# Practical Finite State Optimality Theory

**Dale Gerdemann**
University of Tübingen
dg@sfs.nphil.uni-tuebingen.de

**Mans Hulden**
University of the Basque Country
IXA Group
IKERBASQUE, Basque Foundation for Science
mhulden@email.arizona.edu

## Abstract

Previous work for encoding Optimality Theory grammars as finite-state transducers has included two prominent approaches: the so-called 'counting' method where constraint violations are counted and filtered out to some set limit of approximability in a finite-state system, and the 'matching' method, where constraint violations in alternative strings are matched through violation alignment in order to remove suboptimal candidates. In this paper we extend the matching approach to show how not only markedness constraints, but also faithfulness constraints and the interaction of the two types of constraints can be captured by the matching method. This often produces exact and small FST representations for OT grammars which we illustrate with two practical example grammars. We also provide a new proof of nonregularity of simple OT grammars.

## 1 Introduction

The possibility of representing Optimality Theory (OT) grammars (Prince and Smolensky, 1993) as computational models and finite-state transducers, in particular, has been widely studied since the inception of the theory itself. In particular, constructing an OT grammar step-by-step as the composition of a set of transducers, akin to rewrite rule composition in (Kaplan and Kay, 1994), has offered the attractive possibility of simultaneously modeling OT parsing and generation as a natural consequence of the bidirectionality of finite-state transducers. Two main approaches have received attention as practical options for implementing OT with finite-state transducers: that of Karttunen (1998) and Gerdemann and van Noord (2000).[1] Both approaches model constraint interaction by constructing a GEN-transducer, which is subsequently composed with filtering transducers that mark violations of constraints, and remove suboptimal candidates—candidates that have received more violation marks than the optimal candidate, with the general template:

```
Grammar = Gen .o. MarkC1 .o. FilterC1 ...
                  MarkCN .o. FilterCN
```

In Karttunen's system, auxiliary 'counting' transducers are created that first remove candidates with maximally $k$ violation marks for some fixed $k$, then $k-1$, and so on, until nothing can be removed without emptying the candidate set, using a finite-state operation called *priority union*. Gerdemann and van Noord (2000) present a similar system that they call a 'matching' approach, but which does not rely on fixing a maximal number of distinguishable violations $k$. The matching method is a procedure by which we can in many cases (though not always) distinguish between infinitely many violations in a finite-state system—something that is not possible when encoding OT by the alternative approach of counting violations.

In this paper our primary purpose is to both extend and simplify this 'matching' method. We will include interaction of both markedness and faithfulness constraints (MAX, DEP, and IDENT violations)—going beyond both Karttunen (1998) and Gerdemann and van Noord (2000), where only markedness constraints were modeled. We shall also clarify the notation and markup used in the matching approach as well as present a set of generic transducer templates for EVAL by which modeling varying OT grammars becomes a simple matter of modifying the necessary constraint transducers and ordering them correctly in a series of compositions.

[1]Earlier finite-state approaches do exist, see e.g. Ellison (1994) and Hammond (1997).

We will first give a detailed explanation of the 'matching' approach in section 2—our encoding, notation, and tools differ somewhat from that of Gerdemann and van Noord (2000), although the core techniques are essentially alike. This is followed by an illustration of our encoding and method through a standard OT grammar example in section 3. In that section we also give examples of debugging OT grammars using standard finite state calculus methods. In section 4 we also present an alternate encoding of an OT account of prosody in Karttunen (2006) illustrating devices where GEN is assumed to add metrical and stress markup in addition to changing, inserting, or deleting segments. We also compare this grammar to both a non-OT grammar and an OT grammar of the same phenomenon described in Karttunen (2006). In section 5, we conclude with a brief discussion about the limitations of FST-based OT grammars in light of the method developed in this paper, as well as show a new proof of nonregularity of some very simple OT constraint systems.

### 1.1 Notation

All the examples discussed are implemented with the finite-state toolkit *foma* (Hulden, 2009b). The regular expressions are also compilable with the Xerox tools (Beesley and Karttunen, 2003), although some of the tests of properties of finite-state transducers, crucial for debugging, are unavailable. The regular expression formalism used is summarized in table 1.

| | |
|---|---|
| `AB` | Concatenation |
| `A\|B` | Union |
| `~A` | Complement |
| `?` | Any symbol in alphabet |
| `%` | Escape symbol |
| `[` and `]` | Grouping brackets |
| `A:B` | Cross product |
| `T.l` | Output projection of T |
| `A -> B` | Rewrite A as B |
| `A (->) B` | Optionally rewrite A as B |
| `\|\| C _ D` | Context specifier |
| `[..] -> A` | Insert one instance of A |
| `A -> B ... C` | Insert B and C around A |
| `.#.` | End or beginning of string |

Table 1: Regular expression notation in *foma*.

