Proceedings

of

Formal Grammar 1999

Geert-Jan M. Kruijff and Richard T. Oehrle
(editors)

Chapter 1

Generation and Parsing in OT-based Morphology

Jochen Trommer

Abstract.

In this paper I present a formalized approach to morphology based on Optimality Theory (OT, [PS93]). The basic tools are extended versions of finite state machines using feature structures as alphabet. According to constraint type, constraints are evaluated in the form of finite state transducers as proposed by [Ell94] or finite state automata ([Kar98]). Departing from formalizations of OT-phonology constraint machines are created and applied dynamically, minimizing both machine size and the number of optimization steps in derivations. A limited set of constraint types and the use of ranking invariant properties of the system allow a simple parsing procedure determining a small number of possible inputs which are then tested against by generation.

1.1 Introduction

Since the development of Optimality Theory (OT, [PS93]) an increasing number of researchers tries to apply its basic principles to morphological phenomena (e.g. [Noy93], [Gri97]). I will concern myself here mainly with what I think is the core problem in the cited literature: the conflict between 'parse' and complexity constraints. Parse constraints (section 1.7) demand that word forms are maximally transparent in the sense that morphosyntactic features are expressed by morphemes indicating their featural content. Complexity (or 'blocking') constraints (section 1.6) limit the complexity of words, e.g. by restricting the cooccurrence possibilities of morphemes. 1.3 introduces some Georgian data which serve as illustration for both constraint types. The basic algorithm for generating word forms is described in sections 1.2, 1.4 and 1.5. The parsing algorithm is given in 1.8. A format for constraints on the linear ordering of morphemes is developed in 1.9. Further constraint types are discussed in 1.10. Finally I address some open questions (1.11).

Proceedings of Formal Grammar 1999
Geert-Jan M. Kruijff & Richard T. Oehrle (editors).

1.2 Finite State Implementations of OT

Finite-state phonology has developed two main techniques for the implementation of OT-constraints, both of which are used in the current formalization of OT-morphology.

[Ell94] realizes OT-constraints by finite state transducers (*FSTs*) which map string symbols onto sequences of '0's and '1's where '1' stands for a constraint violation. These transducers are then applied by the way of left restriction([FS98]) to candidate sets in the form of finite state automata (*FSAs*). An optimization algorithm prunes all paths from the transducer that have more '1's than the optimal paths with the minimum of violation marks. The input language of the transducer then represents the set of optimal candidates.

A second approach using finite-state machines (*FSMs*) without explicit violation marks is introduced in [Kar98]. It's used here in a slightly modified form, which I will call 'conditioned intersection': constraint automata are intersected with the actual candidate automaton. When intersection leads to an empty automaton the actual candidate automaton is retained, otherwise the intersection automaton represents the set of optimal candidates.

In both formalizations for a given candidate automaton $Cand$ and a constraint $Cons$ a new automaton $\mathbf{Opt}(Cons, Cand)$ is generated containing only the optimal candidates w.r.t. $Cons$. The automaton $\mathbf{Opt}(R, Cand)$ containing the optimal candidates w.r.t. a candidate set $Cand$ and a Constraint ranking $R = C_1 \ldots C_n$ can then be determined recursively as follows, (cf. [Eis97]):

(1) $\mathbf{Opt}(R, Cand) = Cand$, if R is empty
$\mathbf{Opt}(R, Cand) =$, $\mathbf{Opt}(R', Cand')$ otherwise,
where $R' = C_2 \ldots C_n$ and $Cand' = \mathbf{Opt}(C_1, Cand)$

As alphabet symbols (possibly complex) feature structures (*FSs*) without recursion are used. Transition labels consist of symbols or predicates that denote a final set of symbols, thus preserving the formal equivalence to standard finite state machines. Automata and transducers will be notated without mention by equivalent regular expressions and binary regular relations respectively(cf. [KK98]).

