Squeezing the Infinite into the Finite:
Handling the OT Candidate Set with Finite State Technology

Tamás Bíró

Humanities Computing, University of Groningen
t.s.biro@rug.nl

Abstract. Finite State approaches to Optimality Theory have had two goals. The earlier and less ambitious one was to compute the optimal output by compiling a finite state automaton for each underlying representation. Newer approaches aimed at realizing the OT-systems as FS transducers mapping any underlying representation to the corresponding surface form. After reviewing why the second one fails for most linguistically interesting cases, we use its ideas to accomplish the first goal. Finally, we present how this approach could be used in the future as a—hopefully cognitively adequate—model of the mental lexicon.

1 Introduction

Although very popular in linguistics, *Optimality Theory* by Prince and Smolensky (OT, [17], [18]) poses a serious problem for being computationally very complex. This fact could question the relevance of much contemporary linguistic work both for cognitive research, and language technology. Is our brain doing such a hard computation? Could language technology make us of OT models? Fortunately, things are not so bad.

Figure 1 presents the architecture of an Optimality Theoretical grammar, which consists of two modules, *Gen* and *Eval*. The input—the underlying representation (UR)—is mapped by the universal Gen onto a *set of candidates*. The candidate set, or a subset of it, reflects language typology: for each language, the language-specific Eval chooses the element (or elements) that appears as the surface form SR. Eval is a function assigning a harmony value to each candidate, and the most harmonic one will surface. Alternatively, Eval can also be seen as a pipeline in which the constraints filter out the sub-harmonic candidates. This second approach is most often used in practice, and the finite state realizations presented in this paper are also based on this vision of an OT system.

In many models advanced by theoretical linguists, the set of candidates is infinite, leading to serious questions. How could our brain process an infinite set? How could language technology make use of a model involving an infinite set?

Different approaches have been, then, proposed in order to handle an infinite candidate set. Chart parsing (dynamic programming) is probably the best known among them (chapter 8 in [19] for syllabification; [16] for implementing it to OT

A. Yli-Jyrä, L. Karttunen, and J. Karhumäki (Eds.): FSMNLP 2005, LNAI 4002, pp. 21–31, 2006.

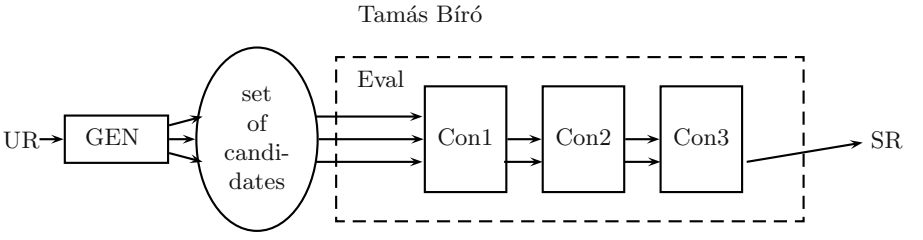

Fig. 1. The architecture of an OT grammar, which maps the underlying representation onto the surface representation. Gen is followed by the Eval module: the latter is a series of constraints, acting as filters.

LFG). It presupposes on the one hand that applying a recursive rule (usually insertion) incurs some constraint violation; and on the other, that "all constraints are structural descriptions denoting bounded structures". The interplay of these two assumptions guarantees that the algorithm may stop applying the recursive rule after a finite number of steps, for no hope is left to find better candidates by more insertions.

Alternatives include using heuristic optimization techniques. Genetic algorithms were proposed by Turkel [20], whereas simulated annealing by Bíró [5]. Such approaches involve only relatively low computational cost; nonetheless, they do not guarantee finding the optimal candidate. Simulated annealing, for instance, returns a "near-optimal" form within constant time, but you cannot know if the algorithm has found the good solution. Even though Bíró [4] argues that this algorithm models language production, one may still wish to have a perfectly working algorithm for language technology.

The present paper proposes an alternative: determining the optimal candidate by using *finite state* technologies. We first present the previous approaches to Finites State Optimality Theory (FS OT) in section 2, with an emphasis on the *matching approach*. This is followed by a new proposal in section 3, further developed into a model of the lexicon in section 4.

