisation step: the PRED from the input structure needs to be realised as a lexical item which governs the governable functions given in the input. The instantition of PRED also assigns concrete grammatical functions and case features to the GFs in the input structure and assigns morphology or lexical items to the features mentioned in MAX(SPEC,PRED,MODE,TENSE,...). This step can be seen as the definition of "legal lexicalised f-structure corresponding to the abstract input".

2. OP-SPEC

This constraint assigns the syntactic position feature SCP (specifier of CP) to the operator (long distance dependencies are captured in the f-structure by having a discourse function DF in the outermost f-structure unified with the operator. The value of DF is the operator.)

3. A combination of HEADS and *LEX-F

Together they force lexical heads of category X to occupy the structural position HXP (head of an XP). Without *LEX-F, the position can be classified as an HYP with Y a functional projection of category Y. Especially HVP could also be HIP or HCP. The combination instantiates the syntactic position parameter of an V, N, P or A to be HVP, HNP, HPP or HAP respectively.

4. OB-HEAD

a. It creates heads for CP (if needed: CP must be non-empty to have a HCP) or for IP (if CP does not have a head of category I and I is needed because of a SIP or IAdjunct). "Do" is the dummy HCP or HIP and inserted when there is no lexical element of category I in the f-structure.

b. auxiliaries are moved to CP: assign HCP to the auxiliary if there is a SCP but no C.

5. FP COMPLEMENTS

If TENSE, MODE or SPEC is realised by auxiliaries, complementisers or determiners these will get syntactic position features HIP or HCP (auxiliaries can be both the head of C and the head of I) and HDP.

6. LP COMPLEMENTS + PROM

They sort the LP complements in the order given by the functional hierarchy and can be used to assign syntactic position features XARG1 to XARGn to these complements.

7. AGR

1. assign SIP (specifier of IP) to the subject if there is an IP, SSU (sentential subject) otherwise.

2. assign agreement features and morphology to the predicate of the subject (the HIP or HCP, the HVP if there is no IP or CP.

8. NEG-TO-I

assign the IADJUNCT label to NEG

FULL INTERPRETATION has no good interpretation in this setup, STAY does not need to be implemented because it is emergent: the procedure does not produce unnecessary violations of it.

The annotated f-structures determine a string of words. Take the lexical word and morphology annotations and order them (cyclically) by their position annotations in the order:
SCP HCP SIP HIP Iadjunct SSU HVP VPARG1 VPARG2 VPARG3.

Example:

q=gf2
gf1:pred:pro
gf1:agr:pl
gf2:pred:pro
gf2:agr:nsg
pred:read(gf1,gf2)

CC + MAX(SPEC,PRED,MODE,TENSE,...)

q=obj
subj:pred:pro + they
subj:agr:pl
obj:pred:pro + what
obj:agr:nsg
pred:read(subj,obj)+ read

OP-SPEC

q=obj
subj:pred:pro + they
subj:agr:pl
obj:pred:pro + what+SCP

obj:agr:nsg
pred:read(subj,obj)+ read
HEADS + *LEX-F
q=obj
subj:pred:pro + they
subj:agr:pl
obj:pred:pro + what+SCP
obj:agr:nsg
pred:read(subj,obj)+ read+HVP
OB-HEAD q=obj
subj:pred:pro + they
subj:agr:pl
dummy:do+HCP
obj:pred:pro + what+SCP
obj:agr:nsg
pred:read(subj,obj)+ read+HVP
LP COMPLEMENTS + PROM
q=obj
subj:pred:pro + they +VARG1
subj:agr:pl
dummy:do+HCP
obj:pred:pro + what+SCP
obj:agr:nsg
pred:read(subj,obj)+ read+HVP
AGR
q=obj subj:pred:pro + they +VARG1
subj:agr:pl
dummy:do-0+HCP
obj:pred:pro + what+SCP
obj:agr:nsg
pred:read(subj,obj)+ read+HVP The last f-structure can be read out as the c-structure:

$[_{cp}what\ do\ [_{ip}they\ [_{vp}read]]]$

3 Dutch OT Syntax for Word Order

The constraint system of Bresnan does not seem to be applicable to Dutch or German, except for the formation of questions: German and Dutch allow lexical elements to be the heads of functional categories. The agenda in this section and the next two ones will be to develop some ideas about Dutch, then German and finally English. It is not the aim to be complete or even correct: its aim is to demonstrate the feasibility of an interesting and typologically valid account within concrete constructive OT. In the treatment, the problems in arriving at suitable f-structures from semantic representations are ignored as well as morphological issues. These aspects could be integrated, possibly along the lines of the reconstruction of [1] in the last section.

