Learnability in Optimality Theory and Finite-State Automata

Tamás Bíró

November 1, 2002

Contents

Chapter 1

Introduction

In this paper I shall combine three current research fields within computational linguistics. The first one is learnability. This refers to a special aspect of linguistics: the adequateness of a linguistic theory from the point of view of acquiring the language. A theory should possibly lead to some learning algorithm of the language, and it is even better if this learning algorithm can somehow simulate real life cases of language acquisition (L1 and / or L2).

The second component of this paper is Optimality Theory (OT; cf. Prince & Smolensky, 1993). This model has been extremely popular in the last decade in most fields of linguistics, especially in phonology. The basic reference about learnability in OT is Tesar & Smolensky [2000]. As we were told at ESSLLI'02: "*A theory without exemples is like a car without an engine: it might be beautiful, but does not bring you anywhere*". Therefore I will work on stress assignment, a classical paradigm in OT (*c.f.* Tesar & Smolensky, chapter 4).

The last ingredient is Finite State technology. Karttunen [1998], Frank & Satta [1998] and Gerdemann & van Noord [2000] have presented the way an Optimality Theoretical model can be implemented using Finite State Transducers (FSTs). FSA Utilities (van Noord [1997], van Noord [1999], van Noord & Gerdemann [1999]) will serve as the computational tool for implementing any OT model, and therefore this investigation can help in gauging the capacities of finite state Optimality Theory.

It will turn out that when using real language data, the story is not as simple as presented in Tesar & Smolensky [2000], chapter 4. The "noise" appearing in the data will motivate a new approach to the lexicon. From a learning point of view we cannot neglect irregular data (how could the algorithm know in advance which forms are irregular?), and we would like to avoid overfitting, as well, *i.e.* learning a too complicated model that could account for all the data. I rather propose to speak about the learnability of a complex lexicon, in which all lexical items are associated with one (or more) co-grammars (hierarchies, in the case of OT). I believe that this approach should replace the classical paradigm of trying to find a uniform model which accounts for all data, but is too complex and far from being universal. (In fact in most linguistic models the OT constraints tend to become too language specific, and this contradicts the basic ideas of OT.) The crucial question then is to make this complex lexicon as simple as possible.

To summarize, the outcome of this investigation should be the followings: a learning algorithm usable for word stress analysis, within an OT framework; a Finite State Optimality model for stress, espacially for Dutch; side results about Finite State Optimality (for example: what constraints can be realized with finite state transducers?); and finally a new model for the lexicon.

In the 2nd chapter I shall summarize in a nutshell what should be known about these ingredients. In Chapter 3 I shall formulate the problem of learning (Dutch) stress, and present to way to solve it. Chapter 4 will present my preliminary results, including a discussion about contraints that cannot be realized using FSTs. The question of possible lexical models will be addressed in Chapter 5. At the moment, this is just mere speculation. Finally my PhD proposal shall be discussed in Chapter 6, followed by my time planing in Chapter 7.

Chapter 2

Optimality Theory, Learnability and Finite State Technology

2.1 Basics of OT

Optimality Theory has been a leading paradigm in linguistics, especially in phonology, since its appearance in 1993 (Prince & Smolensky [1993], from here on I will refer to it as P&S). Another key reference will be Tesar & Smolensky [2000] (T&S from now on).

Its main claims can be summarized as follows:

- The grammar is composed of two parts: the *Gen* modul generates a (possibly infinite) set of *candidates* out of the given underlying representation, while the *Eval* modul determines the optimal element of this set. The optimal element will be the grammatical form (the surface representation). [1]

- There is an universal set of *constraints*, each of them assigning a given number of violations to each of the candidates.

[1] There is here a couple of points for possible confusion. First, the term "derivation" is avoided by people within the OT paradigm, when refering to the mapping between the underlying representation and the surface representation. This term is used only to refer to the derivations within the pre-OT generative frameworks, whereas OT sees itself as being a "non-derivational" framework. The term "production-directed parsing" is used instead by T&S. Futhermore the term "generating" is mostly used to refer to production of the set of candidates by Gen.

Another point of possible confusion is the use of the term "input". The underlying representation is the "input of the production-directed parsing", while the "input of the learning algorithm" is a data, a surface form, corresponding to one or more of the possible outputs of the production-directed parsing.

- For each constraint these violations assigned define a strict partial order called *harmonic ranking* on the set of the candidates.
- For each language there is a (fully) ranked hierarchy (*i.e.* a sequence of application) of these constraints. The way Gen determines the optimal candidate is dependent only upon the ranking of these constraints. Informally, the highest ranked constraint filters out the candidates for which there is another candidate being assigned less violation marks (being "more harmonic" according to harmonic ranking). Then the second highest ranked candidate filters out further elements of the remaining set of candidates, using the same method, etc.

The **Richness of the Base** principle (P&S, section 9.3, T&S p. 30 and 75) says that *all inputs are possible in all languages, and distributional and inventory regularities follow from the way the universal input set is mapped onto an output set by the grammar.*

In other words, cross-linguistics varieties are exclusively due to the different rankings of the same universal constraints. It is not always true, however, that different rankings lead to different languages. Furthermore, not all rankings represent possible languages, if one introduces the notion of **universal subhierarchies** (P&S, ch. 9, p. 198; T&S p. 47). The latter imposes universal restrictions on possible constraint rankings.

2.2 Finite State Automata and their application to OT

Classical generative morphology and phonology, based on Noam Chomsky and Morris Halle's *The Sound Pattern of English* [1968], consisted of context sensitive rules of the form *x -¿ y / a _ b*, but it has been early proposed that phonological rules do not need such powerful tools (the dissertation of C. Douglas Johnson: *Formal Aspects of Phonological Description* [1972]). In the 80s, this idea led to early investigations at Xerox in using finite tools in phonology: the classical source for using Finite State Automata in this field is Kaplan & Kay [1994].

Kimmo Koskenniemi introduced the term "two-level rules" in 1983, that is a finite-state transducer mapping between the underlying form and the surface form. A major step, giving a big impetus to the researches in Finite State technology, was when Koskenniemi made their implementation freely available in 1996, under the name PC-KIMMO (Antworth [1990] and also the web site at `http://www.sil.org/pckimmo/`)). Further works were done in the 90s by Karttunen (*e.g.* Karttunen et al. [1996]).

For an overview of the history of Finite State morphology, look at Karttunen & Beesley [2001].

The first steps to use this technology on Optimality Theory are represented by Frank & Satta [1998] and Karttunen [1998]. The next step, that will be the starting point of my investigation is Gerdemann & van Noord [2000].

The idea is to see both *Gen* and *Eval* as being Finite State Transducers (FSTs). The highly non-deterministic FST that implements Gen will output the set of candidates corresponding to the underlying form that it receives as its input.

Then we compose this FST with the FSTs representing the constraints in a serial way. These latter ones function as filters, outputting only the harmonic subset of their input set, and "killing" the candidates that are not optimal for the given constraint. The output of the last transducer, corresponding to the lowest ranked constraint, will be then the optimal for of the given grammar.

The FST representing each constraint as a filter consists of two components. The first one (usually called `mark_OT_violation(Constraint)`) is a deterministic FST, characteristic to the constraint, adding the required number of violation marks to each candidate (usually the `@` symbol will be used for the violation marks). The second component is the same for all constraints, and is in charge of filtering out those candidates that have more than the minimal number of violation marks.

There has been two approaches for realizing this. The approach of Karttunen [1998] is called the "counting approach", and is based on the so-called "lenient composition", defined by using what they call "priority union". In this approach a constraint would filter out the candidates having more than zero violation marks if there is a candidate having no violation marks, otherwise all are let further. Therefore additional steps are needed for the same constraint, filtering out respectively those having at least two violation marks if there is at least one candidate that has less violation marks; another one filtering out those having at least three violation marks if there is a candidate having less, etc. This approach can be exact, and not only an approximation, only if there is a maximal number of violations that a candidate can be assigned (*e.g.* because there is a maximal length for words).

The approach by Gerdemann & van Noord [2000] is called the "matching approach". This latter filters out the candidates having more than minimal violation marks by adding a further violation mark to the set of candidates, and then taking the difference of the two sets. This approach works only if all non-optimal candidates can be be produced by adding one violation mark to a more harmonic candidate. If this is not the case, further tricks should be used (like the `permute_marker` operator), with a given precision.

2.3 Learnability of OT

Since Gen and the set of constraints [2] are universal, the learning task in an OT framework means to find the (or one) hierarchy that would describe the set of data.

T&S have given an algorithm called Constraint Demotion (CD) that can do that this task pretty efficiently. As they show it (p. 99, theorem 7.29), CD

[2]As well as the input set according to the Richness of the Base Principle, but the importance of this might depend upon the concrete linguistic problem in question.

converges to a hierarchy generating the language after no more than $N(N-1)$ informative examples, where N stands for the number of constraints.

The major problem I've found with this algorithm is the fact that it uses stratified hierarchies. A *stratified domination hierarchy* is (*c.f.* p. 37, for a more exact definition *c.f.* p. 91) a list of strata, each of them being a set of constraints. Harmonic ordering is defined by "collapsing all the constraints of a stratum": *minimal violation with respect to a stratum is determined by the candidate incurring the smallest sum of violations assessed by all constraints in the stratum.* As we shall see in the next section, this will lead us to serious problems.