2 Finite State Optimality Theory

The idea of computing the optimal candidate of an OT system by building a finite state (FS) automaton goes back to Ellison [10].[1] He requires the set of candidates for a given input be a regular expression, and realizes the constraints as transducers (weighted automata) assigning violation marks. The number of violation marks assigned by a constraint to a candidate is reflected by the sum of the weights along the path representing the given candidate. Ellison subsequently proposes a series of algorithms resulting in an automaton in which "the only paths from the initial state to the final state will be optimal and define optimal candidates." This approach builds a new automaton for each input.

[1] Eisner [9] summarizes existing work on FS OT, and proposes a framework very similar to the one to be presented here.

Later work in FS OT aims at realizing the OT-system as a regular relation mapping any correct UR to the corresponding surface form. By compiling such a transducer, one would enjoy all advantages of finite state techniques, such as robustness, speed and relatively low memory requirements in both directions (production and parsing). This approach includes Frank and Satta [11] and Karttunen [15], on the one hand (the *counting approach*), as well as Gerdemann and van Noord [12], generalized by Jäger [13], on the other (the *matching approach*). The hope for a regular mapping from the UR to the SR goes back to Douglas Johnson [14].

In short, finite state approaches to OT require Gen, as well as each of the constraints be realizable—in some sense—as a regular expression or transduction. In many linguistic theories, Gen produces a regular set, as exemplified by syllabification [12] or metrical stress assignment [3]. However, many further examples, such as reduplicative morphology or numerous phenomena in syntax, are not finite state-friendly.[2] Concerning Eval, Eisner [7] [8] and Bíró [3] discuss what constraints can be realized as finite transducers. Eisner's Primitive Optimality Theory (OTP) [7] launches a challenging research program the goal of which is to model as many phonological phenomena as possible by restricting ourselves to finite state-friendly constraints.

Nonetheless, the more ambitious program of FS OT to create a transducer mapping any underlying representation to any surface representation cannot be fully carried out. Even with a FS representation of Gen and of the constraints at hand, filtering is not always possible. Frank and Satta [11] (following Smolensky and Hiller) show a simple counter-example:

Example 1: Let Gen map string $a^n b^m$ to the candidate set $\{a^n b^m, b^n a^m\}$, and let the only constraint penalize each occurrence of a. The resulting language is $\{a^n b^m | n \leq m\} \cup \{b^n a^m | n \geq m\}$, which is clearly not regular. And yet, Gen is regular, similarly to the proposed constraint.[3]

Although *Example 1* might look very artificial, its constraint, actually, is a prototype form many constraints used in linguistics. In Syllable Structure Theory [17], each segment may be parsed or underparsed, and constraint PARSE punishes underparsing. Metrical stress may be assigned to each syllable, and constraint WSP ("Weight-to-Stress Principle") requires each heavy syllable to be stressed (cf. e.g. [19]). In fact, most constraints in phonology penalize each occurrence of some local substructure a (underparsed, unstressed,...), and prefer its alternative, substructure b (parsed, stressed,...). The above example shows that all these constraints could realize a non-regular language with some specific input (output of Gen and the previous constraints) [8].

[2] Albro [1] shows how to combine a non-finite state Gen with finite state constraints.

[3] Authors differ in what is meant by a "regular constraint". For Frank and Satta [11], the set of strings incurring exactly k violation marks should form a regular set, for all k. Gerdemann and van Noord [12] use transducers inserting violation mark characters into the candidate strings. The given counter-example satisfies both of these definitions, unlike that of Jäger [13].

The *counting approach* proposed by [11] and [15] requires an upper bound on the number of violations a given candidate can incur. The *matching approach* is closer to the model proposed by linguists: it can, in theory, distinguish between any number of level of violations. Yet, in many cases, only an approximation is possible, as we shall soon see. By supposing that the length of candidates is bounded due to restrictions in the working memory, both approaches can be used in applications. Nevertheless, not only do we lose here the linguistic "point", but the size of the automata also increases quickly.

2.1 The Matching Approach

Both the *counting approach* and the *matching approach* share the agenda composed of the following three steps:

- first, formulate a finite state transducer `Gen`;
- then, formulate an optimality operator `oo`, which makes use of
- the finite state realizations `Con-i` of the constraints Con-i.

