University of California

Los Angeles

Evaluation, Implementation, and Extension of Primitive Optimality Theory

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Arts in Linguistics

by

Daniel Matthew Albro

1997

The thesis of Daniel Matthew Albro is approved.

Bruce P. Hayes

Donca Steriade

Edward P. Stabler, Committee Chair

University of California, Los Angeles

1997

To Seán

Table of Contents

List of Figures

List of Tables

Abstract of the Thesis

Evaluation, Implementation, and Extension of Primitive Optimality Theory

by

Daniel Matthew Albro

Master of Arts in Linguistics

University of California, Los Angeles, 1997

Professor Edward P. Stabler, Chair

Eisner's (1997a) *Primitive Optimality Theory* is a simple formal model of a subset of Optimality Theory (Prince and Smolensky 1993). The work presented here implements this model and extends it. The implementation is used to evaluate the Primitive Optimality Theory model, and is in itself a useful tool for linguistic analysis. The model is evaluated in terms of its success or failure as an attempt to formulate a cognitively plausible, computationally tractable, and mathematically formal model of the Optimality Theoretic framework of phonological theory. As part of this evaluation, a comprehensive, implemented analysis is given for the harmony and disharmony phenomena of Turkish. In addition to an evaluation of the Primitive Optimality Theory model, concrete proposals are suggested for possible extensions to the model, and for improved models that, unlike Primitive Optimality Theory, can model non-concatenative morphology, Paradigm Uniformity, and reduplication.

CHAPTER 1

Introduction

The goal of the work described here is to formulate a cognitively plausible, computationally tractable, and mathematically formal computational model of the Optimality Theoretic framework of phonological theory[1], where a computational model of a system is a rigorous encoding of that system in mathematical terms, put together in such a way as to permit reasoning about how computations may be made within the system. A further goal is to implement the model. This thesis in itself does not propose to achieve these goals in their entirety, but rather to take one or two steps toward them by evaluating and building upon an earlier attempt at such a model—Primitive Optimality Theory (Eisner 1997b)—which in turn is based on the finite state model of Optimality Theory proposed by Mark Ellison (1994). Three questions may be expected at this point. Why bother to formulate such a model, when phonologists are doing perfectly well without one, why implement the model once it exists, and why examine and modify the Primitive Optimality Theory (OTP) model instead of starting afresh.

[1] Although Optimality Theory is indeed being used for other branches of linguistics, particularly syntax, a formal model of the framework as applied to these branches might be quite different from the model used for phonology

1.1 Why Formal Models

As matters currently stand, there has been fairly little said about what Optimality Theoretic constraints may legitimately be hypothesized, that is, about meta-constraints. It is true that most subscribe to the notion that constraints should represent cross-linguistic tendencies—that is, that any group of constraints describing a particular language, when reranked, should be likely to describe another language, and this notion seems to be one valid meta-constraint. If the only goal of work in Optimality Theory is to develop a set of constraints, which, when ranked in different ways, results in the phonology of all studied languages, then this meta-constraint is sufficient.

However, if we wish to say something about human cognitive function, if we wish to describe the set of possible human languages rather than simply to summarize the set of known languages, something must be said about the formal structure and limitations of a permissible constraint. Yes, it is possible to describe the set of human languages using Optimality Theoretic constraints. After all, given that a constraint may be any function from outputs to constraint violations, we could specify constraint number one as "output satisfies the true grammar of language number one", and so forth, and then the top-ranked constraint would determine the particular language. Ranked constraints of the sort used in Optimality Theory could be used to describe any arbitrary set, not just the set of human languages. It is certainly also possible, for that matter, to describe the set of human languages using rewrite rules and principles, assuming possible rewrite rules are as unrestricted as possible constraints are today. If Optimality Theory is to say something interesting about the set of human languages, and if it is to be falsifiable, then, in addition to ensuring empirical accuracy and plausibility, we must limit as strictly as possible what sort of constraints may be

hypothesized.

The basic definition of an Optimality Theoretic constraint states that a constraint is a function from output forms to integers, where the integers represent the number of constraint violations incurred by the output forms. Among such functions, infinitely many are not computable. Infinitely many others are computationally intractable. At the very least, a description of what sorts of constraints are computable in finite time in a finite-memory machine would help to limit hypothesized constraints to those that have a chance of actually describing what the brain does.

Finally, some linguists have hypothesized that, rather than all Optimality Theoretic constraints being installed in babies before birth and language learning thus being a process of constraint ranking, children actually create some or all of the constraints in their grammars from the data with which they are faced. An example of this is Hayes' (1996) theory of Inductive Grounding, which suggests how phonetic data might be used to learn constraints. In order to learn constraints from data, there must be some limit on what sorts of constraints may be hypothesized, or the learning process will be either non-computable or highly intractable.

A formal model of Optimality Theory should thus provide an explicit definition of what sort of constraints may be hypothesized. This definition must allow for only those constraints whose violations may be efficiently computed and which may be efficiently combined together into a constraint system.

The reason for developing the formal model presented in this thesis is thus to explore whether Optimality Theory derivations are in general computable by a finite device such as the human brain, to see what subset of the proposals given within the framework of Optimality Theory might be workable from a

computability standpoint, and by seeing how the computability requirement limits the analyses that may be proposed for phonological phenomena to gain some insight into what the capabilities and limitations of the human language facility might be. The formal model to be presented here, or any attempted model of Optimality Theory, should be considered a success to the extent that all data patterns that are accounted for in Optimality Theory can be accounted for by analyses defined in the terms of the model, and also the extent that the data patterns can be computed in a finite amount of time by some implementation of the model.

1.2 Why Implement Optimality Theory

Once a valid and computable formal model of Optimality Theory has been constructed, it will already be of enormous help in understanding what sort of grammars should be possible and deciding which constraints may be considered in an analysis. An implementation of such a model, besides providing a proof by existence that it is computable[2], would make it even more useful. Implementations even of invalid models are useful, in that their existence might help to prove the models invalid and to move toward a valid one. An implementation of Optimality Theory is useful as a tool for analysis and as a step toward better formal models of the theory.

[2]This assumes that the implementation can be shown to be faithful to the model and to run to completion for all inputs.

1.2.1 Implementations as Tools for Analysis

An implementation of Optimality Theory takes some input representation of a word or phrase—perhaps an abstract underlying form, perhaps a set of numbers representing morphemes, perhaps a set of activated paradigmatic elements—and produces a corresponding output form that best satisfies a set of constraints ranked in some language-particular manner. As such, it is a perfect tool to force the analyst of a language to be explicit with his or her constraints. If the analysis is incorrect or inconsistent, the implementation will not produce the expected outputs. All constraints must be expressed in a clear, unambiguous manner, or the implementation will not run.

In addition to enforcing explicitness and consistency, a computational implementation allows researchers to cover much more data than they could by themselves. Huge arrays of inputs can be read into the program and their outputs may be checked against the program output. One of the perennial problems in linguistics is that theories are often evolved to explain a certain set of facts, and then when new facts appear, the theories are modified in such a way that they explain the new facts, but no longer explain some of the old ones. If implementations of the theories are employed, the researcher may keep databases of the facts they have explained and run implementations of the new theories on those data to ensure that the new theory is an improvement over the old one.

1.2.2 Implementations as Prods Toward Better Models

A good implementation of Optimality Theory must in itself embody a computable, efficient, and tractable model of the theory. It may, however, not be a valid one. However, if an invalid implementation is used often enough as a tool for analysis, eventually some researcher will come upon a set of data which

cannot be modeled by the implementation. If a new implementation is produced which models everything the old implementation did, plus the new set of data, then it will embody a better model of Optimality Theory.

1.3 Why Modify Primitive Optimality Theory

Three different approaches might be taken to the problem of creating a valid, computable formal model of Optimality Theory. One could start with the assumption that all functions from outputs to integers are valid constraints and begin to whittle away at the types of functions that are not computable, are intractable, and so on. One could attempt to encode every constraint in the Optimality Theory literature and try to find some reasonable description of what sort of functions these are, perhaps eliminating some constraints from consideration as intractable or incomputable. Finally, one could take a very simple and restricted model of Optimality Theory and expand or rework it until it can cover a high percentage of the empirical data. After having given in-depth consideration to the first two approaches, it was decided to adopt the third in this work. The reason for this decision is that the first approach seems unlikely to lead to a useful result (given that it is not informed by anything other than mathematical computability results, *i.e.,* it has nothing specifically to do with Optimality Theory), and the second, while probably a legitimate and valuable method to employ, would entail too much effort for any one person to complete within a year's time.

One of the most restricted types of grammars that allows infinite languages (Human languages are demonstrably infinite) is the Regular Grammar. These grammars are defined as the set of grammars that may be computed by Fi-

nite State Machines (FSMs)[3]. Ellison's (1994) model of Optimality Theory is a computable formal model which allows for any type of constraint which may be represented by a finite state transducer (FST, a finite state machine with outputs). This model takes the FSTs corresponding to the constraints, plus an FSM corresponding to the input, and combines them into a FSM corresponding to the output. Thus, this model allows constraints from a regular grammar.

Primitive Optimality Theory (OTP) (Eisner 1997a) is based on Ellison's mechanisms, but restricts constraints to those describable in a highly restricted subset of Regular Grammar[4]. It is reasonable, therefore, to suppose that OTP is the most restricted model of possible constraints that we might want to consider. By attempting to apply OTP to real data, it might be possible to build it up into a robust model of Optimality Theory. At any rate, something will certainly be learned from the attempt. OTP also has the benefit of providing a simple grammar in which to specify constraints, and its creator has provided a translation into OTP of many constraints from the Optimality Theory literature (Eisner 1997b).

For the reasons given above, this project consists of an implementation of the OTP model of Optimality Theory, its application to a complex phonological system, evaluations of the model's soundness and completeness, and some proposed changes to the model.

[3]These will be covered in more detail in the next chapter, and in Appendix A.

[4]In terms of Finite State Transducers, only constraints that may be expressed as deterministic transducers with fewer than ten states can be stated in OTP. It may be that multiple OTP constraints may act together to give the full set of finite state transductions.

CHAPTER 2

The Implementation

Having motivated the project, we will now move on to describe the implementation, first describing Primitive Optimality Theory itself, then presenting the usage and functionality of the implementation developed here, and finally touching on the major points of the design.

2.1 Primitive Optimality Theory

2.1.1 Background

As stated in the introduction, Mark Ellison (1994) showed how finite state transducers (finite state machines that output numbers of constraint violations[1]) can be used to represent Optimality Theoretic constraints, and how they can be combined to produce output forms. What he did not discuss, however, was how to create finite state machines that duplicate the standard constraint types of Optimality Theory. Part of the reason for this is that it cannot be done. Many of the constraints standardly used in Optimality Theory are too complex to be represented by finite state machines, because some constraints require potentially infinite memory. Examples of these constraints include constraints of base-reduplicant identity and many (though not all) gradiently-valued constraints.

[1]See Appendix A for an explanation of finite state machines and finite state transducers

These non-finite-state constraints will be discussed later. Because of these problems it does not seem possible to lay out a clear scheme for encoding all currently used Optimality Theoretic constraints as finite state automata. Primitive Optimality Theory is an attempt to define a class of constraints that can easily be converted into finite state transducers, but at the same time can duplicate the functionality of most well-motivated Optimality Theoretic constraints.

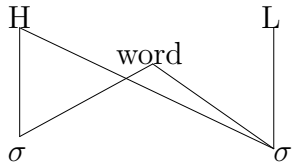
2.1.2 Primitive Optimality Theory's Representation

In order to simplify the task of producing finite state constraints, Primitive Optimality Theory does not employ the standard node-based representation of Feature Geometry (Clements 1985, Sagey 1986) nor the set- and index-based representation of Feature Classes[2] (Padgett 1995), but rather uses an edge- and alignment-based representation of gestural scores, based loosely on Correspondence Theory (McCarthy and Prince 1995), the theory of Generalized Alignment (McCarthy and Prince 1993) (although it does not maintain the gradient evaluation characteristic of Generalized Alignment, nor all of its descriptive power[3]), and Optimal Domains Theory (Cole and Kisseberth 1994). Primitive Optimality Theory thus represents utterances as time lines or gestural scores, with multiple tiers, each of which represents a type of constituent. Tiers can be used to represent autosegments such as tones, privative features such as [Nasal], prosodic structures, stress marks, morphological categories, and so forth. Within each tier, constituents are represented by noting their beginning and end along the time-line. Correspondence, set membership, and association of features are represented by a temporal overlap of these edges and interiors rather than by recourse

[2]However, it would probably be possible to adapt it to take advantage of the notions involved in Feature Class theory.

[3]See (Eisner 1997b) for a discussion of this.

to association lines as in Autosegmental Phonology (Goldsmith 1976). For example, in Autosegmental Phonology a two-syllable word with a high tone on the first syllable and a falling tone on the second, *e.g.,* [dónê], might be represented as follows (leaving out the segmental material),



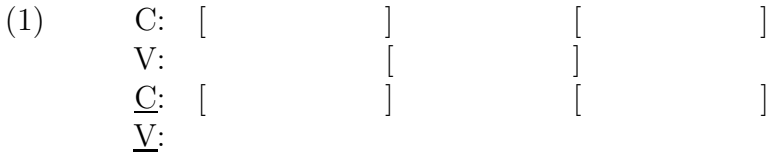
whereas Primitive Optimality Theory would employ a grid-like structure, as in the following,

Word:	[						]
HiTone:	[				]		
LoTone:					[		]
σ:	[		]				]

where "]" represents a right edge immediately followed by a left edge (in the implementation "|" is used for this symbol). In this structure we can see, for example, that the low tone is associated with the second syllable only, because the tone's interior overlaps with the interior of that syllable, but not with the interior of the other syllable. Association is thus represented by overlap of interiors, and inclusion by the exhaustive overlap of one interior by another. For example, in the OTP representation above, everything in the grid is included in the word because their interiors are all included within the interior of the word constituent, which is defined by its edges.

The representation in Primitive Optimality has one more facet: representations of output forms are assumed to contain the input forms from which they were derived. The input form has its own separate tiers, distinguished from the output tiers by underlining them. For example, the following gestural score rep-

resents an underlying CC sequence that has been broken up by the insertion of an epenthetic vowel:

(1)


This example brings up the dilemma of how to represent syncope and epenthesis in Primitive Optimality Theory. As originally defined by Eisner (1997a), epenthesis would be represented as in the above, and syncope as in the following:

(2)


Here an underlying vowel has disappeared on the surface, but it remains in the underlying form as "|", which in this case represents a left edge immediately followed by a right edge ("[]")[4]. These representations are sufficient for simple CV strings. However, they are inadequate for more complex structures. For example, in the following CCVC sequence,

(3)


if the first two consonants were deleted, the method of representation presented above would not be able to express the fact that the first, but not the second, of the consonants was underlyingly [+sonorant]. Upon being presented with this and other problems, Eisner (personal communication) devised the following solution: In addition to the surface and underlying tiers afforded by the framework,

[4]In the implementation "|" always represents a right edge followed by a left edge.

there is a DEL tier to represent deleted blocks of underlying material and an INS tier to represent inserted blocks of material. More specifically, a constituent on the DEL tier represents a period of time on the gestural score during which no surface material corresponds to any of the underlying material included within the constituent, and a constituent on the INS tier represents a period of time during which the surface material on all tiers has no underlying equivalent. The DEL tier represents deleted time with respect to the underlying tier, and the INS tier represents inserted time. No underlying material may overlap an INS constituent and no surface material may overlap a DEL constituent. It is important to note that these constituents do not represent epenthesis and syncope directly. For example, if a tone is inserted to align with a surface vowel that was underlying, the INS tier is not needed—the presence of the tone on the surface together with its absence on the underlying form represents the epenthesis.

With the addition of these tiers, the example in (1) could be represented as follows, with an INS constituent marking the locus of insertion:

(4)

C:	[		]	[		]
V:		[		]		
C:	[		]	[		]
V:						
INS:		[		]		

Constraints acting upon this representation will treat the right edge of the first underlying segment as adjacent to the left edge of the second. The deletion example of (2) would now be represented as follows, where constraints will see the two surface consonants as being adjacent:

(5)

C:	[		]	[		]
V:						
DEL:		[		]		
C:	[		]	[		]
V:		[		]		

These complications to the representation were formalized and implemented as part of this thesis.

2.1.3 Constraints in Primitive Optimality Theory

As we have seen, relationships between constituents in Primitive Optimality Theory are expressed in terms of temporal overlap between edges and interiors. Constraints in OTP are just a formalization of this idea. There are two types of constraints within OTP: *implication* constraints, and *clash* constraints. Implication constraints are of the form "(α_1 and α_2 and ... and α_n) $\rightarrow$ (β_1 or β_2 or ... or β_m)," where α_i and β_i are specifications of edges or interiors. For example, "]C" represents the right edge of a consonant, and "V" represents the interior of a vowel. The significance of an implication constraint is that for every instance on a time line where all of the edges or interiors in $\alpha_1 \ldots \alpha_n$ occur, at least one of the edges or interiors in $\beta_1 \ldots \beta_m$ must also occur during that same instance—they must overlap. For example, the constraint "word[$\rightarrow$ σ[" is more or less equivalent to the standard Optimality Theoretic constraint "ALIGN(word, L; σ, L)," and states that every word must begin with a syllable left edge, whereas "σ[$\rightarrow$ word[" is more or less equivalent to "ALIGN(σ, L; word, L)," and states that every syllable must begin a word, *i.e.,* that all words must be at most one syllable long.

In addition to edge alignment, implication constraints may be used to express correspondence and spreading constraints, among other things. For example, "MAX(sonorant)" might be expressed as "sonorant $\rightarrow$ sonorant," which may be interpreted as: "Every underlying sonorant must overlap with (project, correspond to) a surface sonorant." Spreading of low tones over syllables might be enforced by the constraint "L $\rightarrow$ σ[," that is, "Low tone interiors must span syl-

lable boundaries." This constraint will assign a violation for each low tone which does not span a syllable boundary. Establishing how far to spread will require additional constraints.

The second variety of constraints is the *clash* constraint, which is of the form "α_1 and α_2 and ... and $\alpha_n \perp \beta_1$ or β_2 or ... or β_m," or, alternatively, "$\alpha_1 \perp \alpha_2 \perp \ldots \perp \alpha_n$," where once again each α_i or β_i represents an edge or interior. A clash constraint assesses a violation for each instance on a time line in which all of the edges or interiors in $\alpha_1 \ldots \alpha_n$ and $\beta_1 \ldots \beta_m$ overlap. Thus, clash constraints are used to express constraints limiting alignment or forbidding certain feature combinations. For example, "*[+round, +low]" might be represented as "round $\perp$ low." The appearance of certain constituents may be penalized by forbidding them to overlap with themselves. For example, "$\sigma \perp \sigma$" assesses a violation for every syllable appearing within an utterance. Clash constraints can be used to constrain spreading. For example, "L $\perp$ $]\sigma$" states that low tones may not cross syllable boundaries. For more examples of constraints expressed in this notation, see Eisner (1997b) and the next chapter of this thesis.

