SYSTEMATIC PARAMETERIZED COMPLEXITY ANALYSIS IN COMPUTATIONAL PHONOLOGY

By

© Harold Todd Wareham

A thesis submitted to the School of Graduate Studies in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Department of Computer Science
University of Victoria
September 1998

Abstract

Many computational problems are NP-hard and hence probably do not have fast, i.e., polynomial time, algorithms. Such problems may yet have non-polynomial time algorithms, and the non-polynomial time complexities of these algorithms will be functions of particular aspects of that problem, i.e., the algorithm's running time is upper bounded by $f(k)|x|^c$, where f is an arbitrary function, $|x|$ is the size of the input x to the algorithm, k is an aspect of the problem, and c is a constant independent of $|x|$ and k. Given such algorithms, it may still be possible to obtain optimal solutions for large instances of NP-hard problems for which the appropriate aspects are of small size or value. Questions about the existence of such algorithms are most naturally addressed within the theory of parameterized computational complexity developed by Downey and Fellows.

This thesis considers the merits of a systematic parameterized complexity analysis in which results are derived relative to all subsets of a specified set of aspects of a given NP-hard problem. This set of results defines an "intractability map" that shows relative to which sets of aspects algorithms whose non-polynomial time complexities are purely functions of those aspects do and do not exist for that problem. Such maps are useful not only for delimiting the set of possible algorithms for an NP-hard problem but also for highlighting those aspects that are responsible for this NP-hardness.

These points will be illustrated by systematic parameterized complexity analyses of problems associated with five theories of phonological processing in natural languages – namely, Simplified Segmental Grammars, finite-state transducer based rule systems, the KIMMO system, Declarative Phonology, and Optimality Theory. The aspects studied in these analyses broadly characterize the representations and mechanisms used by these theories. These analyses suggest that the computational complexity of phonological processing depends not on such details as whether a theory uses rules or constraints or has one, two, or many levels of representation but rather on the structure of the representation-relations encoded in individual mechanisms and the internal structure of the representations.

Contents

List of Tables

List of Figures

Acknowledgments

This dissertation is the product of several years work carried out at the University of Victoria and McMaster University. In that time, I have come to owe a great deal to a great many. Lack of space precludes mentioning them all. The efforts of those not named below must be acknowledged by the fact that I am, at long last, writing these acknowledgments.

First off, I would like to thank various departmental staff members at the University of Victoria and McMaster University, namely, Marg Belec, Isabel Campos, Nancy Chan, Helen Graham, Sharon Moulson, Marla Serfling, and Natasha Swain, for their courteous and ever-helpful escorting through the red tape that is an unavoidable part of both graduate studies and academia. I would also like to thank the various systems personnel I've dealt with, namely Chris Bryce, Alan Idler, Will Kastelic, Matt Kelly, and Derek Lipiec, for answering an almost never-ending stream of odd questions about obscure programs and LaTeX typesetting and, through their efforts, keeping my files and my sanity intact.

I have had the good fortune to attend many meetings over the course of my Ph.D., which has very much enriched my work – thanks for this goes to my Ph.D. advisor, Mike Fellows, and my postdoc supervisor, Tao Jiang. I would also like to acknowledge Eric Ristad for providing an NSF grant for me to attend the DIMACS Workshop on Human Language at Princeton in March of 1992, which awakened my interest in my dissertation topic, and Mark Kas for providing a very generous student grant for me to attend the Summer School in Behavioral and Cognitive Neurosciences at BCN Groningen in July of 1996, which introduced me to the wide world of finite-state natural language processing.

I have had the good fortune to meet and interact with many colleagues while researching and writing my dissertation. I would like to thank the members of the Theory Group at University of Victoria, namely John Ellis, Val King, Wendy Myrvold, and Frank Ruskey, for their many calming talks and advice about academic life. I would also like to thank John Coleman, Jason Eisner, Mark Ellison, Susan Fitzgerald, Ayse Karaman, Lauri Karttunen, Martin Kay, András Kornai, Eric Laporte, Mehryar Mohri, Thomas Ngo, Alexis Manaster Ramer, Eric Ristad, Giorgio Satta, Bruce Tesar, Alain Theriault, John Tsotsos, Gertjan van Noord, and Markus Walther for sharing unpublished manuscripts, reprints, and delightful conversations over the years, both in person and by e-mail. Last but by no means least in this list are various members of the parameterized complexity research community, namely Liming Cai, Kevin Cattell, Marco Cesati, Mike Dinneen, Rod Downey, Patricia Evans, Mike Fellows, and Mike Hallett. Many of the ideas in my dissertation came from conversations with these people – in particular, Mike Hallett, through his early concern with using parameterized complexity to analyze real-world data sets, is the man behind systematic parameterized complexity analysis. To all of them, for both intellectual and emotional support, I owe a great thanks.

I would like to thank the members of my Ph.D. committee, namely Ewa Czaykowska-Higgins, Val King, Frank Ruskey, and Bruce Watson, for seeing me through the process of writing and and defending my dissertation. I would especially like to thank Ewa Czaykowska-Higgins, for constant encouragement, detailed and prompt commentaries on various drafts of my dissertation, and probing questions about the relationship between complexity-theoretic results and linguistic research. She has made me re-think many things and, in that process, has had a great impact on both my thought and my dissertation. If I am ever called upon to supervise graduate students, I hope that I can do so with some part of the grace, energy, and intelligence she has shown in her dealings with me.

I would like to thank my friends past and present for helping me through the bad times and making the good times better. A short list would include Marg Belec, Kathy Beveridge, Gord Brown, Bette Bultena, Kevin Cattell, Claudio Costi, Bill Day, Gianluca Della Vedova, Mike Dinneen, Patricia Evans, Susan Fitzgerald, Emmanuel Francois, Alan Goulding, Mike Hallett, Mike Hu, Lou Ibarra, Matt Kelly, Scott Lausch, Dianne Miller, John Nakamura, Joe Sawada, Brian Shea, Bill Threlfall, Chris Trendall, Jeff Webb, and Xian Zhang; there are many others. Thank you all.

I would like to thank my Ph.D. advisor, Mike Fellows, for taking me on as his student and helping me to appreciate more fully the beauty at the heart of computational complexity theory. I have admired his intelligence, his curiosity, his energy, his commitment to rigor, and his openness and generosity with ideas since the day we met, and will hold these qualities as ideals in my own academic life in years to come. I admire the same things in my postdoc supervisor, Tao Jiang; in addition, I must thank him for his patience in allowing me to complete the research and writeup of my dissertation at McMaster University.

I would like to thank the members of my family – my parents, my brother and sister and their spouses, and my nieces – for being who they are, and making the life that surrounds my work worthwhile.

Finally, I would like to thank Vit Bubenik and Aleksandra Steinbergs. Many years ago, Dr. Bubenik taught me my first courses in linguistic analysis and encouraged me to continue in linguistics; Dr. Steinbergs subsequently taught me phonology, in two of the best courses I have ever had the pleasure to take, and was my supervisor for an uncompleted undergraduate honours dissertation on word stress in Russian. I owe them both for introducing me to phonology and for showing me its often unappreciated beauty. I hope that this dissertation is partial payment on that debt.

To Bill Day and Bill Threlfall
for getting me here.

All this is fact. Fact explains nothing.
On the contrary, it is fact that requires explanation.

Marilynne Robinson, *Housekeeping*

Chapter 1

Introduction

It is not unreasonable to assume that if one is given a computational problem, one wants the algorithm that will solve that problem using the least amount of computer time. The discipline of algorithm design responds to this need by creating efficient algorithms for problems, where efficiency is usually judged in terms of **(asymptotic worst-case) time complexity** i.e., a function that upper-bounds the worst-case time requirements of an algorithm over all input sizes. As one is typically interested in larger and larger problem sizes (a trend some have characterized as "living in asymptopia"), one would like algorithms whose worst-case time complexities grows slowly as a function of input size. Ideally, this function should be a (low-order) polynomial. A problem that has such an algorithm is said to be **polynomial time tractable** and one that does not is said to be **polynomial time intractable**.

There are a number of computational problems that have defied almost fifty years of effort to create polynomial time algorithms for them, and are thus widely suspected (but not proven to be) polynomial time intractable. In part to prevent wasting effort in trying to solve other such problems efficiently, various theories of computational complexity have been developed from the mid-1960's onwards [GJ79, Pap94]. Essentially, each such theory proposes a class $\mathcal{C}$ of problems that is either known or strongly conjectured to properly include the class P of polynomial time tractable problems. By establishing that a given problem is exactly as ($\mathcal{C}$-complete) or at least as ($\mathcal{C}$-hard) computationally difficult as the most computationally difficult problems in $\mathcal{C}$, one can show that this problem does not have a polynomial time algorithm unless the conjecture $P \neq \mathcal{C}$ is false. The strength of such conjectures is typically based on the $\mathcal{C}$-hardness of at least one of the suspected polynomial time intractable problems alluded to above. The most famous and widely-used of these theories is that for NP-completeness [GJ79].

When one must solve large instances of a problem, a proof of NP-hardness for that problem is often taken as a license to indulge in any and all manners of heuristic algorithms that give approximate solutions in polynomial time, e.g., randomized algorithms, bounded-cost approximation schemes, simulated annealing. However, this reaction may be premature. A proof of NP-hardness only establishes that a problem does not have a polynomial time algorithm unless $P = NP$. An NP-hard problem may still have non-polynomial time algorithms; moreover, some of these algorithms may have time complexity functions such that the non-polynomial terms of these functions are purely functions of particular aspects of that problem, i.e., the time complexity of the algorithm is $f(k)||x|^c$ where f is an arbitrary function, $|x|$ is the size of a given instance x of the problem, k is some aspect of that problem, and c is a constant independent of $|x|$ and k. Given such algorithms, it may still be possible to obtain optimal solutions for large instances of NP-hard problems encountered in practice for which the appropriate aspects are of bounded size or value (as such bounded aspects may reduce the non-polynomial terms in the time complexity function of such an algorithm to polynomials or constants and thus make the algorithm run in polynomial time).

This raises the following questions:

- How does one determine if a problem of interest has such algorithms?
- Relative to which aspects of that problem do such algorithms exist?

These questions are by and large unanswerable within classical theories of computational complexity like NP-completeness because these theories can show only whether a problem does or (probably) does not have a polynomial time algorithm. However, such issues can be addressed within the theory of parameterized computational complexity [DF95a, DF95b, DF99]. That is, given a problem that is suspected to be polynomial time intractable and a set of aspects of that problem, one can prove within this theory whether there does or does not (modulo various conjectures) exist an algorithm for that problem whose non-polynomial time complexity is purely a function of those aspects.

Individual parameterized results have proven useful in answering algorithmic questions about problems drawn from areas as diverse as VLSI design, computational biology, and scheduling (see [DF99] and references). For example, one such result establishes that there is probably no algorithm for the 3-D robot motion planning problem whose non-polynomial time complexity is purely a function of the number of joints in the given robot [CW95]. In this thesis, I will discuss the merits of a systematic parameterized complexity analysis in which parameterized results are derived relative to all subsets of a specified set of aspects of a given NP-hard problem. Modulo various conjectures, this set of results defines a **polynomial time intractability map** that shows relative to which sets of aspects non-polynomial algorithms do and do not exist for that problem. Such maps are useful not only for delimiting the set of possible algorithms for a particular NP-hard problem but also for highlighting those aspects of that problem that are responsible for this NP-hardness, i.e., those aspects of that problem that are **sources of polynomial-time intractability**.

The process of creating and using intractability maps will be illustrated by systematic parameterized complexity analyses of problems associated with five theories of phonological processing. Phonology is the area of linguistics concerned with the relationship between spoken (surface) and mental (lexical) forms in natural languages [Ken94]. Each theory of phonological processing proposes types of representations for lexical and surface forms and mechanisms that implement the mappings between these forms. Two problems associated with each such theory are the encoding and decoding problems, which are concerned with the operations within that theory of creating surface forms from lexical forms and recovering lexical forms from surface forms, respectively. The following five theories of phonological processing will be examined in this thesis:

1. Simplified Segmental Grammars [Ris93b].

2. Finite-state transducer (FST) based rule systems [KK94].

3. The KIMMO system [Kar83, Kos83].

4. Declarative Phonology [Sco92, SCB96].

5. Optimality Theory [MP93, PS93].

These theories together constitute a historical and methodological continuum of the various types of theories that have been used over the last 30 years to describe phonological phenomena. The encoding and decoding problems for each theory will be analyzed relative to a set of aspects that broadly characterizes both the representations and mechanisms proposed by that theory. In addition to providing some of the first NP-hardness results for problems associated with several of these theories, these analyses also comprise the first formal framework in which various published speculations about the sources of polynomial-time intractability in these theories can be investigated.

Grand rhetoric aside, a note is in order concerning the choice of the five theories listed above for analysis in this thesis. Optimality Theory, Declarative Phonology, and FST-based rule systems are obvious candidates for analysis, as Optimality Theory and Declarative Phonology are the most popular descriptive frameworks at the present time and the finite-state methods underlying FST-based rule systems are the preferred manner of software implementation of such frameworks. The cases for Simplified Segmental Grammars and the KIMMO system are slightly weaker, as both are older formalisms that are no longer in favor. However, they are still well worth analyzing, both because they are two of the very few linguistic theories that have had their computational complexities analyzed and debated in the literature and, perhaps more importantly, because their computational mechanisms are much more closely related to those in currently-favored theories than many researchers seem to realize, cf. [Kar98], and analyses of these older theories make for interesting comparisons with results derived for the more current theories.

The contributions made in this thesis (in order of appearance) are as follows:

- Systematic parameterized complexity analysis is defined in Section 2.1.3. This section also contains a discussion of what aspects of a problem can be usefully considered to be responsible for the polynomial-time intractability of a computational problem, and gives the first formal definition of a source of polynomial-time intractability.

- A new type of finite-state automaton called a contextual finite-state automaton is defined in Section 2.2.3 to assist complexity-theoretic analyses of the role of bounded constraint context-size in phonological processing.

- Chapter 4 gives the first systematic parameterized complexity analyses for Simplified Segmental Grammars, FST-based rule systems, the KIMMO system, Declarative Phonology, and Optimality Theory. As such, it extends and provides the first complete proofs of various results that were given (often without proof) in previously published parameterized analyses of Simplified Segmental Grammars [DFK+94] and Declarative Phonology and Optimality Theory [War96a, War96b]. These analyses are particularly notable in that they are the first to address the role of constraint context-size in the computational complexity of phonological processing.

  The results derived in the analyses described above are used to refute several conjectures made in the literature concerning the sources of polynomial-time intractability in Simplified Segmental Grammars, FST-based rule systems, and the KIMMO system. As the parametric reductions used in these analyses are also polynomial-time many-one reductions, the following results are also obtained:

  - Section 4.1.3 gives the first NP-hardness proof for the FINITE-STATE TRANSDUCER COMPOSITION problem and proofs in Section 4.3.2 extend this result to the restricted case of ϵ-free FST composition. These results in turn give the first NP-hardness proofs for the encoding and decoding problems associated with FST-based rule systems.
  - Section 4.5.2 gives the first NP-hardness proofs for the encoding and decoding problems associated with Declarative Phonology.
  - Section 4.6.2 gives the first NP-hardness proofs for the encoding and decoding problems associated with a formulation of Optimality Theory that is simpler than that proposed in [Eis97a]. These proofs in turn are used to give the first NP-hardness proof for the problem of learning constraint-rankings in Optimality Theory when surface forms instead of full forms are given as examples [Tes96, Tes97a, Tes97b].

  The results in Sections 4.3.2 and 4.4.2 can also be interpreted as the first parameterized analyses of the ϵ-free FST composition and intersection operations within the framework of Kay and Kaplan's finite-state calculus ([KK94]; see also [KCGS96, XFST]).

As is discussed in more detail in Section 4.7, the results described above suggest that the computational complexity of phonological processing depends not on such details of theory formulation as whether a theory uses rules or constraints or has one, two, or many levels of representation but rather on the structure of the representation-relations encoded in individual mechanisms and the internal structure of the representations used within that theory.

Ultimately, the main contributions of this thesis are the results derived in the systematic parameterized complexity analyses described above and the (hopefully, linguistically relevant) conclusions drawn from these results. The techniques used are by no means new – polynomial-time intractability maps have previously been constructed for various problems [BDFW95, BDF+95, Ces96, CW95, Eva98, Hal96, War96a] and many complexity-theoretic (albeit unparameterized) analyses of linguistic theories have been done over the last 25 years (see [BBR87, Ris93a] and references). What is different here is the size of the derived polynomial-time intractability maps and the use of such maps to compare and contrast several closely-related real-world computational problems, cf. the extensive parameterized analysis of Turing machine computation given in [Ces96]. The intractability maps given in this thesis are admittedly incomplete in that all possible results have not been derived relative to the sets of aspects considered here, there are other aspects of the problems that are not considered here at all, and no attempt has been made to derive the best possible non-polynomial algorithms where such algorithms have been shown to exist. However, this is consistent with the intent of this thesis, which is to show how systematic parameterized complexity analyses should be done and interpreted.

The intended audience of this thesis can be split into three groups:

1. Computational complexity theorists (ideally those who are disillusioned by the increasing abstractness of the field and would like to know where it contacts reality).

2. Algorithm designers (ideally those who wonder if those in the first group have done or ever will do anything that is relevant to them).

3. Computational linguists (ideally those who wonder if *any* computer scientists (particularly those in the first two groups) have done or ever will do anything that is relevant to them).

My hope is that systematic parameterized complexity analysis will form at least two kinds of bridges between these groups: a "little bridge" between computational complexity theorists and algorithm designers (by showing how cutting-edge complexity-theoretic techniques can both be initiated by and applied to practical problems in algorithm design) and a "big bridge" between computer scientists and computational linguists (by showing how techniques from computer science can be used both to derive better computer implementations for problems from linguistics and to suggest potentially useful restrictions on the computational power of the linguistic theories associated with these problems).

This thesis is organized as follows. Chapter 2, **Background**, consists of two sections which give overviews of parameterized complexity analysis and phonology. The introduction to this chapter also gives general notation that will be used throughout this thesis. Chapter 3, **Computational Analyses of Phonological Theories**, ties together the material presented in Chapter 2 by describing how computational analyses of phonological theories are done in practice and showing how certain flaws in such analyses that are introduced by using classical theories of computational complexity such as NP-completeness can be remedied by applying the conceptual framework and techniques of parameterized complexity analysis. Chapter 4, **A Systematic Parameterized Complexity Analysis of Phonological Processing in Rule- and Constraint-Based Formalisms**, devotes a section to the systematic parameterized complexity analysis of each of the five phonological theories mentioned above. Each section is broken into subsections which give an overview of a particular theory, the analysis of that theory, and a discussion of the implications of the results derived within that analysis and suggestions for future research. The organization of these discussions is somewhat involved and is described in more detail in the introduction to this chapter. This chapter concludes with a brief discussion of the implications of all results derived in this thesis for phonological processing in general. Chapter 5, **Conclusions**, summarizes the main contributions of this thesis and gives directions for future research common to all theories examined in Chapter 4. The thesis finishes up with two appendixes, the first giving the pages where the computational problems used in this thesis are first defined, and the latter giving the first published proofs of results for Simplified Segmental Grammars that were given without proof in [DFK+94].

Chapter 2

Background

This chapter is divided into two main sections, each of which gives an introductory overview of a topic addressed within this thesis. The first section is on parameterized complexity analysis, and consists of reviews of computational complexity theory and parameterized computational complexity theory and a discussion of how parameterized complexity analyses can be applied in a systematic manner to determine the sources of polynomial time intractability in computational problems. The second section is on phonology, and gives a brief introduction to the phenomena studied within phonology as well as the various types of representations and mechanisms by which these phenomena are described within phonological theories.

The following notation will be used throughout this thesis. The main objects of interest will be sets of objects, strings of symbols over some alphabet, and graphs. Some operators will have slightly different meanings depending on whether the operand is a set or a string.

- **Sets of Objects**: A **set** is a collection of objects. With respect to a set S, let $|S|$ denote the number of elements in S, i.e., the **size** of S, and ϕ denote the empty set, i.e., the set with no elements. Given a set A, let $\{x_1, x_2, \ldots\}$ denote the elements of A and $x \in A$ denote that x is an element of A. Given two sets A and B, let $A \cup B$ be the union of A and B, i.e., the set consisting of the elements in A and/or B, $A \times B$ be the Cartesian product of A and B, i.e., the set consisting of all ordered pairs (x, y) such that $x \in A$ and $y \in B$, and $A - B$ be the difference of A and B, i.e., the set consisting of all elements in A that are not also in B.

- **Strings of Symbols**: A **string** is a sequence of symbols drawn from some alphabet. With respect to a string x over an alphabet Σ, let $|x|$ denote the number of symbols in x, i.e., the **length** of x, and ϵ denote the empty string, i.e., the string of length 0. Given a string x, let $x_1x_2\cdots x_{|x|}$, $x_i \in \Sigma$ for $1 \leq i \leq |x|$, denote the concatenation of symbols from Σ that is x, and $xy = x_1x_2\ldots x_{|x|}y_1y_2\ldots y_{|y|}$, $x_i \in \Sigma$ for $1 \leq i \leq |x|$ and $y_i \in \Sigma$ for $1 \leq i \leq |y|$, be the string formed by concatenating the symbols in string y onto the right end of string x. Given a string $x = x_1x_2\ldots x_n$ and an integer $k \leq n$, the k-length **prefix of** x is the string $x_1, x_2\ldots x_k$ and the k-length **suffix of** x is the string $x_{n-(k-1)}x_{n-(k-2)}\ldots x_n$. Let Σ^n, $\Sigma^{\leq n}$, Σ^*, and Σ^+ denote the sets of all strings over Σ that have length n, length less than or equal to n, length greater than or equal to zero, and length strictly greater than zero, respectively. A **(formal) language** over an alphabet Σ is a subset of Σ^*. Occasionally, it will be useful to encode a set of strings into a single string without specifying the details; let $\langle\rangle : \overset{k}{\times}\, \Sigma^* \mapsto \Sigma^*$ be an invertible function that encodes a set of k given strings over Σ onto a single string over Σ.

- **Graphs**: A **graph** $G = (V, E)$ is a set of **vertices** V and a set of **edges** E that link pairs of vertices, i.e., $E \subseteq V \times V$. If an edge between vertices u and v has a direction from u to v, the edge is called a **directed edge** or **arc** and is written $\langle u, v\rangle$; otherwise, if the edge has no direction attached, it is called an **undirected edge** and is written (u, v). If all edges in a graph G are directed, it is called a **directed graph** and may alternatively be written as $G = (V, A)$. If all edges in G are undirected, G is an **undirected graph**. Unless otherwise noted, all graphs in this thesis are undirected. If two vertices u and v in G are joined by an edge (if G is undirected) or there is an arc from u to v (if G is directed), then u is **adjacent** to v in G; otherwise, u and v are not adjacent in G.

Let the set of integers be denoted by $\mathcal{N}$.

A final note is perhaps in order for readers who are not linguists. One of the most influential works in modern phonology is Chomsky and Halle's *The Sound Pattern of English* [CH68]. Following common practice in the linguistics literature, the acronym *SPE* will be used throughout this thesis to denote either the phonological theory or phonological rule-systems of the type described in [CH68].

2.1 Parameterized Complexity Analysis

2.1.1 Computational Complexity Theory

The two basic concepts of interest in computational complexity theory are problems and algorithms. Essentially, a problem is a question and an algorithm is a finite sequence of instructions for some computer which answers that question. Computational complexity theory establishes upper and lower bounds on how efficiently problems can be solved by algorithms, where "efficiency" is judged in terms of the amounts of computational resources, e.g., time or space, required by an algorithm to solve its associated problem. As noted by Rounds [Rou91, page 10], "The presuppositions of an already established theory, such as complexity theory, are perhaps the properties of the theory most easily ignored in making an application". Yet, it is precisely these presuppositions that we need to be aware of to fully appreciate both the advantages of parameterized complexity analysis and how such analyses can improve on previous computational analyses of phonological theories. With this in mind, the basics of computational complexity theory are reviewed in this section. The material given below is drawn largely from Sections 1.2, 2.1, and 5.1 of [GJ79]. Readers wishing more detailed treatments are referred to [GJ79, HU79, Pap94].

Let us first look at computational problems. A **computational problem** is a question to be answered, typically containing several variables whose values are unspecified. An **instance** of a problem is created by specifying particular values for its variables. A problem is described by specifying both its instances and the nature of solutions for those instances. Problems typically have two levels of description – an abstract description in terms of the general structure and type of instances and solutions, and a concrete description in terms of the formal objects, e.g., languages and string relations, onto which instances and solutions are mapped for manipulation by computer models and subsequent analysis. There are many types of problems. Two of the most common types are described below.

Definition 2.1.1 *[GJ79, Section 2.1] A* **decision problem** Π *is a set* D_Π *of instances and a set* $Y_\Pi \subseteq D_\Pi$ *of yes-instances. A decision problem is described informally by specifying:*

1. *A generic instance in terms of its variables.*
2. *A yes-no question stated in terms of the generic instance.*

A decision problem Π *is formally described by a language* $L[\Pi, e] \subseteq \Sigma^+$ *for some alphabet* Σ*, where* $e : Y_\Pi \mapsto \Sigma^+$ *is an instance encoding function and* $L = \{x \mid \exists I \in Y_\Pi$ *such that* $e(I) = x\}$*.*

Definition 2.1.2 *[GJ79, Section 5.1] A* **search problem** Π *is a set* D_Π *of instances and a set* S_Π *of solutions such that for each* $I \in D_\Pi$*, there is an associated set* $S_\Pi[I] \subseteq S_\Pi$ *of solutions for* I*. A search problem is described informally by specifying:*

1. *A generic instance in terms of its variables.*
2. *The properties that must be satisfied by any set of solutions associated with an instance created from the generic instance.*

A search problem Π *is formally described by a string relation* $R[\Pi, e] \subseteq \Sigma^+ \times \Sigma^+$ *for some alphabet* Σ*, where* $e = (e_D, e_S)$*,* $e_D : D_\Pi \mapsto \Sigma^+$ *and* $e_S : S_\Pi \mapsto \Sigma^+$*, is a pair of instance and solution encoding functions and* $R[\Pi, e] = \{(x, y) \mid \exists I \in Y_\Pi\ \exists S \in S_\Pi[I]$ *such that* $e_D(I) = x$ *and* $e_S(S) = y\}$*.*

The instance and solution encoding functions should be "reasonable" in the sense that they are concise and decodable – that is, they create encodings that are not artificially larger than the information in the given instance warrants and that can be decoded easily to extract any of the variables specified in the generic instance [GJ79, p. 21].

Example 2.1.3 Consider the problem of finding vertex covers associated with a given graph, i.e., a subset of the vertices of a a given graph such that each edge in the graph has at least one endpoint in the subset. Two possible informal descriptions of decision and search versions of this problem are given below:

VERTEX COVER (Decision) [GJ79, Problem GT1]
Instance: A graph $G = (V, E)$, a positive integer k.
Question: Is there a vertex cover of G of size at most k, i.e., a set of vertices $V' \subseteq V$, $|V'| \leq k$, such that for every edge $(x, y) \in E$, either x or y is in V'?

VERTEX COVER (Search)
Instance: A graph $G = (V, E)$, a positive integer k.
Solution: Any vertex cover of G of size $\leq k$.

In formal descriptions of theses problems, the input graphs could be specified as $|V| \times |V|$ edge-adjacency matrices and instances could be encoded as strings of the form $\langle |V|, G, k\rangle$ with length $\log_2 |V| + |V|^2 + \log_2 k$ bits. In the case of the search problem, if a numbering from 1 to $|V|$ on the vertices is assumed, each solution is a subset of $\{1, \ldots, |V|\}$ and can be represented by a string of $|V|$ bits. ∎

Informal descriptions are useful to the extent that they are faithful to our mental intuitions about what problems are and how they behave, and formal descriptions are useful to the extent that they both mirror informal descriptions and are amenable to mathematical analysis. In short, formal descriptions make reasoning about problems rigorous and informal descriptions ensure that such reasoning remains relevant; hence, each has a role to play in computational complexity theory.

Given the above, an **algorithm** A for a problem Π is a finite sequence of instructions for some computer which **solves** Π, i.e., for any given instance of Π, A computes the appropriate solution. What constitutes an appropriate solution depends on the type of the problem. For example, the solution to a decision problem is "Yes" if the given instance is in Y_Π and "No" otherwise, and the solution to a search problem is "No" if the solution-set associated with the given instance is empty and an arbitrary element of this solution-set otherwise. Note that as we are now discussing problems and algorithms in relation to formal computer models, the instances and solutions manipulated by algorithms are the string-encodings of the abstract instances and solutions discussed above. Readers interested in the details of how problems and algorithms are mapped onto formal computer models are referred to Chapters 2 and 5 of [GJ79].

We can now talk about algorithm efficiency. Typical computational resources of interest are time and space, which correspond to the number of instructions executed or the amount of memory used by the algorithm when it is implemented on some standard type of computer, e.g., a deterministic Turing machine (for detailed descriptions of the various kinds of Turing machines, see [GJ79, HU79, LP81]). For some resource R and problem Π, let $R_A : D_\Pi \mapsto \mathcal{N}$ be be the function that gives the amount of resource R that is used by algorithm A to solve a given instance of Π. The resource-usage behavior of an algorithm over all possible instances of its associated problem is typically stated in terms of a function of instance size that summarizes this behavior in some useful manner. The creation of such functions has three steps:

1. Define an instance-length function such that each instance of the problem of interest can be assigned a positive integer size. Let the size of instance I be denoted by $|I|$.
2. Define a "raw" resource-usage function that summarizes the resource-usage behavior of A for each possible instance size. Let $R_A^n = \{R_A(I) \mid I$ is an instance of the problem solved by A and $|I| = n\}$ be the R-requirements of algorithm A for all instances of size n. For each instance-size n, choose either one element of or some function of R_A^n to represent R_A^n. Several popular ways of doing this are:
   - The highest value in R_A^n (**worst-case**).
   - The lowest value in R_A^n (**best-case**).
   - The average value of R_A^n relative to some probability distribution on instances of size n (**average-case**).

   Let $S_A : \mathcal{N} \mapsto \mathcal{N}$ be the function that gives this chosen value for $n > 0$.
3. "Smooth" the raw resource-usage function $S_A(n)$ via a function $C_A : \mathcal{N} \mapsto \mathcal{N}$ that asymptotically bounds $S_A(n)$ in some fashion. Several standard types of asymptotic bounding functions g on a function f are:
   - **Asymptotic upper bound**: $f \in O(g)$ if there exists a constant c and $n_0 \geq 0$ such that for all $n > n_0$, $f(n) < c \cdot g(n)$.
   - **Asymptotic lower bound**: $f \in \Omega(g)$ if there exists a constant c and $n_0 > 0$ such that for all $n > n_0$, $f(n) > c \cdot g(n)$.
   - **Asymptotic tight bound**: $f \in \Theta(g)$ if $f \in O(g)$ and $f \in \Omega(g)$.

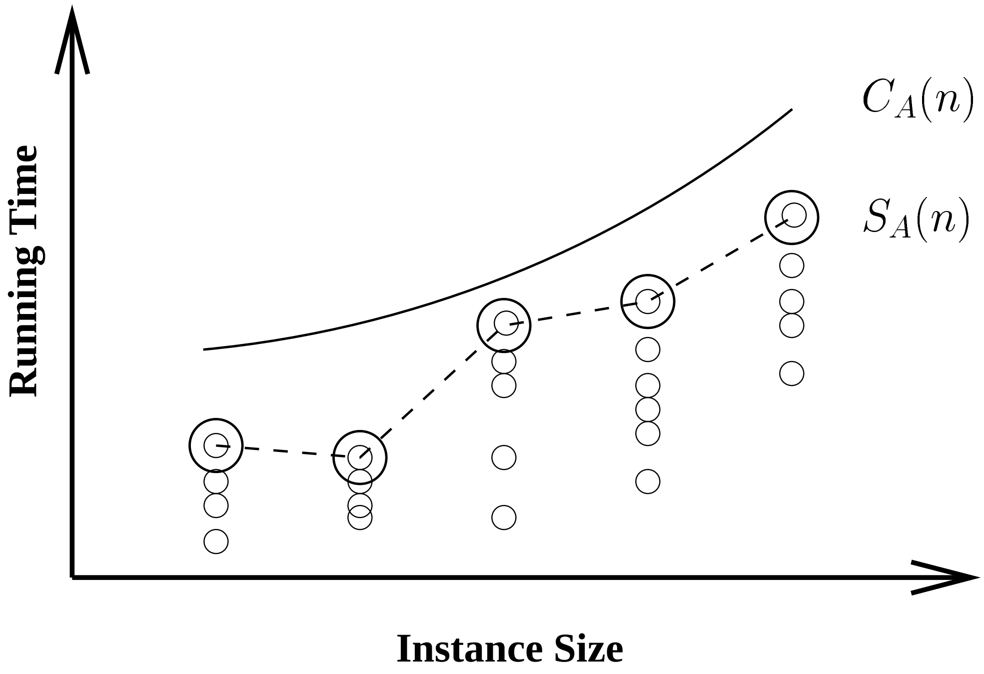


Figure 2.1: Algorithm Resource-Usage Complexity Functions. Let A be the given algorithm for a problem Π with instance-set D_Π. In the figure above, each of the small circles denote the resource-usage value $R_A(I)$ of A for a particular instance $I \in D_\Pi$, the large circles denote the largest resource-usage value in R_A^n for each instance size n, the dotted line denotes the "raw" worst-case resource-usage function $S_A(n)$, and the solid line denotes the asymptotic upper bound worst-case resource-usage function $C_A(n)$ which "smoothes" $S_A(n)$. See main text for further explanation of terms.

The function $C_A(n)$ is called a R-**complexity function** for algorithm A. The process of creating such functions is illustrated in Figure 2.1. The complexity functions most commonly used in the literature (and hence in this thesis) choose resource R to be time relative to a deterministic Turing machine and summarize $R_A(I)$ in terms of asymptotic upper bounds on the worst case; hence, they are known as asymptotic worst-case time complexity functions. Given that an algorithm's asymptotic worst-case time complexity function is $f(n)$, the algorithm is said to be a $f(n)$ time algorithm, and its associated problem is said to be solvable in $f(n)$ time. An algorithm is **efficient** if its complexity function satisfies some criterion, e.g., the complexity function is a polynomial of the instance size. A problem is **tractable** if it has an efficient algorithm; otherwise, the problem is said to be **intractable**. As there are many possible criteria which can be used to define efficiency, there are many possible types of tractability and intractability.

Computational complexity theory is concerned with establishing not only what problems can be solved efficiently but also what problems *cannot* be solved efficiently. The former is done by giving an efficient algorithm for the problem, and the latter by defining the following four components:

1. A universe $\mathcal{U}$ of problems.

2. A class $\mathcal{T} \subset \mathcal{U}$ of tractable problems.

3. A reducibility $\propto$ between pairs of problems in $\mathcal{U}$. A **reduction** from a problem Π to a problem Π' (written $\Pi \; \alpha \; \Pi'$) is essentially an algorithm that can use any algorithm for Π' to solve Π. A **reducibility** is a set of reductions that satisfies certain properties. The reducibilities of interest here must satisfy the following two properties:

   (a) **Transitivity**: For all problems Π, Π', $\Pi'' \in \mathcal{U}$, if $\Pi \propto \Pi'$ and $\Pi' \propto \Pi''$ then $\Pi \propto \Pi''$.

   (b) **Preservation of Tractability**: For all problems Π, $\Pi' \in \mathcal{U}$, if $\Pi \propto \Pi'$ and $\Pi' \in \mathcal{T}$ then $\Pi \in \mathcal{T}$.

4. One or more classes of problems $\mathcal{C} \subset \mathcal{U}$ such that $\mathcal{T} \subset \mathcal{C}$.

Given a class of problems $\mathcal{K} \subset \mathcal{U}$, a problem Π is **$\mathcal{K}$-hard** if for all problems $\Pi' \in \mathcal{K}$, $\Pi' \propto \Pi$; if Π is also in $\mathcal{K}$, then Π is **$\mathcal{K}$-complete**. Informally, the reducibility $\propto$ establishes the computational difficulty of problems relative to each other – that is, if $\Pi \propto \Pi'$ then Π' is at least as computationally difficult as Π. Hence, $\mathcal{K}$-complete problems are the most computationally difficult problems in $\mathcal{K}$, and $\mathcal{K}$-hard problems are at least as computationally difficult as the most computationally difficult problems in $\mathcal{K}$. These notions are significant because if a given problem Π is $\mathcal{C}$-hard relative to $\propto$ for any class $\mathcal{C}$ and reducibility $\propto$ as defined above then Π is not in $\mathcal{T}$ and hence does not have an efficient algorithm (see Figure 2.2).

In practice, it is often very difficult to prove that a problem is hard for classes $\mathcal{C}$ as defined above. In these cases, it is convenient to use slightly weaker classes $\mathcal{C}'$ such that it is strongly conjectured (but not proven) that $\mathcal{T} \subset \mathcal{C}'$. One can still derive hardness and completeness results relative to such classes; however, the validity of the conclusion that a $\mathcal{C}'$-hard problem does not have an efficient algorithm now depends on the strength of the conjecture that $\mathcal{T} \neq \mathcal{C}'$. Though such conjecture-dependent results are weaker than actual intractability results, such results can function as proofs of intractability for all practical purposes if the strength of the conjectures can be tied to practical experience in developing (or rather, failing to develop) efficient algorithms for $\mathcal{C}'$-hard problems.

Such has been the case for polynomial-time intractability. Polynomial time algorithms are useful in practice because the values of polynomial functions grow much more slowly than the values of non-polynomial functions as instance size goes to infinity, and hence algorithms that run in polynomial time can solve much larger instances in a given period of time than those that run in non-polynomial time (for a graphic illustration of this, see [GJ79, Figure 1.2]). A number of classes of decision problems that properly include the class of decision problems that are solvable in polynomial time were developed in the 1960's, e.g., $EXPTIME$. However, very few problems of interest have been shown to be hard for these classes. This motivated the development of what is arguably the most famous theory of computational complexity to date, namely, the theory of NP-completeness (see [GJ79] and references). Relative to the scheme above, the components of this theory are:

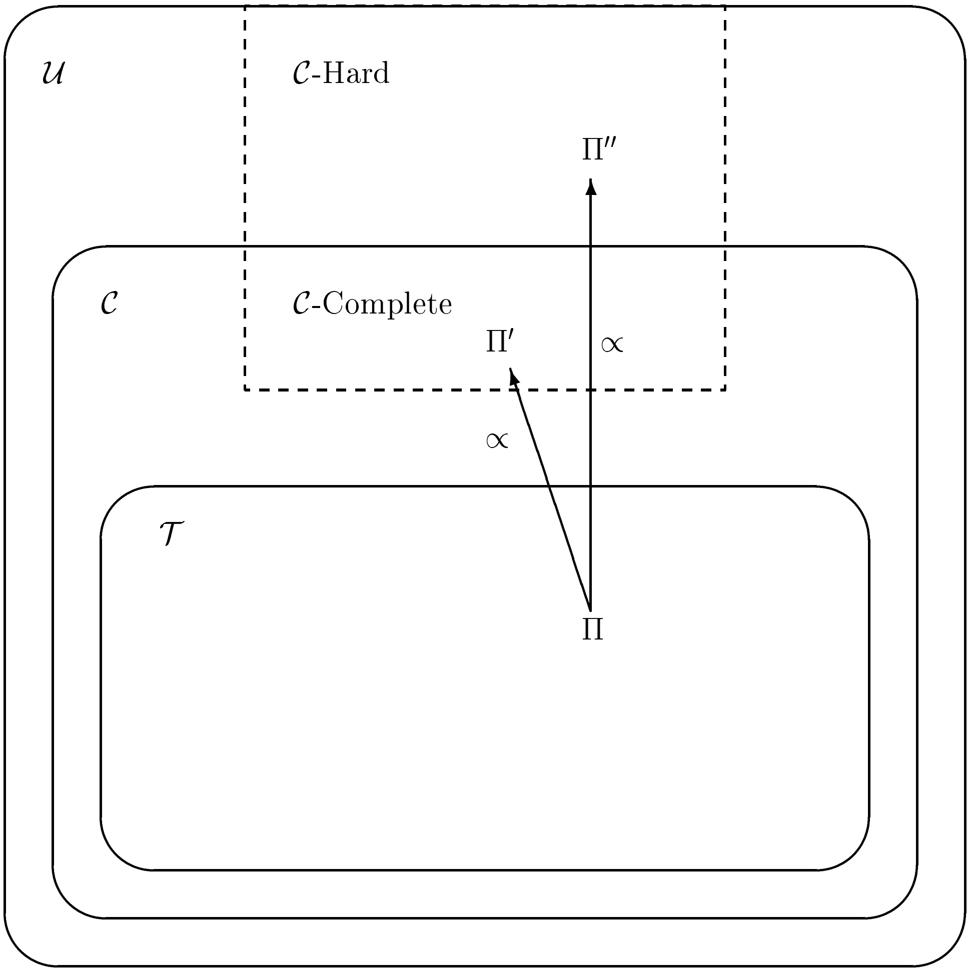


Figure 2.2: The Relationship Between Hardness and Completeness Results in Computational Complexity Theory. See main text for explanation of symbols.

$\mathcal{U}$: The universe of decision problems.

$\mathcal{T}$: The class **P** of decision problems that can be solved in polynomial time by an algorithm running on a deterministic Turing machine.

$\propto$: The polynomial time many-one reducibility $\leq_m$, i.e., given two decision problems Π and Π', $\Pi \leq_m \Pi'$ if there exists an algorithm that, given an instance x of Π, constructs an instance x' of Π' in time polynomial in $|x|$ such that $x \in Y_\Pi$ if and only if $x' \in Y_{\Pi'}$.

$\mathcal{C}$: The class **NP** of decision problems that can be solved in polynomial time by an algorithm running on a nondeterministic Turing machine. Such an algorithm essentially has access to some polynomial number of bits, and is said to solve its associated problem if the execution of that algorithm relative to at least one of the possible choices of values for these bits solves the problem in a polynomial number of steps.

As any deterministic polynomial time algorithm can be rephrased as a nondeterministic polynomial time algorithm that ignores the values of the nondeterministic bits, $P \subseteq NP$. If a decision problem can be shown to be NP-hard then it is not in P (and hence does not have a polynomial time algorithm) modulo the strength of the conjecture that $P \neq NP$. Many theorems (which themselves are stated relative to other conjectures) support this conjecture (see [Sip92] and references); however, the best evidence to date that $P \neq NP$ is that in almost 50 years of algorithm research, no one has yet produced a polynomial time algorithm for a decision problem that is NP-hard. Hence, it is a widely accepted working hypothesis that $P \neq NP$.

When interpreting NP-hardness results, it is very important to remember that an NP-hardness result for a problem Π only implies that Π is polynomial-time intractable if $P \neq NP$. However, given the widespread confidence in this working hypothesis, an NP-hardness result can, for all practical purposes, be considered equivalent to a proof of polynomial-time intractability, and will be treated as such in the remainder of this thesis.

The focus above on decision problems may seem irrelevant as it is search problems that are typically of interest. However, the theory of NP-completeness (as are many theories of computational complexity) is phrased in terms of decision problems for two reasons:

1. The formal model underlying decision problems, i.e., formal languages, is much easier to manipulate and analyze than the formal model underlying search problems, i.e., string relations.

2. As efficient algorithms for search problems can be used to solve appropriately-defined decision problems efficiently, intractability results for such decision problems imply (and can be used to prove) the intractability of their associated search problems. For instance, if it can be shown that any polynomial time algorithm for a search problem of interest can be used to solve an appropriately defined decision problem in polynomial time and that this decision problem is NP-hard (and hence not solvable in polynomial time unless $P = NP$), then the search problem cannot be solved in polynomial time unless $P = NP$.

Example 2.1.4 Such a relationship holds for the vertex cover problems defined in Example 2.1.3. Any algorithm for the search version of VERTEX COVER also solves the decision version of VERTEX COVER, and as the decision version is NP-complete (see [GJ79, Problem GT1] and references), the search version of VERTEX COVER does not have a polynomial time algorithm unless $P = NP$. ▮

Ideally, a complexity-theoretic analysis of a problem is not just a one-sided quest for either algorithms or hardness results. Rather, it is an ongoing dialogue in which both types of results are used to fully characterize the problem by showing which restrictions make that problem tractable and which don't [GJ79, Section 4.1].

2.1.2 Parameterized Computational Complexity Theory

The theory of NP-completeness described in the previous section was developed to show which problems probably do not have polynomial time algorithms. Since the inception of this theory in 1971 with Cook's proof that BOOLEAN CNF FORMULA SATISFIABILITY is NP-complete [Coo71], thousands of other problems have been shown to be NP-hard and NP-complete. Though it is nice to know that such problems do not have polynomial time algorithms unless $P = NP$, the inconvenient fact remains that these problems (especially those with real-world applications) must still be solved.

How does solve NP-hard problems in practice? To date, the two most popular approaches within computer science have been:

(1) Invoke some type of non-polynomial time "brute force" optimal-solution technique, e.g., dynamic programming, branch-and-bound search, mathematical programming.

(2) Invoke some type of polynomial time approximate-solution algorithm, e.g., bounded-cost approximation schemes, randomized algorithms, simulated annealing, heuristics.

When the instances to be solved are large, approach (1) may not be feasible, thus forcing the invocation of approach (2). Indeed, fast approximation algorithms seem the most popular method for dealing with NP-hard problems at this time. However, there is another approach (which is arguably an informed version of (1)):

(3) Invoke a non-polynomial time algorithm that is effectively polynomial time because its non-polynomial time complexity is purely a function of some set of aspects of the problem that are of bounded size or value in instances of that problem encountered in practice.

Serious discussion of this alternative requires the following definitions.

Definition 2.1.5 *Given a decision or search problem* Π*, an* **aspect** a *of* Π *is a function* $a : D_\Pi \mapsto \Sigma^+$ *for some alphabet* Σ.

Definition 2.1.6 *Given a decision or search problem* Π*, aspects* a *and* a' *of* Π*, and a non-polynomial time algorithm* P *for* Π*,* **the non-polynomial time complexity of P is purely a function of** a *if the time complexity function of* P *can be written as* $f(a)|a'|^c$*, where* $f : \Sigma^+ \mapsto \mathcal{N}$ *is an arbitrary function and* c *is a constant independent of* a *and* a'.

An aspect of a problem is essentially some characteristic that can be derived from instances of that problem. Convenient aspects of a problem are the variables in a generic instance of that problem or various numerical characteristics of those variables. For example, some aspects of the vertex cover problems defined in Example 2.1.3 are the given graph (G), the number of vertices in that graph ($|V|$), the given bound on the size of the vertex cover (k), or the maximum degree of any vertex in G. Each problem has many aspects, and some problems have aspects of bounded size or value that are potentially useful in the sense of (3) above.

Example 2.1.7 Consider the following decision problem from robotics:

3-D ROBOT MOTION PLANNING
Instance: A robot composed of linked polyhedra, an environment composed of some set of polyhedral obstacles, and initial and final positions p_I and p_F of the robot within the environment.
Question: Is there a sequence of motions of the robot, i.e., a sequence of translations and rotations of the linked polyhedra making up the robot, that move it from p_I to p_F without intersecting any of the obstacles?

This problem is known to be $PSPACE$-hard [Rei87] and hence does not have a polynomial time algorithm unless $P = PSPACE$. Though certain aspects such as the number of or description-complexity of the environmental obstacles tend to be very large, other aspects do not, e.g., the number of joints in the robot is less than 7 for commercially-available robot arms and less than 20 for robot "hands" (see [CW95] and references). An algorithm for this problem whose non-polynomial time complexity is purely a function of the number of joints of the given robot might be useful in practice. ∎

There seem to be many other real-world problems that have such bounded aspects [BDF+95, DFS98], and anecdotal evidence suggests that non-polynomial time algorithms that exploit these aspects play an important role in solving NP-hard problems that occur in commercial and industrial applications [DFS98].

The attractiveness of non-polynomial time algorithms that exploit aspects of bounded size or value immediately suggests two questions:

(1) How does one determine if a problem of interest has such "reasonable" non-polynomial time algorithms?

(2) Relative to which aspects of that problem do such algorithms exist?

In order to answer these questions, one must be able to answer the following question:

(3) Given a problem and a set of aspects of that problem, does there exist an algorithm for that problem whose non-polynomial time complexity is purely a function of those aspects?

This question can be partially addressed within classical theories of computational complexity like that for NP-completeness. One can try using conventional algorithm-design strategies to find such an algorithm. Alternatively, one can show that such an algorithm does not exist by establishing the polynomial-time intractability of the version of that problem in which the values of those aspects are fixed (in the case of numerical aspects, to small constants).

Example 2.1.8 Consider the 3-D robot motion planning problem defined in Example 2.1.7. If one could establish the $PSPACE$-hardness of the version of this problem in which the number of joints in the given robot is fixed at some constant c, then the 3-D robot motion planning problem would not have an algorithm whose non-polynomial time complexity is purely a function of the number of joints in the robot unless $P = PSPACE$ (as the polynomial-time algorithm created by fixing the number of joints to c in any such non-polynomial time algorithm could be used in conjunction with the $PSPACE$-hardness reduction for the 3-D robot motion planning problem to solve any problem in $PSPACE$ in polynomial time). ▮

Such proofs are part of the strategy advocated in Garey and Johnson's discussion on how to use algorithms and NP-hardness results to map a problem's "frontier of tractability" [GJ79, Sections 4.1 and 4.3]. However, it is not obvious that one can, for every possible aspect of a problem, always either give the required type of non-polynomial time algorithm relative to that aspect or establish the polynomial-time intractability of that problem relative to some fixed value of that aspect. A large part of the difficulty here stems from one of the idealizations underlying classical theories of computational complexity – namely, that instances are indivisible and are characterized in analyses by a single instance size value. As long as it is impossible to extract aspects of an instance and consider their effects in isolation on the time complexity of an algorithm for a problem, attempts to analyze the effects of individual aspects within computational complexity theory will necessarily be indirect and incomplete.

The theory of parameterized computational complexity [DF95a, DF95b, DF99] is the first theory of computational complexity to provide explicit and elegant mechanisms for analyzing the effects of individual aspects on problem complexity. This theory is founded on two basic insights:

1. Problem instances can be split into two distinct parts.
2. The effects of individual aspects on problem complexity can be isolated by redefining problem tractability such that the time complexity function of an algorithm for a problem need only be efficient relative to the aspects contained in one of these parts.

The details by which these insights are fleshed out into a theory of computational complexity are sketched below. Consider first the conception of a problem within this theory and how individual aspects of problem instances are isolated.

Definition 2.1.9 *Given a decision problem* Π, *a* **parameter** p *of* Π *is a function* $p : D_\Pi \mapsto \Sigma^+$ *for some alphabet* Σ.

A parameter is a function which extracts a particular aspect or set of aspects of a problem from instances of that problem; it can also be considered as that set of aspects. As such, a parameter is both a mechanism for isolating aspects of a problem and the "container" in which these aspects are packaged for subsequent manipulation. As defined here, a parameter is formally indistinguishable from an aspect; however, it informally has an internal structure that is not present in an aspect.

Definition 2.1.10 *A* **parameterized problem** Π *is a set* D_Π *of instances, a set* Y_Π *of yes-instances, and a parameter* p. *A parameterized problem is described informally by specifying:*

1. *A generic instance in terms of its variables.*
2. *The aspects of an instance that comprise the parameter.*
3. *A yes-no question stated in terms of the generic instance.*

A parameterized problem Π *is formally described by a string relation* $R[\Pi, e] \subseteq \Sigma^+ \times \Sigma^+$ *for some alphabet* Σ, *where* $e : D_\Pi \mapsto \Sigma^+$ *is an instance encoding function and* $R[\Pi, e] = \{(x, y) \mid \exists I \in Y_\Pi$ *such that* $e_D(I) = x$ *and* $p(I) = y\}$. *By analogy with decision problems,* $R[\Pi, e]$ *will be the* parameterized language *associated by* e *with* Π.

Given an instance (x, y) of a parameterized problem, x is called the **main part** and y is called the **parameter**.

Example 2.1.11 Two possible parameterized vertex cover problems are:

VERTEX COVER I
Instance: A graph $G = (V, E)$, a positive integer k.
Parameter: G
Question: Is there a vertex cover of G of size at most k?

VERTEX COVER II
Instance: A graph $G = (V, E)$, a positive integer k.
Parameter: k
Question: Is there a vertex cover of G of size at most k?

In a formal description, the input graphs could be specified as $|V| \times |V|$ edge-adjacency matrices and instances could be encoded as string-pairs of the form $(\langle |V|, G, k\rangle, \langle G\rangle)$ and $(\langle |V|, G, k\rangle, \langle k\rangle)$, respectively. ▮

Parameterized problems will often be equated with their parameterized languages and an instance (x, y) of a parameterized problem Π will be said to be either in or not in that problem's parameterized language, e.g., $(x, y) \in \Pi$. Note that parameterized problems answer yes-no questions, and are thus a type of decision problem. In the remainder of this thesis, unless otherwise specified, the term "decision problem" will denote an unparameterized decision problem.

Given this explicitly divisible conception of a problem instance, we can now define a more useful version of polynomial time tractability under which a problem's non-polynomial time complexity is purely a function of aspects selected by the parameter.

Definition 2.1.12 *A parameterized problem* Π *is* **fixed-parameter tractable** *if there exists algorithm* A *to determine if instance* (x, y) *is in* Π *in time* $f(y) \cdot |x|^{\alpha}$*, where* $f : \Sigma^{+} \mapsto \mathcal{N}$ *is an arbitrary function and* α *is a constant independent of* x *and* y*. The algorithm* A *is called an* **FPT algorithm** *for* Π.

Definition 2.1.13 *A parameterized problem* Π *belongs to the class* **FPT** *if* Π *is fixed-parameter tractable.*

There are a variety of techniques for deriving FPT algorithms for parameterized problems (see [DF95c, DF99] and references). One can establish that a parameterized problem is fixed-parameter intractable (modulo certain conjectures) via the following additional definitions. The requisite reducibility between parameterized problems is given below.

Definition 2.1.14 *A* **parametric (many-one) reduction** *from a parameterized problem* Π *to a parameterized problem* Π' *is an algorithm* M *that transforms an instance* (x, y) *of* Π *into an instance* (x', y') *of* Π' *such that:*

1. *M runs in time $f(y)|x|^{\alpha}$ time for an arbitrary function f and a constant α independent of both x and y.*
2. *$y' = g(y)$ for some arbitrary function g.*
3. *$(x, y) \in \Pi$ if and only if $(x', y') \in \Pi'$.*

If such a parametric reduction exists between Π *and* Π' *then* Π **parametrically (many-one) reduces to** Π.

Lemma 2.1.15 (Transitivity) *Given parameterized problems* Π, Π', *and* Π'', *if* Π *parametrically reduces to* Π' *and* Π' *parametrically reduces to* Π'' *then* Π *parametrically reduces to* Π''.

Lemma 2.1.16 (Preservation of (Fixed-Parameter) Tractability) *Given parameterized problems* Π *and* Π', *if* Π *parametrically reduces to* Π' *and* Π' *is fixed-parameter tractable then* Π *is fixed-parameter tractable.*

The following definitions give the parameterized analogs of the class NP in the theory of NP-completeness. These classes are, for the most part, based on a series of successively more powerful solution-checking circuits in which solutions are encoded as input vectors to these circuits and parameters are encoded in the weights (see below) of these input vectors.