The only information we can say is that applying the CD algorithm results in a *stratified hierarchy* that produces a language including the data set being the input of the learning algorithm. In fact one underlying representation (one input to Gen) can result in a number of outputs (alternating surface forms, according to the theory), [3] and possibly only some of them appear among the training data of the learning algorithm. [4] A further problem is that it has been a general assumption that languages should be modelled by using fully ranked hierarchies (*c.f.* T&S p. 24), and – as we shall see it in the next section – the relation between the winner set of a stratified hierarchy and the winner sets of its refinements (T&S, p. 92, definition 7.6) is far not obvious.

The version of CD that one can easily implement is *Error Driven Constraint Demotion* (EDCD, p. 50). This algorithm compares the *winner*, that is the item taken from the positive dataset, with the *loser*, that is the optimal candidate generated from the underlying form of the winner, using the actual constraint hierarchy. If they coincide, the actual learning data is said to be not informative, otherwise some constraints can be demoted, leading to a new hierarchy being closer to the one to be learned.

Another algorithm described by Tesar and Smolensky (*c.f.* Chapter 7.6) is *Recursive Constraint Demotion* (RCD). Its advantage is that it can determine if the data-set is inconsistent (p. 110), a situation that is very likely to happen in the cases we will soon describe (either when using RIP, or having noisy data,

[3]This can be the case also with fully ranked hierarchies. But for fully ranked hierarchies such a case can only occur when each of the winner candidates is assigned exactly the same violations. This would mean that the constraints are not able to distinguish between some candidates, and consequently these equivalent candidates are not real "multiple solution" from an OT point of view. Such a case can be rulled out only by restricting Gen or by adding new constraints, both being violations of the supposed universality of these componants. In the case of stratified domination hierarchy however, on the other hand, we may have real, not equivalent alternates.

[4]Tesar and Smolensky would propose to use these equally harmonic alternative outputs as negative data (supposing we know that there are no alternating surface forms in the language) for a further learning step of CD. This is what I also did in my algorithm. But this will be not always sufficient for us when using data not comming form one target hierarchy. Suppose we have learned a stratified hierarchy from a subset of our data in the sense that all elements of this subset are the unique outputs of the model for the relevant underlying representations. In this case, when applying the given hierarchy to other learning data, it will sometimes turn out that the model outputs the correct form among other incorrect ones. If we used this fact as a trigger for further learning step, the new, refined hierarchy could possibly not model anymore some of the original data it did.

or if the language cannot be covered with one grammar).

In fact both EDCD and RCD presuposes that the set of learning data are fully parsed, in the sense that the data (the *overt form*) is equivalent to exactly one of the candidates generated by Gen. This is the case for syllable structure, the *par excellence* paradigm in OT since P & S, but is not the case for stress assignment based on feet. In other words, different parses (different structural descriptions) can lead to the same surface (overt) form. For instance a three-syllabic word with a stress on its middle syllabe ($\sigma\ \boldsymbol{\sigma}\ \sigma$) corresponds at least to two parses: either the first two syllables are parsed into a right-headed (iambic) foot ($(\sigma\ \boldsymbol{\sigma}\ )\ \sigma$), or the last two syllables are footed into a left-headed (trohaic) foot ($\sigma\ (\ \boldsymbol{\sigma}\ \sigma\ )$).

Therefore an extra step is needed in this case. Tesar and Smolensky call it *Robust Interpretive Parsing* (RIP): this is choosing the optimal (harmonic) form (with respect to a given hierarchy) in a similar way to the standard "production-directed parsing" of OT, but taking into consideration only the set of candidates that correspond to the surface form of the given data. This process is said to be robbust because it assigns a description to an overt form even when there is no description matching that overt form that is grammatical (i.e. optimal for its underlying form) according to the current ranking (p. 58).

In a case like learning stress assignment therefore an itterative algorithm is needed: in the first step a parse (a structural description) is assigned to each overt form of the data set, presupposing an initial constraint hierarchy. Then a new hierarchy is tried to be learnt by Error Driven Constraint Demotion, based on this set of robbustly parsed data. As it is not garanteed that this new ranking can fully account for the set of data, the RIP process is used again to create a new set of parsed data before going once again into EDCD, etc. There is no garantee that this RIP/CD algorithm will converge to a hierarchy, although it did in a relatively high percentage of Tesar and Smolensky's experiments (cf. pp. 68-70). Chapter 4.4 of T&S presents the cases where the algorithm can fail.

2.4 Problems with unranked hierarchies

Tesar & Smolensky propose the notion of stratified hierarchies (p. 37, p. 91) as a working tool when aiming to reach the fully ranked target hierarchy. They propose that the violations should be sumed up within one stratum before choosing the set of the optimal candidates surviving the given stratum. They propose this by hoping that the fully ranked target hierarchy should be a refinement (*cf.* p. 92, definition 7.6) of the algorithm's output hierarchy. In fact this is not the case, at least if constraints can assign multiple violations (which is the case in most applications, including those presented in T & S). Let ut look at the following example:

	c_1	c_2	c_3
A	*	*	
B	**		*
C		**	*

Figure 1.

Let us suppose that we have three candidates, A, B and C, violating the three constraints as shown by Fig. 1. It is clear that out of the six possible full rankings of these constraints, candidate A will be the winner one if and only if c_3 is ranked higher than the two others. If c_2 is the highest ranked one, B will be the optimal candidate, whereas C is the best if c_1 is the first in the hierarchy. Consequently, if the input data of the learning algorithm is A, the target hierarchy is either $c_3 >> c_1 >> c_2$ or $c_3 >> c_2 >> c_1$. In fact, if the initial hierarchy $\mathcal{H}_0$ of the CD algorithm contains all of the three rankings in the same stratum, A will turn out to be the only output of the given hierarchy already in the first step, therefore the algorithm will immidiately stop. This result is not bad in the sense that the fully ranked target hierarchy is indeed a refinement of the output hierarchy of the algorithm. But in order to find it, we should try out all possibilities, an additional task that makes actually the CD algorithm totally unnecessary. Furthermore, if for some reasons the initial hierarchy is $c_1, c_2 >> c_3$ (e.g. because we want initially the markedness constraints to dominate the faithfullness constraints, or another learning data has previously led the algorithm into this state), candidate A will again turn out to be the optimal candidate, but it is clear that none of the refinements of this hierarchy would return A.

A widely accepted alternative approach for defining unranked constraints is proposing that the output of a stratified hierarchy is the union set of the outputs of all its refinements. But this approach is highly unpractical for learning purposes, because if the initial hierarchy is totally or almost totally unranked, then the number of all its fully ranked refinements to be tried out can be extremely high. In fact, in many cases there won't be too much advantages of using the CD algorithm as compared to a blind search.

Unless we do it the following way: we never calculate all the outputs of a given stratified hieachy, but take only a random refinement of it. If the candidate it returns as the optimal one coincides with the winner (the input data of the learning algorithm), we stop, since we have found one fully ranked hierarchy that returns us the winner. We don't care about seeking other possible hierarchies, or a more general class of solutions. On the other hand, if this refinement returns another optimal form, different from the winner, this latter can be used as an informative competitor for demoting some constraints.

Another possibility is not to use stratified hierarchies, but fully ranked ones. The initial hierarchy is already a (maybe randomly chosen) fully ranked one, and in each step when we have to demote a given constraint c_1 below an other constraint c_2, the way to do that is to insert c_1 between c_2 and the one imidiately below it. In fact, in a situation when both c_1 and c_2 are to be demoted below c_3, it will be arbitrary whether the algorithm will result in $c_3 >> c_1 >> c_2$ or $c_3 >> c_2 >> c_1$.

Chapter 3

Formulating the problem: learning stress

Learnability researches usually presuppose that the set of data can be described with one, consistent model. What happens, as it is the case in many occasions, when the data from the language can be put apart into a number of classes, each of them describable with a different model. Language acquisition observations (the so-called ”U-shape development”, when the behaviour changes from good performance to poor performance, before improving again, *cf.* *e.g.* Harley [2001:96 and 125]) have proven the adequateness of the use of so-called ”minor rules”.

The typical example is when some of the English speaking children learn the correct past tense form of the verb *bring* in four steps. In the first one these children use the correct form *brought*, since that is the one they have heard. Due to their enriched vocabulary and increasing number of data, in the next step they establishe the rule according to which the past tense can be formed by adding the suffix *-ed* to the base. This results in the incorrect form ∗*bringed*. In the third step, after having some feed-back about irregular verbs, they set up a ”minor rule” producing the past tense by an umlaut: using the analogy *sing-sang-sung* and *ring-rang-rung* (s)he will form *bring-∗brang-∗brung*. Only the last step will bring back to the correct form *bring-brought-brought*.

Although there are significant arguments against the claim that all children always prove to show a ”U-shape learning”, this maybe too simplistic model might still have some bits of truth. Furthermore it might be useful for computational applications, therefore it may be worth trying out the following algorithm:

```
Algorithm U-shaped-learning
Given:  L: set of data
U-shaped-learning(L)
L':= empty-set
G:=empty-set
```

```
Repeat
      Find ($L^*$ subset of $L \setminus L'$ and $G^*$ grammar) such that $G^*$ models $L^*$
      L':= L' union $L^*$
      G:= G union {($L^*$, $G^*$)}
Until (L'=L)
Return G={($L_1$, $G_1$), ($L_2$, $G_2$),...($L_n$, $G_n$)}
      // subsets of L, each of them described by a model
```

To sum up, I would suggest to replace the actual paradigm of seeing the grammar of a given language as a uniform, homogeneous model, by supposing that the grammar of a given language is in fact composed of a number of "co-grammars". Or of co-phonologies, as we want to stick to phonology (including morpho-phonology).