2.2 Extensions to Primitive Optimality Theory

The implementation of OTP devised here provides a few extensions. First, it improves the representation of syncope and epenthesis by providing INS and DEL tiers, as described above. Second, it allows for the specification of *inviolable* constraints, meaning constraints that may not be violated in the output. The primary reason for this extension is to increase the speed with which the system processes data. Finally, the implementation adds many ease-of-use features, such as a facility for graphical examination of constraint and output finite state machines, automatic transcription of output forms into phonetic symbols, a facility

for viewing constraint violations of individual candidates, and a program that automatically generates constraint tableaux. These features will be described below.

2.3 Using the Implementation

The implementation discussed here consists of four separate executables: *generate*, *evaluate*, *viewcons*, and *tableaux*. *Generate* takes a set of constraints, a ranking, and a set of inputs, and derives the most harmonic output candidates consistent with the constraints and inputs. *Evaluate* takes a set of constraints, a set of inputs, and a set of corresponding output candidates and outputs the number of constraint violations incurred by the candidates against each constraint. *Viewcons* allows the user to view a picture of the finite state machines that represent the constraints. This feature is primarily useful for understanding how the system works and what a given constraint does. Finally, *tableaux* allows the user to specify constraint tableaux by specifying for each tableau an input, a set of output candidates, and a sequence of ranked constraints. With these specifications, *tableaux* calculates the violations for each candidate and outputs a constraint tableau, in text or LaTeX format[5].

The source code for these programs and a complete user's manual are available separately from the author. Here we will only briefly discuss the usage of the programs by walking through the stages of an analysis. When analyzing a language using the OTP implementation, the analyst first specifies a set of inputs in an inputs file. This file can also include specifications of the segments of the language and a specification of how to transcribe the output. An example input

[5]At this stage, *tableaux* has not yet been fully tested.

file for Turkish appears in Appendix B. Once the inputs have been entered, the analyst will specify a set of constraints that might generate the expected outputs. These will appear in a constraints file. A separate file records the rank order of the constraints with respect to one another and specifies which constraints are inviolable, that is, surface-true. A single constraints file could be used in combination with multiple rankings files to perform a cross-linguistic analysis. Having specified the constraints, rankings, and input files, the analyst then runs *generate*, supplying the appropriate filenames. *Generate* produces an output consisting of inputs paired with the winning output candidates, with the outputs represented as gestural scores. The output file also contains a phonetic transcription of each output, given according to the instructions in the input file.

Upon viewing the outputs, the analyst may see that the outputs generated by the analysis do not match the actual data (the expected outputs). He or she could then run *evaluate*, specifying the actual output received and the expected output, and see which constraints are responsible for the preference of the actual output over the expected one. The analyst would then adjust the constraints and/or rankings such that they prefer the correct output and rerun *generate*. When all of the output data to be accounted for have been generated, the complete analysis will be contained in the constraints and ranking files. This analysis will have certain advantages over a conventional one. It will have been conclusively shown to account for the data, with no possibility that some unexpected candidate would better fit the constraints than the actual data. In addition, the analysis will tend to be more complete than one generated by hand. At any rate, in order to write up this analysis, the analyst can then use the *tableaux* program to produce tables.

2.4 Design

Having briefly discussed the surface behavior of the OTP implementation, we will now move on to its design, starting with the overall algorithm, and finishing up with a presentation of the major component algorithms. Only the *generate* program will be covered here.

2.4.1 Overall Algorithm

Generate operates according to the following algorithm. It first reads the constraints file and builds corresponding finite state transducers in a process that will be discussed below. Once the constraints have been built, the program reads the rankings file and sorts the list of constraints accordingly. With the constraint ranking thus completed, the program enters into an input-output loop, reading inputs one at a time and generating the corresponding outputs.

The input-output loop proceeds as follows. The input is first created as a list of *labels*, where a label is a set of edge and interior specifications representing a single time slice from a gestural score. For example, an input comprised of a single voiced consonant of the form,

C:	[+]
Voice:	[+]

where "+" represents an interior, would be given as the list (⟨[C, [Voice⟩, ⟨+C, +Voice⟩, ⟨]C,]Voice⟩). This list is next converted into a finite state machine (FSM), in a process to be illustrated below. Once an input FSM has been constructed, the highest-ranked constraint finite state transducer (FST) is retrieved and examined. The program maintains a list of tiers which have been mentioned so far, which at this point includes only the input tiers. If the constraint men-

tions a tier that has yet to be seen, an FSM is constructed to enforce the most general requirements on any tier—that the brackets within it match. The tier FSM for the F ("foot") tier would appear as in Figure 2.1 (Eisner 1997a)[6]. The

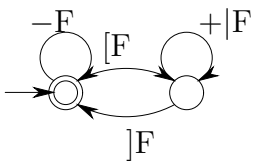


Figure 2.1: Tier FSM for the F tier

FSM embodies the requirement that the F tier must be occupied by zero or more minus signs (representing exteriors) followed optionally by a left bracket, which may be followed by zero or more "+" or "|" signs, where "+" represents the interior of a constituent and "|" the boundary between two adjoining constituents. The left bracket, after the optional "+" or "|" sign, must be followed by a right bracket, which brings us back to the beginning—this whole sequence may be followed by zero or more minus signs, a left bracket, etc. Thus, for example, "$--[+++|+++]--[+]$," "$[++]$," "$[+++]--$," and "$[++|++|+++++]$" are valid occupants of the tier, but "$-]$," "$[--]$," and "$]++[$" are not. At any rate, this tier FSM is *intersected*[7] with the input FSM.

A finite state machine represents a set of strings, and the intersection of the two finite state machines produces a new FSM that represents the intersection of the two sets of strings represented by the two FSMs. Thus the intersection of the input FSM with the tier FSM for the F tier produces a finite state machine that accepts all outputs containing the input and valid F constituents. Once

[6]See Appendix A for information on how to interpret FSMs.

[7]The details of this process may be found in Ellison (1994), and a mathematical definition may be found in Appendix A.

the appropriate tier FSMs have been intersected with the input, the constraint FST will be intersected with the resulting FSM. Note that the constraint is represented by a finite state *transducer* rather than a finite state machine. A finite state machine simply accepts a set of strings and rejects others, whereas a finite state transducer also produces outputs corresponding to the strings. In this case, constraint FSTs produce numbers of constraint violations incurred by particular outputs. Viewed as an FSM, constraint FSTs actually accept all possible outputs. Thus, when the constraint FST is intersected with the FSM that resulted from the last stage of operations, what results is an FST that accepts only the outputs accepted by that FSM, but which assigns the same violations as the constraint FST. This FST is then pared down by removing all but the lowest-cost paths, in terms of numbers of violations, from the beginning of the FST to each of its states, using Dijkstra's single-source shortest paths algorithm (Dijkstra 1959). The effect of this step is to produce an FST that accepts only the candidates to which the constraint assigns the smallest number of violations. To the standard single-source shortest paths minimization suggested by Eisner (1997a), this implementation adds a further step that removes all paths that do not lead to an accepting state[8]. The effect of this step is to make the resulting FST as small as possible without changing its behavior. At this stage the program retrieves the next constraint FST and intersects the FST from the previous step with it and with any appropriate tier FSMs. This process continues until no constraint FSTs remain. The result of this process is a finite state transducer that accepts only the most harmonic candidate or candidates, in the case of free variation. At this stage, *generate* employs Brzozowski's (1962) algorithm for finite-state minimization to produce the minimal equivalent FSM, converts the resulting FSM into

[8]See Appendix A for the definition of *accepting state*

a regular expression, and outputs the regular expression. Finally, the program evaluates the regular expression against the user's output segment definitions and outputs a corresponding transcription.

At this point the whole process repeats, if there are further inputs. Otherwise, the program terminates.

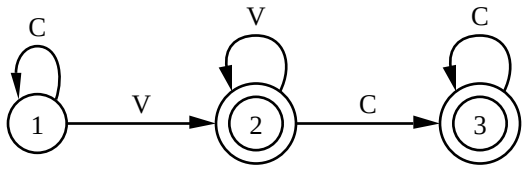
2.4.2 Component Algorithms

The major components of *generate* are FSM intersection, FSM minimization, building of constraint FSTs, and building of input FSMs. A description of the first of these components may be found in Ellison (1994), and the second is discussed in Ellison (1994), Dijkstra (1959), and Brzozowski (1962). Although there is not room here for the actual intersection and minimization algorithms, a short example will be given below, and a mathematical definition of FSM/FST intersection is given in Appendix A. In addition, the remaining components will be discussed as well.

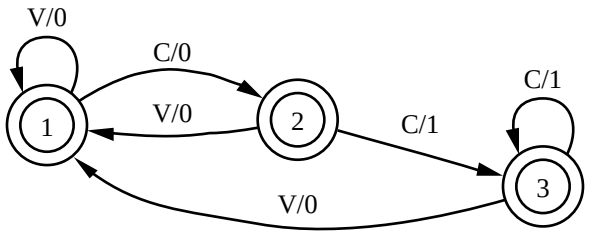
2.4.2.1 An Example of Intersection and Minimization

With only a few exceptions, the languages of the world limit their syllable inventory to syllables consisting of zero or more consonants followed by one or more vowels and ending with zero or more consonants. This set of possible syllables can be described by the *regular expression*[9] C*VV*C*, where "*" indicates zero or more of something. This regular expression is equivalent to the following finite state machine:

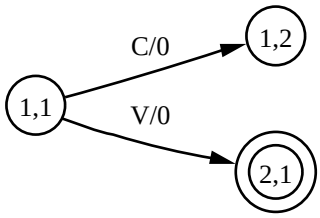
[9] see Appendix A for a short introduction to regular expressions.



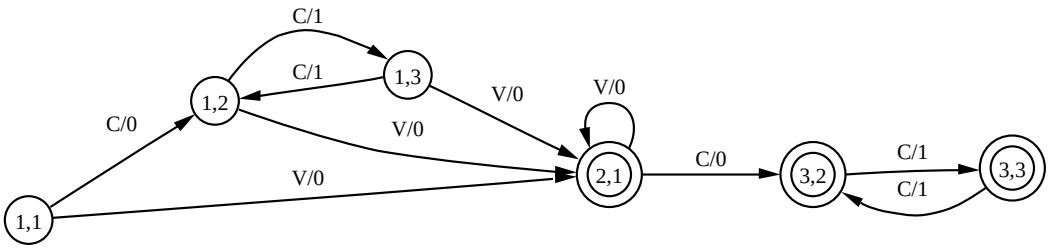
Now suppose that a particular language forbids adjacent consonants. This constraint would be expressed by the following finite state transducer:



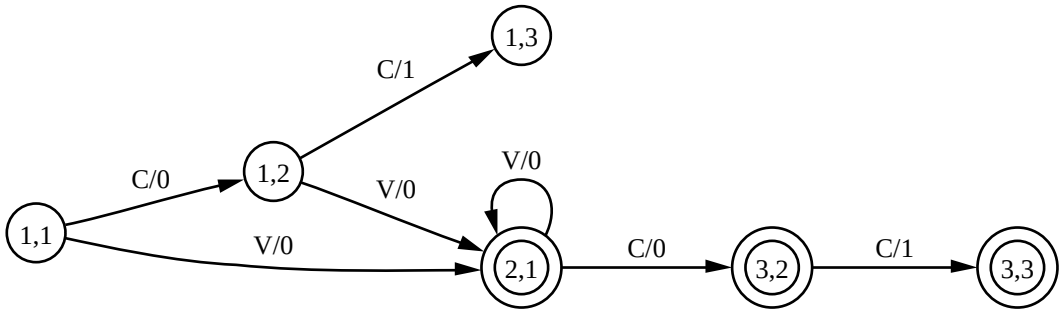
Intersection is basically a process of going simultaneously through two machines. Thus, to intersect the FSM with the FST, we start at the start state of each of them (state one), and create a new state $\langle 1, 1\rangle$. From here an input of "C" goes to state 1 in the first machine and to state 2 in the second, so we add state $\langle 1, 2\rangle$ with a transition to it from state $\langle 1, 1\rangle$, edge label "C/0". From the start states an input of "V" goes to state 2 in the first machine and to state 1 in the second, so we add state $\langle 2, 1\rangle$ with a transition to it from state $\langle 1, 1\rangle$, edge label "V/0". The result FST, at this point, appears as follows:



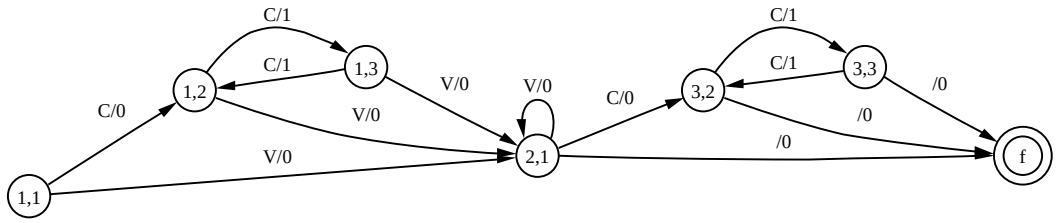
Notice that state $\langle 2, 1\rangle$ is labeled as an accepting state, because the corresponding states in the two machines are both so labeled. Now, from state $\langle 2, 1\rangle$, we see that "V" leads in machine 1 to state 1 and in machine 2 to state 1. Thus we add an edge labeled "V/0" from $\langle 2, 2\rangle$ to state $\langle 1, 1\rangle$. This process continues in the same way, giving us the final result, an FST that accepts all strings the FSM would have accepted and assigns a violation whenever the input FST would have:



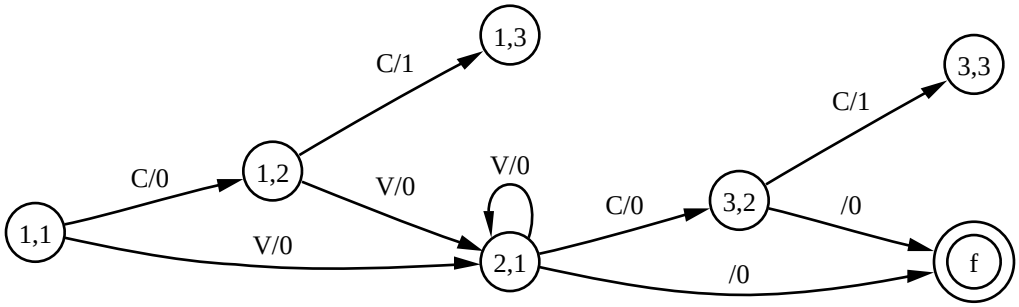
At this point we need to remove all of the non-optimal candidates from the FST. This brings up an instance of inexplicitness in Eisner's representation of the process. Eisner (1997a) states that the application of Dijkstra's Single-Source Best Paths algorithm to an FST will remove all non-optimal candidates, but, while this is the case for most FSTs, it is not always true. In this case, for example, applying the Best Paths algorithm (which removes all edges that are not in the best path, in terms of minimizing output violations, from the start state to some state in the FST) yields the following FST:



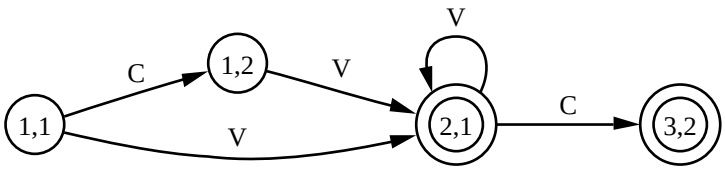
Removing dead-end paths gets rid of state $\langle 1, 3 \rangle$, but note that this machine still accepts consecutive consonants, contrary to the constraint. Instead of getting the best paths to every state in the machine, we actually want the best paths to any one of the final states. This can be accomplished by adding a fake final state led to from every final state of the intersection:



Applying the Best Paths algorithm to this FST yields the following FST:



Removing dead-end paths gets rid of states $\langle 1, 3\rangle$ and $\langle 3, 3\rangle$. If we then remove the fake final state and mark the states that led to it as acceptors, we get the correct FST. We now remove the outputs from each edge, as what we have produced is actually the FSM representing all syllables without adjacent consonants. The final result is as follows:



2.4.2.2 Building Constraint Transducers

A constraint FST represents a single clash or implication constraint, as defined in section 2.1.3. An implication constraint $\alpha \rightarrow \beta$ outputs a violation for each instance of α during which none of the edges or interiors of β occurs, and a clash constraint $\alpha \perp \beta$ outputs a violation for each simultaneous occurrence of α and β. The primary idea of how to generate FSTs from these constraints comes from Eisner (1997a), but as stated previously, the addition of the DEL and INS representations greatly complicates matters. Constituents on the input tier that overlap with DEL constituents must be invisible to constraints of the "Dep" variety, because these constraints require that surface forms correspond to underlying forms, and deleted underlying forms should not count. Additionally, account must be taken of the fact that underlying edges separated by INS constituents are deemed to be adjacent, as in example (4), and similarly with surface constituents separated by DEL constituents, as in example (5).

In order to construct finite state transducers corresponding to constraints, we employ a Boolean algebra over edge-interior descriptions, called *tier descriptors*.