1.3 Affix Blocking in Georgian

Georgian shows especially clear examples of affix blocking:

(2) *g-xedav* 'I see you'(sg) *g-xedav-t* 'I see you'(pl) *g-xedav-s* 'he sees you'(sg) *g-xedav-(*s)-t-(*s)* 'he sees you'(pl)

Since *-t* marks ('parses') plurality of 1st and 2nd person arguments and *-s* marks 3rd singular subjects, the ungrammatical form *g-xedav-s-t* would be optimally transparent. However it would violate a constraint valid for the whole present paradigm of Georgian prohibiting multiple agreement suffixes(**AT_MOST_1**($Suffix$)). In OT-terms this constraint can be assumed to dominate all parse constraints like **PARSE**($Number$) or **PARSE**($Person$). As in other applications of OT, we expect that different rankings give rise to different possible grammars. This too is born out for the Georgian data. In certain (non-standard) varieties of the language *g-xedav-s-t* is grammatical and *g-xedav-t* ungrammatical, exactly what we would expect under the ranking **PARSE**($Number$) $\ll$ **PARSE**($Person$) $\ll$ **AT_MOST_1**($Suffix$)

1.4 Input and Output

As in most OT-based work on morphology I assume the input for morphological computation to derive from the phonologically unspecified output of syntactic operations. The morphological input is represented by a list of feature structures, where each feature structure stands for a syntactic X^0 category and list precedence mirrors the hierarchical structure of syntax in a way that I won't discuss here. The basic resources of morphology consist of a set of 'vocabulary items' (*VIs*), i.e. strings of phonemes associated with affixal status information (prefix, suffix or stem) and underspecified feature structures of the same type as the input feature structures. The following diagram shows the pairing of input and output for the Georgian verb form g-xedav-t, 'we see you(pl)':

(3) INPUT:

[(cat v)(ind xedav)]
[(cat agr)(cas acc)(1 -)(3 -)(pl +)]
[(cat agr)(cas nom)(1 +)(3 -)(pl +)]

OUTPUT:

[g- [(cat agr)(cas acc)(1 -)(3 -)]
[xedav (cat v)(ind xedav)]
[-t [(cat agr)(3 -)(pl +)]]

The categories of subject and object agreement in the feature system are distinguished by the corresponding case features. Open class categories are assumed to contain an index feature ('ind') which distinguishes their members. *VIs* are represented as complex feature structures. Dashes in strings like 'gv-' mark iconically prefixes and suffixes while the absence of a dash designates a stem. Formally such strings are meant as a shorthand for *FSs* like [(phon gv) (status prefix) [....]].

In making statements about the relationship between input *FSs* and *VIs* it is often convenient to treat a *VI* as if consisting only of the embedded *FS* of syntactic features. So I will say that a *VI* and a *FS* unify when the *FS* and the *VIs* embedded *FS* unify.

1.5 The Lexicon and GEN

The lexicon simply consists of the intersection of M^*, where M is the set of *VIs* and the following automaton:

(4) [(status prefix)]* [(status stem)]* [(status suffix)]*

This means that all prefixes precede all stems and all suffixes, and all stems precede all suffixes.[1] Following [Noy93] I take it for granted that no features are ever introduced by vocabulary items, [2] i.e. each *VI* in the candidate set subsumes at least one *FS* in the input. **GEN** then creates a candidate set for a given input I by taking the lexicon automaton erasing all transitions whose labels don't satisfy this requirement for I. The candidate set for the above form will thus be:

(5) [g- ...] [v-...] * [xedav ...]* [-t ...]*

1.6 Blocking constraints

Blocking constraints are written **BLOCK**(FS), where FS is a feature structure or predicate. Thus **BLOCK**([(status prefix)]) is the constraint that evaluates each prefix except one as a constraint violation. The notation corresponds to the following regular relation scheme, where **NOT**(FS) denotes all *VIs* not subsumed by FS.

(6) (**NOT**(FS)/0)* (FS/0) (**NOT**(FS)/0)* (FS/1)*

1.7 Parse constraints

Parsing in the sense required here (which has to be distinguished from the standard notion of parsing applied in section 1.8) is defined as follows:

(7) A feature structure FS_{output} parses a feature F in a feature structure FS_{input} if and only if FS_{output} subsumes FS_{input} and F is specified in FS_{output} and FS_{input}.