Definition 2.1.17 *An* **(unbounded fan-in) Boolean circuit** α_n *with input* $x = x_1 x_2 \cdots x_n$ *of length* n *is a directed acyclic graph. The nodes of fan-in 0 are called* **input nodes** *and are labeled from the set* $\{0, 1, x_1, \overline{x}_1, x_2, \overline{x}_2, \ldots, x_n, \overline{x}_n\}$*. The nodes of fan-in greater than 0 are called* **gates** *and are labeled either* AND *or* OR*. A special node is designated the* **output** *node. The* **size** *is the number of gates and the* **depth** *is the maximum distance from an input node to the output node.*

Definition 2.1.18 *The* **weight** *of a binary string is the number of 1's in that string.*

Definition 2.1.19 *[BS90] An unbounded fan-in Boolean circuit* α *is a* Π_t^h**-circuit** *if* α *has depth at most* $t+1$ *with an* AND *gate at the output and gates of fan-in at most* h *at the input level.*

Definition 2.1.20 *A parameterized problem* Π *belongs to the class* **W[t]**, $t \geq 1$, *if* Π *parametrically reduces to the parameterized problem* $\mathrm{WCS}(t,h) = \{\langle \alpha, k \rangle \mid$ *The* Π_t^h*-circuit* α *accepts an input of weight* $k\}$ *for some constant* h.

Definition 2.1.21 *A parameterized problem* Π *belongs to the class* **W[P]** *if* Π *parametrically reduces to the parameterized problem* WCS$= \{\langle \alpha, k \rangle \mid$ *The Boolean circuit* α *accepts an input of weight* $k\}$.

Definition 2.1.22 *A parameterized problem* Π *belongs to the class* **XP** *if there exists an algorithm* A *to determine if instance* (x, y) *is in* Π *in time* $f(y) \cdot |x|^{g(y)}$, *where* $f : \Sigma^+ \mapsto \mathcal{N}$ *and* $g : \Sigma^+ \mapsto \mathcal{N}$ *are arbitrary functions.*

More parameterized classes are defined in [DF99], e.g., $W[SAT]$. All such classes and FPT comprise what is known as the **W hierarchy**. By the definitions given above, the classes of the W hierarchy are related as follows:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P] \subseteq \cdots \subseteq XP$$

Relative to the scheme given in the previous section, the four components necessary for establishing the fixed-parameter intractability (modulo certain conjectures) of a parameterized problem are thus:

$\mathcal{U}$: The universe of parameterized problems.

$\mathcal{T}$: The class FPT of fixed-parameter tractable problem.

$\propto$: The parameterized reducibility between parameterized problems.

$\mathcal{C}$: Any of the classes in the W hierarchy above FPT.

If a parameterized problem can be shown to be $\mathcal{C}$-hard for any class $\mathcal{C}$ in the W hierarchy above FPT, then that problem is not in FPT (and hence does not have a FPT algorithm) modulo the strength of the conjecture that $FPT \neq \mathcal{C}$. At present, XP is the only class known to properly contain FPT, and it is conjectured that all other inclusions in the W hierarchy are proper, i.e., $FPT \subset W[1] \subset W[2] \subset \ldots W[P] \subset XP$. There are various theorems (which themselves are stated relative to other conjectures) which support this conjecture [CCDF94, DF93, DF95a]; however, the best evidence to date that the inclusions in the W hierarchy are proper is that no-one has yet produced an FPT algorithm for a parameterized problem that is hard for any class in the W hierarchy above FPT. Hence, though the evidence supporting the separateness of the levels of the W hierarchy is not nearly as extensive as that supporting the conjecture that $P \neq NP$, the conjecture that all inclusions in the W hierarchy are proper seems an acceptable (if not yet widely accepted) working hypothesis.

When interpreting parameterized hardness results, it is very important to remember that a $\mathcal{C}$-hardness result for a parameterized problem Π for some class $\mathcal{C}$ above FPT in the W hierarchy only implies that Π is fixed-parameter intractable if $FPT \neq \mathcal{C}$. However, given the confidence in the working hypothesis that all inclusions in the W hierarchy are proper, such a $\mathcal{C}$-hardness result can, for all practical purposes, be considered equivalent to a proof of fixed-parameter intractability, and will be treated as such in the remainder of this thesis.

Parameterized results are intrinsically interesting in that they populate the various levels of the W hierarchy with parameterized problems; however, such results also have implications for decision problems. It follows from the definition of a parameterized problem that each such problem Π based on sets D_Π and Y_Π and parameter p has an associated decision problem based on sets D_Π and Y_Π; moreover, as each decision problem has many possible aspects and parameters that encode various subsets of these aspects, each decision problem has an associated family of parameterized problems. This should perhaps be reflected in the way problems are denoted; hence, given a decision problem Π with a parameter p, let $\langle p \rangle$-Π denote the parameterized problem associated with Π that is based on parameter p.

Example 2.1.23 Given parameters G and k for the vertex cover problems defined in Example 2.1.3 in the previous section, one might speak of the decision problem VERTEX COVER and its associated parameterized problems $\langle G \rangle$-VERTEX COVER and $\langle k \rangle$-VERTEX COVER (which correspond to the parameterized problems VERTEX COVER I and VERTEX COVER II defined in Example 2.1.11). ∎

One consequence of the close relationship between decision and parameterized problems is that in certain cases, properties of and algorithms for one type of problem also apply to the other. For instance, it is obvious that any algorithm for a decision problem will also solve its associated parameterized problems, and vice versa. It is also easy to prove that under certain conditions, a reduction that holds between a pair of decision problems will also hold between a pair of their associated parameterized problems, and vice versa.

Lemma 2.1.24 *Given decision problems* Π *and* Π' *with parameters* p *and* p'*, respectively, if* $\langle p \rangle$-Π *parametrically reduces to* $\langle p' \rangle$-Π' *such that for an instance* (x, y) *of* $\langle p \rangle$-Π *the reduction runs in time polynomial in* x *and* y*, then* $\Pi \leq_m \Pi'$.

Lemma 2.1.25 *Given decision problems* Π *and* Π' *with parameters* p *and* p'*, respectively, if* $\Pi \leq_m \Pi'$ *such that* $p' = g(p)$ *for an arbitrary function* g*, then* $\langle p \rangle$-Π *parametrically reduces to* $\langle p' \rangle$-Π'.

Example 2.1.26 Consider the following decision problems that are very closely related to the VERTEX COVER decision problem:

CLIQUE [GJ79, Problem GT19]
Instance: A graph $G = (V, E)$, a positive integer k.
Question: Is there a clique in G of size at least k, i.e., a set of vertices $V' \subseteq V$, $|V'| \geq k$, such that for every pair of vertices $x, y \in V'$, $(x, y) \in E$?

INDEPENDENT SET [GJ79, Problem GT20]
Instance: A graph $G = (V, E)$, a positive integer k.
Question: Is there an independent set in G of size at least k, i.e., a set of vertices $V' \subseteq V$, $|V'| \leq k$, such that for every pair of vertices $x, y \in V'$, $(x, y) \notin E$?

For any graph $G = (V, E)$ and subset $V' \subseteq V$ of the vertices of G, the following statements are equivalent [GJ79, Lemma 3.1]:

1. V' is a vertex cover for G.
2. $V - V'$ is an independent set for G.
3. $V - V'$ is a clique for the *complement* G^c of G, where $G^c = (V, E^c)$ with $E^c = \{(u, v) \mid u, v \in V \text{ and } (u, v) \notin E\}$.

This suggests natural polynomial time many-one reductions between each pair of problems in the set {CLIQUE, INDEPENDENT SET, VERTEX COVER}, e.g.,

- INDEPENDENT SET $\leq_m$ CLIQUE by the polynomial time algorithm that transforms an instance $\langle G, k \rangle$ of INDEPENDENT SET into an instance $\langle G', k' \rangle$ of CLIQUE where $G' = G^c$ and $k' = k$.
- VERTEX COVER $\leq_m$ CLIQUE by the polynomial time algorithm that transforms an instance $\langle G, k \rangle$ of VERTEX COVER into an instance $\langle G', k' \rangle$ of CLIQUE where $G' = G^c$ and $k' = |V| - k$.

Note however that only some of these reductions also hold between the versions of these decision problems that are parameterized relative to the size k of the selected set of vertices V', e.g., $\langle k \rangle$-INDEPENDENT SET parametrically reduces to $\langle k \rangle$-CLIQUE by the first reduction above as $k' = k = f(k)$ but $\langle k \rangle$-VERTEX COVER does not parametrically reduce to $\langle k \rangle$-CLIQUE by the second reduction above as $k' = |V| - k \neq f(k)$. ∎

A more important consequence of the close relationship between decision and parameterized problems is that under certain conditions, results for parameterized problems can propagate to decision and search problems. As noted above, an algorithm for a decision problem can be used to solve any of that problem's associated parameterized problems. Hence, given a polynomial-time intractable decision problem Π with parameter p, if it can be shown that the parameterized problem $\langle p \rangle$-Π is $\mathcal{C}$-hard for some class $\mathcal{C}$ in the W hierarchy above FPT, then Π cannot be solved by an algorithm whose non-polynomial time complexity is purely a function of the aspects encoded in p unless $\mathcal{C} = \mathcal{FPT}$. If in turn this decision problem is related to a search problem in the manner described in Section 2.1.1, fixed-parameter intractability results for parameterized problems propagate to and have implications for search problems as well.

Example 2.1.27 Such a relationship cannot hold relative to the parameter k for the vertex cover problems, as $\langle k \rangle$-VERTEX COVER is in FPT (see [DF99] and references). However, this relationship does hold relative to the parameter d for the 3-D robot motion planning problems defined in Example 2.1.7, where d is the number of joints in the given robot: Any algorithm for the decision version of this problem also solves the version of this problem parameterized relative to d, and since $\langle d \rangle$-3-D ROBOT MOTION PLANNING is $W[SAT]$-hard [CW95], the decision version (and hence, by the reasoning in the previous section, the search version) of this problem does not have an algorithm whose non-polynomial time complexity is purely a function of the number of joints in the given robot unless $FPT = W[SAT]$. ∎

Parameterized computational complexity has been defined here relative to the class P, in that parameterized problems in FPT must have algorithms whose time complexities may be non-polynomial in the parameter but polynomial in the main part of the problem instance. Theories of parameterized computational complexity can be defined relative to any resource-usage class $\mathcal{C}$, e.g., $PSPACE$ or $DLOGTIME$, by restating fixed-parameter tractability to require an algorithm whose resource requirements on any instance (x, y) are bounded by $f(y) \cdot g(x)$ where $f : \Sigma^+ \mapsto \mathcal{N}$ is an arbitrary function and g satisfies the resource-usage restrictions implicit in $\mathcal{C}$. One such parameterized analogue relative to parallel complexity classes is given in [CI97]. Hence, it is possible to create parameterized analyses that can isolate non-$\mathcal{C}$ sources of algorithmic behavior for any resource-usage class $\mathcal{C}$ of interest. This shows that parameterized computational complexity theory is not so much a single theory as a family of theories which all embody a general technique for analyzing problem complexity that is orthogonal yet complementary to computational complexity theory as defined to date.

Parameter	Alphabet Size $\|\Sigma\|$	
	Unbounded	Parameter
—	NP-complete	$\not\in XP$
k	$W[t]$-hard for $t \geq 1$	$W[t]$-hard for $t \geq 1$
m	$W[2]$-hard	FPT
k, m	$W[1]$-complete	FPT

Table 2.1: A Systematic Parameterized Complexity Analysis of the LONGEST COMMON SUBSEQUENCE Problem Relative to $S = \{|\Sigma|, k, m\}$ (adapted from [BDF+95, Table 1]). See page 100 for a definition of this problem.

2.1.3 Systematic Parameterized Complexity Analysis

Individual parameterized results are very good at establishing whether or not a given problem has an algorithm whose non-polynomial time complexity is purely a function of a particular set of aspects of that problem, e.g., the 3-D robot motion planning problem relative to the number of joints in the given robot (see Example 2.1.27). However, if one is interested in fully characterizing the set of non-polynomial time algorithms for a polynomial-time intractable problem Π relative a set S of aspects of Π, individual results are not sufficient because a fixed-parameter tractability (intractability) result relative to a particular set $S' \subseteq S$ says *nothing* about which aspect-subsets (supersets) of S' also render Π fixed-parameter tractable (intractable). Rather, such a characterization requires that all possible parameterized results be derived relative to S.

Definition 2.1.28 *Given a decision problem* Π *and some subset* $S = \{s_1, \ldots, s_n\}$ *of the aspects of* Π*, a* **systematic parameterized complexity analysis of Π relative to S** *determines the parameterized complexity of* Π *relative to all* $2^n - 1$ *non-empty subsets of* S*.*

Several such analyses have been done to date for various problems [BDFW95, BDF+95, Ces96, CW95, Eva98, Hal96, War96a]. The list of parameterized results produced by a systematic parameterized complexity analysis relative to some set of aspects S for a problem Π can be visualized as a **polynomial time intractability map**[1] that shows the manners (and *some* of the forms) in which the polynomial time intractability of Π can manifest itself as algorithms whose non-polynomial time complexities are purely functions of the aspects in S (see Table 2.1).

Performing a systematic parameterized complexity analysis relative to a set S of aspects of a decision problem Π may involve proving far less than $2^{|S|} - 1$ parameterized results.

[1]This mapping metaphor, as well as the idea of using parameterized complexity analysis as a tool to discover the roots of a problem's polynomial time intractability, was inspired by Garey and Johnson's classic discussions on how to apply computational complexity theory to analyze polynomial time intractable problems [GJ79]. The historically-minded reader is strongly encouraged to re-read this book (especially Sections 4.1 and 4.3) if only to see an early and elegant framing of various issues that would subsequently be addressed in parameterized computational complexity theory.

Indeed, as the following lemmas show, each proven result automatically implies many others. These lemmas have been implicit in many previous parameterized analyses, and are stated here for the sake of clarity. Note that depending on context in the proofs below, S denotes either a set of aspects or an encoding of this set into a string over some alphabet Σ.

Lemma 2.1.29 *Given sets S and S' of aspects of a decision problem Π, if $S' \subseteq S$ then parameterized problem $\langle S \rangle$-Π parametrically reduces to parameterized problem $\langle S' \rangle$-Π.*

Proof: Let $S'' = S - S'$, i.e., the set of aspects in S that are not in S'. Given an instance (x, S) of $\langle S \rangle$-Π, construct an instance (x', S') of $\langle S' \rangle$-Π such that $S' = S - S''$ and $x' = x$. This can be done in $f(S)|x|^\alpha$ time for some function $f : \Sigma^+ \mapsto \mathcal{N}$ and some constant α that is independent of x and S. To complete the proof, note that as S' is trivially a function of S by the construction given above, $x = x'$, and the set Y_Π is the set of yes-instances underlying both $\langle S \rangle$-Π and $\langle S' \rangle$-Π, the given instance of $\langle S \rangle$-Π has a solution if and only if the constructed instance of $\langle S' \rangle$-Π has a solution. ∎

Lemma 2.1.30 *Given sets S and S' of aspects of a decision problem Π, if parameterized problem $\langle S' \rangle$-Π is fixed-parameter tractable and $S' \subseteq S$, then parameterized problem $\langle S \rangle$-Π is fixed-parameter tractable.*

Proof: Follows from Lemmas 2.1.29 and 2.1.16. ∎

Lemma 2.1.31 *Given sets S and S' of aspects of a decision problem Π, if parameterized problem $\langle S \rangle$-Π is $\mathcal{C}$-hard for some class $\mathcal{C}$ in the W hierarchy above FPT and $S' \subseteq S$, then parameterized problem $\langle S' \rangle$-Π is $\mathcal{C}$-hard.*

Proof: Follows from Lemma 2.1.29 and the $\mathcal{C}$-hardness of $\langle S \rangle$-Π. ∎

Example 2.1.32 Consider the LONGEST COMMON SUBSEQUENCE (LCS) decision problem as analyzed in Table 2.1 relative to the set of aspects $A = \{|\Sigma|, k, m\}$. As $\langle |\Sigma|, m \rangle$-LCS is in FPT [BDFW95], $\langle |\Sigma|, k, m \rangle$-LCS is also in FPT by Lemma 2.1.30. Moreover, as $\langle k, m \rangle$-LCS is $W[1]$-complete [BDFW95] (and hence $W[1]$-hard by definition), $\langle k \rangle$-LCS and $\langle m \rangle$-LCS are also $W[1]$-hard by Lemma 2.1.31. ∎

There are additional ways of linking sets of numerical-valued aspects from the same problem. Given two sets S and S' of numerical-valued aspects for some decision problem Π, write $S \leq S'$ if for each aspect $s \in S$, there is a function $g_s : \Sigma^+ \mapsto \mathcal{N}$ such that it is always the case in any instance of Π that $s \leq g_s(S')$. For example, in the LONGEST COMMON SUBSEQUENCE decision problem, $\{m\} \leq \{\min_{1 \leq i \leq k} |X_i|\}$ because the common subsequence cannot be longer than the shortest given string.

Lemma 2.1.33 *Given sets S and S' of numerical-valued aspects of a decision problem Π, if parameterized problem $\langle S \rangle$-Π is fixed-parameter tractable and $S \leq S'$, then $\langle S' \rangle$-Π is fixed-parameter tractable.*

Proof: Let A be any FPT algorithm for parameterized problem $\langle S \rangle$-Π with instances of the form (x, S). By definition, A runs in time $f(S)|x|^\alpha$ for some function $f : \Sigma^+ \mapsto \mathcal{N}$ and some constant α that is independent of x and S. Create a function $g : \Sigma^+ \mapsto \mathcal{N}$ by replacing every occurrence of every aspect $s \in S$ in function f by the appropriate function $g_s(S')$. As $s \leq g_s(S')$ for every $s \in S$ by definition, algorithm A runs in time $g(S')|x|^\alpha$. To complete the proof, note that FPT algorithm for $\langle S \rangle$-Π will also solve $\langle S' \rangle$-Π. ∎

Lemma 2.1.34 *Given sets S and S' of numerical-valued aspects of a decision problem Π, if parameterized problem $\langle S' \rangle$-Π is $\mathcal{C}$-hard for some class $\mathcal{C}$ in the W hierarchy above FPT and $S \leq S'$, then $\langle S \rangle$-Π is not fixed-parameter tractable unless $\mathcal{C} = \mathcal{FPT}$.*

Proof: Follows from Lemmas 2.1.33 and the $\mathcal{C}$-hardness of $\langle S' \rangle$-Π. ∎

Under certain conditions, very powerful parameterized results may be implicit in an NP-hardness result for a decision problem.[2]

Lemma 2.1.35 *Given a set S of aspects of a decision problem Π, if Π is NP-hard when the value of every aspect $s \in S$ is fixed, then the parameterized problem $\langle S \rangle$-Π is not in XP unless $P = NP$.*

Proof: If $\langle S \rangle$-Π is in XP then by definition this problem is solved by an algorithm that runs in time $f(S)|x|^{g(S)}$ for some functions f and g. When the value of every aspect in S is fixed, the values of $f(S)$ and $g(S)$ become constants and this running time becomes polynomial in $|x|$. As this algorithm also solves Π and Π is NP-hard when the value of every aspect in S is fixed, every NP-complete decision problem (and hence every decision problem in NP) can be solved in polynomial time, i.e., $P = NP$. ∎

Example 2.1.36 Consider again the LONGEST COMMON SUBSEQUENCE (LCS) decision problem. As LCS is NP-complete (and hence NP-hard by definition) when $|\Sigma| = 2$ [Mai78], $\langle |\Sigma| \rangle$-LCS $\not\in XP$ unless $P = NP$ by Lemma 2.1.35. ∎

The analyses given in Chapter 4 will show that it is possible to create large polynomial time intractability maps by applying the lemmas above to a relatively small core of parameterized results.

---

[2] Lemma 2.1.35 has been noticed independently by several researchers over the last few years. It has recently gained significance in light of the following:

Lemma [**DFS98**] *Given a set S of numerical-valued aspects of a decision problem Π, if Π is NP-hard when the value of every aspect $s \in S$ is fixed to a constant, then the parameterized problem $\langle S \rangle$-Π is not XP-hard unless $P \neq NP$.*

Taken together, these results imply that XP is a natural "top" class of the W hierarchy, in the sense that the relationship of certain parameterized problems to XP and any classes above it cannot be resolved without first answering certain open questions within classical computational complexity theory, i.e., whether P is or is not equal to NP [Fel97]. This and other relationships between classical and parameterized theories of computational complexity are promising topics for future research.

Though the polynomial-time intractability maps produced by systematic parameterized complexity analyses will typically contain both W-hardness results and FPT algorithms, it is perhaps the algorithms that are of the most interest. Two possible groups of users for such algorithms and the manners in which they would use these algorithms are as follows:

1. **Algorithm Users**: Given a polynomial time intractable problem and typical ranges of aspect-values from actual instances of the problem, one can use these ranges of aspect-values to select the FPT algorithms within the intractability map that will operate the most efficiently on problem instances that occur in practice.

2. **Algorithm Designers**: Given a polynomial time intractable problem, one can use the FPT algorithms within the polynomial time intractability map to guide research on new algorithms for the problem. This is possible because many general algorithmic techniques for polynomial time intractable problems have characteristic non-polynomial running times, e.g., both backtracking and branch-and-bound search are exponential in the depth and degree of the search tree and dynamic programming is exponential in the number of dimensions and maximum dimension size of the implicit table. Hence, knowing which sets of aspects can express the non-polynomial time complexity of an algorithm for a problem specifies both the algorithmic techniques that can be productively applied to that problem and the sets of aspects relative to which these applications can be made.

These two approaches to using the FPT algorithms in polynomial-time intractability maps are illustrated in Figure 2.3. These approaches apply intractability maps in opposite manners and to different ends. In the former case, one uses aspects to select good algorithms from the intractability map for practical applications; in the latter case, one uses the algorithms in the intractability map to select significant groups of aspects for further algorithm research and enhancement of the map. In both cases, note that fixed-parameter intractability results are important because they establish relative to which sets of aspects FPT algorithms do *not* exist.

The ability to systematically delimit the manners of FPT algorithms that can exist for a given problem allows us to answer an outstanding question in computer science: namely, given a polynomial-time intractable problem, which sets of aspects of the problem can be said to be responsible for (and hence are sources of) that problem's polynomial-time intractability? As algorithms are the ultimate goals of complexity analysis, perhaps the most useful definition of a problem's sources of polynomial-time intractability is those sets of aspects of the problem that encode this intractability *relative to FPT algorithms for that problem*, i.e., those sets of aspects that can be exploited in subsequent applications and research as described above.

Definition 2.1.37 *Given a polynomial-time intractable decision problem* Π *and some subset* S *of the aspects of* Π*,* S **is a source of polynomial-time intractability for** Π *if* $\langle S \rangle$-Π *is in FPT.*

Parameter	c Unbounded	c Parameter
—	—	$W[1]$-complete
a	FPT A1: $O(2^ab^2 \log c)$	FPT A2: $O(a^cb)$
b	$W[2]$-hard	$W[2]$-complete
a, b	FPT A3: $O(a^bc^3)$	FPT A4: $O(c^{b\sqrt{a}})$

(a)

a	b	c small	c large
small	small	A1, A2, A3, A4	A3, A4
small	large	A1, A2	—
large	small	A2, A3	A3
large	large	A2	—

(b)

Figure 2.3: Systematic Parameterized Complexity Analysis and Polynomial Time Intractability Maps. a) The polynomial time intractability map resulting from the systematic parameterized complexity analysis of a hypothetical polynomial-time intractable decision problem Π relative to a set of aspects $S = \{a, b, c\}$ of Π. The time complexity of each FPT algorithm is given in O-notation. b) The list of practical FPT algorithms relative to the aspects in S when the typical values for each aspect are either small or large. Note that no given FPT algorithm is practical when both b and c have large values.

For example, the sources of polynomial-time intractability in the intractability map given in Figure 2.3 relative to aspect-set $S = \{a, b, c\}$ for problem Π are $\{a\}$, $\{a, c\}$, $\{a, b\}$, and $\{a, b, c\}$ via the FPT algorithms A1, A2, A3, and A4, respectively. In this thesis, the focus will be on sources of polynomial-time intractability that are "minimal" in the sense that their associated FPT algorithms are not trivial extensions of other FPT algorithms along lines such as those described in Lemmas 2.1.30 and 2.1.33. Though this kind of minimality is almost impossible to define formally and must thus be evaluated on a case-by-case basis (as is done in the analyses given in Chapter 4), such minimal sources are a useful characterization of the "core" of distinct FPT algorithms for a problem relative to some set of its aspects. At first glance, the definition of a source of polynomial-time intractability given above may seem trivial and obvious. However, as will be discussed further in Chapter 3, the failure to make such a definition underlies certain misinterpretations of the results of classical complexity analyses.

When interpreting polynomial-time intractability maps, it is very important to remember that an FPT algorithm for a parameterized problem $\langle S \rangle$-Π based on a decision problem Π only implies that S is a source of polynomial-time intractability for Π if Π is polynomial-time intractable, and that any subsequent conclusions based on interpreting the aspects associated with the FPT algorithms in this map as sources of polynomial-time intractability for Π are valid to the extent that all hardness results in the map are stated relative to classes of the W hierarchy that properly include FPT. These assumptions will often be violated in practice, in that the decision problem being analyzed will only be known to be NP-hard and the intractability map will be constructed from FPT algorithms and W-hardness results. However, given the confidence in the working hypotheses that P and NP as well as the levels of the W hierarchy are separate and the promising applications described above that are latent in intractability maps, the FPT algorithms in a polynomial-time intractability map for a problem Π relative to a set of aspects S of Π can, for all practical purposes, be considered as representing the sources of polynomial-time intractability in Π relative to S, and will be treated as such in the remainder of this thesis.

Even given the acceptance of the working hypotheses above, a polynomial-time intractability map for a given decision problem Π relative to some set S of the aspects of Π will typically be incomplete in two senses:

1. Such a map does not summarize *all* possible non-polynomial time algorithms for Π, as some of these algorithms may have non-polynomial time complexities that are purely functions of sets of aspects that include aspects that are not in S.

2. Such a map need not even summarize all possible FPT algorithms relative to S because parameterized complexity analysis can only say whether at least one or no FPT algorithm can exist relative to a particular subset S' of S. There may be many FPT algorithms relative to S', and knowing some of them does not necessarily imply anything about the others.

This incompleteness is inherent in the map, and is different from the potentially resolvable uncertainty introduced by conjecture-dependent intractability results. Experience has shown that these deficits are not as serious as they may first appear. As for the first point, my own experience in doing systematic parameterized analyses of phonological processing systems [DFK+94, War96a, War96b] suggests that such analyses on some initial set of aspects will often lead to the discovery and subsequent analysis of new and hopefully more relevant aspects. As for the second point, the proliferation of successively more efficient FPT algorithms for the parameterized problem $\langle k \rangle$-Vertex cover (see [DF95a, SF99] and references) suggests that once the discovery of the first FPT algorithm establishes that a decision problem Π can be fixed-parameter tractable relative to some set of aspects S, further research (aided by various techniques developed specifically for creating FPT algorithms (see [DF95c, DF99] and references)) will uncover more FPT algorithms relative to S. Hence, the inherent incompleteness of an intractability map should be seen not as an insurmountable difficulty but rather as a reflection of the incomplete knowledge that characterizes the intermediate stages of any scientific investigation.

Ideally, a systematic parameterized complexity analysis should be part of an ongoing discovery procedure for analyzing a polynomial-time intractable problem, in which classical and parameterized computational complexity are used in an alternate and complementary fashion, the former to show what subproblems of the given problem are polynomial-time intractable and the latter to diagnose the sources of this intractability. Such analyses do not preclude and indeed should actively encourage further research into various kinds of polynomial time approximation algorithms for the given problem. It must be remembered that the ultimate goal of algorithm research is to solve a given problem in the most efficient manner by whatever means are most appropriate. What is important is ensuring that when this choice of means is made, we know all our options. It is in aiding this, by making possible the discovery of the sets of non-polynomial time optimal-solution algorithms for polynomial-time intractable problems, that systematic parameterized complexity analysis will probably have its greatest impact in years to come.

2.2 Phonology

2.2.1 What is Phonology?

Phonology is the area of linguistics that studies the mapping between mental and spoken representations in human languages [Ken94]. Every human language (be it spoken, signed, or written) exhibits regularities in the way that symbols are combined to form utterances in that language; the phonology of a language both describes these regularities and uses them to discover the nature of the mapping between mental and spoken representations in that language. To the extent that such mappings are computable and can thus be used to recover mental representations from spoken representations and vice versa, phonology is an important component of natural language processing systems. This section gives a very brief introduction to some of the types of phenomena studied within phonology. Readers wishing more comprehensive treatments should consult a standard textbook such as [Ken94].

The most basic type of phonological phenomenon is the dependence of the form of a symbol on its surrounding symbols.

Example 2.2.1 (English) The spoken form of the plural suffix in English seems to be a function of the final sound in the noun being pluralized.[3]

noun	plural	spoken form of plural suffix
mop	mops	[**s**]
pot	pots	[**s**]
pick	picks	[**s**]
mob	mobs	[**z**]
pod	pods	[**z**]
pig	pigs	[**z**]

∎

Such regularities can involve not only change in symbol form but also insertion or deletion of symbols.

Example 2.2.2 (Maori [SB78, Exercise 7.4]) Consider the following verb forms from Maori, a language spoken in New Zealand.

[3] In the following, boldface symbols enclosed in square brackets will represent sounds according to the conventions of the International Phonetic Alphabet (IPA). For instance, [**s**] and [**z**] will stand for the "s" and "z" sounds at the beginning of the words "sop" and "zoo" as pronounced by a native speaker of English. When IPA notation is not used in presenting language data, it will be assumed that word spelling accurately reflects pronunciation.

active	passive	gerund	(hypothesized) root verb	meaning
afi	afitia	afitaŋa	afit	"embrace"
hopu	hopukia	hopukaŋa	hopuk	"catch"
aru	arumia	arumaŋa	arum	"follow"
paa	paaŋia	paaŋaŋa	paaŋ	"shut"
mau	mauria	mauraŋa	maur	"carry"
wero	werohia	werohaŋa	weroh	"stab"
patu	patua	patuŋa	patu	"strike"
kite	kitea	kiteŋa	kite	"see"

This somewhat confusing mass of data is simplified if one assumes that there is a common root verb underlying each triple of verb forms given above, and that the spoken form of each verb form of a root verb depends on whether the final sound in that root is a consonant or a vowel. Given this, active forms are created by deleting root-final consonants, passive forms are created by adding a suffix "-ia" ("-a") if the final sound in the root is a consonant (vowel), and gerund forms are created by adding a suffix "-aŋa" ("-ŋa") if the final sound in the root is a consonant (vowel). ∎

Regularities may also be a function of symbols in more distant parts of the word. In one such phenomenon, vowel harmony, certain aspects of vowels in one particular part of a word propagate to vowels in the rest of the word.

Example 2.2.3 (Turkish [Ken94, p. 25]) In Turkish, the form of the vowel in many suffixes seems to be a function of the vowel in the root word, e.g., the vowel in the plural suffix:

noun	plural	meaning
dal	dallar	"branch"
kol	kollar	"arm"
kul	kullar	"slave"
yel	yeller	"wind"
diš	dišler	"tooth"
gül	güller	"rose"

∎

It is tempting to suggest that such regularities are simply encoded in the mental lexicon, i.e., all possible forms of a word are stored in the mental lexicon. In this case, the mapping between mental and spoken forms is that of identity and hence is trivial. However, several lines of evidence suggest otherwise. Consider for instance languages which have both vowel harmony and complex inflectional systems.

Example 2.2.4 (Turkish [Spr92, p. 44]) In Turkish, whole sentences may be formed by concatenating suffixes onto a root word. By vowel harmony, the vowels in each of these suffixes is a function of the vowel in the root word, e.g.,

çöplüklerimizdekilerdenmiydi =
çöp + *lük* + *ler* + *imiz* + *de* + *ki* + *ler* + *den* + *mi* + *y* + *di*
"was it from those that were in our garbage cans?"

∎

The combinatorial explosion of possible word forms (many of which may never have been heard or used by the speaker before) makes the storage of all such word forms in a mental lexicon unlikely. More compelling evidence comes from regularities involving words which are introduced into a language, e.g., nonsense words and words borrowed from other languages. As such words could not previously have been in the mental lexicon, these regularities suggest that at least some observed symbolic regularities are not static patterns but are rather the result of productive mechanisms operating on spoken and mental representations. For instance, English speakers flawlessly pluralize nonsense nouns according to the regularity given in Example 2.2.1, e.g., "grikip" ⇒ "grikips" ([**s**]) and "grikib" ⇒ "grikibs" ([**z**]). The treatment of borrowed words can be even more provocative.

Example 2.2.5 (Chinenglish [DeF91, Exercise 3.10]) Chinenglish is a hybrid of English and Cantonese that is spoken in Hong Kong. Words borrowed from English that contain [**r**] are changed to conform with the sound-patterns of Cantonese.

List I: [**r**] deleted

1.	tart	4.	guitar	7.	sergeant
2.	party	5.	bar		

List II: [**r**] pronounced as [**l**] [4]

1.	strawberry	3.	aspirin	5.	curry
2.	brandy	4.	Listerine		

It seems that [**r**] is deleted if it occurs before a consonant or as the final sound in a word, and [**r**] becomes [**l**] if it occurs before a vowel. Note that the pattern of [**r**]-deletion in List I is also characteristic of certain New England dialects of English such as that spoken in Boston. ∎

[4] I've literally seen this particular phenomenon in action. Several years ago, I shared an office with a native speaker of Cantonese who was learning English. I came in one morning to find the following note he had left on my desk: "Phone Let Kloss about giving blood."

Example 2.2.6 (Japanese [Lov73]) Japanese does not recognize any difference between the [**r**] and [**l**] sounds – that is, there are no pairs of words in Japanese with different meanings whose pronunciations are distinguished only by an [**l**] sound in a certain position in one word and an [**r**] sound in the corresponding position in the other word, cf., "ball" and "bar" in English. Moreover, Japanese words have a very restricted syllabic structure that does not allow many types of multi-consonant clusters. Hence, words borrowed from English are changed both by having [**l**]-sounds either deleted or transformed into [**r**] and by having vowels inserted to break up certain multi-consonant clusters.

dosutoru	"duster"
sutoroberri	"strawberry"
kurippaa	"clippers"
sutoraiki	"strike"
katsuretsu	"cutlet"
parusu	"pulse"
gurafu	"graph"

∎

Such evidence as that given above suggests that (1) there are at least two distinct levels of representation in natural language, the mental (at which elements of the lexicon are combined) and the spoken, and (2) the regularities observed in spoken forms are not encoded in the lexicon directly but are rather properties of the non-trivial mapping between mental and spoken representations. As phonology is inherently intertwined with **morphology**, the area of linguistics that studies how elements of the lexicon are stored and subsequently combined to create mental representations, the two are often merged into the area of linguistics called **morphophonology**. Note that the emphasis on symbols rather than sounds in the preceding paragraphs is intentional, because phonological phenomena occur in all types of natural human languages. For example, regularities similar to those described above have also been observed in the manner in which hand and body movements are combined in sign languages used by the deaf [Bre95].

There are many theories within phonology, each of which proposes types of representations for mental and spoken forms and mechanisms for describing the mapping between these representations. Broadly speaking, phonological theories can be divided into two types which differ in their conception of these mechanisms:

1. **Rule-Based**: Mental representations are transformed into spoken representations by the application of rewriting rules.

2. **Constraint-based**: A set of candidate spoken representations is associated with a given mental representation, and constraints are applied to this set in some manner to select the actual spoken representation.

These mechanisms are descriptively equivalent, in that any phonological phenomenon can be described using either rules or constraints. Hence, these theories are evaluated more in terms of the ease with which regularities such as those described in the examples in this section can be captured by their respective mechanisms. At present, constraint-based theories are preferred because they focus on capturing the essence of phonological mappings without invoking (possibly unjustifiable) mechanisms for implementing these mappings (see [Bir95, LP93] for further discussion). This concern with unjustifiable mechanisms seems to have been motivated in part by the computational intractability associated with older phonological theories based on rewriting rules. Implicit in many discussions of constraint-based theories is the belief that the simple machinery by which constraints are invoked to create phonological mappings will make computations associated with these theories tractable. However, as will be shown in Chapter 4, such faith is very sadly misplaced.

2.2.2 Phonological Representations

The cornerstone of any phonological theory are the types of mental and spoken representations used within that theory. Indeed, as noted by Kenstowicz, the chief theoretical problem in examining phonological phenomena is " ... to discover a representation that permits the facts to be stated clearly so that generalizations emerge which provide the basis for an explanatory theory." [Ken94, p. 548]. This section gives an overview of the representations used by the phonological theories examined in this thesis.

There are three types of phonological representations (see Figures 2.4 and 2.5). Each is linear in the sense that it describes the order of phonological events in an utterance; however, they differ in the detail and manner in which these events are described and related.

1. **Strings of symbols** (Figure 2.4(a)): A phonological representation is a string of symbols $s_1 s_2 s_3 \ldots s_n \in \Sigma^n$ for some alphabet Σ. This is the simplest form of representation, and it is motivated by the recognition of discrete items in speech, e.g., sounds, syllables, stress.

2. **Strings of segments** (Figure 2.4(b) and Figure 2.5(e)): A phonological representation is a string of segments, where a segment is essentially a symbol that has an internal structure. This internal structure is defined relative to a set F of **features**, each of which has an associated set of possible values. In principle, there is no limit to the number of values that can be in this set; however, in practice most features are binary-valued and have the value-set $\{+, -\}$. A **segment** relative to a feature-set F is defined by a subset $F' \subseteq F$ and a value for each of the selected features in F'. Features in $F - F'$ for such a segment are said to be undefined for that segment. Following linguistic convention, segments will be written as sets of feature-value pairs, e.g. $\{[f_1\ v_1], [f_2\ v_2], \ldots, [f_m\ v_m]\}$ where v_i is one of the values associated with feature f_i for $1 \leq i \leq m$. Typically, only a small subset of the possible subsets of F can define valid segments. One popular way of encoding these constraints on feature co-occurrence is to use the features in F to label the vertices of a rooted tree called a **feature geometry** such that for each valid segment, the set of features defining that

(a)

$$\mathbf{A} \quad \mathbf{B} \quad \mathbf{C} \quad \mathbf{D}$$

(b)

$$\begin{bmatrix} \mathbf{a} & + \\ \mathbf{b} & - \end{bmatrix} \begin{bmatrix} \mathbf{a} & + \\ \mathbf{b} & + \\ \mathbf{c} & - \end{bmatrix} \begin{bmatrix} \mathbf{a} & - \\ \mathbf{b} & - \\ \mathbf{c} & + \end{bmatrix} \begin{bmatrix} \mathbf{a} & - \\ \mathbf{b} & - \\ \mathbf{c} & - \end{bmatrix}$$

(c)

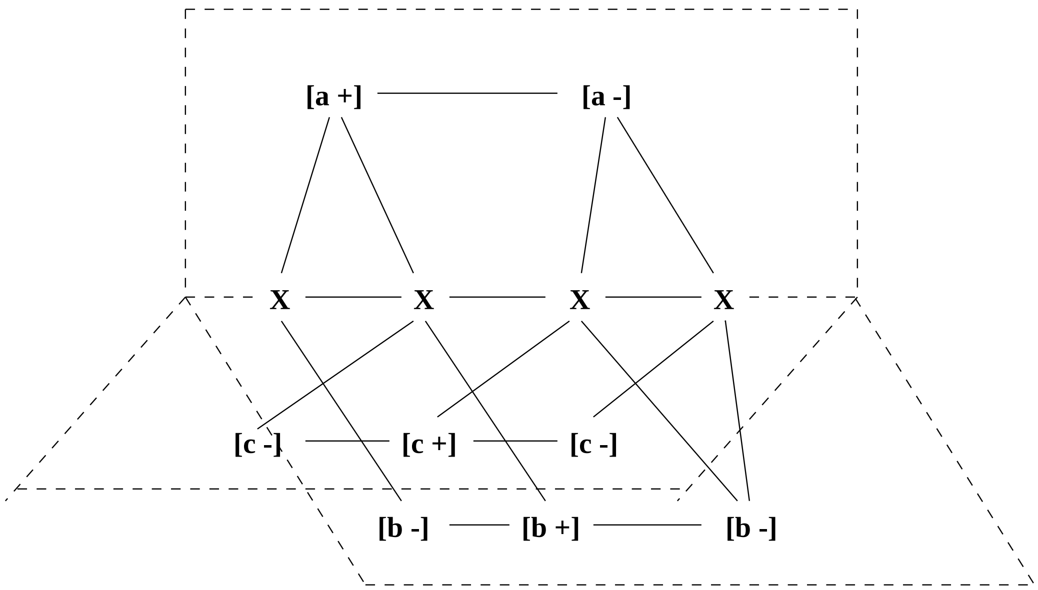
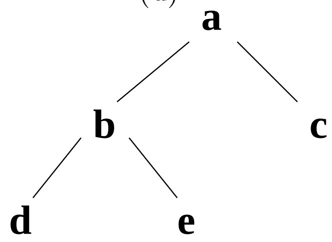


Figure 2.4: Phonological Representations. a) A string of symbols over the alphabet $\Sigma = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$. b) A string of segments over the binary-valued features **a**, **b**, and **c**. Note that feature **c** is undefined for the first segment. c) A "paddle wheel" autosegmental structure for the segment-string in (b), in which the timing-tier slots are denoted by **X**'s and each of the features in (b) has its own tier.

(d)



(e)

$$\begin{bmatrix} \mathbf{a} & - \\ \mathbf{b} & + \\ \mathbf{c} & + \\ \mathbf{d} & + \\ \mathbf{e} & + \end{bmatrix} \begin{bmatrix} \mathbf{a} & + \\ \mathbf{b} & - \\ \mathbf{c} & - \end{bmatrix} \begin{bmatrix} \mathbf{a} & - \\ \mathbf{b} & + \\ \mathbf{d} & - \\ \mathbf{e} & - \end{bmatrix}$$

(f)

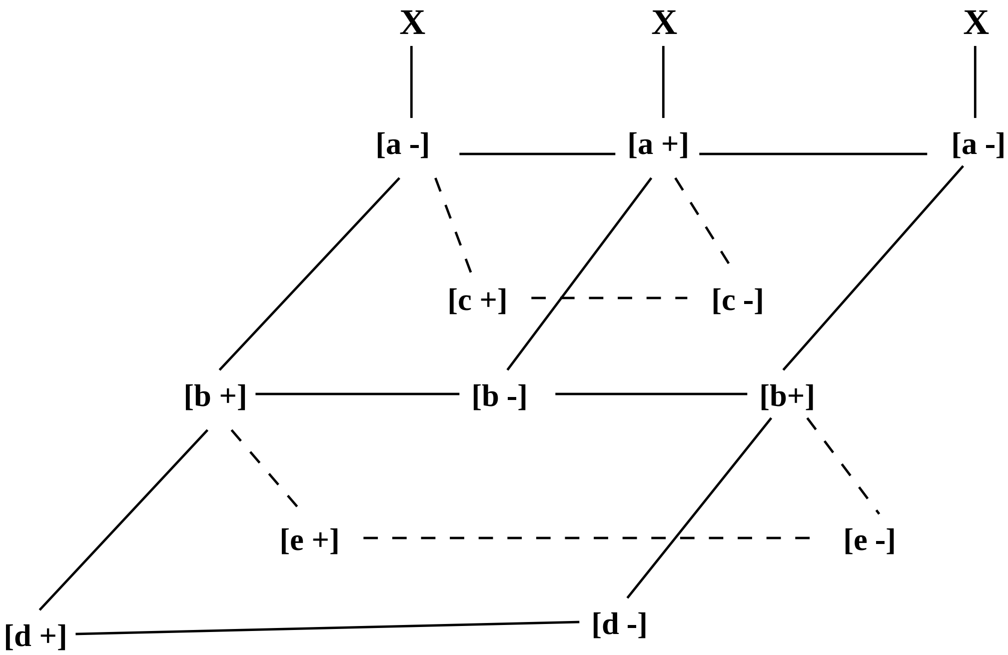


Figure 2.5: Phonological Representations (Cont'd). d) A feature geometry over the binary-valued features **a**, **b**, **c**, **d**, and **e** which is rooted at feature **a**. e) A string of valid segments over the feature-geometry in (d). f) The autosegmental structure associated with the segment-string in (e) relative to the feature geometry in (d), in which the timing-tier slots are denoted by **X**'s and each of the features in (d) has its own tier.

segment corresponds to the set of features labeling the nodes of a subtree of this rooted tree. A sample feature geometry is shown in Figure 2.5(d).

Segmental strings are the representation on which most phonological research has been based; in such representations, each feature in a feature-set F represents some significant feature of sounds in the language of interest, e.g., vowel length, consonant voicing. Features and segments in phonology were originally proposed by Jakobson and Trubetzkoy in the 1930's, and the first formal theory of segmental phonology was given by Chomsky and Halle in their classic book *The Sound Pattern of English (SPE)* in 1968 [CH68]. The division of symbols into bundles of feature-value pairs was motivated by the realization that, given an appropriate set of features, the seemingly random alternation of symbols in many phonological phenomena could be accounted for by simple mechanisms that propagate feature-values across adjacent segments.

Example 2.2.7 Under standard sets of phonological features, sounds [**s**] and [**z**] differ only in the feature **voiced** – that is, [**s**] has the feature-value pair [**voiced** $-$] and [**z**] has the feature-value pair [**voiced** $+$]. The same relationship also holds for the pairs of sounds [**p**] / [**b**], [**t**] / [**d**], and [**k**] / [**g**]. Hence, the alternation of [**s**] and [**z**] for the English plural suffix in Example 2.2.1 can be accounted for by postulating a mechanism which ensures that the plural suffix-segment takes on the value of the feature **voiced** in the final segment of the word to which it is affixed. ▮

3. **Autosegmental structures** (Figure 2.4(c) and Figure 2.5(f)): A phonological representation is a collection of single-feature segment-strings called **tiers**, one of which is a special string of timing slots. Each feature can appear on only one tier, and a tier consists of segments based on a single feature. Each segment in a tier is linked to some set of segments in other tiers by **association lines**. The set of features describing the sound at any particular point in the utterance encoded by the representation is the set of features in all segments linked by association lines to the timing slot corresponding to the moment of interest. Such a representation is called autosegmental because each segment in the representation is autonomous, i.e., it exists and can be manipulated independently of other segments.

   Since their introduction by Goldsmith in 1976 [Gol76] (see also [CL92, Gol90] and references), autosegmental representations have become the dominant type of representation in phonological research. Originally, all tiers were assumed to be associated directly to the timing tier (the so-called "paddle-wheel" configuration) (Figure 2.4(c)). However, current research organizes tier associations according to feature geometries (Figure 2.5(f)). The division of strings of segments into collections of tiers of linked autosegments was motivated by the realization that the description of certain long-range phonological patterns could be simplified by allowing features that could be associated with (and hence could automatically propagate changes in value by being associated with) multiple, possibly non-adjacent, segments.

Example 2.2.8 Under standard sets of phonological features, the vowels [**e**] and [**a**] differ only in the feature **back** – that is, [**e**] has the feature-value pair [**back** $-$] and

[a] has the feature-value pair **[back +]**. Moreover, all vowels in the set { **[e], [i], [ü]** } ({ **[a], [o], [u]** }) have the feature-value pair **[back −]** (**[back +]**). Hence, Turkish vowel harmony as described in Example 2.2.3 can be accounted for by postulating a tier for feature **back** and a mechanism which ensures that the **back** autosegment associated with the (rightmost) vowel in the root is also associated with the vowels in all suffixes. ▮

Note that these representations are, for the most part, equivalent in the sense that almost anything that can be encoded in one representation can be encoded into one of the others:

- Strings of symbols over an alphabet Σ can be encoded as strings of single-feature / $|\Sigma|$-valued or $\log_2 |\Sigma|$-feature / binary-valued segments, or as $\log_2 |\Sigma|$-tier autosegmental structures.
- Strings of segments relative to some set of features F can be encoded as strings of symbols over an alphabet composed of all possible segments defined by F or as autosegmental structures under an appropriate feature geometry for F.
- An autosegmental structure can be encoded as a set of symbol-strings such that each string corresponds to the string of autosegments on a particular tier and the associations between autosegments on different tiers are encoded in the relation of a given multi-tape finite-state transducer [Kay87] (see Section 2.2.3) or as a segment (symbol)-string such that each segment (symbol) encodes the features on all tiers that are linked to a particular slot on the timing tier, i.e., the features describing a sound at a particular point in an utterance [BE94, Eis97a].

More complex encodings of autosegmental structures that explicitly record the associations between autosegments on different tiers are reviewed in [BE94, Kor95].

Every phonological theory is based on one of the types of representations described above. Each such theory posits mental (**lexical**) and spoken (**surface**) forms that are described using this representation, and the mechanisms posited by that theory are used to relate these forms. The representations used by the phonological theories examined in this thesis are given in Table 2.2. Simple phonological phenomena such as those described in the examples above can be accounted for in a framework in which both the lexical and surface forms are instances of a particular kind of representation and phonological processing occurs by transforming one of these forms into the other (possibly via a chain of intermediate forms that are themselves instances of this representation). However, certain more complex phonological phenomena, e.g., stress-placement, seem to require form-internal structure latent within and computed from the lexical form that is not itself pronounced but is used to produce aspects of the surface form. For example, metrical grids [Ken94, Chapter 10] and prosodic trees [Ken94, Chapters 6 and 11] are not themselves pronounced but are used to compute the positions of various kinds of stress in words and the syllable-structure of words, respectively (a somewhat analogous situation occurs in Optimality Theory where unpronounced "silent" copies of lexical forms must be included in candidate surface forms in order to properly evaluate those candidate surface forms relative to the constraint set

Phonological Theory	Characteristics
Simplified Segmental Grammars	Representation: Segment strings Aspects: • Number of segments in the given (lexical or surface) form • Number of features used to define segments • Maximum number of values associated with any feature
FST-based rule systems	Representation: Symbol strings Aspects: • Number of symbols in the given (lexical or surface) form • Size of the alphabet from which lexical and surface forms are drawn
The KIMMO System	Representation: Symbol strings Aspects: • Number of symbols in the given (lexical or surface) form • Size of the alphabet from which the lexical form is drawn • Size of the alphabet from which the surface form is drawn
Declarative Phonology & Optimality Theory	Representation: Simplified autosegmental structures (symbol strings) Aspects: • Number of symbols in the given (lexical or surface) form • Size of the hidden-component alphabet Σ_h • Size of the visible-component alphabet Σ_v

Table 2.2: Characteristics of Representations of Phonological Theories Examined in this Thesis. This table lists, for each phonological theory examined in this thesis, the type of phonological representation used and the aspects of that representation relative to which that theory is analyzed. Note that the analyses for Declarative Phonology and Optimality Theory are done relative to simplified autosegmental structures; see main text and Figures 2.7 and 2.8 and for details.

(see Section 4.6.1)). This implies that lexical and surface forms are actually different kinds of representations. Tesar and Smolensky [Tes95, Tes96, Tes97a, TS96] have elegantly handled such cases by proposing the existence of a "master" form composed of pronounced and unpronounced material that is intermediate in any processing of surface and lexical forms – that is, a surface form must be transformed into a master form by a reconstruction of the unpronounced material in order to recover its associated lexical form(s) and a lexical form must be transformed into a master form by the addition of the appropriate pronounced and unpronounced material to generate its associated surface form. Under this scheme, lexical and master forms are (respectively, partially and fully-specified) instances of a representation and surface forms are extracted from this representation. In this thesis, let the following

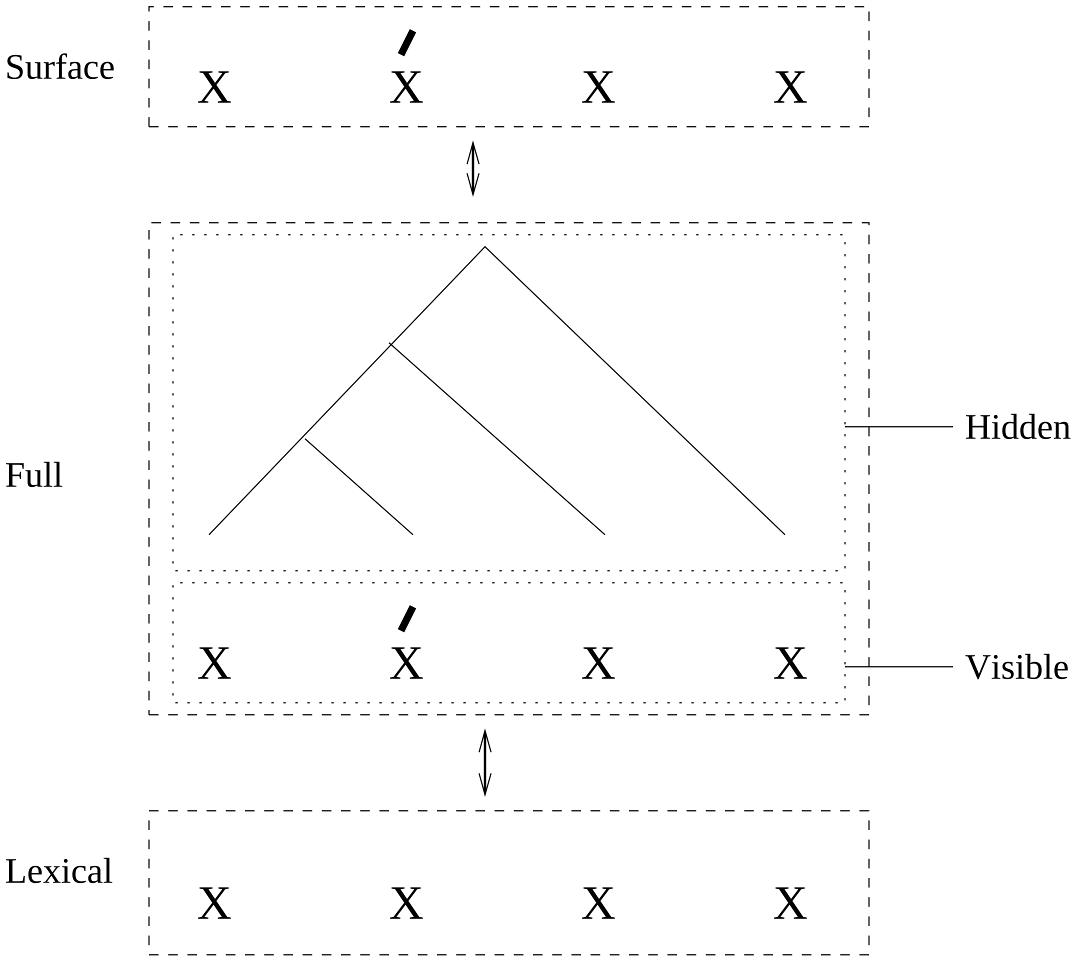


Figure 2.6: Form Decomposition of Phonological Representations (adapted from [Tes96]). This diagram shows the relation of lexical, full, and surface forms and the hidden and visible components of the full form in the case of stress-placement. The **X**'s denote syllables and the heavy dash indicates a stressed syllable. See main text for further explanation of terms.

terms (adopted with slight modifications from those of Tesar and Smolensky) denote the new entities described above:

- **Full form** (full structural description [TS96]): A well-formed and fully-specified phonological representation created by applying the mechanisms of the phonological theory of interest to a lexical form.
- **Hidden component** (non-overt, "hidden" structure [TS96]): Unpronounced portion of the lexical form and its associated full form.
- **Visible component** (overt form / structure [TS96]): Pronounced portion of the lexical form and its associated full form.

If one conceives of the full form as an autosegmental structure, segment string, or symbol string, one can imagine the associated hidden and visible components as a partition of the set of tiers, the set of features, and the symbol-positions in the string, respectively. Within this framework, a surface form would consist of the entire visible component of its associated full form and a lexical form would consist of those portions of the hidden and visible components of the associated full form that are stored in the lexicon. These various forms in the case of stress-placement are shown in Figure 2.6.