Constraints (ordered from strong to weak)

CLOSE(REL,SQ)
relative clauses and indirect questions are closed off for long distance movement

ONE = WH
the element ONE is the WH-phrase (finite verbs in main yes-no questions are also wh-elements in this view)

CLOSE(DP)
DPs are closed off for long distance movement

ONE = CT
the element ONE is the contrastive topic

CLOSE(SCOMP)
S structures are closed off for long distance movement

ONE = SUBJ
the element ONE is the subject

ONE < X
ONE comes first

V[FIN,MAIN] < X
the main finite verb comes first

SUBJ < X
the subject comes first

TOP(PP) < X
topical PPs come first

IO < X
the indirect object comes first

OBJ < X
the clause's object comes first

S > X
SCOMPS and Relative clauses come last

HV < X (headverbs come last) pending with VCOMP > X (VCOMPS come last)

input:
hij^{su} *leert* Jan^{obj} $[zwemmen]^{vcomp}$

	11	v1	su1	ob1	hv1	vc>
zwemmen Jan leert hij	***	**	***	*		***
zwemmen Jan hij leert	**	***	**	*		***
zwemmen leert Jan hij	***	*	*	**		***
zwemmen leert hij Jan	**	*	***	***		***
zwemmen hij Jan leert	*	***	**	**		***
zwemmen hij leert Jan	*	**	***	***		***
Jan zwemmen leert hij	***	**	***		*	**
Jan zwemmen hij leert	**	***	**		*	**
Jan leert zwemmen hij	***	*	***		**	*
Jan leert hij zwemmen	**	*	**		***	
Jan hij zwemmen leert	*	***	*		**	*
Jan hij leert zwemmen	*	**	*		***	
leert Jan zwemmen hij	***		***	*	**	*
leert Jan hij zwemmen	**		**	*	***	
leert zwemmen Jan hij	***		***	**	*	**
leert zwemmen Jan hij	***		***	**	*	**
leert hij Jan zwemmen	*		*	**	***	
leert hij zwemmen Jan	*		*	***	**	*
⇒hij leert Jan zwemmen		*		**	***	
hij leert zwemmen Jan		*		***	**	*
hij zwemmen leert Jan		**		***	*	**
hij zwemmen Jan leert		***		**	*	**
hij Jan zwemmen leert		***		*	**	*
hij Jan leert zwemmen		**		*	***	*

Procedural interpretation

1. Initialise by putting f-structure nodes between open and closing brackets $<$ and $>$. (These are really sets, linear order emerges by taking elements out of these sets and is coded by the string).

2. Apply the constraints topdown to all constituent lists $< \ldots >$ of the input, going freely into closed constituent lists. Put αs immediately before left bracket (after the left bracket) for ordering constraints $\alpha < X$ and $\alpha > X$. α's can be dragged out from anywhere except from a $close(\ldots)$ structure.

3. CLOSE(α): replace anything $Y^{\cdot\cdot\alpha\cdot\cdot}$ by close($Y^{\cdot\cdot\alpha\cdot\cdot}$)

4. ONE = X: assign a feature ONE to X if ONE has not been assigned yet

Example

$< hij^{subj}\ heeft^{main}\ Maria^{obj}\ < laten\ Jan^{obj}\ < leren\ zwemmen^{vcomp} >^{vcomp} >^{vcomp} >$

ONE=SUBJ

$< hij^{subj,one}\ heeft^{main}\ Maria^{obj}\ < laten\ Jan^{obj}\ < leren\ zwemmen^{vcomp} >^{vcomp} >^{vcomp} >$

ONE < X

$hij^{subj,one}\ < heeft^{main}\ Maria^{obj}\ < laten\ Jan^{obj}\ < leren\ zwemmen^{vcomp} >^{vcomp} >^{vcomp} >$

MAIN < X

$hij^{subj,one}\ heeft^{main}\ < Maria^{obj}\ < laten\ Jan^{obj}\ < leren\ zwemmen^{vcomp} >^{vcomp} >^{vcomp} >$