Once we have all these at hand, the grammar is realized by the finite state transducer obtained after having compiled the following expression:

$$\Big(\big(\texttt{Gen}\ \texttt{oo}\ \textsc{Con}\text{-}1\big)\ \texttt{oo}\ \textsc{Con}\text{-}2\Big)\ldots\ldots\ \texttt{oo}\ \textsc{Con}\text{-}\mathrm{N} \tag{1}$$

From now on, we suppose that a FS transducer `Gen` is given. The task is to formulate the optimality operator `oo`; the latter will then determine what realization `Con-i` of each constraint Con-i is required.

The key idea of the *matching approach* is to build a set $Worse(Input, \textsc{Con})$ that includes all sub-harmonic candidates of the input *Input* with respect to the constraint Con, as well as possibly other strings; but excludes all harmonic candidates. This set will then serve as a filtering set in the definition of the optimality operator `oo`:

$$Input\ \texttt{oo}\ \textsc{Con} := Input\ \texttt{o}\ Id\ \overline{}^{\,Worse(Input,\textsc{Con})} \tag{2}$$

Here, the identity transduction filters out the elements of $Worse(Input, \textsc{Con})$, only the elements of its complement may become outputs. This approach is a straightforward implementation of the idea behind OT—supposing that the set $Worse(Input, \textsc{Con})$ can be constructed.

Without referring to violation marks, Jäger [13] proposes to realize a constraint Con with a transducer $\texttt{Con}_J$ that directly will create the filtering set, a superset of the sub-harmonic candidates. The candidate w is mapped onto a set containing: (1) all candidates of $Gen(Gen^{-1}(w))$ that are less harmonic than w; and (2) possibly strings not belonging to $Gen(Gen^{-1}(w))$. Now, the range of $Input\ \texttt{o}\ \texttt{Con}_J$ contains all sub-harmonic elements of *Input* but no harmonic ones. Hence, it can serve as the filter in (2):

$$Input\ \texttt{oo}\ \textsc{Con} := Input\ \texttt{o}\ Id\ \overline{}^{\,Ran(Input\ \texttt{o}\ \texttt{Con}_J)} \tag{3}$$

The draw-back of Jäger's approach is the difficulty of defining the required transducers corresponding to constraints in linguistics. Even worse, it is not

possible very often—otherwise a finite automaton could accept a non-regular language in Example 1. A constraint that is finite-state in Jäger's sense would lead automatically to a finite-state realization of OT.

It is more fruitful to realize constraints with transducers assigning violation marks to the strings, as done by Ellison [10], Gerdemann & v. Noord [12] and Eisner [9]. One can simply construct a finite state transducer that inserts a special violation mark symbol after each disfavored substructure—supposing that the latter are simple enough to be recognized with an FSA, which is usually the case. In this sense, most constraints are finite-state.[4] Can we make any use of these transducers?

Suppose that for constraint CON, a transducer `Con` exists that introduces the required number of violation marks into any candidate string. Importantly, we only know that the output of `Con` includes the correct number of violation mark symbols, but we do not know how these symbols are dispersed in the string. Furthermore, let `remove_viol` denote the transducer removing all violation mark symbols from its input. The only task now is to define the transducer `make_worse`, and then we can rewrite definition (2) as follows:

$$\textit{Input} \texttt{ oo } \textsc{Con} := \textit{Input} \texttt{ o Con o } \\ \texttt{o } \textit{Id}\,\overline{\textit{Ran}(\textit{Input} \texttt{ o Con o make\_worse})}\texttt{o remove\_viol} \qquad (4)$$

Now, we have to define `make_worse`. Imagine that the set of candidates *Input* entering the constraint filter CON includes only candidates that are assigned N violation marks, or more. Let us add at least one violation mark to each of them: we thus obtain a set of strings with not less than $N + 1$ violation marks. If we ignored the characters other than the violation marks, this latter set could simply be used for filtering, because only the candidates of the input set with the least (namely, N) violation marks are not element of the filtering set thus constructed. Consequently, the finite state transducer `make_worse` will have to add any positive number of extra violation marks to the input, using a finite state transducer `add_viol`.

Nevertheless, we cannot ignore the characters in the candidate strings. The filtering set will not yet include all the sub-harmonic candidates, because the candidate strings vary not only in the number of violation marks. The different elements of the candidate set have to diverge from each other, for instance, in the position of parsing brackets. Most probably, the violation marks should also be permuted around the segments of the strings.