As shown in the previous chapter, I have been examining the learnability of metrical stress. Stress is a good field for such studies, since – as pointed out by Tesar & Smolensky [2000:53] – a lot is known about it and can be treated somewhat in isolation from other aspects of phonology. Dutch metrical stress is a very complex issue (Gilbers & Jansen [1996], Joanisse & Curtin), understanding this will be one of the ultimate goals of this study. A quick look to our data proves that no single and simple grammar can account for all of them. Just consider words of three light syllables, that can have a stress on the first (e.g. *Pánama*), on the second (e.g. *pijáma*) or on the third (e.g. *Tahití*) syllable. An even better example for showing that Dutch stress is not predictable is *de régeling* ('rule') as opposed to *de regéring* ('government'). Supposing that the only difference in their underlying form is the [l] *vs.* [r] opposition, it is highly improbable that stress assignment would make reference to some features that differentiate between these two very similar phonemes. A further complicating factor is the recognition of compound words. (And even if one could automatically recognize the most common morphemes, some cases are not predictable: the prefix *onder* is for instance stressed in *ondergaan* and unstressed in*onderstrepen*.) [1]

I believe that the fact that the group represented by *pijáma* is told to be the regular one as opposed to *Pánama* or to *Tahití* is irrelevant from a learnability approach, since the learner is simultaniously exposed to all data (not speaking of the noice present), and the property of "being regular", *i.e.* being the biggest group is only an *a posteriori* observation. Furthermore, as seen in the above example of "U-shape learning with minor rules", such as in the following case of segolate nouns in Modern Hebrew, subgroups of the irregular words may also show regularities, sometimes even being productive, therefore the "general" or "major" rule is seen as such *only* because of the size of the set of covered words.

One would argue that productivity is the main characteristics of the "major rule" that makes it special, as opposed to the "minor rules". But productivity

[1]These facts suggest that the idea of improving the finite state grapheme-to-phoneme conversion in Bouma [2000] by using information on predicted stress would not always work. Although the idea might be tried out, and one could construct an FST that works for lexemes belonging to the widest class.

is probably irrelevant for learning, and I believe that it is again an *a posteriori* property: the very fact that this group is the biggest may cause the new words to join this group. A fact that results in the biggest group become “far the biggest”. Furthermore, some “minor rules”, such as the segolate-stress in Hebrew, can also be productive, if the newly introduced word shares the most characteristic properties of the group (*e.g.* the Modern Hebrew word *seret*, 'movie', that has two [e] sounds, too, became a segolate).

As the first example I will use Hungarian data, where the stress is always on the first syllable of words (bearing stress). Modern Hebrew stress will serve as the second set of data, where the situation is just slightly more complicated: the major rule puts the stress on the final syllable, whereas a minor rule puts it on the penultimate. The exact conditions for using this or that rule will not interest us. Then we go on to the very complex Dutch data.

The learning algorithm I have used is the following:

```
Algorithm Learning Co-phonologies
Given:  L: set of data
Learning_co-phonologies(L)
L':= empty-set
G:=empty-set
Repeat
   $T$:= random subset of (L\L') // in each step it gets smaller and smaller
   Find (a grammar $G^*$) such that $G^*$ models $T$
   If found then
      $L^*$ := the subset of L that can be modelled by $G^*$
      L':= L' union $L^*$
      G:= G union $\{(L^*, G^*)\}$
   end-if
Until (L'=L)
Return G=$\{(L_1, G_1), (L_2, G_2), \ldots (L_n, G_n)\}$
      // subsets of L, each of them modelled by a grammar
```

Remark that each grammar G^* found is associated with the biggest subset L^* of L that can be modelled by G^*. Therefore these subsets of L can overlap.

Another direction would be to find a model independently for each word, instead of trying to look for random subsets of the data set. In the case of noisy data, or in the case of a number of “minor rules”, each of them applicable to a significant part of the lexicon, this approach could be even more successful.

3.1 The “program package”

The “software package” I have been developing so far can be downloaded from `http://odur.let.rug.nl/~birot/stress-group`. It contains the following files:

1. Program files written in C

- `cd.c`: This is the implementation of T&S's RIP/CD algorithm. It makes use of the the OT model applied to Finite State Transducers, implemented on FSA6. In the following I shall explain more details about it.
- `par.c`: Distributes the data to parallel child processes, each of them running `cd.c`, and then checking if a hierarchy has been learned.
- `check.c`: Evaluates all the data with respect to a given hierarchy, whether they can be modelled by the given hierarchy. Each data is given a value: 0 stands for the overt form of the learning data not occuring among the outputs. 1 stands for the overt form occuring among the outputs, but there are several outputs with different violation marks (the case typical to stratified hierarchies). 2 stands for the case when the learning data occurs in the output, together with other candidates, but all of them are assigned exactly the same violation marks. Whereas 3 is assigned when the learning data is the only output of the hierarchy if inputting the corresponding underlying form.

2. cd.c uses the following Prolog files describing FSA's:

   - `stress-prod.pl`: The main file for production directed parsing.
   - `stress-ip.pl`: The main file for the RIP algorithm. Both are auxiliary files for FSA6, loading some of the following files.
   - `con.pl`: The FSTs assigning the violation marks of the constraints.
   - `gen.pl`: contains the Gen-modul
   - `ip.pl`: contains the interpretive parsing modul
   - `def.pl`: basic definitions of FSA
   - `matching.pl`: matching algorithm according to Gerdemann & van Noord [2000].
   - `replace.pl`: Context-sensitive replacing operator, adapting the solution of Gerdemann & van Noord [1999]. Both the two last files are included among the examples of FSA Utilities.

3. Inputs and outputs of the algorithm

   - `data`: The list of input words, given in a form like `ta1L.masH`. Syllable borders should be marked with a period (only within the word, and all syllables should end with the specification of the syllable type: `L` for light, `H` for heavy and `S` for super heavy. Stress is marked with numbers before syllable type specification: `1` stands for primary stress and `2` stands for secondary stress. A word must contain exactly one primary stress. I recommend using lower case letters for giving the word, but in fact they do not play any role in the algorithm (as long as they do not coincide with some character used as a special symbol, like the ones mentioned so far, or `@` used for constraint violation mark, `*` refering to the beginning of the word, `#` to the end of the word, etc.).

- `hinit`: The initial hierarchy is given in this file. The syntax of this file is the following: each stratum is introduced by a line beginning and ending with the `#` symbol. I recommend writting the number of the stratum here, but this is not necessary, since the program will automatically fill in the first strata. Then all constraints are given in a separate line. Remarks can be put before the first line containing two `#` symbols. The file given in the stress-group package is an example file, that can be rewritten.
- `hout`: The output hierarchy, produced by the algorithm. It is given in the same format, therefore it can be reentered to the algorithm after simply renaming it.

4. Documentation:
   - `read.me`: a documentation file.
5. The following files are created by running cd.c, but are not needed beforehand:
   - `0.constdem.tmp`: The results of running the FSA's are written here.
   - `0.constdem.tmp.c`: It contains the c program generated by FSA Utilities.
   - `0.mark.xxx.fsa` where 'xxx' are the names of the used constraints.
   - `0.stress.fsa` : production and RIP.
   - `0.constdem1.tmp`: this is an executable file.

Remark: the temporary file names all start with the "0." string, and can be deleted by "`rm 0.*`".

A few more words about the `cd.c` program: it realizes the RIP/CD learning algorithm described in T&S. It calls FSA Utilities to build certain transducers: the ones assigning the violation marks, the one doing the production directed parsing, and the one realizing the Robust Interpretive Parsing. Since the ones assigning the violation marks are deterministic, they are stored and run as compiled C-programs for the sake of saving time. The FSTs realizing the two kinds of parsing are rebuilt and saved each time the hierachy changes. In fact, in order to save time, the production directed FST representing the hierarchy of the previous learning data is saved, so that if learning turns to be resultless after five learning steps, the going back to the previous hierarchy should not cause wasting time (*c.f.* T&S, p. 69).

I have gone through the all data-set five times, before concluding that the dataset is not learnable. That is I repeated the iterative RIP + CD algorithm five times. Each time I first parsed all the data using RIP, with respect to the actual hierarchy, and then I used these data for CD learning. This latter consisted of going through each data, and executing maximum five learning steps for each data (*c.f.* T&S, p. 69). If the hierarchy after the fifth step did not account for the learning data, then I went back to the hierarchy before taking that data, and went forth to another data.

In each learning step I produced the harmonic candidate with respect to the actual hierarchy, and tested it. If the harmonic candidate coincided with the target parsed data (the winner), or alternatively, if one of the harmonic candidates coinceded with it, and all the harmonic candidates were assigned exactly the same violation marks, then the actual hierarchy was said to account for the actual learning data. Otherwise a random element of the harmonic set was taken as the loser. If the winner was also in this set, but not all the harmonic candidates were assigned the same violation marks (as a result of the hierarchy being stratified), an element not having the same violation marks as the winner was taken as the loser.

In reality I redefined a little bit the CD algorithm defined by T&S (p. 95). My algorithm is equivalent, but has a different form ($C(a)$ refers to the number of violation marks assigned by constraint C to form a):

```
$i_0$ := the highest stratum in which there is a constraint $C^*$
      such that $C$(winner) $<$ $C$(loser).

for each constraint C (being in a stratum not lower then $i_0$):
   if ($C$(winner) $>$ $C$(loser))
      then {demote C to stratum $i_0 + 1$}.
```

(Remark: the strata are numbered from 0 onwards, stratum 0 being the highest ranked one.)