$hij^{subj,one}\ heeft^{main}\ < Maria^{obj}\ < laten\ Jan^{obj}\ < leren\ zwemmen^{vcomp vcomp vcomp}$

OBJ < X

$hij^{subj,one}\ heeft^{main}\ Maria^{obj}\ < Jan^{obj}\ < laten\ < leren\ zwemmen^{vcomp} >^{vcomp} >^{vcomp} >$

VCOMP < X

$hij^{subj,one}\ heeft^{main}\ Maria^{obj}\ <>< Jan^{obj}\ < laten\ < leren\ zwemmen^{vcomp} >^{vcomp} >$

$hij^{subj,one} heeft^{main}\ Maria^{obj}\ <><Jan^{obj}>^{vcomp} < laten\ < leren\ zwemmen^{vcomp}>^{vcomp}>$
$hij^{subj,one} heeft^{main}\ Maria^{obj}\ <><Jan^{obj}>^{vcomp} < laten >^{vcomp} < lerenzwemmen^{vcomp}>$
$hij^{subj,one} heeft^{main}\ Maria^{obj}\ <><Jan^{obj}>^{vcomp} < laten >^{vcomp} < leren >^{vcomp}$
zwemmen Omitting brackets and features this gives: *hij heeft Maria Jan laten leren zwemmen.*

Calling HV > X instead on:

$hij^{subj,one} heeft^{main}\ Maria^{obj}\ << Jan^{obj}\ < laten\ < leren\ zwemmen^{vcomp}>^{vcomp}>^{vcomp}>$

will give

HV > X

$hij^{subj,one} heeft^{main}\ Maria^{obj} << Jan^{obj} << leren\ zwemmen^{vcomp}>^{vcomp}>^{vcomp}>$
laten

$hij^{subj,one}\ heeft^{main}\ Maria^{obj}\ << Jan^{obj}\ << zwemmen^{vcomp}>^{vcomp}\ leren>^{vcomp}>$
laten i.e. the alternative Dutch order (the only German one):
hij heeft Maria Jan zwemmen leren laten.

4 Provisional German

The following gives a rough version of German. German allows slightly more long distance dependencies and considerably more freedom in word order. After the main verb, priorities are better given by a constraint PROM which allows fronting any constituent that more prominent in the dimensions of grammatical obliqueness, animacy, topicality, referentiality. PROM would be doing the work of the Dutch constraints: SUBJ < X, TOP < X, IO < X or OBJ < X. German has also chosen against the cross-serial dependency order in the verbal complex.

CLOSE(REL,SQ)
ONE = WH
ONE = CT
CLOSE(DP, SCOMP)
ONE = SUBJ
ONE < X
V[FIN,MAIN] < X
PROM
S > X
HV < X
VCOMP > X

5 English

Is it possible to see English, Dutch and German as typological variants of each other? It seems to work, if another approach is chosen for the auxiliary verb syntax than the one adopted by Bresnan. Remember that Dutch and German put finite main verbs in the second position. And that Dutch and German put subjects first. In English, the restriction to main clauses has disappeared: it is a general property of finite verbs that they come first. The hypothesis is that the Dutch/German subject constraint and verb second in English (finite verbs come

first after the occupants of ONE) have become merged: they are ranked equally, but instead of choosing the verb or the subject to come first they must both be obeyed[4] This goes together with free insertion of finite dummy *do*.

This gives the complex constraint (2):

(2) V[FIN] < X >=< SUBJ < X[-AUX]

(2) can be implemented by the following procedure.

a. Put the finite verb out to the left if there is no subject in the constituent list.

b. Put finite aux out and then the subject. If there is no aux, put a version of *do*.

With this complex constraint, the English system is quite close to Dutch and German. The German PROM < X has to be split up even further than for Dutch.

CLOSE(REL,SQ)
ONE = WH
ONE = CT
CLOSE(DP,PP,SCOMP.VCOMP)
ONE = SUBJ
ONE < X
V[FIN] < X >=< SUBJ < X[-AUX]
V < X
IO < X
OBJ < X
PP < X
S > X
VCOMP > X

6 Analysis

The above gives a simple and fast model of generation and it should be possible to use it in a model of parsing. The idea would be to use a statistical dependency parser like [8]. These are able to generate the kind of analyses that can be used as input by the generator defined above[5]. There are two things the combination can do: one is a check on parsing. If the generator cannot generate the parse, use the next parse until one is found that can be generated. This increases the quality of the parsing and allows the parser to be imperfect. Second, it is the ideal setting for improving the generator.