Therefore, we redefine `make_worse`: besides adding extra violation marks (`add_viol`), it will delete all characters that are not violation marks using the simple transducer `delete_char`, and then insert any new characters (transducer `insert_char`) (cf. (4) and (5) to the formalism in Eisner [9]):

$$\texttt{make\_worse} := \texttt{delete\_char o add\_viol o insert\_char} \qquad (5)$$

The range of *Input* `o Con o make_worse` is now the set of *all* strings with more violation marks than the minimal in the range of *Input* `o Con`: all candidates

[4] Quadratic alignment constraints assigning a number of violation marks growing faster than the length of the string, are not regular even in that sense [7] [8] [3].

to be filtered out, further uninteresting strings, but no candidates to be left in. This fact guarantees that the harmonic candidates, and only they will survive the filtering in definition (4). Or almost.

Yes, we still face a problem. Suppose that underlying representation W_1 is mapped by *Inp* `o Con` to candidates involving at least N_1 violation marks, and w_1 is an optimal one. Further, suppose that UR W_2 is mapped to candidates containing N_2 violation marks or more, with an optimal w_2. Suppose also that $N_1 < N_2$. Because W_1 is in the domain of *Inp* `o Con o make_worse`, the latter's range will include all strings with more than N_1 violation marks, w_2 among them. Consequently, all candidates corresponding to W_2 will be filtered out, and W_2 is predicted to be ineffable, without any corresponding output.

Gerdemann and van Noord [12], therefore, define `make_worse` such a way that it will keep the underlying material unchanged. Suppose that what Gen does is nothing but to add some extra material, like parsing brackets. In such a case, deleting and reintroducing only the brackets introduced originally by Gen ensures that different underlying representations cannot interfere:

$$\texttt{make\_worse} = \texttt{add\_viol o del\_brackets o ins\_brackets} \tag{6}$$

Nonetheless, a new problem arises! Let the underlying representation `abab` yield two candidates, namely `a[b]ab` and `[a]b[a]b`. Let the constraint insert a violation mark `@` after each closing bracket, so the set entering `make_worse` is {`a[b]@ab`, `[a]@b[a]@b`}. By applying the operation `make_worse` as defined in (6), we get among others the strings `[a]b@a[b]@` or `[a]@b@[a]b`; but not `[a]@b[a]@b`, the candidate to be filtered out. An extra operation is, therefore, required that will permute the violation marks: in our case, we need to remove the `@` between the first `b` and the second `a`, and simultaneously insert a `@` following one of the `a`'s; the second violation mark will be inserted by `add_viol` after the other `a`.

The real problem arises when one has to compare two candidates, such that the first one may have an unbounded number of violation marks in its first part, while the second one any number of violation marks in its last part. This happens in Example 1, and in the many analogous linguistic applications. Then, the transducer should have to keep track of the unbounded number of violation marks deleted at the beginning of the string, before it reaches the end of the string and re-inserts them. That is clearly a non-finite state task.

If the transducer permuting the marks `perm` is able to move one violation at the same time, then the following definition of `make_worse` yields an exact OT-system only for the case where not more than n violation marks should be moved at once:[5]

[5] The same transducer can move a second violation mark after having accomplished its task with the first one. Note that such a finite-state friendly case can theoretically result from the interplay of Gen and the previously ranked constraints; and not only from restricting the number of violation marks assigned, due, for instance, to a bound in the length of the candidates. Further research should reveal whether the linguistically relevant cases are indeed finite-state, or languages do produce extreme candidate sets, such as the one in Example 1. See the research line launched by Eisner's OTP [8].

$$\texttt{make\_worse} := \texttt{add\_viol o del\_brackets}$$
$$\texttt{o ins\_brackets o perm}_1 \texttt{ o perm}_2 \texttt{ o...o perm}_n \qquad (7)$$

Thus, we have run into the "permute marker problem": only an approximation is offered by Gerdemann and van Noord for the general case. Besides, introducing `perm` n times makes the automaton enormous.

3 Planting the Input into the FST

The matching approach, as proposed by Gerdemann and van Noord [12], has two main advantages over its competitors. First, it does not require the number of levels of violations to be finite, as opposed to the counting approach. Second, it makes use of transducers assigning violation marks to the strings, which is much easier to realize than the transducers in Jäger's generalized matching approach.