3.2 The FSTs used

What does the files mentioned in the previous section contain? The file `def.pl` give the general definitions used in the remaining files:

```
   :- multifile macro/2.
   % You should have the input in the form:  [[phoneme*, syllable-type, '.']*,[phoneme*,
syllable-type]]
   % In interpretative parsing:  [BOW,[phoneme*, stress-type,[], syllable-type, '.']*,EOW]
   % phonemes:  a..z
   % syllable types:  L=3, H=4,S=5
   % end of syllable:  .
   % beginning of word:  *; end of wor(l)d:  #
   % Inserted by Gen:
   % foot brackets:  left:  [ ; right,];
   %stress:  primary stress:  1, secondary stress:  2
   macro(phoneme,a,b,c,d,e,f,g,h,i,j,k,l,m,n,'o',p,q,r,s,t,u,v,w,'x',y,z).
   % @ is the mark for constraint violation, always put after '.', among the phonemes
   % Other symbol used:  T is used in con.pl (temporary violation), C, D.
```

```
macro(ls,'L'). % light syllable
macro(hs,'H'). % heavy syllable
macro(shs,'S'). % super heavy syllabe
macro(syllable-type, ls,hs,shs).
macro(eos,'.').
macro(i-eos,[]:'.').
macro(bow,'*').
macro(eow,'#').
macro(i-bow,[]:'*').
macro(i-eow,[]:'#').
macro(i-sfl,[]:'[').  % secondary feet (not the main foot)
macro(i-sfr,[]:']').
macro(sfl,'[').
macro(sfr,']').
macro(i-mfl,[]:'').  % main foot
macro(i-mfr,[]:'').
macro(mfl,'').
macro(mfr,'').
macro(fl, '' , '[' ).
macro(fr, '' , ']' ).
% eos = end of syllable; bow = beginnign of word; eow = end of word
% i-fl = insert foot left, i-fr = insert foot right
% stresses:
macro(i-ps,[]:'1').
macro(i-ss,[]:'2').
macro(ps,'1').
macro(ss,'2').
macro(stress,ps,ss).
macro(underlying-form, ?  - stress, stress:[]*).
macro(make_output, [bow:[],?  -fl, fr, fl, fr:[]*, eos:[], eow:[]]).
macro(output-production-directed-parsing,fl,fr,stress).
macro(output-interpretive-parsing,fl,fr).
```

The constraints are formulated in `con.pl`:

```
% structure of the word:  % [bow, [fl^, phoneme*, stress^, syll-type, fr^, eos,
@*]+, eow]
:-ensure_loaded(def).
:- ensure_loaded(replace).
:- multifile macro/2.
macro(tmpviol, {'T'}).  % temporary violation
% Constraints mentioned in Tesar & Smolensky, 2000, p.  54:
% Foot Binarity:
% "Each foot must be either bimoraic or bisyllabic"
% (A foot should contain more than one mora, i.e.  a heavy syllable or two syllables.)
macro(mark_ot_constraint(footbin,M), replace([]:M, [fl, phoneme*, stress, ls, fr,
eos],[])).
% Weight to Stress Principle (WSP):
% "Each heavy (and super heavy) syllable must be stressed."
macro(mark_ot_constraint(wsp,M),replace([]:M,[(?  -stress),hs,shs,fr;eos],[])).
% Parse syllable:
% "Each syllable must be footed" (remark:  each foot contains max.  2 syllables,
i.e.  each
% footed syllable is either on the left or on the right border of a foot).
macro(mark_ot_constraint(parse,M), replace([]:M,[{bow,eos}, {?  -{fl, fr, eos}}*,eos],
[])).
```

```
  % Main-Right:
  % "Align the head-foot with the word, right edge."
  macro(mark_ot_constraint(main-right,M), replace([]:M, [mfr, eos, ?  *, eos], [])
).
  % Main-Left:
  % "Align the head-foot with the word, left edge."
  macro(mark_ot_constraint(main-left,M), replace(eos:[eos, tmpviol]) o replace(tmpviol:[],
[mfl, ?  *],[]) o replace(tmpviol:M)).
  % remark:  that is the only place where I couldn't avoid using 'mfl' in order to
have a t-determinizable FST.
  % Word-foot-right:
  % "Align the word with some foot, right edge"
  macro(mark_ot_constraint(wfr,M), replace([]:M, [(?  -fr), eos], [M*, eow])).
  % It is crucial that all mark_ot_constraint's assign the same violation symbol,
  % and that nothing else come after the last eos, before the eow symbol.
  % Word-foot-left:
  % "Align the word with some foot, left edge"
  macro(mark_ot_constraint(wfl,M), replace([]:M, [bow, (?  -{fl, eos})*, eos], [])).
  % Iambic:
  % "Align each foot with its head syllable, right edge."
  macro(mark_ot_constraint(iambic, M), replace([]:M, [?  -stress,syllable-type,fr,eos],
[])).
  % Foot Nonfinal:
  % "Each head syllable must not be final in its foot"
  macro(mark_ot_constraint(footnonfinal, M), replace([]:M, [stress,syllable-type,fr,eos],
[])).
  % remark:  in this formalism, Iambic and FootNonfinal are really each other's opposite,
in one sence.
  % Nonfinal:
  % "Do not foot the final syllable of the word"
  macro(mark_ot_constraint(nonfinal, M), replace([]:  M, [fr,eos], [M*,eow])).
  % It is crucial that all mark_ot_constraint's assign the same violation symbol,
  % and that nothing else come after the last eos, before the eow symbol.
  % Remarks:  originally:  macro(mark_ot_constraint(nonfinal, M), replace([]:M, [fr,eos],
[(?  -eos)*, eow])).
  % But this cannot be determinized
  % Constraints in Dicky Gilbers & Wouter Jansen:  Klemtoon en Ritme in Optimality
Theory
  % Primary-stress-to-superheavy
  % Superheavy syllable must bear primary stress (it is Peak-prominence for SH syllables,
p.  68-69.)
  macro(mark_ot_constraint(pssh,M),replace([]:M,[(?  -ps),{shs},fr,eos],[])).
```

The file `replace.pl` is the adaptation of the algorithm described in Gerdemann & van Noord [1999] to implement the left most match replacement.

The `matching.pl` defines two operators. The `oo` optimality operator has been adapted from Gerdemann & van Noord [2000], and its main idea has been explained in section 2.2. I have introduced another operator, in order to be able to work with stratified hierarchies. Based on T&S's proposal to simply summarize the violation marks within one stratum, the `xx` operator refers to the recursive composition of the FSTs assigning the violation marks of the given constraints:

```
macro(mark_ot_constraint(Con1 xx Con2, M),
  mark_ot_constraint(Con1, M) o mark_ot_constraint(Con2, M)).
```

This results in a "complex constraint" that can enter Gerdemann & van Noord's optimality operator.

Here is the way to formulize the Gen modul in `gen.pl`:

```
   :-ensure_loaded(def).  %% Needed:  def.pl :- multifile macro/2.
   macro(data_preparation,[i-bow,{?}*,i-eos,i-eow]).
   macro(non-footed-syllable, [phoneme*, syllable-type, eos]).
   macro(non-head-foot, {[i-sfl,phoneme*, i-ss, syllable-type, i-sfr, eos],
     [i-sfl,phoneme*, i-ss, syllable-type, eos, phoneme*, syllable-type, i-sfr, eos],
     [i-sfl,phoneme*, syllable-type, eos, phoneme*, i-ss, syllable-type, i-sfr, eos]
}).
   macro(head-foot, {[i-mfl,phoneme*, i-ps, syllable-type, i-mfr, eos],
     [i-mfl,phoneme*, i-ps, syllable-type, eos, phoneme*, syllable-type, i-mfr, eos],
     [i-mfl,phoneme*, syllable-type, eos, phoneme*, i-ps, syllable-type, i-mfr, eos]
}).
   macro(gen, data_preparation o [bow,{non-footed-syllable, non-head-foot}*, head-foot,
{non-footed-syllable, non-head-foot}*, eow]).
   macro(output,output-production-directed-parsing).
```

The first step of IP (Interpretive Parsing) is very simillar to it. The only difference is that the generated set of candidates must represent the same overt form, *i.e.* they should match the input with respect to the stress location, instead of insterting stress:

```
   :-ensure_loaded(def).  %% Needed:  def.pl :- multifile macro/2.
   macro(data_preparation,[i-bow,{?}*,i-eos,i-eow]).
   macro(ip-non-footed-syllable, [phoneme*, syllable-type, eos]).
   macro(ip-non-head-foot, {[i-sfl,phoneme*, ss, syllable-type, i-sfr, eos],
     [i-sfl,phoneme*, ss, syllable-type, eos, phoneme*, syllable-type, i-sfr, eos],
     [i-sfl,phoneme*, syllable-type, eos, phoneme*, ss, syllable-type, i-sfr, eos]
}).
   macro(ip-head-foot, {[i-mfl,phoneme*, ps, syllable-type, i-mfr, eos],
     [i-mfl,phoneme*, ps, syllable-type, eos, phoneme*, syllable-type, i-mfr, eos],
     [i-mfl,phoneme*, syllable-type, eos, phoneme*, ps, syllable-type, i-mfr, eos]
}).
   macro(ip, data_preparation o [bow,{ip-non-footed-syllable, ip-non-head-foot}*,
ip-head-foot, {ip-non-footed-syllable, ip-non-head-foot}*, eow]).
   macro(output,output-interpretive-parsing).
```

This is found in `ip.pl`. The files `stress-prod.pl` and `stress-ip.pl` loads all necessary further files, as well as add some information, so that the first one can be used as the auxiliary file for production directed parsing, while the second one for Robust Interpretive Parsing.