## 2 OT evaluation with matching

In order to clarify the main method used in this paper to model OT systems, we will briefly recapitulate the 'matching' approach to filter out suboptimal candidates, or candidates with more violation marks in a string representation, developed in Gerdemann and van Noord (2000).[2]

### 2.1 Worsening

The fundamental technique behind the finite-state matching approach to OT is a device which we call 'worsening', used to filter out strings from a transducer containing more occurrences of some designated special symbol $s$ (e.g. a violation marker), than some other candidate string in the same pool of strings. This method of transducer manipulation is perhaps best illustrated through a self-contained example.

Consider a simple morphological analyzer encoded as an FST, say of English, that only adds morpheme-boundaries—+-symbols—to input words, perhaps consulting a dictionary of affixes and stems. Some of the mappings of such a transducer could be ambiguous: for example, the words **deconstruction** or **incorporate** could be broken down in two ways by such a morpheme analyzer:

| **Input** | **Output** |
|---|---|
| deconstruction | de+construct+ion |
| | deconstruct+ion |
| incorporate | in+corporate |
| | incorporate |

Suppose our task was now to remove alternate morpheme breakdowns from the transducer so that, if an analysis with a smaller number of morphemes was available for any word, a longer analysis would not be produced. In effect, **deconstruction** should only map to **deconstruct+ion**, since the other alternative has one more morpheme boundary. The worsening trick is based on the idea that we can use the existing set of words from the output side of the morphology, add at least one morpheme boundary to all of them, and use the resulting set of words to filter out longer 'candidates' from the original morphology. For example, one way of adding a **+**-symbol to **de+construction** produces

[2] Also discussed in Jäger (2002).

**de+construct+ion**, which coincides with the original output in the morphology, and can now be used to knock out this suboptimal division. This process can be captured through:

```
AddBoundary = [?* 0:%+ ?*]+;
Worsen      =  Morphology .o. AddBoundary;
Shortest    =  Morphology .o. ~Worsen.l;
```

the effect of which is illustrated for the word **deconstruction** in figure 1. Here, `AddBoundary` is a transducer that adds at least one +-symbol to the input. The `Worsen` transducer is simply the original transducer composed with the `AddBoundary` transducer. The `Shortest` morphology is then constructed by extracting the output projection of `Worsen`, and composing its negation with the original morphology.

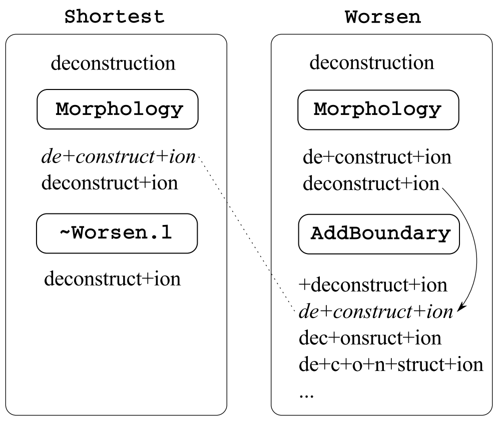
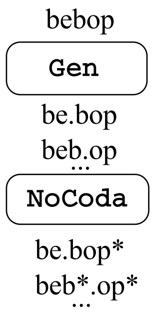


Figure 1: Illustration of a worsening filter for morpheme boundaries.

## 2.2 Worsening in OT

The above 'worsening' maneuver is what the 'matching' approach to model OT syllabification is based upon. Evaluation of competing candidates with regard to a single OT constraint can be performed in the same manner. This, of course, presupposes that we are using transducers to mark constraint violations in input strings, say by the symbol *. Gerdemann and van Noord (2000) illustrate this by constructing a GEN-transducer that syllabifies words,[3] and another set of transducers that mark violations of some constraint. Then, having a constraint, NOCODA, implemented as a transducer that adds violation marks when syllables end in consonants, we can achieve the following sequence of markup by composition of GEN and NOCODA, for a particular example input **bebop**:



The above transducers could be implemented very simply, by epenthesis replacement rules:

```
# Insert periods arbitrarily inside words
Gen    =  [..] (->) %. || \.#. _ \.#.  ;
# Insert *-marks after C . or C .#.
NoCoda = [..] -> %* || C+ [%. | .#.] _ ;
```

Naturally, at this point in the composition chain we would like to filter out the suboptimal candidates—that is, the ones with fewer violation marks, then remove the marks, and continue with the next constraint, until all constraints have been evaluated. The problem of filtering out the suboptimal candidates is now analogous to the 'worsening' scenario above: we can create a 'worsening'-filter automaton by adding violation marks to the entire set of candidates. In this example, the candidate **be.bop*** would produce a worse candidate **be*.bop***, which (disregarding for the moment syllable boundary marks and the exact position of the violation) can be used to filter out the suboptimal **beb*.op***.

## 3 An OT grammar with faithfulness and markedness constraints

As previous work has been limited to working with only markedness constraints as well as a somewhat impoverished GEN—one that only syllabifies words—our first task when approaching a more complete finite-state methodology of OT needs to address this point. In keeping with the 'richness of the base'-concept of OT, we require a suitable

[3]Although using a much more complex set of markup symbols than here.

GEN to be able to perform arbitrary deletions (elisions), insertions (epentheses), and changes to the input. A GEN-FST that only performs this task (maps $\Sigma^* \rightarrow \Sigma^*$) on input strings is obviously fairly easy to construct. However, we need to do more than this: we also need to keep track of which parts of the input have been modified by GEN in any way to later be able to pinpoint and mark faithfulness violations—places where GEN has manipulated the input—through an FST.

### 3.1 Encoding of GEN

Perhaps the simplest possible encoding that meets the above criteria is to have GEN not only change the input, but also mark each segment in its output with a marker whereby we can later distinguish how the input was changed. To do so, we perform the following markup:

- Every surface segment (output) is surrounded by brackets [ . . . ].
- Every input segment that was manipulated by GEN is surrounded by parentheses ( . . . ).

For example, given the input **a**, GEN would produce an infinite number of outputs, and among them:

| | |
|---|---|
| `[a]` | GEN did nothing |
| `(a)[]` | GEN deleted the a |
| `(a)[e]` | GEN changed the `a` to `e` |
| `()[d](a)[i]` | GEN inserted a `d` and changed `a` to `i` |
| `...` | |

This type of generic GEN can be defined through:

```
Gen = S -> %( ... %) %[ (S) %]    ,,
      S -> %[ ... %]              ,,
      [..] (->) [%( %) %[ S %]]*  ;
```

assuming here that `S` represents the set of segments available.

### 3.2 Evaluation of faithfulness and markedness constraints

As an illustrative grammar, let us consider a standard OT example of word-final obstruent devoicing—as in Dutch or German—achieved through the interaction of faithfulness and markedness constraints. The constraints model the fact that underlyingly voiced obstruents surface as devoiced in word-final position, as in **pad** → **pat**. A set of core constraints to illustrate this include:

- *VF: a markedness constraint that disallows final voiced obstruents.
- IDENTV: a faithfulness constraint that militates against change in voicing.
- VOP: a markedness constraint against voiced obstruents in general.

The interaction of these constraints to achieve devoicing can be illustrated by the following tableau.[4]

| bed | *VF | IDENTV | VOP |
|---|---|---|---|
| ☞ bet | | * | * |
| pet | | **! | |
| bed | *! | | ** |
| ped | *! | * | * |

The tableau above represents a kind of shorthand often given in the linguistic literature where, for the sake of conciseness, higher-ranked faithfulness constraints are omitted. For example, there is nothing preventing the candidate **bede** to rank equally with **bet**, were it not for an implicit high-ranked DEP-constraint disallowing epenthesis. As we are building a complete computational model with an unrestricted GEN, and no implicit assumptions, we need to add a few constraints not normally given when arguing about OT models. These include:

- DEP: a faithfulness constraint against epenthesis.
- MAX: a faithfulness constraint against deletion.
- IDENTPL: a faithfulness constraint against changes in place of articulation of segments. This is crucial to avoid e.g. **bat** or **bap** being equally ranked with **bet** in the above example.[5]

[4]The illustration roughly follows (Kager, 1999), p. 42.

[5]Note that a generic higher-ranked IDENT will not do, because then we would never get the desired devoicing in the first place.