In the parameterized analyses given in this thesis, phonological representations will be characterized very broadly by aspects that capture the size of the given (lexical or surface) form and the composition of both the given and computed forms. These aspects are listed in Table 2.2. Note that an aspect describing the size of the computed form will not be necessary as all problems associated with phonological theories examined in this thesis will be restricted to have lexical and surface forms of the same size, i.e., no mechanisms that allow insertion or deletion of material in relating lexical and surface forms will be permitted. The aspects listed for Simplified Segmental Grammars, FST-based rule systems, and the KIMMO system can be derived in a straightforward fashion from their respective representations. The situation is somewhat more complex for Declarative Phonology and Optimality Theory, which have no fixed type of representation. It would be interesting to examine the complexity of these problems relative to autosegmental structures with associated hidden and and visible components; however, certain results [Ris90, Ris94] suggest that fairly basic manipulations of arbitrarily-complex autosegmental structures independent of the application of any theory-specific phonological mechanisms are NP-hard.

In order to factor out the complexity-theoretic effects of manipulating arbitrarily-complex representations while retaining some sort of internal hidden and visible structure in the representations, the formulations of Declarative Phonology and Optimality Theory examined in this thesis will use a simplified autosegmental representation consisting of two strings $h_1h_2 \ldots h_n$ and $v_1v_2 \ldots v_n$ drawn from alphabets Σ_h and Σ_v, respectively. This representation will be encoded as a single symbol string of the form $v_1h_1v_2h_2 \ldots v_nh_n$. The manner in which parts of this representation correspond to the various forms and components described above is illustrated in Figures 2.7 and 2.8. Note that in this representation, various forms can incorporate underspecified elements, which are denoted by the subsets Σ_{hU} and Σ_{vU} of the hidden- and visible-component alphabets Σ_h and Σ_v, respectively. The appearance of elements from Σ_{vU} or Σ_{hU} indicates that a particular element is underspecified in a particular manner, e.g., symbol $\Delta \in \Sigma_{vU}$ can be specified as any of the symbols in a particular subset of $\Sigma_v - \Sigma_{uV}$. Such underspecified elements are the simplest way to allow mechanisms to make changes to lexical forms without invoking more complex representations that allow arbitrary addition and deletion of elements. Within this representation, lexical forms may have underspecified elements in their associated visible and hidden components, surface forms have fully specified visible components and empty hidden components, and full forms cannot have any underspecified elements in their associated visible and hidden components. This matches well with linguistic intuitions that lexical forms represent the unpredictable (relative to phonological mechanisms) portions of visible and hidden components that must be stored in the lexicon, surface forms consist of all of the visible and none of the hidden structure in full forms,

$$
\begin{array}{lccccccccccc}
\text{Hidden} & h_1 & — & h_2 & — & h_3 & — & h_4 & — & h_5 & — & h_6 \\
 & | & & | & & | & & | & & | & & | \\
\text{Visible} & v_1 & — & v_2 & — & v_3 & — & v_4 & — & v_5 & — & v_6
\end{array}
$$

String: $v_1h_1v_2h_2v_3h_3v_4h_4v_5h_5v_6h_6$

(a)

$$
\begin{array}{lccccccccccc}
\text{Hidden} & h_1 & — & h_2 & — & h_3 & — & h_4 & — & h_5 & — & h_6 \\
\text{Visible} & & & & & & & & & & &
\end{array}
$$

String: $h_1h_2h_3h_4h_5h_6$

(b)

$$
\begin{array}{lccccccccccc}
\text{Hidden} & & & & & & & & & & & \\
\text{Visible} & v_1 & — & v_2 & — & v_3 & — & v_4 & — & v_5 & — & v_6
\end{array}
$$

String: $v_1v_2v_3v_4v_5v_6$

(c)

Figure 2.7: A Simplified Autosegmental Representation. a) A full form in the simplified autosegmental representation used in this thesis. b) The hidden component for the full form in (a). c) The visible component for the full form in (a).

and full forms are fully specified. This representation can be viewed as an autosegmental structure consisting of a timing tier and two sets of tiers encoding the hidden and visible components that consist of $\log_2|\Sigma_h|$ and $\log_2|\Sigma_v|$ binary-valued feature tiers apiece, and is thus very similar to the second encoding of autosegmental structures into symbol strings described earlier in this section. As such, this representation is perhaps the simplest form of autosegmental representation which incorporates both visible and hidden components, and should thus be a special case of more complex types of autosegmental representation used in practice.

The seemingly arbitrary restrictions on phonological representations considered here may seem mystifying at this point. However, as will be explained more fully in Chapter 3, these restrictions will allow the parameterized analyses both to focus on the most basic mechanisms responsible for NP-hardness in various problems associated with the phonological theories examined in this thesis and to make more obvious the tradeoffs by which various of these aspects can be encoded into other aspects to maintain this NP-hardness.

$$
\begin{array}{lccccccccccc}
\text{Hidden} & & & & & & & & & & & \\
\text{Visible} & v_1 & — & v_2 & — & v_3 & — & v_4 & — & v_5 & — & v_6
\end{array}
$$

String: $v_1 \odot v_2 \odot v_3 \odot v_4 \odot v_5 \odot v_6 \odot$

(d)

$$
\begin{array}{lccccccccccc}
\text{Hidden} & & & h_2 & & & & & & h_5 & — & h_6 \\
 & & & | & & & & & & | & & | \\
\text{Visible} & v_1 & — & v_2 & & & & v_4 & & v_5 & — & v_6
\end{array}
$$

String: $v_1 \odot v_2 h_2 \Box \odot v_4 \odot v_5 h_5 v_6 h_6$

(e)

Figure 2.8: A Simplified Autosegmental Representation (Cont'd). d) The visible form consistent with the full form in (a). e) A lexical form consistent with the full form in (a). The string-encodings for each form and component are also given, in which symbols $\odot$ and $\Box$ denote underspecified hidden and visible elements in Σ_{hU} and Σ_{vU} that can specify any element in $\Sigma_h - \Sigma_{hU}$ and $\Sigma_v - \Sigma_{vU}$, respectively.

2.2.3 Phonological Mechanisms as Finite-State Automata

If, as noted by Kenstowicz at the beginning of Section 2.2.2, a good phonological representation is important because it allows facts to be stated clearly so that generalizations can emerge, good phonological mechanisms that manipulate these representations are equally important because those generalizations are stated in terms of these mechanisms. Several important classes of phonological mechanisms are:

- the mechanisms (rules or constraints) that implement the mapping between surface and lexical forms;
- the various types of lexicons in which the components of lexical forms are stored; and
- the manners in which components in the lexicon can be combined to create lexical forms.

Each of these mechanisms can be implemented in terms of finite-state automata, which are essentially a type of very simple computational device for manipulating strings of symbols. All of the phonological theories examined in this thesis are either defined to use or can be rephrased such that they use finite-state automata to implement their phonological mechanisms. Hence, this section gives an overview of several important types of finite-state automata (using the configuration-notation for describing automaton computation given in [Brs79, LP81]) and discusses how these automata can be used to implement

(a)

Q	Σ	
	a	b
q_1	q_2	q_1
q_2	q_2	q_3
q_3	$Fail$	q_3
$Fail$	$Fail$	$Fail$

(b)

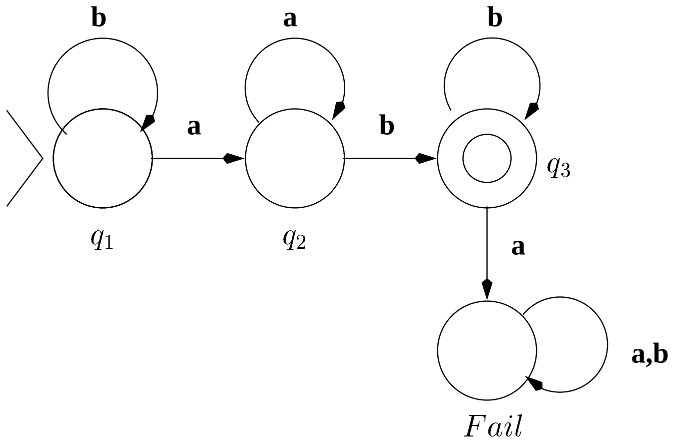


Figure 2.9: A Finite-State Acceptor. The FSA is specified by the 5-tuple $\langle Q, \Sigma, \delta, s, F\rangle$ where $Q = \{q_{,1}\,, q_2, q_3, Fail\}$, $\Sigma = \{a, b\}$, δ is defined by the table in (a) above, $s = q_1$ and $F = \{q_3\}$. The transition diagram for this FSA is given in (b)

phonological mechanisms. Readers wishing more comprehensive treatments should consult [Brs79, HU79, LP81, Per90, RS97b].

The most basic type of finite-state automaton operates on a single string of symbols over some alphabet.

Definition 2.2.9 *A* **finite state acceptor (FSA)** *is a 5-tuple $\langle Q, \Sigma, \delta, s, F\rangle$ where Q is some set of states, Σ is an alphabet, $\delta : Q \times \{\Sigma \cup \{\epsilon\}\} \times Q$ is a transition relation, $s \in Q$ is the start state, and $F \subseteq Q$ is a set of final states.*

Each FSA has an associated edge-labeled directed graph called a **transition diagram** in which the vertices correspond to states and the arcs have labels from Σ such that each arc corresponds to one entry of δ, e.g., $(q, s, q') \in \delta$ becomes $q \xrightarrow{s} q'$ in the transition diagram. A sample FSA and its transition diagram are given in Figure 2.9. The notion of a computation of a FSA on a given input string can be stated formally in terms of configurations of a FSA.

Definition 2.2.10 *A* **configuration of a FSA** $A = \langle Q, \Sigma, \delta, s, F \rangle$ *is a pair* (q, x) *where* $q \in Q$ *and* $x \in \Sigma^*$.

Definition 2.2.11 *Given two configurations* (q, x) *and* (q', x') *of a FSA* $A = \langle Q, \Sigma, \delta, s, F \rangle$, (q, x) **yields** (q', x') *in one step, i.e.,* $(q, x) \vdash (q', x')$, *if* $x = wx'$ *for some* $w \in \Sigma \cup \{\epsilon\}$ *and* $(q, w, q') \in \delta$.

Let $\vdash^*$ represent the reflexive transitive closure of the yield relation, i.e., $(q, x) \vdash^* (q', x')$ if and only if either $q = q'$ and $x = x'$ or there exists some sequence $(q_1, x_1), (q_2, x_2), \ldots, (q_n, x_n)$ of one or more configurations such that $(q, x) \vdash (q_1, x_1) \vdash (q_2, x_2) \vdash \cdots \vdash (q_n, x_n) \vdash (q', x')$.

Definition 2.2.12 *Given a FSA* $A = \langle Q, \Sigma, \delta, s, F \rangle$ *and string* $x \in \Sigma^*$, x *is* **accepted** *by* A *if and only if* $(s, x) \vdash^* (q, \epsilon)$ *for some* $q \in F$.

Essentially, a computation of a FSA on a given string x is a path p in the transition diagram for that FSA such that p starts at the vertex corresponding to s and the concatenation of the edge-labels of the edges in p is x; if the final vertex in p corresponds to a state in F, then x is accepted by the FSA.

Example 2.2.13 Consider the computation of the FSA in Figure 2.9 on several given strings. If the given string is $x = baabb$, x is accepted as there is a computation of the FSA on x which ends at state $q_3 \in F$.

$$\begin{array}{rcl}
(q_1, baabb) & \vdash & (q_1, aabb) \\
& \vdash & (q_2, abb) \\
& \vdash & (q_2, bb) \\
& \vdash & (q_3, b) \\
& \vdash & (q_3, \epsilon)
\end{array}$$

However, if $x = baabbab$, as the state q_4 in the final configuration is not in F, x is not accepted.

$$\begin{array}{rcl}
(q_1, baabbab) & \vdash & (q_1, aabbab) \\
& \vdash & (q_2, abbab) \\
& \vdash & (q_2, bbab) \\
& \vdash & (q_3, bab) \\
& \vdash & (q_3, ab) \\
& \vdash & (q_4, b) \\
& \vdash & (q_4, \epsilon)
\end{array}$$

■

Each FSA can be visualized as encoding a set of strings.

Definition 2.2.14 *Given a FSA $A = \langle Q, \Sigma, \delta, s, F \rangle$, the* **(regular) language** *L associated with A is the set of all strings that are accepted by A, i.e., $L = \{x \mid x \in \Sigma^*$ and $\exists q \in F$ such that $(s, x) \vdash^* (q, \epsilon)\}$.*

For instance, the regular language associated with the FSA in Figure 2.9 is the set of strings composed of zero or more b's, followed by one or more a's, followed by one or more b's. Finite-state automata can be classified into two types depending on the structure of the transition-relation δ. If δ has no entries of the form (q, ϵ, q') and is also a function, i.e., for each $q \in Q$ and $s \in \Sigma$ there is at most one state $q' \in Q$ such that $(q, s, q') \in \delta$, the FSA is **deterministic**; else, it is **nondeterministic**. For example, the reader can verify that the FSA in Figure 2.9 is deterministic. This thesis will be concerned primarily with deterministic FSA (abbreviated henceforth as **DFA**).

The observant reader will have noticed that none of the definitions given for FSA or FSA computation require that the transition relation be defined for every possible $q \in Q$ and $x \in \Sigma$. If a FSA has a transition relation δ such that for every $q \in Q$ and $x \in \Sigma$ there is a state $q' \in Q$ such that $(q, x, q') \in \delta$, the FSA is **total**; else, the FSA is **partial**. Though the majority of the constructions given in this thesis work whether the given FSA are partial or total, the constructions given in and subsequently based on that in part (5) of Theorem 4.5.4 require that the FSA be total; hence, without loss of generality, all FSA used in this thesis will be assumed to be total. If the reader is rankled by the baldness of this assumption, note that one can trivially transform a partial FSA into a total FSA by adding a new non-final "failure" state and modifying the transition relation such that all previously undefined transitions (including those originating from the new state) go to this new state. Indeed, it is the presence of just such a state (labeled, appropriately enough, $Fail$) that makes the DFA in Figure 2.9 total.

The other type of finite-state automaton of interest in this thesis operates on pairs of symbol-strings.

Definition 2.2.15 *A* **finite state transducer (FST)** *is a 6-tuple $\langle Q, \Sigma_i, \Sigma_o, \delta, s, F \rangle$ where Q is some set of states, Σ_i and Σ_o are the input and output alphabets, respectively, $\delta : Q \times \{\Sigma_i \cup \{\epsilon\}\} \times \{\Sigma_o \cup \{\epsilon\}\} \times Q$ is a transition relation, $s \in Q$ is the start state, and $F \subseteq Q$ is a set of final states.*

The transition diagram associated with a FST is much like that for a FSA except that arcs have labels from $\Sigma_i \cup \{\epsilon\} \times \Sigma_o \cup \{\epsilon\}$; these labels are written as $x : y$, where $x \in \Sigma_i \cup \{\epsilon\}$ and $y \in \Sigma_o \cup \{\epsilon\}$. A sample FST and its transition diagram are given in Figure 2.10. The notion of computation of a FST on a given pair of input strings can be stated formally in terms of configurations of a FST.

Definition 2.2.16 *A* **configuration of a FST** *$A = \langle Q, \Sigma_i, \Sigma_o, \delta, s, F \rangle$ is a triple (q, x, y) where $q \in Q$, $x \in \Sigma_i^*$, and $y \in \Sigma_o^*$.*

(a)

Q	$\Sigma_i \times \Sigma_o$			
	$a:c$	$a:d$	$b:c$	$b:d$
q_1	q_2	$Fail$	$Fail$	q_1
q_2	q_2	$Fail$	$Fail$	q_3
q_3	$Fail$	$Fail$	$Fail$	q_3
$Fail$	$Fail$	$Fail$	$Fail$	$Fail$

(b)

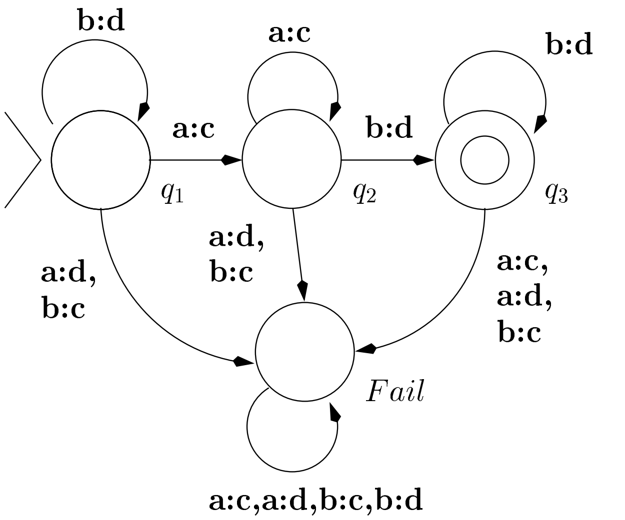


Figure 2.10: A Finite-State Transducer. The FST is specified by the 6-tuple $\langle Q, \Sigma_i, \Sigma_o, \delta, s, F\rangle$ where $Q = \{q_{,1}, q_2, q_3, Fail\}$, $\Sigma_i = \{a, b\}$, $\Sigma_o = \{c, d\}$, δ is defined by the table in (a) above, $s = q_1$ and $F = \{q_3\}$. The transition diagram for this FST is given in (b).

Definition 2.2.17 *Given two configurations (q, x, y) and (q', x', y') of a FST $A = \langle Q, \Sigma_i, \Sigma_o, \delta, s, F\rangle$, (q, x, y)* **yields** *(q', x', y') in one step, i.e., $(q, x, y) \vdash (q', x', y')$, if $x = w_i x'$ for some $w_i \in \Sigma_i \cup \{\epsilon\}$, $y = w_o y'$ for some $w_o \in \Sigma_o \cup \{\epsilon\}$, and $(q, w_i, w_o, q') \in \delta$.*

Again, let $\vdash^*$ represent the reflexive transitive closure of the yield relation.

Definition 2.2.18 *Given a FST $A = \langle Q, \Sigma_i, \Sigma_o, \delta, s, F\rangle$, a string $x \in \Sigma_i^*$, and a string $y \in \Sigma_o^*$, string-pair x/y is* **accepted** *by A if and only if $(s, x, y) \vdash^* (q, \epsilon, \epsilon)$ for some $q \in F$.*

As with FSA, computations in FST can be visualized as paths in their associated transition diagrams.

Example 2.2.19 Consider the computation of the FST in Figure 2.10 on given string-pairs. If the given string-pair is $x/y = baabb/dccdd$, x/y is accepted as the computation of the FST ends at state $q_3 \in F$.

$$\begin{array}{rcl}(q_1, baabb, dccdd) & \vdash & (q_1, aabb, ccdd) \\ & \vdash & (q_2, abb, cdd) \\ & \vdash & (q_2, bb, dd) \\ & \vdash & (q_3, b, d) \\ & \vdash & (q_3, \epsilon, \epsilon)\end{array}$$

However, if $x/y = baaa/dccc$, as the state q_2 in the final configuration is not in F, x/y is not accepted.

$$\begin{array}{rcl}(q_1, baaa, dccc) & \vdash & (q_1, aaa, ccc) \\ & \vdash & (q_2, aa, cc) \\ & \vdash & (q_2, a, c) \\ & \vdash & (q_2, \epsilon, \epsilon)\end{array}$$

∎

In addition to accepting string-pairs, a FST can also reconstruct strings that can pair with a given string to form a valid string-pair.

Definition 2.2.20 *Given a FST $A = \langle Q, \Sigma_i, \Sigma_o, \delta, s, F\rangle$ and a string $x \in \Sigma_i^*$, x is i-**accepted** by A if and only if there exists a string $y \in \Sigma_o^*$ such that $(s, x, y) \vdash^* (q, \epsilon, \epsilon)$ for some $q \in F$. Each such string y is called an i-**reconstruction** for x relative to A.*

The notions of o-acceptance and o-reconstruction can be defined analogously. Note that depending on the structure of the transition relation δ, there may be more than one i-reconstruction associated with a given string from Σ_i^* or more than one o-reconstruction associated with a string from Σ_o^*. By analogy with relation-based (see below) definitions in [KK94], let the set of i-reconstructions for a string $x \in \Sigma_i^*$ relative to a FST A be written as x/A and the set of o-reconstructions for a string $y \in \Sigma_o^*$ relative to A be written as A/y.

Example 2.2.21 Relative to the FST in Figure 2.10, string $x = baabb$ is i-accepted because there is a string $y = dccdd$ such that x/y is accepted by the FST. This string y can be constructed by concatenating the o-labels encountered while traversing the transition diagram for the FST using the i-labels relative to x. The reader can verify by the same sort of traversal argument that there is no i-reconstruction for $x = baaa$. ∎

Each FST can be visualized as encoding a set of string-pairs.

Definition 2.2.22 *Given a FST $A = \langle Q, \Sigma_i, \Sigma_o, \delta, s, F\rangle$, the* **(regular) relation** *R associated with A is the set of all string-pairs that are accepted by A, i.e., $R = \{x/y \mid x \in \Sigma_i^*, y \in \Sigma_o^*, \text{ and } \exists q \in F \text{ such that } (s, x, y) \vdash^* (q, \epsilon, \epsilon)\}$.*

For instance, the regular relation associated with the FST in Figure 2.10 is the set of string-pairs x/y in which x is composed of zero or more b's, followed by one or more a's, followed by one or more b's, and y is the version of x in which each a has been replaced by c and each b has been replaced by d. If each string-pair in a regular relation consists of strings of the same length (as in the preceding example), the relation is called a **same-length regular relation**. Like FSA, finite-state transducers can also be classified according to the structure of the transition-relation δ. However, in this case, there are several possible definitions of determinism [RRBP92, pp. 19–20]. The following types of determinism for FST will be of interest in this thesis:

1. ***i*- (*o*-) Deterministic FST**: For each $q \in Q$ and $s_i \in \Sigma_i$ ($s_o \in \Sigma_o$), there is at most one $s_o \in \Sigma_o$ ($s_i \in \Sigma_i$) and $q' \in Q$ such that $(q, s_i, s_o, q') \in \delta$. That is, the FSA created from the FST by taking the Σ_i-components (in the case of i-determinism) or the Σ_o-components (in the case of o-determinism) as the leaf labels is deterministic. Note that i-deterministic FST compute functions from input to output strings and o-deterministic FST compute functions from output to input strings. FST that are i-deterministic are also known as **sequential transducers** [Moh97].

2. ***i/o*-Deterministic FST**: For each $q \in Q$, $s_i \in \Sigma_i$ and $s_o \in \Sigma_o$, there is at most one $q' \in Q$ such that $(q, s_i, s_o, q') \in \delta$. That is, the FST is a DFA over the alphabet $\Sigma = \Sigma_i \times \Sigma_o$ of symbol-pairs.

Note that none of the types of determinism defined above allow ϵ to appear in δ. Each type of determinism has an associated type of nondeterminism, i.e., if a FST is not i-deterministic then it is i-nondeterministic. The reader can verify that the FST in Figure 2.10 is i-, o-, and i/o-deterministic. The following observations about the various kinds of deterministic FST will be of use in the analyses given in Chapter 4.

- All i- and o-deterministic FST are also i/o-deterministic FST; however, an i/o-deterministic FST is not necessarily either an i- or o-deterministic FST.

- The reconstruction-process relative to individual strings is only guaranteed to produce a single i- (o-) reconstruction for a string from Σ_i^* (Σ_o^*) if the FST is i- (o-) deterministic. If the FST is i/o-deterministic, reconstruction may associate a set of strings with the given string.

- It is known that a regular relation is same-length if and only it is associated with an ϵ-free FST [KK94, Lemma 3.3], and as $i-$, $o-$, and i/o-deterministic FST are ϵ-free, their associated regular relations are all same-length.

This thesis will be concerned primarily with i/o-deterministic FST.

A number of operations have been defined on single finite-state automata and pairs of finite-state automata [HU79, KK94, LP81, RS97b]. The following operations will be of particular interest in this thesis:

1. **FST composition**: Given a pair of string-relations $R_1 \subseteq \Sigma \times \Sigma'$ and $R_2 \subseteq \Sigma' \times \Sigma''$, the composition of R_1 and R_2 is the string-relation $R = \{x/y \mid x \in \Sigma^*,\ y \in \Sigma''^*$, and $\exists z \in \Sigma'^*$ such that $x/z \in R_1$ and $z/y \in R_2\}$. Given a pair of ϵ-free FST $A_1 = \langle Q_1, \Sigma_{i,1}, \Sigma_{o,1}, \delta_1, s_1, F_1\rangle$ and $A_2 = \langle Q_2, \Sigma_{o,1}, \Sigma_{o,2}, \delta_2, s_2, F_2\rangle$, the composition FST of A_1 and A_2 is the FST $A = \langle Q_1 \times Q_2, \Sigma_{i,1}, \Sigma_{o,2}, \delta, (s_1, s_2), F_1 \times F_2\rangle$ such that $\delta = \{((q_1, q_2), a, c, (q_1', q_2')) \mid \exists b \in \Sigma_{o,1}$ such that $(q_1, a, b, q_1') \in \delta_1$ and $(q_2, b, c, q_2') \in \delta_2\}$. An algorithm for constructing the composition FST of arbitrary FST is described in [PR97].

   By the construction above, the composition FST of any two ϵ-free FST is also ϵ-free. However, the composition FST of two i/o-deterministic FST is not necessarily i/o-deterministic. To see this, note that any two given i/o-deterministic FST whose transition relations contain transition-sets of the form $\{\{q_1, a, b, q_2\}, \{q_1, a, c, q_3\}\}$ and $\{\{q_1', b, d, q_2'\}, \{q_1', c, d, q_3'\}\}$ respectively will by the construction above have a composition FST whose transition-relation contains the transition-set $\{\{(q_1, q_1'), a, d, (q_2, q_2')\}, \{(q_1, q_1'), a, d, (q_3, q_3')\}\}$. As this composition FST has two transitions with the same label originating from the same state, this FST is not i/o-deterministic.

   Given two ϵ-free FST $A_1 = \langle Q_1, \Sigma_{i,1}, \Sigma_{o,1}, \delta_1, s_1, F_1\rangle$ and $A_2 = \langle Q_2, \Sigma_{o,1}, \Sigma_{o,2}, \delta_2, s_2, F_2\rangle$, the construction above creates a composition FST that has at most $|Q_1||Q_2|$ states and at most $(|Q_1||Q_2|)^2|\Sigma_{i,1}||\Sigma_{o,2}|$ transitions (assume that each set of transitions with the same label between the same state-pairs created during the construction is collapsed to a single transition). A naive implementation of this construction for ϵ-free FST compares each transition in A_1 against each transition in A_2; as there are at most $|Q_1||\Sigma_{i,1}||\Sigma_{o,1}||Q_1|$ and $|Q_2||\Sigma_{o,1}||\Sigma_{o,2}||Q_2|$ transitions in A_1 and A_2, respectively, this implementation runs in $O((|Q_1||Q_2|)^2 |\Sigma_{i,1}||\Sigma_{o,1}|^2|\Sigma_{o,2}|)$ time.

2. **Finite-state intersection**:

   (a) **FSA**: Given a pair of languages $L_1 \subseteq \Sigma^*$ and $L_2 \subseteq \Sigma^*$, the intersection of L_1 and L_2 is the language $L = \{x \mid x \in L_1$ and $x \in L_2\}$. Given a pair of FSA $A_1 = \langle Q_1, \Sigma, \delta_1, s_1, F_1\rangle$ and $A_2 = \langle Q_2, \Sigma, \delta_2, s_2, F_2\rangle$, the intersection FSA of A_1 and A_2 is the FSA $A = \langle Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2\rangle$ such that $\delta = \{(q_1, q_2), a, (q_1', q_2') \mid (q_1, a, q_1') \in \delta_1$ and $(q_2, a, q_2') \in \delta_2\}$.

   By the construction above, the intersection FSA of any two DFA is also a DFA. To see this, note that if the intersection FSA is not deterministic, then its transition relation must contain a transition-set of the form $\{\{(q_1, q_1'), a, (q_2, q_2')\}, \{(q_1, q_1'), a, (q_3, q_3')\}\}$. However, by the construction above, this would imply that the transition relations for the two given DFA contain the transition-sets $\{\{q_1, a, q_2\}, \{q_1, a, q_3\}\}$ and $\{\{q_1, a, q_2\}, \{q_1, a, q_3\}\}$ respectively, and that these DFA are not deterministic, which is a contradiction.

   Given two DFA $A_1 = \langle Q_1, \Sigma, \delta_1, s_1, F_1\rangle$ and $A_2 = \langle Q_2, \Sigma, \delta_2, s_2, F_2\rangle$, the construction above creates an intersection DFA that has at most $|Q_1||Q_2|$ states and at most $|Q_1||Q_2||\Sigma|$ transitions. A naive implementation of this construction for DFA compares each transition in A_1 against each transition in A_2; as there are at most $|Q_1||\Sigma|$ and $|Q_2||\Sigma|$ transitions in A_1 and A_2, respectively, this implementation runs in $O(|Q_1||Q_2||\Sigma|^2)$ time.

(b) **FST**: Given a pair of string-relations $R_1 \subseteq \Sigma \times \Sigma'$ and $R_2 \subseteq \Sigma \times \Sigma'$, the intersection of R_1 and R_2 is the string-relation $R = \{x/y \mid x/y \in R_1$ and $x/y \in R_2\}$. Given a pair of ϵ-free FST $A_1 = \langle Q_1, \Sigma_i, \Sigma_o, \delta_1, s_1, F_1 \rangle$ and $A_2 = \langle Q_2, \Sigma_i, \Sigma_o, \delta_2, s_2, F_2 \rangle$, the intersection FST of A_1 and A_2 is the FST $A = \langle Q_1 \times Q_2, \Sigma_i, \Sigma_o, \delta, (s_1, s_2), F_1 \times F_2 \rangle$ such that $\delta = \{((q_1, q_2), a, b, (q_1', q_2')) \mid (q_1, a, b, q_1') \in \delta_1$ and $(q_2, a, b, q_2') \in \delta_2\}$. Note that the intersection of two regular relations is not necessarily a regular relation. At present, it is known only that the intersection of two same-length relations is itself a regular relation [KK94] (see also discussion in Section 4.4.1).

By the construction above, the intersection FST of any two ϵ-free FST is also ϵ-free. Moreover, the intersection FST of any two i/o-deterministic FST is also i/o-deterministic. To see this, note that if the intersection FST is not i/o-deterministic, then its transition relation must contain a transition-set of the form $\{\{(q_1, q_1'), a, d, (q_2, q_2')\}, \{(q_1, q_1'), a, d, (q_3, q_3')\}\}$. However, by the construction above, this would imply that the transition relations for the two given FST contain the transition-sets $\{\{q_1, a, d, q_2\}, \{q_1, a, d, q_3\}\}$ and $\{\{q_1, a, d, q_2\}, \{q_1, a, d, q_3\}\}$ respectively, and that these FST are not i/o-deterministic, which is a contradiction.

Given two ϵ-free FST $A_1 = \langle Q_1, \Sigma_i, \Sigma_o, \delta_1, s_1, F_1 \rangle$ and $A_2 = \langle Q_2, \Sigma_i, \Sigma_o, \delta_2, s_2, F_2 \rangle$, the construction above creates an intersection FST that has at most $|Q_1||Q_2|$ states and at most $(|Q_1||Q_2|)^2|\Sigma_i||\Sigma_o|$ transitions. A naive implementation of this construction for ϵ-free FST compares each transition in A_1 against each transition in A_2; as there are at most $|Q_1||\Sigma_i||\Sigma_o||Q_1|$ and $|Q_2||\Sigma_i||\Sigma_o||Q_2|$ transitions in A_1 and A_2, respectively, this implementation thus runs in $O((|Q_1||Q_2|)^2(|\Sigma_i||\Sigma_o|)^2)$ time. If A_1 and A_2 are i/o-deterministic, they have at most $|Q_1||\Sigma_i||\Sigma_o|$ and $|Q_2||\Sigma_i||\Sigma_o|$ transitions, respectively and by reasoning analogous to that given above, their intersection FST will have at most $|Q_1||Q_2|$ states and at most $|Q_1||Q_2||\Sigma_i||\Sigma_o|$ transitions and can be created by a naive implementation of the construction above in $O(|Q_1||Q_2|(|\Sigma_i||\Sigma_o|)^2)$ time.

Note that the given upper bounds on the sizes of automata created by these operations are exact, in that automata may be created that have numbers of states and transitions that are equal to these upper bounds. Hence, though there exist implementations of some of these operations that can be much more efficient than the given naive implementations in certain applications, e.g., lazy composition [PR97], the worst-case running times of all such implementations are lower-bounded by the given upper bounds on the sizes of the created automata.

In the case of those operations defined above which create automata of exactly the same type as their given pair of automata, e.g., i/o-deterministic FST intersection, it is possible to define versions of those operations that take as input an arbitrarily large number of automata. Two possible ways of defining these operations are (1) extend the constructions given above relative to cross products on arbitrary numbers of rather than pairs of state-sets and (2) iterate the pairwise operations over the given set of automata, i.e., repeatedly remove two automata from the given set, apply the pairwise operation, and put the created automaton back in the set until only one automaton is left in the set. For the sake of simplicity, only alternative (2) will be considered in more detail here. In the case of intersection, the automata can be combined in a pairwise fashion in any order; however, as composition is sensitive to the order of its operands, e.g., the composition of A_1 and A_2 is not necessarily equivalent to the composition of A_2 and A_1, the automata must be combined in a specified order. For simplicity in the analyses below, assume that automata in a given set are combined in a pairwise manner relative to their order of appearance when that set is written down, i.e., given a set of automata $A = \{A_1, A_2, \ldots, A_k\}$, A_1 and A_2 will be combined to create A', A' and A_3 will be combined to create A'', and so on. Under this scheme, for a given set of automata $A = \{A_1, A_2, \ldots, A_k\}$ of the appropriate type such that $|Q|$ is the maximum number of states in any automaton in A and $|\Sigma|$ is the maximum number of symbols in any alphabet associated with an automaton in A, the time complexities of this iterative process relative to several pairwise operations on automata are derived as follows:

- **ϵ-free FST composition**: Given that the composition FST of two ϵ-free FST $A_1 = \langle Q_1, \Sigma_{i,1}, \Sigma_{o,1}, \delta_1, s_1, F_1\rangle$ and $A_2 = \langle Q_2, \Sigma_{o,1}, \Sigma_{o,2}, \delta_2, s_2, F_2\rangle$ in A can be computed in $O((|Q_1||Q_2|)^2|\Sigma_{i,1}||\Sigma_{o,1}|^2|\Sigma_{o,2}|) = O(|Q|^4||\Sigma|^4) = c|Q|^4|\Sigma|^4$ time for some constant $c > 0$, the composition FST of A can be computed in

$$\begin{aligned}\Sigma_{i=2}^{k} O(|Q|^{2i}|\Sigma|^4) &= c|\Sigma|^4\Sigma_{i=2}^{k}|Q|^{2i} \\ &\leq c|\Sigma|^4 k|Q|^{2k} \\ &= O(|Q|^{2k}|\Sigma|^4 k)\end{aligned}$$

  time.

- **DFA intersection**: Given that the intersection DFA of two DFA $A_1 = \langle Q_1, \Sigma, \delta_1, s_1, F_1\rangle$ and $A_2 = \langle Q_2, \Sigma, \delta_2, s_2, F_2\rangle$ in A can be computed in $O(|Q_1||Q_2||\Sigma|^2) = O(|Q|^2|\Sigma|^2) = c|Q|^2|\Sigma|^2$ time for some constant $c > 0$, the intersection DFA of A can be computed in

$$\begin{aligned}\Sigma_{i=2}^{k} O(|Q|^{i}|\Sigma|^2) &= c|\Sigma|^2\Sigma_{i=2}^{k}|Q|^{i} \\ &\leq c|\Sigma|^2\Sigma_{i=0}^{k}|Q|^{i} \\ &= c|\Sigma|^2(|Q|^{k+1} - 1)/(|Q| - 1) \\ &\leq c|\Sigma|^2|Q|^{k+1} \\ &= O(|Q|^{k+1}|\Sigma|^2)\end{aligned}$$

  time.

- **i/o-deterministic FST intersection**: Given that the intersection FST of two i/o-deterministic FST $A_1 = \langle Q_1, \Sigma_i, \Sigma_o, \delta_1, s_1, F_1\rangle$ and $A_2 = \langle Q_2, \Sigma_i, \Sigma_o, \delta_2, s_2, F_2\rangle$ in

Automaton Operation	Upper Bound on Automaton Size		Asymptotic Worst-Case Time Complexity of Automaton Creation
	States	Transitions	
ϵ-free FST composition	$\|Q\|^k$	$\|Q\|^{2k}\|\Sigma\|^2$	$O(\|Q\|^{2k}\|\Sigma\|^4 k)$, $\Omega(\|Q\|^{2k}\|\Sigma\|^2)$
DFA intersection	$\|Q\|^k$	$\|Q\|^k\|\Sigma\|$	$O(\|Q\|^{k+1}\|\Sigma\|^2)$, $\Omega(\|Q\|^k\|\Sigma\|)$
i/o-deterministic FST intersection	$\|Q\|^k$	$\|Q\|^k\|\Sigma\|^2$	$O(\|Q\|^{k+1}\|\Sigma\|^4)$, $\Omega(\|Q\|^k\|\Sigma\|^2)$
ϵ-free FST intersection	$\|Q\|^k$	$\|Q\|^{2k}\|\Sigma\|^2$	$O(\|Q\|^{2k}\|\Sigma\|^4 k)$, $\Omega(\|Q\|^{2k}\|\Sigma\|^2)$

Table 2.3: Characteristics of Iterated Finite-State Automaton Operations. This table gives the asymptotic worst-case time complexities of the operations of (as well as upper bounds on the sizes of automata created by) iterating various operations defined on pairs of automata over sets of automata, where k is the number of automata in the given set, $|Q|$ is the maximum number of states in any automaton in that set, and $|\Sigma|$ is the maximum number of symbols in any alphabet associated with an automaton in that set.

A can be computed in $O(|Q_1||Q_2|(|\Sigma_i||\Sigma_o|)^2) = O(|Q|^2|\Sigma|^4) = c|Q|^2|\Sigma|^4$ time for some constant $c > 0$, the intersection FST of A can be computed in

$$\begin{aligned}
\Sigma_{i=2}^{k} O(|Q|^i|\Sigma|^4) &= c|\Sigma|^4\Sigma_{i=2}^{k}|Q|^i \\
&\leq c|\Sigma|^4\Sigma_{i=0}^{k}|Q|^i \\
&= c|\Sigma|^4(|Q|^{k+1}-1)/(|Q|-1) \\
&\leq c|\Sigma|^4|Q|^{k+1} \\
&= O(|Q|^{k+1}|\Sigma|^4)
\end{aligned}$$

time.

- **ϵ-free FST intersection**: Given that the intersection FST of two ϵ-free FST $A_1 = \langle Q_1, \Sigma_i, \Sigma_o, \delta_1, s_1, F_1\rangle$ and $A_2 = \langle Q_2, \Sigma_i, \Sigma_o, \delta_2, s_2, F_2\rangle$ in A can be computed in $O((|Q_1||Q_2|)^2(|\Sigma_i||\Sigma_o|)^2) = O(|Q|^4|\Sigma|^4) = c|Q|^4|\Sigma|^4$ time for some constant $c > 0$, the intersection FST of A can be computed in

$$\begin{aligned}
\Sigma_{i=2}^{k} O(|Q|^{2i}|\Sigma|^4) &= c|\Sigma|^4\Sigma_{i=2}^{k}|Q|^{2i} \\
&\leq c|\Sigma|^4 k|Q|^{2k} \\
&= O(|Q|^{2k}|\Sigma|^4 k)
\end{aligned}$$

time.

The time complexities of these operations and upper bounds on the sizes of the created automata are given for each of these operations in Table 2.3. Note that if the number of states in each of the given automata is the same, the given upper bounds on the sizes of automata created by these operations are exact, in that automata may be created that have numbers of states and transitions that are equal to these upper bounds. Hence, though there exist implementations of some of these operations that can be much more efficient than the given naive implementations in certain applications [PR97, Tap97], the worst-case running times of all such implementations are lower-bounded by the given upper bounds on the sizes of the created automata.

Each of the types of finite-state automata described above can implement several phonological mechanisms.

1. Finite-state acceptors:

   (a) **Constraints on symbol-strings**: A constraint can be viewed as the set of strings satisfying that constraint; this set of strings can be encoded as the language associated with a FSA. A given string thus satisfies a constraint if that string is accepted by the FSA corresponding to that constraint.

   (b) **Symbol-string lexicons**: A lexicon can be viewed as a set of strings; this set of strings can be encoded as the language associated with a FSA. A given string is in the lexicon if that string is accepted by the FSA corresponding to that lexicon.

   (c) **Symbol-string lexical form creation**: The simplest way of creating lexical forms is to concatenate elements of the lexicon together, e.g., $denationalize = de + nation + al + ize$. This can be encoded by modifying the lexicon FSA that accepts the language D of symbol-strings in the lexicon to create another FSA that accepts some subset of D^+ that corresponds to the set of valid lexical forms.

2. Finite-state transducers:

   (a) **Constraints on pairs of symbol-strings**: A constraint on pairs of strings can be viewed as set of string-pairs that satisfies that constraint; this set of string-pairs can be encoded as the relation associated with a FST. A given string-pair thus satisfies a constraint if that string-pair is accepted by the FST corresponding to that constraint.

   (b) **Symbol-string rewriting rules**: A string rewriting rule can be viewed as a relation between input strings and the sets of strings generated by applying the rule to input strings; this relation can be encoded as the relation associated with a FST. A rule produces a string y from an input string x if y is a possible i-reconstruction of x relative to the FST corresponding to that rule.

The operations on automata defined above can be used to make such automata simulate the operation of their corresponding mechanisms in phonological computations. For instance, FST composition can be used to create cascaded sequences of rewriting rules and FST and FSA intersection can be used both to simulate simultaneous applications of constraints and to integrate the lexicon into the process of relating surface forms to lexical forms. All of the phonological theories examined in this thesis use automata to implement some (if not all) of their phonological mechanisms; the interested reader is referred to the relevant sections in Chapter 4 for details. The one use common to all of these theories is representing lexicons and word-formation processes as DFA – that is, the lexicon will be a set D of segment- or symbol-strings and will be specified in problem instances as a DFA $DFA(D)$ which recognizes the language D^+. Such a finite-state model is admittedly too simple because it does not capture known constraints on how lexical elements can be combined, e.g., *pod* + *s* = *pods* is a valid English word with multiple lexical elements but *s* + *pod* = *spod* is not, and it cannot easily account for some of the more exotic types of word-formation processes in human languages, e.g., infixation or reduplication [And94, Spr92]. However, finite-state models of word formation underlie many natural language processing systems [Spr92, p. 90], and analyses framed in terms of the simple finite-state model used in this thesis should hold both for these systems and systems based on more complex and realistic models.

In the parameterized analyses given in this thesis, the focus is on those phonological mechanisms that implement the relation between lexical and surface forms. These mechanisms will be characterized very broadly by aspects listed in Table 2.4 that capture the number of these mechanisms (constraints or rules) and some conception of the internal complexity of these mechanisms. An aspect of the latter type that is of particular interest is the **context-size** of a rule or constraint – that is, the size of the largest area of the phonological representation that an individual rule or constraint can operate on. For instance, if the representation is symbol-strings, the context-size is the length of the largest substring which a rule or constraint can examine and/or change. It is possible to characterize the context-size of a known automaton in terms of the state-relationship structure of that automaton, e.g., the concept of locality [Lap97, Section 14.5.3]. However, one may not always be able to independently assess the effects of bounded context-size and bounded automaton-size in conventional finite-state automata because, at least in some cases, the number of states and the context-size are correlated in such automata.

Example 2.2.23 Consider the following constraint with context size n, $n > 1$, on strings over the alphabet $\{0, 1\}$:

"String $x \in L$ *if and only if* every n-length substring of x contains exactly one 1-symbol."

Note that any string that satisfies this constraint must have adjacent 1's separated by exactly $n-1$ 0's. It can be shown that any FSA that accepts the language of strings specified by this constraint (and hence encodes this constraint) will require at least n states. To see this, suppose that an FSA A existed for this constraint which had $k < n$ states. Let $x = x_1x_2\ldots x_m$ $m > n$, be any string accepted by A that contains exactly two 1's, and let $s = q_1x_1q_2x_2q_3\cdots q_mx_mq_{m+1}$ be the sequence of alternating states and symbols that label the vertices and arcs of the directed path in the transition diagram for A that corresponds to one of the accepting computations of A on x. Consider the subsequence $s' = 1q_i0q_{i+1}0\ldots 0q_{i+(n-1)}1$ of s corresponding to the computation of A on the substring of x between and including the 1's. Note that there are n states in s'. As $k < n$, some state of A must appear twice in this sequence; call this state q'. Replace s' in s by the sequence s''

Phonological Theory	Characteristics
Simplified Segmental Grammars	Mechanism: Rewriting rules implemented as context-sensitive rewriting rules Aspects: • The number of rewriting rules • The maximum context-size of any rewriting rule • Various aspects characterizing the number of rewriting rules of particular types (see Section 4.1.2 for details)
FST-based rule systems	Mechanism: Rewriting rules implemented as i/o-deterministic ϵ-free FST Aspects: • The number of rule FST • The maximum number of states in any rule FST
The KIMMO system	Mechanism: Constraints implemented as i/o-deterministic ϵ-free FST Aspects: • The number of constraint FST • The maximum number of states in any constraint FST
Declarative Phonology & Optimality Theory	Mechanism: Constraints implemented as CDFA Aspects: • The number of constraint CDFA • The maximum number of states in any constraint CDFA • The maximum context-size of any constraint CDFA

Table 2.4: Characteristics of Mechanisms of Phonological Theories Examined in this Thesis. This table lists, for each phonological theory examined in this thesis, the type of mechanism used to implement relations between lexical and surface forms and the aspects of this mechanism relative to which that theory is analyzed. Note that the analyses for Declarative Phonology and Optimality Theory are done relative to contextual deterministic finite automata (CDFA); see main text for details.

which replaces the subsequence s''' of states and symbols between the occurrences of q' in s' by two copies of s'''. This modified sequence is also an accepting computation of A; however, the string accepted by this computation has more than $n-1$ 0's between the 1's, which is a contradiction. ∎

While further work is necessary to gauge exactly how often the situation in the example above occurs, this example does establish that it can occur. Hence, in order to investigate the context-size of constraints or rules in a manner independent of the number of states in the associated finite-state automata, it will be necessary to define new types of automata. Consider the following definitions relative to FSA.

substring #	substring								DFA accept	DFA reject
	b	a	a	b	a	b	b	a	accept	reject
1	b	a	a							$\surd$
2		a	a	b					$\surd$	
3			a	b	a					$\surd$
4				b	a	b			$\surd$	
5					a	b	b		$\surd$	
6						b	b	a		$\surd$
									3	3

Figure 2.11: Operation of a Contextual Deterministic Finite-State Automaton. The underlying DFA is that given in Figure 2.9 and the context size is 3. Given the string $x = baababba$, the result of the application of the CDFA to x is 3, i.e., 3 out of 6 substrings of length 3 are not accepted.

Definition 2.2.24 *A* **contextual finite-state automaton (CFSA)** *is a 6-tuple* $\langle Q, \Sigma, \delta, s, F, c\rangle$ *such that* $A = \langle Q, \Sigma, \delta, F, s\rangle$ *is a finite-state acceptor and and* c *is a positive integer. The FSA associated with a CFSA* A *is the* **underlying FSA of** A *and* c *is the* **context-size of** A. *A CFSA whose underlying FSA is deterministic is a* **contextual deterministic finite-state automaton (CDFA)**.

Definition 2.2.25 *Given a string* $x = x_1x_2\cdots x_n$ *over an alphabet* Σ *and an integer* c *such that* $0 < c \leq n$, *the* c**-decomposition of** x *is the set of strings* $S_{x,c} = \{y_i \mid 1 \leq i \leq (n-c)+1 \text{ and } y_i = x_ix_{i+1}\cdots x_{i+(c-1)}\}$.

Definition 2.2.26 *Given a CFSA* $A = \langle Q, \Sigma, \delta, s, F, c\rangle$, *the* **result function associated with** A *is the function* $res_A : \Sigma^* \mapsto \mathcal{N}$ *such that for a string* $x \in \Sigma^*$, $res_A(x) = |\{\ y \mid y \in S_{x,c}$ *and* y *is not accepted by the underlying FSA of* $A\}|$. *Given a string* $x \in \Sigma^*$, *the* **result of applying** A **to** x *is* $res_A(x)$.

The operation of a CFSA based on the DFA from Figure 2.9 is shown in Figure 2.11. As defined above, a CFSA is not a conventional finite-state automaton with a well-defined notion of computation and an associated language but rather an abstract specification of a procedure for evaluating strings which can be used to define string acceptance and languages. One such set of definitions that will be used later in this thesis is as follows.

Definition 2.2.27 *Given a CFSA* $A = \langle Q, \Sigma, \delta, s, F, c\rangle$ *and a string* $x \in \Sigma^*$, x **is accepted by** A *if and only if* $res_A(x) = 0$.

Example 2.2.28 Consider the CDFA A with context-size 3 whose underlying DFA is that given in Figure 2.9. As shown in Figure 2.11, A does not accept the string $x_1 = baababba$ as

$res_A(x_1) = 3$. However, the reader can verify that strings $x_2 = aab$, $x_3 = babb$, and $x_4 = ba$ are accepted by A as $res_A(x_2) = res_A(x_3) = res_A(x_4) = 0$. Note that acceptance is trivial in the case of x_4 as the 3-decomposition of x_4 is empty. ∎

Definition 2.2.29 *Given a CFSA $A = \langle Q, \Sigma, \delta, s, F, c \rangle$, the* **language** *$L$ associated with A is the set of all strings that are accepted by A, i.e., $L = \{x \mid x \in \Sigma^* \text{ and } res_A(x) = 0\}$.*

Languages associated with CFSA may in turn have associated FSA. For example, a construction for creating a DFA for the language associated with a CDFA as defined above is given in Part (7) of Theorem 4.5.4. Contextual finite-state automata will be used in the analysis of Declarative Phonology and Optimality Theory given in Chapter 4 to model constraints with bounded context-size. Analogous definitions for FST which can be applied to finite-state implementations of rewriting rules are not obvious, and are a topic for future research.

Chapter 3

Computational Analyses of Phonological Theories

In part because of the integral role of phonology in processing speech and written text, there have been a number of implementations of various phonological theories as modules in natural language processing systems (see [Col95, Spr92] and references). This work is important because it shows that these theories can be implemented computationally; however, such systems and the algorithms encoded in them do not characterize the full range of algorithmic possibilities and limitations latent within these theories. Such a characterization requires a more formal and abstract computational analysis. In this chapter, I will give a brief overview of the two main approaches to the computational analysis of linguistic theories and discuss how the analysis of the computational complexity of linguistic theories can be improved by incorporating the techniques of parameterized complexity analysis.

There are two main approaches to the formal computational analysis of theories of natural language processing. Each of these approaches focuses on a different measure of the computational power encoded within the mechanisms postulated by such a theory. In the following, let a **mapping system** be a particular specification of the mechanisms postulated by a linguistic theory that relate the representations manipulated by that theory. For instance, a mapping system could be either a formal grammar that links mental representations and derivation trees in a theory of syntax or a collection of rules or constraints that relate lexical and surface forms in a theory of phonology.

1. **Generative Capacity**: The generative capacity of a linguistic theory is the set of formal languages or relations that can be generated by mapping systems within that theory. In the case of formal languages, this set is often summarized by considering the most powerful type of formal language within this set relative to the Chomsky hierarchy of formal languages:

   (a) Regular Languages (Type 0; generated by regular grammars / finite-state automata);

(b) Context-free languages (Type 1; generated by context-free grammars / pushdown automata);
(c) Context-sensitive languages (Type 2; generated by context-sensitive grammars / bounded-tape Turing machines); and
(d) Turing-equivalent languages (Type 3; generated by unrestricted grammars / Turing machines)

In the case of relations, this set may be similarly summarized relative to the known hierarchy of relation-classes and their associated automata [Brs79, RS97b]. The generative capacity of a linguistic theory is a broad characterization of types of linguistic phenomena that can be encoded within that theory (and, by the equivalence of language-types and generating mechanisms, a rough measure of the computational power encoded in that theory in terms of the types of automata needed to implement the mapping systems implicit in that theory).

This approach originated in Chomsky's classic 1957 work *Syntactic Structures* [Cho57], in which he defined the Chomsky hierarchy and used it to examine the generative capacities of various theories of English syntax. This approach, both in terms of formal-language characterizations of various linguistic theories and phenomena as well as algorithmic work on efficient parsing of various types of formal-language grammars, has been the dominant approach to computational analysis of natural language (see [Ber84, Gaz85] and references). Examples of such analyses within computational phonology include Johnson's characterization of the automata required to implement *SPE*-style phonological rules under various rule-application schemes [JoC72], Ritchie's characterization of the relations encoded within sets of KIMMO-style Two-Level rules [Rit92], and the investigations by Eisner [Eis97a], Ellison [Ell94, Ell95], Frank and Satta [FS97], and Karttunen [Kar98] into the types of constraints that can and cannot be accommodated within a finite-state implementation of Optimality Theory.

2. **Computational Complexity**: The computational complexity of a linguistic theory is the computational complexity (phrased in terms of both algorithms and intractability results) of various search problems defined within that theory. For reasons outlined in Section 2.1.1, these analyses are often done on decision problems associated with these search problems. Of typical interest are problems in which mapping systems for a linguistic theory **X** access the representations postulated by **X** in the following natural ways:

   (a) **Encoding Problems**: Problems that determine the surface forms of given lexical forms. There are several types of encoding problems:

   X-ENCODE-I
   Instance: A mapping system G for theory **X**, a lexical form u, and a surface form s.
   Question: Does G produce s when it is applied to u?

X-ENCODE-II
Instance: A mapping system G for theory **X** and a lexical form u.
Question: Does G produce a surface form that satisfies some property P when it is applied to u?

X-ENCODE-III
Instance: A mapping system G for theory **X** and a lexical form u.
Question: Does G produce a surface form when it is applied to u?

The first two types of encoding problem are appropriate for analyzing theories that always produce surface forms, e.g., Simplified Segmental Grammars (Section 4.1), FST-based rule systems (Section 4.3), or Optimality Theory (Section 4.6). The third type of problem is more appropriate for analyzing constraint-satisfaction based systems in which a given lexical form may or may not have associated surface forms, e.g., the KIMMO system (Section 4.4) or Declarative Phonology (Section 4.5).

(b) **Decoding Problems**: Problems that determine the lexical forms associated with a given surface form, e.g.,

X-DECODE
Instance: A mapping system G for theory **X** with an associated lexicon D and a surface form s.
Question: Is there a lexical form u that is generated relative to D such that the result of applying G to u is s?

Encoding and decoding problems are also referred to as generation and comprehension problems, as these problems correspond informally to the operations of natural language generation and comprehension, i.e., creating surface forms from lexical forms and recovering lexical forms from surface forms, respectively. However, I will follow the naming conventions above established by Ristad [Ris93b], as this will constantly remind the reader both that the problems defined here only manipulate mathematical objects within a particular formal framework and that the relevance of these frameworks and results derived using them to natural language processing must be established separately ([Ris93b, Footnote 2]; see also [MaR95]). One can also define problems that correspond to the learning of natural language grammars and lexicons [Ber85, JoM92, TS96, WC80]. The computational complexity of a linguistic theory is a direct characterization of the computational power encoded within that theory in terms of the amounts of computational resources required to (and hence the possible types of algorithms that can) implement operations within that theory.

This approach was developed by Berwick in various papers published in the late 1970's and early 1980's (summarized in [Ber85, BW84]) which focused on the computational complexity associated with various theories of syntax. Perhaps the clearest statement of how to use this approach in practice is given in [BBR87].