The algebra is defined only within single time slices, thus an expression in the algebra must necessarily refer to a single time slice. A tier descriptor gives the possible symbols (from the alphabet $\Sigma = \{+, |, -, [,]\}$) that may appear on a particular tier during a given time slice. For example, the tier descriptor "–+[F" signifies that the *F* tier may contain a constituent interior ("+"), exterior ("−"), or left edge ("["). These tier descriptors are the basic symbols of the algebra, to which we add the Boolean operators for *not, and,* and *or. Not*, applied to a tier descriptor ϕT, gives the set negation $(\Sigma - \phi)$ of the set of symbols employed in that descriptor. For example, the negation of "–+[F" is "]|F," which contains all of the possible time-line symbols that the first did not. *And,* applied to two or more tier descriptors, states that the time slice must comply with both of them. For example, "–+[F and +]F" is equivalent to "+F," "–+[F and]F" is equivalent to *false* (no possible time slice could comply), and "–+[F and +]σ" is equivalent to the *label*[10] $\langle$–+[F, +]$\sigma\rangle$. *Or*, applied to two or more tier descriptors, states that the time slice must comply with either of them. For example, "–+[F or +]F" is equivalent to "–+[]F", "–+[F or]|F" is equivalent to *true* (equivalent to "–+[]|F", with which all possible time slices comply), and "–+[F or +σ" is irreducible. Edge labels in constraint FSTs are equivalent to an *and* expression comprised of one or more tier descriptors. The basic constraint FST types are defined by listing their edges in this algebra. In Eisner's original design, there were four basic types of constraint FSTs: implications whose left sides contain only interiors of constituents, implications whose left sides contain edges, clashes concerning only interiors, and clashes containing edges. These could all be implemented with one or two-state FSTs. With the complications added by the INS and DEL tiers, there are now sixteen different basic constraint transducer types, and

[10]See section 2.4.1, paragraph 2.

their implementation requires up to seven states and many times as many edges. Due to space limitations, it is not possible to lay out how to construct each of the new basic FSM types here, but this information will be provided by request, along with the code and the user's manual for the implementation. To actually construct these FSTs, the edge expressions must be simplified into conjunctive normal form—disjunctions of conjunctions of tier descriptors—where each conjunction of tier descriptors will be realized as a distinct edge in the resulting FST. The process of simplification makes use of the equivalences (using prefix notation, where a and b are arbitrary expressions) in Table 2.1[11], as well as the

and(a, and(b, c))	$\equiv$	and(a, b, c)
or(a, or(b, c))	$\equiv$	or(a, b, c)
and(a, b)	$\equiv$	and(b, a)
or(a, b)	$\equiv$	or(b, a)
and(a, a)	$\equiv$	and(a)
or(a, a)	$\equiv$	or(a)
and(a)	$\equiv$	a
or(a)	$\equiv$	a
not(not(a))	$\equiv$	a
or(a, b, c)	$\equiv$	or(a, c) if $a \supset b$

Table 2.1: Equivalences exploited in expression simplification

tier-descriptor-specific equivalences laid out previously. In addition to simplifying the expressions, they must also be disambiguated. Because it must always be clear which edge to take from a give state with a given time slice, a given time slice may be accepted by at most one of the subexpressions of a disjunction. The expression "a or b" is disambiguated by calculating the overlap "overlap(a, b)" between the two expressions, where "overlap(a, b)" is an expression describing all

[11] In this table, "$a \supset b$" means that all of the time slices that meet b's requirements also meet a's requirements

time slices that match both a and b, and either replacing a with "(not(overlap(a, b)) and a)" or replacing b with "(not(overlap(a, b)) and b)."

Thus, constructing a finite state transducer to represent a given constraint is a matter of examining the transducer and selecting the proper FST type from the sixteen basic types, encoding the particulars of a constraint into the appropriate edge expressions, simplifying the expressions, disambiguating them, and then converting the resulting expressions into edge labels.

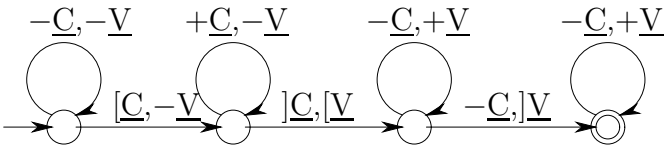
2.4.2.3 Building Input FSMs

The FSM representing an input needs to accept all possible outputs that include the input. When building one, we must take into account the fact that the length of exteriors and interiors of constituents is not specified, only the alignment of the edges with respect to one another. The procedure for building an input FSM begins with a list of labels (time slices) representing the input. Some of the labels have edges, and some do not. Since the length of exteriors and interiors is not specified, when building the input FSM we require there to be zero or more of each non-edge-containing time slice type. For example, for the input "CV":

$$\begin{array}{ll} \underline{\mathrm{C}}: & [\quad\quad\quad\quad\quad\quad] \\ \underline{\mathrm{V}}: & \quad\quad\quad\quad\quad\quad [\quad\quad\quad\quad\quad\quad] \end{array}$$

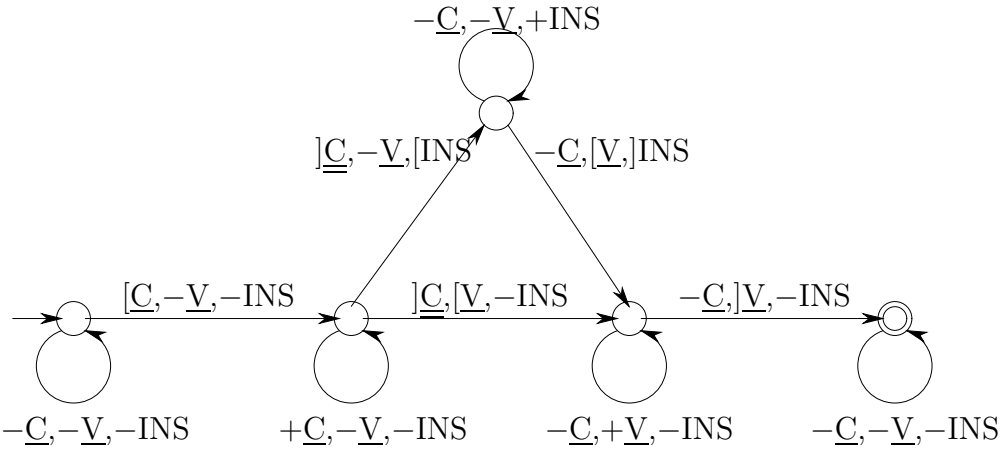
there are seven time slices to be considered: (1) the area before the consonant, specified $\langle -\underline{\mathrm{C}}, -\underline{\mathrm{V}} \rangle$; (2) the left edge of the consonant, specified $\langle [\underline{\mathrm{C}}, -\underline{\mathrm{V}} \rangle$; (3) the interior of the consonant, specified $\langle +\underline{\mathrm{C}}, -\underline{\mathrm{V}} \rangle$; (4) the right edge of the consonant and the left edge of the vowel, specified $\langle]\underline{\mathrm{C}}, [\underline{\mathrm{V}} \rangle$; (5) the interior of the vowel, specified $\langle -\underline{\mathrm{C}}, +\underline{\mathrm{V}} \rangle$; (6) the right edge of the vowel, specified $\langle -\underline{\mathrm{C}},]\underline{\mathrm{V}} \rangle$; and (7) the area after the vowel, specified $\langle -\underline{\mathrm{C}}, -\underline{\mathrm{V}} \rangle$. Note that slices (1), (3), (5) and (7) have no edges, and thus may occur an unspecified number of times. Thus,

temporarily disregarding the possibility of epenthesis, we have the following FSM:



Note that if a tier is not explicitly specified on a given edge, then any value may appear there.

This FSM is not complete, however. There is still the possibility of epenthesis between the C and the V (epenthesis to the left and right of the utterance is already possible with the current input FSM) to be accounted for. In order to account for this possibility, we must specify the INS tier and add edges and states corresponding to the possibility of inserted material. The final FSM, accounting for insertion, is as follows:



In the general case, such an "insertion pyramid" must be constructed wherever a left edge and a right edge overlap. Thus, building an input takes a list of *labels* (time slices) specified by the user, identifies those with no edges and implements them as FSM self-edges, implements the rest as FSM state-to-state edges, and adds "insertion pyramids" wherever epenthesis is possible.

CHAPTER 3

An Analysis of Turkish Harmony and Disharmony

In order to evaluate the effectiveness of this implementation and see whether Primitive Optimality Theory has enough descriptive power to handle complex phenomena, it was decided to attempt an analysis in Primitive Optimality Theory of the Turkish harmony and disharmony phenomena described in Clements and Sezer (1982)[1]. These vowel and consonant harmony and opacity effects, while quite complex, do not appear at first sight to require any descriptive power beyond the finite state—*e.g.,* they involve no copying and no long-distance metathesis. Thus, they should be a good test of whether OTP can handle complex, but still finite-state, phenomena.

It is necessary at this point to warn the reader that the upcoming analysis will not be a conventional one, for two reasons. First, the presentation style will be unusual. Due to the fact that this analysis is implemented, and that the implementation implicitly considers every possible output candidate for every input, if the implementation produces the actual output form found in the data, then it is certain that all possible candidates were considered, and it is not necessary to prove explicitly that the analysis precludes all incorrect candidates for that input.

[1] All empirical statements in this chapter, unless stated otherwise, may be assumed to be taken from this source.

Second, the analysis itself will be different. The usual goal of linguistic analyses is to describe the phenomena under consideration while minimizing the number of constraints used[2], maximizing their simplicity, and maximizing the potential of the constraints for typological coverage. The goals of this analysis were different, however, and will lead perhaps to a different type of analysis. In particular, the goals were to analyze the phenomena of Turkish harmony and disharmony within the formal limitations of Primitive Optimality Theory, and for the analysis thus produced to allow the *generate* program to actually generate the output data in a reasonable amount of time—*i.e.,* the analysis should be computationally tractable. Due to the computational characteristics of Primitive Optimality Theory as implemented here, this goal leads to constraint systems containing a very large number of sometimes redundant constraints, as it is necessary to limit as early as possible the growth of candidate spaces—weak, redundant highly-ranked constraints, although not necessary for covering the data, are essential for this purpose. It is also necessary to limit as much as possible the number of tiers considered, as the size of the finite state machines produced at a given level grows exponentially with the number of tiers at that level that have weak specifications. Finally, wherever the relative ranking of constraints with respect to one another is not constrained by the data itself, the computational characteristics of Primitive Optimality Theory impose a preferred ordering. Whenever possible, it is preferable for constraints to be clustered such that new tiers are introduced as slowly as possible, for the reasons given above. If the human brain actually deals in constraints, and if they are represented in any similar way to the representation used here, there may be an impetus to cluster constraints in

[2]Although this is not necessarily a goal for those linguists following the assumption that all constraints are innate, it is a consideration even for them to the extent that there should be some attempt to minimize the number of hypothesized innate constraints, due to the finite size of the human brain.

this way as well, and to produce redundant constraints.

3.1 Features

To begin the analysis, it is first necessary to represent the segments and features of Turkish. Primitive Optimality Theory, like Optimal Domains Theory (Cole and Kisseberth 1994), supports only privative features. The privative features used are: [Nasal], to indicate opening of the velar port; [Labial], to indicate use of the lips as an articulator; [Coronal], to indicate use of the tongue blade; [Anterior], to indicate constriction in the front part of the mouth; [Voiced], to indicate vibration of the vocal cords; [Continuant], to indicate absence of closure; and [Non-Strident], to indicate a low level of acoustic noise. The feature [Non-Strident] was chosen in preference to [Strident] for reasons of analytic convenience, and assumes the independent existence of a complementary [Strident] feature, unnecessary for the present analysis. In order to analyze Turkish harmony, it was necessary to invoke binary and ternary features, in addition to the privative features listed above. These may be represented within OTP by postulating multiple privative features with constraints forbidding overlap and requiring specification. For example, the binary feature [±round] was modeled by two privative features: [Round] and [Flat], with the additional constraints RDFLTMANDATORY ("V → Round or Flat") and *RDFLT ("Round ⊥ Flat"). Similarly, the ternary feature [Height] was modeled by three privative features: [High], [Low], and [Non-peripheral] (abbreviated "NP"), along with the following constraints:

*HILO:	High ⊥ Low
*HINP:	High ⊥ NP
*LONP:	Low ⊥ NP
ASSIGNHEIGHT:	V → High or Low or NP

Additional binary features are [±back], modeled by [Palatal] and [Velar], and [±sonorant], modeled by [Sonorant] and [Obstruent]. A specification of the features of the segments of Turkish may be found in Table 3.1. Note that in this table "N" stands for "not specified," and signifies that it is immaterial to this analysis what value the segment has for that feature, both in the input and in the output.

3.1.1 Input-Output Constraints

For each of the features listed in Table 3.1 there is a corresponding MAX constraint to cause that feature to surface, and a DEP constraint to prevent the feature from being introduced anywhere except where it is underlying. There are several possible ways to encode these requirements in OTP. Here, the simplest of them is used—surface edges must align with underlying edges, and vice versa. For example, the MAX constraint for [Voice] appears as follows:

Voice[→ Voice[,
]Voice →]Voice

and the corresponding DEP constraint is similarly:

Voice[→ Voice[,
]Voice →]Voice

Note that each of these Optimality Theoretic constraints actually consists of two Primitive Optimality Theoretic constraints whose ranking is arbitrarily fixed with respect to one another[3]. It is often the case that a single logical constraint will be comprised of multiple primitive constraints.

[3]Conceptually the ranking is equal, but as implemented the constraints are ranked in the order given—the sense in which the ranking is fixed is that no rankings file can change the ranking of the constraints with respect to one another.

Segment	Round	Flat	Palatal	Velar	High	Low	NP	Nasal	Sonor.	Obstruent	Labial	Coronal	Anterior	Voiced	Continuant	Non-strident
i	−	+	+	−	+	−	−	−	+	−	N	N	N	+	+	+
ɨ	−	+	−	+	+	−	−	−	+	−	N	N	N	+	+	+
ü	+	−	+	−	+	−	−	−	+	−	N	N	N	+	+	+
u	+	−	−	+	+	−	−	−	+	−	N	N	N	+	+	+
e	−	+	+	−	−	−	+	−	+	−	N	N	N	+	+	+
ɨ	−	+	−	+	−	+	−	−	+	−	N	N	N	+	+	+
ö	+	−	+	−	−	−	+	−	+	−	N	N	N	+	+	+
o	+	−	−	+	−	−	+	−	+	−	N	N	N	+	+	+
p	N	N	N	N	−	N	N	−	−	+	+	−	+	−	−	+
b	N	N	N	N	−	N	N	−	−	+	+	−	+	+	−	+
f	N	N	N	N	−	N	N	−	−	+	+	−	+	−	+	+
t	N	N	N	N	−	N	N	−	−	+	−	+	+	−	−	+
d	N	N	N	N	−	N	N	−	−	+	−	+	+	+	−	+
s	N	N	N	N	−	N	N	−	−	+	−	+	+	−	+	−
z	N	N	N	N	−	N	N	−	−	+	−	+	+	+	+	−
č	N	N	N	N	+	N	N	−	−	+	−	+	−	−	−	−
ǰ	N	N	N	N	+	N	N	−	−	+	−	+	−	+	−	−
š	N	N	N	N	+	N	N	−	−	+	−	+	−	−	+	−
ž	N	N	N	N	+	N	N	−	−	+	−	+	−	+	+	−
k	N	N	−	+	+	N	N	−	−	+	−	−	−	−	−	+
g	N	N	−	+	+	N	N	−	−	+	−	−	−	+	−	+
k^j	N	N	+	−	+	N	N	−	−	+	−	−	−	−	−	+
g^j	N	N	+	−	+	N	N	−	−	+	−	−	−	+	−	+
m	N	N	N	N	−	N	N	+	+	−	+	−	+	+	−	+
n	N	N	N	N	−	N	N	+	+	−	−	+	+	+	−	+
v	N	N	N	N	−	N	N	−	+	−	+	−	+	+	+	+
l	N	N	−	+	+	N	N	−	+	−	−	+	+	+	−	+
l^j	N	N	+	−	+	N	N	−	+	−	−	+	+	+	−	+
r	N	N	N	N	−	N	N	−	+	−	−	+	+	+	−	+
y	N	N	N	N	+	N	N	−	+	−	−	+	−	+	−	+
h	N	N	N	N	−	N	N	−	+	−	−	−	−	−	+	+

Table 3.1: Turkish Segments

In addition to the features listed above, each Turkish segment is associated with a constituent of the "C" or "V" tiers. These represent a combination of root nodes and the [±syllabic] feature. C and V are subject to MAX and DEP constraints equivalent to those given above, and are also restrained to be in complementary distribution by *CV ("C ⊥ V"). In addition to these requirements, C and V constituents are required to be adjacent to one another (no gaps are permitted) by the constraints PACKVS ("Word and]V → C[or V[") and PACKCS ("Word and]C → C[or V["). *Word* refers to the prosodic word, which brings up the subject of morphology. Input forms in this analysis consist of Root and Suffix constituents together with the features, C constituents, and V constituents within them. Root and Suffix constituents project corresponding surface Root and Suffix constituents via the usual DEP and MAX constraints, and they also project the Word constituent according to the following constraints:

PROJECTWD:	Root → Word,
	Suffix → Word,
	Word → Root
*WD:	Word ⊥ Word

Of these, PROJECTWD says that every Root must overlap with a Word, as must every Suffix, and every Word must overlap with a Root. This constraint is inviolable. *WD must be ranked below PROJECTWD, and serves to prevent the appearance of multiple words within a single root.

3.2 Syllabification

Turkish syllables are maximally of the form CCVCC[4]. However, syllable-initial consonant clusters only appear in careful speech, and only certain consonant

[4]Also CCVV, but this analysis does not attempt to account for long vowels.

combinations are possible in syllable-final clusters. The analysis here attempts to account only for colloquial speech, and thus bans syllable-initial clusters. In order to get these patterns, it is first necessary to create syllables and moras. For expository purposes, the constraints governing the distribution of syllables and moras have been divided up depending on their universality, violability, and contents.

The following constraints might be considered universal principles of syllabic representation, rendered here as universally inviolable constraints:

AlignSyl:	$\sigma[\rightarrow$ C[or V[, $]\sigma \rightarrow]$C or]V
AllInWord:	$\sigma \rightarrow$ Word
PackSyls:	Word and $]\sigma \rightarrow \sigma[$

These include AlignSyl, which requires syllables to align with segment boundaries (so no segments can be ambisyllabic—geminates are represented by a featural specification linked to two C or V constituents); AllInWord, which forbids syllables from appearing outside of words; and PackSyls, which forbids gaps between syllables.

The next set of constraints are the principles of the moraic representation, once again hypothesized to be universally inviolable constraints:

PackMoras:	Word and $]\mu_s \rightarrow \mu_s[$ or $\mu_w[$,
	Word and $]\mu_w \rightarrow \mu_s[$ or $\mu_w[$
Align(σ, L; μ_s, L):	$\sigma[\rightarrow \mu_s[$
Align(σ, R; μ, R):	$]\sigma \rightarrow]\mu_s$ or $]\mu_w$
Align(μ_s, L; σ, L):	$\mu_s[\rightarrow \sigma[$
$^*\mu_s\mu_w$	$\mu_s \perp \mu_w$
Parse(μ_s):	$\mu_s \rightarrow \sigma$
Parse(μ_w):	$\mu_w \rightarrow \sigma$
*Repeatμ_w:	$]\mu_w \perp \mu_w[$
IdentLength:	$\mu_s[\rightarrow$ C[or V[, $\mu_w[\rightarrow$ C[or V[,
	$]\mu_s \rightarrow]$C or]V, $]\mu_w \rightarrow]$C or]V

Of these, PACKMORAS forbids gaps between moras, ALIGN(σ, L; μ_s, L) requires syllables to begin with strong moras, ALIGN(σ, R; μ, R) requires syllables to end with moras, ALIGN(μ_s, L; σ, L) requires strong moras to begin syllables, $^*\mu_s\mu_w$ forbids strong and weak moras from overlapping, PARSEμ_s and PARSEμ_w require moras to be in syllables, *REPEATμ_w forbids adjoining weak moras, and IDENTLENGTH acts similarly to ALIGNSYL in forbidding ambimoraic C/V constituents (it requires moras to line up with C or V boundaries). The idea of IDENTLENGTH and ALIGNSYL is that C and V act as time slots.