[1] This treatment of precedence relations will be modified essentially in section 1.9.
[2] For further discussion see section 1.10.3.

This is intended to cover the fact that a single *VI* can parse features of different input *FSs* as long as it subsumes all of them. Thus in (3) the plural *-t* subsumes both the *FSs* for object and subject agreement. Parse constraints -written **PARSE**($[A_1 \ldots A_n]$) where $A_1 \ldots A_n$ are attributes in feature structures - require that for each *FS* in the input that contains feature specifications of all these attributes, there must be at least one VI in the output that parses all of its features (for these attributes) in *FS*. For a given single input *FS* this is realized by optimizing the output automaton through a elementary constraint - written **PARSE**(*FS*) - of the form:

(8) $(\mathbf{ANY}())^*\ (\mathbf{SUBSUMES}(FS))\ (\mathbf{ANY}())^*$

The predicate **ANY**() is true for any *VI* in the vocabulary, **SUBSUMES**(FS) for any *VI* subsuming FS. For the grammar to work there must be such an automaton for any FS in the input alphabet, meaning that the number of constraint automata can grow quite large, but since all of these automata differ only in the specification of FS they can be easily generated dynamically during the generation process. The application of a parse constraint **PARSE**($[A_1 \ldots A_n]$) to a candidate automaton CA given the input $FS_1 \ldots FS_n$ then proceeds as follows:

(9)
Set C to CA
For $FS_i \Leftarrow FS_1$ to FS_n
  if all $A_1 \ldots A_n$ are specified in FS_i
    generate(**PARSE**(FS_i))
    Conditioned_Intersection(C, **PARSE**(FS_i))

1.8 Parsing

Analyzing Input lists as strings of feature structures the space of possible inputs can formalized by a finite state automaton. The following regular expression stands for the inputs of Georgian verb forms:

(10) [(cat agr) (cas nom)] ([(cat agr) (cas acc)])? [(cat v)]

The parsing procedure uses invariant properties of the input-output mapping to narrow down the possible inputs for a given output form to a small number of candidates, which are then tested against by generation. The output-to- input inferences are accomplished by methods also used in generation, namely creating dynamically *FSAs* and (this time unconditioned) automata intersection. Output-input inference proceeds in three steps, the last of which('Inference from the blocking of more specific *VIs*') will be neglected here for reasons of space.

1.8.1 Inference using the structure of GEN

Since **GEN** licenses *FSs* in output forms only when they subsume some *FS* in the input each *VI* in an output form allows the inference that there is at least one input *FS* that is subsumed by it. Thus for each output *VI M* we create an automaton of the form [ ]* *M* [ ]* and intersect the resulting automata successively with the automaton describing the possible input lists.

1.8.2 Inference from the absence of nonblocked morphemes

The basic idea here is that in the absence of a *VI* you can infer the absence of a corresponding *FS* in the input. Especially the following claim is made: If no instance of a *VI* M is present in an output word form *W*, if *M* is not blocked in *W* and *M* is not redundant in *W*, then there is no *FS* in the input that subsumes *M*. Some definitions are at hand:

(11) a. A *VI M* is not blocked in *W* if and only if *W* contains no *VI M'* such that there is a constraint **BLOCK**(*X*), where *M* and *M'* are subsumed by *X*.

b. A *VI M* is not redundant in *W* if and only if there's at least one feature *F* in *M* such that there isn't a *M'* unifying with *M* and containing *F*.

Since the validity of this inference scheme isn't obvious, a proof follows:

(12) Assume in the input of *W* there is a feature structure *FS* subsumed by *M*. From the definition of subsumption follows that F is in *FS*. Assume now a word form *W'* that differs from *W* only by the correct addition of *M*. Since there is a input feature structure subsumed by it, *M* is licensed in *W* by **GEN**. By assumption *M* is not blocked and thus *W'* doesn't violate any blocking constraints that *W* doesn't. As is clear from the structure of parsing constraints no string containing a subset of another strings *VI*s can violate a parsing constraint that the other doesn't. Since there are only blocking and parsing constraints *W'* does violate only constraints that W violates either. But *W* violates at least one constraint that *W'* doesn't, namely **PARSE**(*F*), because to satisfy it there would have to be a morpheme *M"* in *W* containing *F* and subsuming *FS*. But by the definition of unification this would contradict the initial assumption about *F*. So *W* would be less optimal than *W'*, which leads to an obvious contradiction.