Chapter 4

Results

4.1 Preliminary results on stress

For the first test I used Hungarian data: all of them had a primary stress on the first syllable, and no secondary stress. (The four words are in the file `data-hun`).

The resulting FSA was:

```
gen oo (footbin xx footnonfinal xx main-left xx wfl xx nonfinal) oo (parse
xx iambic xx main-right xx wfr xx wsp)
```

As explained earlier, this is to be read in the following way: we use the `oo` optimality operator to create the composition of the output of Gen with two "complex constraints". The first one is the summarizing of the violation marks assigned by the constraints in the first stratum, while the second one is defined as assigning as many violation marks as the sum of the violation marks assigned by the constraints in the second stratum. This combination the constraints within one stratum is realized by the `xx` operator.

In my second test I used nine Modern Hebrew data (`data-hb2`): seven of them having the (primary) stress on the ultimate syllable, while two of them had a penultimate stress (belonging to the "segolate" pattern). Here they are:

aL.mar1H
oL.mer1H
aL.ni1L
eL.daL.ber1H
telH.aL.viv1H
peL.taxH.tikH.va1L
sanH.hedH.rin1H
meL.daL.be1L.retH
se1L.ferH

The `par.c` algorithm succeded in finding two rankings. The one corresponding for the regular pattern is:

```
gen oo (footbin xx iambic xx main-right xx wfr xx nonfinal) oo (footnonfinal
xx main-left) oo (parse xx wfl xx wsp)
```

While for the irregular pattern (the so-called geminates):

`gen oo (footbin xx footnonfinal xx main-right xx wfr xx pssh) oo (wsp xx iambic xx main-left xx nonfinal) oo (parse xx wfl)`

Then I used 24 data taken from D. Gilbers & W. Jansen's article (p. 73), with Dicky Gilbers's evaluation about the syllable-structure (personal communication):

ma2L.caL.ro1L.nieL	gor2H.gonH.zo1L.laL
a2L.necH.do1L.teL	Mul2H.taL.tu1L.liL
fo2L.noL.loL.gie1L	co2L.rresH.ponH.dent1S
in2H.diL.viL.du1L	a2L.lekH.sanH.drijn1S
ho2L.riL.zonH.taal1S	Con2H.stanH.tiL.no1L.pelH
me2L.lanH.choL.liek1S	ar2H.chiL.tecH.tuur1S
caL.deau1L	taL.bak1H
dicH.tee1L	serH.vies1S
iL.dee1L	heL.laas1S
aL.buis1S	haL.bijt1S
o2L.noL.maL.toL.pee1L	e2L.tyL.moL.loL.gie1L
en2H.cyL.cloL.peL.die1L	di2L.aL.lecH.toL.loog1S

I had to add one more constraint (PPSS) to the ones used by T&S (p. 54) that accounts for the behaviour of super-heavy syllables. In fact neither the Hungarian nor the Hebrew data needed the supposition of super-heavy syllables, so this constraint would stay unaffected. The PPSS constraint assigns one violation mark to each super heavy syllable that does not bear a primary stress.

In the first step the `par.c` algorythm could successfully find one ranking that matches 15 data out of the 24. (This hierarchy was found when already only six data was taken into account, but then it turned out to match nine further data.)

This hierarchy is:

H_A: `gen oo (footbin xx wfl xx pssh) oo (main-right xx wfr) oo (nonfinal xx main-left) oo (footnonfinal) oo (iambic xx wsp xx parse)`

After having removed these fifteen data, a next FSA was found:

H_B: `gen oo (footbin xx parse xx wsp xx main-right xx wfr xx wfl xx pssh) oo (iambic xx main-left xx nonfinal) oo (footnonfinal)`

This can explain four out of the still unexplained data, and three of the previously already explained ones.

Remark that in fact the following hierarchy

H_C: `gen oo (footbin xx footnonfinal xx parse xx wsp xx iambic xx main-left xx main-right xx wfr xx wfl xx nonfinal xx pssh)` can also account for four data, but the H_A is a refinement of H_C, and explains a larger set of data.

There are still five data that cannot be explained. After several trials, it turned out that no hierarchy can be found for them even individually.

Here are the results, as evaluated with the `check.c` program:

data	H_A	H_B	H_C	Explained by
ma2L.caL.ro1L.nieL	3	0	1	H_A
gor2H.gonH.zo1L.laL	3	0	1	H_A
a2L.necH.do1L.teL	3	0	0	H_A
mul2H.taL.tu1L.liL	3	0	1	H_A
fo2L.noL.loL.gie1L	0	0	1	not explainable
co2L.rresH.ponH.dent1S	3	0	0	H_A
in2H.diL.viL.du1L	0	3	1	H_B
a2L.lekH.sanH.drijn1S	3	0	0	H_A
ho2L.riL.zonH.taal1S	3	0	1	H_A
con2H.stanH.tiL.no1L.pelH	3	0	0	H_A
me2L.lanH.choL.liek1S	3	0	0	H_A
ar2H.chiL.tecH.tuur1S	3	0	1	H_A
caL.deau1L	0	3	1	H_B
taL.bak1H	0	3	3	H_B
dicH.tee1L	0	0	0	not explainable
serH.vies1S	3	0	1	H_A
iL.dee1L	0	3	1	H_B
heL.laas1S	3	3	3	all
aL.buis1S	3	3	3	all
haL.bijt1S	3	3	3	all
o2L.noL.maL.toL.pee1L	0	0	1	not explainable
e2L.tyL.moL.loL.gie1L	0	0	1	not explainable
en2H.cyL.cloL.peL.die1L	0	0	1	not explainable
di2L.aL.lecH.toL.loog1S	3	0	0	H_A

0 - Not among outputs of production.

1 - Among outputs of production, but only because of the stratified nature of the hierarchy (more outputs with different constraints assigning violation marks).

2 - More outputs, but all outputs are assigned the same constraint violations.

3 - Only output of production.

After having changed the weight of the last syllable in the five non-explainable data from light to heavy, resulting in:

fo2L.noL.loL.gie1H
dicH.tee1H
o2L.noL.maL.toL.pee1H
e2L.tyL.moL.loL.gie1H
en2H.cyL.cloL.peL.die1H

it was easy to find a hierarchy for them:

H_D: `gen oo (parse xx wsp xx footbin xx pssh xx main-right xx wfr xx footnonfinal xx wfl xx nonfinal) oo (main-left xx iambic)`

H_E: `gen oo (nonfinal xx wsp xx iambic xx main-left xx footbin xx main-right xx wfr xx wfl xx pssh xx parse) oo (footnonfinal)`

H_F: `gen oo (footbin xx footnonfinal xx wsp xx main-right xx wfr xx wfl xx nonfinal xx pssh) oo (iambic xx main-left) oo (parse)`

H_G: `gen oo (footbin xx footnonfinal xx wsp xx iambic xx main-right xx wfr xx wfl xx nonfinal xx pssh) oo (parse xx main-left)`

H_D was found for fo2L.noL.loL.gie1H, H_E was found for dicH.tee1H, H_F was found for o2L.noL.maL.toL.pee1H and
for e2L.tyL.moL.loL.gie1H, while H_G for en2H.cyL.cloL.peL.die1H.

Reevaluating all the data with these new hierarchies, and the new data with the old ones:

data	H_A	H_B	H_C	H_D	H_E	H_F	H_G	Expl'd by
ma2L.caL.ro1L.nieL	3	0	1	3	1	3	1	3
gor2H.gonH.zo1L.laL	3	0	1	1	0	3	1	2
a2L.necH.do1L.teL	3	0	0	0	0	0	0	1
mul2H.taL.tu1L.liL	3	0	1	3	1	3	1	3
fo2L.noL.loL.gie1H	0	0	1	3	0	3	1	2
co2L.rresH.ponH.dent1S	3	0	0	0	0	0	0	1
in2H.diL.viL.du1L	0	3	1	0	1	0	1	1
a2L.lekH.sanH.drijn1S	3	0	0	0	0	0	0	1
ho2L.riL.zonH.taal1S	3	0	1	3	0	3	1	3
con2H.stanH.tiL.no1L.pelH	3	0	0	0	0	0	0	1
me2L.lanH.choL.liek1S	3	0	0	0	0	0	0	1
ar2H.chiL.tecH.tuur1S	3	0	1	3	0	3	1	3
caL.deau1L	0	3	1	0	3	0	1	2
taL.bak1H	0	3	3	3	3	3	3	6
dicH.tee1H	0	0	1	0	3	0	1	1
serH.vies1S	3	0	1	3	3	3	1	4
iL.dee1L	0	3	1	0	3	0	1	2
heL.laas1S	3	3	3	3	3	3	3	7
aL.buis1S	3	3	3	3	3	3	3	7
haL.bijt1S	3	3	3	3	3	3	3	7
o2L.noL.maL.toL.pee1H	0	0	1	0	0	3	1	1
e2L.tyL.moL.loL.gie1H	0	0	1	0	0	3	1	1
en2H.cyL.cloL.peL.die1H	0	0	1	0	0	3	3	2
di2L.aL.lecH.toL.loog1S	3	0	0	0	0	0	0	1
Number of words in total:	15	7	4	10	8	14	5	

This way, we have found a hierarchy for all data. Remark, that H_A and H_B still are not able to explain the five data (I mean, after having changed the syllable-type of their last syllable.) In fact H_G turns out to be superflous, because the data that has produced it (en2H.cyL.cloL.peL.die1H) can also be explained by H_F. There is no data that could be produced only by H_G.