Of the following constraints many are almost universal, but may be violable in some languages:

PARSEC:	C $\rightarrow$ σ
PARSEV:	V $\rightarrow$ σ
HNUC:	σ $\rightarrow$ V
ALIGN(Wd, L; σ, L):	Word[$\rightarrow$ σ[
ALIGN(Wd, R; σ, R):	]Word $\rightarrow$]σ

They are inviolable in Turkish. Here PARSEC and PARSEV require every segment to be in a syllable (no extrasyllabic segments), HNUC requires every syllable to contain a vowel, ALIGN(Wd, L; σ, L) requires words to begin with syllables, and ALIGN(Wd, R; σ, R) requires words to end with syllables. The joint effect of these constraints is to require that every syllable have a vowel in it and to require that no extrasyllabic segments or features appear within a word.

The following constraints regulate moras in Turkish, and are inviolable in most languages:

CODAμ_w:	]V $\rightarrow$ V[or]σ or μ_w[
	]V and V[and σ $\rightarrow$ μ_w[
NUCμ_s:	μ_s $\rightarrow$ V
μ_wSON:	μ_w $\perp$]C and V[

Codaμ_w specifies that weak moras begin after the first vowel of a syllable, Nucμ_s specifies that a strong mora includes a vowel (the syllable nucleus), and μ_wSon forbids weak moras from being CV sequences ("weak moras must decrease in sonority").

There are only three violable syllable-structure constraints in this analysis:

Ons:	$\sigma[\rightarrow$ C[
*Coda:	$\mu_w \perp \mu_w$
*BrOns:	$\mu_s \perp]$C and C[

Ons requires syllables to have onsets, *Coda forbids syllable codas, and *BrOns forbids branching onsets. These three are crucially ranked below the other syllable constraints, as well as Dep(C), Max(C), and Max(V). Among these three, the ranking Ons $\gg$ *BrOns $\gg$ Dep(V) $\gg$ *Coda enforces syllables with no initial consonant clusters (underlying clusters are broken up by vowel epenthesis), and with consonant clusters allowed in the coda. In addition to the requirements listed above, there are requirements on allowable coda clusters. Allowable coda clusters are:

- sonorant + obstruent
- voiceless fricative + obstruent stop
- k + s

These requirements are embodied in the rather complicated constraint CodaClusterProps ("Coda Cluster Properties"):

]C and C[and $\mu_w \rightarrow$ Obs[,
]C and C[and μ_w and]Obs $\perp$]Voiced,
]C and C[and μ_w and]Obs $\rightarrow$]Cont or]High,

]C and C[and μ_w and]Obs and]High $\rightarrow$]Non-strident,
]C and C[and μ_w and]Obs and Obs[and]Cont $\perp$ Cont[,
]C and C[and μ_w and]Obs and]High and]Non-strident $\rightarrow$ Cont[,
]C and C[and μ_w and]Obs and]High and]Non-strident $\rightarrow$ Anterior[
]C and C[and μ_w and]Obs and]High and]Non-strident $\rightarrow$ Coronal[,
]C and C[and μ_w and]Obs and]High and]Non-strident $\perp$ Voiced[

Each line of this constraint begins with "]C and C[and μ_w," which signifies that coda clusters are being constrained. The first line says that all coda clusters must have an obstruent as the second member. The next line states that if the first member is an obstruent, it must be voiceless. The third line states that if the first member is an obstruent, it must either be a fricative or [+High]. The fifth line states that if the first member is a voiceless fricative then the second member must be an obstruent stop. The remaining lines combine to say that if the first member is an obstruent stop, then the cluster must be "ks." CodaClusterProps is interspersed in the constraint ranking between *BrOns and Dep(V), and is crucially outranked by the consonantal[5] Max and Dep constraints for the features mentioned in it.

3.2.1 Final Devoicing

Word-final obstruent stops in Turkish may not be voiced. This restriction is implemented by the constraint FinalObstruentDevoicing ("]Obs and]Word $\perp$]Voiced"), which devoices word-final obstruents. This constraint is crucially outranked by VoiceSon, which forces sonorants to be voiced, and MaxCnt(Voice) (preserve voicing in underlying continuants), which exists only because [Continuant] is a privative feature and as such its reverse cannot directly be referred to in

[5]This qualification is necessary because the [High] feature has a stronger Dep constraint for consonants than for vowels—the reason for this is to allow for a default of [High] for epenthetic vowels (see Section 3.2.2.). Except for [Voiced], which will be discussed shortly, the other features mentioned do not split up their Max and Dep constraints.

FinalObstruentDevoicing, which ought to be a constraint that devoices final obstruent stops. Final devoicing is mentioned in this context because it must intermix with constraints of syllabification. With final devoicing included, the final ranking for constraints in this area is Ons ≫ *BrOns ≫ Max and Dep for [Obstruent], [Coronal], [Anterior], [Continuant] and [Non-strident], plus DepC(Hi), Max(Hi), Segmental(Hi), Segmental(Vcd), Max(Son), *SonObs, VoiceSon, MaxCnt(Vcd) ≫ CodaClusterProps ≫ Dep for [Voice], [Son], and V ≫ FinalObstruentDevoicing ≫ Max(Vcd) ≫ *Coda.

3.2.2 Constraints on Epenthesis

As noted above, Turkish employs vowel epenthesis to break up illegal consonant clusters. Epenthesis occurs between the offending consonants rather than after them. This distribution of epenthesis must be enforced, because post-cluster epenthesis often produces erroneously better candidates with respect to vowel harmony. Inter-cluster epenthesis is enforced by constraints on the alignment of morphemes. When a segment is inserted at the periphery of a morpheme, Primitive Optimality Theory as implemented here treats the segment as being outside of that morpheme, and therefore if morphemes are required to be adjacent and to be aligned with the prosodic word, then epenthesis between morphemes or in word-final position cannot occur. The relevant constraints are:

AlignWdMorph:	]Word →]Root or]Suffix
NoMorphGaps:	]Root → Suffix[or]Word,
	]Suffix → Suffix[or]Word

An additional note with regard to epenthesis is that epenthetic vowels are invariably [High]. This is enforced by ranking the constraint VowelHeightDefaults, which provides a height default of [High], above Dep(Hi).

3.3 Vowel Harmony

The basic story of vowel harmony in Turkish is that a vowel will agree in backness with the vowel to its left, and a high vowel will agree in roundness with the vowel to its left. This simple story is, however, complicated by vowel and consonant opacity effects, marked and unmarked vowels, and the existence of leftward harmony. Vowel harmony in [±back] and [±round] will be treated separately here, but the basic analysis for each will be the same.

3.3.1 Backness Harmony

The analysis to be given here will be expressed in terms of Optimal Domains Theory (Cole and Kisseberth 1994). Harmony derives from a desire to avoid effort by maintaining articulators in a steady state, and it is opposed by a desire for clarity of expression. The latter impulse is expressed here primarily by reference to constraints of input-output correspondence, and the former by the construction of domains in which harmony is enforced. In standard Optimal Domains Theory, three special constraint types are provided to construct these domains and enforce harmony within them. First, there is *Basic Alignment*, which aligns the harmony domain with the segment from which the domain was projected. For example, here a [+back] vowel would project a [+back] harmony domain, which would contain nothing other than that vowel if Basic Alignment (BA) were dominant. The second constraint type is *Wide Scope Alignment* (WSA), which tries to extend the harmony domain to coincide with some other domain. Here, WSA wants the [+back] alignment domain to align with the word boundary. In standard Optimal Domains Theory, WSA constraints are gradiently evaluated, so a constraint violation is assessed for every segment intervening between the edge of the harmony domain and the edge of the word. Finally, Optimal Domains

Theory provides *Express* constraints to mandate that the harmony domain be expressed. Here the Express constraints assess a violation for every [-back] vowel within a [+back] harmony domain. If Basic Alignment is ranked below Wide Scope Alignment, no harmony occurs. If no highly ranked constraints interfere with Express, and Wide Scope Alignment is ranked above Basic Alignment, then unrestrained harmony will occur. If some interfering constraint out-ranks Express, then the type of harmony that occurs will depend on the relative ranking of Express versus Wide Scope Alignment. If Express out-ranks Wide Scope Alignment, then there will be opacity effects. Any vowel that may not be [+back] due to highly ranked constraints will block the spread of [+back] harmony. On the other hand, if Wide Scope Alignment out-ranks Express, then there will instead be *transparency* effects. Any vowel that may not be [+back] due to highly-ranked constraints will surface as [-back], but [+back] harmony will continue to spread beyond that vowel.

Applying Optimal Domains Theory to this analysis is complicated by two factors. First, not only does [+back] (here called [Velar]) spread from vowel to vowel, but [-back] ([Palatal]) spreads as well. Second, Primitive Optimality Theory does not allow direct expression of most gradiently evaluated constraints. The first of these factors would complicate any analysis using Optimal Domains Theory, but the second is unique to Primitive Optimality Theory. Unique, that is, except that it would also apply to any finite-state implementation of Optimality Theory. As Ellison (1994) points out, gradiently-valued constraints may be simulated by binary-valued constraints in the case where a single constituent is to be penalized gradiently for its distance from a particular edge. This simulation is done by projecting a new constituent on a new tier, with the new constituent extending from the constituent to be evaluated to the edge the distance to which is to be measured. The gradiently valued constraint is then simulated by penalizing

every appearance of some other constituent (for example, every syllable) within the newly projected constituent. For example, a gradient constraint that assesses one violation for every syllable intervening between the primary stress of a word and that word's right edge can be simulated by projecting on a new tier a constituent that extends from the primary stress to the right edge of the word and employing a constraint that penalizes every overlap of this new constituent with syllable edge. This strategy only works, however, when there is a known number of constituents to be evaluated gradiently. For example, a gradient constraint that assesses a violation for every syllable intervening between *any* stress in a word and the word's right boundary could not be modeled this way, because it would require that a constituent be created extending from each of the stresses to the edge of the word. These constituents would have to be on separate tiers, or else they would overlap, which is not possible in this representation. However, there is no way using finite state methods to know how to pair each stress up to a different tier, as such a pairing would require the ability to count, something that finite state machines cannot do. Thus, gradiently evaluated constraints referring to an unbounded number of constituents cannot be expressed with finite methods. At any rate, the first complicating factor can be overcome by postulating two competing harmonic domains: a [Palatal] spreading domain, and a [Velar] spreading domain. The extent to which one domain succeeds in spreading at the expense of the other depends on the relative ranking of the spreading constraints of the one versus the existence and spreading constraints of the other. The complication of gradient evaluation is overcome by separating Wide Scope Alignment into separate constraints, one of which limits the domain from spreading outside of the enclosing domain (here [Word]), and the others of which encourage the harmony domain to be as large as possible. Here the first type of these retains the name of Wide Scope Alignment, and the others will be referred to as *Ex-*

tension constraints. With this split, the difference between transparency and opacity effects will now depend on the relative ranking of Express constraints versus Extension constraints.

3.3.1.1 The Constraints

Backness harmony is implemented here by reference to two separate domains: *PDom* ("Palatal Harmony Domain") and *VDom* ("Velar Harmony Domain"). These domains are in complementary distribution, they must align with segment boundaries, and they must include at least one vowel, if they exist. These characteristics are enforced by the following inviolable constraints:

SEGMENTAL(VDom):	VDom[→ V[or C[,]VDom →]C or]V
SEGMENTAL(PDom):	PDom[→ V[or C[,]PDom →]C or]V
HASSIZE(VDom):	VDom → V
HASSIZE(PDom):	PDom → V
*VDOMPDOM:	VDom ⊥ PDom

The existence of the domains is encouraged by the violable constraint ANCHOR-V/P-DOM ("V → VDom or PDom").

Basic Alignment Constraints The following constraints enforce Basic Alignment for vowel backness harmony:

BA-LEFT(VDom):	V[and Velar[→ VDom[
BA-RIGHT(VDom):	]V and]Velar →]VDom
BA-LEFT(PDom):	V[and Palatal[→ PDom[
BA-RIGHT(PDom):	]V and]Palatal →]PDom

Domain Dependency Constraints In order to place the domains in reasonable places, there are a number of DEP constraints specific to backness harmony domains. The constraints are as follows:

DepLeft(VDom):	VDom[→ Velar[
DepLeftV(VDom):	VDom[→ Velar[or]Opq
Dep(VDom):	VDom → Velar
DepLeft(PDom):	PDom[→ Palatal[
DepLeftV(PDom):	PDom[→ Palatal[or]Opq
Dep(PDom):	PDom → Palatal

Wide Scope Alignment Constraints The following constraints enforce Wide Scope Alignment for backness harmony:

WSA-right(VDom):	]VDom →]Word or PDom[or C[, VDom ⊥]Word
WSA-right(PDom):	]PDom →]Word or VDom[or C[, PDom ⊥]Word
WSA-left(VDom):	VDom[→ Word[, VDom ⊥]Word
WSA-left(PDom):	PDom[→ Word[, PDom ⊥ Word[
VDomWide:	VDom → V[
PDomWide:	PDom → V[
*VDomVDom:	VDom[⊥]VDom
*PDomPDom:	PDom[⊥]PDom
Extend-V/P-Dom:	]VDom ⊥ PDom[,]PDom ⊥ VDom[

Of these, the WSA constraints provide boundaries to domain spreading, the DomWide constraints encourage the domains to cover at least two vowels, the *DomDom constraints forbid adjacent domains of the same type, and Extend-V/P-Dom encourages consecutive domains of different types (thus, in combination with the *DomDom constraints, encouraging there to be as few domains as possible in a word).

Expression Constraints The constraints mandating expression of the harmony domains are fairly straight-forward. The only wrinkle is that they are stronger at the left edge of the domain. The constraints are as follows:

Express(VDom):	VDom and V → Velar
Express(PDom):	PDom and V → Palatal
VDomBeginsVel:	VDom[→ Velar[or Velar
PDomBeginsPal:	PDom[→ Palatal[or Palatal

Ranking The following constraints regulating vowel backness harmony are inviolable: Dep(VDom), Segmental(VDom), VDomBeginsVelar, HasSize(VDom), *VDomVDom, Dep(PDom), Segmental(PDom), PDomBeginsPalatal, HasSize(PDom), *PDomPDom, *VDomPDom. These out-rank the violable constraints, and the violable constraints are ranked in the following order: Anchor-V/P-Dom ≫ *VDomPDom ≫ Express(VDom), Express(PDom) ≫ WSA-right(VDom), WSA-right(PDom) ≫ VDomWide, PDomWide ≫ Extend-V/P-Dom ≫ DepLeft(VDom), DepLeft(PDom) ≫ BA-left(VDom), BA-left(PDom) ≫ *VDom, *PDom ≫ BA-right(VDom), BA-right(PDom) ≫ Dep(Vel), Dep(Pal), Dep(NP), Dep(Lo), Max(Velar), Max(Pal), Max(NP), Max(Lo) ≫ WSA-left(VDom), WSA-left(PDom).

3.3.1.2 Opacity

The analysis given above for vowel backness harmony, complicated as it is, is further complicated by certain opacity effects. Although the Turkish vowel system contains eight vowels, five of the eight vowels, that is, /a, e, i, o, u/, seem to be more preferred than the others, (/ö, ü, ɨ/). Not coincidentally, these five preferred vowels form the vowel inventory of the majority of the world's languages, and the others are much less common. There are two ways in which the five primary vowels are preferred over the others. First, the dispreferred vowels may only appear when they are in harmony with some other vowel, and second, the preferred vowels can be opaque to harmony. In roots, vowels that on the surface belong to the set /a, e, i, o, u/ may disagree in backness with the preceding

vowel, and are in this case opaque to spreading. Additionally, there are certain suffixes in Turkish that contain one or more vowels that exhibit opacity to vowel harmony. These vowels are invariably from the set /a, e, i, o, u/.

As noted above, vowels from the set /ö, ü, ɨ/ may only appear in harmony with other vowels, with the exception of the combinations /ü...i/ and /i...ü/, which will be dealt with in section 3.3.2. This restriction is enforced by means of the constraint:

*SoloMarked: V and Round and Palatal → RdDom or PDom,
V and Flat and Velar and High → FltDom or VDom.

This constraint is ranked between the *Dom constraints and the BA-right constraints. The other preference effect, opacity, may be dealt with by having inviolable Max constraints for the left edges of vowels in the set /a, e, i, o, u/ when they are in the root or when they have been lexically marked opaque. Note that this means that /ö, ü, ɨ/ vowels may be marked opaque, but it will have no effect. The constraints are as follows:

RtMaxLt(Vel, Rd):	Velar[and V[and Root and Rnd[→ Velar[,
	Velar[and V[and Root[and Rnd[→ Velar[,
	Velar[and V[and Root and Rnd[→ Rnd[,
	Velar[and V[and Root[and Rnd[→ Rnd[
RtMaxLt(Lo):	Low[and V[and Root → Low[,
	Low[and V[and Root[→ Low[
RtMaxLt(Flt, Pal):	Flat[and Pal[and V[and Root → Flat[,
	Flat[and Pal[and V[and Root[→ Flat[,
	Flat[and Pal[and V[and Root → Pal[,
	Flat[and Pal[and V[and Root[→ Pal[
OpqMaxLt(Vel, Rd):	Velar[and V[and Opq[and Rnd[→ Velar[,
	Velar[and V[and Opq[and Rnd[→ Rnd[
OpqMaxLt(Lo):	Low[and V[and Opq[→ Low[
OpqMaxLt(Flt, Pal):	Flat[and Pal[and V[and Opq[→ Flat[,
	Flat[and Pal[and V[and Opq[→ Pal[

3.3.2 Roundness Harmony

The mechanisms enforcing roundness harmony are similar to those that enforce backness harmony. The only difference is that [+round] and [-round] only spread onto high vowels, whereas backness harmony applies to all vowels. Because the analysis is so similar, only constraints that differ to a large extent from those given for backness harmony will be given here. For the rest, simply replacing "Velar" by "Round," "Palatal" by "Flat," "VDom" by "RdDom," and "PDom" by "FltDom" will give the equivalent constraints. There are some differences, however, in the Wide Scope Alignment, Anchor, Extend, and Express constraints. The differences are that Wide Scope Alignment is limited to disallow crossing over non-high vowels, that only high vowels need to be in roundness harmony domains, that roundness harmony domain intersections are only dispreferred before high vowels, and that only high vowels in roundness domains need express the domain vowel:

WSA-RIGHT(RdDom):	]RdDom →]Word or FltDom[or Low[or NP[, RdDom ⊥]Word, RdDom ⊥ Low[, RdDom ⊥ NP[
WSA-RIGHT(FltDom):	]FltDom →]Word or RdDom[or Low[or NP[, FltDom ⊥]Word, FltDom ⊥ Low[, FltDom ⊥ NP[.
ANCHOR-RD/FLT-DOM:	V and High → RdDom or FltDom
EXTEND-RD/FLT-DOM:	]RdDom ⊥ FltDom[⊥ High[,]FltDom ⊥ RdDom[⊥ High[
EXPRESS(RdDom):	RdDom and V and High → Round, RdDom and C → Round
EXPRESS(FltDom):	FltDom and V and High → Flat, FltDom and C → Flat

Additionally, roundness harmony employs two constraints that are not necessary in backness harmony due to special provisions for consonantal backness

harmony, as described in Section 3.4: SpreadRdDomOverC and SpreadFltDomOverC. These constraints encourage the roundness spreading domains to contain consonants at their right ends:

SpreadRdDomOverC:]RdDom ⊥ C[
SpreadFltDomOverC:]FltDom ⊥ C[

In addition to these constraint differences, there are some differences in ranking. In general the roundness harmony constraints are ranked together with the vowel backness harmony constraints, except that *RdDomRdDom and *FltDomFltDom are ranked lower than their equivalents—these constraints are ranked in the area between the right-edge Wide Scope Alignment constraints and the left-edge Basic Alignment constraints. The ranking within this area is a bit different as well. Between the WSA-right and BA-left constraints the ranking is RdDomWide, FltDomWide ≫ DepLeft(FltDom), DepLeft(RdDom) ≫ SpreadRdDomOverC, SpreadFltDomOverC, *RdDomRdDom, *FltDomFltDom ≫ Extend-Rd/Flt-Dom.