Including these constraints explicitly allows us to rule out unwanted candidates that may otherwise rank equal with the candidate where word-final obstruents are devoiced, as illustrated in the following:

| bed | DEP | MAX | IDENTPL | *VF | IDENTV | VOP |
|---|---|---|---|---|---|---|
| ☞ bet | | | | | * | * |
| pet | | | | | **! | |
| bed | | | | *! | | ** |
| ped | | | | *! | * | * |
| bat | | | *! | | * | * |
| bep | | | *! | | * | * |
| be | | *! | | | | * |
| bede | *! | | | | | ** |

Once we have settled for the representation of GEN, the basic faithfulness constraint markup transducers—whose job is to insert asterisks wherever violations occur—can be defined as follows:

```
Dep      = [..] -> {*} || %( %) _ ;
Max      = [..] -> {*} || %[ %] _ ;
Ident    = [..] -> {*} || %( S %) %[ S %] _ ;
```

That is, DEP inserts a *-symbol after ( )-sequences, which is how GEN marks epenthesis. Likewise, MAX-violations are identified by the sequence **[ ]**, and IDENT-violations by a parenthesized segment followed by a bracketed segment. To define the remaining markup transducers, we shall take advantage of some auxiliary template definitions, defined as functions:

```
def Surf(X)      [X .o. [0:%[ ? 0:%]]*].l/
                 [ %( (S) %) | %[ %] ];
def Change(X,Y) [%( X %) %[ Y %]];
```

Here, `Surf(X)` in effect changes the language `X` so that it can match every possible surface encoding produced by GEN; for example, a surface sequence **ab** may look like **[a][b]**, or **[a](a)[b]**, etc., since it may spring from various different underlying forms. This is a useful auxiliary definition that will serve to identify markedness violations. Likewise `Change(X,Y)` reflects the GEN representation of changing a segment `X` to `Y` needed to concisely identify changed segments. Using the above we may now define the remaining violation markups needed.

```
CVOI     = [b|d|g];
Voiced   = [b|d|g|V];
Unvoiced = [p|t|k];
define VC       Change(Voiced,Unvoiced) |
                Change(Unvoiced,Voiced);
define Place   Change(p,?-b)|Change(t,?-d)|
                Change(k,?-g)|Change(b,?-p)|
                Change(d,?-t)|Change(g,?-k)|
                Change(a,?)|Change(e,?)|
                Change(i,?)|Change(o,?)|
                Change(u,?);

VF       = [..] -> {*} || Surf(CVOI) _ .#.   ;
IdentV   = [..] -> {*} || VC _               ;
VOP      = [..] -> {*} || Surf(CVOI)  _      ;
IdentPl = [..] -> {*} || Place _            ;
```

The final remaining element for a complete implementation concerns the question of 'worsening' and its introduction into a chain of transducer composition. To this end, we include a few more definitions:

```
AddViol =    [?* 0:%* ?*]+;
Worsen  =    [Gen.i .o. Gen]/%* .o. AddViol;
def Eval(X) X .o. ~[X .o. Worsen].l .o. %*->0;
Cleanup =    %[|%] -> 0 .o. %( \%)* %) -> 0;
```

Here, `AddViol` is the basic worsening method discussed above whereby at least one violation mark is added. However, because GEN adds markup to the underlying forms, we need to be a bit more flexible in our worsening procedure when matching up violations. It may be the case that two different competing surface forms have the same underlying form, but the violation marks will not align correctly because of interfering brackets. Given two competing candidates with a different number of violations, for example **(a)[b]*** and **[a]**, we would like the latter to match the former after adding a violation mark since they both originate in the same underlying form **a**. The way to achieve this is to *undo* the effect of GEN, and then redo GEN in every possible configuration before adding the violation marks. The transducer `Worsen`, above, does this by a composition of the inverse GEN, followed by GEN, ignoring already existing violations. For the above example, this leads to representations such as:

$$\texttt{[a]} \overset{\texttt{Gen.i}}{\rightarrow} \texttt{a} \overset{\texttt{Gen}}{\rightarrow} \texttt{(a)[b]} \overset{\texttt{AddViol}}{\rightarrow} \texttt{(a)[b]*}.$$

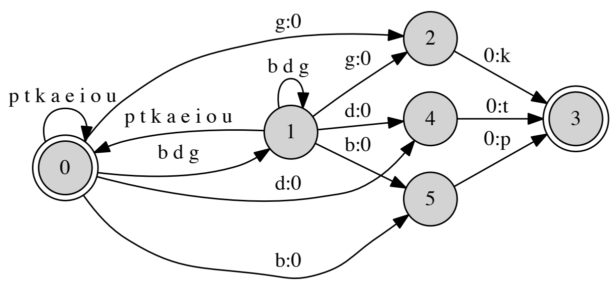
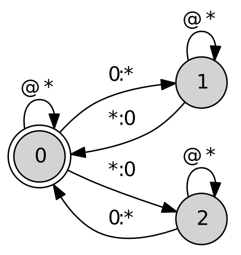


Figure 2: OT grammar for devoicing compiled into an FST.



```
Permute = [?* [%*:0 ?* 0:%*|0:%* ?* %*:0]* ?*]*;
```

Figure 3: Violation permutation transducer.