For each *VI* that satisfies the condition for the inference scheme an automaton of the form (**NOT_SUBSUMED_BY**(*VI*))* is created. Again these automata are successively intersected with the input automaton.

1.8.3 An example parse

(13) *m-* xedav
[(cat agr)(cas acc)(1 +)(3 -)] [(cat v)(ind xedav)]
'you(sg) see me'

Application of the first inference scheme to the input automaton gives

(14) [(cat agr)(cas nom)][(cat agr) (cas acc)(1 +)(3 -)] [(cat v)(ind xedav)]

From the absence of suffixal plural *-t* ([(cat agr)(3 -)(pl +)]) it follows that there are no 1st or 2nd person plural morphemes. By the absence of suffixal *-s* ([(cat agr) (cas nom)(3 +)(1 -)]) and *-en* ([(cat agr) (cas nom)(3 +)(1 -) (pl +)]) 3rd person subjects are excluded. We thus get:

(15) [(cat agr)(cas acc)(1 +)(3 -)(pl -)] [(cat agr) (cas nom)(3 -)(pl -)]

The only indeterminacy that remains is, if the (cas nom) *FS* is 1st or second person , i.e. (1 +) or (1 -)

1.9 Linear Order

The only constraint on the linear order of morphemes up to this point is the (unviolable) requirement that suffixes follow and prefixes precede stems, where affixal status is stipulated in the *FSs* of single affixes. This is obviously not sufficient for fixing the linear order of word forms with multiple suffixes or prefixes. Neither does it capture the fact that affixal status is highly systematical both in single languages and crosslinguistically. Consider for an example the following paradigms from Amharic:

(16)

	Imperfect	**Perfect**
3. sg. mas	*yë-säbër*	*säbbär-ä*
3. sg. fem	*të-säbër*	*säbbär-äcc*
2. sg. mas	*të-säbër*	*säbbär-h*
2. sg. fem	*të-säbr-i*	*säbbär-sh*
1. sg.	*ë-säbër*	*säbbär-hu*
3. pl.	*yë-säbr-u*	*säbbär-u*
2. pl.	*të-säbr-u*	*säbbär-accuh*
1. pl.	*ënnë-säbër*	*säbbär-n*

All perfect affixes are suffixes, while all affixes in the imperfect carrying (subject) person features are prefixes. Imperfect affixes expressing exclusively subject gender (*-i*) or number (*-u*) are again suffixes. Assuming that the prefixes in the imperfect forms carry only agreement features (for person), while perfect suffixes carry additionally tense features, the observed

precedence pattern can be adduced to the interaction of two constraints expressing universally observable preferences(cf. [ACG85]: [3]

(17) a. (Person) Agreement Affixes are Prefixes
b. Tense Affixes are Suffixes

1.9.1 Single LP Constraints

Single linear Precedence (LP) constraints[4] are characterized by two disjunct sets of symbols (*Anchors* and *Targets*) and a direction requirement (*Right* or *Left*). Thus for **PRECEDENCE**(*Anchors*, *Targets*, *Right*) each symbol $\in$ *Targets* which occurs on the left of at least one symbol $\in$ *Anchors* induces one constraint violation. As a regular relation this gives (18a), where the right member of the union contains all strings without an anchor, (inducing no violations at all). The left member contains all strings with at least one anchor, and violations for preceding targets:

(18) a. $\{\mathbf{NOT}(Targets)/0, Targets/1\}^* \; Anchors/0 \; (\mathbf{NOT}(Anchors)/0)^*) \cup (\mathbf{NOT}(Anchors))^*$
b. $(\mathbf{NOT}(Anchors)/0)^* \; Anchors/0 \; \{\mathbf{NOT}(Targets)/0, Targets/1\}^*) \cup (\mathbf{NOT}(Anchors))^*$

(18b) standing for $\mathbf{PRECEDENCE}(Anchors, Targets, Right)$ is the mirror image of (18a). (17) can now be expressed schematically as (19):