3.3.2.1 Opacity

Most of the opacity effects that affect roundness harmony were covered in section 3.3.1.2 above. However, in that section it was mentioned that the sequences /ü...i/ and /i...ü/ are allowed even if the "ü" is not in roundness harmony with any other vowel. In these instances, this analysis assumes that the "ü" is lexically marked opaque and is preserved due to the inviolable constraint OpaqueMaxUEI ("High and Round and Palatal and Opq and PDom → Round"), which preserves opaque "ü" when it is in a Palatal harmony domain.

3.4 Consonant Harmony

In addition to the well-known vowel harmony of Turkish, there is also a sort of "consonant harmony" wherein certain consonants assimilate the feature [±back] from one of the vowels in their neighborhood. This consonant backness harmony is only consistently detectable in the consonants /k/, /g/, and /l/. In some dialects the harmony effects are also visible in some coronals, but those dialects will not be analyzed here. The behavior of /k/ and /g/ versus that of /l/ is quite distinct, so they will be treated separately.

3.4.1 Consonant Backness Assimilation in Velar Obstruent Stops

The velar obstruent stops, in general, take the [±back] value of the nucleus of the syllable in which they are located. This behavior is enforced by adding the following constraints to those governing backness harmony: UniformSyl-V/P-Dom, which forces backness harmony domains to spread right until they reach syllable boundaries; ExpressC(VDom) and ExpressC(PDom), which force obstruents to take the value of the harmony domain in which they are located; and AssimilateC(VDom) and AssimilateC(PDom), which encourage obstruents not found in harmony domains to assimilate the backness value of the domain to their left or right. The definitions of these constraints are as follows:

UniformSyl-V/P-Dom:	]VDom $\perp \sigma$,]PDom $\perp \sigma$
ExpressC(VDom):	VDom and C and Obs $\rightarrow$ Velar
ExpressC(PDom):	PDom and C and Obs $\rightarrow$ Palatal
AssimilateC(VDom):	]C and]Obs and V[and VDom[$\rightarrow$]Velar, C[and Obs[and]V and]VDom $\rightarrow$ Velar[
AssimilateC(PDom):	]C and]Obs and V[and PDom[$\rightarrow$]Palatal, C[and Obs[and]V and]PDom $\rightarrow$ Palatal[

The ExpressC constraints are ranked together with the Express constraints for backness harmony, the UniformSyl-V/P-Dom constraint is ranked

immediately after that, and the AssimilateC constraints are ranked after the BA-right constraints for backness harmony.

3.4.1.1 Opacity

Not surprisingly, some velar obstruent stops can exhibit opacity to backness spreading. First, lexically selected velar obstruent stops in onsets can take a backness value different from that of the previous syllable, as in *infiɫaķi* ("explosion (acc.)"). In this case, the backness value of the opaque velar obstruent stop spreads to the right in the same manner as for vowels. Second, velar obstruent stops in syllable codas must take on the backness value of their syllable, but lexically selected velar obstruent stops in that position can initiate spreading of their underlying, yet unrealized, backness value to the right, as in *idrakɫer* ("perception (nom. pl.)"). Finally, all velar obstruent stops that begin underlying word-initial clusters surface as [+back], and spread this value to the right. These clusters are pronounced in careful speech, but in colloquial speech they are broken up by epenthetic vowels. An example is *gɨrip* ("grippe"). The first opacity effect is analyzed by reference to inviolable Max(Vel/Pal) constraints for opaque velar obstruent stops in onsets:

OpaqueMaxC(Vel):	Velar and Opq and C and Obs and High and Non-strident and $\mu_s \rightarrow$ Velar
OpaqueMaxC(Pal):	Palatal and Opq and C and Obs and High and Non-strident and $\mu_s \rightarrow$ Palatal

The second effect is accounted for by inviolable constraints that prevent VDom and PDom from spreading beyond an opaque high non-strident consonant (these include the velar obstruent stops and /l/) that is underlyingly of a differing backness value:

OpaqueCBarrier(VDom):	VDom ⊥]C and]Opq and]Palatal and]High and]Non-strident
OpaqueCBarrier(PDom):	PDom ⊥]C and]Opq and]Velar and]High and]Non-strident

The third effect is a bit more difficult to get. The characteristics of Primitive Optimality Theory make it difficult to identify underlying initial consonant clusters, as a direct identification of these clusters would require simultaneous reference within a single constraint to the left and right boundaries of a consonant, but Primitive Optimality Theory can only refer to a single time-slice at a time, so it is necessary to project a special domain that includes the initial consonant or consonants of a word. This domain is projected by the following constraints, ranked in the order given, and crucially below the constraints on syllabification:

Project1stCs:	C[and Word[→ 1stCs[
Bound1stCs:	]1stCs → V[, 1stCs ⊥ V[
*1stCs:	1stCs ⊥ 1stCs

Given the existence of the "1stCs" domain, the [Velar] value of /k/ or /g/ in underlying word-initial consonant clusters is enforced by the constraint WdInitKGVelar[6]:

1stCs and]C and C[and]Obs and]High and]Non-strident → Velar,
1stCs and]C and]Obs and]High and]Non-strident ⊥ Velar[

All other aspects of the opacity effects described above happen automatically as a result of the already-existing constraints on the backness spreading domains.

3.4.2 Consonant Backness Assimilation in Laterals

The pattern of assimilations involving /l/ differs substantially from the pattern described above for /k/ and /g/. Initial /l/ is always palatal, and if there is a

[6]This constraint relies on the ins representation presented in Section 2.1.2.

front vowel in either the syllable preceding or the syllable following an /l/, then the /l/ will be palatal as well. However, this palatal assimilation does not cross morpheme boundaries. Everywhere else /l/ is unpredictably palatal or velar, although most often velar. Whenever the backness value of /l/ is not predictable from the context, and it appears on the surface with a backness value of [Palatal], it is opaque to backness spreading, and its backness value of [Palatal] spreads to the right.

As stated above, word-initial laterals are invariably palatal. This condition is accounted for by the straightforward inviolable constraint FIRSTLATPAL:

Word[and Ant[and High[and Sonor[and C[→ Palatal[

The remainder of the pattern requires a new alignment domain, *PLDom* ("Palatal Lateral Assimilation Domain"), due to the morpheme-internal limitations of lateral backness assimilation, and to the fact that it is only [Palatal] that spreads. This domain coincides with the syllable, except that it may not cross morpheme boundaries. All syllables with front vowels project PLDom domains. These characteristics of the lateral palatal harmony domain are enforced by the following constraints, ranked in the order given:

PROJECTPLDOM:	V and Palatal → PLDom
SEGMENTAL(PLDom):	PLDom[→ V[or C[,]PLDom →]C or]V
HASSIZE(PLDom):	PLDom → σ
MORPHINTERNAL(PLDom):	PLDom ⊥ Root[, PLDom ⊥ Suffix[, PLDom ⊥]Root, PLDom ⊥]Suffix
WSA-LEFT(PLDom):	PLDom[→ σ[, PLDom ⊥ σ[
WSA-RIGHT(PLDom):	]PLDom →]σ, PLDom ⊥]σ
WSA-SETTLE-LEFT(PLDom):	]PLDom →]Root or]Suffix
WSA-SETTLE-RIGHT(PLDom):	PLDom[→ Root[or Suffix[
*PLDOM:	PLDom ⊥ PLDom

In the constraints just given, the WSA-SETTLE constraints are an attempt

to duplicate the behavior of gradient wide scope constraints in this situation. If a domain is required to extend as far as possible toward a syllable boundary, but cannot pass morpheme boundaries, and has no other limitations, then it will either end at a syllable boundary or at a morpheme boundary.

Ranked below the constraints establishing the PLDom domain are three constraints that regulate the backness value of laterals. First is the requirement (LatAssimPal) that laterals in a lateral palatal assimilation domain or immediately to the right of one must have the value [Palatal]. Second is a heightened Max constraint for opaque laterals—LatMaxOpaque. The relative ranking here has the effect that underlyingly velar laterals that are marked opaque will still surface as palatal when in the neighborhood of a front vowel. Finally, the constraint LatDefault assigns a default value of [Velar]. The constraint definitions, in their ranking order, are:

LatAssimPal:	]PLDom and C[and Son[and High[→ Palatal[,
	PLDom and C and Son and High → Palatal
LatMaxOpq:	C and Son and High and Opq and Velar → Velar,
	C and Son and High and Opq and Palatal → Palatal

Because the patterns of backness assimilation for laterals differ from the patterns of the other backness harmony domains, the two systems must be insulated from one another to some extent. In order to accomplish this, the constraint *LatPVDomLeft is hypothesized. The effect of this constraint is to prevent a palatal or velar harmony domain from beginning with a lateral. The constraint is ranked just after the WSA-right constraints for vowel harmony, and its definition is as follows:

C[and Son[and High[and Ant[⊥ PDom[,
C[and Son[and High[and Ant[⊥ VDom[

Aside from the constraint above, no further constraints are necessary to model

the opacity effects of laterals. Note that, with the exception of any constraints described as inviolable, the constraints described in this section are ranked below the constraints for vowel backness and roundness harmony and velar obstruent backness assimilation, but above the DEP and MAX constraints for backness, roundness, and vowel height of non-high vowels.

3.5 Conclusion

The constraints and rankings presented above account for all of the phenomena discussed in Clements and Sezer (1982) except for some dialect-specific assimilation effects of coronal consonants and some optional and dialect-specific effects of roundness opacity. These effects should fit easily into the system described above, but space limitations do not permit an analysis here. Constraint listings, rankings, and data files for the analysis given above are found in Appendix B.

The analysis presented above shows conclusively that Primitive Optimality Theory as implemented here can be used to analyze and model at least some complex phonological systems. However, it does point to some limitations of the model. First, the finite-state limitations of the model make it difficult at times to analyze phenomena that seem to require gradient constraints. The analysis above would be much simpler except for the necessity to multiply gradient constraints into redundant binary constraints. Second, the computational characteristics of the model often affect the type of analyses that can be employed. For example, it might have been preferable to intermix the constraints regulating backness assimilation in laterals with those that regulate backness harmony in other segments. Such an intermingling might have led to more natural constraints, and might have necessitated the creation of fewer constraints overall. However, this would have been impossible because it would have created incompletely constrained PDom,

VDom, and PLDom tiers at the same time, which would have led to an explosive increase in the size of the finite state transducers generated. It is possible that the use of more advanced finite state techniques, such as factored automata (Eisner 1997a), might alleviate this type of limitation.

CHAPTER 4

Future Directions

Primitive Optimality Theory suffices to describe some very complex phonological phenomena, as the previous section has shown. However, there are still phenomena that cannot be accounted for using OTP. Additionally, there are countless ways in which the implementation presented here could be improved to be a better research tool and to be usable by more people. Some of them will be counted here. Finally, the formal simplicity of OTP suggests some projects related to the OTP model but not involving modifications to the implementation itself.

4.1 Phenomena Beyond the Scope of OTP

In general, OTP seems sufficient for much of pure phonology. However, where phonology interfaces with morphology and phonetics there is an additional level of complexity that seems beyond the formal power of OTP.

4.1.1 Morphology

Three areas for which OTP is not adequate, in increasing order of formal difficulty, are non-concatenative morphology, paradigm uniformity effects, and reduplication.

4.1.1.1 Non-Concatenative Morphology

In Primitive Optimality Theory, at least as specified in Eisner's (1997a, 1997b) current papers, it is possible to delete underlying material and to insert new surface material, but it is not possible to change the relative order of underlying constituents. This means that OTP cannot model floating tones, infixation, or templatic morphology. The solution proposed here, in broad outline, seems to have originated independently both here and with Eisner (personal communication). At present, input representations are constructed such that it is possible to insert surface material between underlying constituents, but doing so adds a constituent on the INS tier. If the constraints of OTP were divided into two stages, an underlying stage and a surface stage, then the input representation used by the underlying stage could be modified to permit the insertion of underlying material between underlying constituents. Morphemes then would be introduced as separate finite state machines and intersected together. The result of this intersection would be a finite state machine allowing any possible interleaving of the morphemes. Constraints could then control the order seen on the surface. Concatenative morphology, for example, would make use of highly-ranked constraints that require alignment of morpheme edges. At the end of the underlying stage, a special constraint would impose the current limitation that only surface constituents may be inserted. Note that because morphemes in this system are simply finite state machines, the system leaves open the possibility of abstract morphemes. In addition to standard morpheme FSMs that require the input to contain specified constituents, the system would, for example, allow the specification of morphemes of subtractive morphology: morphemes that require some part of the input to be silent in the output.

4.1.1.2 Paradigm Uniformity

OTP supports correspondence between surface constituents in the same output form, and between surface and underlying constituents. In addition to these correspondences, and perhaps instead of input-output correspondences, certain linguistic phenomena seem to require correspondence between output forms and other, related output forms, members of the same paradigm (Steriade 1996, Flemming 1995, Kenstowicz 1995). For example, the third person singular present indicative active form of a Latin verb can be said to obey constraints that require it to be similar to other forms of the same verb. Adding support to Primitive Optimality Theory for these types of correspondences should not be terribly difficult from a formal standpoint, but it remains to be seen whether it is possible computationally. The basic model would work as follows. The system must maintain a database of output forms, stored as lists of time slices (*labels*—see Section 2.4.1, paragraph 2), together with information that can be used to associate them to paradigms. These output forms could represent previous outputs of the system, or representations of utterances that have been heard. When a new output form is to be generated, the system will *activate* the members of the appropriate paradigm[1] by inserting them into special tiers in the input form. Because the proper alignment between paradigm elements and the output form is not specified, this insertion will need to make use of the techniques outlined in the previous section. Just as the underlying tiers are marked by a single underline, the paradigm tiers will be marked by a distinct number (greater than two) of underlines for each member of the paradigm. Paradigm Uniformity constraints will then have to be constructed such that they output a different number of

[1]Note that this imposes a limitation on the Paradigm Uniformity effects that can be modeled—there must be no more than a single paradigm for each output to be produced.

violations depending on the number of paradigm elements for which the specified correspondence with the output does not hold true. There are two ways in which this violation-counting could be accomplished. First, the constraint FSTs themselves could be modified to compute the correct number of violations, as follows. The paradigm uniformity constraints would be expressed in a formal notation similar to that proposed here for other types of constraints, restricted to constraints enforcing alignment between constituents rather than forbidding it (that is, implication rather than clash constraints). The finite state transducers for non-paradigm-uniformity constraints contain weighted edges[2] which encode particular disharmonic time slices. These edges would have to be multiplied such that there was one edge for each possible combination of matches and non-matches among the different paradigm elements. For example, if some edge should be aligned between all paradigm elements, one edge, having an output of 2 violations, might correspond to the situation where the candidate aligns with all paradigm members except the first and third, and so forth. This modification would result in extremely large transducers. Alternatively, the procedure of finite state machine intersection could be modified such that when one edge is intersected with another, the resulting edge weight (the number of violations the resulting FST will output when a label matches that edge), rather than, as now, being set to the sum of the weights of the edges, is set in accordance with the number of offending paradigm elements involved. Such a method might be complex, but would probably be more efficient overall.

OTP, together with these changes, should theoretically be able to model the effects of paradigm uniformity. However, because of the number of tiers involved and the immensity of the finite state machines that will result when adding

[2]That is, edges that output a number of violations other than zero

paradigm elements to the input representation, this solution may prove to be impractical. Another possible solution might be to compute in parallel the output candidates that best correspond with both the input and each individual paradigm element. The result of this process would be a single output candidate for each paradigm element. Some method would then have to be devised for either combining these candidates into a single best candidate or selecting the best candidate from among them. A third solution might be to implement base identity (Kenstowicz 1995), that is, to implement constraints that refer only to the base element of a paradigm, rather than implementing the full paradigm-wide effects of paradigm uniformity/uniform exponence.

4.1.1.3 Reduplication

Reduplicative phenomena have four possible outcomes[3]. The result of reduplication may resemble reduplication of the underlying form followed by phonology, as in Dakota; the result may resemble phonology done to the base followed by reduplication, as in Madurese; the result may resemble reduplication followed by phonology applied to the reduplicant, which is then copied to the base, as in Southern Paiute; and the result may preserve underlying distinctions that have been lost in the base, as in Klamath. These outcomes suggest three ways in which reduplication might be implemented in OTP. These three methods could be implemented simultaneously and selected by the user when specifying that a particular morpheme is reduplicative. First, the morpheme specified as the base of reduplication could be inserted into the underlying representation twice, using the method of Section 4.1.1.1, except that the reduplicant would appear in tiers

[3]See (McCarthy and Prince 1995).

marked with two underlines[4]. Phonology would then apply, and could lead to any of the outcomes listed above except for the second or third. Second, the entire process of derivation could be completed without including a reduplicative morpheme, and then the result could be copied. This method would allow the second outcome, but only for cases of total reduplication. Third, reduplication could be interleaved with the process of derivation. This method would initiate reduplication in the same manner as the first method. After each finite state transducer intersection[5], the reduplicant and the base will be divided into two separate FSTs. The doubly-underlined tiers, representing the base, would then be removed from the FST representing the reduplicant, and the resulting FST would be minimized using Brzozowski's algorithm (Brzozowski 1962). The surface tiers in the FST representing the base would then be copied into a separate FST wherein the surface tiers from the base would be entered in the doubly-underlined tiers. This FST would then be intersected with the reduplicant FST, and the reduplicant and base FSTs would be concatenated back into a single FST. To use this method, the user would have to specify a level within the constraint ranking at which the following conditions applied: (1) the reduplicant and the base are not interleaved with each other, (2) the reduplicant and the base are adjacent, and (3) the utterance contains no material outside the reduplicant and the base. Interleaving would occur after FSM intersection reaches the specified level. This third method may be able to duplicate all four of the outcomes, although some experimentation will be necessary to ensure this, and it might be that the method may be too inefficient to be practical.