We also define a `Cleanup` transducer that removes brackets and parts of the underlying form.

Now we are ready to compile the entire system into an FST. To apply only GEN and the first constraint, for example, we can calculate:

```
  Eval(Gen .o. Dep) .o. Cleanup;
```

and likewise the entire grammar can be calculated by:

```
  Eval(Eval(Eval(Eval(Eval(Eval(
  Gen .o. Dep) .o. Max) .o. IdentPl) .o.
  VF) .o. IdentV) .o. VOP) .o. Cleanup;
```

This yields an FST of 6 states and 31 transitions (see figure 2)—it can be ascertained that the FST indeed does represent a relation where word-final voiced obstruents are always devoiced.

## 3.3 Permutation of violations

As mentioned in Gerdemann and van Noord (2000), there is an additional complication with the 'worsening'-approach. It is not always the case that in the pool of competing candidates, the violation markers line up, which is a prerequisite for filtering out suboptimal ones by adding violations—although in the above grammar the violations do line up correctly. However, for the vast majority of OT grammars, this can be remedied by inserting a violation-permuting transducer that moves violations markers around before worsening, to attempt to produce a correct alignment. Such a permuting transducer can be defined as in figure 3.

If the need for permutation arises, repeated permutations can be included as many times as warranted in the definition of `Worsen`:

```
Permute = [%*:0 ?* 0:%*|0:%* ?* %*:0]*/?;
Worsen  = [Gen.i .o. Gen]/%* .o.
          Permute .o. ... .o. Permute .o.
          AddViol;
```

Knowing how many permutations are necessary for the transducer to be able to distinguish between any number of violations in a candidate pool is possible as follows: we can can calculate for some constraint `ConsN` in a sequence of constraints,

```
Eval(Eval(Gen .o. Cons1) ... .o. ConsN) .o.
     ConsN .o. \%* -> 0;
```

Now, this yields a transducer that maps every underlying form to $n$ asterisks, $n$ being the number of violations with respect to `ConsN` in the candidates that have successfully survived `ConsN`. If this transducer represents a function (is single-valued), then we know that two candidates with a different number of violations have not survived `ConsN`, and that the worsening yielded the correct answer. Since the question of transducer functionality is known to be decidable (Blattner and Head, 1977), and an efficient algorithm is given in Hulden (2009a), which is included in *foma* (with the command `test functional`) we can address this question by calculating the above for each constraint, if necessary, and then permute the violation markers until the above transducer is functional.

## 3.4 Equivalence testing

In many cases, the purpose of an OT grammar is to capture accurately some linguistic phenomenon through the interaction of constraints rather than by other formalisms. However, as has been noted by

Karttunen (2006), among others, OT constraint debugging is an arduous task due to the sheer number of unforeseen candidates. One of the advantages in encoding an OT grammar through the worsening approach is that we can produce an exact representation of the grammar, which is not an approximation bounded by the number of constraint violations it can distinguish (as in Karttunen (1998)), or by the length of strings it can handle. This allows us to formally calculate, among other things, the equivalence of an OT grammar represented as an FST and some other transducer. For example, in the above grammar, the intention was to model end-of-word obstruent devoicing through optimality constraints. Another way to model the same thing would be to compile the replacement rule:

```
Rule = b -> p, d -> t, g -> k || _ .#. ;
```

The transducer resulting from this is shown in figure 4.

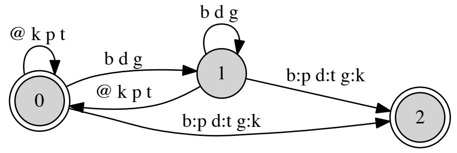


Figure 4: Devoicing transducer compiled through a rule.