[4] The constructive approach forces a distinction between the situation that two constraints are ranked equal and a choice has to be made for the first constraint to apply and the merging of two constraints: they together determine a procedure which is called as a unit.

[5] This is the reverse to what is proposed in [3] where LFG parsing is improved by a kind of interpretational OT

But there is a third application. An OT description of the kind that I have been considering above cannot deal with word order freezing (presumably the reason why English and Dutch syntax have cut up PROM into a small more specific system of constraints. All the older versions of Germanic are more like German, including Anglosaxon. A direct precursor of Dutch is not known.

What sets English apart from German is the almost complete disappearance of the case system which in German prevents accidents with the liberal rules of word order in the middle field of German. It prevents accidents in German only to some extent since the German case system is not nearly as robust as the Latin or the Russian case system: many times the case morphology does not tell subjects and objects apart. In those situations, the thematic dimension of PROM becomes all important and the dominant order is canonical: subjects precede indirect and direct objects. (Freezing is only one of many phenomena of this kind.)

This can be solved by looking at the most probable parse: if it is not the generator's input, other generations and variations of the input must be considered until the parser can return the input. The consideration of other generations will deal with word order freezing in the middle field. For similar freezing effects around the first position, a possibility of switching off the CT-feature is an option. This pragmatic feature can be marked by word order, but it can also be clear from the context or marked by intonation. An option could be to have a variant UCT of the feature CT in the input which does not force the constituent to become ONE.

So these are the claims of this paper:

1. Concrete and constructive OT is possible in syntax. It is much less different from LFG or HPSG than one would think. It is essentially generation oriented and this restriction seems hard to remove.

2. Unification grammar has always been concrete and this never was an obstacle to description. At some level of abstraction, concrete constructive OT is constraint-based unification grammar with unification replaced by default unification.

3. Pending constraints cannot be reduced to considering both orders in constructive OT.

And there is one more claim that needs to be proved in future work:

Constructive OT syntax is a learnable system and can be learnt with the help of a state of the art probabilistic parser. This parser can also help in acquiring a capacity to monitor generation for understanding and so reach proper syntactic correctness (including freezing) and improve the quality of its output for a human user. Inversely, a fully developed generator can increase the correctness of the probabilistic parser. If this works, OT syntax can have technological impact after all.

References

1. Joan Bresnan. Optimal syntax. In J. Dekkers, F. van der Leeuw, and J. van de Weijer, editors, *Optimality Theory: Phonology, Syntax, and Acquisition*, pages 334–

385. Oxford University Press, Oxford, 2000.
2. Jason Eisner. Directional constraint evaluation in optimality theory. In *COLING 2000*, pages 257–263. ACL, 2000.
3. Martin Forst, Jonas Kuhn, and Christian Rohrer. Corpus-based learning of ot constraint rankings for large-scale lfg grammars. In *Proceedings of the LFG05 Conference*. CoNLL, Bergen, Norway, 2005.
4. Robert Frank and Giorgio Satta. Optimality theory and the generative complexity of constraint violability. computational linguistics. *Computational Linguistics*, 24(3):307–315, 1998.
5. R.O. Jacobson. Morphological observations on Slavic declension (the structure of Russian case forms). In Linda R. Waugh and Morris Halle, editors, *Roman Jakobson. Russian and Slavic grammar: Studies 1931-1981*, pages 105–133. Mouton de Gruyter, Berlin, 1958/1984.
6. Lauri Karttunen. The proper treatment of optimality in computational phonology. In Lauri Karttunen, editor, *FSMNLP'98: International Workshop on Finite State Methods in Natural Language Processing*, pages 1–12. Association for Computational Linguistics, Somerset, New Jersey, 1998.
7. Jonas Kuhn. *Optimality-Theoretic Syntax: A Declarative Approach*. CSLI Publications, Stanford, 2003.
8. J. Nivre, J. Hall, and J. Nilsson. Memory-based dependency parsing. In H. T. Ng and E. Riloff, editors, *Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL)*, pages 49–56. CoNLL, Boston, Massachusetts, 2004.
9. Paul Smolensky and Geraldine Legendre. *The Harmonic Mind From Neural Computation to Optimality-Theoretic Grammar*. MIT Press, Cambrdige (MA), 2006.