[4]The kernel of this idea appears in Eisner (1997b).

[5]See Section 2.4 for the overall algorithm used in the current implementation.

4.1.2 Phonetics

Primitive Optimality Theory, like most of Optimality Theory itself, is discrete. It can have only a finite number of constraints and a finite number of tiers, and to the extent that it can manipulate numerical quantities at all, OTP deals exclusively with integers. Phonetics, however, can really only be approached in terms of real numbers. Although this problem is much too large to be fully addressed here, there are some possibilities worthy of mention. It seems likely that Optimality Theory itself, as a discrete system of finite, rigidly ranked constraints, is fundamentally unsuited to the problems of phonetics and the phonology-phonetics interface. Formally, Optimality Theory is an example of an Over-Constrained System where the problems associated with being over-constrained are solved by hierarchical ranking of constraints, *i.e.,* it is a Hierarchical Constraint System (Jampel 1996a, Jampel 1996b). There is another standard approach to over-constrained systems—Partial Constraint Satisfaction. This method uses weighted constraints and global cost functions to best satisfy the constraints, and works well with real-valued problems. A tentative suggestion is to use a two-stage approach where Optimality Theory or Primitive Optimality theory is employed for phonology, morphology, and perhaps syntax and semantics, and then the final result is passed to a system of Partial Constraint Satisfaction, where the phonetic details are elaborated. It is also possible that the two systems might be interleaved or applied in a cyclic fashion.

4.2 Possible Extensions to the Implementation

Currently, users of the implementation presented here must work within the representation and constraint system of OTP. Linguists who are used to other repre-

sentations may find this prospect unattractive. Therefore it may be useful either to implement alternative systems using different representations, or to add a layer of abstraction between users and the functionality of OTP. For example, it might be useful to provide a library of constraint names from the Optimality Theoretic literature, together with their equivalents in OTP. Users could then specify constraints to the system by referring to the more standard names. Additionally, a translator could be developed for translating constraint names from constraint families such as MAX and ALIGN into the equivalent OTP constraints. More ambitiously, it might be possible to translate the types of constraints suggested by Feature Class theory (Padgett 1995) into multiple OTP constraints by expanding constraints referring to feature classes into separate OTP constraints referring to individual features. This facility might additionally allow the user to specify feature assimilation in a simple way and generate the appropriate spreading domains and related constraints automatically and invisibly.

4.3 Learnability

As mentioned in the introduction[6], some linguists hypothesize that at least a subset of the constraints used in an Optimality Theoretic grammar might be learned from the input data rather than being innate. Primitive Optimality Theory provides a sufficiently restricted formal model of constraints that it should be possible to model a learner of OTP constraints from data. Earlier work on learning finite state machines to describe input data, such as Deng (1996), Freund *et al.* (1993), and Kay (1977), should be useful in this endeavor.

[6]Section 1.1.

4.4 Conclusion

Primitive Optimality Theory has shown itself to be a useful and powerful tool for phonological analysis, but it is also clear that it does not allow for analysis of the entire range of phenomena phonologists study and is therefore inadequate as a model for how human beings actually process phonological data. However, it is possible that, using OTP as a foundation, we might build a model that does account for the phonological data. Such a model may be quite useful both as a tool for analysis and as a starting point for understanding the human language device.

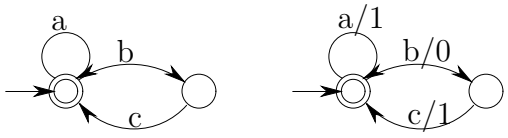
APPENDIX A

Finite State Automata and Regular Expressions

A *regular expression* is any expression built up from a regular algebra (also known as a sigma algebra). Regular expressions describe sets of strings. The basic symbols of a regular algebra are called an alphabet, and are arbitrary. In our case, they are time-slices from a gestural score, such as $\langle$+F, $]\sigma$, $[\mu\rangle$, which signifies the interior of a foot overlapping the right edge of a syllable and the left edge of a mora, but they could just be letters, numbers, etc. In our examples here we will use elements from the set $\{a, b, c\}$. The operators of a regular algebra are: concatenation—two or more expressions in a row—*e.g.,* ab signifies "a followed by b"; disjunction—one expression or the other—*e.g.,* $a|b$ signifies "a or b"; and Kleene star—zero or more of an expression—*e.g.,* a^* signifies "zero or more of a." Balanced parentheses may be used wherever appropriate. An example is "$(a|bc)^*$," which specifies the set of strings {a,aa,aaa,...,bc,$bcbc$,...,abc,bc,$bcabc$,$aabca$, ...} ("zero or more of a or bc").

A *finite state machine* is a device that corresponds to a regular expression. When given a string as an input, a finite state machine will *accept* the string if it matches the corresponding regular expression or *reject* it if it does not. The basic idea is that the machine starts out in a state, which is represented by a circle: ◯. The machine then reads in a single symbol from the input string. If there is no edge leading out from the current state that is labeled with the symbol, the input string is rejected. Otherwise, the machine changes its current state to the

state at the other side of the appropriately labeled edge. When all of the symbols have been processed, it is necessary to examine the current state. If the current state is an "accepting state"—depicted by a double circle ◎—then the string is accepted. Otherwise, it is rejected. The regular expression "$(a|bc)^*$" would thus be represented by the finite state machine in figure A.1a, where the initial state is denoted by a sourceless arrow.



(a) FSM (b) FST

Figure A.1: An FSM and an FST for $(a|bc)^*$

A *finite state transducer* is like a finite state machine, except that the edges have additional information—output symbols. Whenever an edge is traversed, the machine outputs the output symbol on that edge. For example, the FST in figure A.1b accepts the same strings as the FSM, but it also outputs a string of ones and zeros for every string that it receives. For example, the input "*abcaa*" would produce the output "10111". Thus, a finite state transducer accepts or rejects input strings in the manner of a finite state machine, but it also maps strings to other strings.

In mathematical terms, a deterministic finite state machine (an FSM of the sort where given any state and an input symbol there is at most one state to which a transition can be made) can be described as a five-tuple $M = \langle Q, \Sigma, \delta, s, F \rangle$, where Q is the set of states used in the machine, Σ is the alphabet of edge labels, $\delta : Q \times \Sigma \to Q$ is the transition function that takes as its input a state and a

member of Σ and returns the state to which a transition will be made from that source state given the input symbol from Σ, $s \in Q$ is the start state, and $F \subseteq Q$ is the set of accepting states. For the finite state machines described in this thesis, the elements of Σ consist of time-slice descriptors which describe a set of possible time slices. Each element $e \in \Sigma$ consists of a sequence of sets of symbols, where each set corresponds to a tier, and the symbols are the elements of the gestural score which may appear on that tier during the time-slice. For example, in a system with 5 tiers, numbered 1–5, a time slice constrained to allow only left edges in tier 3 but to allow any other symbol in the other tiers would appear as follows: $\langle$\{-,+,[,],|\}, \{-,+,[,],|\}, \{[\}, \{-,+,[,],|\}, \{-,+,[,],|\}$\rangle$, or, more compactly, $\langle$-+[]|,-+[]|,[,-+[]|,-+[]|$\rangle$. Generally the sets are constrained not to be empty. An extended version of δ, $\delta^* : Q \times \Sigma^* \rightarrow Q$ can be defined on the basis of δ as follows. Let w be a sequence $\langle e_1, \ldots, e_n \rangle$ such that for all i from 1 to n, $e_i \in \Sigma$. Then $\delta^*(q_1, w) = \delta(\delta(\ldots \delta(q_1 e_1), \ldots e_{n-1}), e_n)$, if such a statement is definable. That is, $\delta^*(q_1, w)$ returns the state, if any, that would be reached by following the appropriate transitions for the elements of w in order starting from state q_1. A finite state machine M can be viewed as representing a (possibly infinite) set of strings $L(M)$ where $L(M) = \{w | \delta^*(s, w) \in F\}$.

A deterministic finite state transducer can be described as a seven-tuple $T = \langle Q, \Sigma, \Omega, \delta, \omega, s, F \rangle$, where Q, Σ, δ, s, and F are as before; Ω is the set of output symbols produced my the machine; and $\omega : Q \times \Sigma \times Q \rightarrow \Omega$ is the transition function—it takes a source state, an input symbol, and a destination state (reachable from the source via the input symbol) and produces as its output the symbol that will be produced when going through that transition. A deterministic finite state transducer defines a function from strings to strings, the domain of which is $L(M)$ for $M = \langle Q, \Sigma, \delta, s, F \rangle$.

Given these definitions of deterministic finite state machines and deterministic finite state transducers, the intersection $M_1 \cap T_2$ of a deterministic finite state machine $M_1 = \langle Q_1, \Sigma, \delta_1, s_1, F_1 \rangle$ with a deterministic finite state transducer $T_2 = \langle Q_2, \Sigma, \Omega, \delta_2, \omega_2, s_2, F_2 \rangle$ can be defined as $M_1 \cap T_2 = \langle Q_1 \times Q_2, \Sigma, \Omega, \delta', \omega', \langle s_1, s_2 \rangle, F_1 \times F_2 \rangle$, where $\delta'(\langle q_1, q_2 \rangle, e_1 \cap e_2) = \langle \delta_1(q_1, e_1), \delta_2(q_2, e_2) \rangle$ and $\omega'(\langle q_{1,1},\ q_{1,2} \rangle,\ e_1 \cap e_2,\ \langle q_{2,1}, q_{2,2} \rangle) = \omega_2(q_{1,2},\ e_2,\ q_{2,2})$. These two functions are only defined when $e_1 \cap e_2$ is a member of Σ. Recall that Σ is composed of sequences of sets of symbols, where all of the sets are constrained to be non-empty. For any two elements of Σ, $e_1 = \langle T_{1,1}, \ldots, T_{1,n} \rangle$ and $e_2 = \langle T_{2,1}, \ldots T_{2,n} \rangle$, where n is the number of tiers, $e_1 \cap e_2 = \langle T_{1,1} \cap T_{2,1}, \ldots, T_{1,n} \cap T_{2,n} \rangle$. This sequence is a member of Σ if and only if no set within it is the empty set. The output of the intersection $M_1 \cap T_2$ is a deterministic finite transducer whose domain is the intersection of $L(M_1)$ with the domain of T_2 and which maps each element of its domain to the output of T_2 for that element. In the OTP system, each constraint transducer T is defined to have a domain of Σ^*, that is, every possible combination of time slices, and to output strings from the alphabet $\{0, 1, 2\}$. The number of violations assessed by the constraint on any particular candidate is the sum of the symbols output by the transducer when given that candidate as an input. Because the domain of constraint transducers in OTP is Σ^*, the domain of the intersection of the finite state machine M representing a set of optimal candidates of some stage of the derivation and T is just the domain of M. The application of the best paths minimization algorithm on the product of this intersection produces a new finite state machine M' which accepts the subset of $L(M)$ to which T assesses the least number of violations.

APPENDIX B

Data Files Relevant to the Turkish Analysis

The files listed here are complete as used, except that the output transcriptions produced for each input have been added to the input files. Additionally, the segment specifications have been significantly abbreviated. See Table 3.1 for the features of Turkish.

B.1 Segment Specifications, a Selection

```
SubUtterance ibar:
    Segment [ +_V, -_Rd, +_Flt, -_Pal, +_Vel, +_Hi, -_Lo, -_NP ].

SubUtterance i-opq:
    Segment [ +_V, -_Rd, +_Flt, +_Pal, -_Vel, +_Hi, -_Lo, -_NP,
              +_Opq ].

SubUtterance k:
    Segment [ +_C, +_Hi, -_Ns, -_Son, +_Obs, -_Lab, -_Cor, -_Ant,
              -_Vcd, -_Cnt, +_NSt].

SubUtterance ^k-opq:
    Segment [ +_C, +_Hi, -_Ns, -_Son, +_Pal, -_Vel, +_Obs, -_Lab,
              -_Cor, -_Ant, -_Vcd, -_Cnt, +_NSt, +_Opq].

SubUtterance k-opq:
    Segment [ +_C, +_Hi, -_Ns, -_Son, -_Pal, +_Vel, +_Obs, -_Lab,
              -_Cor, -_Ant, -_Vcd, -_Cnt, +_NSt, +_Opq].
```