As is seen, the OT transducer (figure 2) and the rule transducer (figure 4) are not structurally identical. However, both transducers represent a function—i.e. for any given input, there is always a unique winning candidate. Although transducer equivalence is not testable by algorithm in the general case, it is decidable in the case where one of two transducers is functional. If this is the case it is sufficient to test that domain($\tau_1$) = domain($\tau_2$) and that $\tau_2^{-1} \circ \tau_1$ represents identity relations only. As an algorithm to decide if a transducer is an identity transducer is also included in *foma*, it can be used to ascertain that the two above transducers are in fact identical, and that the linguistic generalization captured by the OT constraints is correct:

```
regex Rule.i .o. Grammar;
test identity
```

which indeed returns TRUE. For a small grammar, such as the devoicing grammar, determining the correctness of the result by other means is certainly feasible. However, for more complex systems the ability to test for equivalence becomes a valuable tool in analyzing constraint systems.

## 4 Variations on GEN: an OT grammar of stress assignment

Most OT grammars that deal with phonological phenomena with faithfulness and markedness grammars are implementable through the approach given above, with minor variations according to what specific constraints are used. In other domains, however, in may be the case that GEN, as described above, needs modification. A case in point are grammars that mark prosody or perform syllabification that often take advantage of only markedness constraints. In such cases, there is often no need for GEN to insert, change, and delete material if all faithfulness constraints are assumed to outrank all markedness constraints. Or alternatively, if the OT grammar is assumed to operate on a different stratum where no faithfulness constraints are present. However, GEN still needs to insert material into strings, such as stress marks or syllable boundaries.

To test the approach with a larger 'real-world' grammar we have reimplemented a Finnish stress assignment grammar, originally implemented through the counting approach of Karttunen (1998) in Karttunen (2006), following a description in Kiparsky (2003). The grammar itself contains nine constraints, and is intended to give a complete account of stress placement in Finnish words. Without going into a line-by-line analysis of the grammar, the crucial main differences in this implementation to that of the previous sections are:

- GEN only inserts symbols `(` `)` `ˋ` and `ˊ` to mark feet and stress
- Violations need to be permuted in `Worsen` to yield an exact representation
- GEN syllabifies words correctly through a replacement rule (no constraints are given in the grammar to model syllabification; this is assumed to be already performed)

```
kainostelijat      ->    (ka´i.nos).(te`.li).jat
kalastelemme       ->    (ka´.las).te.(le`m.me)
kalasteleminen     ->   *(ka´.las).te.(le`.mi).nen
kalastelet         ->    (ka´.las).(te`.let)
kuningas           ->    (ku´.nin).gas
strukturalismi     ->    (stru´k.tu).ra.(li`s.mi)
ergonomia          ->    (e´r.go).(no`.mi).a
matematiikka       ->    (ma´.te).ma.(ti`ik.ka)
```

Figure 5: Example outputs of matching implementation of Finnish OT.

Compiling the entire grammar through the same procedure as above outputs a transducer with 134 states, and produces the same predictions as Karttunen's counting OT grammar.[6] As opposed to the previous devoicing grammar, compiling the Finnish prosody grammar requires permutation of the violation markers, although only one constraint requires it (STRESS-TO-WEIGHT, and in that case, composing `Worsen` with one round of permutation is sufficient for convergence).

Unlike the counting approach, the current approach confers two significant advantages. The first is that we can compile the entire grammar into an FST that does not restrict the inputs in any way. That is, the final product is a stand-alone transducer that accepts as input *any* sequence of any length of symbols in the Finnish alphabet, and produces an output where the sequence is syllabified, marked with feet, and primary and secondary stress placement (see figure 5). The counting method, in order to compile at all, requires that the set of inputs be fixed to some very limited set of words, and that the maximum number of distinguishable violations (and indirectly word length) be fixed to some $k$.[7] The second advantage is that, as mentioned before, we are able to formally compare the OT grammar (because it is not an approximation), to a rule-based grammar (FST) that purports to capture the same phenomena. For example, Karttunen (2006), apart from the counting OT implementation, also provides a rule-based account of Finnish stress, which he discovers to be distinct from an OT account by finding two words where their respective predictions differ. However, by virtue of having an exact transducer, we can formally analyze the OT account together with the rule-based account to see if they differ in their predictions for *any* input, without having to first intuit a differing example:

```
regex RuleGrammar.i .o. OTGrammar;
test identity
```

Further, we can subject the two grammars to the usual finite-state calculus operations to gain possible insight into what kinds of words yield different predictions with the two—something useful for linguistic debugging. Likewise, we can use similar techniques to analyze for redundancy in grammars. For example, we have assumed that the VOP-constraint plays no role in the above devoicing tableaux. Using finite-state calculus, we can prove it to be so for any input if the grammar is constructed with the method presented here.