Examples of such analyses within computational phonology include work done by Barton on the KIMMO system [Bar86, BBR87], Ristad on various formalizations of segmental [Ris90, Ris93b] and autosegmental [Ris94] phonology, Eisner [Eis97a] and Wareham [War96a, War96b] on Optimality Theory, Wareham [War96a] on Declarative Phonology, and Johnson on the learning of rules within segmental phonology [JoM92].

Though certain authors have advocated using computational complexity instead of generative capacity for describing and analyzing the computational power of linguistic theories [BBR87, Ber89], each approach characterizes the computational power of a linguistic theory in a different way, and the choice of which approach to use in a particular situation depends on the investigator's goals. If one is interested in characterizing a linguistic theory in terms of the formal languages that it can generate or the types of automata that can implement it, then that theory should be evaluated in terms of its generative capacity. However, if one is more concerned with how efficiently that theory can be implemented relative to any computational model, then an analysis in terms of computational complexity is preferable.

Now that these two approaches have been introduced, the remainder of this chapter will focus on analyses of the computational complexity of linguistic theories – in particular, the manner in which these analyses are done, critiques of these analyses, and the role that the techniques of parameterized complexity analysis can play in improving these analyses.

An analysis of the computational complexity of a linguistic theory can be broken down into five phases:

I. Formalize the linguistic theory of interest in a particular computational framework of well-defined and fully-specified representations and mechanisms, and establish the assumptions under which computational results derived relative to this framework apply back to the original theory.

II. Define computational problems of interest within this framework.

III. Analyze the computational complexity of these problems.

IV. Relate results derived in (III) to the framework.

V. Relate the results as interpreted within the framework back to the linguistic theory.

Let us consider some of the characteristics of each phase in turn.

- Phase I is perhaps the most crucial part of this process. It may require less work if the linguistic theory of interest is already stated formally (as is the case with segmental phonological grammars [CH68] and the KIMMO system); however, some simplification and abstraction is always involved in order to focus the analysis on those aspects of the theory that are of particular interest to the investigation. The defined framework must satisfy the often contradictory requirements of rigor and relevance:

  - It must be simple enough to analyze but complex enough to reflect important aspects of the linguistic theory.
  - It must be well-defined and abstract enough to make analysis possible but not so abstract that derived results are irrelevant to the linguistic theory.
  - It must allow the derivation of results that are simple enough to be comprehensible but complex enough to have nontrivial implications for the linguistic theory.

  Some sample derivations of such frameworks for various linguistic theories are given in [Ber85, BBR87, BW84, Ris90, Ris93a, Ris94]. Formal frameworks that satisfy the requirements given above often seem to be a bewildering mix of formal simplifications and informal justifying assumptions, and their creation and interpretation has more the flavor of an arcane art than scientific method. Some consolation can be taken in realizing that these difficulties seem to be an unavoidable part of building mathematical models of any natural phenomenon, be it another human cognitive ability like vision [Tso90, Tso93, Tso95], the three-dimensional folding of proteins [NMK94], or the hydrology and geochemistry of landmasses [OSB94]; if the creation of appropriate frameworks is an art, it is at least one that can draw on similar practice and experience within many disciplines besides linguistics.

  Note that at this point in the analysis, simplifications and justifying assumptions must also be made for the actual methods that will be used to do the computational analysis. For example, as noted in Section 2.1.1, computational complexity analyses are often simplified to deal only with asymptotic worst-case complexity relative to deterministic Turing machines. The usual justifying assumption in this case is to invoke simplicity in order to be able to do the analysis at all; the use of more complex justifications is fraught with peril.[1] The interested reader is referred to [BBR87, BW83, BW84, Rou91, War96a] for more detailed discussions of the assumptions under which computational complexity results can be interpreted within linguistics (see also [Tso90, Tso93, Tso95] for a similar perspective relative to vision).

- Phase II is an extension of Phase I, in that it further focuses the scope of the analysis by defining problems that embody both the operations in the framework that

[1] The alleged correspondence of computational complexity to measured human linguistic performance is a particularly insidious justifying assumption, in that it implies that tractable language processing problems can be solved quickly by humans and intractable language processing problems cannot be solved quickly by humans. Hence we get what Ristad [Ris93a] has called Cordemoy's Paradox: If language processing is polynomial-time intractable, how can humans process language so fast? The problem here is that measures of asymptotic worst-case complexity and observed human linguistic performance are not directly comparable by virtue of the wildly different architectures underlying general-purpose algorithmic and human neural computing systems and the fundamentally different sets of inputs in each case, e.g., asymptotically large and worst-case on one hand and small and (perhaps) average-case on the other ([BW83, BW84]; see also [War96a]).

It is interesting that similar paradoxes based on similarly flawed correspondences have also arisen when computational complexity has been used to analyze other natural phenomena such as protein folding and vision. The interested reader is strongly encouraged to read [NMK94] (especially Section 5) and [Tso90, Tso93, Tso95] for insightful perspectives on such paradoxes and their resolution.

are of interest and those aspects of the framework that will be examined. If the intent of the analysis is to investigate a particular problem within the framework such as a problem arising in an practical implementation of the linguistic theory, this phase is straightforward; however, if the intent is to characterize the theory in general, there is much more choice involved, e.g., the encoding and decoding problems defined earlier in this chapter. Once again, this is somewhat of an art that can draw on the experience of many disciplines. A particularly interesting discussion of how problems should be defined for computational complexity analyses within linguistics is given in [Ris93a, Chapters 1 and 2]; for similar discussions relative to vision, see [Tso90, Tso93, Tso95].

As results derived later in the analysis will be phrased in terms of the problems defined during this phase, problems must be defined very carefully in order to obtain relevant results. The assumptions relating results within the framework to the original theory are frequently stated in terms of correspondences between representations and mechanisms in the linguistic theory and those in the problems defined in this phase (if the theory is stated informally) or as formally-defined reductions between the problems in the theory and the problems defined in this phase (if the theory is stated formally).

- Phase III is perhaps technically the most demanding but conceptually the easiest – the problems have been stated and they have to be analyzed. A potentially confusing aspect of this phase is that complexity analyses often work relative to either restricted or generalized versions of the problems defined in Phase II. This is a product of the way complexity results derived for one problem propagate to related problems. Given an efficient algorithm for a problem Π, that algorithm also works for (and thus establishes the tractability of) appropriately restricted versions of Π; conversely, a proof of the intractability of Π also establishes the intractability of any problem which has Π as a restricted version of that problem. Thus, by establishing the tractability of the most general possible versions of a problem and the intractability of the most restricted possible versions of that problem, individual results can be made to have the widest impact on characterizing the computational complexity of that problem. Such results are therefore merely a way of summarizing and saving time in deriving the set of results underlying a comprehensive analysis of a problem, and should not themselves be misconstrued as comprising the only results of that analysis.

- In Phases IV and V, the "raw" algorithms and hardness and completeness results derived in Phase III are applied in some manner to say something about the framework and then the original linguistic theory. Phase V tends to be specific to the assumptions relating the theory and the framework derived in Phase I and will not be discussed here further. Phase IV, however, has some interesting general aspects. There seem to be three approaches to applying raw complexity-theoretic results within a framework:

    1. As a tool for establishing the polynomial-time intractability of individual problems. This follows directly from hardness and completeness results.

2. As a tool for ranking and classifying linguistic theories [BBR87, Ris90, Ris93a]. This follows from hardness and completeness results and known hierarchical inclusion relations between the complexity classes for which these results were derived (see [JoD90] for a list of known complexity classes and their relationships).
3. As a probe for discovering the sources of polynomial-time intractability in linguistic theories [BBR87, Ris93a]. This is typically done by examining the reductions used to prove hardness results for the various problems examined and selecting aspects of those problems that seem to be important to proving hardness and hence to polynomial-time intractability, e.g., those aspects that are required to have unbounded values in order for the reduction to work ([BBR87, KC88, Rou87, Spr92]; see also discussions in Sections 4.4.3 and 4.6.3).

The third application is of particular interest for how it can inform us not only about the set of non-polynomial time algorithms available for implementing the examined problems (as discussed in Section 2.1.3) but also about the relationship between descriptive adequacy and computational power in the framework (and, if Phase I was done correctly, the linguistic theory of interest). To the extent that aspects responsible for polynomial-time intractability characterize the types of phenomena that can be dealt with by a linguistic theory and the level of detail with which such phenomena can be described, knowing the sources of polynomial-time intractability of problems associated with that theory can help us assess the computational consequences of various manners of describing linguistic phenomena within that theory. These consequences may then be used to guide future research in (and possibly revisions to) that theory.

Such a research process seems to have been in action with respect to Transformational Grammar over the last 30 years [BF95]: The various failures to implement the original theory of Transformational Grammar in an efficient parser in the mid-1960's led to 25 years of collaborative linguistic and computational research into restrictions on linguistic and algorithmic mechanisms which culminated in the development of both a formally constrained but nonetheless empirically adequate theory of Transformational Grammar and an implementation of this theory in an efficient natural language parser. This research process has been formalized as "the language complexity game" by Ristad in [Ris93a], wherein he shows how linguistic and computational research can be applied iteratively to derive a theory of anaphoric reference that has the lowest computational complexity and the highest descriptive power, cf. [MaR95] (see also [Tso90, Tso93, Tso95] for a description of a similar research process rooted in the physical realizability of derived algorithms). Such examples highlight the importance of this application of complexity-theoretic results in particular and the interaction of linguistics and computer science in general.

It is very important to realize that by virtue of the justifying assumptions made in Phases I and II and used in V, the computational analyses in Phases III and IV can be carried out independently of any considerations about the linguistic theory being analyzed –

that is, the techniques used and the results derived in Phase III and IV need only be faithful to the conventions of computational complexity theory, and need not mirror or correspond to any aspect of natural language processing in humans. This point will be particularly relevant in the discussion of critiques of complexity-theoretic analyses of linguistic theories given below.

The analyses of phonological theories given in Chapter 4 are examples of the computational analyses described above. All are stated in terms of simplified and somewhat abstract frameworks that do not directly correspond to versions of these theories that are typically used in practice. As the phonological theories examined in that chapter are all defined more or less formally, the assumptions linking the results derived in the analyses to these theories are for the most part stated as implicit reductions from the restricted problems studied here to general operations within these theories. It is important to realize that the restrictions on the phonological theories examined here, such as the simplified autosegmental representations used in the formulations of Declarative Phonology and the restriction on all theories to deal exclusively with mechanisms that map lexical forms onto surface forms of the same size, are meant not to accurately and directly model these theories but are rather a convenient way of simplifying the problems involved such that they can be analyzed and the relationships between various aspects in these problems (and hence in the associated phonological theories) can be more easily recognized. If the problems examined in this thesis have been defined correctly, various kinds of intractability results will automatically propagate to various operations in the associated phonological theories. Though algorithms for the restricted problems studied here do not propagate so readily, they may yet be useful in suggesting strategies for implementing those operations.

There have been many critiques of complexity-theoretic analyses in linguistics (see also [Tso90, Tso93] and references for critiques of complexity-theoretic analyses of vision). The criticisms seem to fall into three groups:

1. The assumptions linking the formal framework and the linguistic theory are flawed, e.g., the model of natural language processing implicit in the framework is incomplete or is contradicted by empirical language data [KC88, MaR95].

2. The assumptions underlying computational complexity theory (and hence computational complexity analysis) are unrealistic, e.g., asymptotic worst-case complexity measures do not adequately model either the small finite inputs or the bounded computational power of the human brain that characterize natural language processing in humans [BW84, MaR95, Rou91, War96a].

3. The assumptions underlying the derived results are unrealistic, e.g., the instances of linguistic systems produced by reductions do not correspond to natural linguistic systems and thus invalidate the applicability of any derived intractability results to natural language processing in humans [KC88, Rou87, Spr92].

When considered in light of the five-phase analysis process described above, these criticisms are not as damaging as they may first appear to be. Criticisms of the first kind are valid for the particular frameworks they address, but ultimately point out only that one must be careful when constructing those frameworks and relating both those frameworks and results derived within them to linguistic theories. Criticisms of the second kind would be valid if (as noted in the descriptions of Phases I and II above) the simplifications they deride were not necessary to do formal analysis of the computational power of natural language systems. That such simplifications need to be made is undeniable; to be truly effective, criticism of extant simplifications would have to present a more reasonable alternative set of simplifications, which none has to date. Criticisms of the third kind are misguided; as noted above, neither reductions or results in Phase III need to be consistent with the reality of natural language processing, and hence cannot of themselves be unrealistic. Plainly put, any correct mathematically derived result is valid and does not in itself imply anything that can be realistic or unrealistic. Criticism based on realism should rather be directed at the justifying assumptions underlying the framework that relate results to a linguistic theory, and the way those assumptions are used to interpret formal results relative to the linguistic theory in Phase V.

It is interesting that perhaps the most damning criticism to date stems from the portion of the analysis not addressed by any previous criticism – namely, Phase IV and the interpretation of complexity theoretic results in the formal framework. In particular, consider the second and third of the three applications of complexity-theoretic results within Phase IV that were described several pages back. Using hardness and completeness results to rank problems associated with linguistic theories is perhaps justifiable (though one must wonder just how valid such rankings obtained by collapsing all information about the set of algorithms for a problem into single quantity can be, or indeed what they really mean). However, the sources of polynomial-time intractability in problems cannot be discovered using aspects derived from the reductions underlying classical hardness results. Indeed, this frequently-used approach is doubly flawed:

1. There may be many possible reductions that establish the polynomial-time intractability of a problem, and each of these reductions may employ different sets of aspects to encode instances of one problem into instances of another. It is not clear that an aspect that is crucial in the operation of a particular reduction will also be important in any other reduction, let alone all of them. Hence, it is not clear how an aspect selected in such a manner could be considered to be responsible for the polynomial-time intractability of a problem.

2. A reduction can only show that a particular type of algorithm does *not* exist for a problem. It cannot in itself prove the existence of an algorithm, let alone prove that any aspect that is crucial to the operation of that reduction has an associated algorithm whose non-polynomial time complexity is purely a function of this aspect (and hence that this aspect can potentially be exploited to in subsequent applications and research as described in Section 2.1.3).

This is, granted, a flawed use of a sound technique; as noted in Section 2.1.2, it is possible to structure a classical complexity-theoretic analysis (albeit in a somewhat cumbersome manner) to search for the sources of polynomial-time intractability in a problem. However, the issues noted above do highlight disturbing technical and conceptual deficits in current approaches to the computational analysis of linguistic theories – that is, technically, classical theories of computational complexity like NP-completeness are not designed to investigate increasingly important questions concerning individual aspects and their roles as sources of polynomial-time intractability, and conceptually, the lack of an appropriate theory to answer these questions seems to have stunted the formulation of realistic notions about the computational complexity of a problem, such as what a source of polynomial-time intractability is or what classical polynomial-time intractability results really mean.

It is precisely these deficits that the theory of parameterized computational complexity addresses. It addresses the conceptual deficit by putting forward definitions that explicitly acknowledge the role of individual aspects (via their selection by parameters) in problem complexity, thus allowing realistic conceptions of the set of non-polynomial time algorithms associated with a polynomial-time intractable problem and the sources of polynomial-time intractability for such a problem. It addresses the technical deficit by defining the appropriate formal machinery, i.e., the parametric reducibility and the rich set of complexity classes forming the W hierarchy, for investigating the sources of polynomial-time intractability in problems via systematic parameterized complexity analysis. Thus, parameterized computational complexity theory (as embodied in systematic parameterized complexity analysis) fills an important void in current analyses of the computational complexity of linguistic theories, and should be added to the list of techniques employed in such analyses.

In conclusion, parameterized computational complexity theory satisfies known technical and conceptual needs in the analysis of the sources of polynomial-time intractability of computational problems associated with linguistic theories in particular and of computational problems in general. It is my hope that the systematic parameterized complexity analyses of five phonological theories given in the next chapter will amply illustrate the promise of parameterized computational complexity theory, and will serve as exemplars on how to do and interpret the results of such analyses for computational problems in other disciplines.

Chapter 4

A Systematic Parameterized Complexity Analysis of Phonological Processing under Rule- and Constraint-Based Formalisms

This chapter contains systematic parameterized complexity analyses of five phonological theories, namely Simplified Segmental Grammars (Section 4.1), FST-based rule systems that operate by FST composition (Section 4.3), the theory of Two Level Morphology as implemented by the KIMMO system (Section 4.4), Declarative Phonology (Section 4.5), and Optimality Theory (Section 4.6). Each section will consist of an overview of a particular theory and a review of previous complexity-theoretic analyses of that theory, a systematic parameterized analysis relative to some set of aspects, and a discussion of the implications of this analysis. Many of the results in these analyses are derived by using reductions from the BOUNDED DFA INTERSECTION (BDFAI) problem (Section 4.2). This chapter concludes with a brief discussion of some patterns in the results derived in the preceding sections and the implications of these patterns for phonological processing in general.

The analyses in this chapter will be done relative to the appropriate parameterized versions of the encoding and decoding problems introduced in Chapter 3 which will be further restricted to operate on lexical and surface forms that are of the same length, i.e., mechanisms that add portions to or delete portions from the given form are not allowed. Hardness results will be derived by a chain of reductions from four basic problems: BOUNDED DFA INTERSECTION, BOUNDED NONDETERMINISTIC TURING MACHINE ACCEPTANCE, DOMINATING SET, and LONGEST COMMON SUBSEQUENCE. This chain of reductions is shown in Figure 4.1. The analysis of each theory **X** $\in$ {**SSG, FST, KIM, DP, OT**} will be followed by a discussion of the derived results. Each such discussion will consider the following issues:

1. **The sources of polynomial-time intractability in X-**ENCODE **and X-**DECODE: These sources will be listed and (where possible) characterized in terms of particular mechanisms in theory **X**. This portion of the discussion will also

examine the justifications for including in **X** the mechanisms underlying these sources of polynomial-time intractability to see if the computational complexity of problems associated with **X** can be reduced by modifying or eliminating these mechanisms, and compare results derived here with previous speculation about the sources of polynomial-time intractability in **X**.

2. **The computational complexity of the versions of X-ENCODE and X-DECODE that allow additions and deletions**: Where such problems can be meaningfully defined, this part of the discussion will involve brief sketches of how results derived here apply to these more general problems.

3. **The computational complexity of search problems associated with X-ENCODE and X-DECODE**: This part of the discussion will focus on the following three search problems, which are phrased as functions:

   - $\text{CHK}_{\mathbf{X}}(g, u, s)$: Given a lexical form u, a surface form s, and a phonological mapping system g for theory **X**, return **True** if s can be produced from u under g, and **False** otherwise.
   - $\text{ENC}_{\mathbf{X}}(g, u)$: Given a lexical form u and a phonological mapping system g for theory **X**, return *any* surface form s that g produces from u if such an s exists, and special symbol $\perp$ otherwise.
   - $\text{DEC}_{\mathbf{X}}(g, s)$: Given a surface form s and a phonological mapping system g for theory **X**, return *any* lexical form u such that g produces s from u if such a u exists, and special symbol $\perp$ otherwise.

   The encoding and decoding problems associated with a phonological theory may have multiple solutions, i.e., multiple surface forms may be associated with a given lexical form or multiple lexical forms may be associated with a given surface form. The search problems defined above are a first step in investigating the complexity of accessing the elements in these solution-sets. Problems ENC and DEC access this set in the simplest possible way, i.e., see if it is empty and if not, return an arbitrary element. Problem CHK is the simplest problem that accesses particular elements in this set, i.e., see if a particular element is in the set. More complex search problems are possible, and are a topic for future research.

4. **Applications of derived results**. This part of the discussion will look at the implications of the results derived here for implementations of theory **X** and for related phonological theories.

5. **Future research**. This part of the discussion will list topics specific to theory **X** that show promise for future research, and (where possible) list connections of this future research to other theories examined in this thesis.

The intent of these discussions is both to interpret the results derived for each theory and to illustrate various points made in Section 2.1.3 and Chapter 3 about how systematic parameterized complexity analysis can and should be used in practice.

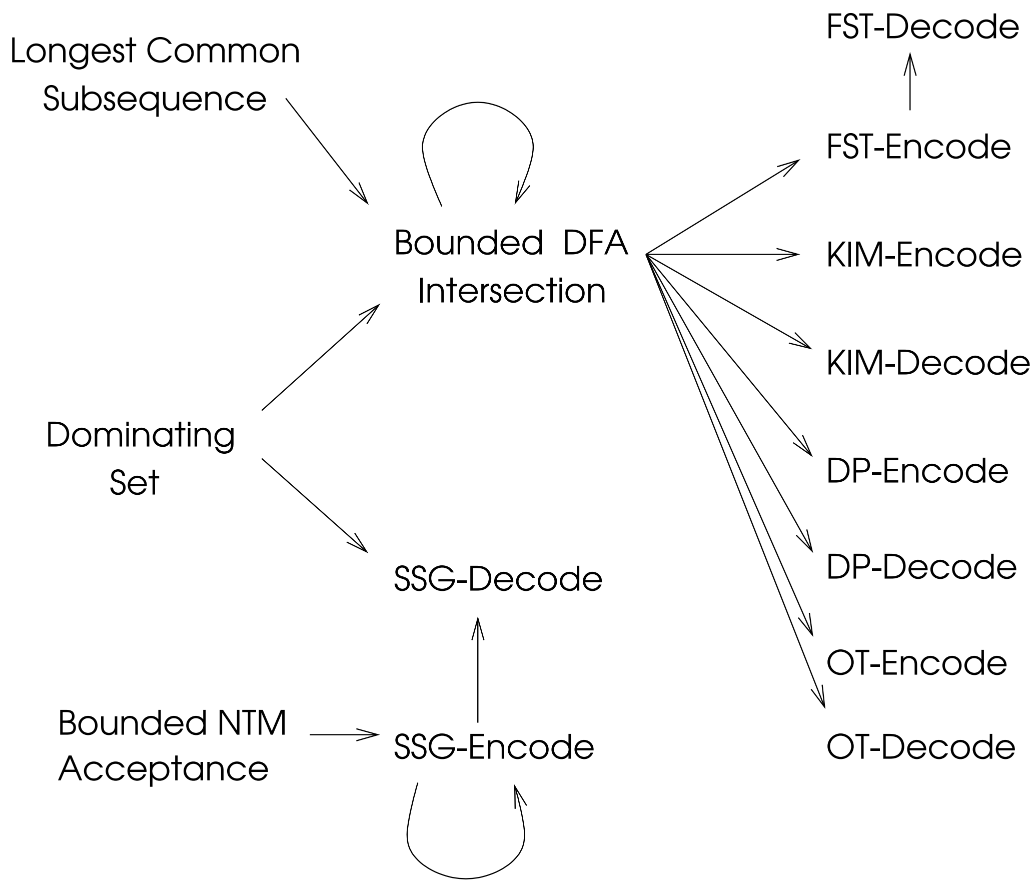


Figure 4.1: Reductions in Chapter 4. An arrow from problem Π to Π' means that there is at least one parametric reduction between parameterized versions of Π and Π'.

4.1 Simplified Segmental Grammars

4.1.1 Background

The earliest formal model of phonology was given by Chomsky and Halle in their classic book *The Sound Pattern of English (SPE)* [CH68]. The *SPE* model is based on rewriting rules operating on strings of segments — in particular, a sequence of sets of context-sensitive rewriting rules is applied repeatedly to transform given lexical segment-strings into surface segment-strings. Though the *SPE* model has been immensely influential, it has not had a great impact on practical natural language processing because both attempts at building systems based on (see [Col95, Spr92] and references) as well as theoretical investigations of [JoC72] the *SPE* model suggest that it encodes unnecessary and often troublesome computational power. This was confirmed by Ristad [Ris90, Ris93b], who showed that the encoding and decoding problems associated with the *SPE* model are undecidable. In an effort to reduce the computational power inherent in the *SPE* model while retaining its ability to describe natural languages, Ristad defined the Simplified Segmental Grammar (SSG) model [Ris93b].

An important part of this model is its conception of a rewriting rule and how such rules are applied to segment-strings. The notations used and definitions given below are adapted from those given to the appendix to Chapter 8 of [CH68]. Recall from Section 2.2.2 that a segment is defined relative to a set of features F. Let S_F denote the set of all possible segments relative a feature-set F, let the set of features that have values for a particular segment s, i.e., those features that are defined relative to s, be denoted by $F(s)$ and for each feature $f \in F(s)$, let $s(f)$ denote the value of feature f in segment s. If $F(s) = \phi$, s is the **null segment** and is written as ϕ. A traditional rewriting rule matches some substring of a given symbol-string and replaces it with another specified symbol-string. Segmental rewriting rules are slightly more complex, as they may match and replace both portions of segments and whole segments. Consider first how segments can match each other.

Definition 4.1.1 A segment s **matches a segment** t *if* $F(s) \subseteq F(t)$ *and for all* $f \in F(s)$, $s(f) = t(f)$.

Definition 4.1.2 a segment-string $x = x_1 x_2 \cdots x_k$ **matches a segment-string** $y = y_1 y_2 \cdots y_k$ *if* x_i *matches* y_i, $1 \leq i \leq k$.

Example 4.1.3 Consider the set of segments S defined relative to a set $F = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ of binary-valued features. In this and subsequent examples in this section, write each segment as the concatenation of its defined feature-value pairs, e.g., segment { **[a +]**,**[c -]** } will be written as **[a+c-]**. Segment **[a+]** matches segments **[a+b-]**, **[a+b+c-]**, and **[a+c+]**; however, it does not match the segments **[a-]**, **[b+c-]**, **[a-b-c-]**, and ϕ. Similarly, the segment-string **[a+][a-b+]** matches the segment-strings **[a+b-][a-b+]**, **[a+c+][a-b+c-]**, and **[a+b-c+][a-b+c+]** but does not match the segment-strings **[a-]**, **[a+][a+b+]**, **[b+][a+c+]**, and ϕ. ∎

Let the number of segments in a segment-string x be denoted by $|x|$. Consider next how a matched segment can be modified within a rule.

Definition 4.1.4 *Given segments c, r, s, and s', c* **modifies** *s* **via** *r* **to create** *s' if*

1. *c matches s; and*
2. *(a) $s' = \phi$ (if $r = \phi$) or*

   (b) $F(s') = F(s) \cup F(r)$, $f(s') = f(s)$ for all $f \in F(s) - F(r)$, and $f(s') = f(r)$ for all $f \in F(r)$ (if $r \neq \phi$).

This operation can add or delete segments, depending on whether s or r are equal to ϕ, respectively. Note, however, that this operation cannot delete features from segments but can only add new features or modify the values of existing ones.

Example 4.1.5 Segment [**a**+] modifies [**a**+**c**-] via [**a**+**b**-] to create [**a**+**b**-**c**-]. Similarly, [**a**+] modifies [**a**+**c**-] via [**b**+**c**+] to create [**a**+**b**+**c**+] and [**a**+] modifies [**a**+**c**-] via ϕ to create ϕ. However, it is not the case that [**a**+] modifies [**a**-**b**+] via [**c**+] to create [**a**-**b**+**c**-] (as [**a**+] does not match [**a**-**b**+]), [**a**+] modifies [**a**+**b**+] via [**c**+] to create [**b**+] (as { **b** } $\neq$ { **a**,**b** }) $\cup$ { **c** }), or [**a**+] modifies [**a**+**b**+] via [**c**+] to create [**a**+**b**+**c**-] (as [**c**+](**c**) $= + \neq - =$ [**a**+**b**+**c**-](**c**)). ∎

Definition 4.1.6 *A* **segmental rewriting rule** *r is an ordered pair $(x, y) \in S_F^* \times S_F^*$ such that x and y have the structures XAY and XBY, respectively, where $X, Y \in S_F^*$, $A, B \in S_F^{\leq 1}$, and it is not the case that $F(A) = F(B) = \phi$. The first component of r is called the* **structural description** *(or* **context***) and the second component is called the* **structural change**. *Rule r can be written either as $XAY \Rightarrow XBY$ or as $A \rightarrow B/X$ ____ Y.*

Some examples of segmental rewriting rules are given in Figure 4.2. Consider now how such a rule applies to a segment-string.

Definition 4.1.7 A segmental rewriting rule $XAY \Rightarrow XBY$ **applies to a segment-string** *Z if XAY matches Z, i.e., Z has the structure $X'A'Y'$ such that X matches X', A matches A', and Y matches Y'. The result of applying this rule to Z is a segment-string $Z' = X'A''Y'$, where A modifies A' via B to create A''. If the rule does not apply to the given segment-string, the result of the application is the given segment-string.*

Example 4.1.8 Rule (a) in Figure 4.2 applies to segment-string [**c**+][**a**+][**c**-], and the result of this application is the segment-string [**c**+][**a**+**b**-][**c**-]. Similarly, rules (b) and (c) apply to the segment-strings [**c**+][**c**-] and [**c**+][**a**+**b**-][**c**-], and the results of these applications are the segment-strings [**c**+][**a**+**b**+][**c**-] and [**c**+][**c**-], respectively. However, rules (b) and (c) do not apply to the segment-strings [**c**+][**b**-] and [**c**+][**a**-][**c**-], and the results of these applications are [**c**+][**b**-] and [**c**+][**a**-][**c**-], respectively. ∎

$$\begin{bmatrix} \mathbf{a} & + \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{a} & + \\ \mathbf{b} & - \end{bmatrix} \ / \ \begin{bmatrix} \mathbf{c} & + \end{bmatrix} \_\_\_ \begin{bmatrix} \mathbf{c} & - \end{bmatrix}$$

(a)

$$\phi \rightarrow \begin{bmatrix} \mathbf{a} & + \\ \mathbf{b} & + \end{bmatrix} \ / \ \begin{bmatrix} \mathbf{c} & + \end{bmatrix} \_\_\_ \begin{bmatrix} \mathbf{c} & - \end{bmatrix}$$

(b)

$$\begin{bmatrix} \mathbf{a} & + \\ \mathbf{b} & - \end{bmatrix} \rightarrow \phi \ / \ \begin{bmatrix} \mathbf{c} & + \end{bmatrix} \_\_\_ \begin{bmatrix} \mathbf{c} & - \end{bmatrix}$$

(c)

$$\begin{bmatrix} \mathbf{a} & + \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{a} & + \\ \mathbf{b} & - \end{bmatrix} \ / \ \begin{bmatrix} \mathbf{c} & + \end{bmatrix} \_\_\_ \begin{bmatrix} \mathbf{c} & - \end{bmatrix}$$

$$\begin{bmatrix} \mathbf{a} & + \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{a} & + \\ \mathbf{b} & + \end{bmatrix} \ / \ \begin{bmatrix} \mathbf{c} & + \end{bmatrix} \_\_\_ \begin{bmatrix} \mathbf{c} & + \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} \mathbf{a} & + \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{a} & + \\ \mathbf{b} & \alpha \end{bmatrix} \ / \ \begin{bmatrix} \mathbf{c} & + \end{bmatrix} \_\_\_ \begin{bmatrix} \mathbf{c} & \alpha \end{bmatrix}$$

(d)

Figure 4.2: Segmental Rewriting Rules. Each rule is defined relative to the set $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ of binary-valued features. (a) Feature modification. (b) Segment insertion. (c) Segment deletion. (d) Summary of two feature modification rules via by one rule which uses feature-value variable α.

Note that either A or B (but not both) may be the empty segment ϕ; in these cases, a rule would insert a segment into a segment-string (if $A = \phi$; see Figure 4.2(b)) or delete a segment from a segment-string (if $B = \phi$; see Figure 4.2(c)). Rules may also contain features whose values are variables; such feature-values are denoted by Greek letters. Each such rule summarizes a group of rules constructed by letting each variable take on all possible values of its associated feature and restricting the assignment of values within a rule such that each instance of a particular variable has the same value wherever it appears in that rule (see Figure 4.2(d)).

Example 4.1.9 Rule (d) in Figure 4.2 applies to both of the strings **[c+][a+][c-]** and **[c+][a+] [c+]**, and the results of these applications are the strings **[c+][a+b-][c-]** and **[c+][a+b+] [c+]**, respectively. ∎

Given a segment-string that is longer than the context of a rule, there are a variety of orders in which the rule can be applied to segment-substrings that are the length of that rule's context, e.g., iterative left to right, iterative right to left. In simplified segmental grammars, rules are applied simultaneously, i.e., a rule applies to all matching context-length segment-substrings in a given segment-string at the same time. A rule is said to successfully apply to a given segment-string if the context of the rule matches at least one substring of the given string.

Given the above, a **simplified segmental grammar** $g = \langle F, D, R, c_p, C, f_C \rangle$ consists of the following:

- A set of features F which defines a set of segments S_F.
- A lexicon $D \subseteq S_F^+$.
- A sequence $R = r_1, \ldots, r_{|R|}$ of segmental rewriting rules. This sequence R is divided into blocks $B_R = b_1 b_2 \cdots b_{|B_R|}$, $1 \leq |B_R| \leq |R|$, of contiguous subsequences of rules such that each block either contains a single rule or more than one rule. Each block of several rules is called a **mutually exclusive (m.e.) rule-set**.
- A **progression constant** c_p.
- A set of **control strings**, $C \subset \{0, 1, ?\}^{|R|}$ and a recursive function $f_C : D^+ \mapsto C$.

Given a lexical segment-string $u \in D^+$, the control-string $f_C(u)$ describes the status of each rule in the application of R to u. For $1 \leq i \leq |R|$, if the ith symbol in $f_C(u)$ is 1 then rule r_i is **obligatory**, if the ith symbol is 0 then rule r_i is **ignored**, and if the ith symbol is ? then rule r_i is **optional**. Given a lexical segment-string u, the blocks of rules in R are applied to u in their order in R. Let $res(i)$, $0 \leq i \leq |B_R|$, be the set of segment-strings generated by block i in R, and set $res(0) = \{u\}$. We can now define the result of applying R to u recursively as follows.

- If the current block b_i is a single rule, the result of applying this block depends on the status of the rule relative to $f_C(u)$:
  - If the rule is obligatory, $res(i)$ is the set of segment-strings generated by applying the rule to each segment-string in $res(i-1)$.
  - If the rule is ignored, $res(i) = res(i-1)$.
  - If the rule is optional, $res(i)$ is the union of $res(i-1)$ and the set of segment-strings generated by applying the rule to each segment-string in $res(i-1)$.
- If the current block b_i is a m.e. rule-set, the result of applying this block depends on the statuses of the rules in the block relative to $f_C(u)$:
  - If b_i consists of ignored rules, $res(i) = res(i-1)$.

– If b_i consists of some combination of ignored rules and optional rules, $res(i)$ is the union of $res(i-1)$ and the sets of segment-strings R_x, $x \in res(i-1)$, such that R_x consists of the segment-strings generated by applying each optional rule in b_i to x.

– If b_i consists of some combination of ignored rules, optional rules, and obligatory rules, $res(i)$ consists of the union of the sets of segment-strings R_x, $x \in res(i-1)$, such that R_x consists of either the segment-strings generated by applying to x all optional rules in b_i preceding the first obligatory rule that can be successfully applied to x, as well as the segment-string generated by applying that obligatory rule to x (if such an obligatory rule exists in b_i) or the segment-strings generated by applying each optional rule in b_i to x (if no obligatory rule in b_i can be successfully applied to x).

Example 4.1.10 Given a rule-set R in which the first rule is rule (b) from Figure 4.2 and the second rule is rule (a) from Figure 4.2, the assignment of different control strings to the segment-string **[c+][c-]** produces the following results:

Control String	Results
00	{ **[c+][c-]** }
01	{ **[c+][c-]** }
0?	{ **[c+][c-]** }
10	{ **[c+][a+b+][c-]** }
11	{ **[c+][a+b-][c-]** }
1?	{ **[c+][a+b+][c-]**, **[c+][a+b-][c-]** }
?0	{ **[c+][c-]**, **[c+][a+b+][c-]** }
?1	{ **[c+][c-]**, **[c+][a+b-][c-]** }
??	{ **[c+][c-]**, **[c+][a+b+][c-]**,**[c+][a+b-][c-]** }

If these two rules are in turn considered not as separate rules but as a m.e. rule-set, the results are as follows:

Control String	Results
00	{ **[c+][c-]** }
01	{ **[c+][c-]** }
0?	{ **[c+][c-]** }
10	{ **[c+][a+b+][c-]** }
11	{ **[c+][a+b+][c-]** }
1?	{ **[c+][a+b+][c-]** }
?0	{ **[c+][c-]**, **[c+][a+b+][c-]** }
?1	{ **[c+][c-]**, **[c+][a+b+][c-]** }
??	{ **[c+][c-]**, **[c+][a+b+][c-]** }

∎

The rule-sequence R will be constrained such that no rule in R can be applied to a segment-string s to create a segment-string s' that is more than c_p segments longer or shorter than s, i.e., $c_p - |s| \leq |s'| \leq c_p + |s|$. Let $g(u) = res(|R|)$ be the set of surface segment-strings resulting from the application of the rules of simplified segmental grammar g to lexical segment-string $u \in D^+$.

The SSG model was defined and analyzed by Ristad in [Ris93b]. His analysis focused on the following two problems:

SSG-ENCODE
Instance: A simplified segmental grammar $g = \langle F, D, R, c_p, C, f_C\}$ and segment-strings $u, s \in S_F^+$.
Question: Is it the case that $s \in g(u)$?

SSG-DECODE
Instance: A simplified segmental grammar $g = \langle F, D, R, c_p, C, f_C\}$ and a segment-string $s \in S_F^+$.
Question: Is there a string $u \in D^+$ such that $s \in g(u)$?

These problems are NP-complete by the proofs in [Ris93b, Theorem 3]. The versions of these problems that are analyzed in this thesis differ from those defined by Ristad in two ways: (1) The lexicon D is included in the problem instances, and (2) problem instances are restricted to simplified segmental grammars in which rules can only change feature values — that is, segments cannot be inserted or deleted, i.e., $c_p = 0$. The lexicon D is included in the problem instances so that it can be accessed in algorithms for SSG-DECODE that are described in the next section. The lexicon D will be specified as a DFA $DFA(D) = \langle Q_D, S_F, \delta_D, s_D, F_D\rangle$ on $|Q_D|$ states whose transitions are labeled with segments and which recognizes the language D^+. The restriction to rules that can only change feature values makes results derived here for simplified segmental grammars comparable to results derived for the other theories examined in this thesis, which are similarly restricted to having surface and lexical forms of the same length.

For various technical and theoretical reasons, the *SPE* model has ceased to be a formalism of choice in current linguistics research. Hence, it can be argued that further analysis of either the *SPE* or SSG models is not worthwhile. However, I believe that the systematic parameterized analysis given in the next section is of interest for two reasons. Firstly, the SSG model is one of the few linguistic theories that has been derived within the framework proposed by Ristad in [Ris93a], in which classical complexity-theoretic analyses are used to diagnose and subsequently restrict the sources of computational power in linguistic theories (see Chapter 3). In light of criticisms of such uses of classical complexity-theoretic results in Section 2.1.2 and Chapter 3, the systematic parameterized analysis in the next section illustrates how further complexity-theoretic analysis of linguistic theories like the SSG model can still proceed along the lines suggested by Ristad. Secondly, even if the *SPE* and SSG models are not themselves currently in favor, they are still the prototypical rule-based phonological theories, and (as will be discussed in Section 4.1.3) analyses of the SSG model are still relevant to other more popular rule-based theories, such as those which encode phonological rules as finite-state transducers [KK94, Lap97, PR97].

4.1.2 Analysis

The systematic parameterized analysis in this section will focus on the following aspects:

- The number of rules in R ($|R|$).
- The number of m.e. rule-sets in R, where individual rules not in m.e. rule-sets are counted as m.e. rule-sets of size 1 ($|R_{m.e.}|$).
- The maximum number of rules in any m.e. rule-set in R ($\#(R_{m.e.})$).
- The maximum number of optional rules in any control-string ($|R_?|$).
- The maximum length of the context of any rule in R (c).
- The length in segments of the given lexical / surface form ($|u|$ / $|s|$).
- The number of features ($|f|$).
- The maximum number of values for any feature ($|v|$).

Aspects $|R|$, $|R_{m.e.}|$, $\#(R_{m.e.})$, and $|R_?|$ are measures of the complexity of the rule-application control machinery embedded within R. Though $|R|$ may at first sight be the most natural measure of rule-set complexity, $|R_{m.e.}|$ and $\#(R_{m.e.})$ are in fact more accurate measures of the number and complexity of phonological phenomena encoded in R. This is so because each m.e. rule-set in the SSG model corresponds to a rule-set in the *SPE* model, which in turn corresponds to an particular phonological phenomenon [Ris93b, pp. 10–11].[1]

Many of the hardness results given in this section will be derived via reductions from the following problem:

[1] One might also define aspects that characterize R in a broader sense by restricting rule applications or interactions within individual derivations. For instance, define the **length of a derivation** of s from u via g as the number of rules in R that are successfully applied in transforming u into s [Ris90, pp. 20–23 and 32–33]. As the number of m.e. rule-sets is an upper bound on the length of a derivation, the W-hardness results relative to $|R_{m.e.}|$ in Theorems 4.1.16 and 4.1.18 in conjunction with Lemma 2.1.34 show that the versions of SSG-ENCODE and SSG-DECODE that are parameterized relative to derivation length are not in FPT unless at least part of the W-hierarchy collapses to FPT. Several other rule-based aspects that may be of interest in future investigations are given in [Ris90, pp. 26–30].

Bounded nondeterministic Turing machine acceptance (BNTMA)
Instance: A nondeterministic Turing machine $N = \langle Q, \Sigma, \delta, s, F \rangle$, a string $u \in \Sigma^*$, and a positive integer t.
Question: Does N accept u via a computation that requires at most t steps?

A Turing machine is essentially a finite-state automaton augmented by a read/write head which can move both forward and backward on a one-way infinite tape of symbols. At the beginning of a computation, the input is at the beginning of the tape (the rest of the tape squares are filled with special blank (#) symbols), the read/write head is on the first tape square, and the state is s. Using the transition relation $\delta : Q \times \Sigma \mapsto Q \times \Sigma \times \{L, R\}$, a Turing machine computes next states and actions (each invocation of δ being considered a step) much like a finite-state automaton, accepting (rejecting) its input if the computation ends (does not end) in a final state. Readers wishing a more detailed description of Turing machines should consult standard textbooks on computation theory such as [HU79, LP81]). The unbounded version of problem BNTMA, i.e., $t = \infty$, is the Halting Problem, which is known to be undecidable [HU79, LP81]. Problem BNTMA is NP-hard because every problem in NP trivially reduces to it (given a problem $\Pi \in NP$ whose associated nondeterministic polynomial time algorithm requires $p(|x|)$ time for an input x when implemented on a nondeterministic Turing machine, create the appropriate instance of BNTMA in which $t = p(|x|)$).

In the following, assume without loss of generality that the set of final states F in the nondeterministic Turing machine in each instance of BNTMA consists of a single state h.

Lemma 4.1.11 *BNTMA* $\leq_m$ SSG-Encode *(adapted from [Ris93b, Theorem 3]).*

Proof: This is essentially the reduction given in [Ris93b, Theorem 3], modified such that the constructed instance of SSG-Encode uses rules that can only change feature values. This reduction simulates Turing machine computations by using segment-strings to encode Turing machine configurations and rules to simulate the nondeterministic transitions between configurations. By analogy with the notion of configuration for finite-state automata, the configuration of a Turing machine will have three components — namely, the current non-blank tape contents, the current position of the read/write head on the tape, and the current state. Note that a t-step NTM computation can use at most t tape squares; hence, as current head position and state can be encoded onto a tape by additional markings in the appropriate tape squares, each configuration of the Turing machine can be encoded as a t-segment string. Each segment in such a string has three features, f_1, f_2, and f_3, with feature-value sets $Q \cup \{ \_ \}$, $\Sigma \cup \{\#\}$, and $\{L, R, S\}$, respectively. Feature f_1 indicates the current state and the position of the read/write head on the tape. If $s(f_1) = \_$ for some segment s, the read/write head is not on the square corresponding to that segment; else, the read/write head is on the tape square corresponding to that segment and the current state is $s(f_1)$. Feature f_2 indicates the contents of the tape square corresponding to the segment, and feature f_3 indicates the read/write head's current direction of motion. For instance, a configuration in which the tape has four squares that contains the string 110, the read-write head is on the second tape square, and the current state is q_5 would correspond to the segment-string

$$\begin{bmatrix} f_1 & \_ \\ f_2 & 1 \\ f_3 & S \end{bmatrix} \begin{bmatrix} f_1 & q_5 \\ f_2 & 1 \\ f_3 & S \end{bmatrix} \begin{bmatrix} f_1 & \_ \\ f_2 & 0 \\ f_3 & S \end{bmatrix} \begin{bmatrix} f_1 & \_ \\ f_2 & \# \\ f_3 & S \end{bmatrix}$$

Let $[x_1, x_2, x_3]$ represent a segment s in which $s(f_1) = x_1$, $s(f_2) = x_2$, and $s(f_3) = x_3$. If a segment is undefined for a feature, leave that feature's portion of the encoding blank, e.g., the segment s such that f_1 and f_3 are undefined and $s(f_2) = x_2$ is written as $[, x_2,]$. Under this scheme, the segment-string described above would be written as

$$[\,\_\,, 1, S][q_5, 1, S][\,\_\,, 0, S][\,\_\,, \#, S]$$

Note that feature f_3 will only have a feature-value other than S when a tape head is being moved to the left or right (see below).

Given an instance $\langle N = \langle Q, \Sigma, \delta, s, F = \{h\}\rangle, u, t\rangle$ of BNTMA, construct the following instance $\langle g' = \langle F', D', R', C', f'_C\rangle, u', s'\rangle$ of SSG-ENCODE: Let $D' = \{u'\}$ where u' is the t-segment string corresponding to a configuration in which u is the string in the first $|u|$ squares of the tape, the read/write head is on the first tape square, and the current state is s, s' be the t-segment string where each segment has the form $[h, \#, S]$ and $f' = \{f_1, f_2, f_3\}$ as described above. The rule-sequence R has the following structure:

1. Simulate the transitions of N computing on u.

   This is implemented by t blocks of rules, where each block has the following form:

   (a) One m.e. rule-set of $|\delta|$ rules such that each transition-relation entry of the form $\delta(q, x) = \{(q_1, x_1, d_1), (q_2, x_2, d_2), \ldots, (q_k, x_k, d_k)\}$ is replaced by $k - 1$ optional rules of the form $[q, x, S] \rightarrow [q_i, x_i, d_i]$ for $1 \leq i \leq (k - 1)$ and a final obligatory rule of the form $[q, x, S] \rightarrow [q_k, x_k, d_k]$.

   (b) Two obligatory rules of the forms $[\,\_\,, ,] \rightarrow [\alpha, ,]$ / ___ $[\alpha, , L]$ and $[, , L] \rightarrow [\,\_\,, , S]$, respectively.

   (c) Two obligatory rules of the forms $[\,\_\,, ,] \rightarrow [\alpha, ,]$ / $[\alpha, , R]$ ___ and $[, , R] \rightarrow [\,\_\,, , S]$, respectively.

   Note that the (a)-rules perform the appropriate transitions from δ and the (b)- and (c)-rules respectively move the read/write head left or right as necessary. Each of these t blocks of rules thus corresponds to a step in the computation of N on u.

2. Clean up the tape at end of computation.

   This is implemented by the following three blocks of rules:

(a) $t-1$ obligatory rules of the form $[\_\,,,] \rightarrow [h,,]$ / ___ $[h,,]$.

(b) $t-1$ obligatory rules of the form $[\_\,,,] \rightarrow [h,,]$ / $[h,,]$ ___ .

(c) An obligatory rule of the form $[,\alpha,\beta] \rightarrow [,\#,S]$.

The (a)- and (b)-rules propagate a final state to all segments, and the (c) rule effectively erases the tape contents.

Note that the only rules above that differ from those in Ristad's original reduction are those in (2). To complete the construction, let C' contain the single control-string implicit in the rule-sequence described above and f'_C associate this control string with every member of D'^+. This construction can be done in time polynomial in the size of the given instance of BNTMA.

To see that the construction above is a many-one reduction, note that any solution to the constructed instance of SSG-ENCODE is a sequence of applications of rules in R' that transforms u' into s' such that the rules applied in block (1) described above correspond to a sequence of transitions in an accepting computation of N on u. Moreover, any solution to the given instance of BNTMA is a sequence of transitions in an accepting computation of N on u, which can be used to construct a sequence of rule applications (modulo the addition of the appropriate rule-applications from block (2) described above) that can transform u' into s'. Hence, the given instance of BNTMA has a solution if and only if the constructed instance of SSG-ENCODE has a solution.

Note that in the constructed instance of SSG-ENCODE, $|R'| = (|\delta|+4)t + 2(t-1)+1 \leq (|Q||\Sigma|+4)t + 2(t-1)+1 = (|Q||\Sigma|+6)t-1$, $|R'_?| \leq |\delta|-1 \leq |Q||\Sigma|-1$, $|R'_{m.e.}| = (1+4)t+2(t-1)+1 = 7t-1$, $\#(R'_{m.e.}) \leq |\delta| \leq |Q||\Sigma|$, $c' = 2$, $|u'| = |s'| = t$, $|f'| = 3$, and $|v'| = \max(|Q|+1, |\Sigma|+1, 3)$. ∎

Lemma 4.1.12 SSG-ENCODE $\leq_m$ SSG-DECODE.

Proof: Given an instance $\langle g = \langle F, D, R, C, f_C\rangle, u, s\rangle$ of SSG-ENCODE, construct the following instance $\langle g' = \langle F', D', R', C', f'_C\rangle, s'\rangle$ of SSG-DECODE: Let $F' = F$, $R' = R$, $C' = C$, $f'_C = f_C$, $s' = s$, and $D' = \{u\}$. This construction can be done in time polynomial in the size of the given instance of SSG-ENCODE. Note that, as rules cannot add or delete segments, the only possible member of D'^+ that can possibly be transformed into s' is u. Hence, the given instance of SSG-ENCODE has a solution if and only if the constructed instance of SSG-DECODE has a solution.

Note that all aspects in the constructed instance of SSG-DECODE have the same values as those in the given instance of SSG-ENCODE. ∎

Lemma 4.1.13 SSG-ENCODE $\leq_m$ SSG-ENCODE *such that* $|v| = 2$.

Proof: Given an instance $\langle g = \langle F, D, R, C, f_C \rangle, u, s \rangle$ of SSG-ENCODE, construct the following instance $\langle g' = \langle F', D', R', C', f'_C \rangle, u', s' \rangle$ of SSG-ENCODE: Let $C' = C$ and F' be the set of binary-valued features created by transforming each feature $f \in F$ with an associated set of feature-values $\{v_1, v_2, \ldots, v_k\}$ into a set of binary-valued features $(f = v_1), (f = v_2), \ldots, (f = v_k)$. Create D', R', f'_C, u', and s' from D, R, f_C, u, and s by replacing every occurrence of feature-value pair $[f \quad v_j]$ with the equivalent set of feature-value pairs $[(f = v_1) \quad -], [(f = v_2) \quad -], \ldots, [(f = v_{j-1}) \quad -], [(f = v_j) \quad +]$, $[(f = v_{j+1}) \quad -], \ldots, [(f = v_k) \quad -]$ and replacing every occurrence of feature-value pair $[f \quad \alpha]$ incorporating a variable α with the equivalent set of feature-value pairs $[(f = v_1) \quad \alpha_1]$, $[(f = v_2) \quad \alpha_2], \ldots, [(f = v_k) \quad \alpha_k]$. This construction can be done in time polynomial in the given instance of SSG-ENCODE. It is obvious from this construction that the given instance of SSG-ENCODE has a solution if and only if the constructed instance of SSG-ENCODE has a solution.

Note that in the constructed instance of SSG-ENCODE, $|f'| \leq |f||v|$, $|v'| = 2$, and all other aspects have the same values as in the given instance of SSG-ENCODE. ∎

Lemma 4.1.14 SSG-DECODE $\leq_m$ SSG-DECODE *such that* $|v| = 2$.

Proof: Similar proof to that given for Lemma 4.1.13. ∎

Lemma 4.1.15 SSG-ENCODE $\leq_m$ SSG-ENCODE *such that* $\#(R_{m.e.}) = 1$.

Proof: Given an instance $\langle g = \langle F, D, R, C, f_C \rangle, u, s \rangle$ of SSG-ENCODE, construct the following instance $\langle g' = \langle F', D', R', C', f'_C \rangle, u', s' \rangle$ of SSG-ENCODE: Let $F' = F \cup \{applied\}$ where *applied* is a binary-valued feature, create D', u', and s' from D, u, and s by adding the feature-value pair $[applied \quad -]$ to every segment, and create R', C', and f'_C from R, C, and f_C by replacing each m.e. rule-set of size k in R by $k(2|u| - 1) + 1 \leq 2k|u|$ obligatory rules in R' which can be split into the following two rule-sets:

1. k blocks of $2(|u| - 1) + 1 = 2|u| - 1$ rules apiece, where block i consists of a modified version of the i-th rule from the original m.e. rule-set which only changes a segment if it has the feature-value pair $[applied \quad -]$ and which adds the feature-value pair $[applied \quad +]$ to every segment it changes, followed by $|u| - 1$ obligatory rules that propagate feature-value pair $[applied \quad +]$ to the right and $|u| - 1$ obligatory rules that propagate feature-value pair $[applied \quad +]$ to the left (these last two sets of rules are analogous to rule-sets 2(a) and 2(b) in Lemma 4.1.11).
2. A final obligatory rule that sets all feature-value pairs $[applied \quad +]$ to $[applied \quad -]$.

Note that each block of the first rule-set attempts to apply a rule from the m.e. rule-set, and if that rule applies, propagates feature-value pair $[applied \quad +]$ to all segments in the segment string such that no subsequent rule from the m.e. rule-set can apply (because such an application would require a segment to have the feature-value pair [*applied* $-$]). The lone rule in the second rule-set ensures that subsequent m.e. rule-sets in R' can function properly. Given this equivalence of the operation of R and R', it is obvious that the given instance of SSG-ENCODE has a solution if and only if the constructed instance of SSG-ENCODE has a solution. To complete this proof, note that the construction above can be done in time polynomial in the given instance of SSG-ENCODE.

Note that in the constructed instance of SSG-ENCODE, $|R'| \leq 2|R||u|$, $|R'_{m.e.}| = |R'|$, $\#(R'_{m.e.}) = 1$, $c' = \max(c, 2)$, $|f'| = |f| + 1$, $|v'| = \max(|v|, 2)$, and all other aspects have the same values as in the given instance of SSG-ENCODE. ∎

Theorem 4.1.16

1. SSG-ENCODE *is NP-hard when* $\#(R_{m.e.}) = 1$, $c = 2$, *and* $|f| = 4$.
2. SSG-ENCODE *is NP-hard when* $\#(R_{m.e.}) = 1$, $c = 2$, *and* $|v| = 2$.
3. $\langle |R_?| \rangle$-SSG-ENCODE *is in FPT.*
4. $\langle |R| \rangle$-SSG-ENCODE *is in FPT.*
5. $\langle |R_{m.e.}|, \#(R_{m.e.}) \rangle$-SSG-ENCODE *is in FPT.*
6. $\langle |u|, |f|, |v| \rangle$-SSG-ENCODE *is in FPT.*
7. $\langle |R_{m.e.}|, c_2, |u|, |f|_3 \rangle$-SSG-ENCODE *is* $W[1]$*-hard.*
8. $\langle |R_{m.e.}|, c_2, |u|, |v|_2 \rangle$-SSG-ENCODE *is* $W[1]$*-hard.*
9. $\langle \#(R_{m.e.})_1, c_2, |u|, |f|_4 \rangle$-SSG-ENCODE *is* $W[1]$*-hard.*
10. $\langle \#(R_{m.e.})_1, c_2, |u|, |v|_2 \rangle$-SSG-ENCODE *is* $W[1]$*-hard.*
11. $\langle \#(R_{m.e.})_1, c_2, |f|_4 \rangle$-SSG-ENCODE $\notin XP$ *unless* $P = NP$.
12. $\langle \#(R_{m.e.})_1, c_2, |v|_2 \rangle$-SSG-ENCODE $\notin XP$ *unless* $P = NP$.