B.2 Constraints

B.2.1 Constraints File

```
Tiers:
        Wd ("word"), Rt ("Root"), _Rt, Sfx ("Suffix"), _Sfx,
        VDom ("Velar Spreading Domain"),
        PDom ("Palatal Spreading Domain"),
        RdDom ("Round Spreading Domain"),
        FltDom ("Flat Spreading Domain"),
        PLDom ("Lateral Palatal Assignment Domain"),
        1stCs ("First Consonants in the Word"),
        s ("syllable"), ms ("strong mora"), mw ("weak mora"),
        C ("consonant"), _C, V ("vowel"), _V,
        Rd ("round"), _Rd, Flt ("flat"), _Flt,
        Pal ("palatal"), _Pal, Vel ("velar"), _Vel,
        Hi ("high"), _Hi, Lo ("low"), _Lo,
        NP ("non-peripheral"), _NP, Ns ("nasal"), _Ns,
        Son ("sonorant"), _Son, Obs ("obstruent"), _Obs,
        Lab ("labial"), _Lab, Cor ("coronal"), _Cor,
        Ant ("anterior"), _Ant, Vcd ("voiced"), _Vcd,
        Cnt ("continuant"), _Cnt, NSt ("non-strident"), _NSt,
        _Opq ("opaque").

%%%%%%%%%%%%%%%%%%%%%%
% I-O Correspondence %
%%%%%%%%%%%%%%%%%%%%%%

Constraint Dep(V): V[ --> _V[, ]V --> ]_V.
Constraint Max(V): _V[ --> V[, ]_V --> ]V.

Constraint Max(C): _C[ --> C[, ]_C --> ]C.
Constraint Dep(C): C[ --> _C[, ]C --> ]_C.

Constraint Max(Rd): _Rd[ --> Rd[, ]_Rd --> ]Rd.
Constraint Dep(Rd): Rd[ --> _Rd[, ]Rd --> ]_Rd.

Constraint Max(Flt): _Flt[ --> Flt[, ]_Flt --> ]Flt.
Constraint Dep(Flt): Flt[ --> _Flt[, ]Flt --> ]_Flt.
```

```
Constraint Max(Vel): _Vel[ --> Vel[, ]_Vel --> ]Vel.
Constraint Dep(Vel): Vel[ --> _Vel[, ]Vel --> ]_Vel.

% These constraints reflect
% the fact that Max is higher-ranked for unmarked vowels in
% roots than for marked ones.
Constraint RootMaxLeft(Vel, Rd):
        _Vel[ and V[ and _Rt and _Rd[ --> Vel[,
        _Vel[ and V[ and _Rt[ and _Rd[ --> Vel[,
        _Vel[ and V[ and _Rt and _Rd[ --> Rd[,
        _Vel[ and V[ and _Rt[ and _Rd[ --> Rd[.

Constraint RootMaxLeft(Lo):
        _Lo[ and V[ and _Rt --> Lo[,
        _Lo[ and V[ and _Rt[ --> Lo[.

Constraint RootMaxLeft(Flt, Pal):
        _Flt[ and _Pal[ and V[ and _Rt --> Flt[,
        _Flt[ and _Pal[ and V[ and _Rt[ --> Flt[,
        _Flt[ and _Pal[ and V[ and _Rt --> Pal[,
        _Flt[ and _Pal[ and V[ and _Rt[ --> Pal[.

Constraint OpaqueMaxLeft(Vel, Rd):
        _Vel[ and V[ and _Opq[ and _Rd[ --> Vel[,
        _Vel[ and V[ and _Opq[ and _Rd[ --> Rd[.

Constraint OpaqueMaxLeft(Lo):
        _Lo[ and V[ and _Opq[ --> Lo[.

Constraint OpaqueMaxLeft(Flt, Pal):
        _Flt[ and _Pal[ and V[ and _Opq[ --> Flt[,
        _Flt[ and _Pal[ and V[ and _Opq[ --> Pal[.

Constraint OpaqueMaxC(Vel):
        _Vel and _Opq and _C and _Obs and _Hi and _NSt
             and ms --> Vel.

Constraint OpaqueMaxC(Pal):
        _Pal and _Opq and _C and _Obs and _Hi and _NSt
             and ms --> Pal.
```

```
Constraint OpaqueMaxUEI:
        _Hi and _Rd and _Pal and _Opq and PDom --> Rd.

Constraint Max(Pal): _Pal[ --> Pal[, ]_Pal --> ]Pal.
Constraint Dep(Pal): Pal[ --> _Pal[, ]Pal --> ]_Pal.

Constraint Max(Hi): _Hi[ --> Hi[, ]_Hi --> ]Hi.
Constraint Dep(Hi): Hi[ --> _Hi[, ]Hi --> ]_Hi.

Constraint DepC(Hi):
        Hi[ and C[ --> _Hi[, ]Hi and ]C --> ]_Hi.

Constraint Max(Lo): _Lo[ --> Lo[, ]_Lo --> ]Lo.
Constraint Dep(Lo): Lo[ --> _Lo[, ]Lo --> ]_Lo.

Constraint Max(NP): _NP[ --> NP[, ]_NP --> ]NP.
Constraint Dep(NP): NP[ --> _NP[, ]NP --> ]_NP.

Constraint Max(Ns): _Ns[ --> Ns[, ]_Ns --> ]Ns.
Constraint Dep(Ns): Ns[ --> _Ns[, ]Ns --> ]_Ns.

Constraint Max(Son): _Son[ --> Son[, ]_Son --> ]Son.
Constraint Dep(Son): Son[ --> _Son[, ]Son --> ]_Son.

Constraint Max(Obs): _Obs[ --> Obs[, ]_Obs --> ]Obs.
Constraint Dep(Obs): Obs[ --> _Obs[, ]Obs --> ]_Obs.

Constraint Max(Lab): _Lab[ --> Lab[, ]_Lab --> ]Lab.

Constraint Dep(Lab): Lab[ --> _Lab[, ]Lab --> ]_Lab.

% Coronal
Constraint Max(Cor): _Cor[ --> Cor[, ]_Cor --> ]Cor.
Constraint Dep(Cor): Cor[ --> _Cor[, ]Cor --> ]_Cor.

% Anterior
Constraint Max(Ant): _Ant[ --> Ant[, ]_Ant --> ]Ant.
Constraint Dep(Ant): Ant[ --> _Ant[, ]Ant --> ]_Ant.

% Voiced
```

```
Constraint Max(Vcd): _Vcd[ --> Vcd[, ]_Vcd --> ]Vcd.
Constraint Dep(Vcd): Vcd[ --> _Vcd[, ]Vcd --> ]_Vcd.

% Continuant
Constraint Max(Cnt): _Cnt[ --> Cnt[, ]_Cnt --> ]Cnt.
Constraint Dep(Cnt): Cnt[ --> _Cnt[, ]Cnt --> ]_Cnt.

% Non-strident
Constraint Max(NSt): _NSt[ --> NSt[, ]_NSt --> ]NSt.
Constraint Dep(NSt): NSt[ --> _NSt[, ]NSt --> ]_NSt.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Surface Relationships Between Features %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Constraint Segmental(Vel): Vel[ --> V[ or C[, ]Vel --> ]V or ]C.
Constraint Segmental(Pal): Pal[ --> V[ or C[, ]Pal --> ]V or ]C.
Constraint Segmental(Rd): Rd[ --> V[ or C[, ]Rd --> ]V or ]C.
Constraint Segmental(Flt): Flt[ --> V[ or C[, ]Flt --> ]V or ]C.
Constraint Segmental(Lo): Lo[ --> V[, ]Lo --> ]V.
Constraint Segmental(NP): NP[ --> V[, ]NP --> ]V.

Constraint VelPalMandatory:
        _C and _Hi and _NSt --> Vel or Pal, _V --> Vel or Pal.

Constraint RdFltMandatory: _V --> Rd or Flt, V --> Rd or Flt.

Constraint *RdFlt: Rd _|_ Flt.
Constraint *PalVel: Pal _|_ Vel.
Constraint *HiLo: Hi _|_ Lo.
Constraint *HiNP: Hi _|_ NP.
Constraint *LoNP: Lo _|_ NP.
Constraint *SonObs: Son _|_ Obs.

Constraint AssignHeight: V --> Hi or Lo or NP.

Constraint BackVowelInv:
        Flt and Vel and V --> Hi or Lo,
        Rd and Vel and V --> Hi or NP.

Constraint LoDefaults: Lo --> Flt, Lo _|_ Pal, Lo --> Vel.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Syllabification and Prosody %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Constraint AlignSyl: s[ --> C[ or V[, ]s --> ]C or ]V.
Constraint ParseC: C --> s.
Constraint ParseV: V --> s.
Constraint HNuc: s --> V.
Constraint PackCs: Wd and ]C --> C[ or V[.
Constraint PackVs: Wd and ]V --> C[ or V[.
Constraint Ons: s[ --> C[.
Constraint *Coda: mw _|_ mw.
Constraint *CV: C _|_ V.
Constraint ProjectWd: _Rt --> Wd, _Sfx --> Wd, Wd --> _Rt.

% No syllables strictly outside Wd.
Constraint AllInWord: s --> Wd.
Constraint Align(Wd, L, s, L): Wd[ --> s[.
Constraint Align(Wd, R, s, R): ]Wd --> ]s.

% No gaps between syllables.
Constraint PackSyls: Wd and ]s --> s[.

Constraint *Wd: Wd _|_ Wd.

% Moras:
Constraint PackMoras:
   Wd and ]ms --> ms[ or mw[, Wd and ]mw --> ms[ or mw[.

Constraint Align(s, L, ms, L): s[ --> ms[.
Constraint Align(s, R, m, R): ]s --> ]ms or ]mw.
Constraint Align(ms, L, s, L): ms[ --> s[.
Constraint Parse(ms): ms --> s.
Constraint Parse(mw): mw --> s.
Constraint *RepeatMw: ]mw _|_ mw[.
Constraint CodaMw:
   ]V --> V[ or ]s or mw[, ]V and V[ and s --> mw.
Constraint *MsMw: ms _|_ mw.

Constraint IdentLength:
```

```
        ms[ --> C[ or V[, mw[ --> C[ or V[,
        ]ms --> ]C or ]V, ]mw --> ]C or ]V.

Constraint NucMs: ms --> V.
Constraint *BrOns: ms _|_ ]C and C[.
Constraint MwSon ("weak moras must decrease in sonority"):
        mw _|_ ]C and V[.

%%%%%%%%%%%%%%%%%%%
% Backness Harmony %
%%%%%%%%%%%%%%%%%%%

Constraint Segmental(VDom):
   VDom[ --> V[ or C[, ]VDom --> ]C or ]V.
Constraint Segmental(PDom):
   PDom[ --> V[ or C[, ]PDom --> ]C or ]V.

Constraint HasSize(VDom): VDom --> V.
Constraint HasSize(PDom): PDom --> V.

Constraint VDomBeginsVelar: VDom[ --> Vel[ or Vel.
Constraint DepLeft(VDom): VDom[ --> _Vel[.
Constraint DepLeftV(VDom): VDom[ --> _Vel[ or ]_Opq.
Constraint Dep(VDom): VDom --> _Vel.

Constraint PDomBeginsPalatal: PDom[ --> Pal[ or Pal.
Constraint DepLeft(PDom): PDom[ --> _Pal[.
Constraint DepLeftV(PDom): PDom[ --> _Pal[ or ]_Opq.
Constraint Dep(PDom): PDom --> _Pal.

% Basic alignment of Velar Spreading Domain
Constraint BA-left(VDom) ("Align(Vel-V, L, VDom, L)"):
        V[ and _Vel[ --> VDom[.

Constraint BA-right(VDom) ("Align(Vel-V, R, VDom, R)"):
        ]V and ]_Vel --> ]VDom.

% Basic alignment of Palatal Spreading Domain
Constraint BA-left(PDom) ("Align(Pal-V, L, PDom, L)"):
        V[ and _Pal[ --> PDom[.
```

```
Constraint BA-right(PDom) ("Align(Pal-V, R, PDom, R)"):
        ]V and ]_Pal --> ]PDom.

% Wide Scope Alignment of Velar Spreading Domain
Constraint WSA-right(VDom) ("Align(VDom, R, Wd, R)"):
        ]VDom --> ]Wd or PDom[ or C[, VDom _|_ ]Wd.

Constraint WSA-left(VDom) ("Align(VDom, L, Wd, L)"):
        VDom[ --> Wd[, VDom _|_ ]Wd.

Constraint VDomWide: VDom --> V[.
Constraint PDomWide: PDom --> V[.

% Wide Scope Alignment of Palatal Spreading Domain
Constraint WSA-right(PDom) ("Align(PDom, R, Wd, R)"):
        ]PDom --> ]Wd or VDom[ or C[, PDom _|_ ]Wd.

Constraint WSA-left(PDom) ("Align(PDom, L, Wd, L)"):
        PDom[ --> Wd[, PDom _|_ Wd[.

Constraint *VDomVDom: VDom[ _|_ ]VDom.
Constraint *PDomPDom: PDom[ _|_ ]PDom.

Constraint Anchor-V/P-Dom: V --> VDom or PDom.

Constraint *VDom: VDom _|_ VDom.
Constraint *PDom: PDom _|_ PDom.

Constraint *VDomPDom: VDom _|_ PDom.

Constraint Extend-V/P-Dom: ]VDom _|_ PDom[, ]PDom _|_ VDom[.

Constraint Express(VDom): VDom and V --> Vel.
Constraint Express(PDom): PDom and V --> Pal.

Constraint VDomUniformity: VDom _|_ ]Vel, VDom _|_ Vel[.
Constraint PDomUniformity: PDom _|_ ]Pal, PDom _|_ Pal[.

Constraint *SoloMarked:
        V and Rd and Pal --> RdDom or PDom,
        V and Flt and Vel and Hi --> FltDom or VDom.
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Obstruent Backness Harmony %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Constraint ExpressC(VDom): VDom and C and Obs --> Vel.
Constraint ExpressC(PDom): PDom and C and Obs --> Pal.

Constraint OpaqueCBarrier(VDom):
        VDom _|_ ]C and ]_Opq and ]_Pal and ]_Hi and ]_NSt.

Constraint OpaqueCBarrier(PDom):
        PDom _|_ ]C and ]_Opq and ]_Vel and ]_Hi and ]_NSt.

Constraint UniformSyl-V/P-Dom: ]VDom _|_ s, ]PDom _|_ s.

Constraint AssimilateC(VDom):
        ]C and ]Obs and V[ and VDom[ --> ]Vel,
        C[ and Obs[ and ]V and ]VDom --> Vel[.

Constraint AssimilateC(PDom):
        ]C and ]Obs and V[ and PDom[ --> ]Pal,
        C[ and Obs[ and ]V and ]PDom --> Pal[.

Constraint AssignVelPal: C --> Vel or Pal.

%%%%%%%%%%%%%%%%%%%%%
% Roundness Harmony %
%%%%%%%%%%%%%%%%%%%%%

Constraint Segmental(RdDom):
   RdDom[ --> V[, ]RdDom --> ]C or ]V.
Constraint Segmental(FltDom):
   FltDom[ --> V[, ]FltDom --> ]C or ]V.

Constraint HasSize(RdDom): RdDom --> V.
Constraint HasSize(FltDom): FltDom --> V.

Constraint RdDomBeginsRd: RdDom[ --> Rd[.
Constraint DepLeft(RdDom): RdDom[ --> _Rd[.
Constraint Dep(RdDom): RdDom --> _Rd.
```

```
Constraint FltDomBeginsFlt: FltDom[ --> Flt[.
Constraint DepLeft(FltDom): FltDom[ --> _Flt[.
Constraint Dep(FltDom): FltDom --> _Flt.

% Basic alignment of Rd Spreading Domain
Constraint BA-left(RdDom) ("Align(Rd-Hi-V, L, RdDom, L)"):
        V[ and _Rd[ --> RdDom[.

Constraint BA-right(RdDom) ("Align(Rd-Hi-V, R, RdDom, R)"):
        ]V and ]_Rd --> ]RdDom.

% Basic alignment of Flt Spreading Domain
Constraint BA-left(FltDom) ("Align(Flt-Hi-V, L, FltDom, L)"):
        V[ and _Flt[ --> FltDom[.

Constraint BA-right(FltDom) ("Align(Flt-Hi-V, R, FltDom, R)"):
        ]V and ]_Flt --> ]FltDom.

% Wide Scope Alignment of Rd Spreading Domain
Constraint WSA-right(RdDom) ("Align(RdDom, R, Wd, R)"):
        ]RdDom --> ]Wd or FltDom[ or _Lo[ or _NP[,
        RdDom _|_ ]Wd, RdDom _|_ _Lo[, RdDom _|_ _NP[.

% Wide Scope Alignment of Flt Spreading Domain
Constraint WSA-right(FltDom) ("Align(FltDom, R, Wd, R)"):
        ]FltDom --> ]Wd or RdDom[ or _Lo[ or _NP[,
        FltDom _|_ ]Wd, FltDom _|_ _Lo[, FltDom _|_ _NP[.

Constraint RdDomWide: RdDom --> V[.
Constraint FltDomWide: FltDom --> V[.
Constraint SpreadRdDomOverC: ]RdDom _|_ C[.
Constraint *RdDomRdDom: RdDom[ _|_ ]RdDom.
Constraint SpreadFltDomOverC: ]FltDom _|_ C[.
Constraint *FltDomFltDom: FltDom[ _|_ ]FltDom.
Constraint Anchor-Rd/Flt-Dom: V and Hi --> RdDom or FltDom.
Constraint *RdDom: RdDom _|_ RdDom.
Constraint *FltDom: FltDom _|_ FltDom.
Constraint *RdDomFltDom: RdDom _|_ FltDom.

Constraint Extend-Rd/Flt-Dom:
```

```
        ]RdDom _|_ FltDom[ _|_ Hi[, ]FltDom _|_ RdDom[ _|_ Hi[.

Constraint Express(RdDom):
   RdDom and V and Hi --> Rd, RdDom and C --> Rd.

Constraint Express(FltDom):
   FltDom and V and Hi --> Flt, FltDom and C --> Flt.

Constraint RdDomUniformity: RdDom _|_ ]Rd, RdDom _|_ Rd[.
Constraint FltDomUniformity: FltDom _|_ ]Flt, FltDom _|_ Flt[.

%%%%%%%%%%
% Laterals %
%%%%%%%%%%

Constraint FirstLatPal:
        Wd[ and Ant[ and Hi[ and Son[ and C[ --> Pal[.

Constraint LargePLDom: ]PLDom and PLDom[ _|_ s.
Constraint ProjectPLDom: V and Pal --> PLDom.
Constraint *PLDom: PLDom _|_ PLDom.

% Wide Scope Alignment of Lateral Palatal Spreading Domain
Constraint WSA-right(PLDom) ("Align(PLDom, R, s, R)"):
        ]PLDom --> ]s, PLDom _|_ ]s.

Constraint WSA-left(PLDom) ("Align(PLDom, L, s, L)"):
        PLDom[ --> s[, PLDom _|_ s[.

Constraint MorphemeInternal(PLDom):
   PLDom _|_ _Rt[, PLDom _|_ _Sfx[,
   PLDom _|_ ]_Rt, PLDom _|_ ]_Sfx.

Constraint WSA-settle-right(PLDom): ]PLDom --> ]_Rt or ]_Sfx.
Constraint WSA-settle-left(PLDom): PLDom[ --> _Rt[ or _Sfx[.

Constraint HasSize(PLDom): PLDom --> s.

Constraint Segmental(PLDom):
   PLDom[ --> V[ or C[, ]PLDom --> ]C or ]V.
```

```
Constraint LatAssimPal:
        ]PLDom and _C[ and _Son[ and _Hi[ --> Pal[,
        PLDom and _C and _Son and _Hi --> Pal.

Constraint LatMaxOpaque:
        _C and _Son and _Hi and _Opq and _Vel --> Vel,
        _C and _Son and _Hi and _Opq and _Pal --> Pal.

Constraint LatDefault:
        _C and _Son and _Hi and _Ant --> Vel.

% A weird one -- a lateral may not begin a P or V domain.
Constraint *LatPVDomLeft:
        _C[ and _Son[ and _Hi[ and _Ant[ _|_ PDom[,
        _C[ and _Son[ and _Hi[ and _Ant[ _|_ VDom[.

%%%%%%%%%%%%%
% Epenthesis %
%%%%%%%%%%%%%

Constraint Segmental(Hi): Hi[ --> C[ or V[, ]Hi --> ]C or ]V.

Constraint VowelHeightDefaults:
   V and NP --> _V, V and Lo --> _V.

% Word-initial velar consonants before another underlying
% consonant must be [+back].

% These three produce a silly domain that identifies the first
% underlying consonant cluster in a word, if there is such
% a thing.
Constraint Project1stCs: _C[ and Wd[ --> 1stCs[.
Constraint Bound1stCs: ]1stCs --> _V[, 1stCs _|_ _V[.
Constraint *1stCs: 1stCs _|_ 1stCs.

Constraint WdInitKGVelar:
   1stCs and ]_C and _C[ and ]_Obs and ]_Hi and ]_NSt --> Vel,
   1stCs and ]_C and ]_Obs and ]_Hi and ]_NSt _|_ Vel[.

% Constraints on allowable coda clusters.
% Allowed clusters are:
```

```
%       Son & Obs
%       Voiceless Fricative & Oral Stop
%       k + s
Constraint CodaClusterProps:
        ]C and C[ and mw --> Obs[,
        ]C and C[ and mw and ]Obs _|_ ]Vcd,
        ]C and C[ and mw and ]Obs --> ]Cnt or ]Hi,
        ]C and C[ and mw and ]Obs and ]Hi --> ]NSt,
        ]C and C[ and mw and ]Obs and Obs[ and ]Cnt _|_ Cnt[,
        ]C and C[ and mw and ]Obs and ]Hi and ]NSt --> Cnt[,
        ]C and C[ and mw and ]Obs and ]Hi and ]NSt --> Ant[,
        ]C and C[ and mw and ]Obs and ]Hi and ]NSt --> Cor[,
        ]C and C[ and mw and ]Obs and ]Hi and ]NSt _|_ Vcd[.

% Can't add vowels outside the root!
Constraint AlignWdMorph ("Align(Wd, R; Morph, R)"):
        ]Wd --> ]_Rt or ]_Sfx.

% In order to detect an inter-morpheme insertion, we have
% to project the Rt and Sfx tiers out.

Constraint Dep(Rt):
   Rt[ --> _Rt[, ]Rt --> ]_Rt,
   Rt _|_ _Rt[, Rt _|_ ]_Rt.

Constraint Max(Rt): _Rt[ --> Rt[, ]_Rt --> ]Rt.

Constraint Dep(Sfx):
   Sfx[ --> _Sfx[, ]Sfx --> ]_Sfx,
   Sfx _|_ _Sfx[, Sfx _|_ ]_Sfx.

Constraint Max(Sfx): _Sfx[ --> Sfx[, ]_Sfx --> ]Sfx.

Constraint NoMorphGaps:
   ]Rt --> Sfx[ or ]Wd, ]Sfx --> Sfx[ or ]Wd.

%%%%%%%%%
% Misc. %
%%%%%%%%%

% Final Obstruent Devoicing
```

```
Constraint Segmental(Vcd):
   Vcd[ --> C[ or V[, ]Vcd --> ]C or ]V.

Constraint MaxSon(Vcd):
   ]V --> ]Vcd, V[ --> Vcd[, ]Son --> ]Vcd, Son[ --> Vcd[.

Constraint MaxCnt(Vcd):
   ]_Cnt and ]_Vcd --> ]Vcd, _Cnt[ and _Vcd[ --> Vcd[.

Constraint Final_Obstruent_Devoicing: ]_Obs and ]Wd _|_ ]Vcd.
```