The hierarchy explaining the biggest number of the data is H_A: it models 15 out of 24, that is it explains more than half of it. As we shall prove it in the next chapter, the best distribution is classing all of these data into the class of H_A. There are 9 data left. H_F can account for 5 out of these 9, while all the other hierarchies only for a less number of them: H_B and H_E for 4, H_D and H_G for 2, while H_C only for 1. That is, the next class shall contain the 5 data accounted for by H_F. Then we have two equal possibilities for the remaining 4 data: either H_B accounts for 3 and H_E for 1, or just the opposite, H_E for 3 and H_B for 1.

4.2 Constraints that cannot be formulated with FSTs

We have mentioned that two out of the constraints used by T&S cannot be formulated using Finite State Transducers. These are All-Feet-Left and All-Feet-Right. The reason they cannot be formulated within Finite State technology is that they may assign a number of violation marks that is quadratic in the length of the word.

This is related to, but not a necessary consequence of their being *gradient constraints*. Other constraints, such as Main-Left and Main-Right, that require counting can be implemented on FSTs. In fact counting by Main-Left and Main-Right could be circumvented by a single pass through the word: instead of assigning the word or the head syllable as many violation marks as the number of syllables intervening between the relevant foot edge and the relevant word edge, we could assign one violation marks for each foot intervening between the relevant foot edge and the relevand word edge. But in the case of All-Feet-Left and All-Feet-Right one would need a number of pass through the word, first marking each foot one-by-one, and then assigning a violation mark for each foot intervening between the relevant edge and the marked foot. (This can be approximated, as shown later.)

In order to prove in a mathematical way that these two constraints cannot be formulated by FSTs, first we shall present a lemma, that is in fact a simple consequence of the so-called *pumping lemma*.

Lemma: Let **T** be a functional Finite State Transducer, that is for any input string σ it produces at most one output $T(\sigma)$. Then there exists a linear upper bound on the length of the output, *i.e.* there exists a positive integer k such that for any input string σ for which there exists an output $T(\sigma)$ the following inequality holds:

$$\mid T(\sigma) \mid \leq k \mid \sigma \mid$$

where $\mid \alpha \mid$ denotes the length of the string α.

Proof:

A Finite State Transducer with an input alphabet A and an output alphabet B can be seen as a Finite State Automaton over the alphabet $X = (A \cup \{\epsilon\}) \times (B \cup \{\epsilon\}) \setminus \{\epsilon, \epsilon\}$. A string $(a_1, b_1)(a_2, b_2)...(a_n, b_n)$ accepted by the automaton corresponds the the input-output pair $(a_1a_2...a_n, b_1b_2...b_n)$ of the transducer, with the ϵ-s being simply deleted.

For a string $f = (a_1, b_1)(a_2, b_2)...(a_n, b_n)$ accepted by the automaton let us call the first projection $f_i = a_1a_2...a_n$ the *left-hand* or *input string* of f, and let the second projection $f_o = b_1b_2...b_n$ be the *right-hand* or *output string* of f.

Now we will make use of a corollary of *Ogden's Iteration Lemma for Regular Languages*, a variation of the *Pumping Lemma* (Corollary 4.7 in Berstel [1979],

p. 21). This claims that if $L \subset X^*$ is a regular language, and $Y \subset X$, then there is an integer $N \geq 1$ such that for any $f \in L$ and for any factorization $f = hgh'$ with $\mid g \mid_Y \geq N$, [1] g admits a factorization $g = aub$ such that (i) $0 < \mid u \mid_Y \leq N$, and (ii) $hau^*bh' \subset L$. [2]

Let L be the language accepted by the FSA corresponding to the Finite State Transducer **T**, as explained above. And let be $Y = \{\epsilon\} \times B \subset X$. This means that there exists a positive integer N, such that for any $f \in L$ and for any factorization $f = hgh'$: if $\mid g \mid_Y \geq N$, then g can be factorized such as $g = aub$, $\mid u \mid > 0$ and $hau^*bh' \subset L$.

The case $u \in Y^*$ would mean that some inputs of the transducer can generate several (an infinite number of) outputs, since the elements of hau^*bh' would correspond to the same input and to different outputs. Therefore the fact that **T** is functional means that for any $f \in L$ and any factorization $f = hgh'$, if $\mid g \mid \geq \mid g \mid_Y \geq N$ then g contains at least one character from $X \setminus Y$ (since its substring u should contain one).

In other words: it is not possible to find a substring of more then N elements of Y within any $f \in L$. This means that the input (left-hand) string f_i of f does not contain a series of more than N consequtive ϵ-s.

Remember the way we constructed our automaton from the transducer **T**: the input string σ of the transducer is the non-ϵ substring of f_i. We can thus infer that

$$\mid f \mid = \mid f_i \mid \leq (N+1) \mid \sigma \mid$$

Making use of the fact that the output $T(\sigma)$ of the transducer is the non-ϵ substring of the right-hand string f_o, we can conclude:

$$\mid T(\sigma) \mid \leq \mid f_o \mid = \mid f \mid \leq (N+1) \mid \sigma \mid$$

Thus we have proven our lemma.$\diamondsuit$

The next step is to realize that All-Feet-Left and All-Feet-Right can assign a number of violation marks that is quadratic in the length of the input. In fact if the input consists for instance of n syllables, each of them parsed into a separate foot, then the number of violation marks to be assigned to the word is $n(n-1)/2$. Therefore no linear bound can be given (in function of the input's length) to the length of the ouput of the process assigning violation marks. But this process should be functional, since assigning violation marks is a function. Supposing we had a functional transducer realizing the All-Feet-Left or the All-Feet-Right constraint, there would be an integer N such that the maximum number of violation marks assigned would be $N-1$ times the length of the input (no deletion takes place in violation mark assignment). If we suppose that ξ is the maximum length of a syllable [3] we get the inequality:

[1] $\mid u \mid_Y$ refers to the number of occurences of elements of Y in the string u.

[2] L' appears in Berstel [1979] at this last point, but this should be a mistake.

[3] Although such a supposition cannot be made in general, since stress assignment is independent from the phonemes in the word, one could just delete the phonemic content of the input, without altering the process. In such a case an upper limit can already be given.

$$\frac{n(n-1)}{2} \leq (N-1)n\xi$$

It is possible to choose n great enough that this would not be true. As there is no theoretical limit on the number of syllables in one word, we have proven that no Finite State Transducers exist realizing All-Feet-Left and All-Feet-Right in an ezact way.

But this does not mean that no approximation can be given. One can suppose in real life languages, that the number of syllables (or even more the number of feet) in one word is indeed bound.

Here I am giving an approximation for All-Feet-Right. The FST called `one_feet_right` assignes a violation mark (`M`) to all syllables right to the foot just being checked (marked by a `C` character). A step consists of marking the first unchecked foot from the left by this `C` symbol, then running `one_feet_right` and finally marking that foot as have been already ckecked (`D`). If we have a bound n on the number of feet in a word, repeating this process n times would practically result in a realization of All-Feet-Right.

```
macro(checking,'C').
macro(checked,'D').
macro(one_feet_right(M),
     replace([]:M, [checking, eos, ?  *, eos], [])).
macro(one_step(M),
     replace([]:checking,[bow, (?  -checking)*, fr], checked)
     o one_feet_right(M)
     o replace(checking:checked)).
macro(mark_ot_constraint(all_feet_right,M),
     one_step(M) o one_step(M) o ...  o one_step(M)
     o replace(checked:[]) ).
```

It is noteworthy that even three steps result in an FST that has 472 states. This "explosion" in the number of the states show the inheritingly not Finite State-likeness of the problem.

A similar procedure is possible for All-Feet-Left.

Chapter 5

The structure of the lexicon

5.1 Introduction to the problem

As a result of our experiments we received a number of hierarchies H_i, each of them describing a subset L_i of the lexicon:

$$(H_1, L_1), (H_2, L_2), ..., (H_n, L_n)$$

In an optimal case all elements of the lexicon are covered by at least one subset L_i, but there will be probably overlaps. The question is how to interpret this results, what to do with them. Obviously we can eliminate a hierarchy (like H_G for our Dutch data) that account for data that are accounted for by others, too. But in a complicated situation, it is not obvious either which one to eliminate, if there are more than one like this.

Then, how to distribute the words among the subsets? One approach would be to let a word be associated to more than one hierarchies If we suppose that only one hierarchy (one transducer) is built up (or: is active) in a given point in time during production (either mentally or computationally speaking), then we need to rebuild (reactivate) a new hierarchy each time that we want to produce a word belonging to a different class from the class in which the previous word belongs to. Rebuilding (reactivating) a new FST can be very expensive (building new FSTs takes most of the CPU time of the learning algorithm). In this there is an advantage in letting a word belong to more than one class, since this would increase the probability of not having to build a new transducer for that word.

If we reject the possibility of a word belonging to more classes then we want to find a partition of the lexicon into these disjunct classes. How to do that? What are our criteria or priorities when doing this?

I can suggest three approaches. The first one follows the previous line of reasoning and tries to minimalize the probability of having to rebuilt the transducer during the production of a list of words. The first steps towards this are shown in the next section.

A second approach would minimalize the storing capacities needed. A way to do that would be to look for hierarchies that differ only minimally, so that a few binary parameters would be enough to refer to the one or to the other of them. This could lead a decision tree-like hierarchical structure of the lexicon. The second section gives some initial speculations of mine in this field.

A third approach would look at the similarities of the words to be put into one class. In the case of the Hebrew irregular pattern (the "segolates") the similarity of these words are shocking, and this is the reason why this minor rule turns out to be productive for new words sharing the same pattern. This approach would thus try to create classes that could then predict the class into which new words would belong to.