Proof:

Proof of (1): Follows from the NP-hardness of BNTMA, the reduction in Lemma 4.1.11 from BNTMA to SSG-ENCODE in which $c = 2$ and $|f| = 3$, and the reduction in Lemma 4.1.15 from SSG-ENCODE to SSG-ENCODE in which $\#(R_{m.e.}) = 1$.

Proof of (2): Follows from (1) and the reduction in Lemma 4.1.13 from SSG-ENCODE to SSG-ENCODE in which $|v| = 2$.

Proof of (3): Note that given an unambiguous, i.e. no ?-positions, control string, we can simulate the application of R to u in $O(|R||u||f|c)$ time. As each optional rule either applies or doesn't apply, there are $2^{|R_?|}$ unambiguous resolutions of $|R_?|$?-positions. Consider the algorithm that simulates the application of R to u under each of the possible $2^{|R_?|}$ unambiguous resolutions of the ?-positions in R and checks if s is produced. This algorithm runs in $O(2^{|R_?|}|R||u||f|c)$ time, which is fixed-parameter tractable relative to $|R_?|$.

Proof of (4): As it is always the case that $|R_?| \leq |R|$, the result follows from (3) and Lemma 2.1.33.

Proof of (5): As it is always the case that $|R| \leq |R_{m.e.}|\#(R_{m.e.})$, the result follows from (4) and Lemma 2.1.33.

Proof of (6): This result is based on the following observation: there are $|v|^{|f|}$ possible segments in an SSG grammar, and hence $|v|^{|f||u|}$ possible segment-strings of length $|u|$. Pick arbitrary total orders on f and v and use these orders to establish a lexicographic ordering on the set of all possible segment-strings of length $|u|$ such that each segment-string of length $|u|$ has a unique integer index in the range 1 to $|v|^{|f||u|}$. For instance, the binary form of such an index could be $|u||f||v|$ bits divided into $|u|$ zones of $|f||v|$ bits apiece corresponding to individual segments, each of which is in turn subdivided into $|f|$ zones of $|v|$ bits corresponding to values of individual features within a segment. Let $ind(s)$ be the index so generated for segment-string s, and $str(i)$ be the segment-string corresponding to index i. Under the scheme described above, both of these functions can be computed in $O(|u||f||v|)$ time.

Given the above, consider the following algorithm:

1. Initialize an array *pos* of $|v|^{|f||u|}$ bits to 0, and then set to 1 the bit of *pos* whose index corresponds to u.

2. For each m.e. rule-set in R in their order of occurrence in R do:

   (a) If the m.e. rule-set does not consist entirely of ignored rules then

      i. Initialize an array *buff* of $|v|^{|f||u|}$ bits to 0.

      ii. Initialize an array *mark* of $|v|^{|f||u|}$ bits to 0.

      iii. For each rule r in the m.e. rule-set in their order of occurrence in the rule-set do:

         A. For each segment-string u such that $pos(ind(u)) = 1$ and $mark(ind(u)) = 0$, compute the segment-string u' resulting from the application of rule r to u. If $u \neq u'$, i.e., rule r applied successfully, set $buff(ind(u')) = 1$, and set $mark(ind(u)) = 1$ (if r is obligatory) and $buff(ind(u)) = 1$ (if r is optional).

      iv. Copy *buff* into *pos*.

3. If $pos(ind(s)) = 1$, the answer is "yes"; else, the answer is "no".

Note that in the algorithm above, a lone rule not in an m.e. rule set counts as an m.e. rule-set of size 1. This algorithm essentially keeps track of all possible segment-strings that can be generated by any legal sequence of rule-applications possible under R. When processing a particular m.e. rule-set, the array $mark$ keeps track of segment-strings that have not yet been transformed by a successful application of an obligatory rule and can hence still be transformed by optional rules and obligatory rules. If s is in the set of possible segment-strings when all rules in R have been processed, the answer is "yes"; else, the answer is "no". Steps (1), (2), and (3) can be done in time $O(|v|^{|f||u|})$, $O(|v|^{|f||u|}|R|(|u|c+|u||f||v|))$, and $O(|u||f||v|)$, respectively. Hence, the algorithm as a whole runs in $O(|v|^{|f||u|}|R|(|u|c+|u||f||v|))$ time, which is fixed-parameter tractable relative to $|u|$, $|f|$, and $|v|$.

Proof of (7): Follows from the $W[1]$-completeness of $\langle t \rangle$-BNTMA [DFK+94], the reduction in Theorem 4.1.11 from BNTMA to SSG-ENCODE in which $|R_{m.e.}| = 7t - 1$, $c = 2$, $|u| = t$, and $|f| = 3$, and Lemma 2.1.25.

Proof of (8): Follows from (7), the reduction in Lemma 4.1.13 from SSG-ENCODE to SSG-ENCODE in which $|v| = 2$, and Lemma 2.1.25.

Proofs of (9): Follows from (7), the reduction in Lemma 4.1.15 from SSG-ENCODE to SSG-ENCODE in which $\#(R_{m.e.}) = 1$, and Lemma 2.1.25.

Proof of (10): Follows from (8), the reduction in Lemma 4.1.13 from SSG-ENCODE to SSG-ENCODE in which $|v| = 2$, and Lemma 2.1.25.

Proofs of (11) and (12): Follow from (1) and (2) and Lemma 2.1.35. ∎

The most important result above is the fixed-parameter tractability of SSG-ENCODE relative to $|R_?|$ because $|R_?|$ is always less than (and hence by Lemma 2.1.33 grants fixed-parameter tractability relative to) many other aspects. Unfortunately, as the next reduction shows, this stroke of luck does not extend to problem SSG-DECODE. This reduction is from the following problem:

DOMINATING SET [GJ79, Problem GT2]
Instance: A graph $G = (V, E)$ and a positive integer k.
Question: Does G have a dominating set of size at most k, i.e., a set of vertices $V' \subseteq V$, $|V'| \leq k$, such that each vertex in V is either in V' or adjacent to a vertex in V'?

Lemma 4.1.17 DOMINATING SET $\leq_m$ SSG-DECODE *such that all rules are obligatory.*

Proof: This reduction will use an appropriately structured lexicon to generate the set of all possible solutions and use the rules to check these solutions, cf., the reduction in Lemma 4.1.11 which uses rules to generate and check solutions. Given an instance $\langle G = (V, E), k \rangle$ of DOMINATING SET, construct the following instance $\langle g' = \langle F', D', R', C', f'_C \rangle, s' \rangle$ of SSG-DECODE: Let $F = \{SegNum, Value, Format, Check\}$ with the associated feature-value sets $\{1, \ldots, k\}$, $\{1, \ldots, |V|\}$, $\{-, +\}$, and $\{0, \ldots, |V|\}$, respectively. Following the convention used in Lemma 4.1.11, let $[x_1, x_2, x_3, x_4]$ represent a segment s in which $s(SegNum) = x_1$, $s(Value) = x_2$, $s(Format) = x_3$, and $s(Check) = x_4$, and if a segment is undefined for a feature then that feature's portion of the encoding is left blank. Let s' be the segment-string $[1, |V|, +, |V|][2, |V|, +, |V|] \cdots [k, |V|, +, |V|]$ and $D = \{[i, j, -, 0] \mid 1 \leq i \leq k, 1 \leq j \leq |V|\}$. The rule sequence R has the following structure:

1. Verify that any selected lexical form u' encodes a candidate solution to the given instance of Dominating set.

   The requisite structure of such a string u' is $[1, v_1, -,][2, v_2, -, 0] \cdots [k, v_k, -, 0]$ where $v_i \in \{1, \ldots, |V|\}$, $1 \leq i \leq k$. This structure is ensured by the following block of k obligatory rules, such that the first $k-1$ of these rules have the form $[i, , -, 0] \rightarrow [i, , +, 0] \; / \; \_\_ \; [(i+1), , ,]$, $1 \leq i \leq k-1$, and the kth rule has the form $[k, , -, 0] \rightarrow [k, , +, 0] \; / \; [(k-1), , ,] \; \_\_$. These rules ensure that the lexical form u' selected from D'^{+} consists of one or more sequences of k segments whose values for feature $Segnum$ run from 1 to k; as no rules can add or delete segments, any valid u' must be the same length as s', and hence must have exactly k segments.

2. Check whether the candidate solution encoded in lexical form u' is a dominating set of size k in G.

   This is implemented by $|V|$ blocks of $O(k(|V|+2))$ obligatory rules apiece, where block i consists of

   (a) $O(k|V|)$ rules of the form $[j, l, +, (i-1)] \rightarrow [j, l, +, i]$, where $1 \leq j \leq k$ and $l \in V_i^{adj}$, where V_i^{adj} is the set of vertices adjacent to and including vertex i in G;

   (b) k rules of the form $[, , , (i-1)] \rightarrow [, , , i] \; / \; \_\_ \; [, , , i]$; and

   (c) k rules of the form $[, , , (i-1)] \rightarrow [, , , i] \; / \; [, , , i] \; \_\_$.

   Note that, as each edge contributes two vertex-adjacencies in the graph G, the total number of rules of type (a) over all blocks is $2|E|$. Each block i above checks that at least one segment in u' has a value for feature $Value$ that is in V_i^{adj}; if so, it advances that segment's value for feature $Check$ by 1 and makes sure that this new value for $Check$ is propagated to all other segments in u'.

3. Clean up the string s' created by applying R' to u'.

   This is done by a single obligatory rule of the form $[, \alpha, ,] \rightarrow [, |V|, ,]$, which effectively erases the value of feature $Value$ in every segment.

Finally, let C' contain the single control-string consisting of $|R|$ 1's, i.e., all rules are obligatory, and f'_C associate this control string with every member of D'^{+}. This construction can be done in time polynomial in the size of the given instance of Dominating set.

To see that the construction above is a many-one reduction, note the following two facts:

1. The feature-value pair $[Format\ \ +]$ can occur in all segments of s' only if all k rules in (1) above apply to some lexical form u', i.e., u' has the necessary k-segment structure.

2. The feature-value pair $[Check\ \ |V|]$ can occur in all segments of s' only if at least one rule from 2(a) applies for each block i in (2) above, i.e., u' encodes a dominating set of size at most k. Note that none of the rules above guarantee that the vertices encoded in u' are distinct. However, this does not matter as any dominating set X of size $k' < k$ can be trivially turned into a dominating set of size k by adding $k - k'$ distinct vertices from V that are not already in X to X.

This means that any solution to the constructed instance of SSG-DECODE is a lexical form $u' \in D'^{+}$ that can be transformed by R' into s', which by the logic above corresponds to a set of at most k vertices that form a dominating set for G. Moreover, any solution for the given instance of DOMINATING SET is a k-vertex dominating set for G, which corresponds to a set of k-segment lexical forms from D'^{+} that can be transformed by R' into s'. Hence, the given instance of DOMINATING SET has a solution if and only if the constructed instance of SSG-DECODE has a solution.

Note that in the constructed instance of SSG-DECODE, $|R'| = |R'_{m.e.}| = k + (2|E| + 2k|V|) + 1 = 2|E| + k(2|V| + 1) + 1$, $|R'_?| = 0$, $\#(R'_{m.e.}) = 1$, $c' = 2$, $|s'| = k$, $|f'| = 4$, and $|v'| = |V| + 1$. ∎

Theorem 4.1.18

1. SSG-DECODE *is NP-hard when* $|R_?| = 0$, $\#(R_{m.e.}) = 1$, $c = 2$, *and* $|f| = 4$.
2. SSG-DECODE *is NP-hard when* $|R_?| = 0$, $\#(R_{m.e.}) = 1$, $c = 2$, *and* $|v| = 2$.
3. $\langle |R|, |s| \rangle$-SSG-DECODE *is in FPT.*
4. $\langle |R_{m.e.}|, \#(R_{m.e.}), |s| \rangle$-SSG-DECODE *is in FPT.*
5. $\langle |s|, |f|, |v| \rangle$-SSG-DECODE *is in FPT.*
6. $\langle |R_{m.e.}|, c_2, |s|, |f|_3 \rangle$-SSG-DECODE *is W[1]-hard.*
7. $\langle |R_{m.e.}|, c_2, |s|, |v|_2 \rangle$-SSG-DECODE *is W[1]-hard.*
8. $\langle |R_?|_0, \#(R_{m.e.})_1, c_2, |s|, |f|_4 \rangle$-SSG-DECODE *is W[2]-hard.*
9. $\langle |R_?|_0, \#(R_{m.e.})_1, c_2, |s|, |v|_2 \rangle$-SSG-DECODE *is W[2]-hard.*
10. $\langle |R_?|_0, \#(R_{m.e.})_1, c_2, |f|_4 \rangle$-SSG-DECODE $\notin XP$ *unless* $P = NP$.
11. $\langle |R_?|_0, \#(R_{m.e.})_1, c_2, |v|_2 \rangle$-SSG-DECODE $\notin XP$ *unless* $P = NP$.

Proof:

Proof of (1): Follows from the NP-hardness of DOMINATING SET [GJ79, Problem GT2] and the reduction in Lemma 4.1.17 from DOMINATING SET to SSG-DECODE in which $|R_?| = 0$, $\#(R_{m.e.}) = 1$, $c = 2$, and $|f| = 4$.

Proof of (2): Follows from (1) and the reduction in Lemma 4.1.14 from SSG-DECODE to SSG-DECODE in which $|v| = 2$.

Proof of (3): This result follows from the algorithm that runs a rule-set R in reverse on the given surface form s. Consider first how various types of individual rules may be run in reverse. Given an obligatory rule r of the form $A \rightarrow B$ / X ___ Y and a segment-string s, one can apply r in reverse to s by picking some subset of the occurrences of B in s, creating the string u from s by replacing all selected occurrences of B with A, and then verifying that s can be created from u by r. This verification step is necessary because certain occurrences of B in s may have been present in the underlying forms used to create s and may in turn have interacted with the context of r to create the occurrences of B in s. As there are at most $|s|$ occurrences of B in s and any possible subset of these occurrences could have been produced by r, there will be at most $2^{|s|}$ underlying forms r could have produced s. The same logic suffices to establish that there may be at most $2^{|s|}$ underlying forms when r is an optional rule (keeping in mind that in this case, as r may not apply at all, s is automatically a possible underlying form). The situation is slightly trickier when one is dealing with the underlying forms that can produce s relative to an m.e. rule-set M with k rules $r_1, r_2, \ldots, r_k$. In this case, the set of underlying forms is $\bigcup_{1=1}^{k} U_i$, where U_i is the set of strings such that for each $u \in U_i$, the result of applying rule r_i to u is s and there is no obligatory rule r_j, $j < i$, in M that successfully applies to u and creates a string other than s, i.e., the application of r_i to u could not have been blocked by the successful application of an obligatory rule occurring earlier than r_i in M. As each rule r_i essentially produces underlying forms independently of the other rules in M, there will be at most $k2^{|s|}$ underlying forms for s relative to the m.e. rule-set.

Given the above, rule-set R is run in reverse by applying the rules and m.e. rule-sets of $|R|$ in reverse to s in reverse of their order in R. Note that the number of lexical forms that can be produced by this process is bounded by $(|R|2^{|s|})^{|R|} = |R|^{|R|}2^{|s||R|}$, as is the number of intermediate forms that need to be computed to produce these lexical forms. As the derivation of each of these forms takes $O(|s||f|c)$ time and each lexical form can be checked against $DFA(D)$ to see if it is in D^+ in $O(|s|)$ time, the running time of the algorithm as a whole is $O(|R|^{|R|}2^{|s||R|}|s||f|c)$, which is fixed-parameter tractable relative to $|R|$ and $|s|$.

Proof of (4): As it is always the case that $|R| \leq |R_{m.e.}|\#(R_{m.e.})$, the result follows from (3) and Lemma 2.1.33.

Proof of (5): Modify step (1) in the algorithm given in part (6) of Theorem 4.1.16 such that each possible segment-string of length $|s|$ is tested against the lexicon DFA $DFA(D)$, and only those strings that are members of D^+ have their corresponding bits in *pos* set to 1. As each such string can be tested against $DFA(D)$ in $O(|s|)$ time, the algorithm as a whole runs in $O(|v|^{|f||s|}|s|c)$ time, which is fixed-parameter tractable relative to $|s|$, $|f|$, and $|v|$.

Proofs of (6) and (7): Follows from parts (7) and (8) of Theorem 4.1.16, the reduction in Lemma 4.1.12 from SSG-ENCODE to SSG-DECODE, and Lemma 2.1.25.

Proof of (8): Follows from the W[2]-hardness of $\langle k \rangle$-DOMINATING SET [DF95a], the reduction in Lemma 4.1.17 from DOMINATING SET to SSG-DECODE in which $|R_?| = 0$, $\#(R_{m.e.}) = 1$, $c = 2$, $|s| = k$, and $|f| = 4$, and Lemma 2.1.25.

Proof of (9): Follows from (8), the reduction in Lemma 4.1.14 from SSG-DECODE to SSG-DECODE in which $|v| = 2$, and Lemma 2.1.25.

Proofs of (10) and (11): Follow from (1) and (2) above and Lemma 2.1.35. ∎

4.1.3 Implications

All parameterized complexity results for problems SSG-ENCODE and SSG-DECODE that are either stated or implicit in the lemmas and theorems given in the previous section are shown in Tables 4.1, 4.2, 4.3, and 4.4. Consider the implications of these results for each problem in turn:

- SSG-ENCODE: The sources of polynomial-time intractability are $\{|R_?|\}$ and $\{|u|, |f|, |v|\}$. The mechanisms associated with these sources are the degree of nondeterministic choice in rule-set R and the set of all possible surface forms for the given lexical form. The aspects in the latter of these sources are defining properties of the segment-string representation, and while some of them may be small in practice, e.g., $|v|$ will typically be 2, none of these aspects can be eliminated to reduce the complexity of this problem. The case for $|R_?|$ is much weaker. Optional rules were introduced so that a single lexical form could generate multiple surface forms, each of which is characteristic of a different speech-situation, e.g., technical vs. oratorical vs. colloquial spoken English. However, in current linguistic practice, such matters are handled as part of phonetic processing (that is, the transformation of phonological surface forms into speech). Even if optional rules are still required to handle other phenomena, the results derived here suggest that every effort should be made to limit the number of such rules as SSG-ENCODE can be solved in polynomial time by the algorithm in part (3) of Theorem 4.1.16 if $|R_?|$ is bounded to a constant.

- SSG-DECODE: The sources of polynomial-time intractability are $\{|R|, |s|\}$ and $\{|s|, |f|, |v|\}$. The mechanism associated with both of these sources is the set of lexical forms that can be associated with a given surface form. As with SSG-ENCODE, the aspects in these sources are defining properties of simplified segmental grammars and hence none of them can be eliminated to reduce the complexity of the problem.

  In light of the results derived above, it can be seen that the lexicon plays a very interesting role in the NP-hardness of SSG-DECODE. The reduction in

Lemma 4.1.17 and the algorithm in part (3) of Theorem 4.1.18 show that optional rules are not required to generate nondeterministic choice in SSG-DECODE — rather, nondeterminism is implicit in the rule-reversal process, i.e., the guesses that have to be made about which appearances of a pattern in a form x are generated by a rule and which are already in the underlying form used to create x. This implicit nondeterminism is a well-known source of the computational difficulties that have plagued various attempts to directly reverse *SPE*-style rule systems to recover lexical forms from surface forms [KK94, Spr92]. However, a closer examination of the various reductions above suggests that in order for this implicit nondeterminism to be exploited, it must be "focused" relative to a lexicon D that can encode into D^+ every possible solution to a polynomial-time intractable or NP-hard problem (an observation originally made by Ristad [Ris90, Footnote 9, page 23]). There are at least two ways to do this:

- Let the lexicon be rich enough to directly encode the set of solutions into D^+.
- Augment the rule-set such that a simple lexicon can be used to indirectly encode the set of solutions. For instance, the lexicon D in any instance of SSG-DECODE can be reduced to a lexicon D' consisting of a single string of length $|s|$ composed of single-feature segments if the appropriate $|s||d|$ m.e. rule-sets are added to the beginning of R, where $|d|$ is the length of the longest string in D (*sketch*: nondeterministically "fill in" the blank form in D' from left to right by selecting and adding in up to $|s|$ elements of D; each rule-set uses optional rules to implement the choice at each step in a manner not unlike that in rule-set 1(a) of Lemma 4.1.11).

These two schemes are the ends of a spectrum of possibilities which show that there are subtle and surprising tradeoffs between D and R in the manners by which the computational power of implicit nondeterminism can be harnessed in simplified segmental grammars. One goal of future research should be to characterize these tradeoffs in terms of aspects of D and R relative to the computational complexity SSG-DECODE and to see if any of these tradeoffs lead to algorithms for this problem that are efficient in practice.

An important open question in the analyses of both SSG-ENCODE and SSG-DECODE is whether or not $\{|f|, |v|\}$ is a source of polynomial-time intractability. Intuition suggests that the ability to succinctly encode via F an implicit segment-alphabet S of exponential size $|v|^{|f|}$ should render a problem polynomial-time intractable all by itself. However, such exponential-size alphabets are not necessary — indeed, the reader can verify that the sizes of the segment-alphabets implicit in each of the reductions given above are polynomial in the size of their given instances, and low-order polynomial at that, e.g., $|S| = ((|Q|+1) \times (|\Sigma|+1) \times 3)$ in the reduction given in Lemma 4.1.11. Answering this question is yet another topic for future research.

In retrospect, it is surprising that so many of the aspects considered here are not by themselves sources of polynomial-time intractability. Indeed, SSG-ENCODE and

SSG-DECODE remain NP-hard when the values of many of these aspects are bounded to small constants, e.g., $c = 2$. This somewhat odd state of affairs also holds for the other phonological theories examined in this thesis, and will be discussed in Section 4.7.

When rules that add or delete segments are allowed, all NP- and W-hardness results derived above still hold (as such rules would imply $c_p \geq 0$ and the results above hold when $c_p = 0$). As shown in Appendix B, such rules allow certain results to hold in more restricted cases or hardness to be established relative to higher levels of the W hierarchy. It seems inevitable that certain parameterized problems that were in FPT will be shown to be W-hard when such rules are allowed. The full extent of this change will not be addressed here. For now, simply observe that the FPT algorithm for $\langle |R_?| \rangle$-SSG-ENCODE given in part (3) of Theorem 4.1.16 will continue to work as stated when rules that add or delete segments are allowed (albeit with a slightly higher running time), whereas all other FPT algorithms proposed in the previous section seem to require the addition of aspect c_p to the parameter to remain fixed-parameter tractable.

Consider now what these results have to say about the various search problems associated with simplified segmental grammars. First, note the following relationships between SSG-ENCODE and SSG-DECODE and their associated search problems:

- Any instance of SSG-ENCODE can be solved by a single call to $\text{CHK}_{\textbf{SSG}}$; moreover, any instance of $\text{CHK}_{\textbf{SSG}}$ can be solved by any algorithm for SSG-ENCODE.
- Any instance of SSG-DECODE can be solved by a single call to $\text{DEC}_{\textbf{SSG}}$; moreover, any instance of $\text{DEC}_{\textbf{SSG}}$ can be solved by any algorithm for SSG-DECODE.

Hence, modulo various conjectures, $\text{CHK}_{\textbf{SSG}}$ and $\text{DEC}_{\textbf{SSG}}$ do not have polynomial-time algorithms and have the same sources of polynomial-time intractability as their corresponding decision problems. The case of problem $\text{ENC}_{\textbf{SSG}}$ is more interesting because, by the algorithm in part (3) of Theorem 4.1.16, it is solvable in polynomial time (namely, pick *any* resolution of the ?-positions in R to derive a surface form associated with the given lexical form). The situation here, in which encoding is easy, is the opposite of that for all other theories examined in this thesis. It is tempting to suggest that this is characteristics of rule-based phonological theories, but the results in Section 4.3 show otherwise. The reasons for this and other patterns in the computational complexity of the search problems associated with the phonological theories examined in this thesis will become clearer in light of results derived for these other theories, and will be discussed in Section 4.7.

The results derived above have three practical applications:

1. **Covering-Grammar Approximations for General *SPE* Decoding Problems**: One solution to the *SPE* decoding problem that has been adopted in practice is to use a *covering grammar* to create a set of plausible lexical forms for a given surface form, and then to determine if any of those lexical forms could have produced the surface form [KK94, Spr92]. However, as the second part of this procedure is equivalent to $\text{CHK}_{\textbf{SSG}}$ and hence cannot have a polynomial-time algorithm unless $P = NP$,

it seems unlikely that such a procedure can run in polynomial time (unless the number of optional rules is restricted; see the discussion at the beginning of this section).

2. **FST Implementations of General *SPE*-style Rule Systems**: Kaplan and Kay ([KK94]; see also [KCGS96, XFST]) have proposed a calculus of basic operations on FST and have shown how these operations can be used to implement various types of rule-based phonological systems operating on either symbol- or segment-strings as the composition of FST. As *SPE*-style rule systems are one such system [KK94, Sections 5 and 6] and simplified segmental grammars are a restricted type of *SPE*-style rule system, the results above for SSG-ENCODE show that the following problem is NP-hard:

   FINITE-STATE TRANSDUCER COMPOSITION
   Instance: A set A of FST over an alphabet Σ, an order O on A, and strings $u, s \in \Sigma^*$.
   Question: Is the string-pair u/s accepted by the FST formed by composing the FST in A in the order O?

   Note that as the process described in [KK94] that constructs FST from phonological rules creates FST that can add or delete segments, this NP-hardness result only holds relative to sets of FST that allow insertions and deletions. However, the results in Section 4.3 show that this problem remains NP-hard when additions and deletions are not allowed.

3. **FST Implementations of Obligatory-Rule-Only *SPE*-style Rule Systems**: Laporte [Lap97] has described an FST implementation of a rule-based system operating on symbol-strings in which all rules are obligatory and rules are allowed to have contexts specified by FSA. As the segment-string contexts used in simplified grammars can be encoded as simple linear FSA, the reduction in Lemma 4.1.17 and the results in (2) show that the decoding problem associated with such a system is NP-hard. Though the encoding problem in such a system when contexts are specified as symbol-strings is solvable in polynomial time (this is implicit in the proof of part (3) of Theorem 4.1.16), it would be interesting to know if this problem becomes NP-hard when FSA contexts are allowed.

Note that as many of the aspects considered here for simplified segmental grammars are mapped onto aspects of the systems described above by the reductions implicit in (1) – (3), the results derived in this section also imply various W-hardness results and FPT algorithms for these systems.

The above suggests many directions for future research. Several of the more intriguing directions are:

1. Characterize the relationship between the lexicon and the rule-set in the computational complexity of SSG-DECODE.

2. Characterize the relationship between the structure of the rule-set and structure of individual rules in the computational complexity of both SSG-ENCODE and SSG-DECODE.

3. Characterize the computational complexity of the FINITE-STATE TRANSDUCER COMPOSITION problem in terms of trade-offs between various aspects of the individual FST being composed, e.g., type of (non)determinism, structure of the regular relation encoded by the FST.

The research in (1) may be aided by reformulating all segmental rewriting rules and the lexicon as FST, as is suggested in Section 4.3.3. The research in (3) could be part of a more general investigation of the computational complexity of the operations in Kaplan and Kay's FST calculus [KK94] (which was subsequently extended to create the Xerox Finite State Calculus [KCGS96, XFST]). As such, the results derived above for general FST composition should be compared to those in Sections 4.3 and 4.4 for ϵ-free FST composition and intersection.

Table 4.1: The Parameterized Complexity of the SSG-Encode Problem.

Parameter	Segment size ($\|f\|$,$\|v\|$)			
	(Unb,Unb)	(Unb,Prm)	(Prm,Unb)	(Prm,Prm)
–	NP-hard	$\notin XP$	$\notin XP$	???
$\|R\|$	FPT	FPT	FPT	FPT
$\|R_?\|$	FPT	FPT	FPT	FPT
$\|R_{m.e.}\|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	???
$\#(R_{m.e.})$	$\notin XP$	$\notin XP$	$\notin XP$	???
c	$\notin XP$	$\notin XP$	$\notin XP$	???
$\|u\|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$\|R\|$, $\|R_?\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_{m.e.}\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\#(R_{m.e.})$	FPT	FPT	FPT	FPT
$\|R\|$, c	FPT	FPT	FPT	FPT
$\|R\|$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R_?\|$, $\|R_{m.e.}\|$	FPT	FPT	FPT	FPT
$\|R_?\|$, $\#(R_{m.e.})$	FPT	FPT	FPT	FPT
$\|R_?\|$, c	FPT	FPT	FPT	FPT
$\|R_?\|$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R_{m.e}\|$, $\#(R_{m.e.})$	FPT	FPT	FPT	FPT
$\|R_{m.e}\|$, c	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	???
$\|R_{m.e}\|$, $\|u\|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$\#(R_{m.e.})$, c	$\notin XP$	$\notin XP$	$\notin XP$	???
$\#(R_{m.e.})$, $\|u\|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
c, $\|u\|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\#(R_{m.e.})$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, c	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_{m.e}\|$, $\#(R_{m.e.})$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_{m.e}\|$, c	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_{m.e}\|$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\#(R_{m.e})$, c	FPT	FPT	FPT	FPT
$\|R\|$, $\#(R_{m.e})$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, c, $\|u\|$	FPT	FPT	FPT	FPT

Table 4.2: The Parameterized Complexity of the SSG-ENCODE Problem (Cont'd).

Parameter	Segment size ($\|f\|$,$\|v\|$)			
	(Unb,Unb)	(Unb,Prm)	(Prm,Unb)	(Prm,Prm)
$\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$	FPT	FPT	FPT	FPT
$\|R_?\|$, $\|R_{m.e.}\|$, c	FPT	FPT	FPT	FPT
$\|R_?\|$, $\|R_{m.e.}\|$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R_?\|$, $\#(R_{m.e.})$, c	FPT	FPT	FPT	FPT
$\|R_?\|$, $\#(R_{m.e.})$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R_?\|$, c, $\|u\|$	FPT	FPT	FPT	FPT
$\|R_{m.e.}\|$, $\#(R_{m.e.})$, c	FPT	FPT	FPT	FPT
$\|R_{m.e.}\|$, $\#(R_{m.e.})$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R_{m.e.}\|$, c, $\|u\|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$\#(R_{m.e.})$, c, $\|u\|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, c	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\#(R_{m.e.})$, c	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\#(R_{m.e.})$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, c, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, c	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_{m.e.}\|$, c, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\#(R_{m.e.})$, c, $\|u\|$	FPT	FPT	FPT	FPT
$\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, c	FPT	FPT	FPT	FPT
$\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R_?\|$, $\|R_{m.e.}\|$, c, $\|u\|$	FPT	FPT	FPT	FPT
$\|R_?\|$, $\#(R_{m.e.})$, c, $\|u\|$	FPT	FPT	FPT	FPT
$\|R_{m.e.}\|$, $\#(R_{m.e.})$, c, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, c	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, c, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\#(R_{m.e.})$, c, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, c, $\|u\|$	FPT	FPT	FPT	FPT
$\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, c, $\|u\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, c, $\|u\|$	FPT	FPT	FPT	FPT

Table 4.3: The Parameterized Complexity of the SSG-DECODE Problem.

Parameter	Segment size ($\|f\|$,$\|v\|$)			
	(Unb,Unb)	(Unb,Prm)	(Prm,Unb)	(Prm,Prm)
–	NP-hard	$\notin XP$	$\notin XP$	???
$\|R\|$	???	???	???	???
$\|R_?\|$	$\notin XP$	$\notin XP$	$\notin XP$	???
$\|R_{m.e.}\|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	???
$\#(R_{m.e.})$	$\notin XP$	$\notin XP$	$\notin XP$	???
c	$\notin XP$	$\notin XP$	$\notin XP$	???
$\|s\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$\|R\|$, $\|R_?\|$	???	???	???	???
$\|R\|$, $\|R_{m.e.}\|$	???	???	???	???
$\|R\|$, $\#(R_{m.e.})$	???	???	???	???
$\|R\|$, c	???	???	???	???
$\|R\|$, $\|s\|$	FPT	FPT	FPT	FPT
$\|R_?\|$, $\|R_{m.e.}\|$	???	???	???	???
$\|R_?\|$, $\#(R_{m.e.})$	$\notin XP$	$\notin XP$	$\notin XP$	???
$\|R_?\|$, c	$\notin XP$	$\notin XP$	$\notin XP$	???
$\|R_?\|$, $\|s\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$\|R_{m.e}\|$, $\#(R_{m.e.})$	???	???	???	???
$\|R_{m.e}\|$, c	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	???
$\|R_{m.e}\|$, $\|s\|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$\#(R_{m.e.})$, c	$\notin XP$	$\notin XP$	$\notin XP$	???
$\#(R_{m.e.})$, $\|s\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
c, $\|s\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$	???	???	???	???
$\|R\|$, $\|R_?\|$, $\#(R_{m.e.})$	???	???	???	???
$\|R\|$, $\|R_?\|$, c	???	???	???	???
$\|R\|$, $\|R_?\|$, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_{m.e}\|$, $\#(R_{m.e.})$	???	???	???	???
$\|R\|$, $\|R_{m.e}\|$, c	???	???	???	???
$\|R\|$, $\|R_{m.e}\|$, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\#(R_{m.e})$, c	???	???	???	???
$\|R\|$, $\#(R_{m.e})$, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, c, $\|s\|$	FPT	FPT	FPT	FPT

Table 4.4: The Parameterized Complexity of the SSG-DECODE Problem (Cont'd).

Parameter	Segment size ($\|f\|$,$\|v\|$)			
	(Unb,Unb)	(Unb,Prm)	(Prm,Unb)	(Prm,Prm)
$\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$	???	???	???	???
$\|R_?\|$, $\|R_{m.e.}\|$, c	???	???	???	???
$\|R_?\|$, $\|R_{m.e.}\|$, $\|s\|$	???	???	???	FPT
$\|R_?\|$, $\#(R_{m.e.})$, c	$\not\in XP$	$\not\in XP$	$\not\in XP$	???
$\|R_?\|$, $\#(R_{m.e.})$, $\|s\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$\|R_?\|$, c, $\|s\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$\|R_{m.e.}\|$, $\#(R_{m.e.})$, c	???	???	???	???
$\|R_{m.e.}\|$, $\#(R_{m.e.})$, $\|s\|$	FPT	FPT	FPT	FPT
$\|R_{m.e.}\|$, c, $\|s\|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$\#(R_{m.e.})$, c, $\|s\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$	???	???	???	???
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, c	???	???	???	???
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\#(R_{m.e.})$, c	???	???	???	???
$\|R\|$, $\|R_?\|$, $\#(R_{m.e.})$, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, c, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, c	???	???	???	???
$\|R\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_{m.e.}\|$, c, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\#(R_{m.e.})$, c, $\|s\|$	FPT	FPT	FPT	FPT
$\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, c	???	???	???	???
$\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, $\|s\|$	FPT	FPT	FPT	FPT
$\|R_?\|$, $\|R_{m.e.}\|$, c, $\|s\|$	???	???	???	FPT
$\|R_?\|$, $\#(R_{m.e.})$, c, $\|s\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$\|R_{m.e.}\|$, $\#(R_{m.e.})$, c, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, c	???	???	???	???
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, c, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\#(R_{m.e.})$, c, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, c, $\|s\|$	FPT	FPT	FPT	FPT
$\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, c, $\|s\|$	FPT	FPT	FPT	FPT
$\|R\|$, $\|R_?\|$, $\|R_{m.e.}\|$, $\#(R_{m.e.})$, c, $\|s\|$	FPT	FPT	FPT	FPT

4.2 A Most Useful Special Case: The Bounded DFA Intersection Problem

The remaining four phonological theories examined in this thesis explicitly use finite-state automata to implement their phonological mechanisms. The analyses of these theories will be simplified by reference to the following problem:

BOUNDED DFA INTERSECTION (BDFAI)
Instance: A set A of DFA over an alphabet Σ, a positive integer k.
Question: Is there a string in Σ^k that is accepted by all DFA in A?

As sections 4.3.2, 4.4.2, 4.5.2, and 4.6.2 will show, this problem is a special case of the computational problems underlying FST-based rule systems, the KIMMO system, Declarative Phonology, and Optimality Theory. Hence, complexity-theoretic hardness results derived for problem BDFAI (as well as certain FPT algorithms) also hold for the problems associated with these theories.

The systematic parameterized analysis given in this section will focus on the following aspects:

- The number of DFA in A ($|A|$).
- The required length of the result-strings (k).
- The maximum number of states in any DFA in A ($|Q|$).
- The size of the alphabet ($|\Sigma|$).

Hardness results will be derived via reductions from problem DOMINATING SET (see page 87) and the following problem:

LONGEST COMMON SUBSEQUENCE (LCS) [GJ79, Problem SR10]
Instance: A set of strings $X = \{x_1, \ldots, x_k\}$ over an alphabet Σ and a positive integer m.
Question: Is there a string $y \in \Sigma^m$ that is a subsequence of x_i for $i = 1, \ldots, k$?

Lemma 4.2.1 *LCS $\leq_m$ BDFAI.*

Proof: Given an instance $\langle X, k, \Sigma, m\rangle$ of LCS, construct the following instance $\langle A', \Sigma', k'\rangle$ of BDFAI: Let $\Sigma' = \Sigma$ $k' = m$, and A' be the set of DFA created by applying to each string $x \in X$ the $O(|x| \max(|x|, |\Sigma|))$ time algorithm given in [Bae91] for creating a partial DFA that recognizes all subsequences of a given string. An example of such a subsequence DFA is given in Figure 4.3. Transform each subsequence DFA into a total DFA by adding a new non-final state *Fail* and the appropriate transitions as described in Section 2.2.3. This construction can be done in time polynomial in the size of the given instance of LCS.

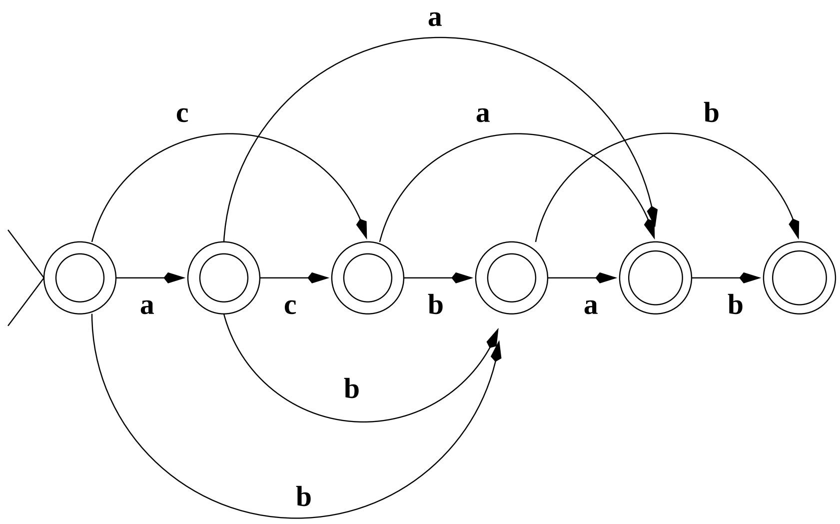


Figure 4.3: A Subsequence Deterministic Finite-State Acceptor. The subsequence DFA shown above recognizes all subsequences of the string $x = acbab$ and was constructed using the algorithm given in [Bae91].

To see that this construction is a many-one reduction, note that any solution to the constructed instance of BDFAI is a string of length k' that is accepted by each DFA in A', and hence is a subsequence of length m of each string in X. Moreover, any solution to the given instance of LCS is a string of length m which is a subsequence of each string in X, and hence is a string of length k' that is accepted by each DFA in A'. Thus, the constructed instance of BDFAI has a solution if and only if the given instance of LCS has a solution.

Note that in the constructed instance of BDFAI, $|A'| = k$, $k' = m$, and $|\Sigma'| = |\Sigma|$. ▮

Lemma 4.2.2 DOMINATING SET $\leq_m$ *BDFAI.*

Proof: Given an instance $\langle G = (V, E), k\rangle$ of DOMINATING SET, construct the following instance $\langle A', \Sigma', k'\rangle$ of BDFAI: Let Σ' be an alphabet such that $|\Sigma'| = |V|$ and there is a function $sym : V \mapsto \Sigma'$ such that each vertex $v \in V$ has a distinct corresponding symbol $sym(v)$ in Σ', and let $k' = k$. For each $v \in V$, let $adj(v)$ be the set of vertices in V that are adjacent to v in G (including v itself) and $nonadj(v) = V - adj(v)$. For each vertex $v \in V$, construct a two-state DFA $A_v = \langle\{q_1, q_2\}, \Sigma', \delta, q_1, \{q_2\}\rangle$ with transition relation $\delta = \{(q_1, sym(v'), q_1) \mid v' \in nonadj(v)\} \cup \{(q_1, sym(v'), q_2) \mid v' \in adj(v)\} \cup \{(q_2, sym(v'), q_2) \mid v' \in V\}$. Note that a string x is accepted by A_v if and only if x contains at least one symbol corresponding to a vertex that is adjacent to v in G. Let A' be the set consisting of all DFA A_v corresponding to vertices $v \in V$. This construction can be done in time polynomial in the length of the given instance of DOMINATING SET.

To see that this construction is a many-one reduction, note that any solution to the constructed instance of BDFAI is a string of length k' which is accepted by each DFA in

(a)

State	Alphabet Σ					
	a	b	c	d	e	f
q	q_a	q_b	q_c	q_d	q_e	$\{q_f, q'_f\}$

(b)

	Alphabet Σ					
	a	b	c	d	e	f
Code	000	001	010	011	100	101

Figure 4.4: The Decoding Tree Construction from Lemma 4.2.3. a) Transition-entry for state q in some hypothetical nondeterministic FSA over alphabet $\Sigma = \{a, b, c, d, e, f\}$. b) Binary codewords assigned to symbols of Σ. As $|\Sigma| = 6$, the codeword length $l = \lceil \log_2 |\Sigma| \rceil = \lceil \log_2 6 \rceil = 3$.

A', and hence corresponds to a set of size k from V that contains, for each v in V, at least one vertex that is adjacent or equal to v; moreover, any solution to the given instance of DOMINATING SET is a set of vertices of size k which such that each vertex in G is adjacent or equal to at least one vertex in that set, and hence corresponds to a string of k' symbols that is accepted by each DFA in A'. Thus, the constructed instance of BDFAI has a solution of size k' if and only if the given instance of DOMINATING SET has a solution of size k.

Note that in the constructed instance of BDFAI, $|A'| = |\Sigma'| = |V|$, $k' = k$, and $|Q| = 2$. ∎

Lemma 4.2.3 *BDFAI* $\leq_m$ *BDFAI such that* $|\Sigma| = 2$.

Proof: Given an instance $\langle A, \Sigma, k\rangle$ of BDFAI, construct the following instance $\langle A', \Sigma', k'\rangle$ of BDFAI: Let $\Sigma' = \{0, 1\}$ and assign each symbol in Σ a binary codeword of fixed length $l = \lceil \log |\Sigma| \rceil$. For each DFA $a \in A$, create a DFA $a' \in A'$ by adjusting Q and δ such that each state q and its outgoing transitions in a is replaced with a "decoding tree" on $2^l - 1$ states in a' that uses l bits to connect q to the appropriate states (see example in Figures 4.4 and 4.5). If $|\Sigma| \neq 2^l$, there will be l-bit strings that do not correspond to any symbol in Σ; in this case, create a new non-final state $Fail$ to which all such l-bit strings are directed and from which subsequent processing cannot escape (this state is analogous to state $Fail$ created in the transformation of partial to total DFA described in Section 2.2.3). Finally, set $k' = lk$. This construction can be done in time polynomial in the size of the given instance of BDFAI.

To see that this construction is a many-one reduction, note that any solution to the constructed instance of BDFAI is a string on $k' = kl$ bits which is accepted by each DFA in

(c)

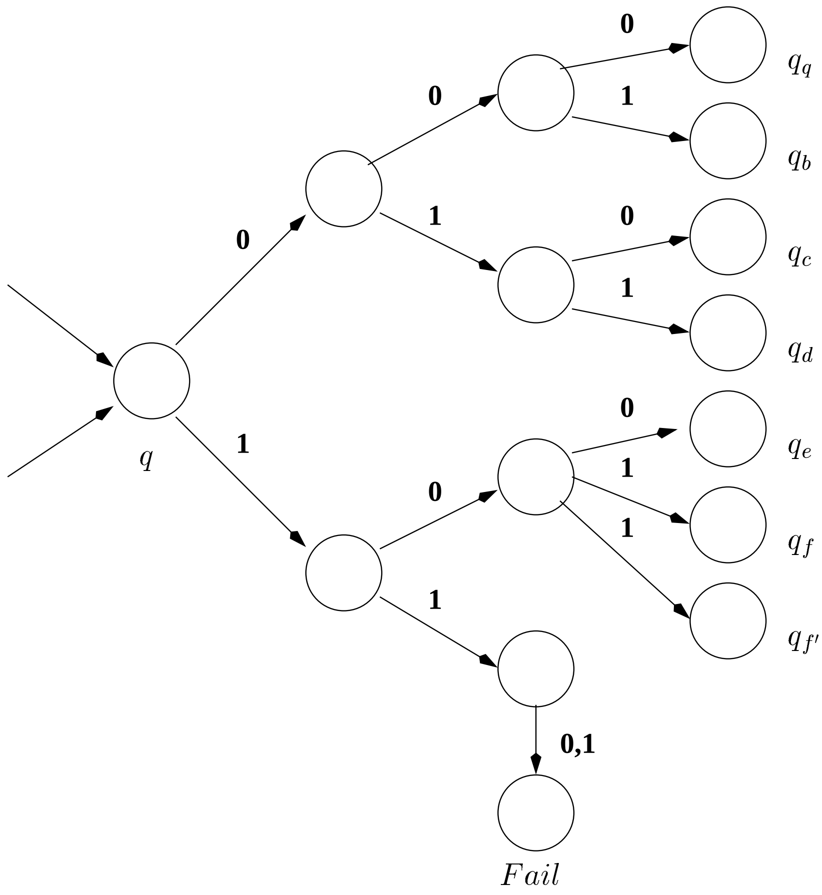


Figure 4.5: The Decoding Tree Construction from Lemma 4.2.3 (Cont'd). c) Decoding tree for q relative to transition-entry specified in (a) and the symbol-codes given in (b).

A', and which can be transformed by substitution of the appropriate symbol in Σ for each l-bit codeword into a string of length k that is accepted by each DFA in A. Moreover, any solution to the given instance of BDFAI is a string of k symbols accepted by every DFA in A, which can by substitution of the appropriate l-bit codeword be transformed into a string of length $k' = kl$ bits that is accepted by each DFA in A'. Thus, the constructed instance of BDFAI has a solution if and only if the given instance of BDFAI has a solution.

Note that in the constructed instance of BDFAI, $|A'| = |A|$, $|k'| = kl = k\lceil\log|\Sigma|\rceil$, $|Q'| \leq (|\Sigma| - 1)|Q| + 1$, and $|\Sigma'| = 2$. ∎

Theorem 4.2.4

1. *BDFAI is NP-hard when* $|Q| = 2$.

2. *BDFAI is NP-hard when $|\Sigma| = 2$.*

3. *$\langle k, |\Sigma|\rangle$-BDFAI is in FPT.*

4. *$\langle |A|, |Q|\rangle$-BDFAI is in FPT.*

5. *$\langle |A|, k\rangle$-BDFAI is $W[1]$-hard.*

6. *$\langle k, |Q|_2\rangle$-BDFAI is $W[2]$-hard.*

7. *$\langle |A|, |\Sigma|_2\rangle$-BDFAI is $W[t]$-hard for all $t \geq 1$.*

8. *$\langle |Q|_2\rangle$-BDFAI $\notin XP$ unless $P = NP$.*

9. *$\langle |\Sigma|_2\rangle$-BDFAI $\notin XP$ unless $P = NP$.*

Proof:

Proof of (1): Follows from the NP-hardness of DOMINATING SET [GJ79, Problem GT2] and the reduction in Lemma 4.2.2 from DOMINATING SET to BDFAI in which $|Q| = 2$.

Proof of (2): Follows from (1) and the reduction in Lemma 4.2.3 from BDFAI to BDFAI in which $|\Sigma| = 2$.

Proof of (3): Follows from the algorithm that generates all $|\Sigma|^k$ possible k-length strings over alphabet Σ and checks each string in $O(|A|k)$ time to see whether that string is accepted by each of the DFA in A. The algorithm as a whole runs in $O(|\Sigma|^k k|A|)$ time, which is fixed-parameter tractable relative to k and $|\Sigma|$.

Proof of (4): Follows from the algorithm that constructs the intersection DFA of all DFA in A and the $k+1$-state DFA that recognizes all strings in Σ^k, i.e., a DFA $\langle Q, \Sigma, \delta, s, F\rangle$ such that $Q = \{q_1, q_2, \ldots, q_{k+1}\}$, $s = q_1$, $F = \{q_{k+1}\}$, and $\delta(q_i, x) = q_{i+1}$ for i, $1 \leq i \leq k$, and $x \in \Sigma$, and then applies depth-first search to the transition diagram for this intersection DFA to determine if any of its final states are reachable from its start state. By Table 2.3, the intersection DFA can be created in $O(|Q|^{|A|+1}(k+1)|\Sigma|^2) = O(|Q|^{|A|+1}2k\,|\Sigma|^2) = O(|Q|^{|A|+1}k|\Sigma|^2)$ time. As the graph $G = (V, E)$ associated with the transition diagram of this intersection DFA has $|V| \leq (k+1)|Q|^{|A|} \leq 2k|Q|^{|A|}$ states and $|A| \leq (k+1)|Q|^{|A|}|\Sigma| \leq 2k|Q|^{|A|}|\Sigma|$ arcs and depth-first search runs in $O(|V| + |A|)$ time, the algorithm as a whole runs in $O(|Q|^{|A|+1}k|\Sigma|^2)$ time, which is fixed-parameter tractable relative to $|A|$ and $|Q|$.

Proof of (5): Follows from the $W[1]$-completeness of $\langle k, m\rangle$-LCS [BDFW95], the reduction in Lemma 4.2.1 from LCS to BDFAI in which $|A'| = k$ and $k' = m$, and Lemma 2.1.25.

Proof of (6): Follows from the $W[2]$-completeness of $\langle k\rangle$-DOMINATING SET [DF95a], the reduction in Lemma 4.2.2 from DOMINATING SET to BDFAI in which $k' = k$ and $|Q| = 2$, and Lemma 2.1.25.

Parameter	Alphabet Size $\|\Sigma\|$	
	Unbounded	Parameter
–	NP-hard	$\not\in XP$
$\|A\|$	$W[t]$-hard	$W[t]$-hard
k	$W[2]$-hard	FPT
$\|Q\|$	$\not\in XP$	???
$\|A\|$, k	$W[1]$-hard	FPT
$\|A\|$, $\|Q\|$	FPT	FPT
k, $\|Q\|$	$W[2]$-hard	FPT
$\|A\|$, k, $\|Q\|$	FPT	FPT

Table 4.5: The Parameterized Complexity of the BOUNDED DFA INTERSECTION Problem.

Proof of (7): Follows from the $W[t]$-hardness of $\langle k \rangle$-LCS for $t \geq 1$ [BDFW95], the reduction in Lemma 4.2.1 from LCS to BDFAI in which $|A'| = k$, the reduction in Lemma 4.2.3 from BDFAI to BDFAI in which $|A'| = |A|$ and $|\Sigma'| = 2$, and Lemma 2.1.25.

Proofs of (8) and (9): Follow from (1) and (2) and Lemma 2.1.35. ∎

All parameterized complexity results for BDFAI that are either stated or implicit in the theorem above are shown in Table 4.5. An interesting way of viewing the sources of polynomial-time intractability listed in this table arises when BDFAI is interpreted as a type of set intersection problem, i.e., given a collection of set-encodings, determine if the intersection of the sets associated with these encodings is empty. Under this interpretation of BDFAI, the strings in Σ^* are the possible elements of the sets, the DFA in A are set-encodings, and the languages associated with these DFA are the sets encoded by these DFA. The sources of polynomial-time intractability derived above can be then be seen as general strategies for computing the intersections of such encoded sets:

1. Create the encoding corresponding to the intersection of the encoded sets and and the encoded set corresponding to Σ^k and apply the efficient (in the size of the encoding) emptiness-check operation to that encoding ($\{|A|, |Q|\}$).
2. Check each element in Σ^k against each of the encoded sets by applying the efficient (in the size of the encoding) element-check operation ($\{|\Sigma|, k\}$).

One can also apply such reasoning in reverse, and use strategies for solving the set-intersection formulation of BDFAI to propose possible sources of polynomial-time intractability for BDFAI. For instance, consider the following strategy suggested by (2) above:

3. Find the smallest of the encoded sets and check each element of that set against each of the other encoded sets by applying the efficient (in the size of the encoding) element-check operation.

Given that T_{fs} and T_{ls} are the times required to find and list the elements of the smallest of the encoded sets, the running time of the algorithm in (3) is $O(T_{fs} + T_{ls}(|A| - 1)|u|)$. In light of the NP-hardness of BDFAI, this algorithm suggests possible sources of polynomial-time intractability in BDFAI that are based on some combination of aspects associated with the following:

1. The complexity of finding the smallest encoded set.
2. The size of the smallest encoded set.
3. The complexity of listing the elements of the smallest encoded set.

The possibility that the NP-hardness of BDFAI may be due to properties of the succinct encoding of very large sets by DFA is intriguing. Perhaps these intuitions can be formalized by rephrasing these DFA (or rather the constraints implicit in the sets of strings accepted by these DFA) as formulas in logic along the lines described in [Str94]. In addition to providing a more formal characterization of the sets encoded by DFA, such a formulation might make BDFAI amenable to known Dichotomy Theorems [Cre95, FV93, Sch78] which, for various logics, rigorously characterize the types of constraint-systems that are solvable in polynomial time. This is an interesting topic for future research, not least of all because BDFAI seems to be a special case of the encoding and decoding problems associated with the four phonological theories examined in the remainder of this thesis.

4.3 FST-Based Rule Systems

4.3.1 Background

Several phonological theories that emerged in the years after Chomsky and Halle's *SPE* model attempted to restrict both the generative capacity and computational complexity inherent in this model by rephrasing its mechanisms in terms of finite-state automata. This line of research was initiated by Johnson [JoC72], who noted that under the most commonly-used schemes of rule application, most phonological rules proposed by linguists could be encoded as finite-state transducers (FST). In the early 1980's, Kaplan and Kay [KK94] subsequently described but did not (at that time) implement a system in which were rules encoded as FST and these FST were combined by the composition operation to simulate the serial application of these rules to a lexical form to produce a surface form.[2] These proposals were subsequently implemented in the Xerox Finite State Calculus and Tool [KCGS96, XFST] and in various other systems (see papers in [RS97a] and references). Each such implementation is based on particular types of FST and rules. In order to draw general conclusions about the computational complexity of such systems, the section will focus on generic FST-based rule systems.

A FST-based rule system $g = \langle A, O, D, \Sigma \rangle$ consists of the following:

1. A lexicon $D \subseteq \Sigma^+$ for some alphabet Σ.
2. A set A of i/o-deterministic FST, all of whose input and output alphabets are Σ, and a composition-order O on these FST.

Though there has been research on the generative complexity of such FST-based systems [KK94, RS97b] and various implementations, there has been no work done to date on the computational complexity of FST-based rule systems. This section will contain the first presentation of such results. The analysis will focus on the following problems:

FST-ENCODE
Instance: A FST-based rule system $g = \langle A, O, D, \Sigma \rangle$, a string $u \in \Sigma^+$, and a string $s \in \Sigma^+$ such that $s = |u|$.
Question: Is s generated when the composition of A as specified by O is applied to u?

FST-DECODE
Instance: A FST-based rule system $g = \langle A, O, D, \Sigma \rangle$ and a string $s \in \Sigma^+$.
Question: Is there a string $u \in \Sigma^{|s|}$ such that $u \in D^+$ and s is generated when the composition of A as specified by O is applied to u?