B.2.2 Rankings File

```
Inviolable:
        Dep(C), Max(C), Max(V), *CV
        AlignSyl, ParseC, ParseV, HNuc
        ProjectWd, AllInWord, PackCs, PackVs
        Align(Wd, L, s, L), Align(Wd, R, s, R)
        PackSyls, Align(s, L, ms, L), NucMs, *MsMw
        Align(s, R, m, R), Align(ms, L, s, L)
        Parse(ms), Parse(mw), *RepeatMw, CodaMw, MwSon,
        PackMoras, IdentLength

Violable:
        *Wd
        Ons
        *BrOns

        Dep(Obs), Max(Obs), Dep(Cor), Max(Cor)
        Dep(Ant), Max(Ant), Dep(Cnt), Max(Cnt)
        Dep(NSt), Max(NSt)
        DepC(Hi), Max(Hi), Segmental(Hi)
        Segmental(Vcd), Max(Son), *SonObs, VoiceSon,

        MaxCnt(Vcd)
        CodaClusterProps
        Dep(V), Dep(Vcd), Dep(Son)
        Final_Obstruent_Devoicing
```

```
Max(Vcd)
*Coda
AlignWdMorph
Dep(Rt), Max(Rt), Dep(Sfx), Max(Sfx)
NoMorphGaps

Dep(Ns), Max(Ns)
Dep(Lab), Max(Lab), Segmental(Vel)

Project1stCs
Bound1stCs
*1stCs
WdInitKGVelar

Dep(VDom), Segmental(VDom), VDomBeginsVelar
HasSize(VDom), OpaqueCBarrier(VDom), *VDomVDom
Segmental(Pal), VelPalMandatory, *PalVel
FirstLatPal
Dep(PDom), Segmental(PDom), PDomBeginsPalatal
HasSize(PDom), OpaqueCBarrier(PDom), *PDomPDom
Anchor-V/P-Dom
*VDomPDom
Segmental(Lo), *HiLo
Segmental(NP), *LoNP, *HiNP
VowelHeightDefaults
AssignHeight
Dep(Hi)
LoDefaults
Segmental(Rd), Segmental(Flt), *RdFlt
BackVowelInv
RdFltMandatory

Dep(RdDom), Segmental(RdDom), RdDomBeginsRd
HasSize(RdDom)

Dep(FltDom), *RdDomFltDom, Segmental(FltDom)
FltDomBeginsFlt, HasSize(FltDom)
Anchor-Rd/Flt-Dom

RootMaxLeft(Vel, Rd), RootMaxLeft(Flt, Pal)
RootMaxLeft(Lo)
```

```
OpaqueMaxLeft(Vel, Rd), OpaqueMaxLeft(Flt, Pal)
OpaqueMaxLeft(Lo)
OpaqueMaxUEI
OpaqueMaxC(Vel), OpaqueMaxC(Pal)

Express(VDom), Express(PDom)
ExpressC(VDom), ExpressC(PDom)
UniformSyl-V/P-Dom
Express(RdDom), Express(FltDom)
WSA-right(VDom), WSA-right(PDom)
WSA-right(RdDom), WSA-right(FltDom)
*LatPVDomLeft
VDomWide, PDomWide
RdDomWide, FltDomWide
DepLeftV(VDom), DepLeftV(PDom)
DepLeft(FltDom), DepLeft(RdDom)
SpreadRdDomOverC, *RdDomRdDom
SpreadFltDomOverC, *FltDomFltDom
Extend-V/P-Dom
DepLeft(VDom), DepLeft(PDom)
Extend-Rd/Flt-Dom
BA-left(VDom), BA-left(PDom)
BA-left(RdDom), BA-left(FltDom)
*VDom, *PDom
*RdDom, *FltDom
*SoloMarked
BA-right(VDom), BA-right(PDom)
BA-right(RdDom), BA-right(FltDom)
AssimilateC(VDom), AssimilateC(PDom)

ProjectPLDom, Segmental(PLDom), HasSize(PLDom)
MorphemeInternal(PLDom)
LargePLDom
WSA-left(PLDom)
WSA-right(PLDom)
WSA-settle-left(PLDom)
WSA-settle-right(PLDom)
*PLDom
LatAssimPal
LatMaxOpaque
LatDefault
```

```
Dep(Vel), Max(Vel), Dep(Pal), Max(Pal)
Dep(Rd), Max(Rd), Dep(Flt), Max(Flt)
Max(Lo), Dep(Lo), Max(NP), Dep(NP)

RdDomUniformity, FltDomUniformity
VDomUniformity, PDomUniformity
WSA-left(VDom), WSA-left(PDom)
AssignVelPal
```

B.3 Inputs and Outputs

Here the first two lines of each block represent the input and the third line is the transcription of the output given by the program for that input.

B.3.1 Standard Vowel Harmony, Genitive Plurals

```
Input of-ropes:
    { [_Rt]{ i p } [_Sfx]{ l e r } [_Sfx ]{ i n } }.
 i  p . ^l  e . r  i  n

Input of-girls:
    { [_Rt]{ k ibar z } [_Sfx]{ l e r } [_Sfx]{ i n } }.
 k  -i  z . l  a . r  -i  n

Input of-faces:
    { [_Rt]{ y ue z } [_Sfx]{ l e r } [_Sfx]{ i n } }.
 y  ue  z . ^l  e . r  i  n

Input of-stamps:
    { [_Rt]{ p u l } [_Sfx]{ l e r } [_Sfx]{ i n } }.
 p  u  l . l  a . r  -i  n

Input of-hands:
    { [_Rt]{ e l } [_Sfx]{ l e r } [_Sfx]{ i n } }.
 e  ^l . ^l  e . r  i  n
```

```
Input of-stalks:
    { [_Rt]{ s a p } [_Sfx]{ l e r } [_Sfx]{ i n } }.
 s  a  p . l  a . r  -i  n

Input of-villages:
    { [_Rt]{ k oe y } [_Sfx]{ l e r } [_Sfx]{ i n } }.
 ^k  oe  y . ^l  e . r  i  n

Input of-ends:
    { [_Rt]{ s o n } [_Sfx]{ l e r } [_Sfx]{ i n } }.
 s  o  n . l  a . r  -i  n
```

B.3.2 Standard Vowel Harmony, Genitive Singulars

```
Input of-rope:
    { [_Rt]{ i p } [_Sfx ]{ i n } }.
 i . p  i  n

Input of-girl:
    { [_Rt]{ k ibar z } [_Sfx]{ i n } }.
 k  -i . z  -i  n

Input of-face:
    { [_Rt]{ y ue z } [_Sfx]{ i n } }.
 y  ue . z  ue  n

Input of-stamp:
    { [_Rt]{ p u l } [_Sfx]{ i n } }.
 p  u . l  u  n

Input of-hand:
    { [_Rt]{ e l } [_Sfx]{ i n } }.
 e . ^l  i  n

Input of-stalk:
    { [_Rt]{ s a p } [_Sfx]{ i n } }.
 s  a . p  -i  n
```

```
Input of-village:
    { [_Rt]{ k oe y } [_Sfx]{ i n } }.
 ^k  oe . y  ue  n

Input of-end:
    { [_Rt]{ s o n } [_Sfx]{ i n } }.
 s  o . n  u  n
```

B.3.3 Regularization

```
Input communism:
    { [_Rt]{ k o m ue n i z i m } }.
 k  o . m  u . n  i . z  i  m

Input mercerized:
    { [_Rt]{ m e r s oe r i z e } }.
 m  e  r . s  e . r  i . z  e

Input cigar:
    { [_Rt]{ p ue r o } }.
 p  u . r  o
```

B.3.4 Opaque Vowels in Suffixes

```
Input I_am_coming:
    { [_Rt]{ g e l } [_Sfx]{ i y o-opq r } [_Sfx]{ i m } }.
 ^g  e . ^l  i . y  o . r  u  m

Input I_am_running:
    { [_Rt]{ k o sh } [_Sfx]{ i y o-opq r } [_Sfx]{ i m } }.
 k  o . sh  u . y  o . r  u  m

Input I_am_laughing:
    { [_Rt]{ g ue l } [_Sfx]{ i y o-opq r } [_Sfx]{ i m } }.
 ^g  ue . ^l  ue . y  o . r  u  m
```

```
Input I_am_looking:
    { [_Rt]{ b a k } [_Sfx]{ i y o-opq r } [_Sfx]{ i m } }.
 b  a . k  -i . y  o . r  u  m

Input triangles:
    { [_Rt]{ ue ch } [_Sfx]{ g e-opq n } [_Sfx]{ l e r } }.
 ue  ch . ^g  e  n . ^l  e  r

Input hexagonals:
    { [_Rt]{ a l t ibar } [_Sfx]{ g e-opq n } [_Sfx]{ l e r } }.
 a  l . t  -i . ^g  e  n . ^l  e  r

Input octagonals:
    { [_Rt]{ s e k i z } [_Sfx]{ g e-opq n } [_Sfx]{ l e r } }.
 s  e . ^k  i  z . ^g  e  n . ^l  e  r

Input polygonals:
    { [_Rt]{ ch o k } [_Sfx]{ g e-opq n } [_Sfx]{ l e r } }.
 ch  o  k . ^g  e  n . ^l  e  r

Input Arabia:
    { [_Rt]{ a r a b } [_Sfx]{ i-opq s t a-opq n } [_Sfx]{ i } }.
 a . r  a . b  i  s . t  a . n  -i

Input Armenia:
    { [_Rt]{ e r m e n } [_Sfx]{ i-opq s t a-opq n }
      [_Sfx]{ i } }.
 e  r . m  e . n  i  s . t  a . n  -i

Input Mongolia:
    { [_Rt]{ m o o l } [_Sfx]{ i-opq s t a-opq n } [_Sfx]{ i } }.
 m  o  o . l  i  s . t  a . n  -i

Input Turkestan:
    { [_Rt]{ t ue-opq r k } [_Sfx]{ i-opq s t a-opq n }
      [_Sfx]{ i } }.
 t  ue  r . ^k  i  s . t  a . n  -i

Input let_him_keep_going:
    { [_Rt]{ g i d } [_Sfx]{ e d u-opq r } [_Sfx]{ s i n } }.
 ^g  i . d  e . d  u  r . s  u  n
```

```
Input let_him_keep_running:
    { [_Rt]{ k o sh } [_Sfx]{ e d u-opq r } [_Sfx]{ s i n } }.
 k  o . sh  a . d  u  r . s  u  n

Input let_him_keep_laughing:
    { [_Rt]{ g ue l } [_Sfx]{ e d u-opq r } [_Sfx]{ s i n } }.
 ^g  ue . ^l  e . d  u  r . s  u  n

Input let_him_keep_looking:
    { [_Rt]{ b a k } [_Sfx]{ e d u-opq r } [_Sfx]{ s i n } }.
 b  a . k  a . d  u  r . s  u  n
```

B.3.5 Opaque Velars

```
Input bachelor:
    { [_Rt]{ b e ^k-opq a r } }.
 b  e . ^k  a  r

Input bachelor-non-opq:
    { [_Rt]{ b e k a r } }.
 b  e . k  a  r

Input residence:
    { [_Rt]{ m a l i k } [_Sfx]{ a-opq n e-opq } }.
 m  a . ^l  i . k  a . n  e

Input residence-opq:
    { [_Rt]{ m a l i ^k-opq } [_Sfx]{ a-opq n e-opq } }.
 m  a . ^l  i . ^k  a . n  e

Input explosion-nom:
   { [_Rt]{ i n f i l a ^k-opq } }.
 i  n . f  i . ^l  a  k

Input explosion-acc:
   { [_Rt]{ i n f i l a ^k-opq } [_Sfx]{ i } }.
 i  n . f  i . ^l  a . ^k  i
```

```
Input perception-nom-pl:
    { [_Rt]{ i d r a ^k-opq } [_Sfx]{ l e r } }.
 i  d . r  a  k . ^l  e  r

Input rak-nom-pl:
    { [_Rt]{ r a k } [_Sfx]{ l e r } }.
 r  a  k . l  a  r
```

B.3.6 Lateral Backness Assimilation

```
Input expression:
    { [_Rt]{ l a f } }.
 ^l  a  f

Input gurgle:
    { [_Rt]{ l ibar k ibar r d a } }.
 ^l  -i . k  -i  r . d  a

Input expression-falsely-opaque:
    { [_Rt]{ l-opq a f } }.
 ^l  a  f

Input heart:
    { [_Rt]{ k a ^l-opq p } }.
 k  a  ^l  p

Input counterfeit:
    { [_Rt]{ k a l p } }.
 k  a  l  p

Input drink:
    { [_Rt]{ b o ^l-opq } }.
 b  o  ^l

Input abundant:
    { [_Rt]{ b o l } }.
 b  o  l

Input cello1:
```

```
    { [_Rt]{ ch e l o } }.
 ch  e . ^l  o

Input cello2:
    { [_Rt]{ ch e ^l o } }.
 ch  e . ^l  o

Input cello3:
    { [_Rt]{ ch e l-opq o } }.
 ch  e . ^l  o

Input pen:
    { [_Rt]{ k a l e m } }.
 k  a . ^l  e  m

Input Islam:
    { [_Rt]{ i s l a m } }.
 i  s . ^l  a  m

Input ball:
    { [_Rt]{ b a l o } }.
 b  a . l  o

Input justly:
    { [_Rt]{ a d i l } [_Sfx]{ a-opq n e-opq } }.
 a . d  i . ^l  a . n  e

Input system-nom:
    { [_Rt]{ u s u ^l-opq } }.
 u . s  u  ^l

Input system-acc:
    { [_Rt]{ u s u ^l-opq } [_Sfx]{ i } }.
 u . s  u . ^l  ue

Input fork-nom:
    { [_Rt]{ ch a t a l } }.
 ch  a . t  a  l

Input fork-acc:
    { [_Rt]{ ch a t a l } [_Sfx]{ i } }.
```

```
 ch  a . t  a . l  -i
```

B.3.7 Epenthesis

```
Input prince:
   { [_Rt]{ p r e n s } }.
 p  i . r  e  n  s

Input bromide:
   { [_Rt]{ b r o m } }.
 b  u . r  o  m

Input direct:
   { [_Rt]{ d r e k } }.
 d  i . r  e  ^k

Input probation:
   { [_Rt]{ s t a zh } }.
 s  -i . t  a  zh

Input king:
   { [_Rt]{ k r a l } }.
 k  -i . r  a  l

Input grippe:
   { [_Rt]{ g r i p } }.
 g  -i . r  i  p

Input idea:
   { [_Rt]{ f i k r } }.
 f  i . ^k  i  r

Input his-idea:
   { [_Rt]{ f i k r } [_Sfx]{ i } }.
 f  i  ^k . r  i

Input bosom:
   { [_Rt]{ k o y n } }.
 k  o . y  u  n
```

```
Input her-bosom:
   { [_Rt]{ k o y n } [_Sfx]{ i } }.
 k  o  y . n  u

Input judgement:
   { [_Rt]{ h ue k m } }.
 h  ue . ^k  ue  m

Input his-judgement:
    { [_Rt]{ h ue k m } [_Sfx]{ i } }.
 h  ue  ^k . m  ue

Input part:
    { [_Rt]{ k ibar s m } }.
 k  -i . s  -i  m

Input its-part:
    { [_Rt]{ k ibar s m } [_Sfx]{ i } }.
 k  -i  s . m  -i

Input patience:
    { [_Rt]{ s a b r } }.
 s  a . b  -i  r

Input her-patience:
    { [_Rt]{ s a b r } [_Sfx]{ i } }.
 s  a  b . r  -i

Input text:
    { [_Rt]{ m e t n } }.
 m  e . t  i  n

Input his-text:
    { [_Rt]{ m e t n } [_Sfx]{ i } }.
 m  e  t . n  i

Input strange:
    { [_Rt]{ g a r i b } }.
 g  a . r  i  p
```

```
Input his-strange:
    { [_Rt]{ g a r i b } [_Sfx]{ i } }.
 g  a . r  i . b  i

Input fake-strange:
    { [_Rt]{ g a r b } }.
 g  a  r  p

Input time:
    { [_Rt]{ v a ^k-opq t } }.
 v  a . ^k  i  t

Input his-time:
    { [_Rt]{ v a ^k-opq t } [_Sfx]{ i } }.
 v  a  k . t  i

Input from-time:
    { [_Rt]{ v a ^k-opq t } [_Sfx]{ t e n } }.
 v  a . ^k  i  t . t  e  n
```

Bibliography

Brzozowski, J. A. 1962. Canonical regular expressions and minimal state graphs for definite events. In *Mathematical theory of Automata*, Vol. 12 of *MRI Symposia Series*, 529–561, Brooklyn, N.Y. Polytechnic Press of Brooklyn, Polytechnic Press.

Clements, G. N., and E. Sezer. 1982. Vowel and consonant disharmony in Turkish. In H. van der Hulst and N. Smith (Eds.), *The structure of phonological representations: Part II*, 213–356. Dordrecht: Foris.

Clements, G. N. 1985. The geometry of geometrical features. *Phonology Yearbook* 2:223–52.

Cole, J., and C. Kisseberth. 1994. An optimal domains theory of harmony. *Studies in the Linguistic Sciences* 24(2).

Deng, L. 1996. Finite-state automata derived from overlapping articulatory features: a novel phonological construct for speech recognition. In R. Sproat (Ed.), *Proceedings of the Second Meeting of the ACL Special Interest Group in Computational Phonology*, Santa Cruz, July. ACL.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* 1:269–271.

Eisner, J. 1997a. Efficient generation in primitive Optimality Theory. In *Proceedings of the ACL*.

Eisner, J. 1997b. What constraints should OT allow? Handout for talk at LSA, Chicago, January.

Ellison, T. M. 1994. Phonological derivation in Optimality Theory. In *Coling*, Vol. II, 1007–1013, Kyoto, Japan.

Flemming, E. 1995. The analysis of contrast and comparative constraints in phonology. MIT Phonology Circle Lecture.

Freund, Y., M. Kearns, D. Ron, R. Schapire, and L. Sellie. 1993. Efficient learning of typical finite automata from random walks. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, 315–324.

Goldsmith, J. A. 1976. *Autosegmental Phonology*. PhD thesis, Massachusetts Institute of Technology.

Hayes, B. P. 1996. Phonetically driven phonology: The role of Optimality Theory and Inductive Grounding. In *Proceedings of the Milwaukee Conference on Formalism and Functionalism in Linguistics*, October.

Jampel, M. 1996a. A brief overview of over-constrained systems. In M. Jampel, E. Freuder, and M. Maher (Eds.), *Over-Constrained Systems*, no. 1106 in LNCS, 1–22. Berlin: Springer.

Jampel, M. 1996b. *Over-Constrained Systems in CLP and CSP*. PhD thesis, City University, September.

Kay, M. 1977. Morphological analysis and syntactic analysis. In *Linguistic Structures Processing*. Amsterdam: North-Holland.

Kenstowicz, M. 1995. Base-identity and uniform exponence: Alternatives to cyclicity. In J. Durand and B. Laks (Eds.), *Current Trends in Phonology: Models and Methods*. CNRS, Paris-X, and University of Salford Publications.

McCarthy, J., and A. Prince. 1993. Generalized alignment. In G. Booij and J. van Marle (Eds.), *Yearbook of Morphology*, 79–153. Dordrecht: Kluwer.

McCarthy, J., and A. Prince. 1995. Faithfulness and reduplicative identity. In J. Beckman, S. Urbanczyk, and L. Walsh (Eds.), *Papers in Optimality Theory*, no. 18 in University of Massachusetts Occasional Papers, 259–384. UMass, Amherst: GLSA.

Padgett, J. 1995. Feature classes. In J. Beckman, S. Urbanczyk, and L. Walsh (Eds.), *Papers in Optimality Theory*, no. 18 in University of Massachusetts Occasional Papers. UMass, Amherst: GLSA.

Prince, A., and P. Smolensky. 1993. Optimality Theory: Constraint interaction in generative grammar. Technical Report 2, Center for Cognitive Science, Rutgers University.

Sagey, E. 1986. *The Representation of Features and Relations in Nonlinear Phonology*. PhD thesis, Massachusetts Institute of Technology.

Steriade, D. 1996. Paradigm uniformity and the phonetics-phonology boundary. In *5th Conference in Laboratory Phonology*, Evanston, Illinois.