(19) a. $\mathbf{PRECEDENCE}(Stems, PersonAgreement, Left)$
b. $\mathbf{PRECEDENCE}(Stems, Tense, Right)$

1.9.2 The status of precedence constraints

Combinations of precedence constraints can in principle have undesirable side effects and act as blocking constraints[5]. Thus the following ranking will block all strings containing occurrences of A and B simultaneously if the input candidate set contains strings not doing so:

(20) $\mathbf{PRECEDENCE}(A, B, Left) \ll \mathbf{PRECEDENCE}(B, A, Left)$

[3] The position of *-u* and *-i* obviously isn't explained by this. But since Amharic as Georgian has a high-ranked constraint blocking multiple prefixes, the position of these affixes follows.

[4] [PS93] assumes that linear order is achieved differently by alignment constraints. In their generalized forms such constraints can't be implemented by the finite-state devices that I use here (cf. [Eis97]).

[5] Note that this is due to the violability of constraints, not to their implementations as finite-state machines.

The first constraint will filter out all candidates where any B precedes an A. leaving strings containing exclusively As or Bs or containing both standing in the right precedence relation. The second constraint will eliminate the latter set prohibiting in this way the coocurrence of As and Bs. Conceptually it seems desirable to minimize such side effects by modularizing the constraint system, in a way that the computation of precedence relations optimizes the output of a component regulating the appearance of morphemes. In other words all LP constraints precede all Parse and Blocking Constraints.

1.9.3 Parsing and LP constraints

A technical advantage of this modularization is that parse constraints thus don't affect the validity of the introduced parsing techniques. This is trivially true for the first inference scheme (section 1.8.1). To see that the second scheme remains valid conceive an morphological OT-grammar as consisting of two parts G_1 and G_2 the first containing a hierarchy of parse and blocking constraints, the second a ranking of LP constraints. By (1) G_1 maps a regular language L_0 to a regular language L_1 (the set of strings from L_0 which are optimal w.r.t. G_1) G_2 maps L_1 to L_2 the subset of L_1 which is optimal w.r.t. L_2. Now, the second parsing scheme is obviously valid for each string $\in$,L_1 and hence due to the subset relation to each string of L_2 as well.

1.10 Further Constraint Types

In this section I discuss loosely a number of constraint types which are empirically plausible or have been proposed in the literature, but aren't actually integrated in the current formalism.

1.10.1 Minimum Constraints

While blocking constraints strive to reduce the number of morphemes in word forms languages often impose minimum requirements on the appearence of morphemes. For example in Classical Nahuatl nouns have to carry at least one affix. If no meaningful affix is justified for the form the semantically empty 'absolutive' affix *-li* appears:

	no-cal	*cal-tin*	*no-cal-huan*	*cal-li/*cal*
(21)	my-house	house-plu	my-house-plu	house-abs
	'my house'	'houses'	'my houses'	'house'

Constraints of this type can be formalized as:

(22) $[\,]^* M [\,]^*$

where M is the set of morphemes for which the minimum requirement is valid, in the case of Nahuatl the set of affixes. Note that in the context of our formalism minimum constraints naturally can't lead to the insertion of morphemes not subsuming input *FSs*. For Nahuatl it would be actually predicted that the *FS* for the absolutive morpheme has to be unspecified, apart from a feature marking its affixal status.

1.10.2 OCP constraints

The OCP (Obligatory Contour Principle) which is originally a purely phonological constraint has been applied to morphology i.a. by [Gri97]) to explain data like certain opacity effects in Romance clitic clusters. For example in Italian the impersonal ('one') and the reflexive clitic ('him-/her-self') in isolation have both the form *si*, while the first *si* is replaced by the first plural clitic *ci*, when both are combined([Gri97]:12):

(23) *Ci si lava*
one oneself washes
'One washes oneself.'

[Gri97] assumes a constraint which she writes *XX to prohibit sequences of identical clitics. The formal nature of this is somewhat unclear, since the constraint seems also to block the coocurrence of clitics that aren't strictly identical as *le* (dative feminine singular) and *lo* (accusative masculine singular) neither phonologically nor morphologically. Because of the highly restricted ordering of clitic groups the data don't even show that linear adjacency is a necessary prerequisite for the blocking effect. Technically this means that OCP constraints can be implemented as blocking constraints.