Note that these problems are restricted in that they do not allow individual FST to delete or add symbols when transforming one string into another. In these problems, the lexicon D

[2] The mismatched dates in the preceding sentence are not a misprint. For over a decade until its publication in 1994, drafts of [KK94] circulated privately but were cited widely, earning this work a certain notoriety within the computational linguistics community [RRBP92, page 20].

will be specified as a DFA $DFA(D) = \langle Q_D, \Sigma, \delta_D, s_D, F_D\rangle$ on $|Q_D|$ states which recognizes the language D^+.

4.3.2 Analysis

The systematic parameterized analysis in this section will focus on the following aspects:

- The number of FST in A ($|A|$).
- The length of the given lexical / surface form ($|u|$ / $|s|$).
- The maximum number of states in any FST in A ($|Q|$).
- The size of the alphabet ($|\Sigma|$).

Consider the following reductions.

Lemma 4.3.1 *BDFAI* $\leq_m$ FST-ENCODE.

Proof: Given an instance $\langle A, \Sigma, k\rangle$ of BDFAI, construct the following instance $\langle A', O', D', \Sigma', u', s'\rangle$ of FST-ENCODE: Let $\Sigma' = \Sigma \cup \{\Delta\}$ for some symbol $\Delta \not\in \Sigma$, $u' = s' = \Delta^k$, and $D' = \{u'\}$. Given a DFA $a = \langle Q, \Sigma, \delta, s, F\rangle$, let $FST_u(a) = \langle Q, \Sigma, \Sigma, \delta_F, s, F\rangle$ be the FST such that $\delta_F = \{(q, x, x, q') \mid (q, x, q') \in \delta\}$. Let A'' be the set consisting of all FST $FST_u(a)$ corresponding to DFA $a \in A$. Construct the following two special FST:

- Let $FST_{init} = \langle \{q_1\}, \{\Delta\}, \Sigma, \delta, q_1, \{q_1\}\rangle$ be the single-state FST such that $\delta = \{(q_1, \Delta, x, q') \mid x \in \Sigma\}$, i.e., FST_{init} has the associated relation $R = \{\Delta^{|y|}/y \mid y \in \Sigma^*\}$.
- Let $FST_{final} = \langle \{q_1\}, \Sigma, \{\Delta\}, \delta, q_1, \{q_1\}\rangle$ be the single-state FST such that $\delta = \{(q_1, x, \Delta, q') \mid x \in \Sigma\}$, i.e., FST_{final} has the associated relation $R = \{x/\Delta^{|x|} \mid x \in \Sigma^*\}$.

Let $A' = A'' \cup \{FST_{init}\} \cup \{FST_{final}\}$. Let O' be an ordering on A' such that FST_{init} is the first FST in O, FST_{final} is the last FST in O, and the FST in A'' are ordered in some arbitrary manner between FST_{init} and FST_{final} in O'. This construction can be done in time polynomial in the size of the given instance of BDFAI.

To see that this construction is a many-one reduction, note that if the answer to the constructed instance of FST-ENCODE is "yes", then there is some string $x \in \Sigma^{|u'|}$ generated by FST_{init} from u' that is transformed by each FST in A' into x and transformed by FST_{final} into s', which corresponds to a string $x \in \Sigma^k$ that is accepted by each DFA in A. Moreover each solution to the given instance of BDFAI is a string $x \in \Sigma^k$ that is accepted by every DFA in A, which corresponds to a string $x \in \Sigma^{|u'|}$ such that x can be generated by FST_{init} from u', transformed into x by each FST in A', and transformed by FST_{final} into s'. Thus, the given instance of BDFAI has a solution if and only if the constructed instance of FST-ENCODE has a solution.

Note that in the constructed instance of FST-ENCODE, $|A'| = |A| + 2$, $|u'| = k$, $|Q'| = \max(|Q|, 1) = |Q|$, and $|\Sigma'| = |\Sigma| + 1$. ∎

Lemma 4.3.2 FST-ENCODE $\leq_m$ FST-DECODE.

Proof: Given an instance $\langle A, O, D, \Sigma, u, s\rangle$ of FST-ENCODE, let the constructed instance of FST-DECODE be $\langle A, O, D', \Sigma, s\rangle$ where $D' = \{u\}$. This construction can be done in time polynomial in the size of the given instance of FST-ENCODE. Note that, as FST cannot add or delete symbols, the only member of D'^{+} that can possibly be transformed into s is u. Hence, the given instance of FST-ENCODE has a solution if and only if the constructed instance of FST-DECODE has a solution.

Note that all aspects in the constructed instance of FST-DECODE have the same values as those in the given instance of FST-ENCODE. ∎

Theorem 4.3.3

1. FST-ENCODE *is* NP*-hard when* $|Q| = 2$.
2. FST-ENCODE *is* NP*-hard when* $|\Sigma| = 3$.
3. $\langle |u|, |\Sigma|\rangle$-FST-ENCODE *is in* FPT.
4. $\langle |A|, |Q|\rangle$-FST-ENCODE *is in* FPT.
5. $\langle |A|, |u|\rangle$-FST-ENCODE *is* $W[1]$*-hard.*
6. $\langle |u|, |Q|_2\rangle$-FST-ENCODE *is* $W[2]$*-hard.*
7. $\langle |A|, |\Sigma|_3\rangle$-FST-ENCODE *is* $W[t]$*-hard for all* $t \geq 1$.
8. $\langle |Q|_2\rangle$-FST-ENCODE $\notin XP$ *unless* $P = NP$.
9. $\langle |\Sigma|_3\rangle$-FST-ENCODE $\notin XP$ *unless* $P = NP$.

Proof:

Proof of (1): Follows from the NP-hardness of BDFAI when $|Q| = 2$ as established in part (1) of Theorem 4.2.4 and the reduction in Lemma 4.3.1 from BDFAI to FST-ENCODE in which $|Q'| = |Q|$.

Proof of (2): Follows from the NP-hardness of BDFAI when $|\Sigma| = 2$ as established in part (2) of Theorem 4.2.4 and the reduction in Lemma 4.3.1 from BDFAI to FST-ENCODE in which $|\Sigma'| = |\Sigma| + 1$.

Proof of (3): This result follows from an algorithm based on that given in part (6) of Theorem 4.1.16. The algorithm given here is based on the following observation: there are $|\Sigma|^{|u|}$ possible strings of length $|u|$ over an alphabet Σ. Pick an arbitrary total order on Σ and use this order to establish a lexicographic ordering on the set of all possible strings of length $|u|$ such that each string in $\Sigma^{|u|}$ has a unique integer index in the range 1 to $|\Sigma|^{|u|}$. For instance, the binary form of such an index could be $|u||\Sigma|$ bits divided into $|u|$ zones of $|\Sigma|$ bits apiece corresponding to the values of individual symbols within a string. Let $ind(s)$ be the index so generated for string s, and $str(i)$ be the string corresponding to index i. Under the scheme described above, both of these functions can be computed in $O(|\Sigma||u|)$ time.

Given the above, consider the following algorithm:

1. Initialize an array pos of $|\Sigma|^{|u|}$ bits to 0, and then set the bit of pos whose index corresponds to u to 1.
2. For each FST a in A in the order specified by O do:
   (a) Initialize an array $buff$ of $|\Sigma|^{|u|}$ bits to 0.
   (b) For $i = 1$ to $|\Sigma|^{|u|}$ do:
      i. For $j = 1$ to $|\Sigma|^{|u|}$ do:
         A. If $pos(i) = 1$ and $str(i)/str(j)$ is accepted by a, set $buff(j) = 1$.
   (c) Copy $buff$ into pos.
3. If $pos(ind(s)) = 1$, the answer is "yes"; else, the answer is "no".

This algorithm essentially keeps track of all possible strings that can be generated from u by the composition of A as specified by O. If s is in this set of possible strings when all FST in A have been applied as specified by O, the answer is "yes"; else, the answer is "no". Steps (1), (2), and (3) can be done in time $O(|\Sigma|^{|u|}+|\Sigma||u|) = O(2|\Sigma|^{|u|}) = O(|\Sigma|^{|u|})$, $O((|\Sigma|^{|u|})^2|A|(|\Sigma|+1)|u|) = O(|\Sigma|^{2|u|}|A|2|\Sigma||u|) = O(|\Sigma|^{2|u|}|A||\Sigma||u|)$, and $O(|\Sigma||u|)$, respectively (note that each string-pair can be checked against FST a in $O(|u|)$ time in step 2(b)i.A because each FST in A is i/o-deterministic). Hence, the algorithm as a whole runs in $O(|\Sigma|^{2|u|}|A||\Sigma||u|)$ time, which is fixed-parameter tractable relative to $|\Sigma|$ and $|u|$.

Proof of (4): Construct the composition FST of all FST in A relative to O, and then intersect this composition FST with the pair of FST that associate u with all strings in $\Sigma^{|u|}$ and all strings in $\Sigma^{|u|}$ with s respectively (these FST are constructed by variants of the FST_{init} and FST_{final} constructions given in Lemma 4.3.1). As noted in Section 2.2.3, though the FST for u and s are i/o-deterministic, the composition FST is not necessarily

i/o-deterministic; hence, the final intersection FST is not necessarily i/o-deterministic. Note that the only possible string-pair in the relation of the final intersection FST is u/s and that u/s will be in this relation if and only if s is generated when the composition of the FST in A as specified by O is applied to u. Apply depth-first search to the transition diagram for the final intersection FST to determine if any of its final states are reachable from its start state, i.e., there is a string-pair that is accepted by the FST. If so, as the only possible member of the relation is u/s, the answer is "yes"; else, the answer is "no".

Consider the running time of the individual steps in this algorithm. By Table 2.3, the composition FST of A relative to O can be constructed in $O(|Q|^{2|A|}|\Sigma|^4|A|)$ time and will have at most $|Q|^{|A|}$ states and at most $|Q|^{2|A|}|\Sigma|^2$ transitions. By Table 2.3, the intersection FST can be constructed in

$$\begin{aligned} O((|Q|^{|A|}(|u|+1))^2|\Sigma|^4 + (|Q|^{|A|}(|u|+1)(|s|+1))^2|\Sigma|^4) \\ &= O(|Q|^{2|A|}(|u|+1)^2(|s|+1)^2|\Sigma|^4) \\ &= O(|Q|^{2|A|}(|u|+1)^4|\Sigma|^4) \\ &= O(|Q|^{2|A|}(2|u|)^4|\Sigma|^4) \\ &= O(|Q|^{2|A|}|u|^4|\Sigma|^4) \end{aligned}$$

time. As the graph $G = (V, A)$ associated with the transition diagram of the intersection FST has $|V| = |Q|^{|A|}(|u|+1)(|s|+1) = |Q|^{|A|}(|u|+1)^2 \leq |Q|^{|A|}(2|u|)^2 = 4|Q|^{|A|}|u|^2$ states and $|A| = (|Q|^{|A|}(|u|+1)(|s|+1))^2|\Sigma|^2 = |Q|^{2|A|}(|u|+1)^4|\Sigma|^2 \leq |Q|^{2|A|}(2|u|)^4|\Sigma|^2 = 16|Q|^{2|A|}|u|^4|\Sigma|^2$ arcs and depth-first search runs in $O(|V|+|A|)$ time, the final step runs in $O(|Q|^{2|A|}|u|^4|\Sigma|^2)$ time. Hence, the algorithm as a whole runs in $O(|Q|^{2|A|}|u|^4|\Sigma|^4|A|)$ time, which is fixed-parameter tractable relative to $|A|$ and $|Q|$.

Proofs of (5 – 7): Follow from W-hardness results for BDFAI established in parts (5 – 7) of Theorem 4.2.4, the reduction in Lemma 4.3.1 from BDFAI to FST-ENCODE in which $|A'| = |A| + 2$, $|u'| = k$, $|Q'| = |Q|$, and $|\Sigma'| = |\Sigma| + 1$, and Lemma 2.1.25.

Proofs of (8) and (9): Follow from (1) and (2) and Lemma 2.1.35. ∎

Theorem 4.3.4

1. FST-DECODE *is NP-hard when* $|Q| = 2$.
2. FST-DECODE *is NP-hard when* $|\Sigma| = 3$.
3. $\langle |s|, |\Sigma| \rangle$-FST-DECODE *is in FPT.*
4. $\langle |A|, |Q| \rangle$-FST-DECODE *is in FPT.*
5. $\langle |A|, |s| \rangle$-FST-DECODE *is* $W[1]$*-hard.*
6. $\langle |s|, |Q|_2 \rangle$-FST-DECODE *is* $W[2]$*-hard.*
7. $\langle |A|, |\Sigma|_3 \rangle$-FST-DECODE *is* $W[t]$*-hard for all* $t \geq 1$.

8. $\langle |Q|_2 \rangle$-FST-DECODE $\notin XP$ *unless* $P = NP$.

9. $\langle |\Sigma|_3 \rangle$-FST-DECODE $\notin XP$ *unless* $P = NP$.

Proof:

Proof of (1): Follows from the NP-hardness of FST-ENCODE when $|Q| = 2$ as established in part (1) of Theorem 4.3.3 and the reduction in Lemma 4.3.2 from FST-ENCODE to FST-DECODE.

Proof of (2): Follows from the NP-hardness of FST-ENCODE when $|\Sigma| = 3$ as established in part (2) of Theorem 4.3.3 and the reduction in Lemma 4.3.2 from FST-ENCODE to FST-DECODE.

Proof of (3): Modify step (1) in the algorithm given in part (3) of Theorem 4.3.3 such that each possible string of length $|s|$ is tested against the lexicon DFA $DFA(D)$, and only those strings that are members of D^+ have their corresponding bits in *pos* set to 1. As each such string can be tested against $DFA(D)$ in $O(|s|)$ time, the algorithm as a whole runs in $O(|\Sigma|^{2|s|}|A||\Sigma||s|)$ time, which is fixed-parameter tractable relative to $|\Sigma|$ and $|s|$.

Proof of (4): Given the lexicon DFA $DFA(D) = \langle Q_D, \Sigma, \delta_D, s_D, F_D \rangle$, create the lexicon FST $FST(D) = \langle Q_D, \Sigma, \Sigma, \delta_F, s_D, F_D \rangle$ such that $\delta_F = \{(q, x, y, q') \mid (q, x, q') \in \delta_D$ and $y \in \Sigma\}$. Note that $FST(D)$ essentially pairs every lexical string in D^+ with every possible string over Σ of the same length, i.e., the relation associated with $FST(D)$ is the set $\{x/y \mid x \in D^+$ and $y \in \Sigma^{|x|}\}$. Perform the algorithm described in part (4) of Theorem 4.3.3, substituting $FST(D)$ for the FST constructed from u. Note that the relation associated with the final intersection FST is the set $\{x/s \mid x \in D^+$ and s is generated by applying the composition of A as specified by O to $x\}$. Hence, if any of the final states in the final intersection FST are reachable from that FST's start state by depth-first search, the answer is "yes"; else, the answer is "no".

Consider the running time of the individual steps in this algorithm. The running time for the creation of the composition FST is the same as in part (4) of Theorem 4.3.3 under the appropriate substitution of $|Q_D|$ for $|u|$. By Table 2.3, the intersection FST can be constructed in

$$\begin{aligned} &O((|Q|^{|A|}(|s|+1))^2|\Sigma|^4 + (|Q|^{|A|}|Q_D|(|s|+1))^2|\Sigma|^4) \\ &\quad = O(|Q|^{2|A|}|Q_D|^2(|s|+1)^2|\Sigma|^4) \\ &\quad = O(|Q|^{2|A|}|Q_D|^2(2|s|)^4|\Sigma|^4) \\ &\quad = O(|Q|^{2|A|}|Q_D|^2|s|^2|\Sigma|^4) \end{aligned}$$

time. As the graph $G = (V, A)$ associated with the transition diagram of the intersection FST has $|V| = |Q|^{|A|}|Q_D|(|s|+1) \leq 2|Q|^{|A|}|Q_D||s|$ states and $|A| = (|Q|^{|A|}|Q_D|(|s|+1))^2|\Sigma|^2 = |Q|^{2|A|}|Q_D|^2(|s|+1)^2|\Sigma|^2 \leq |Q|^{2|A|}|Q_D|^2(2|s|)^2|\Sigma|^2 = 4|Q|^{2|A|}|Q_D|^2|s|^2|\Sigma|^2$ arcs and depth-first search runs in $O(|V| + |A|)$ time, the final step runs in

$O(|Q|^{2|A|}|Q_D|^2|s|^2|\Sigma|^2)$ time. Hence, the algorithm as a whole runs in $O(|Q|^{2|A|}|Q_D|^2 |s|^2|\Sigma|^4|A|)$ time, which is fixed-parameter tractable relative to $|A|$ and $|Q|$.

Proofs of (5 – 7): Follow from the W-hardness results for FST-ENCODE established in parts (5 – 7) of Theorem 4.3.3, the reduction in Lemma 4.3.2 from FST-ENCODE to FST-DECODE, and Lemma 2.1.25.

Proofs of (8) and (9): Follow from (1) and (2) and Lemma 2.1.35. ∎

4.3.3 Implications

All parameterized complexity results for problems FST-ENCODE and FST-DECODE that are either stated or implicit in the lemmas and theorems given in the previous section are shown in Tables 4.6 and 4.7. Consider the implications of these results for each problem in turn:

- FST-ENCODE: The sources of polynomial-time intractability are $\{|A|, |Q|\}$ and $\{|u|, |\Sigma|\}$. The mechanisms associated with these sources are the FST set (specifically, the composition FST of the FST in A) and the set of all possible surface forms for the given lexical form. All of the aspects in these sources are defining properties of FST-based rule systems, and hence none of them can be eliminated to reduce the complexity of the problem.

- FST-DECODE: The sources of polynomial-time intractability are $\{|A|, |Q|\}$ and $\{|s|, |\Sigma|\}$. Analogously with FST-ENCODE above, the mechanisms associated with these sources are the FST set and the set of possible lexical forms for a given surface form. All of the aspects in these sources are defining properties of FST-based rule systems, and hence none of them can be eliminated to reduce the complexity of the problem.

  In showing that the lexicon (when given as a FSA) can be interpreted as just another FST, the construction given in part (4) of Theorem 4.3.4 performs an unintentionally valuable service by dispelling the notion perhaps inadvertently fostered in Section 4.1.3 that the lexicon is a separate entity of an altogether different type than either rules or constraints in phonological processing. This allows the use of techniques for characterizing the relations associated with rule FST to be used to characterize the lexicon. Given the important role of the lexicon in the NP-hardness of SSG-DECODE, it would be interesting to characterize the role of the lexicon in the computational complexity of FST-based rule systems (possibly in terms of logic, as discussed in Section 4.2).

One very curious aspect of the NP-hardness of FST-ENCODE and FST-DECODE is that it does not depend at all on the makeup of the given lexical or surface form — indeed, in the reductions given in Lemmas 4.3.1 and 4.3.2, the given form is just a string over a one-symbol alphabet whose role is to specify the length of the requested lexical form (in the case of

FST-DECODE) or the intermediate forms in the derivation (in the case of FST-ENCODE). This suggests that the NP-hardness of problems associated with FST-based rule systems may arise in part from these systems having access to intermediate forms that are "hidden", in the sense that these intermediate forms need not depend on and can be manipulated independently of any given forms. This tallies with various intuitions concerning the role of intermediate forms in various computational difficulties that are often encountered when implementing FST-based rule systems [Spr92, pp. 138-139]. However, as will be shown by results derived in subsequent sections and discussed in Section 4.7, the role of such hidden structure in phonological processing is more subtle than it first appears.

Consider the versions of FST-ENCODE and FST-DECODE which allow FST in A that can add or delete symbols during string-transformations. As FST-ENCODE and FST-DECODE are special cases of these more general problems, all NP- and W-hardness results derived above still hold for these new problems. Having insertions and deletions does allow certain hardness results to hold in more restricted cases; for instance, by modifying the FST FST_{init} and FST_{final} in Lemma 4.3.1 appropriately to map the single-symbol string Δ onto all strings in Σ^k and all strings in Σ^k onto Δ, respectively, it is possible to rephrase all hardness results above such that the size of the given form alphabet and the length of the given form are both 1. It seems inevitable that allowing insertions and deletions will also both allow certain W-hardness results to hold relative to higher levels of the W hierarchy and allow parameterized problems that were formerly known to have FPT algorithms to be shown W-hard. The full extent of these changes will not be addressed here. For now, simply observe that the FPT algorithms based on FST composition will still work as given and the FPT algorithms based on brute-force enumeration of all possible requested forms will work if the maximum number of symbols that can be inserted or deleted is also a aspect in the parameter (this is so because the number of possible versions of a form f over an alphabet Σ into which k symbols are inserted or from which k symbols are deleted is bounded by $(|\Sigma|^{|f|} \sum_{i=1}^{k} \begin{pmatrix} |f|+i \\ i \end{pmatrix} |\Sigma|^i) + (|\Sigma|^{|f|} \sum_{i=1}^{k} \begin{pmatrix} |f| \\ i \end{pmatrix}) \leq (|\Sigma|^{|f|} k(|f|+k)^k) + (|\Sigma|^{|f|} k|f|^k) \leq 2|\Sigma|^{|f|} k(|f|+k)^k$, which is a function of $|\Sigma|$, $|f|$, and k).

Consider now what these results have to say about the various search problems associated with FST-based rule systems. First, note the following relationships between FST-ENCODE and FST-DECODE and their associated search problems:

- Any instance of FST-ENCODE created by the reduction in Lemma 4.3.1 can be solved by a single call to $\text{ENC}_{\mathbf{FST}}$ (this is because there is only one possible surface form s that the FST in these instances can associate with u); moreover, any instance of $\text{ENC}_{\mathbf{FST}}$ can be solved by the FPT algorithms given in the previous section for FST-ENCODE.
- Any instance of FST-DECODE can be solved by a single call to $\text{DEC}_{\mathbf{FST}}$; moreover, any instance of $\text{DEC}_{\mathbf{FST}}$ can be solved by the FPT algorithms given in the previous section for FST-DECODE.

- Any instance of FST-ENCODE can be solved by a single call to $\text{CHK}_{\textbf{FST}}$; moreover, any instance of $\text{CHK}_{\textbf{FST}}$ can be solved by the FPT algorithms given in the previous section for FST-ENCODE.

Hence, modulo various conjectures, $\text{ENC}_{\textbf{FST}}$, $\text{DEC}_{\textbf{FST}}$, and $\text{CHK}_{\textbf{FST}}$ do not have polynomial-time algorithms and have the same sources of polynomial-time intractability as their corresponding decision problems. Note that, unlike simplified segmental grammars, checking is polynomial-time intractable for FST-based rule systems. As will be discussed in Section 4.7, the reasons for this become clearer in light of the remaining results derived in this chapter.

The results derived above are relevant to implementations of FST-based rule systems to the extent that they show some of the conditions under which the encoding and decoding problems associated with these systems are NP-hard and suggest (albeit relative to a small set of aspects) what the sources of polynomial-time intractability in these systems may and may not be. These results are also relevant within the investigation proposed in Section 4.1.3 of the sources of polynomial-time intractability in Kaplan and Kay's FST calculus, in that they comprise the first systematic parameterized complexity analysis of the ϵ-free FST composition problem. In addition to these immediate applications, these results also have implications for the computational complexity of phonological processing in general, as reductions similar to those given from BDFAI also hold relative to the constraint-based phonological theories examined in the remainder of this chapter.

The above suggests many directions for future research. Several of the more intriguing directions are:

1. Formalize the notion of context-size for FST and re-do the analyses given here to take this new aspect into account.
2. Characterize the effect of the lexicon on the computational complexity of FST-DECODE.
3. Characterize the computational complexity of FST-ENCODE and FST-DECODE in terms of trade-offs between various aspects of the individual rule FST, e.g., type of (non)determinism, structure of the relation encoded by the FST.

The research in (1) is actually fairly important in light of the role context-size seems to play in the computational complexity of the automaton-based formulations of Declarative Phonology and Optimality Theory examined later in this chapter (see discussion in Section 4.5.3). One approach to the research (3) might be to formulate the relations implicit in FST in logic along the lines suggested in Section 4.2.

Table 4.6: The Parameterized Complexity of the FST-ENCODE Problem.

Parameter	Alphabet Size $\|\Sigma\|$	
	Unbounded	Parameter
–	NP-hard	$\notin XP$
$\|A\|$	$W[t]$-hard	$W[t]$-hard
$\|u\|$	$W[2]$-hard	FPT
$\|Q\|$	$\notin XP$	???
$\|A\|$, $\|u\|$	$W[1]$-hard	FPT
$\|A\|$, $\|Q\|$	FPT	FPT
$\|u\|$, $\|Q\|$	$W[2]$-hard	FPT
$\|A\|$, $\|u\|$, $\|Q\|$,	FPT	FPT

Table 4.7: The Parameterized Complexity of the FST-DECODE Problem.

Parameter	Alphabet Size $\|\Sigma\|$	
	Unbounded	Parameter
–	NP-hard	$\notin XP$
$\|A\|$	$W[t]$-hard	$W[t]$-hard
$\|s\|$	$W[2]$-hard	FPT
$\|Q\|$	$\notin XP$	???
$\|A\|$, $\|s\|$	$W[1]$-hard	FPT
$\|A\|$, $\|Q\|$	FPT	FPT
$\|s\|$, $\|Q\|$	$W[2]$-hard	FPT
$\|A\|$, $\|s\|$, $\|Q\|$,	FPT	FPT

4.4 The KIMMO System

4.4.1 Background

In response to various theoretical and practical problems with FST-based rule systems that operate by FST composition [KK94], Koskenniemi and Karttunen ([Kar83, Kos83]; see also [Kar93]) proposed the theory of Two-Level Morphology and its implementation in the KIMMO system. In this approach, there are only two levels of representation, lexical and surface, and rules are encoded as FST operating in parallel which directly transform lexical into surface forms and vice versa. This approach is surprisingly versatile; indeed, the various software incarnations of the KIMMO system, e.g., PC-KIMMO [Ant90], are the most successful and widely-used tools for simultaneously describing and implementing the morphophonology of human languages [Spr92, p. 153]. As FST can be interpreted both as rewriting rules transforming either lexical into surface or surface into lexical forms and as constraints on the distribution of lexical / surface symbol-pairs in the lexical / surface form string-pair, KIMMO can be seen as both a rule-based and a constraint-based phonological processing system.

A KIMMO grammar $g = \langle A, D, \Sigma_u, \Sigma_s \rangle$ consists of the following:

1. A lexicon $D \subseteq \Sigma_u^+$ for some lexical alphabet Σ_u. This lexicon is specified as a nondeterministic FSA which recognizes some subset of D^+ that corresponds to the set of valid lexical forms (see [BBR87, Spr92] for details).

2. Two tapes, lexical and surface, such that the surface tape contains strings $s \in \Sigma_s^+$ for some surface alphabet Σ_s and the lexical tape contains strings u corresponding to valid lexical forms in which the individual elements of D are separated by +–signs, e.g., *denationalize* = *de* + *nation* + *al* + *ize*.

3. A set A of i/o-deterministic FST such that each FST has the lexical tape as its input tape and the surface tape as its output tape. The input and output alphabets of each FST may be augmented with a special symbol **0** (*null*).

The KIMMO system consists of four levels: the lexical form, the lexical tape, the surface tape, and the surface form (see Figure 4.6). The lexical (surface) tape consists of a copy of the lexical (surface) form into which zero or more nulls have been inserted. The lexical and surface forms can be different lengths, but the lexical and surface tapes must be the same length. Given lexical-surface form string-pair u/s, the KIMMO system accepts u/s if there is some string-pair u'/s' such that u' (s') is a copy of u (s) into which zero or more nulls have been inserted, $|u'| = |s'|$, and the string-pair u'/s' is accepted by every FST in A, i.e., u'/s' is in the regular relation associated with the intersection FST for the FST in A. KIMMO can also be used to reconstruct strings (in the sense of Definition 2.2.20 in Section 2.2.3) such that the system is given either a lexical or a surface form and then uses the constraints on valid symbol-pairs encoded in the FST and the lexicon to reconstruct the contents of the omitted form. It is important to remember that though KIMMO is often described

SURFACE FORM:				*moved*			
SURFACE TAPE:	m	o	v	**0**	**0**	e	d
LEXICAL TAPE:	m	o	v	e	+	e	d
LEXICAL FORM:		*move*				*+ed*	

Figure 4.6: The KIMMO System (adapted from [Rit92, Figure 1]).

purely in terms of FST, it was originally defined relative to rewriting rules that operate on symbol-pairs (see [Kar93, Rit92, Spr92] for details). Though these rules specify KIMMO systems that have a lower generative capacity than those based on ϵ-free FST [Rit92], such rules are still compiled as FST and the KIMMO systems so produced function as described above.

The outer two null-manipulating levels of the KIMMO system may seem redundant, but they (or something like them) are required by any finite-state system based on FST intersection that also allows insertions and deletions. This is because FST that allow ϵ-transitions (and hence directly encode insertions and deletions) are not closed under intersection, as shown by the following example attributed to Kaplan [BBR87, page 158]: Given two FST A and B whose regular relations are described by symbol-pair regular expressions $(a : b)^*(\epsilon : c)^*$ and $(\epsilon : b)^*(a : c)^*$, their intersection is the string-relation described by the expression (a^n/b^nc^n) which cannot be recognized by a FST. To get around this problem, KIMMO uses an internal representation of form strings in which nulls explicitly mark the occurrence of ϵ-transitions during the processing of a given lexical-surface form string-pair. The outer two levels of the system maintain this internal representation by the appropriate insertion and deletion of nulls, and the intersection of *i/o*-deterministic FST (which have same-length regular relations and are thus closed under intersection
[KK94, Section 3.3]) is then applied to this internal representation to do the required linguistic processing. As the outer two null-manipulating levels can be implemented by FST and FST are closed under composition, the KIMMO system is finite-state [KK94, Section 7].

The first complexity analysis of the KIMMO system was done in [Bar86] and is given in full in Chapters 5 and 6 of [BBR87]. In these analyses, the KIMMO system was restricted such that no insertions or deletions were allowed, i.e., the constraint FST encode same-length relations, and +-symbols were not required to separate lexical elements on the lexical tape. These analyses focused on the following problems:

KIM-ENCODE
Instance: A KIMMO grammar $g = \langle A, D, \Sigma_u, \Sigma_s \rangle$ and a string $u \in \Sigma_u^+$.
Question: Is there a string $s \in \Sigma_s^{|u|}$ such that s is generated when g is applied to u?

KIM-DECODE
Instance: A KIMMO grammar $g = \langle A, D, \Sigma_u, \Sigma_s \rangle$ and a string $s \in \Sigma_s^+$.
Question: Is there a string $u \in \Sigma_u^{|s|}$ such that $u \in D^+$ and the result of applying g to u is s?

The problems were shown to be NP-complete in sections 5.5.1 and 5.5.2 of [BBR87], respectively. Note that in the versions of these problems examined in this thesis, the lexicon D is specified not as a nondeterministic FSA that recognizes some subset of D^+ but rather as a DFA $DFA(D) = \langle Q_D, \Sigma_u, \delta_D, s_D, F_D \rangle$ on $|Q_D|$ states which recognizes D^+. This was done both to make results derived in this section compatible with those derived for the other theories examined in this thesis and to make more obvious (by virtue of using a simpler model) the role played by the lexicon in the computational complexity of the KIMMO system.

The complexity-theoretic analyses of the problems defined above were originally motivated by the claim that as KIMMO uses FST which are by nature efficient, KIMMO is also efficient (a motivation that [BBR87] characterized as "The Lure of the Finite State"). While this is true if both lexical and surface strings are given as inputs and there are no insertions or deletions (indeed, in this case, KIMMO runs in linear time), the proofs cited above show that if only one string is given then the computations required to recreate the other input are, in general, NP-hard. These results have been criticized as unnatural and irrelevant because the reductions used to show NP-hardness require unbounded numbers of transducers [KC88] as well as input strings of unbounded length and alphabets of unbounded size [Rou87] and hence are not characteristic of the bounded representations and computations underlying human natural language processing. Implicit in these criticisms is the assumption that the NP-hardness of the encoding and decoding problems associated with the KIMMO system is purely a function of these aspects, and if the values of these aspects are bounded then the KIMMO system operates efficiently. These claims will be evaluated in the systematic parameterized analysis given in next section.

4.4.2 Analysis

The systematic parameterized analysis in this section will focus on the following aspects:

- The number of FST in A ($|A|$).
- The length of the given lexical / surface form ($|u|$ / $|s|$).
- The maximum number of states in any FST in A ($|Q|$).
- The sizes of the lexical and surface alphabets ($|\Sigma_u|$, $|\Sigma_s|$).

The lexical and surface alphabet sizes are considered as separate aspects to give independent characterizations of the possible composition of and encodings within the forms in the proofs given below.

Consider the following reductions, which are loosely based on the NP-hardness reductions for KIM-ENCODE and KIM-DECODE given in [BBR87].

Lemma 4.4.1 *BDFAI* $\leq_m$ KIM-ENCODE.

Proof: Given an instance $\langle A, \Sigma, k \rangle$ of BDFAI, construct the following instance $\langle A', D', \Sigma'_u, \Sigma'_s, u' \rangle$ of KIM-ENCODE: Let $\Sigma'_u = \{\Delta\}$ for some symbol $\Delta \notin \Sigma$, $\Sigma'_s = \Sigma$, $u' = \Delta^k$, and $D' = \{u'\}$. Given a DFA $a = \langle Q, \Sigma, \delta, s, F \rangle$, let $FST_u(a) = \langle Q, \Sigma'_u, \Sigma, \delta_F, s, F \rangle$ be the FST such that $\delta_F = \{(q, \Delta, x, q') \mid (q, x, q') \in \delta\}$. Note that $FST_u(a)$ essentially pairs Δ^n, $n \geq 0$, with every string of length n that is accepted by a, i.e., the relation associated with $FST_u(a)$ is the set $\{x/y \mid |x| = \Delta^{|y|}$ and y is accepted by $a\}$. Let A' be the set consisting of all FST $FST_u(a)$ corresponding to DFA $a \in A$. This construction can be done in time polynomial in the size of the given instance of BDFAI.

To see that this construction is a many-one reduction, note that any solution to the constructed instance of KIM-ENCODE is a string $s' \in {\Sigma'}_s^{|u'|}$ such that u'/s' is accepted by each FST in A', which corresponds to a string in Σ^k that is accepted by each DFA in A. Moreover each solution to the given instance of BDFAI is a string $s \in \Sigma^k$ that is accepted by every DFA in A, which corresponds to a string $s' \in {\Sigma'}_s^{|u'|}$ such that u'/s' is accepted by every FST in A'. Thus, the given instance of BDFAI has a solution if and only if the constructed instance of KIM-ENCODE has a solution.

Note that in the constructed instance of KIM-ENCODE, $|A'| = |A|$, $|u'| = k$, $|Q'| = |Q|$, $|\Sigma'_u| = 1$, and $|\Sigma'_s| = |\Sigma|$. ∎

Lemma 4.4.2 *BDFAI* $\leq_m$ KIM-DECODE.

Proof: Given an instance $\langle A, \Sigma, k \rangle$ of BDFAI, construct the following instance $\langle A', D', \Sigma'_u, \Sigma'_s, s' \rangle$ of KIM-DECODE: Let $\Sigma'_u = \{\Sigma\}$, $\Sigma'_s = \{\Delta\}$ for some symbol $\Delta \notin \Sigma$, $s' = \Delta^k$, and $D' = \{x \mid x \in \Sigma\}$. Note that lexical strings of length $|s'|$ are generated by the interaction of the implicit concatenation of elements in D' to form D'^+ and the FST in A' which, by virtue of not encoding symbol insertions or deletions, can only accept lexical strings of length $|s'|$. Given a DFA $a = \langle Q, \Sigma, \delta, s, F \rangle$, let $FST_s(a) = \langle Q, \Sigma, \Sigma'_s, \delta_F, s, F \rangle$ be the FST such that $\delta_F = \{(q, x, \Delta, q') \mid (q, x, q') \in \delta\}$. Note that $FST_s(a)$ essentially pairs every string of length n, $n \geq 0$, that is accepted by a with Δ^n, i.e., the relation associated with $FST_s(a)$ is the set $\{x/y \mid |y| = \Delta^{|x|}$ and x is accepted by $a\}$. Let A' be the set consisting of all FST $FST_s(a)$ corresponding to DFA $a \in A$. This construction can be done in time polynomial in the size of the given instance of BDFAI.

To see that this construction is a many-one reduction, note that any solution to the constructed instance of KIM-DECODE is a string $u' \in D'^{|s'|}$ such that u'/s' is accepted by each FST in A', which corresponds to a string in Σ^k that is accepted by each DFA in A. Moreover each solution to the given instance of BDFAI is a string $u \in \Sigma^k$ that is accepted by every DFA in A, which corresponds to a string $u' \in D'^{|s'|}$ such that u'/s' is accepted by every FST in A'. Thus, the given instance of BDFAI has a solution if and only if the constructed instance of KIM-DECODE has a solution.

Note that in the constructed instance of KIM-DECODE, $|A'| = |A|$, $|s'| = k$, $|Q'| = |Q|$, $|\Sigma'_u| = |\Sigma|$, and $|\Sigma'_s| = 1$. ∎

Theorem 4.4.3

1. KIM-ENCODE *is* NP*-hard when* $|Q| = 2$ *and* $|\Sigma_u| = 1$.
2. KIM-ENCODE *is* NP*-hard when* $|\Sigma_s| = 2$ *and* $|\Sigma_u| = 1$.
3. KIM-ENCODE *is* NP*-hard when* $|Q| = 4$ *and* $|\Sigma_s| = 3$.
4. $\langle |u|, |\Sigma_s| \rangle$-KIM-ENCODE *is in* FPT.
5. $\langle |A|, |Q| \rangle$-KIM-ENCODE *is in* FPT.
6. $\langle |A|, |u|, |\Sigma_u|_1 \rangle$-KIM-ENCODE *is* $W[1]$*-hard.*
7. $\langle |u|, |Q|_2, |\Sigma_u|_1 \rangle$-KIM-ENCODE *is* $W[2]$*-hard.*
8. $\langle |A|, |\Sigma_u|_1, |\Sigma_s|_2 \rangle$-KIM-ENCODE *is* $W[t]$*-hard for all* $t \geq 1$.
9. $\langle |Q|_2, |\Sigma_u|_1 \rangle$-KIM-ENCODE $\not\in XP$ *unless* $P = NP$.
10. $\langle |\Sigma_s|_2, |\Sigma_u|_1 \rangle$-KIM-ENCODE $\not\in XP$ *unless* $P = NP$.
11. $\langle |Q|_4, |\Sigma_s|_3 \rangle$-KIM-ENCODE $\not\in XP$ *unless* $P = NP$.

Proof:

Proof of (1): Follows from the NP-hardness of BDFAI when $|Q| = 2$ as established in part (1) of Theorem 4.2.4 and the reduction in Lemma 4.4.1 from BDFAI to KIM-ENCODE in which $|Q'| = |Q|$ and $|\Sigma'_u| = 1$.

Proof of (2): Follows from the NP-hardness of BDFAI when $|\Sigma| = 2$ as established in part (2) of Theorem 4.2.4 and the reduction in Lemma 4.4.1 from BDFAI to KIM-ENCODE in which $|\Sigma'_s| = |\Sigma|$ and $|\Sigma'_u| = 1$.

Proof of (3): Follows from the reduction in Section 5.5.1 of [BBR87] which proves that KIM-ENCODE is NP-hard when $|Q| = 4$ and $|\Sigma_s| = 3$.

Proof of (4): Follows from the algorithm that generates all $|\Sigma_s|^{|u|}$ possible k-length strings over alphabet Σ_s and checks each such string s in $O(|A||u|)$ time to see whether u/s is accepted by each of the FST in A (this $O(|u|)$ running time per FST is possible because the FST in A are i/o-deterministic). The algorithm as a whole runs in $O(|\Sigma_s|^{|u|}|A||u|)$ time, which is fixed-parameter tractable relative to $|u|$ and $|\Sigma_s|$.

Proof of (5): Construct the intersection FST of all FST in A and the $|u| + 1$-state FST that associates u with all strings in $\Sigma_s^{|u|}$ (this FST is constructed by a variant of the FST_{init} construction given in Lemma 4.3.1). As noted in Section 2.2.3, if each of the component FST in this intersection is i/o-deterministic then their intersection FST is i/o-deterministic. Note that the relation associated with this intersection FST is the set $\{u/y \mid y \in \Sigma_s^{|u|}$ and u/y is accepted by every FST in $A\}$. Apply depth-first search to the transition diagram for this intersection FST to determine if any of its final states are reachable from its start state, i.e., there is a string-pair that is accepted by the FST. If so, the answer is "yes"; else, the answer is "no".

Consider the running times of the individual steps in this algorithm. By Table 2.3, the intersection FST can be constructed in $O(|Q|^{|A|+1}|u|(|\Sigma_u||\Sigma_s|)^2)$ time. As the graph $G = (V, A)$ associated with this transition diagram has $|V| = |Q|^{|A|}(|u|+1) \leq 2|Q|^{|A|}|u|$ states and $|A| = |Q|^{|A|}(|u|+1)|\Sigma_u||\Sigma_s| \leq 2|Q|^{|A|}|u||\Sigma_u||\Sigma_s|$ arcs and depth-first search runs in $O(|V| + |A|)$ time, the final step runs in $O(|Q|^{|A|}|u||\Sigma_u||\Sigma_s|)$ time. Thus, the algorithm as a whole runs in $O(|Q|^{|A|+1}|u|(|\Sigma_u||\Sigma_s|)^2)$ time, which is fixed-parameter tractable relative to $|A|$ and $|Q|$.

Proofs of (6 – 8): Follow from the W-hardness results for BDFAI established in parts (5 – 7) of Theorem 4.2.4, the reduction in Lemma 4.4.1 from BDFAI to KIM-ENCODE in which $|A'| = |A|$, $|u| = k$, $|Q'| = |Q|$, $|\Sigma_u| = 1$, and $|\Sigma_s| = |\Sigma|$, and Lemma 2.1.25.

Proofs of (9 – 11): Follow from (1 – 3) and Lemma 2.1.35. ∎

Theorem 4.4.4

1. KIM-DECODE *is NP-hard when* $|Q| = 2$ *and* $|\Sigma_s| = 1$.
2. KIM-DECODE *is NP-hard when* $|\Sigma_u| = 2$ *and* $|\Sigma_s| = 1$.
3. KIM-DECODE *is NP-hard when* $|Q| = 4$ *and* $|\Sigma_s| = 1$.
4. $\langle |s|, |\Sigma_u| \rangle$-KIM-DECODE *is in FPT.*
5. $\langle |A|, |Q| \rangle$-KIM-DECODE *is in FPT.*
6. $\langle |A|, |s|, |\Sigma_s|_1 \rangle$-KIM-DECODE *is* $W[1]$*-hard.*
7. $\langle |s|, |Q|_2, |\Sigma_s|_1 \rangle$-KIM-DECODE *is* $W[2]$*-hard.*
8. $\langle |A|, |\Sigma_s|_1, |\Sigma_u|_2 \rangle$-KIM-DECODE *is* $W[t]$*-hard for all* $t \geq 1$.
9. $\langle |Q|_2, |\Sigma_s|_1 \rangle$-KIM-DECODE $\notin XP$ *unless* $P = NP$.
10. $\langle |\Sigma_u|_2, |\Sigma_s|_1 \rangle$-KIM-DECODE $\notin XP$ *unless* $P = NP$.
11. $\langle |Q|_4, |\Sigma_u|_3 \rangle$-KIM-DECODE $\notin XP$ *unless* $P = NP$.

Proof:

Proof of (1): Follows from the NP-hardness of BDFAI when $|Q| = 2$ as established in part (1) of Theorem 4.2.4 and the reduction in Lemma 4.4.2 from BDFAI to KIM-DECODE in which $|Q'| = |Q|$ and $|\Sigma'_s| = 1$.

Proof of (2): Follows from the NP-hardness of BDFAI when $|\Sigma| = 2$ as established in part (2) of Theorem 4.2.4 and the reduction in Lemma 4.4.2 from BDFAI to KIM-DECODE in which $|\Sigma'_u| = |\Sigma|$ and $|\Sigma'_s| = 1$.

Proof of (3): Follows from the reduction in Section 5.5.2 of [BBR87] which proves that KIM-DECODE is NP-hard when $|Q| = 4$ and $|\Sigma_s| = 3$.

Proof of (4): Follows from the algorithm that generates all $|\Sigma_u|^{|s|}$ possible k-length strings over alphabet Σ_u and checks each such string u in $O((|A| + 1)|s|) = O(|A||s|)$ time to see whether u/s is accepted by each of the FST in A and u is accepted by $DFA(D)$. The algorithm as a whole runs in $O(|\Sigma_u|^{|s|}|A||s|)$ time, which is fixed-parameter tractable relative to $|\Sigma_u|$ and $|s|$.

Proof of (5): Given the lexicon DFA $DFA(D) = \langle Q_D, \Sigma_u, \delta_D, s_D, F_D\rangle$, create the lexicon FST $FST(D) = \langle Q_D, \Sigma_u, \Sigma_s, \delta_F, s_D, F_D\rangle$ such that $\delta_F = \{(q, x, y, q') \mid (q, x, q') \in \delta_D$ and $y \in \Sigma_s\}$. Note that $FST(D)$ essentially pairs every lexical string in D^+ with every possible surface string of the same length, i.e., the relation associated with $FST(D)$ is the set $\{x/y \mid x \in D^+$ and $y \in \Sigma_s^{|x|}\}$. Perform the algorithm described in part (5) of Theorem 4.4.3, adding $FST(D)$ to the set of FST to be intersected. Note that the relation associated with the intersection FST is $\{x/s \mid x \in D^+$ and x/s is accepted by each FST in $A\}$. Hence, if any of the final states in the intersection FST are reachable from that FST's start state by depth-first search, the answer is "yes"; else, the answer is "no".

Consider the running times of the individual steps in this algorithm. By Table 2.3, the intersection FST can be constructed in $O(|Q|^{|A|+1}|u||Q_D|(|\Sigma_u||\Sigma_s|)^2)$ time. As the graph $G = (V, A)$ associated with this transition diagram has $|V| = |Q|^{|A|}(|u|+1)|Q_D| \leq 2|Q|^{|A|}|u||Q_D|$ states and $|A| = |Q|^{|A|}(|u| + 1)|Q_D||\Sigma_u||\Sigma_s| \leq 2|Q|^{|A|}|u||Q_D||\Sigma_u||\Sigma_s|$ arcs and depth-first search runs in $O(|V| + |A|)$ time, the final step runs in $O(|Q|^{|A|}|u||Q_D||\Sigma_u||\Sigma_s|)$ time. Thus, the algorithm as a whole runs in $O(|Q|^{|A|+1}|u||Q_D|(|\Sigma_u||\Sigma_s|)^2)$ time, which is fixed-parameter tractable relative to $|A|$ and $|Q|$.

Proofs of (6 – 8): Follow from the W-hardness results for BDFAI established in parts (5 – 7) of Theorem 4.2.4, the reduction in Lemma 4.4.2 from BDFAI to KIM-DECODE in which $|A'| = |A|$, $|s| = k$, $|Q'| = |Q|$, $|\Sigma_s| = 1$, and $|\Sigma_u| = |\Sigma|$, and Lemma 2.1.25.

Proofs of (9 – 11): Follow from (1 – 3) and Lemma 2.1.35. ∎

4.4.3 Implications

All parameterized complexity results for problems KIM-ENCODE and KIM-DECODE that are either stated or implicit in the lemmas and theorems given in the previous section are shown in Tables 4.8 and 4.9. Consider the implications of these results for each problem in turn:

- KIM-ENCODE: The sources of polynomial-time intractability are $\{|A|, |Q|\}$ and $\{|u|, |\Sigma_s|\}$. The mechanisms associated with these sources are the FST set (specifically, the intersection FST of the FST in A) and the set of possible surface forms for a given lexical form. All of the aspects in these sources are defining properties of the KIMMO system, and hence none of them can be eliminated to reduce the complexity of the problem.
- KIM-DECODE: The sources of polynomial-time intractability are $\{|A|, |Q|\}$ and $\{|s|, |\Sigma_u|\}$. Analogously with KIM-ENCODE above, the mechanisms associated with these sources are the FST set and the set of possible lexical forms for a given surface form. All of the aspects in these sources are defining properties of the KIMMO system, and hence none of them can be eliminated to reduce the complexity of the problem.

  As is the case for FST-based rule systems that operate by FST composition, part (5) of Theorem 4.4.4 shows that the lexicon can be treated as just another rule FST in the KIMMO system. Further analyses of the role of the lexicon in the computational complexity of the KIMMO system may profit from techniques that characterize the relations encoded in FST in terms of logic (as discussed in Section 4.2). One such analysis of particular interest would be to establish the role played by the succinctness of the type of encoding used for the lexicon, e.g., nondeterministic or deterministic FSA.

As in the FST-based rule systems examined in Section 4.3, the NP-hardness of KIM-ENCODE and KIM-DECODE does not depend at all on the makeup of the given lexical or surface form — indeed, in the reductions given in Lemmas 4.4.1 and 4.4.2, the given form is just a string over a one-symbol alphabet whose role is to specify the length of the requested form. Once again, hidden forms seem to play a role in this NP-hardness; however, unlike FST-based rule systems based on FST composition that have many such hidden forms, the results above show that if rules can operate in parallel, one such hidden form suffices (but may not be necessary; see further discussion of this point in Section 4.7). The role of such structure that is hidden relative to the given form will be explored in more detail in the analyses of Declarative Phonology and Optimality Theory given later in this chapter.

An immediate consequence of the results derived above is that, contrary to statements in the literature [KC88, Rou87, Spr92], the NP-hardness of the encoding and decoding problems associated with the KIMMO system is not a consequence of either the size of the alphabet, the length of the given form, or the number of FST being individually unbounded. Rather, it is a function of groups of such aspects being unbounded simultaneously. The statements made in [KC88, Rou87] were derived by reasoning from the NP-hardness reductions for the KIMMO system that were given in [BBR87]. As such, they are good examples of the problems described in Chapter 3 that can arise in trying to establish sources of polynomial-time intractability by reasoning from reductions, and are worth discussing in more detail. The reductions in [BBR87] were from the following problem:

3-Satisfiability (3SAT)
Instance: A boolean formula F in conjunctive normal form such that each disjunction-clause has at most 3 variables or literals.
Question: Is there a truth-assignment to the variables of F that *satisfies* F, i.e., makes F evaluate to *True*?

The reductions constructed instances of KIM-Encode and KIM-Decode that encoded the formula F as the given lexical or surface form and the truth-assignments to that formula's variables in the requested surface or lexical form, respectively. The satisfiability of the truth assignment in the requested form is ensured by structuring A such that there is one FST for each variable which ensures that this variable has the same value throughout the requested form and one FST which makes sure that the whole truth-assignment satisfies F. Various authors [KC88, Spr92] have noted that these reductions require an unbounded number of FST of the first type which ensure long-range agreement within the requested form of the values taken by a particular variable in F, and concluded that such FST and their number were a source of polynomial-time intractability in KIMMO. A further conclusion was that as this type of FST is analogous to a vowel harmony process (see Examples 2.2.3 and 2.2.4 in Section 2.2.1) and no human language has more than three such processes [KC88], then the reductions were unrealistic and therefore the derived results were irrelevant.

As was noted in Chapter 3, this kind of reasoning is doubly flawed (once because instances produced by reductions need not be realistic and once more because aspects important to reductions are not necessarily sources of polynomial-time intractability). This has been borne out in the results derived in this section, e.g., the number of rule FST cannot be a source of polynomial-time intractability because both $\langle|A|\rangle$-KIM-Encode and $\langle|A|\rangle$-KIM-Decode are $W[t]$-hard. A result of particular interest in light of the discussion above is that harmony-like processes are not necessary for NP-hardness in KIMMO. This is shown by the reductions ultimately originating from Dominating set in part (7) of both Theorem 4.4.3 and 4.4.4, in which each created FST requires only that one of a particular set of symbols occur *somewhere* in the requested form (which is a distinctly un-harmonic process). Taken as a whole, the various reductions seen to date suggest that the encoding and decoding problems associated with the KIMMO system can be NP-hard relative to many types of rules. Adequately characterizing the types of rules that can contribute to this NP-hardness is yet another promising topic for future research.

As mentioned previously, additions and deletions in KIMMO involve considerations of null symbols in the FST intersection process. Problems KIM-Encode and KIM-Decode can be redefined as follows to handle additions and deletions:

KIM(N)-ENCODE
Instance: A KIMMO grammar $g = \langle A, D, \Sigma_u, \Sigma_s \rangle$ and a string $u \in \Sigma_u^+$.
Question: Is there a string $s \in (\Sigma_s \cup \{\mathbf{0}\})^+$ and a null-augmented version u' of u such that $|s| = |u'|$ and s is generated when g is applied to u'?

KIM(N)-DECODE
Instance: A KIMMO grammar $g = \langle A, D, \Sigma_u, \Sigma_s \rangle$ and a string $s \in \Sigma_s^*$.
Question: Is there a string u accepted by D, a null-augmented version u' of u, and a null-augmented version s' of s such that $|s'| = |u'|$ and the result of applying g to u' is s'?

As KIM-ENCODE and KIM-DECODE are special cases of KIM(N)-ENCODE and KIM(N)-DECODE, respectively, all NP- and W-hardness results derived above still hold for these new problems. Having insertions and deletions does allow certain hardness results to hold in more restricted cases; for instance, using the trick given in the reduction in [BBR87, Section 5.7.2] in which a given form in a reduction consists of two dummy terminator symbols and the FST in A are restructured to construct arbitrary requested forms over the nulls in the given form, it is possible to rephrase all hardness results above such that the size of the given form alphabet and the length of the given form are both 2. It seems inevitable that allowing insertions and deletions will also both allow certain hardness results to hold relative to higher levels of the W hierarchy and allow parameterized problems that were formerly known to have FPT algorithms to be shown W-hard. The full extent of these changes will not be addressed here. For now, simply observe that the FPT algorithms based on FST intersection will still work if each FST is modified to accept arbitrary numbers of lexical or surface nulls at any point in processing (this can be ensured by adding to each FST the sets of transitions $\{\delta(q, \mathbf{0}, x) = q \mid x \in \Sigma_s\}$ and $\{\delta(q, x, \mathbf{0}) = q \mid x \in \Sigma_u\}$ for every state $q \in Q$), and the FPT algorithms based on brute-force enumeration of all possible requested forms will work if the maximum number of nulls that can be added is also a aspect in the parameter (this is so because the number of possible null-augmented versions of a form f over an alphabet Σ that incorporate at most k nulls is $|\Sigma|^{|f|} \sum_{i=1}^{k} \left(\begin{array}{c} |f| + i \\ i \end{array} \right) \leq |\Sigma|^{|f|} k(|f| + k)^k$, which is a function of $|\Sigma|$, $|f|$, and k).

Consider now what these results have to say about the various search problems associated with the KIMMO system. First, note the following relationships between KIM-ENCODE and KIM-DECODE and their associated search problems:

- Any instance of KIM-ENCODE can be solved by a single call to $\text{ENC}_{\mathbf{KIM}}$; moreover, any instance of $\text{ENC}_{\mathbf{KIM}}$ can be solved by the FPT algorithms given in the previous section for KIM-ENCODE.
- Any instance of KIM-DECODE can be solved by a single call to $\text{DEC}_{\mathbf{KIM}}$; moreover, any instance of $\text{DEC}_{\mathbf{KIM}}$ can be solved by the FPT algorithms given in the previous section for KIM-DECODE.