## 5 Limits on FST implementation

We shall conclude the presentation here with a brief discussion of the limits of FST representability, even of simple OT grammars. Previous analyses have shown that OT systems are beyond the generative capacity of finite-state systems, under some assumptions of what GEN looks like. For example, Frank and Satta (1998) present such a constraint system where GEN is taken to be defined through a transduction equivalent to:[8]

```
Gen = [a:b|b:a]* | [a|b]*;
```

That is, a relation which *either* maps all $a$'s to $b$'s and vice versa, or leaves the input unchanged. Now, let us assume the presence of a single markedness constraint ***a**, militating against the letter $a$. In that case, given an input of the format $a^*b^*$ the effective mapping of the entire system is one that is an identity relation if there are fewer $a$'s than $b$'s; otherwise the $a$'s and $b$'s are swapped. As is easily seen, this is not a regular relation.

One possible objection to this analysis of non-regularity is that linguistically GEN is usually assumed to perform any transformation to the input

[6]Including replicating errors in Kiparsky's OT analysis discovered by Karttunen, as seen in figure 5.

[7]Also, compiling the grammar is reasonably quick: 7.04s on a 2.8MHz Intel Core 2, vs. 2.1s for a rewrite-rule-based account of the same phenomena.

[8]The idea is attributed to Markus Hiller in the article.

whatsoever—not just limiting itself to a proper subset of $\Sigma^* \rightarrow \Sigma^*$. However, it is indeed the case that even with a canonical GEN-function, some very simple OT systems fall outside the purview of finite-state expressibility, as we shall illustrate by a different example here.

## 5.1 A simple proof of OT nonregularity

Assume a grammar that has four very basic constraints: IDENT, forbidding changes, DEP, forbidding epenthesis, $^*ab$, a markedness constraint against the sequence $ab$, and MAX, forbidding deletion, ranked IDENT,DEP $\gg$ $^*ab \gg$ MAX. We assume GEN to be as general as possible—performing arbitrary deletions, insertions, and changes.

It is clear, as is illustrated in table 2, that for all inputs of the format $a^n b^m$ the grammar in question describes a relation that deletes all the $a$'s or all the $b$'s depending on which there are fewer instances of, i.e. $a^n b^m \rightarrow a^n$ if $m < n$, and $a^n b^m \rightarrow b^m$ if $n < m$. This can be shown by a simple pumping argument to not be realizable through an FST.

| aaabb | IDENT | DEP | *ab | MAX |
|---|---|---|---|---|
| aaaaa | *!* | | | |
| aaacbb | | *! | | |
| aaabb | | | *! | |
| aaab | | | *! | * |
| bb | | | | ***! |
| ☞ aaa | | | | ** |

Table 2: *Illustrative tableau for a simple constraint system not capturable as a regular relation.*

Implementing this constraint system with the methods presented here is an interesting exercise and serves to examine the behavior of the method.

We define GEN, DEP, MAX, and IDENT as before, define a universal alphabet (excluding markup symbols), and the constraint *ab naturally as:

```
S        = ? - %( - %) - %[ - %] - %* ;
NotAB    = [..] -> {*} || Surf(a b) _ ;
```

Now, with one round of permutation of the violation markers in `Worsen` as follows:

```
Worsen =  [Gen.i .o. Gen]/{*} .o.
          AddViol .o. Permute;
```

we calculate

```
define Grammar Eval(Eval(Eval(Eval(
       Gen .o. Ident) .o. Dep) .o. NotAB) .o.
       Max) .o. Cleanup;
```

which produces an FST that cannot distinguish between more than two $a$'s or $b$'s in a string. While it correctly maps **aab** to **aa** and **abb** to **bb**, the tableau example of **aaabb** is mapped to both **aaa** and **bb**. However, with one more round of permutation in `Worsen`, we produce an FST that can indeed cover the example, mapping **aaabb** uniquely to **bb**, while failing with **aaaabbb** (see figure 6). This illustrates the approximation characteristic of the matching method: for some grammars (probably most natural language grammars) the worsening approach will at some point of permutation of the violation markers terminate and produce an exact FST representation of the grammar, while for some grammars such convergence will never happen. However, if the permutation of markers terminates and produces a functional transducer when testing each violation as described above, the FST is guaranteed to be an exact representation.