Example 1 has shown that there is no hope for solving the "permute marker problem" in general. Can we still bring out the most of the the counting approach? Maybe by stepping back to the lesser goal of Ellison: compiling an automaton to each word, instead of creating a general transducer mapping any underlying representation to the corresponding surface form? This is bad news for people believing in FS OT (despite Example 1), and yet, it opens the way to a new model of the mental lexicon.

We have seen that the radical definition of `make_worse` in (5) creates a problem: the candidates corresponding to some underlying representation may discard *all* candidates of another underlying representation. The solution by [12], that is, to modify `make_worse` as (6) and (7), led to the "permute marker problem". Another solution is to keep the more radical `make_worse` transducer, as defined in (5), for the definition (4) of the optimality operator; but, simultaneously, to introduce a filter at the beginning of the pipeline (or, into Gen, as Ellison did). By restricting the domain of the transduction, this filter—an identity transduction on a singleton—ensures that no other input disturbs the computation. So, for hierarchy Con-1 $\gg$ Con-2 $\gg$ . . . $\gg$ Con-N, and for *each* underlying representation W we have to compile the following regular expression:

$$\Big(\Big(\big(Id_{\{W\}} \texttt{ o Gen}\big) \texttt{ oo } \textit{Con-1}\Big) \texttt{ oo } \textit{Con-2}\Big) \texttt{ ... oo } \textit{Con-N} \qquad (8)$$

Let us prove the correctness of this approach:

Theorem: *Let all constraints* Con-*i be represented by a transducer* `Con-i` *inserting violation marks, and let*

$$\texttt{make\_worse} := \texttt{delete\_char o add\_viol o insert\_char}$$

$$\textit{Input} \texttt{ oo } \textsc{Con} := \textit{Input} \texttt{ o Con o}$$
$$\texttt{o } Id\,\frac{}{Ran(\textit{Input o Con o make\_worse})} \texttt{ o remove\_viol}$$

If for hierarchy $H = \langle$ Con-1 $\gg$ Con-2 $\gg$ . . . $\gg$ Con-N $\rangle$ *and underlying representation* W*,*

$$OT_0 := Id_{\{W\}} \texttt{ o Gen} \qquad\qquad OT_i := OT_{i-1} \texttt{ oo Con-i}$$

then the range of OT_N *is the set of outputs with respect to underlying representation* W *and ranking* H*.*

Proof: By induction on the number of constraints N. For $N = 0$: by definition, the range of OT_0 is the candidate set corresponding to W.

For $N = k > 0$: We suppose that the range of OT_{k-1} is the set of candidates returned by the part of the pipe-line before constraint Con-k. We have to show that OT_k is the optimal subset of OT_{k-1} with respect to constraint Con-k.

Let m denote the number of violation marks assigned by constraint Con-k to the most harmonic candidates in the range of OT_{k-1}. By the definition of `make_worse`, the range of OT_{k-1}`o Con-k o make_worse` includes all strings (words and non-words) with more than m violation marks, and only them. Thus, the identity transduction in the definition of the optimality operator `oo` transduces all strings with no more than m marks, and only them. Consequently, the range of OT_k will include exactly those elements of the range of OT_{k-1} that violate Con-k m times. □

We have stepped back to the less ambitious proposal of Ellison [10]: we compile a regular expression for each input. One first formulates a finite transducer realizing Gen, as well as transducer adding each candidate string the same number of violation marks as the constraints of the linguistic model do. Then, the range of the regular expression (8) has to be compiled and read. Compilation—even if it is a computationally complex task, primarily due to the set complement operation—can be done automatically, with any package handling regular expressions.

Is stepping back to a ten-year old result something worth writing a paper on? The good news, however, is that the approach proposed opens new perspectives about a finite-state model of the lexicon.

4 Modeling a Complex Lexicon

Many linguistic phenomena can be described by using "co-phonologies", by referring to exceptions or to "minor rules". The discussion about the interaction between morphology and phonology (here we just refer to the well-known "past tense debate") has also affected OT [6]. On-going and further research shall analyze whether a finite-state Optimality Theoretical approach has something interesting to say about the issue. In the remaining space of the present paper, we shall present the possible first steps of such a research line.