Hence, modulo various conjectures, $\text{ENC}_{\mathbf{KIM}}$ and $\text{DEC}_{\mathbf{KIM}}$ do not have polynomial-time algorithms and have the same sources of polynomial-time intractability as their corresponding decision problems. The case of problem $\text{CHK}_{\mathbf{KIM}}$ is more interesting because it is solvable in polynomial time (just run the given string-pair u/s against all FST in A). The situation here, in which checking is easy but encoding is hard, is the opposite of that for simplified segmental grammars. It is tempting to suggest that it is a property of constraint-based phonological systems; however, as is shown in Section 4.5.3 and 4.6.3, it is not. This somewhat odd state of affairs will be discussed further in Section 4.7.

The results derived above are relevant to the KIMMO system to the extent that they both put previous speculation about the sources of polynomial-time intractability in this system on a firm formal footing and suggest (albeit relative to a small set of aspects) what these sources may and may not be. These results are also relevant within the investigation proposed in Section 4.1.3 of the sources of polynomial-time intractability in Kaplan and Kay's FST calculus, in that they comprise the first systematic parameterized complexity analysis of the ϵ-free FST intersection problem.

The above suggests many directions for future research. Several of the more intriguing directions are:

1. Formalize the notion of context-size for FST and re-do the analyses given here to take this new aspect into account.
2. Characterize the effect of the lexicon on the computational complexity of KIM-Decode.
3. Characterize the computational complexity of KIM-Encode and KIM-Decode in terms of trade-offs between various aspects of the individual rule FST, e.g., type of (non)determinism, structure of the relation encoded by the FST.
4. Perform a systematic parameterized complexity analysis of the KIMMO system relative to the original rewriting-rule formalism instead of FST.

The research in (1) is actually fairly important in light of the role context-size seems to play in the computational complexity of other constraint-based phonological theories like Declarative Phonology and Optimality Theory (see discussion in Section 4.5.3). One goal of (3) should be to see if there are classes of FST that are not ϵ-free but are closed under intersection and can encode the outer two null-manipulating layers of the KIMMO system. If so, the KIMMO system could finally become a true two-level system.

Table 4.8: The Parameterized Complexity of the KIM-ENCODE Problem.

Parameter	Alphabet Sizes ($\|\Sigma_u\|$,$\|\Sigma_s\|$)			
	(Unb,Unb)	(Unb,Prm)	(Prm,Unb)	(Prm,Prm)
–	NP-hard	$\notin XP$	$\notin XP$	$\notin XP$
$\|A\|$	$W[t]$-hard	$W[t]$-hard	$W[t]$-hard	$W[t]$-hard
$\|u\|$	$W[2]$-hard	FPT	$W[2]$-hard	FPT
$\|Q\|$	$\notin XP$	$\notin XP$	$\notin XP$	???
$\|A\|, \|u\|$	$W[1]$-hard	FPT	$W[1]$-hard	FPT
$\|A\|, \|Q\|$	FPT	FPT	FPT	FPT
$\|u\|, \|Q\|$	$W[2]$-hard	FPT	$W[2]$-hard	FPT
$\|A\|, \|u\|, \|Q\|$	FPT	FPT	FPT	FPT

Table 4.9: The Parameterized Complexity of the KIM-DECODE Problem.

Parameter	Alphabet Sizes ($\|\Sigma_u\|$,$\|\Sigma_s\|$)			
	(Unb,Unb)	(Unb,Prm)	(Prm,Unb)	(Prm,Prm)
–	NP-hard	$\notin XP$	$\notin XP$	$\notin XP$
$\|A\|$	$W[t]$-hard	$W[t]$-hard	$W[t]$-hard	$W[t]$-hard
$\|s\|$	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$\|Q\|$	$\notin XP$	$\notin XP$	$\notin XP$	???
$\|A\|, \|s\|$	$W[1]$-hard	$W[1]$-hard	FPT	FPT
$\|A\|, \|Q\|$	FPT	FPT	FPT	FPT
$\|s\|, \|Q\|$	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$\|A\|, \|s\|, \|Q\|$	FPT	FPT	FPT	FPT

4.5 Declarative Phonology

4.5.1 Background

Over the last thirty years, constraints have played larger and larger roles in phonological theories (see [LP93, Bir95] for a review). Though originally used in the *SPE* model to specify well-formedness of lexical forms, the role of constraints in subsequent theories has expanded to ensuring the well-formedness of intermediate, surface, and full forms. Some theories even incorporate explicit interaction between constraints and rules, e.g., in the Theory of Constraints and Repair Strategies (TCRS) [LP93], a violated constraint can trigger rules that repair the structure which caused the original violation. Such theories have had to invoke progressively more complex formal machinery to specify the interaction of constraints and rules.

Declarative Phonology avoids the need for such machinery by stipulating that *all* phonological representations and mechanisms — from traditional constraints to rewriting rules to lexicons and lexical, surface, and full forms — be formulated as constraints, each of which encodes some information about the structure of valid full forms. Full forms are created by the logical conjunction of the appropriate set of constraints. In this framework, there are no separate levels of representation or order in which mechanisms apply to representations — there are only full forms whose structure is as specific as the information encoded in their associated sets of constraints. The name Declarative Phonology derives from the declarative/procedural distinction in programming styles [Bir95, Section 1.4.1], and is appropriate because Declarative Phonology focuses on declarative specifications for defining well-formed phonological objects rather than procedural specifications for computing those objects, e.g., the SSG model in Section 4.1.

A Declarative Phonology system $S = \langle P, D \rangle$ is a pair of constraint-sets P and D encoding the phonological mechanisms and lexicon of a language, respectively. The only operation within such a system is logical constraint conjunction. By combining P and D with lexical and surface forms in the appropriate ways, constraint conjunction can account for all language operations of interest, e.g.,

- *Speech Generation*: The conjunction of a given lexical form and P yields a set of full forms that are consistent with P and the partial hidden and visible structure specified by the lexical form.
- *Speech Comprehension*: The conjunction of a given surface form, P, and D yields the set of full forms that are consistent with P, D, and the visible structure specified by the surface form.

Note that the goal of any constraint conjunction is a full form which is interpreted accordingly. In the case of speech generation, the visible component within the full form is pronounced; in the case of speech comprehension, the full form is processed in conjunction with the lexicon to identify the lexical elements underlying that form. In order to ensure that information can always be combined to produce a valid full form that satisfies all constraints in a given set, Declarative Phonology requires that constraint-systems be structured such that there is no constraint conflict[3], i.e., two constraints cannot both apply to the same unspecified portion of a form and require different specifications for that portion, and also that the order in which constraints are conjuncted does not matter, i.e., no constraint can add structure that is required for the application of another constraint.

Though there have been software implementations of Declarative Phonology for particular tasks, e.g., text-to-speech conversion (YorkTalk [Col95]) and linguistic theory testing [Bir95], and there has been a great deal of work on implementations and the computational complexity of constraint-satisfaction systems in general [Mac92], the only previous complexity-theoretic (as opposed to purely algorithmic [BE94, Bir95]) work on Declarative Phonology is that done by myself in [War96a]. This section contains the first complete presentation of my results; as such, it replaces the partial analyses given without proofs in [War96a], and extends the framework described in that reference both to use a more complex representation incorporating hidden and visible components and to give a formulation and analysis of the decoding problem for Declarative Phonology systems.

The problems examined below formalize Declarative Phonology in terms of finite-state automata which encode languages over the string-encodings of the simplified autosegmental representation described in Section 2.2.2. The formalization of individual components of a Declarative Phonology system is as follows:

- Lexical, surface, and full forms will be specified in one of two ways:

    1. As strings of the form $v_1h_1v_2h_2\ldots v_nh_n$ with visible and hidden component strings $v_1v_2\ldots v_n$ and $h_1h_2\ldots h_n$, respectively, in which underspecified elements are encoded by the subsets Σ_{vU} and Σ_{hU} of the visible- and hidden-component alphabets (see Figures 2.7 and 2.8).
    2. As DFA on $2n+2$ states that encode strings of the form $v_1h_1v_2h_2\ldots v_nh_n$ with visible and hidden component strings $v_1v_2\ldots v_n$ and $h_1h_2\ldots h_n$, respectively, in which underspecified elements are encoded by allowing multiple transitions between the appropriate states. In order to ensure that these DFA are total, each such DFA incorporates a *Fail* state and the appropriate transitions as described in Section 2.2.3.

  A full form is **consistent** with a lexical or surface form if all fully specified elements of the two forms are the same and all underspecified elements in the surface or lexical form are legally specified in the full form relative to the types of underspecification encoded in those elements. Regardless of the amount of underspecified material, the length of a surface, lexical, or full form is $2n$; this is made clear by reference to

[3]This is not strictly true. A very limited form of conflict is sometimes allowed in which a constraint and a special case of that constraint both apply to some portion of the form and produce different results, and the conflict is resolved in favor of the special-case constraint. This preference for the more specific alternative is known as the *Elsewhere Condition*. See [Sco93, SCB96] for discussions of the role of the Elsewhere Condition in Declarative Phonology.

Figures 2.7 and 2.8. Note that forms will be stated in problems as strings; however, their DFA equivalents will be used in many constructions in the proofs in the following section.

- Constraints in P will be specified as CDFA, and constraint conjunction will correspond to CDFA intersection. Several different ways for computing CDFA intersections are described in the proofs given in the following section.

- The lexicon $D \subseteq (\Sigma_v \Sigma_h)^+$ will be specified as a DFA $DFA(D) = \langle Q_D, \Sigma_h \cup \Sigma_v, \delta_D, s_D, F_D \rangle$ on $|Q_D|$ states which recognizes the language D^+. Some constructions given in the next section will use the modified lexicon DFA $DFA_{FF}(D) = \langle Q_D, (\Sigma_h - \Sigma_{hU}) \cup (\Sigma_v - \Sigma_{vU}), \delta'_D, s_D, F_D \rangle$ such that δ'_D is identical to δ_D except that each transition-entry of the form (q, x, q'), $x \in \Sigma_{hU} \cup \Sigma_{vU}$, in δ_D is replaced by the set of transitions $\{(q, y, q') \mid y \text{ is valid specification of } x\}$ in δ'_D. Note that $DFA_{FF}(D)$ accepts a string x if and only if x is a full form string that is consistent with some lexical form string that is accepted by $DFA(D)$.

Note that in the Declarative Phonology systems defined above, constraints are restricted to specifying underspecified elements in the given form rather than adding arbitrary amounts of structure to this form. This was done not only to focus the analysis on the simplest case of structure-altering constraints within such systems, but also to make the results derived here compatible with those derived in this thesis for the other phonological theories, which are similarly restricted such that they cannot add structure to or delete structure from the given form.

The only other finite-state formulation of Declarative Phonology to date is that given in [BE94]. In this formulation, given forms and constraints are represented as state-labeled finite-state automata (SFA), a variant of the FSA defined in Section 2.2.3 in which states are labeled with subsets of the automaton's associated alphabet, and the conjunction of the given form and the constraints corresponds to the intersection of the SFA for that form and those constraints. Though the finite-state machinery invoked in this formulation is very different from that used in this section, both formulations ultimately represent autosegmental structures as symbol strings in which each symbol encodes the set of features in all autosegments that are are linked to a particular slot on the timing tier (that is, each formulation essentially uses the second of the encodings of autosegmental structures into symbol strings given on page 40 in Section 2.2.3). Hence, it should be possible to adapt the reductions and algorithms given in the following section to produce a systematic parameterized analysis for the formulation of Declarative Phonology given in [BE94]. As the details of SFA computation and intersection are somewhat involved, such an analysis is left as a topic for future research. For the purposes of this thesis, the formulation defined in this section is more appropriate than that given in [BE94] because the formulation defined here allows derived results both to be compared with results derived for the other finite-state phonological theories examined in this thesis and to better reflect the computational complexity of the mechanisms underlying Declarative Phonology rather than effects induced by arbitrarily complex autosegmental representations (see Section 2.2.2).

My analysis will focus on the following problems:

DP-ENCODE
Instance: A Declarative Phonology system $\langle P, D\rangle$ in which the maximum context-size of any CDFA in P is c and a lexical form string u.
Question: Is there a full form string that is consistent with u and is accepted by each CDFA in P?

DP-DECODE
Instance: A Declarative Phonology system $\langle P, D\rangle$ in which the maximum context-size of any CDFA in P is c and a surface form string s.
Question: Is there a full form string that is consistent with both s and some lexical form u generated by D and that is accepted by each CDFA in P?

At first glance, these problems do not seem to correctly formalize Declarative Phonology because they allow invalid systems of constraints, e.g., individual constraints may conflict and/or the system of constraints may not have a solution. However, as will be discussed in Section 4.5.3, results derived relative to these problems are still relevant to problems that are restricted to operate on valid systems of constraints within Declarative Phonology.

4.5.2 Analysis

The systematic parameterized analysis given in this section will focus on the following aspects:

- The number of CDFA in P ($|P|$).
- The maximum context-size of any CDFA in P (c).
- The length of the given lexical / surface form ($|u|$ / $|s|$).
- The maximum number of states in any CDFA in P ($|Q|$).
- The sizes of the hidden- and visible-component alphabets ($|\Sigma_h|$, $|\Sigma_v|$).

Consider the following reductions.

Lemma 4.5.1 *BDFAI* $\leq_m$ DP-ENCODE *such that* $|\Sigma_v| = 1$.

Proof: Given an instance $\langle A, \Sigma, k\rangle$ of BDFAI, construct the following instance $\langle P', c', D', \Sigma'_h, \Sigma'_{hU}, \Sigma'_v, \Sigma'_{vU}, u'\rangle$ of DP-ENCODE: Let $\Sigma'_v = \{\Delta\}$ for some symbol $\Delta \not\in \Sigma$, $\Sigma_{vU} = \phi$, $\Sigma'_h = \Sigma \cup \{\Box\}$ for some symbol $\Box \not\in \Sigma$, $\Sigma'_{hU} = \{\Box\}$ such that $\Box$ may be specified as any symbol in Σ, u' be the lexical form string whose visible component string is Δ^k and whose hidden component string is $\Box^k$, and $D' = \{\Delta\Box\}$. Given a DFA $a = \langle Q, \Sigma, \delta, s, F\rangle$, let $CDFA(a) = \langle Q, \Sigma \cup \{\Delta\}, \delta', s, F, c\rangle$ such that $c = 2k$ and $\delta' = \delta \cup \{(q, \Delta, q) \mid q \in Q\}$. Note that $CDFA(a)$ accepts a string $x \in (\Delta\Sigma)^k$ if and only if a accepts the length-k version of x from which all Δ-symbols have been deleted. Let P' be the set consisting of all CDFA $CDFA(a)$ corresponding to DFA $a \in A$. This construction can be done in time polynomial in the size of the given instance of BDFAI.

To see that this construction is a many-one reduction, note that any solution to the constructed instance of DP-ENCODE is a full form string that is consistent with u' and is accepted by each CDFA in P', and that the hidden component string of this full form string corresponds to a string in Σ^k that is accepted by each DFA in A. Moreover each solution to the given instance of BDFAI is a string $s \in \Sigma^k$ that is accepted by every DFA in A, and that this string corresponds to the hidden component string of a full form that is consistent with u' and is accepted by every CDFA in P'. Thus, the given instance of BDFAI has a solution if and only if the constructed instance of DP-ENCODE has a solution.

Note that in the constructed instance of DP-ENCODE, $|P'| = |A|$, $c' = |u'| = 2k$, $|Q'| = |Q|$, $|\Sigma'_h| = |\Sigma| + 1$, and $|\Sigma'_v| = 1$. ∎

The following lemma takes advantage of the observation that the specified portion of the lexical form can be in the hidden component instead of the visible component. This is possible because DP-ENCODE does not require that given lexical forms have any particular specified hidden or visible elements. In the absence of further restrictions on lexical forms, there is thus a certain symmetry about the roles of hidden and visible components in DP-ENCODE that is explicit in the results stated in parts (9) and (10) of Theorem 4.5.4.

Lemma 4.5.2 *BDFAI* $\leq_m$ DP-ENCODE *such that* $|\Sigma_h| = 1$.

Proof: Given an instance $\langle A, \Sigma, k\rangle$ of BDFAI, construct the following instance $\langle P', c', D', \Sigma'_h, \Sigma'_{hU}, \Sigma'_v, \Sigma'_{vU}, u'\rangle$ of DP-ENCODE: Let $\Sigma'_v = \Sigma \cup \{\Box\}$ for some symbol $\Box \notin \Sigma$, $\Sigma'_{vU} = \{\Box\}$ such that $\Box$ may be specified as any symbol in Σ, $\Sigma'_h = \{\Delta\}$ for some symbol $\Delta \notin \Sigma$, $\Sigma_{hU} = \phi$, u' be the lexical form string whose visible component string is $\Box^k$ and whose hidden component string is Δ^k, and $D' = \{\Box\Delta\}$. Given a DFA $a = \langle Q, \Sigma, \delta, s, F\rangle$, let $CDFA(a) = \langle Q, \Sigma \cup \{\Delta\}, \delta', s, F, c\rangle$ such that $c = 2k$ and $\delta' = \delta \cup \{(q, \Delta, q) \mid q \in Q\}$. Note that $CDFA(a)$ accepts a string $x \in (\Sigma\Delta)^k$ if and only if a accepts the length-k version of x from which all Δ-symbols have been deleted. Let P' be the set consisting of all CDFA $CDFA(a)$ corresponding to DFA $a \in A$. This construction can be done in time polynomial in the size of the given instance of BDFAI.

To see that this construction is a many-one reduction, note that any solution to the constructed instance of DP-ENCODE is a full form string that is consistent with u' and is accepted by each CDFA in P', and that the visible component string of this full form string corresponds to a string in Σ^k that is accepted by each DFA in A. Moreover each solution to the given instance of BDFAI is a string $s \in \Sigma^k$ that is accepted by every DFA in A, and that this string corresponds to the visible component string of a full form that is consistent with u' and is accepted by every CDFA in P'. Thus, the given instance of BDFAI has a solution if and only if the constructed instance of DP-ENCODE has a solution.

Note that in the constructed instance of DP-Encode, $|P'| = |A|$, $c' = |u'| = 2k$, $|Q'| = |Q|$, $|\Sigma'_h| = 1$, and $|\Sigma'_v| = |\Sigma| + 1$. ∎

Unfortunately, it is not so easy to derive reductions in which the constructed instances of DP-Encode have $|\Sigma_h| = 1$ because DP-Decode requires that the given surface form consist only of a fully-specified visible component.

Lemma 4.5.3 *BDFAI* $\leq_m$ DP-Decode *such that* $|\Sigma_v| = 1$.

Proof: Given an instance $\langle A, \Sigma, k\rangle$ of BDFAI, construct the following instance $\langle P', c', D', \Sigma'_h, \Sigma'_{hU}, \Sigma'_v, \Sigma'_{vU}, s'\rangle$ of DP-Decode: Let Σ'_v, Σ'_{vU}, Σ'_h, Σ'_{hU}, P', c', and D' be constructed as in Lemma 4.5.1 above, and s' be the surface form string whose visible component string is Δ^k and whose hidden form string is $\Box^k$. This construction can be done in time polynomial in the size of the given instance of BDFAI. Note that as the surface form string for s' is equivalent to the lexical form string for u' in Lemma 4.5.1, any full forms that are consistent with s' will also be consistent with the lexical form string u'. Hence, the given instance of BDFAI has a solution if and only if the constructed instance of DP-Decode has a solution by the same argument as in Lemma 4.5.1.

Note that in the constructed instance of DP-Decode, $|P'| = |A|$, $c' = |s'| = 2k$, $|Q'| = |Q|$, $|\Sigma'_h| = |\Sigma| + 1$, and $|\Sigma'_v| = 1$. ∎

The observant reader will have noticed that the reductions above are very similar to those for the KIMMO system in Section 4.4.2. However, the use of CDFA to model constraints and the inclusion of context-size in the list of examined aspects makes for some interesting differences in the proofs given below, particularly those for the FPT algorithms.

Theorem 4.5.4

1. DP-Encode *is* NP*-hard when* $|Q| = 2$ *and* $|\Sigma_v| = 1$.
2. DP-Encode *is* NP*-hard when* $|Q| = 2$ *and* $|\Sigma_h| = 1$.
3. DP-Encode *is* NP*-hard when* $|\Sigma_v| = 1$ *and* $|\Sigma_h| = 2$.
4. DP-Encode *is* NP*-hard when* $|\Sigma_v| = 2$ *and* $|\Sigma_1| = 1$.
5. $\langle c, |\Sigma_h|, |\Sigma_v|\rangle$-DP-Encode *is in* FPT.
6. $\langle |u|, |\Sigma_h|, |\Sigma_v|\rangle$-DP-Encode *is in* FPT.
7. $\langle |P|, c, |Q|\rangle$-DP-Encode *is in* FPT.
8. $\langle |P|, |u|, |Q|\rangle$-DP-Encode *is in* FPT.
9. $\langle |P|, c, |u|, |\Sigma_v|_1\rangle$-DP-Encode *and* $\langle |P|, c, |u|, |\Sigma_h|_1\rangle$-DP-Encode *are* $W[1]$*-hard.*
10. $\langle c, |u|, |Q|_2, |\Sigma_v|_1\rangle$-DP-Encode *and* $\langle c, |u|, |Q|_2, |\Sigma_h|_1\rangle$-DP-Encode *are* $W[2]$*-hard.*

11. *$\langle |P|, |\Sigma_h|_2, |\Sigma_v|_1 \rangle$-DP-ENCODE is $W[t]$-hard for all $t \geq 1$.*

12. *$\langle |Q|_2, |\Sigma_v|_1 \rangle$-DP-ENCODE $\notin XP$ unless $P = NP$.*

13. *$\langle |Q|_2, |\Sigma_h|_1 \rangle$-DP-ENCODE $\notin XP$ unless $P = NP$.*

14. *$\langle |\Sigma_h|_2, |\Sigma_v|_1 \rangle$-DP-ENCODE $\notin XP$ unless $P = NP$.*

15. *$\langle |\Sigma_h|_1, |\Sigma_v|_2 \rangle$-DP-ENCODE $\notin XP$ unless $P = NP$.*

Proof:

Proof of (1): Follows from the NP-hardness of $BDFAI$ when $|Q| = 2$ as established in part (1) of Theorem 4.2.4 and the reduction in Lemma 4.5.1 from BDFAI to DP-ENCODE in which $|Q'| = |Q|$ and $|\Sigma'_v| = 1$.

Proof of (2): Follows from the NP-hardness of $BDFAI$ when $|Q| = 2$ as established in part (1) of Theorem 4.2.4 and the reduction in Lemma 4.5.2 from BDFAI to DP-ENCODE in which $|Q'| = |Q|$ and $|\Sigma'_h| = 1$.

Proof of (3): Follows from the NP-hardness of $BDFAI$ when $|\Sigma| = 2$ as established in part (2) of Theorem 4.2.4 and the reduction in Lemma 4.5.1 from BDFAI to DP-ENCODE in which $|\Sigma'_v| = 1$ and $|\Sigma'_h| = |\Sigma| + 1$.

Proof of (4): Follows from the NP-hardness of $BDFAI$ when $|\Sigma| = 2$ as established in part (2) of Theorem 4.2.4 and the reduction in Lemma 4.5.2 from BDFAI to DP-ENCODE in which $|\Sigma'_v| = |\Sigma| + 1$ and $|\Sigma'_h| = 1$.

Proof of (5): The following algorithm is based on an implicit form of CDFA intersection in which the constraint CDFA in P and the given lexical form string u are encoded into a directed graph such that full form strings that are accepted by all CDFA in P and are consistent with u correspond to special paths in this graph. This construction will be done in two stages: the graph encoding all possible full forms for u will be created and then this graph will be "pruned" to delete all paths that correspond to full forms that are not accepted by the CDFA in P.

Consider first the construction of the full form graph associated with the given lexical form string u. For the sake of generality, this given form string will be referred to as g below. Let c be the largest context-size of any CDFA in P. Each full form string consistent with g can be decomposed into $(|g| - c) + 1$ overlapping substrings of length c. Given the c-decomposition $S_{g,c}$ of g (see Definition 2.2.25 in Section 2.2.3), let g_i be the ith string in this decomposition. Note that each such g_i may contain underspecified elements from Σ_{vU} and Σ_{hU}; hence, let $C(g_i)$ be the set of all strings that are consistent with g_i. The full form graph for g is constructed by the following algorithm.

Algorithm **Construct Full Form Graph (CFFG)**
Input: A form string g.
Output: The arc-labeled full form graph for g.

1. Define the vertex set for the full form graph relative to the set of strings that are consistent with the substrings of g of length c.

   Let this vertex set be $V = \{s\} \cup (\bigcup_{i=1}^{|C(g_1)|} VP(i)) \cup (\bigcup_{i=1}^{(|g|-c)+1} VC(g_i))$, where $VC(g_i)$ is the set of vertices associated with $C(g_i)$ such that each string in $C(g_i)$ has its own unique vertex in $VC(g_i)$ and $VP(j) = \{vp_{j,1}, vp_{j,2}, \ldots, vp_{j,(c-1)}\}$ is a set of "private" vertices associated with each vertex j in $VC(g_1)$. Let $vert_{VC}(i,j)$, $1 \leq j \leq |VC(g_i)|$, be the jth vertex in $VC(g_i)$ under some ordering of the vertices of $VC(g_i)$. Given a vertex $v \in VC(g_i)$, let $vs_{VC}(i,v)$ be its associated string in $C(g_i)$; conversely, given a string $x \in C(g_i)$, let $sv_{VC}(i,x)$ be its associated vertex in $VC(g_i)$.

2. Create the CDFA intersection graph by imposing the appropriate arcs on the vertex-set created in (1).

   Define the directed graph $G = (V, A)$ such that $A = A_1 \cup A_2$ such that A_1 and A_2 are the following two sets of arcs:

   A_1: Arcs connecting s to $VC(g_1)$ via VP, i.e., $\bigcup_{i=1}^{|VC(g_1)|}(\{(s, vp_{i,1})\} \cup \bigcup_{j=1}^{c-2}\{(vp_{i,j}, vp_{i,(j+1)})\} \cup \{(vp_{i,(c-1)}, vert_{VC}(1,i))\})$.

   A_2: Arcs connecting $VC(g_i)$ to $VC(g_{i+1})$, i.e., $\{(u,v) \mid 1 \leq i \leq |g| - c,\ u \in VC(g_i),\ v \in VC(g_{i+1})$, and $vs_{VC}(i,u)$ and $vs_{VC}(i+1,v)$ have a length-$(c-1)$ overlap, i.e., the length-$(c-1)$ suffix of $vs_{VC}(i,u)$ equals the length-$(c-1)$ prefix of $vs_{VC}(i+1,v)$ $\}$.

   This graph can be visualized as an ordered set of $(|g|-c)+2$ columns, with special vertex s being the only vertex in the first column and the ith column, $2 \leq i \leq (|g| - c) + 2$, corresponding to the vertices in $VC(g_{(i-1)})$. Within this graph, vertex s is connected to each vertex $vert_{VC}(1,j)$ in the second column corresponding to $VC(g_1)$ by a path made up of the vertices in $VP(j)$, and vertices in subsequent adjacent columns are connected by arcs corresponding to length-$(c-1)$ overlaps in the strings associated with the vertex-endpoints of those arcs.

3. Label the arcs in sets A_1 and A_2 with symbols from $(\Sigma_h - \Sigma_{hU}) \cup (\Sigma_v - \Sigma_{vU})$ as follows. Given an arc (u,v), let $lab(u,v)$ be the symbol-label associated with that arc.

   A_1: For all i and j, $1 \leq i \leq |VC(g_1)|$ and $1 \leq j \leq (c-2)$, set $lab(s, vp_{i,1})$ to the first symbol in $vs_{VC}(1, vert_{VC}(1,i))$, $lab(vp_{i,j}, vp_{i,(j+1)})$ to the $(j+1)$-st symbol in $vs_{VC}(1, vert_{VC}(1,i))$, and $lab(vp_{i,(c-1)}, vert_{VC}(1,i))$ to the final symbol in $vs_{VC}(1, vert_{VC}(1,i))$.

   A_2: For all i, u, and v such that $1 \leq i \leq |g| - c$, $u \in VC(g_i)$, $v \in VC(g_{i+1})$, and $(u,v) \in A$, set $lab(u,v)$ to the final symbol in $vs_{VC}((i+1), v)$.

Observe that by the construction of this graph, the string formed by concatenating the edge-labels on any path from s to a vertex in $VC(g_{(|g|-c)+1})$ is a full form string consistent with g. Moreover, each full form string f consistent with g has a corresponding path from s to some vertex in $VC(g_{(|g|-c)+1})$ whose vertex-sequence is $s, vp_{i,1}, vp_{i,2}, \ldots, vp_{i,(c-1)}$,

$vert_{VC}(1,i) = sv_{VC}(1,f_1), sv_{VC}(2,f_2), \ldots, sv_{VC}((|f|-c)+1, f_{(|f|-c)+1})$, where f_j, $1 \leq j \leq (|f|-c)+1$, is the jth string in the c-decomposition of f. Hence, the graph constructed above encodes all of the full form strings consistent with a given form g. Several examples of full form graphs are given in Figures 4.7 and 4.8.

Algorithm CFFG and the graph it constructs can be usefully characterized in terms of c, $|g|$, and the following quantities:

- $\max(VC) = \max_{i=1}^{(|g|-c)+1} |C(g_i)|$.
- $\max(arc)$ = the maximum number of arcs originating from any vertex v in any set $VC(g_i)$, $1 \leq i \leq |g| - c$.

Given the above, the number of vertices in the constructed full form graph is $|V| \leq 1 + \max(VC)(c-1) + \max(VC)(|g|-c) \leq \max(VC)|g|$ and the number of arcs is $|A| \leq \max(VC)c + \max(VC)\max(arc)(|g|-c) \leq \max(VC)\max(arc)|g|$. Consider now the running times of the individual steps in the algorithm above:

- As step (1) essentially enumerates the elements of V and their associated strings, step (1) can be done in $O(|V|c) = O(\max(VC)|g|c)$ time.
- If the graph is represented as an adjacency matrix, this matrix can be initialized in $O(|V|^2) = O((\max(VC)|g|)^2)$ time, the arcs in A_1 can be entered into the matrix in $O(|A_1|) = O(\max(VC)|g|)$ time, and the arcs in set A_2 can be entered into the matrix in $O(|V|^2c) = O((\max(VC)|g|)^2c)$ time (compare each element of $C(u_i)$ with each element of $C(u_{i+1})$ for length-$c-1$ overlap for all i, $1 \leq i \leq |g|-c$). Hence, step (2) can be done in $O((\max(VC)|g|)^2c)$ time.
- Step (3) involves looking at the strings associated with the endpoints of every arc in the graph, and can thus be done in $O(|A|c) = O(\max(VC)\max(arc)|g|c)$ time.

Thus, the algorithm CFFG runs in $O((\max(VC)|g|)^2\max(arc)c)$ time.

Consider now how the full form graph constructed above can be modified to delete all paths that correspond to full forms that are consistent with g but are not accepted by all CDFA in P. Observe that any string in $C(g_i)$ that is not accepted by all CDFA in P cannot be part of any full form string that is accepted by all CDFA in P, and hence cannot appear on any path that corresponds to such a full form string. Hence, such vertices and their adjacent arcs can be removed without disturbing the ability of the full form graph to encode such full form strings. Moreover, no full form string that is accepted by all CDFA in P can be deleted from the set of such full form strings encoded by the graph by deleting such vertices (as all of the vertices corresponding to substrings of length c in such a full form are accepted by all CDFA in P and hence will be removed from the graph). This suggests the following algorithm.

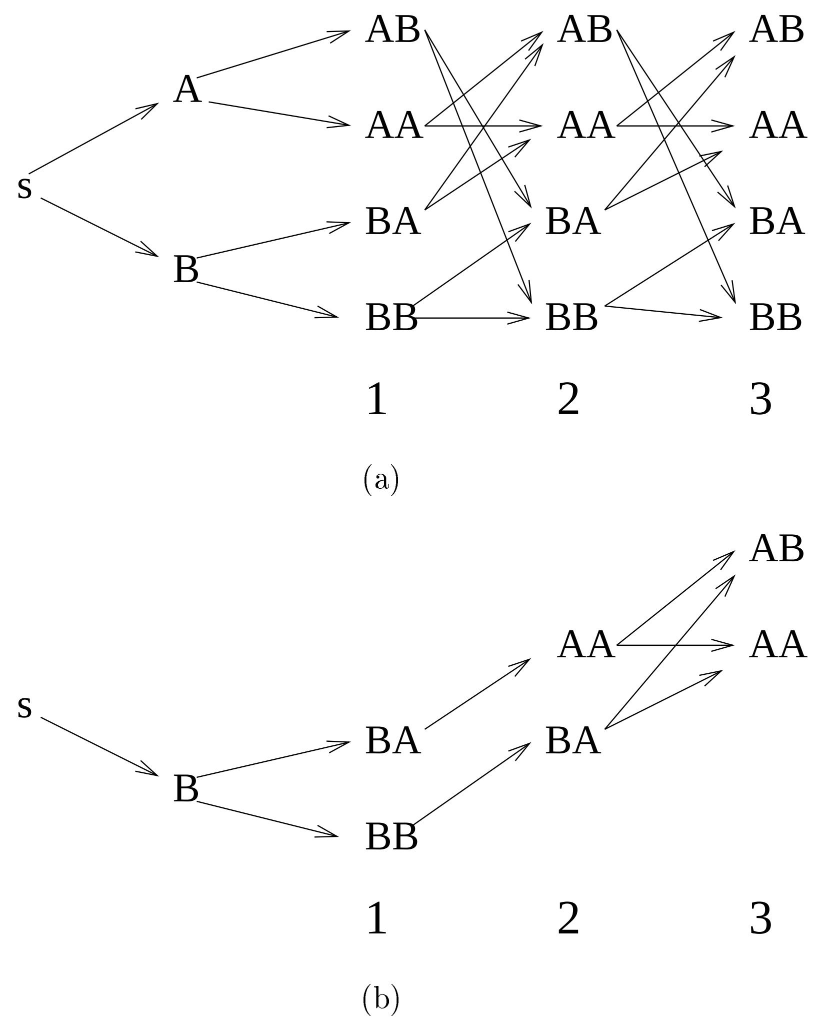
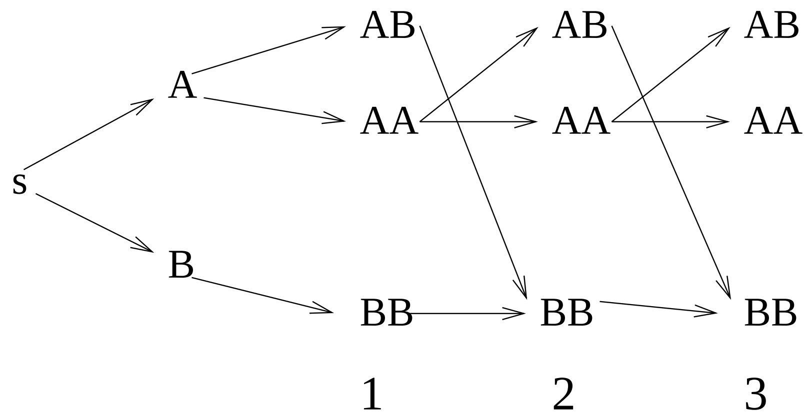


Figure 4.7: Full Form Graphs. This diagram shows various full form graphs constructed relative to an alphabet $\{a, b, \Box\}$ such that symbol $\Box$ can be specified as either a or b, a form string g of length 4, and a context-size $c = 2$. The vertices in the columns labeled 1, 2, and 3 are the vertices in the sets $VC(g_1)$, $VC(g_2)$, and $VC(g_3)$, and the sets of strings labeling the vertices in these sets are the sets $C(g_1)$, $C(g_2)$, and $C(g_3)$, respectively. For the sake of readability, the edge-labelings have been omitted. a) The full form graph for $g = \Box\Box\Box\Box$. b) The full form graph for $g = b\Box a\Box$.



(c)

Figure 4.8: Full Form Graphs (Cont'd). c) The full form graph for g = □□□□ that has been pruned relative to a CDFA with contexts-size 2 that does not allow the substring ba in g. See part (5) of Theorem 4.5.4 for an explanation of terms.

Algorithm **Prune Full Form Graph (PFFG)**
Input: The arc-labeled full form graph for form string g and a set P of CDFA.
Output: The pruned arc-labeled full form graph for g relative to P.

1. Initialize a set *bad* over V to ϕ.
2. For all i, $1 \leq i \leq (|g| - c) + 1$, examine every vertex $v \in VC(g_i)$ and determine if $str(v)$ is accepted by every CDFA in P; if not, add v to *bad*.
3. Remove all vertices in *bad* and their adjacent arcs from the full form graph.

Observe that the pruned full form graph produced by this algorithm encodes the set of full form strings that are consistent with g and accepted by all of the CDFA in P. The pruned full form graph will probably be much smaller than the original full form graph, but the bounds on $|V|$ and $|A|$ derived above will still be applicable. Consider now the running times of the individual steps in the algorithm above:

- If the set *bad* is stored as bit-vector of length $|V|$ such that elements in the set have their corresponding bits set to 1, step (1) can be done in $O(|V|) = O(\max(VC)|g|)$ time.
- Some CDFA may have context sizes that are less than c and hence may have to check several substrings of each string. However, note that the number of such strings is bounded by c; hence, each CDFA can check a string of length c in $O(c^2)$ time. As there are $|P|$ constraints and the string corresponding to each vertex in $VC = \bigcup_{i=1}^{(|g|-c)+1} VC(g_i)$ must be checked, step (2) can be done in $(O|V||P|c^2) = O(\max(VC)$ $|g||P|c^2)$ time.

- In the worst case, *bad* can contain all vertices in VC. Removing all edges adjacent to a vertex in *bad* involves zeroing the row and column vectors in the adjacency matrix for the graph that correspond to arcs originating from and going to that vertex in the graph. Hence, step (3) can be done in $O(|V|^2) = O((\max(VC)|g|)^2)$ time.

Thus, the algorithm PFFG runs in $O((\max(VC)|g|)^2|P|c^2)$ time.

The algorithm for solving DP-ENCODE can now be stated as follows:

1. Construct the full form graph for u by applying algorithm CFFG.
2. Prune the full form graph constructed in (1) relative to P by applying the algorithm PFFG.
3. Apply depth-first search to the graph created in (2) to determine if there is a path in that graph from s to any vertex in $VC(u_{(|u|-c)+1})$. If so, by the manner in which this graph has been constructed, the labels in the arcs on this path can be concatenated to create a full form string that is consistent with u and is accepted by all CDFA in P. If not, there is no such full form string (as any such string could be decomposed into a sequence of overlapping contexts which would correspond to a path in the graph above from s to some vertex in $VC(u_{(|u|-c)+1})$).

Consider first the values of $\max(VC)$ and $\max(arc)$ in this algorithm:

- Each substring of length c in u contains either $\lceil \frac{c}{2} \rceil$ hidden and visible elements (if c is even) or either $\lceil \frac{c}{2} \rceil$ and $\lceil \frac{c}{2} \rceil - 1$ or $\lceil \frac{c}{2} \rceil - 1$ and $\lceil \frac{c}{2} \rceil$ hidden and visible elements (if c is odd), and any of these elements can be underspecified. Hence, there can be at most $|\Sigma_h|^{\lceil \frac{c}{2} \rceil}|\Sigma_v|^{\lceil \frac{c}{2} \rceil} \leq (|\Sigma_h||\Sigma_v|)^c$ strings in any $C(u_i)$. Hence, $\max(VC) \leq (|\Sigma_h||\Sigma_v|)^c$.
- The label on any arc (x, y) in the full form graph such that $x \in VC(u_i)$ and $y \in VC(u_{i+1})$ for any i, $1 \leq i \leq |u| - c$, can be either from the hidden alphabet or the visible alphabet. Hence, $\max(arc) = \max(|\Sigma_h|, |\Sigma_v|)$.

Consider now the running times of the individual steps in this algorithm:

- As algorithm CFFG runs in $O((\max(VC)|g|)^2 \max(arc)c)$ time, step (1) can be done in $O(((|\Sigma_h||\Sigma_v|)^c|u|)^2 \max(|\Sigma_h|, |\Sigma_v|)c) = O((|\Sigma_h||\Sigma_v|)^{2c}|u|^2 c \max(|\Sigma_h|, |\Sigma_v|))$ time.
- As algorithm PFFG runs in $O((\max(VC)|g|)^2|P|c^2)$ time, step (2) can be done in $O(((|\Sigma_h||\Sigma_v|)^c|u|)^2|P|c^2) = O((|\Sigma_h||\Sigma_v|)^{2c}|u|^2|P|c^2)$ time.
- As depth-first search runs in $O(|V|+|A|)$ time, step (3) of this algorithm can be done in $O(\max(VC)|g| + \max(VC)\max(arc)|g|) = O(\max(VC)\max(arc)|g|) = O((|\Sigma_h||\Sigma_v|)^c$ $|u| \max(|\Sigma_h|, |\Sigma_v|))$ time.

Hence the algorithm for DP-ENCODE described above runs in $O((|\Sigma_h||\Sigma_v|)^{2c}|u|^2|P|c^2 \max(|\Sigma_h|, |\Sigma_v|))$ time, which is fixed-parameter tractable relative to c, $|\Sigma_h|$, and $|\Sigma_v|$.

Proof of (6): As it is always the case that $c \leq |u|$, the result follows from (5) and Lemma 2.1.33.

Proof of (7): Unlike the algorithm given in (5), the following algorithm is based on an explicit form of CDFA intersection in which CDFA are transformed into a special kind of DFA that are subsequently intersected in the conventional manner with the lexical form DFA associated with the given lexical form string. This transformation takes advantage of the following observation: given a string x being evaluated by a CDFA a of context-size c, every symbol of x can be part of at most c contexts, and hence at most c contexts are **active**, i.e., being processed by a, at any given time. Hence, the state of a CDFA can be encoded as a c-component vector of states of its underlying DFA that represents the intermediate results of each of the up to c active contexts, and a transition relation on these vector-states can be configured to form a "pipeline" for the parallel processing of up to c simultaneously active contexts on a given string. These pipelines are reminiscent of systolic array architectures used in high-performance computing; hence, the DFA that implement these pipelines are called systolic DFA. In the following, let a vector-state in a systolic DFA with state-components $q_1, \ldots, q_k$ be denoted by $[q_1, \ldots, q_k]$.

Given a CDFA $a = \langle Q, \Sigma, \delta, s, F, c\rangle$, the **systolic DFA corresponding to** a is the DFA $SDFA(a) = \langle Q', \Sigma', \delta', s', F'\rangle$ such that $Q' = \{Fail\} \cup \bigcup_{i=1}^{c} Q_i$ where $Q_i = \{[s, q_1, q_2, \ldots, q_{i-1}] \mid \{q_1, q_2, \ldots, q_{i-1}\} \in Q\}$, $\Sigma' = \Sigma$, $s' = [s]$, $F' = Q' - \{Fail\}$, and $\delta' = \delta_1 \cup \delta_2 \cup \delta_3$, where δ_1, δ_2, and δ_3 are specified as follows:

δ_1**:** Map states in Q_i to Q_{i+1} relative to δ for $1 \leq i \leq (c-1)$, i.e., $\delta_1 = (q, x, q') \mid q = [s, q_1, \ldots, q_{i-1}] \in Q_i$, $q' = [s, \delta(s, x), \delta(q_1, x), \ldots, \delta(q_{i-1}, x)] \in Q_{i+1}$, and $1 \leq i \leq (c-1)\}$.

δ_2**:** Map states in Q_c to $Q_c \cup \{Fail\}$ relative to δ, i.e., $\delta_2 = \{(q, x, q') \mid q = [s, q_1, \ldots, q_c] \in Q_c$, $q' = [s, \delta(s, x), \delta(q_1, x), \ldots, \delta(q_{c-1}, x)] \in Q_c$, and $\delta(q_c, x) \in F\} \cup \{(q, x, Fail) \mid q = [s, q_1, \ldots, q_c] \in Q_c$ and $\delta(q_c, x) \notin F\}$.

δ_3**:** Direct all remaining possible transitions to $Fail$, i.e., $\delta_3 = \{(q, x, Fail) \mid q \in Q'$, $x \in \Sigma$, and $\nexists q' \in Q'$ such that $(q, x, q') \in \delta_1$ or $(q, x, q') \in \delta_2\}$.

Note that the creation of the appropriate transitions for the vector states in δ_1 and δ_2 requires that the transition relation for the underlying DFA of the given CDFA be complete, i.e., the underlying DFA for the given CDFA is a total DFA. An example systolic DFA is given in Figure 4.9. Note that relative to its associated CDFA, a systolic DFA has $|Q'| = 1 + \sum_{i=0}^{c-1} |Q|^i = 1 + (|Q|^c - 1/(|Q| - 1)) \leq |Q|^c$ states and at most $|Q'||\Sigma| \leq |Q|^c|\Sigma|$ transitions, and can be constructed from that CDFA in $O(|Q'||\Sigma|) = O(|Q|^c|\Sigma|)$ time.

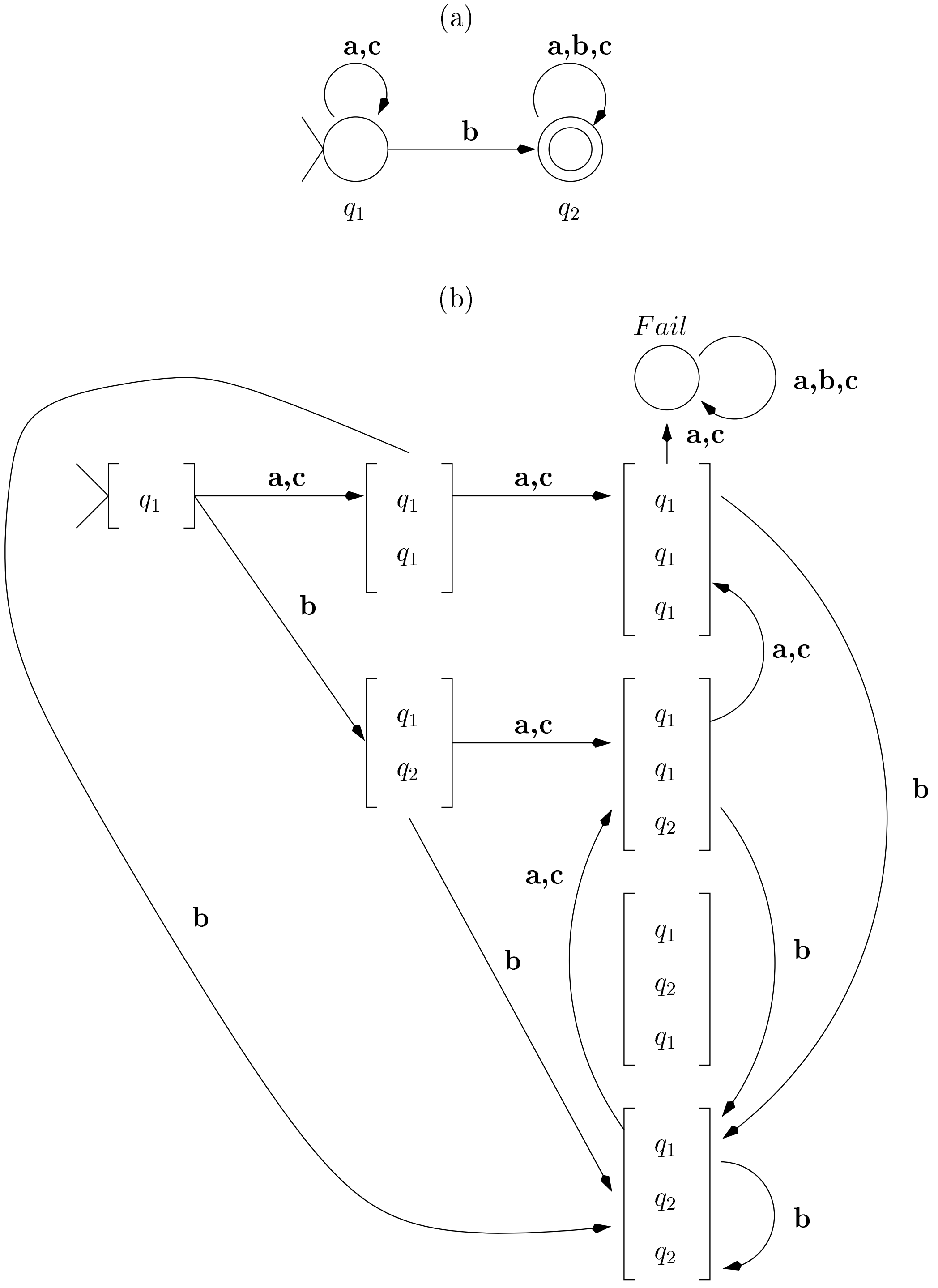


Figure 4.9: A Systolic Deterministic Finite-State Acceptor. a) A DFA of the type constructed in the reduction from DOMINATING SET to BDFAI in Lemma 4.2.2. b) The systolic DFA whose associated CDFA has the underlying DFA in (a) and context-size $c = 3$.

Claim: *A string x is accepted by a CDFA if and only if x is accepted by the systolic DFA created from that CDFA by the above construction.*

Proof of Claim: Let the **yield-path** of a string x relative to a (systolic) DFA be the sequence of states $s, q_1, q_2, \ldots, q_{|x|}$ encountered by the (systolic) DFA in processing x. Given a state $q = [q_1, q_2, \ldots, q_n]$ of a systolic DFA, let $st(q, i) = q_i$, $1 \leq i \leq n$, i.e., $st(q, i)$ is the i-th state-component of vector-state q. Given a yield path $q_1, q_2, \ldots, q_{|x|+1}$ of a systolic DFA on a string x such that x is accepted by that systolic DFA, i.e., $q_{|x|+1} \in F$, let the i-th **trace** of this yield-path, $1 \leq i \leq ((|x| - c) + 1)$, be the sequence of states $st(q_i, 1), st(q_{i+1}, 2), \ldots, st(q_{i+c})$. Note that the i-th trace corresponds to the yield-path for the underlying DFA of the CDFA on which the systolic DFA is based relative to the i-th substring of length c in x.

($\Rightarrow$) If a string x is accepted by the CDFA with context-size c, then all substrings of x of length c are accepted by the underlying DFA for that CDFA — that is, for each of the $(|x| - c) + 1$ substrings of x, the yield-path for each substring relative to the underlying DFA ends in a final state of that DFA. This set of yield-paths can be interpreted as a set of traces, and be used to reconstruct a yield-path for the systolic DFA associated with the CDFA (note that several partial yield-paths will be necessary to fill in state-components of the final $(c - 1)$ states in the systolic DFA yield-path that correspond to partial traces for the substrings of x starting from the last $c-1$ symbols of x). As each of the DFA yield-paths ended in a final state, the constructed yield-path for the systolic DFA must end in a final state. Hence, x is accepted by the systolic DFA.

($\Leftarrow$) If a string x is accepted by the systolic DFA, then all traces in the yield-path for x of the systolic DFA end in states that are final relative to the underlying DFA of the CDFA (else, at some point, the systolic DFA would have entered state $Fail$ and x could not have been accepted). As each trace corresponds to a yield-path of the underlying DFA on a substring of x of length c, this means that all substrings of x of length c must be accepted by the underlying DFA, i.e., the CDFA accepts x. ∎

The construction above is part of the following algorithm for DP-Encode:

1. Construct the set P' of systolic DFA corresponding to the CDFA in P.
2. Intersect the systolic DFA in P' with the given lexical form DFA.
3. Apply depth-first search to the transition diagram associated with the intersection DFA created in (2) to determine if there is a path from the start state to any final state in this DFA. Any such path corresponds to a full form string that is both consistent with the given lexical form string and accepted by all constraint CDFA in P.

Note each systolic DFA encodes all full forms that have no violations relative to that systolic DFA's associated CDFA – hence, in order to encode all full form strings that have no violations relative to any of the CDFA in P *and* are also consistent with u, it is necessary to add the lexical form DFA for u to the intersection. This should be compared with the construction in part (5) of this theorem, in which such u-consistency restrictions are explicitly encoded into the full form graph.

Consider the time complexity of each step in the algorithm described above:

- As the systolic DFA for each CDFA in P can be created in $O(|Q|^c(|\Sigma_h| + |\Sigma_v|))$ time, step (1) runs in $O(|Q|^c|P|(|\Sigma_h| + |\Sigma_v|))$ time.
- As each systolic DFA has $\leq |Q|^c$ states and the given lexical form DFA has $|u| + 1 \leq 2|u|$ states, step (2) can be done in $O(|Q|^{c|P|}|u|(|\Sigma_h| + |\Sigma_v|)^2)$ time (see Table 2.3).
- As the graph associated with the transition diagram for the intersection DFA constructed in (2) has $|V| \leq (|Q|^c)^{|P|}(|u|+1) \leq 2|Q|^{c|P|}|u|$ vertices and $|A| \leq (|Q|^c)^{|P|}$ $(|u| + 1)(|\Sigma_h| + |\Sigma_v|) \leq 2|Q|^{c|P|}|u|(|\Sigma_h| + |\Sigma_v|)$ arcs and depth-first search runs in $O(|V| + |A|)$ time, step (3) runs in $O(|Q|^{c|P|}|u|(|\Sigma_h| + |\Sigma_v|))$ time.

Given the above, the algorithm thus runs in $O(|Q|^{c|P|}|P||u|(|\Sigma_h| + |\Sigma_v|)^2)$ time, which is fixed-parameter tractable relative to $|P|$, c, and $|Q|$.

Proof of (8): As it is always the case that $c \leq |u|$, the result follows from (7) and Lemma 2.1.33.

Proofs of (9 – 11): The first parts of (9) and (10) and all of (11) follow from the W-hardness results for BDFAI established in parts (5 – 7) of Theorem 4.2.4, the reduction in Lemma 4.5.1 from BDFAI to DP-ENCODE in which $|P'| = |A|$, $c' = |u'| = 2k$, $|Q'| = |Q|$, $|\Sigma'_h| = |\Sigma|+1$, and $|\Sigma'_v| = 1$, and Lemma 2.1.25. The second parts of (9) and (10) follow from the W-hardness results established in parts (5) and (6) of Theorem 4.2.4, the reduction in Lemma 4.5.2 from BDFAI to DP-ENCODE in which $|P'| = |A|$, $c' = |u'| = 2k$, $|Q'| = |Q|$, $|\Sigma'_h| = 1$, and $|\Sigma'_v| = |\Sigma| + 1$, and Lemma 2.1.25.

Proofs of (12 – 15): Follow from (1 – 4) and Lemma 2.1.35. ∎

Theorem 4.5.5

1. DP-DECODE *is NP-hard when* $|Q| = 2$ *and* $|\Sigma_v| = 1$.
2. DP-DECODE *is NP-hard when* $|\Sigma_h| = 2$ *and* $|\Sigma_v| = 1$.
3. $\langle c, |\Sigma_h|\rangle$-DP-DECODE *is in FPT.*
4. $\langle |s|, |\Sigma_h|\rangle$-DP-DECODE *is in FPT.*
5. $\langle |P|, c, |Q|\rangle$-DP-DECODE *is in FPT.*
6. $\langle |P|, |s|, |Q|\rangle$-DP-DECODE *is in FPT.*

7. *$\langle |P|, c, |s|, |\Sigma_v|_1 \rangle$-DP-DECODE is $W[1]$-hard.*

8. *$\langle c, |s|, |Q|_2, |\Sigma_v|_1 \rangle$-DP-DECODE is $W[2]$-hard.*

9. *$\langle |P|, |\Sigma_h|_2, |\Sigma_v|_1 \rangle$-DP-DECODE is $W[t]$-hard for all $t \geq 1$.*

10. *$\langle |Q|_2, |\Sigma_v|_1 \rangle$-DP-DECODE $\notin XP$ unless $P = NP$.*

11. *$\langle |\Sigma_h|_2, |\Sigma_v|_1 \rangle$-DP-DECODE $\notin XP$ unless $P = NP$.*

Proof:

Proof of (1): Follows from the NP-hardness of BDFAI when $|Q| = 2$ as established in part (1) of Theorem 4.2.4 and the reduction in Lemma 4.5.3 from BDFAI to DP-DECODE in which $|Q'| = |Q|$ and $|\Sigma'_v| = 1$.