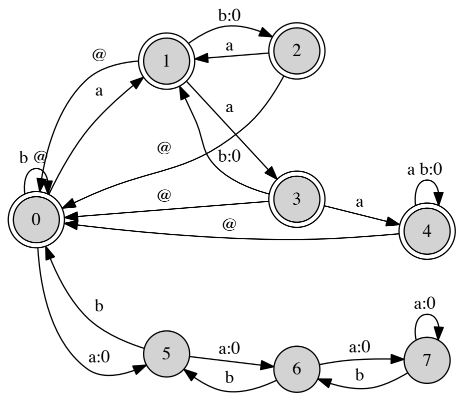


Figure 6: An non-regular OT approximation.

It is an open question if it is decidable by examining a grammar whether it will yield an exact FST representation. We do not expect this question to be easy, since it cannot be determined by the nature of the constraints alone. For example, the above four-constraint system does have an exact FST representation in some orderings of the constraints, but not in the particular one given above.

## 6 Conclusion

We have presented a practical method of implementing OT grammars as finite-state transducers. The examples, definitions, and templates given should be sufficient and flexible enough to encode a wide variety of OT grammars as FSTs. Although no method can encode all OT grammars as FSTs, the fundamental advantage with the system outlined is that for a large majority of practical cases, an FST can be produced which is not an approximation that can only tell apart a limited number of violations. As has been noted elsewhere (e.g. Eisner (2000b,a)), some OT constraints, such as Generalized Alignment constraints, are on the face of it not suitable for FST implementation. We may add to this that some very simple constraint systems, assuming a canonical GEN, and only using the most basic faithfulness and markedness constraints, are likewise not encodable as regular relations, and seem to have the generative power to encode phenomena not found in natural language. However, for most practical purposes—and this includes modeling actual phenomena in phonology and morphology—the present approach offers a fruitful way to implement, analyze, and debug OT grammars.

## References

Beesley, K. R. and Karttunen, L. (2003). *Finite State Morphology*. CSLI Publications, Stanford, CA.

Blattner, M. and Head, T. (1977). Single-valued a-transducers. *Journal of Computer and System Sciences*, 15(3):328–353.

Eisner, J. (2000a). Directional constraint evaluation in optimality theory. In *Proceedings of the 18th conference on Computational linguistics*, pages 257–263. Association for Computational Linguistics.

Eisner, J. (2000b). Easy and hard constraint ranking in optimality theory. In *Finite-state phonology: Proceedings of the 5th SIGPHON*, pages 22–33.

Ellison, T. M. (1994). Phonological derivation in optimality theory. In *Proceedings of COLING'94—Volume 2*, pages 1007–1013.

Frank, R. and Satta, G. (1998). Optimality theory and the generative complexity of constraint violability. *Computational Linguistics*, 24(2):307–315.

Gerdemann, D. and van Noord, G. (2000). Approximation and exactness in finite state optimality theory. In *Proceedings of the Fifth Workshop of the ACL Special Interest Group in Computational Phonology*.

Hammond, M. (1997). Parsing syllables: Modeling OT computationally. *Rutgers Optimality Archive (ROA)*, 222-1097.

Hulden, M. (2009a). *Finite-state Machine Construction Methods and Algorithms for Phonology and Morphology*. PhD thesis, The University of Arizona.

Hulden, M. (2009b). Foma: a finite-state compiler and library. In *EACL 2009 Proceedings*, pages 29–32.

Jäger, G. (2002). Gradient constraints in finite state OT: the unidirectional and the bidirectional case. *More than Words. A Festschrift for Dieter Wunderlich*, pages 299–325.

Kager, R. (1999). *Optimality Theory*. Cambridge University Press.

Kaplan, R. M. and Kay, M. (1994). Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

Karttunen, L. (1998). The proper treatment of optimality theory in computational phonology. In *Finite-state Methods in Natural Language Processing*.

Karttunen, L. (2006). The insufficiency of paper-and-pencil linguistics: the case of Finnish prosody. *Rutgers Optimality Archive*.

Kiparsky, P. (2003). Finnish noun inflection. *Generative approaches to Finnic linguistics. Stanford: CSLI*.

Prince, A. and Smolensky, P. (1993). Optimality theory: Constraint interaction in generative grammar. *ms. Rutgers University Cognitive Science Center*.

Riggle, J. (2004). *Generation, recognition, and learning in finite state Optimality Theory*. PhD thesis, University of California, Los Angeles.