1.10.3 FILL constraints

As illustrated by (24) [Gri97] assumes that **GEN** admits *VIs* not subsuming input *FSs*. More concretely in her analysis (24a) realizes (24b)

(24) a. [+refl +3 + pl]
b. *ci* [+1 + pl]

While this kind of 'feature insertion' is possible in this case to satisfy a higher ranked OCP constraint, it is blocked elsewhere by constraints of the FiLL[6] family, that prohibit features not present in the input. Note that the pruning operation in section 1.5 in fact is a FILL constraint, however an inviolable one. More specific FILL constraints of the form **FILL**(F), for F

[6]The name derives from the 'unfilled' symbol □ used e.g. in[PS93] to depict inserted segments.

a feature structure could be implemented by a operation which prunes all transition *FSs* specified for F that don't subsume a *FS* in the input list. Since these are functions from regular sets into regular sets they can be conceived as operations of the **Opt**(*Cons*, *Cand*) type in the algorithm in (1). This extension however would imply a much less restricted formalism, especially the parsing techniques developed in 1.8 wouldn't work under rendering FILL constraints violable.[7]

1.11 Open Questions

The parse procedure guarantees termination only for input automata denoting finite lists. However it seems to be true for inflectional morphology in general and for most cases in derivational morphology that there are no multiple or even unbounded occurrences of the same 'morpheme' i.e. input *FS* type, which would imply that there are always only finitely many input lists.

A major question not addressed here is the question what constraints restrict redundancy. In the analyses given here this was achieved mainly by blocking constraints. But other solutions have been advocated (cf. ([Noy93]).

Finally, a number of recent developments in optimality theory, namely correspondence theory([MP95]) and output-output constraints ([Ben97]) have not been taken into account. While I suspect that the complexities of the first might be superfluous for OT-based morphology, the formal status of the latter is a promising topic for future research.

[7]Note that Grimshaws analysis is questionable on empirical grounds, since *ci* also functions as a local clitic. For a general discussion on feature insertion in morphology see [Noy98].

Bibliography

[ACG85] John A. Hawkins Anne Cutler and Gary Gilligan. The suffixing preference: a processing explanation. *Linguistics*, pages 723–758, 1985.

[Ben97] Laura Benua. *Transderivational Identity: Phonological Relations between words*. PhD thesis, University of Massachusetts, 1997.

[Eis97] Jason Eisner. Efficient generation in primitive optimality theory. In *ACL'97*, 1997.

[Ell94] T. Mark Ellison. Phonological derivation in optimality theory. In *COLING '94*, pages 1007–1013, 1994.

[FS98] Robert Frank and Giorgio Satta. Optimality theory and the generative complexity of constraint violation. *Computational Linguistics*, 1998.

[Gri97] Jane Grimshaw. The best clitic: Constraint conflict in morphosyntax. Ms. Rutgers University, 1997.

[Kar98] Lauri Karttunen. The proper treatment of optimality in computational phonology. In *Proceedings of FSMNLP98*, 1998.

[KK98] Ronald Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20:331–378, 1998.

[MP95] John McCarthy and Alan Prince. Faithfulness and reduplicative identity. *University of Massachusetts Occasional Papers in Linguistics*, pages 249–384, 1995.

[Noy93] Robert R. Noyer. Optimal words: towards a declarative theory of word formation. Ms. Princeton University, 1993.

[Noy98] Robert R. Noyer. Impoverishment theory and morphosyntactic markedness. In Diane K. Brentari Steven G. Lapointe and Patrick M. Farrell, editors, *Morphology and its relation to morphology and syntax*, pages 264–286, 1998.

[PS93] Alan Prince and Paul Smolensky. Optimality theory: Constraint interaction in generative grammar. Technical reports of the Rutgers University Center of Cognitive Science, 1993.

Proceedings of Formal Grammar 1999
Geert-Jan M. Kruijff & Richard T. Oehrle (editors).