Equation (8) allows for generalization. If *SL* is a subset of the lexicon, the following expression will define the surface representation of elements of *SL*:

$$\Big(\Big(\big(Id_{SL} \circ \texttt{Gen}\big) \circ\!\circ \mathit{Con\text{-}1}\Big) \circ\!\circ \mathit{Con\text{-}2}\Big) \ \ldots \circ\!\circ \mathit{Con\text{-}N} \tag{9}$$

Note that elements of SL may "extinguish" each other: if a candidate w_1 corresponding to some element $W_1 \in SL$ incurs less violation marks than the optimal candidate corresponding to another W_2, then no output is returned for W_2. Therefore, SL should be the set of "analogous" words in the language.

The phonology of the language is then modeled thus:

$$\bigcup_i \Big(\Big(\big(Id_{SL_i} \circ \texttt{Gen}\big) \circ\!\circ \mathit{Con}_{i_1}\Big) \circ\!\circ \mathit{Con}_{i_2}\Big) \ \ldots \circ\!\circ \mathit{Con}_{i_N} \tag{10}$$

The lexicon is composed of subsets of words. Each subset SL_i is associated with a hierarchy $\mathit{Con}_{i_1} \gg \mathit{Con}_{i_2} \gg \ldots \gg \mathit{Con}_{i_N}$. Different subsets may be associated with the same hierarchy, but cannot be unified, unless some words are erased from the language, as explained. Yet, once we have this structure, nothing prohibits us to associate different subsets with different hierarchies.[6]

Until now, if the sub-lexicons are finite, the complex expression in (10) is compiled into a simple finite set of UR-SR pairs. Yet, we claim that expression (10) together with linguistically motivated constraints have a cognitive explanatory value by restricting the possible lexicons: *what* UR-SR mappings are thinkable?

Additionally, our on-going research tries to introduce some sort of *generalization* into the sub-lexicons. Let the *hash* $a\#$ of an element a of the alphabet be the following concatenation:

$$\texttt{a\#} := \ \texttt{pc*} \ \texttt{|} \ \{\texttt{a, pc}\} \ \texttt{|} \ \texttt{pc*}, \tag{11}$$

where `pc` is a *punished change*: whatever character followed by a special *punishment symbol*. Thus, the *hash* of a character is its generalization: you can replace it, you can add anything before and after it, but whatever change introduced is marked by a *punishment symbol*. In the next step, the generalization $W\#$ of a memorized word W is the concatenation of the hash of its characters in the corresponding order. Last, we propose that if a learned (memorized) word $W \in SL_i$, then also $W\# \in SL_i$.

With this generalization, the input of the grammar model (10) can also be an unseen word—yet, not *any* unseen word. The punishment symbols measure the "distance" of the input from previously memorized, "similar" words, in terms of letter changes. An input may match the hash of several learnt lexical items, possibly in different sublexicons, in which case more outputs are generated

[6] One can speculate about how co-phonologies have emerged in languages. Decomposing the lexicon into sub-lexicons is necessary, otherwise some words would be ineffable, *i.e.*, unpronounceable. Thus, an acquisition model should be able to open new sub-lexicons. Then, as the constraint pipe-line is connected to each sub-lexicon independently, nothing prohibits constraint re-ranking for certain sub-lexicons. A prediction is that language varieties differ in the constraint ranking corresponding exactly to these sub-lexicons, which reminds us the similar proposal of [2].

simultaneously in various pipe-lines.[7] These symbols are preserved during the transduction, and the output with the minimal number of punishment symbols is the predicted form. We can use a FS OT-style filter on the punishment symbols, and we obtain a sort of memory-based learning. The consequences of this proposal, learnability issues and its possible cognitive relevance are subject to future research.

5 Conclusion

In the introduction, we have raised the problem of how one can handle the infinite set of OT candidates appearing in contemporary linguistic work within the framework of a plausible psycholinguistic model or a working language technology application. In this paper, we have proposed a new way of using finite state technology in order to solve that problem. We have reviewed why it is not possible to create a FS transducer realizing an OT-system in general, even if Gen is a regular relation, and constraints are also regular (at least in some sense). Subsequently, we have proposed to make the *matching approach* exact by planting a filter before Gen.