5.2 Learning a multi-phonological lexicon

There are many signs that one should stop looking for a grammar or phonology accounting for the whole of a given language. The old-new paradigm should be rather to partition the lexicon into (disjunct?) classes, each of which can be accounted for by one model. The point is to optimize this structure in some way.

Supposing that the most expensive action is building a new hierarchy (anyway, this is the case on computer simulations where building new transducers takes most of the CPU time), the parameter to be optimized is:

$$P = \frac{a_1^2 + a_2^2 + ... + a_n^2}{N^2}$$

where a_i is the number of the elements in class i out of the n classes, and $N = a_1 + a_2 + ... + a_n$ is the number of words in the lexicon. The reason for this is that P is the probability that two neighbouring elements of an uncorrelated sequence of words will belong to the same class, *i.e.* there is no need to change the hierarchy (the active FST). [1]

The way to find the partition maximizing p is by maximizing the biggest a_is (that is minimalizing the entropy). In other words one has to put all the elements of the greatest subset, into the first partition. Then one has to check which subset contains the most elements out of the remaining (non yet distributed) lexical items, and put them into the second partition. Then this step has to be repeated as long as there are undistributed elements.

Although this algorithm since to be obvious, it has to be proven mathematically. At the moment I can prove it only for the case when the biggest subset contains more then half of elements, in each step. As this is the case for our Dutch data, the use of this algorithm in that chapter seems to be correct.

[1] Supposing that the words are distributed equally in the sequence. This approach can be refined by taking into account word frequencies of the items belonging to a group. This would minimalize the probability of rebuilding the transducer in real production tasks. But in fact this is just redefining the a_i parameters as being the sum of the frequencies of the words belonging to that class, instead of being simply the number of the same words.

But what happens if the biggest class contains less than half of the data? Another question that one should raise: what happens if there are two classes with the equal number of data, as it was the case for Dutch data? Do we need to try out which one maximalizes P?

5.3 Further speculations about the structure of the lexicon

This section is the beginning of a speculation that should leave to a method for creating a decision tree for hierarchies, or some other ways for organizing the hierarchies of a given language into a homogenious structure. The outcome may be an algorithm that partition the data of the complex lexicon into a subset whose storing place is minimal. Alternatively we could gain an alternative learning algorithm, if we suppose that the hierarchies describing the different classes are not random ones, but they differe from each other in a systematic way, and therefore we could try a learning algorithm that uses this information when searching for further hierarchies.

I had the idea to use a previous paper of mine (Bíró & Hamp [2001]) on Hebrew morphology to show this idea. That paper introduced binary parameters that would determine the ranking of the constraints. Each subset of the lexicon could be associated with a different parameter setting, therefore with a different hierarchy. But due to some problems for which my solutions were not really conform to main-stream OT, I gave up trying to develop this idea into a tree-like lexicon structure.

Restrictions on hierarchies will become a key term from now on. In fact a restriction r is a statement on hierarchies, and represents the subset of hierarchies (out of the set $\mathbf{U}$ of all possible *fully ranked* hierarchies defined by the finite universal set $\mathbf{C}$ of constraints) for which the statement holds.

A *primitive restriction* is an ordered pair of constraints ($c_1 >> c_2$), representing the logical statement '$c_1 >> c_2$' on hierarchies. In other words, a primitive restriction is a set of hierarchies for which this statement holds: the set of hierarchies satisfying this statement. For each hierarchy $h \in \mathbf{U}$ it can be decided if h is an element of the set represented by a given primitive restriction (using vague notions: $h \in r$) or not.

1 is the zero-restriction, the set of hierarchies associated to it is **H**.

The *set R of restrictions*can be defined as the following:

a. $\mathbf{1} \in R$.

b. all primitive restrictions are within R (if $c_1 \in$ bf C, $c_2 \in$ bf C, and $c_1 \neq c_2$, then $(c_1 >> c_2) \in R$);

c. if r_1 and r_2 are elements of R, then $r_1 \wedge r_2 \in R$;

d. if r_1 and r_2 are elements of R, then $r_1 \vee r_2 \in R$.

The *conjunction* $r_1 \wedge r_2$ of two restrictions r_1 and r_2 is the logical AND-relation between the two statements, that is the intersection of the set represented by r_1 and of the set represented by r_2. Similarly, the *disjunction* $r_1 \vee r_2$

of two restrictions is nothing but the logical OR-relation of the two statements, *i.e.* the union of the two sets in question.

Furthermore, the negation of a primitive restriction $r = (c_1 >> c_2)$ is the primitive restriction $r = (c_2 >> c_1)$, and the corresponding set of hierarchies is the complementary set, since we speak of fully ranked hierarchies. If $r \in R$ is not a primitive restriction, then its negation can also be defined using the standard equations from Boolean algebra. In general it is obvious that restrictions form a Boolean algebra. $\mathbf{1}$ is the one-element, while $\mathbf{0} = \neg\ \mathbf{1}$ is the zero-element of it. The latter is associated with the empty set.

The idea is now the following: the lexicon should be organized in a hierarchical structure. The classes of the lexicon are to be found at the leaves of the tree, while each node represent a binary decision. This statement to be decided is either a primitive restriction of the form "$c_1 >> c_2$", or a conjunction of them. The tree being binary, one branch leaving will signify that the statement holds, while the other means that it doesn't. In other words, one part of the subset of the lexicon below the node satisfies the statement, while the other part doesn't.

To be more exact: each node of the element is associated with a so-called *inherited restriction* and also – with the exception of the final nodes – with a so-called *decision statement*. The inherited statment of each non-root node is the conjunction of the mother's inherited statement with the mother's decision statement, if the node in question is a left daughter, and with negation of the mother's decision statement, if the node in question is a right daughter. The hierarchies describing the classes of the lexicon that are bellow the given node are all members of the set represented by the inherited restriction of the given statement. Furthermore, the inhereted restriction of a final node represents a single hierarchy.

The goal woud be now the following: the structure of the lexicon should be ideally the following:

1. The biggest possible r_0 is the root node's inherited restriction, which is supposed to be the conjunction of universal restrictions (cf. universal subhierarchies in P&S chapter 9, T&S p. 47), and of language specific restrictions.

2. The root node's decision statement is for a given $c_1 \in \mathbf{C}$:

$$r_1 = \wedge_{c \in \mathbf{C}, c \neq c_1} (c_1 >> c)$$

3. For both daughters the decision statement is of a similar form, etc.

Chapter 6

PhD Project Proposal

6.1 The complexity of the problem

If our goal is to build an FSA that generates Dutch stress using an OT approach, and the solution is not trivial, we have recognize that the problem is very complex. The lack of success is not necessary due to the learning algorithm, but there are two extra factors, two undecided phonological points that are outside the field of computational linguistics:

- Although the set of constraints are thought to be universal in theory, in fact each and every phonological article proposes a different set of constraints, and there are usually even slight differences between the formulation of the same constraints. I have used the set proposed by Tesar & Smolensky, supposing that this set can be seen as a general consensus in phonology, but I still had to add a constraint about super-heavy syllables. A phonological way of going further would be to seek for a new set of constraints, or for a best formulation of these. [1]
- One should recognize that even the data are not always clear. Some out of the learning data given by Dicky Gilbers were not learnable, but changing the definition of syllable types made them learnable. Based on this fact one could argue for a new definition of light, heavy and super heavy syllables. Others would reject this approach and would propose to look for different constraints.

I think there is here a question of paradigm. Traditional (non-computational) phonology has very different criteria from computational approaches. Learnability is not a major factor for deciding either the universal set of stress constraints,

[1]Notice that adding or removing constraints can critically affect the outcome. Even if I demote a constraint to the very bottom of the hierarchy, it can filter out a candidate. Therefore adding “superfluous” constraints can produce a situation where some of the learning data become a “looser”, i.e. a candidate that will win for no ranking. (E.g. if that candidate has been previously assigned exactly the same violations as another candidate.)

or the definition of syllable types. These discussions are still in process, and a computational linguists has to wait for a consensus about them, before trying out his or her proposals for new algorithms. But because learnability issues can still influence “traditional” phonology (e.g. through language acquisition) this “22-catch” cannot be avoided.

Therefore I propose the following steps for my research.

6.2 An overview of constraints and syllable type definitions used in current phonological literature

I should go through the relevant articles in ROA, gathering different definitions of syllable types and sets of constraints, paying attention to the slight variations of the same constraints. I do not want to enter phonological discussions, my only point would be to try to set up a definition of syllable types, a definition of constraints in used, and a set of universal constraints that best approximates the ones used in current literature.

6.3 Formal properties of OT

See e.g. Samek-Lodovici [1999], and his lecture notes for ESSLLI 2002. The main question here is how to identify loser candidates, that is candidates that are optimal forms for no constraint ranking. Identifying these candidates (or better: not generating them by Gen, this is the topic of Samek-Lodovici's current research, according to him) could help us in avoiding some of the failures of the RIP/CD algorithm (*cf.* T&S p. 71). What are the implications of these investigations for us?

Further issues that are interesting topics in now-days researches in OT: output-output-correspondence (*cf. e.g.* Burzio [1999], a widely used technique, but problematic to Finite State approaches), stochastic OT (), bi-directional OT (*e.g.* Jäger [2001a, b]), constraint conjunctions (*e.g.* Moreton & Smolensky [2002]).

6.4 Further refinement of the current model

For example it should be proven that the actual constraints are exact, or the level of approximation needed is to be calculated. (Cf. Gerdemann & van Noord [2000]: a candidate with more violations than the minimal can survive sometimes, if one does not apply `permute_marker`.)

Furthermore there are a number of smaller problems to be solved. For instance, based on “calculations” on paper, the hierarchy `all` >> `WSP, FootNonFinal` should be a solution for the word *dicH.tee1L*. The reason for that is maybe the fact that some non-harmonic candidates are not filtered out at a given point, because there are no candidates having one violation mark less. Therefore the optimality operator should be slightly redefined.

6.5 Learning algorithm using fully ranked hierarchies

As explained, the partially ranked hierarchies by T&S are not always convincing. Therefore the case of using always fully ranked hierarchies during the learning algorithm should be also checked.

6.6 Applying to MPI

A possible practical point of this research would be to apply the model on parallel machines, and this would save computing time. The point that can be parallelized is when we randomly distribute the data to parallel processes that are looking for possible rankings. Actually this has been done by parallel processes on my PC. Using High Performance Computing would make possible to make experiments on a radically higher dataset.

6.7 Possible clusterings of the subsets of the lexicon

See chapter above about the possible structure of the lexicon. At this point I still have just speculations about the possible ways to organize the lexicon. But by the time I wish to tackle this problem, I hope to have gathered enough experience and literature about the topic so that I will be able to achieve real results.

6.8 Distribution of lexicon subsets in written texts

I believe that it would be very useful to have a look at the way words associated to different hierarchies are distributed in a pronounced or written text, i.e. to observe the pattern of changing constraint hierarchies. Supposing that changing a hierarchy (building a new FSA) is an extremely expensive operation (as is the case on computers), the outcome of this investigation would motivate the choice of the way we would cluster the lexicon subsets.

Chapter 7

Schedule

7.1 Achievements in the first year (October 2001-September 2002)

- Reading group in Machine Learning (presentation: December 13, 2001: Tesar and Smolensky [2000], chapters 4-6., Jan. 17, 2002: Tesar and Smolensky, [2000], chapters 7-8., Apr. 18, 2002: Support Vector Machines, chapter 4).

- Participation in the meetings of the Language Acquisition Lab (by A. van Hout).

- Dutch Courses, level 1 (February - May, 2002), level 2 (September - December, 2002).

- LOT Winter School (Leiden, January, 2002) and Summer School (Nijmegen, June, 2002). Courses by S. Tagliamonte (*Language Variation and Change: Theory, Method and Analysis*), G. Extra and D. Gorter (*Comparative Perspectives on Minority Languages in Europe*), T. Rietveld and R. van Hout (*Statistics in language research: analysis of variance*), P. Fikkert (*Acquiring phonology*), O. Crasborn (*Cross-linguisitic perspectives on sign language structure*), D. Sandra (*Psycholinguistics*).

- Arnold Meijster (RC): MPI-course (High Performance Computing) (Dec. 11-12, 2001).

- ESSLLI Summer School, Trento, Italy (August 2002). Courses by V. Samek-Lodovici (*Formal Properties of Optimality Theory*), K. van Deemter and M. Stone (*Formal Issues in Natural Language Generation*), C. Martin-Vide (*Formal Language Theory: Classical and Nonclassical Machineries*), G. Huet and C. Rétoré (*Survey of a few Fundamental Representation Structure for Computational Linguistics*),...

- Teaching the Tekstmanipulatie class (September-November, 2002).

Out of the 20 credits required by BCN, I have done: Dutch course, level 1 (3 points?), LOT, 6 courses (6 points?), ESSLLI, teaching assignment.

7.2 Plan for years 2-4

November - December, 2002

- Solving the remaining problems with FSTs.
- Investigations into the level of approximations needed.
- Collecting articles for a reading group in the formal properties of OT.
- Presenting at CLIN 2002 (November 2002): *Learning Dutch Stress in Optimality Theory using FSA Utilities.*

January - June, 2003

- Reading group in formal properties of OT.
- An overview of constraints and syllable type definitions used in current phonological literature.
- Implementing the learning algorithm with fully ranked hierarchies.
- Presenting EACL 2003 (April 2003): *Some Preliminary Remarks on Violation Assignment and Further Issues in Finite State Optimality Theory: The case of stress assignment.*
- LOT Winter and Summer schools (January and June, 2003).
- BCN Poster Session (February 2003), BCN Orientation Course (March 2003), BCN Retreat (April 2003).
- Dutch course, level 3 (February-May 2003).

July - December, 2003

- Reviewing the learning algorithms in the light of theoretical issues.
- Applying the learning algorithm to MPI.
- Collecting literature on how does OT usually deal with exceptions.
- ESSLLI 2003 (August 2003).
- BCN Philosophy of Science and Mind course (November 2003).
- Dutch course level 4 (September-December 2003).

- Course in “Presenting in English”.
- Participation at CLIN 2003.

January - June, 2004

- Working out the lexical model.
- LOT, Conference participation.
- Course in “Publishing in English”.
- Writing a first version of half of the dissertation.

July, 2004 - September, 2005

- Writing the thesis.
- LOT, CLIN, ESSLLI, Conference participation.
- BCN Poster Session (February 2005), BCN Retreat (April 2005).

Further interests

- Course in Phonetics.
- Course in statistical and stochastical methods.
- Course in PROLOG.
- Course in High Performance Computing (Nikolai Petkoff).

Chapter 8

References

- E.L. Antworth [1990]: PC-KIMMO: a two-level processor for morphological analysis, in: Number 16 in Occasional publications in the academic computing, Summer Institute of Linguistics, Dallas.
- Jean Berstel [1979]: Transductions and Context-Free Languages, Teubner, Stuttgart.
- Tamás Bíró & Anna Hamp [2001]: Schwa and Roots: A Non-concatenative Lexical Morpho-phonology, in: Selected Papers of Docsymp 6, the Graduate Students' Sixth Linguistics Symposium, April 28, 2001, Budapest.
- Gosse Bouma [2000]: A Finite State and Data-Oriented Method for Grapheme to Phoneme Conversion,in: Proceedings of the first conference of the North-American Chapter of the Association for Computational Linguistics, pages 303-310, Somerset, NJ, 2000. Association for Computational Linguistics.
- Luigi Burzio [1999]: Missing Players: Phonology and the Past-tense Debate, down-loadable at: `http://hebb.cog.jhu.edu/index.cfm?urlpage=menu7&secx=1`.
- Robert Frank & Giorgio Satta [1998]: Optimality theory and the computational complexity of constraint violability, Computational Linguistics, 24:307-315
- Dale Gerdemann, Gertjan van Noord [1999]: Transducers from Rewrite Rules with Backreferences. EACL 99, Bergen Norway.
- Dale Gerdemann, Gertjan van Noord [2000]: Approximation and Exactness in Finite State Optimality Theory, in: Jason Eisner, Lauri Karttunen, Alain Thriault (editors), SIGPHON 2000, Finite State Phonology.
- Dicky Gilbers, Wouter Jansen [1996]: Klemtoon en Ritme in Optimality Theory, in: TABU 26-2, p. 53-101.

- Trevor Harley [2001]: The Psychology of Language, from data to theory, 2nd edition, Psychology Press, USA-Canada.
- Gerhard Jäger [2001?a]: Some Notes on the Formal Properties of Bidirectional Optimality Theory, ROA 414-0900.
- Gerhard Jäger [2001?b]: Gradient constraints in finite state OT: The unidirectional and the bidirectional case, ROA 479-1101.
- Marc Joanisse & Suzanne Curtin: Dutch Stress Acquisition: OT and Connectionist Approaches, at `http://citeseer.nj.nec.com/288300.html`
- R. Kaplan and M. Kay [1994]: Regular models of phonological rule systems, Computational Linguistics, vol. 20, no. 3, pp. 331–378.
- Lauri Karttunen, Jean-Pierre Chanod, Gregory Grefenstette & Anne Schiller [1996]: Regular expressions for language engineering, Natural Language Engineering, 2 (4):305-328
- Lauri Karttunen [1998]: The proper treatment of optimality theory in computational phonology, in: Finite-state Methods in Natural Language Processing, pp. 1-12, Ankara.
- Lauri Karttune & Kenneth R. Beesley [2001]: A Short History of Two-Level Morphology, presented at ESSLLI 2001: `http://www.folli.org/` or `http://www.folli.uva.nl/CD/2001/courses/20years/twol-history.html`
- Elliott Moreton & Paul Smolensky [2002]: Typological Consequences of Local Constraint Conjunction, ROA 525-0602.
- Alan Prince & Paul Smolensky [1993]: Optimality Theory, Constraint Interaction in Generative Grammar, RuCCS-TR-2, ROA Version 8/2002.
- Gertjan van Noord & Dale Gerdemann [1999]: An extendible regular expression compiler for finite-state approaches in natural language processing, in: O. Boldt, H. Juergensen and L. Robbins (eds.): Workshop on Implementing Automata, WIA99 Pre-Proceedings, Potsdam, Germany.
- Gertjan van Noord [1997]: FSA Utilities: A toolbox to manipulate finite-state automata. In: Darrell Raymond, Derick Wood and Sheng Yu (eds.): Automata Implementation, pp. 87-108, Springer Verlag, Lecture Notes in Computer Science 1260.
- Gertjan van Noord [1999]: FSA6 reference manual, The FSA Utilities toolbox is available free of charge under Gnu General Public License at `http://www.let.rug.nl/ vannoord/Fsa/`.
- Vieri Samek-Lodovici, Alan Prince [1999]: Optima, draft 11/22/99. Distributed at ESSLLI 2002.
- Bruce Tesar, Paul Smolensky [2000]: Learnability in Optimality Theory, The MIT Press, Cambridge, MA, London, England.