This way we have obtained an alternative to Ellison's algorithm [10]. By compiling (8) for each input separately, we can calculate the optimal element of the possibly infinite candidate set. Finally, we have shown how this result can be generalized into a model of the lexicon, yet further research has to prove the cognitive adequateness of such a model. For instance, does it account for observed morpho-phonological minor rules? Preliminary results show that different hierarchies are compiled in significantly different time. If so, do less frequently attested language typologies correspond to rankings more difficult to compile?

The present paper hope to have paved the way for such future research.

Acknowledgments

I wish to acknowledge the support of the University of Groningen's Program for High-Performance Computing. I also would like to thank Gosse Bouma and Gertjan van Noord for valuable discussions.

References

1. D. M. Albro. Taking primitive Optimality Theory beyond the finite state. In *Eisner, J. L. Karttunen and A. Thériault (eds.): Finite-State Phonology: Proc. of the 5th Workshop of SIGPHON*, pages 57–67, Luxembourg, 2000.

[7] The input may turn out to be ineffable in some—or all—of the pipe-lines. Importantly, constraints and the definition of the hash operation should be such that $W\#$ may not render W ineffable in its own subset. Many, yet not all constraints assign an equal or higher number of violation marks to the best candidate of a longer input. This is also the reason why a *punished change* does not include an empty string, allowing for shortening—at any rate, it is quite rare that longer forms influence shorter forms by analogy.

2. A. Anttila and Y. Cho. Variation and change in optimality theory. *Lingua*, 104(1-2):31–56, 1998.
3. T. Bíró. Quadratic alignment constraints and finite state Optimality Theory. In *Proc. of the Workshop on FSMNLP, at EACL-03, Budapest*, pages 119–126, also: ROA-600,[8] 2003.
4. T. Bíró. When the hothead speaks: Simulated Annealing Optimality Theory for Dutch fast speech. presented at CLIN 2004, Leiden, 2004.
5. T. Bíró. How to define simulated annealing for optimality theory? In *Proc. of the 10th Conference on Formal Grammar and the 9th Meeting on Mathematics of Language*, Edinburgh, August 2005.
6. L. Burzio. Missing players: Phonology and the past-tense debate. *Lingua*, 112:157–199, 2002.
7. J. Eisner. Efficient generation in primitive Optimality Theory. In *Proc. of ACL 1997 and EACL-8, Madrid*, pages 313–320, 1997.
8. J. Eisner. Directional constraint evaluation in Optimality Theory. In *Proc. of COLING 2000, Saarbrücken*, 2000.
9. J. Eisner. Comprehension and compilation in Optimality Theory. In *Proc. of ACL 2002, Philadelphia*, 2002.
10. T. M. Ellison. Phonological derivation in Optimality Theory. In *COLING-94, Kyoto*, pages 1007–1013, also: ROA-75, 1994.
11. R. Frank and G. Satta. Optimality Theory and the generative complexity of constraint violability. *Computational Ling.*, 24(2):307–315, 1998.
12. D. Gerdemann and G. van Noord. Approximation and exactness in finite state Optimality Theory. In *J. Eisner, L. Karttunen, A. Thriault (eds): SIGPHON 2000, Finite State Phonology*, 2000.
13. G. Jäger. Gradient constraints in finite state OT: The unidirectional and the bidirectional case. *ROA-479*, 2002.
14. D. C. Johnson. *Formal Aspects of Phonological Description*. Mouton, The Hague [etc.], 1972.
15. L. Karttunen. The proper treatment of Optimality Theory in computational phonology. In *Finite-state Methods in NLP*, pages 1–12, Ankara, 1998.
16. J. Kuhn. Processing optimality-theoretic syntax by interleaved chart parsing and generation. In *Proc. of ACL-2000, Hongkong*, pages 360–367, 2000.
17. A. Prince and P. Smolensky. Optimality Theory, constraint interaction in generative grammar. RuCCS-TR-2, ROA Version: 8/2002, 1993.
18. A. Prince and P. Smolensky. *Optimality Theory: Constraint Interaction in Generative Grammar*. Blackwell, Malden, MA, etc., 2004.
19. B. Tesar and P. Smolensky. *Learnability in Optimality Theory*. The MIT Press, Cambridge, MA - London, England, 2000.
20. B. Turkel. The acquisition of optimality theoretic systems. m.s., ROA-11, 1994.

[8] ROA stands for *Rutgers Optimality Archive* at `http://roa.rutgers.edu/`.