Proof of (2): Follows from the NP-hardness of BDFAI when $|\Sigma| = 2$ as established in part (2) of Theorem 4.2.4 and the reduction in Lemma 4.5.3 from BDFAI to DP-DECODE in which $|\Sigma'_h| = |\Sigma|$ and $|\Sigma'_v| = 1$.

Proof of (3): The following algorithm is is a modification of the implicit CDFA intersection algorithm given in part (5) of Theorem 4.5.4 which takes into account the constraints on the full form that are implicit in the lexicon. As the algorithm description given below will draw heavily on various terms and quantities defined in part (5) of Theorem 4.5.4, the reader should perhaps review that notation before reading the rest of this proof.

The steps in this algorithm for DP-DECODE are as follows:

1. Construct the full form graph for the given surface form s by applying algorithm CFFG from part (5) of Theorem 4.5.4.

2. Prune the full form graph constructed in (1) relative to P by applying algorithm PFFG from part (5) of Theorem 4.5.4.

3. Interpret the pruned full form graph constructed in (2) as the transition diagram of a DFA $A = \langle Q, (\Sigma_h - \Sigma_{hU}) \cup (\Sigma_v - \Sigma_{vU}), \delta, s, F \rangle$ in which $Q = V$, δ is constructed from the graph and its labels, the start state is vertex s in the graph, and the final states are the vertices in $VC(s_{(|s|-c)+1})$.

4. Intersect the DFA constructed in (3) with the lexicon DFA $DFA_{FF}(D)$.

5. Apply depth-first search to the transition diagram for the intersection DFA constructed in (4) to determine if any of its final states are reachable from its start state. If so, by the manner in which this DFA has been constructed, the labels on the arcs in this path can be concatenated to create a full form string that is both consistent with s and accepted by all CDFA in P and the lexicon DFA $DFA_{FF}(D)$. If not, there is no such full form string by reasoning similar to that given in part (5) of Theorem 4.5.4.

Consider first the values of $\max(VC)$ and $\max(arc)$ in this algorithm:

- As all visible elements in s are fully specified, the only elements that can be underspecified in a c-length substring of s are hidden elements. Such a substring can contain either $\lceil \frac{c}{2} \rceil$ hidden elements (if c is even) or $\lceil \frac{c}{2} \rceil$ or $\lceil \frac{c}{2} \rceil - 1$ hidden elements (if c is odd), and as any of these elements can be underspecified, there can be at most $|\Sigma_h|^{\lceil \frac{c}{2} \rceil} \leq |\Sigma_h|^c$ strings in any $C(s_i)$. Hence, $\max(VC) \leq |\Sigma_h|^c$
- The label on any arc (x, y) in the full form graph such that $x \in VC(s_i)$ and $y \in VC(s_{i+1})$ for any i, $1 \leq i \leq |s| - c$, can be either from the hidden alphabet or the visible alphabet but only the hidden elements are underspecified in s. Hence, $\max(arc) = \max(|\Sigma_h|, 1) = |\Sigma_h|$.

Thus, the pruned full form graph created in (2) has $|V| \leq \max(VC)|g| \leq |\Sigma_h|^c|s|$ vertices and $|A| \leq \max(VC)\max(arc)|g| \leq |\Sigma_h|^c|\Sigma_h||s| \leq |\Sigma_h|^{c+1}|s|$ arcs. Consider now the running times of the individual steps in this algorithm:

- As algorithm CFFG runs in $O((\max(VC)|g|)^2 \max(arc)c)$ time, step (1) can be done in $O(((|\Sigma_h|^c|s|)^2|\Sigma_h|)c) = O(|\Sigma_h|^{2c+1}|s|^2c)$ time.
- As algorithm PFFG runs in $O((\max(VC)|g|)^2|P|c^2)$ time, step (2) can be done in $O((|\Sigma_h|)^c|s|)^2|P|c^2) = O(|\Sigma_h|^{2c}|s|^2|P|c^2)$ time.
- As step (3) effectively examines each arc and vertex in the pruned full form graph, step (3) can be done in $O(|V| + |A|) = O(|\Sigma_h|^c|s| + |\Sigma_h|^{c+1}|s|) = O(|\Sigma_h|^{c+1}|s|)$ time.
- As the lexicon DFA $DFA_{FF}(D)$ has $|Q_D|$ states and the DFA created in step (4) has $|V|$ states, step (4) can be done in $O(|V||Q_D|) = O(|\Sigma_h|^c|s||Q_D|)$ time (see Table 2.3). Note that though each state in the lexicon DFA is the source of at most $(|\Sigma_h| + |\Sigma_v|)$ transitions, each state in the DFA created in step (3) is the source of at most $|\Sigma_h|$ transitions; hence, each vertex in the graph associated with the transition diagram of the intersection DFA of these two DFA can be the source of at most $|\Sigma_h|$ arcs. Thus, the graph associated with the transition diagram has $|V'| \leq |V||Q_D| = \max(VC)|g||Q_D| = |\Sigma_h|^c|s||Q_D|$ vertices and $|A'| \leq |V'||\Sigma_h| = |\Sigma_h|^c|s||Q_D||\Sigma_h| = |\Sigma_h|^{c+1}|s||Q_D|$ arcs.
- As depth-first search runs in $O(|V| + |A|)$ time, step (5) of this algorithm can be done in $O(|V'| + |A'|) = O(|\Sigma_h|^c|s||Q_D| + |\Sigma_h|^{c+1}|s||Q_D|) = O(|\Sigma_h|^{c+1}|s||Q_D|)$ time.

Hence the algorithm for DP-Decode described above runs in $O(|\Sigma_h|^{2c+1}|s|^2|P|c^2|Q_D|)$ time, which is fixed-parameter tractable relative to c and $|\Sigma_h|$.

Proof of (4): As it is always the case that $c \leq |s|$, the result follows from (3) and Lemma 2.1.33.

Proof of (5): The required algorithm is a modification of the algorithm given in part (7) of Theorem 4.5.4 in which step (2) is modified such that in the set of DFA to be intersected, the surface form DFA for surface form string s replaces the lexical form DFA and $DFA_{FF}(D)$ is added. Step (2) now runs in in $O(|Q|^{c|P|}|s||Q_D|(|\Sigma_h| + |\Sigma_v|)^2)$ time. As the graph associated with the transition diagram for this modified intersection DFA has $|V| \leq (|Q|^c)^{|P|}(|u| + 1)|Q_D| \leq 2|Q|^{c|P|}|u||Q_D|$ vertices and $|A| \leq (|Q|^c)^{|P|}(|u| + 1)|Q_D|$ $(|\Sigma_h| + |\Sigma_v|) \leq 2|Q|^{c|P|}|u||Q_D|(|\Sigma_h| + |\Sigma_v|)$ arcs, step (3) now runs in $O(|Q|^{c|P|}|u||Q_D|$ $(|\Sigma_h| + |\Sigma_v|))$ time. Thus, the algorithm for DP-Decode runs in $O(|Q|^{c|P|}|P||u||Q_D|$ $(|\Sigma_h| + |\Sigma_v|)^2)$ time, which is fixed-parameter tractable relative to $|P|$, c, and $|Q|$.

Proof of (6): As it is always the case that $c \leq |s|$, the result follows from (5) and Lemma 2.1.33.

Proofs of (7 – 9): Follow from the W-hardness results for BDFAI established in parts (5 – 7) of Theorem 4.2.4, the reduction in Lemma 4.5.3 from BDFAI to DP-Decode in which $|P'| = |A|$, $c' = |s'| = 2k$ $|Q'| = |Q|$, $|\Sigma'_v| = |\Sigma| + 1$, and $|\Sigma'_h| = 1$, and Lemma 2.1.25.

Proofs of (10) and (11): Follow from (1) and (2) and Lemma 2.1.35. ∎

4.5.3 Implications

All parameterized complexity results for problems DP-Encode and DP-Decode that are either stated or implicit in the lemmas and theorems given in the previous section are shown in Tables 4.10 and 4.11. Consider the implications of these results for each problem in turn:

- DP-Encode: The sources of polynomial-time intractability are $\{|P|, c, |Q|\}$ and $\{c, |\Sigma_h|, |\Sigma_v|\}$. The mechanisms associated with these sources are the constraint set (specifically the intersection DFA formed from the systolic DFA associated with all constraint CDFA) and the set of possible context-size strings. All of the aspects in these sources are defining properties of a Declarative Phonology system, and hence none of them can be eliminated to reduce the complexity of the problem. However, as context-sizes are typically very small and alphabets are of constant size, the FPT algorithm underlying the latter source should be efficient in practice (that is, assuming that the simplified representation and associated notion of context-size used here translate via reductions to a version of Declarative Phonology that is actually used). A closer inspection of the algorithm in part (5) of Theorem 4.5.4 shows that the size of each set $C(u_i)$ is actually $|\Sigma_h|^{U_h(u_i)}|\Sigma_v|^{U_v(u_i)}$, where $U_h(u_i)$ and $U_v(u_i)$ are the numbers of underspecified visible and hidden elements in u_i. Though $U_h(u_i) = U_v(u_i) = \lceil \frac{c}{2} \rceil$ in the worst case, they will typically be smaller; thus, $|C(u_i)|$ (and $|VC(u_i)|$) will be less than $|\Sigma_h|^{\lceil \frac{c}{2} \rceil}|\Sigma_v|^{\lceil \frac{c}{2} \rceil}$, and the algorithm will run much faster than the original analysis suggests. This also shows that the latter source of polynomial-time intractability above is actually a generalization (in the sense of Lemma 2.1.30) of another source of polynomial-time intractability that is based on the number of underspecified hidden and visible elements in the given form.

- DP-DECODE: The sources of polynomial-time intractability are $\{|P|, c, |Q|\}$ and $\{c, |\Sigma_h|\}$. The mechanisms associated with these sources are the constraint set (specifically the intersection DFA formed from the systolic DFA associated with all constraint CDFA) and the set of possible context-size strings with respect to a fully-specified visible component. As in DP-ENCODE, all of the aspects in these sources are defining properties of a Declarative Phonology system, and hence none of them can be eliminated to reduce the complexity of the problem; however, for the same reasons given above, the FPT algorithm underlying the latter source should be efficient in practice (though, as the whole hidden component is unspecified and the whole visible component is specified in every given surface form, the original analysis may not be bettered by considering the numbers of unspecified visible and hidden elements in the given surface form as aspects). As is the case for FST-based rule systems that operate by FST composition and the KIMMO system, part (5) of Theorem 4.5.5 shows that the lexicon can be treated as just another constraint. Further analyses of the role of the lexicon in the NP-hardness of DP-DECODE may profit from techniques that characterize the sets encoded in constraints (see discussion in Section 4.2).

The latter sources of polynomial-intractability in both problems above are particularly intriguing because they suggest that what may be important to the computational complexity of constraint-based systems is not the length of the given form (as was suggested by the sources of polynomial-intractability $\{|u|, |\Sigma_s|\}$ and $\{|s|, |\Sigma_u|\}$ for KIM-ENCODE and KIM-DECODE, respectively) but rather the length of the longest context-size of any constraint that evaluates this form. This sheds an interesting light on the discussion of the sources of polynomial-intractability for the KIMMO system, in that it seems that what is important is not so much the number of harmony-like processes (as was suggested in [KC88]) but rather the extent in the representation over which such processes can operate, i.e., how "long-range" such processes are. This makes more urgent the derivation of a notion of context for FST, so that this conjecture can be formally tested against constraint-based systems like KIMMO that are based on FST rather than DFA. Though context-size does not seem so important in simplified segmental grammars (recall that SSG-ENCODE and SSG-DECODE are NP-hard when the context-size is 2), it would be interesting to know if it is important in more general FST-based rule systems such as those examined in Section 4.3.

As noted in Section 4.5.1, the problems considered here do not seem to correctly formalize Declarative Phonology; hence, one might argue, results derived for these problems are irrelevant and discussion of the implications of these results is a waste of time. Several such objections to the problems analyzed here are as follows:

1. The problems analyzed here cannot model arbitrarily complex autosegmental representations.

2. The problems analyzed here allow Declarative Phonology systems in which constraints conflict; moreover, this conflict is not mediated by the Elsewhere Condition (see Footnote 3).

3. The problems analyzed here allow Declarative Phonology systems that do not have solutions, i.e., systems of constraints that cannot be satisfied by any possible full form.

Consider each of these objections in turn.

1. The discussions in Section 2.2.2 and Chapter 3 stressed that the simplified representations used here are not intended to directly model arbitrarily complex autosegmental representations, but are rather broad and abstract characterizations of such representations. The results derived here are useful to the extent that the representations examined here are special cases of, and hence can be translated into, representations used in practice. If this is in fact the case, hardness results (and, to a lesser extent, algorithms) derived relative to the simplified representations considered here will automatically propagate to and hold relative to representations used in practice.

2. Some of the reductions given here for DP-Encode and DP-Decode show that the encoding and decoding problems associated with Declarative Phonology systems are NP-hard when instances are restricted such that there is no pairwise constraint conflict and the Elsewhere condition does not apply. This is the case with the reductions ultimately originating from Dominating set in part (10) of Theorem 4.5.4 and part (8) of Theorem 4.5.5. In the instances constructed by these reductions, each created constraint CDFA requires only that one of a particular set of symbols occurs *somewhere* in the requested form, and thus any pair of constraints can always avoid conflict by setting symbols in different parts of the requested form; moreover, as each constraint CDFA operates over the whole form, no constraint CDFA is more specific than any other and the Elsewhere Condition does not apply. While the reductions ultimately originating from Longest common subsequence sidestep the Elsewhere Condition for the same reason, they cannot in general avoid pairwise constraint conflict as the strings corresponding to two constraints may not share a common subsequence. Thus, while all proofs given here do not seem to hold in the case of Declarative Phonology systems that do not allow pairwise conflict or allow limited pairwise conflict resolved by the Elsewhere condition, enough proofs (namely, those ultimately originating from Dominating set) still work to establish the NP-hardness of and W-hardness relative to certain sets of aspects for such systems. Moreover, as the FPT algorithms given here work as described regardless of constraint conflict or the application of the Elsewhere Condition, all sources of polynomial-time intractability derived here also hold for these systems (modulo the representation-translations in (1)).

3. This is intuitively the most problematic of the objections raised. However, it is ultimately the least problematic, as it can be shown that for an NP-hard decision problem Π, if there is a polynomial-time algorithm that solves the subproblem Π' of Π defined such that each instance of Π' is guaranteed to have a solution then $P = NP$ [JoD85, page 291]. To see this, suppose such an algorithm exists and $p(n)$ is the polynomial bounding the running time of this algorithm. We can use such an algorithm to solve the general problem Π in the following manner: Given any instance

I of Π, run that algorithm on I and keep track of the number of steps used. If the instance has a solution, it will be found in polynomial time; if the instance does not have a solution, this will be detectable as soon as the algorithm executes $p(|I|) + 1$ steps and thus can be verified in polynomial time. Thus, this algorithm solves Π in polynomial time, which implies that $P = NP$. Note that this argument can just as easily be rephrased in terms of W-hardness, FPT algorithms, and partial collapses of the W-hierarchy. Hence, all hardness results derived relative to the problems analyzed here also hold for Declarative Phonology systems that are guaranteed to have solutions. Moreover, as the FPT algorithms given here work as described regardless of where or not the given Declarative Phonology system has a solution, all sources of polynomial-time intractability derived here also hold for these systems (modulo the representation-translations in (1)).

Thus, the results of the analyses done in this thesis are relevant to Declarative Phonology systems.

In Declarative Phonology systems, constraints can only add material to a given surface or lexical form to create a full form that is consistent with that given form; hence, though there can be insertion, there cannot be deletion. As DP-ENCODE and DP-DECODE are special cases of encoding and decoding problems in Declarative Phonology that allow insertion, all NP- and W-hardness results derived above still hold for these problems. To say anything more requires a more explicit model of how such insertion might take place in the simplified representation used here. Consider one such model in which hidden-visible symbol pairs may be inserted within or around given lexical or surface forms and that these given forms specify where such insertion can occur. In the case of lexical or surface form strings, this can be done by special symbols, and in the case of lexical or surface form DFA, this can be done by the addition of appropriate states and transitions. Using techniques described in Section 4.4.3, if insertions are allowed, certain W-hardness results hold in more restricted cases or relative to higher levels of the W-hierarchy. It is also possible that certain parameterized problems that were previously known to have FPT algorithms may be shown W-hard. The full extent of these changes will not be addressed here. For now, simply observe that the FPT algorithms based on DFA intersection will still work relative to the new given form DFA, and the FPT algorithms based on the graph-construction can be modified to accommodate insertion (*sketch of proof*: add additional columns of vertices before $VC(u_1)$, between $VC(u_i)$ and $VC(u_{i+1})$, and after $VC(u_{(|u|-c)+1})$ such that these new columns encode the contexts that surround and comprise arbitrary-length insertions).

Consider now what these results have to say about the various search problems associated with Declarative Phonology systems. First, note the following relationships between DP-ENCODE and DP-DECODE and their associated search problems:

- Any instance of DP-ENCODE can be solved by a single call to $\text{ENC}_{\mathbf{DP}}$; moreover, any instance of $\text{ENC}_{\mathbf{DP}}$ can be solved by the FPT algorithms given in the previous section for DP-ENCODE.

- Any instance of DP-Decode can be solved by a single call to $\text{DEC}_{\mathbf{DP}}$; moreover, any instance of $\text{DEC}_{\mathbf{DP}}$ can be solved by the FPT algorithms given in the previous section for DP-Decode.

The following relationship, however, is perhaps more surprising:

- Any instance of DP-Encode in which the lexical form contains a fully specified surface form can be solved by a single call to $\text{CHK}_{\mathbf{DP}}$; moreover, any instance of $\text{CHK}_{\mathbf{DP}}$ can be solved by slightly modified versions of the FPT algorithms given in the previous section for DP-Encode. Problem DP-Encode is NP-hard under the restriction in the former by the reductions given in Lemmas 4.5.1 and 4.5.2. In the case of the latter, the FPT algorithms must be modified to use the given surface form s to restrict the depth-first searches in the graph and intersection DFA constructions to ensure that derived full forms are consistent with both u and s.

Hence, modulo various conjectures, $\text{ENC}_{\mathbf{DP}}$, $\text{DEC}_{\mathbf{DP}}$, and $\text{CHK}_{\mathbf{DP}}$ do not have polynomial-time algorithms and have the same sources of polynomial-time intractability as their corresponding decision problems. This is very different from the situation with either simplified segmental grammars or the KIMMO system, in that both $\text{ENC}_{\mathbf{DP}}$ *and* $\text{CHK}_{\mathbf{DP}}$ are not solvable in polynomial time unless $P = NP$. However, oddly enough, this situation is the same as that for the FST-based rule systems examined in Section 4.3 and the automaton-based formulation of Optimality Theory examined in Section 4.6. As will be discussed in more detail in Section 4.7, one of the key factors in this pattern seems to be the simplified autosegmental representation used by Declarative Phonology and Optimality-Theoretic systems in this thesis.

For reasons discussed earlier in this section, the results derived above are relevant to implementations of Declarative Phonology systems to the extent that they suggest (albeit relative to a small set of aspects) what the sources of polynomial-time intractability in these systems may and may not be. These results are also useful, in conjunction with those derived for FST-based rule systems that operate by FST composition, the KIMMO system, and Optimality Theory, in characterizing the computational structure of phonological processing in general.

The above suggests many directions for future research. Several of the more intriguing directions are:

1. Fully characterize the effects of constraint conflict and the Elsewhere Condition on the computational complexity of DP-Encode and DP-Decode.
2. Characterize the computational complexity of DP-Encode and DP-Decode in terms of trade-offs between various aspects of the constraint CDFA, e.g., structure of the set of strings encoded by the CDFA.

As with FST-based rule systems that operate by FST composition and the KIMMO system, the research in (2) may profit by formulating the constraints implicit in the constraint CDFA and the lexicon DFA in logic along the lines suggested in Section 4.2.

Table 4.10: The Parameterized Complexity of the DP-ENCODE Problem.

Parameter	Alphabet Sizes ($|\Sigma_h|$,$|\Sigma_v|$)			
	(Unb,Unb)	(Unb,Prm)	(Prm,Unb)	(Prm,Prm)
–	NP-hard	$\not\in XP$	$\not\in XP$	$\not\in XP$
$|P|$	$W[t]$-hard	$W[t]$-hard	$W[t]$-hard	$W[t]$-hard
c	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$|u|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$|Q|$	$\not\in XP$	$\not\in XP$	$\not\in XP$	???
$|P|, c$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$|P|, |u|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$|P|, |Q|$	???	???	???	???
$c, |u|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$c, |Q|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$|u|, |Q|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$|P|, c, |u|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$|P|, c, |Q|$	FPT	FPT	FPT	FPT
$|P|, |u|, |Q|$	FPT	FPT	FPT	FPT
$c, |u|, |Q|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$|P|, c, |u|, |Q|$	FPT	FPT	FPT	FPT

Table 4.11: The Parameterized Complexity of the DP-DECODE Problem.

Parameter	Alphabet Sizes ($|\Sigma_h|$,$|\Sigma_v|$)			
	(Unb,Unb)	(Unb,Prm)	(Prm,Unb)	(Prm,Prm)
–	NP-hard	$\not\in XP$	$\not\in XP$	$\not\in XP$
$|P|$	$W[t]$-hard	$W[t]$-hard	$W[t]$-hard	$W[t]$-hard
c	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$|s|$	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$|Q|$	$\not\in XP$	$\not\in XP$	???	???
$|P|, c$	$W[1]$-hard	$W[1]$-hard	FPT	FPT
$|P|, |s|$	$W[1]$-hard	$W[1]$-hard	FPT	FPT
$|P|, |Q|$	???	???	???	???
$c, |s|$	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$c, |Q|$	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$|s|, |Q|$	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$|P|, c, |s|$	$W[1]$-hard	$W[1]$-hard	FPT	FPT
$|P|, c, |Q|$	FPT	FPT	FPT	FPT
$|P|, |s|, |Q|$	FPT	FPT	FPT	FPT
$c, |s|, |Q|$	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$|P|, c, |s|, |Q|$	FPT	FPT	FPT	FPT

4.6 Optimality Theory

4.6.1 Background

Until recently, purely constraint-based linguistic theories like Declarative Phonology have assumed that surface forms must satisfy all constraints, and hence that constraints cannot be violated. As noted in the previous section, this requires that systems be structured such that constraints do not conflict at all, or at least always have a conflict-free resolution. This often results in descriptions of linguistic phenomena in which individual constraints are complex and very specific to the phenomenon of interest [LP93, Sections 2.1 and 2.6]. An alternative approach to the structuring of constraint-based systems is taken by Optimality Theory [MP93, PS93]. Under Optimality Theory, the surface forms associated with a particular lexical form need not satisfy all given constraints — rather, these surface forms can violate some constraints as long as their associated full forms violate all of the given constraints in some minimal fashion over the space of all possible full forms that are associated with the given lexical form. This allows descriptions of phonological phenomena in which individual constraints are much simpler and more general; however, this simplicity is purchased at the expense of a much more complex manner of constraint-set evaluation.

An Optimality-Theoretic system $g = \langle Gen, C, O\rangle$ consists of a generator Gen that creates the sets of candidate full forms associated with given lexical forms, a set of constraints C, and a total ordering O on these constraints such that C and O collaborate within a function $Eval$ to select the optimal subset of the candidates generated by Gen. The resulting grammar is configured as follows (adapted from example (5) on page 4 of [MP93]):

$$\begin{array}{l} Gen(u) \rightarrow \{f_1, f_2, \ldots\} \\ Eval(C, O, \{f_1, f_2, \ldots\}) \rightarrow \{f_1, f_2, \ldots\}_{opt} \rightsquigarrow \{s_1, s_2, \ldots\} \end{array}$$

It is hypothesized (though by no means yet proven [Ell96]) that Gen and C are the same across all human languages and that only O and the lexicon are language-specific. Every generator Gen must satisfy the following properties:

1. **Freedom of Analysis**: An arbitrary amount of structure can be added to a given lexical form to create a candidate full form.
2. **Containment**: No element may be removed from the given lexical form. This form is thus always contained in every one of its associated candidate full forms.

Each constraint has an associated set of **marks** $\{m_1, m_2, \ldots m_k\}$, each of which denotes a particular kind of violation of that constraint. This set is totally ordered, i.e., $m_1 \succ m_2 \succ \ldots \succ m_k$, such that this order expresses the relative severity of each type of violation, e.g., if $m \succ m'$, a violation denoted by mark m is more severe than that denoted by m'. This order on marks is paralleled by the order O on the constraints, which essentially gives a decreasing order of importance of the constraints. This order O is expressed in term of a relation $\gg$ such that for constraints c and c', $c \gg c'$ means that c has priority over c'

Candidate Full Forms	Constraint Violations		
	c_1	c_2	c_3
f_1	$\{a_1, a_1, b_1\}$	$\{b_2\}$	ϕ
f_2	$\{b_1\}$	$\{a_2\}$	$\{a_3\}$
f_3	$\{b_1\}$	$\{b_2\}$	$\{b_3\}$

(a)

	Candidate Full Forms	Constraint Violations		
		c_1	c_2	c_3
	f_1	$[\ \underline{2}\ 1\ ]$	$[\ 0\ 1\ ]$	$[\ 0\ 0\ ]$
	f_2	$[\ 0\ 1\ ]$	$[\ \underline{1}\ 0\ ]$	$[\ 1\ 0\ ]$
$\Rightarrow$	f_3	$[\ 0\ 1\ ]$	$[\ \boxed{0}\ 1\ ]$	$[\ 0\ 1\ ]$

(b)

	Candidate Full Forms	Constraint Violations
		$[\ c_1\ c_2\ c_3\ ]$
	f_1	$[\ \underline{2}\ 1\ 0\ 1\ 0\ 0\ ]$
	f_2	$[\ 0\ 1\ \underline{1}\ 0\ 1\ 0\ ]$
$\Rightarrow$	f_3	$[\ 0\ 1\ \boxed{0}\ 1\ 0\ 1\ ]$

(c)

Figure 4.10: Evaluation of Candidate Full Forms in Optimality Theory. (a) Marks assigned to candidate full forms f_1, f_2, and f_3 by binary constraints c_1, c_2, and c_3 which have the associated mark-sets $\{a_1, b_1\}$, $\{a_2, b_2\}$, and $\{a_3, b_3\}$, respectively. (b) Evaluation of candidates when $c_1 \gg c_2 \gg c_3$ and $a_i \succ b_i$, $1 \leq i \leq 3$. Note that sets of marks from (a) have been replaced by the appropriately-ordered weight vectors. Optimal candidates are flagged by an arrow ($\Rightarrow$), mark-values that caused the elimination of candidates are underlined, e.g., $\underline{1}$, and mark-values that resulted in a candidate being chosen as optimal are framed by a box, e.g., $\boxed{0}$. (c) Evaluation of (b) relative to appropriately concatenated weight vectors. This shows more clearly the lexicographic optimality ordering on weight vectors.

in O. Each candidate full form has an associated set of marks composed of the union of the sets of marks assigned to that form by each constraint. The optimal full forms are those such that their sets of associated marks, when sorted first by the priority-order O assigned to the constraints and then by the mark order internal to each constraint, are lexicographically least. This manner of constraint evaluation and form selection is illustrated in Figures 4.10 and 4.11. Note that under this scheme, a constraint violation within a candidate full form no longer has an absolute and independently-assignable importance — rather, that violation's severity and subsequent impact on the optimality of that full form is relative not only on what other constraint violations occur within that form, but also on the nature of the constraint violations in all other candidate full forms.

	Candidate Full Forms	Constraint Violations		
		c_3	c_1	c_2
$\Rightarrow$	f_1	$[\ 0\ \ \boxed{0}\ ]$	$[\ 2\ \ 1\ ]$	$[\ 0\ \ 1\ ]$
	f_2	$[\ \underline{1}\ \ 0\ ]$	$[\ 0\ \ 1\ ]$	$[\ 1\ \ 0\ ]$
	f_3	$[\ 0\ \ \underline{1}\ ]$	$[\ 0\ \ 1\ ]$	$[\ 0\ \ 1\ ]$

(d)

	Candidate Full Forms	Constraint Violations		
		c_1	c_2	c_3
	f_1	$[\ \underline{2}\ \ 1\ ]$	$[\ 1\ \ 0\ ]$	$[\ 0\ \ 0\ ]$
$\Rightarrow$	f_2	$[\ 0\ \ 1\ ]$	$[\ \boxed{0}\ \ 1\ ]$	$[\ 1\ \ 0\ ]$
	f_3	$[\ 0\ \ 1\ ]$	$[\ \underline{1}\ \ 0\ ]$	$[\ 0\ \ 1\ ]$

(e)

Figure 4.11: Evaluation of Candidate Full Forms in Optimality Theory (Cont'd). (d) Evaluation of candidates when $c_3 \gg c_1 \gg c_2$ and $a_i \succ b_i$, $1 \leq i \leq 3$. (e) Evaluation of candidates when $c_1 \gg c_2 \gg c_3$, $a_i \succ b_i$ for $i \in \{1, 3\}$, and $b_2 \succ a_2$.

The computational work done on Optimality Theory to date has focused primarily on algorithms for generating optimal full forms from lexical forms [Ell94, FS97, Ham97, Kar98, Tes95, Wal96] and learning constraint orderings from full form examples [Tes96, Tes97a, TS96] (see also [Wal96, Section 1.2] for an excellent review of various implementations of Optimality Theory). The complexity of the encoding problem for two finite-state formulations of Optimality-Theoretic systems has been addressed in [Eis97a] and [War96a, War96b]. This section contains the first complete presentation of the partial results given (without proof) in [War96a, War96b], and extends the framework described in those references both to use a more complex representation incorporating hidden and visible components and to give a formulation and analysis of the decoding problem for Optimality-Theoretic systems.

Following [Eis97a, Ell94, FS97], the problems examined below formalize Optimality Theory in terms of finite-state automata which operate on strings. As in the formalization of Declarative Phonology in the previous section, these strings will be encodings of the simplified representation described in Section 2.2.2. In the Optimality-Theoretic systems considered below, *Gen* will be restricted to specifying underspecified elements in the given lexical form rather than adding arbitrary amounts of structure to this form; as such a *Gen* is implicit in the given lexical or surface form under the representation used in this thesis, *Gen* will not formally be a separate part of the problem instance. This not only simplifies the proofs in the next section, but also focuses the analysis on the simplest type of *Gen* that can alter structures within such systems and makes the results derived here compatible with those derived in this thesis for the other phonological theories, which are similarly restricted such that they cannot add structure to or delete structure from the given form. The formalization of individual components of an Optimality-Theoretic system are as follows:

- Lexical, surface, and full forms will be specified as either strings or DFA as described in Section 4.5.1.

- The constraints in C will be restricted to constraints that have only two kinds of marks (**binary marks**). In such a system, one of these marks is an expression of no violation, and can effectively be ignored. Such constraints will be specified as CDFA, and the number of violations of a constraint relative to a given candidate full form string will be equal to the number of context-size substrings of that candidate full form string that are not accepted by the DFA underlying the CDFA for that constraint.

- The lexicon $D \subseteq (\Sigma_v\Sigma_h)^+$ will be specified as a DFA $DFA(D) = \langle Q_D, \Sigma_h \cup \Sigma_v, \delta_D, s_D, F_D\rangle$ on $|Q_D|$ states which recognizes the language D^+.

The details of the operation of *Eval* relative to given forms and constraints in this formulation are somewhat involved, and will be given in Section 4.6.2.

Several finite-state formulations of Optimality Theory different from that given here have been proposed to date [Ell94, FS97, Kar98]; however, the only such formulation that has been the subject of a complexity-theoretic analysis is that given by Eisner in [Eis97a]. In this formulation, autosegmental structures, *Gen*, and constraints are represented as transition-weighted FSA, and the evaluation of a given form relative to *Eval* corresponds to the intersection of the DFA for the form, *Gen*, and the constraints and the subsequent derivation from the created intersection DFA of the lowest-total-weight paths from start to final states, each of which corresponds to an optimal full form that is consistent with the given form. Eisner has shown that the encoding problem for the version of this formulation in which constraints can only examine the features on all tiers at a single moment in time (and hence have context-size 1) is NP-hard [Eis97a, Section 4.2]. Modulo the use of DFA rather than CDFA for modeling constraints and the use of an unrestricted autosegmental representation rather than the simplified representation used here, the finite-state machinery invoked in Eisner's formulation is identical to that used in this section. Indeed, both formulations ultimately represent autosegmental structures as symbol strings in which each symbol encodes the set of features in all autosegments that are are linked to a particular slot on the timing tier (that is, each formulation essentially uses the second of the encodings of autosegmental structures into symbol strings given on page 40 in Section 2.2.3). Hence, it should be possible to adapt the reductions and algorithms given in the following section to produce a systematic parameterized analysis for Eisner's formulation of Optimality Theory. As the details of Eisner's restrictions on constraints are somewhat involved, such an analysis is left as a topic for future research. For the purposes of this thesis, the formulation defined in this section is more appropriate than that given in [Eis97a] because the formulation defined here is more useful for exploring the computational complexity of the mechanisms underlying Optimality Theory (see Section 2.2.2 and Chapter 3).

My analysis will focus on the following problems:

OT-ENCODE
Instance: An Optimality-Theoretic system $g = \langle Gen, C, O \rangle$ and a lexical form string u.
Question: Do the full form strings created by Gen from u that are optimal relative to C under O have no violations relative to any of the CDFA in C?

OT-DECODE
Instance: An Optimality-Theoretic system $g = \langle Gen, C, O \rangle$, a lexicon D, and a surface form string s.
Question: Is there a lexical form string u such that u is generated by D and there is a full form string f created by Gen from u that is optimal relative to C under O, has surface form string s, and has no violations relative to any of the CDFA in C?

From a computational perspective, the optimization inherent in the constraint-evaluation procedure underlying Optimality-Theoretic systems is very much like the that in various NP-hard optimization problems studied in computer science and operations research, e.g., route planning and task scheduling, and thus gives such systems an unmistakable aura of NP-hardness. From a linguistic perspective, giving constraints and constraint-evaluation procedures the ability to count violations also seems to grant (perhaps unnecessary) power [Ell95, FS97, Kar98]. What is perhaps most surprising about the results derived in the next section is that they suggest that the NP-hardness of the encoding and decoding problems associated with Optimality-Theoretic systems stems neither from the optimization in constraint-evaluation or from the ability to count violations but rather (as is the case with the other automaton-based phonological theories examined in this chapter) from the ability of Optimality-Theoretic systems to encode and solve constraint-satisfaction problems like BDFAI.

4.6.2 Analysis

The systematic parameterized analysis given in this section will focus on the following aspects:

- The number of CDFA in C ($|C|$).
- The maximum context-size of any CDFA in C (c).
- The length of the given lexical/ surface form ($|u|$ / $|s|$).
- The maximum number of states in any CDFA in C ($|Q|$).
- The sizes of the hidden and visible alphabets ($|\Sigma_h|$, $|\Sigma_v|$).

Consider the following reductions.

Lemma 4.6.1 *BDFAI* $\leq_m$ OT-ENCODE *such that* $|\Sigma_v| = 1$.

Proof: Given an instance $\langle A, \Sigma, k\rangle$ of BDFAI, construct the following instance $\langle C', c', O', \Sigma'_h, \Sigma_{hU}, \Sigma'_v, \Sigma_{vU}, u'\rangle$ of OT-ENCODE: Let Σ'_h, Σ_{hU}, Σ'_v, Σ_{vU}, u', and c' be as defined in Lemma 4.5.1, C' be constructed in the same manner as P' in Lemma 4.5.1, and O' be an arbitrary total order over C'. This construction can be done in time polynomial in the given instance of BDFAI. Note that as *Gen* is implicit in u, the only full forms considered by OT-ENCODE are those that are consistent with u. Moreover, as OT-ENCODE is only interested in full forms that do not violate any CDFA in C', OT-ENCODE is actually only interested in full forms that are accepted by all CDFA in C'. Hence, OT-ENCODE is equivalent to DP-ENCODE, and the reduction specified here works for the same reasons as given in Lemma 4.5.1.

Note that in the constructed instance of OT-ENCODE, $|C'| = |A|$, $c' = |u'| = 2k$, $|Q'| = |Q|$, $|\Sigma'_h| = |\Sigma| + 1$, and $|\Sigma'_v| = 1$. ∎

Lemma 4.6.2 *BDFAI* $\leq_m$ OT-ENCODE *such that* $|\Sigma_h| = 1$.

Proof: Given an instance $\langle A, \Sigma, k\rangle$ of BDFAI, construct the following instance $\langle C', c', O', \Sigma'_h, \Sigma_{hU}, \Sigma'_v, \Sigma_{vU}, u'\rangle$ of OT-ENCODE: Let Σ'_h, Σ_{hU}, Σ'_v, Σ_{vU}, u', and c' be as defined in Lemma 4.5.2, C' be constructed in the same manner as P' in Lemma 4.5.2, and O' be an arbitrary total order over C'. This construction can be done in time polynomial in the given instance of BDFAI. By the same reasoning as given above in Lemma 4.6.1, the reduction specified here works for the same reasons as given in Lemma 4.5.2.

Note that in the constructed instance of OT-ENCODE, $|C'| = |A|$, $c' = |u'| = 2k$, $|Q'| = |Q|$, $|\Sigma'_h| = 1$, and $|\Sigma'_v| = |\Sigma| + 1$. ∎

Lemma 4.6.3 *BDFAI* $\leq_m$ OT-DECODE *such that* $|\Sigma_v| = 1$.

Proof: Given an instance $\langle A, \Sigma, k\rangle$ of BDFAI, construct the following instance $\langle C', c', O', \Sigma'_h, \Sigma_{hU}, \Sigma'_v, \Sigma_{vU}, D', s'\rangle$ of OT-DECODE: Let Σ'_h, Σ_{hU}, Σ'_v, Σ_{vU}, D', s', and c' be as defined in Lemma 4.5.3, C' be constructed in the same manner as P' in Lemma 4.5.3, and O' be an arbitrary total order over C'. This construction can be done in time polynomial in the given instance of BDFAI. By the same reasoning as given above in Lemma 4.6.1, the reduction specified here works for the same reasons as given in Lemma 4.5.3.

Note that in the constructed instance of OT-DECODE, $|C'| = |A|$, $c' = |u'| = 2k$, $|Q'| = |Q|$, $|\Sigma'_h| = |\Sigma| + 1$, and $|\Sigma'_v| = 1$. ∎

The previous lemmas show that the reductions for OT-ENCODE and OT-DECODE are identical to those for DP-ENCODE and DP-DECODE in Section 4.5.2. Though the corresponding FPT algorithms will use approximately the same techniques, there are some major differences introduced by the need to be able to count constraint violations in Optimality Theory. In particular, the ubiquitous depth-first searches in directed graphs that characterized the FPT algorithms for Declarative Phonology will be replaced by single-source shortest-path computations in arc-weighted directed graphs in the corresponding FPT algorithms for Optimality Theory. This shortest-path formulation for Optimality Theory was proposed by Ellison in [Ell94] and has subsequently been used by Eisner [Eis97a]. As this approach underlies all FPT algorithms for Optimality Theory given in this thesis, it is worth looking at in more detail.

The shortest-path approach to solving problems associated with Optimality Theory was originally proposed in the context of finite-state implementations of Optimality Theory, and is based on the following reasoning:

1. **Marks assigned by a constraint can be represented within that constraint's associated DFA by weight vectors associated with that DFA's transitions**: Let $m_1, m_2, \ldots, m_k$ be the marks associated with a constraint, and let $v = [v_1, v_2, \ldots, v_k]$ be a weight vector of length k where v_i is the number of marks of type m_i, $1 \leq i \leq k$. Each transition in a constraint DFA can be visualized as assigning some set of marks to the representation being evaluated; let the vector corresponding to this set be the weight vector for that transition. These weight-vectors can be totally ordered lexicographically in decreasing order on mark index, e.g., for a weight vector based on the binary marks m_1 and m_2, this ordering has the form $[0,0] \prec [0,1] \prec [0.2] \prec \ldots \prec [1,0] \prec [1,1] \prec [1,2] \ldots \prec [2,0] \prec [2,1] \prec [2,2] \prec \ldots$. Let $\mathbf{0}^k$ represent the all-zero weight vector of length k. Note that this ordering expresses the relative optimality of various mark-assignments under Optimality Theory, with $\mathbf{0}^k$ being the best possible mark-assignment for any $k > 0$.

2. **The set of marks associated by a transition-weighted constraint DFA with a candidate full form is encoded in the vector-weight that is the sum of all weight vectors associated with transitions invoked in processing that candidate**: This is implicit in (1).

3. **Transition-weighted constraint DFA can be intersected in a manner that preserves a given constraint ordering**: Consider the following modification of the DFA intersection operation given in Section 2.2.3: Given a transition $\delta(q,a)$ for some state $q \in Q$ and symbol $a \in \Sigma$ in some transition-weighted DFA, let $w(\delta(q,a))$ be the vector-weight associated with that transition. Given two vector-weights $w_1 = [w_{11}, w_{12}, \ldots, w_{1k}]$ and $w_2 = [w_{21}, w_{22}, \ldots, w_{2l}]$, the concatenation of these vectors $[w_1 w_2]$ is the $k+l$-length vector $[w_{11}, w_{12}, \ldots, w_{1k}, w_{21}, w_{22}, \ldots, w_{2l}]$. Given two transition-weighted DFA $A_1 = \langle Q_1, \Sigma, \delta_1, s_1, F_1 \rangle$ and $A_2 = \langle Q_2, \Sigma, \delta_2, s_2, F_2 \rangle$, let the **priority weighted intersection of** A_1 **and** A_2 be defined by the transition-weighted DFA $A = \langle Q_1 \times Q_2, \Sigma, \delta, [s_1, s_2], F_1 \times F_2 \rangle$, where for all $q_1 \in Q_1$, $q_2 \in Q_2$, and $a \in \Sigma$, $\delta([q_1, q_2], a) = [\delta_1(q_1), \delta_2(q_2)]$ and $w(\delta([q_1, q_2], a)) = [w(\delta_1(q_1)) w(\delta_2(q_2))]$. By reasoning analogous to that given for DFA intersection in Section 2.2.3, this construction can be done in $O(|Q_1||Q_2|LT^2)$ time, where L is the sum of the lengths of the weight-vectors of A_1 and A_2, and $T = |\Sigma|$, and the intersection DFA that it produces has $|Q_1||Q_2|$ states and $|Q_1||Q_2|T$ transitions. This type of intersection operation is no longer symmetric as the transition-weights associated with the priority weighted intersection of A_1 and

A_2 are not the same as those associated with the priority weighted intersection of A_2 and A_1. However, that is exactly what is necessary here. As the weights produced under this operation assign priority to the first operand DFA under the lexicographic vector-weight ordering given in (1), one can combine constraint DFA and preserve a given constraint ordering by appropriately ordering the priority weighted intersection of those DFA.

4. **A single-source all-destination shortest-path algorithm can be applied to extract the optimal forms from the transition-weighted intersection DFA created in (3)**: Note that there is no vector weight a that can be added to another vector-weight b such that the resulting vector-weight precedes b in the total order given in (1); hence, the graph associated the transition diagram for the intersection DFA created in (3) do not have negative-weight cycles, i.e., a cycle such that the sum of the weights on the arcs in that cycle is negative and any path containing that cycle can lower its summed arc-weight by an arbitrary amount by repeated traversals around the cycle. By (2), the optimal forms encoded in the intersection DFA created in (3) correspond to paths of minimum weight (relative to the total order in (1)) from the start state to a final state. These minimum-weight paths can be computed from the graph associated with the transition diagram for this intersection DFA using a standard single-source shortest-path algorithm (see [CLR90, Chapter 25]); as there are no negative-weight cycles in this graph, Dijkstra's algorithm (modified such that weight inequalities in this algorithm, e.g., $w(e_{ab}) \leq w(e_{ac}) + w(e_{cb})$, are rephrased in terms of the total order in (1) on weight vectors) can be used. Given an arc-weighted directed graph $= (V, A, W)$, $W : A \mapsto \mathcal{N}$, Dijkstra's algorithm can be implemented such that it runs in $O(|V| \log |V| + |A|)$ time.[4]

In Ellison's original proposal, a given FSA encodes the set of candidate full forms associated by *Gen* with a particular lexical form, each constraint is specified as a transition-weighted i/o-deterministic FST that outputs the actual sequence of constraint-violation marks associated with a given input form, the transition-weighted intersection FSA of the *Gen*-output DFA and the constraint FST is computed using the priority weighted intersection operation described above to preserve the ordering on the constraints, and the arc-weighted graph associated with the transition diagram for this intersection FST is analyzed by a shortest-path algorithm to extract the optimal full forms. As noted by Eisner [Eis97a], the constraint FST can be simplified to FSA if the actual sequence of marks is not important; this simplification will be used in several of the FPT algorithms described in the following theorems. Note that the reasoning above also establishes how shortest-path computations can be used to extract optimal full forms from appropriately constructed arc-weighted directed graphs in general; this observation will be used in several of the other FPT algorithms given in these theorems.

[4] Recent work has shown that much faster algorithms are possible (see [Ram97] and references).

Theorem 4.6.4

1. OT-ENCODE *is* NP*-hard when* $|Q| = 2$ *and* $|\Sigma_v| = 1$.
2. OT-ENCODE *is* NP*-hard when* $|Q| = 2$ *and* $|\Sigma_h| = 1$.
3. OT-ENCODE *is* NP*-hard when* $|\Sigma_h| = 2$ *and* $|\Sigma_v| = 1$.
4. OT-ENCODE *is* NP*-hard when* $|\Sigma_h| = 1$ *and* $|\Sigma_v| = 2$.
5. $\langle c, |\Sigma_h|, |\Sigma_v|\rangle$-OT-ENCODE *is in* FPT.
6. $\langle |u|, |\Sigma_h|, |\Sigma_v|\rangle$-OT-ENCODE *is in* FPT.
7. $\langle |C|, c, |Q|\rangle$-OT-ENCODE *is in* FPT.
8. $\langle |C|, |u|, |Q|\rangle$-OT-ENCODE *is in* FPT.
9. $\langle |C|, c, |u|, |\Sigma_v|_1\rangle$-OT-ENCODE *and* $\langle |C|, c, |u|, |\Sigma_h|_1\rangle$-OT-ENCODE *are* $W[1]$*-hard.*
10. $\langle c, |u|, |Q|_2, |\Sigma_v|_1\rangle$-OT-ENCODE *and* $\langle c, |u|, |Q|_2, |\Sigma_h|_1\rangle$-OT-ENCODE *are* $W[2]$*-hard.*
11. $\langle |C|, |\Sigma_h|_2, |\Sigma_v|_1\rangle$-OT-ENCODE *is* $W[t]$*-hard for all* $t \geq 1$.
12. $\langle |Q|_2, |\Sigma_v|_1\rangle$-OT-ENCODE $\not\in XP$ *unless* $P = NP$.
13. $\langle |Q|_2, |\Sigma_h|_1\rangle$-OT-ENCODE $\not\in XP$ *unless* $P = NP$.
14. $\langle |\Sigma_h|_2, |\Sigma_v|_1\rangle$-OT-ENCODE $\not\in XP$ *unless* $P = NP$.
15. $\langle |\Sigma_h|_1, |\Sigma_v|_2\rangle$-OT-ENCODE $\not\in XP$ *unless* $P = NP$.

Proof:

Proof of (1): Follows from the NP-hardness of BDFAI when $|Q| = 2$ as established in part (1) of Theorem 4.2.4 and the reduction in Lemma 4.6.1 from BDFAI to OT-ENCODE in which $|Q'| = |Q|$ and $|\Sigma'_v| = 1$.

Proof of (2): Follows from the NP-hardness of BDFAI when $|Q| = 2$ as established in part (1) of Theorem 4.2.4 and the reduction in Lemma 4.6.2 from BDFAI to OT-ENCODE in which $|Q'| = |Q|$ and $|\Sigma'_h| = 1$.

Proof of (3): Follows from the NP-hardness of BDFAI when $|\Sigma| = 2$ as established in part (2) of Theorem 4.2.4 and the reduction in Lemma 4.6.1 from BDFAI to OT-ENCODE in which $|\Sigma'_h| = |\Sigma|$ and $|\Sigma'_v| = 1$.

Proof of (4): Follows from the NP-hardness of BDFAI when $|\Sigma| = 2$ as established in part (2) of Theorem 4.2.4 and the reduction in Lemma 4.6.2 from BDFAI to OT-ENCODE in which $|\Sigma'_h| = 1$ and $|\Sigma'_v| = |\Sigma|$.

Proof of (5): The following algorithm is a modification of the implicit CDFA intersection construction given in part (5) of Theorem 4.5.4 which takes into account the possibility of multiple violations of constraints and the evaluation of full forms that violate constraints under Optimality Theory. This will be done by appropriately weighting the arcs of the full form graph and looking for shortest paths in the resulting arc-weighted graph. As the algorithm description given below will draw heavily on various terms and quantities defined in part (5) of Theorem 4.5.4, the reader should perhaps review that notation before reading the rest of this proof.

The steps in this algorithm for OT-Encode are as follows:

1. Construct the full form graph for u by applying algorithm CFFG from Theorem 4.5.4.
2. Add the appropriate arc-weights to the full form graph constructed in (1) to keep track of the number of constraint violations in any path corresponding to a full form string.

   Each weight vector will be of length $|C|$ such that the ith position in this vector is the number of violations of the ith constraint in C under O. Given the division of the arcs of the full form graph into A_1 and A_2 as given in step 2 of algorithm CFFG in part (5) of Theorem 4.5.4, let the following weight vectors be assigned to the arcs in those sets:

   A_1: For all i and j, $1 \leq i \leq |VC(u_1)|$ and $1 \leq j \leq c-2$, assign the weight vector to arc $(s, vp_{i,1})$ that counts the number of constraint violations of the first symbol of $sv_{VC}(1, vert_{VC}(1,i))$ relative to all CDFA in C with context-size 1, assign the weight vector to arc $(vp_{i,j}, vp_{i,j+1})$ that counts, for each CDFA in C of context-size $c' \leq (j+1)$, the number of constraint violations of the c'-length suffix of the $j+1$-length prefix of $vs_{VC}(1, vert_{VC}(1,i))$, and assign the weight vector to arc $(vp_{i,(c-1)}, vert_{VC}(1,i))$ that counts, for each CDFA in C of context-size c', the number of constraint violations of the length-c' suffix of $vs_{VC}(1, vert_{VC}(1,i))$.

   A_2: For all i, v, and v' such that $1 \leq i \leq |u| - c$, $v \in VC(u_i)$, $v \in VC(u_{(i+1)})$, and $(v, v') \in A$, assign the weight vector to arc (v, v') that counts, for each CDFA in C of context-size c', the number of constraint violations of the length-c' suffix of $vs_{VC}(i+1, v')$.

   The somewhat counterintuitive stipulation that constraint violations must be counted relative to suffixes of strings is to ensure that the number of constraint violations relative to CDFA in C of context-size $c' \leq c$ are not overcounted, i.e., violations relative to the same substring of u of size c' are not counted multiple times.

3. Apply Dijkstra's algorithm (as modified to operate relative to weight vectors) to the arc-weighted graph constructed in (2) to determine the minimum-weight paths between s and all vertices in $VC(u_{(|u|-c)+1})$ in that graph. By the reasoning given earlier this section and the construction of the graph in (1) and (2) above, these minimum-weight paths will correspond to the optimal full form strings for u relative to C under O.

Note that max(VC, max(arc), $|V|$, and $|A|$ have the same values as they did in part (5) of Theorem 4.5.4:

- $\max(VC) = (|\Sigma_h||\Sigma_v|)^c$.
- $\max(arc) = \max(|\Sigma_h|, |\Sigma_v|)$.
- $|V| = \max(VC)|g| = (|\Sigma_h||\Sigma_v|)^c|u|$, where $|V|$ is the number of vertices in the full form graph constructed in (1).
- $|A| = \max(VC)\max(arc)|g| = (|\Sigma_h||\Sigma_v|)^c|u|\max(|\Sigma_h|, |\Sigma_v|)$, where $|A|$ is the number of arcs in the full form graph constructed in (1).

Consider now the running times of the individual steps in this algorithm:

- As algorithm CFFG runs in $O((\max(VC)|g|)^2\max(arc)c)$ time, step (1) can be done in $O(((|\Sigma_h||\Sigma_v|)^c|u|)^2\max(|\Sigma_h|, |\Sigma_v|)c) = O((|\Sigma_h||\Sigma_v|)^{2c}|u|^2c\max(|\Sigma_h|, |\Sigma_v|))$ time.
- As step (2) effectively examines each arc in the full form graph and can run potentially all CDFA in C on these strings at a cost of $O(c^2)$ time each, step (2) can be done in $O(|A||C|c^2) = O((|\Sigma_h||\Sigma_v|)^c|C|c^2|u|\max(\Sigma_h|, |\Sigma_v|))$ time.
- As Dijkstra's algorithm runs in $O(|V|\log|V| + |A|)$ time, step (3) can be done in

$$\begin{aligned}
&O(\max(VC)|g|\log\max(VC)|g| + \max(VC)\max(arc)|g|) \\
&\quad = O((\max(VC)|g|)^2 + \max(VC)\max(arc)|g|) \\
&\quad = O((\max(VC)|g|)^2\max(arc)) \\
&\quad = O(((|\Sigma_h||\Sigma_v|)^c|u|)^2\max(|\Sigma_h|, |\Sigma_v|)) \\
&\quad = O((|\Sigma_h||\Sigma_v|)^{2c}|u|^2\max(|\Sigma_h|, |\Sigma_v|))
\end{aligned}$$

  time.

Hence the algorithm for OT-ENCODE described above runs in $O((|\Sigma_h||\Sigma_v|)^{2c}|C|c^2|u|^2 \max(\Sigma_h|, |\Sigma_v|))$ time, which is fixed-parameter tractable relative to c, $|\Sigma_h|$, and $|\Sigma_v|$.

Proof of (6): As it is always the case that $c \leq |u|$, the result follows from (5) and Lemma 2.1.33.

Proof of (7): The following algorithm is a modification of the explicit CDFA intersection construction given in part (7) of Theorem 4.5.4 which takes into account the possibility of multiple violations of constraints and the evaluation of full forms that violate constraints under Optimality Theory. This will be done by modifying the construction of the systolic DFA such that the systolic DFA accept all full form strings instead of just those that have no constraint violations, appropriately weighting the transitions of the systolic DFA, and looking for shortest paths in the arc-weighted graph associated with the transition diagram for the intersection DFA. As the algorithm description given below will draw heavily on various terms and quantities defined in part (7) of Theorem 4.5.4, the reader should perhaps review that notation before reading the rest of this proof.

Consider the construction of systolic DFA that accept all full form strings. This can be done by modifying the transition-set δ_2 in the specification of the transition relation δ in the systolic DFA construction given in part (7) of Theorem 4.5.4 as follows:

δ'_2. Map states in Q_c to Q_c relative to δ, i.e., $\delta'_2 = \{(q, x, q') \mid q = [s, q_1, \ldots, q_c] \in Q_c$, $q' = [s, \delta(s, x)$ and $\delta(q_1, x), \ldots, \delta(q_{c-1}, x)] \in Q_c\}$.

This construction can still be done in $O(|Q|^c|\Sigma|)$ time and produces a systolic DFA with at most $|Q|^c$ states and at most $|Q|^c|\Sigma|$ transitions. Observe that systolic DFA created by this modified construction will accept all full form strings by a simplified version of the reasoning given in part (7) of Theorem 4.5.4.

Given this modified systolic DFA construction, the steps in the algorithm are as follows:

1. Construct the set C' of systolic DFA corresponding to the CDFA in C by applying the modified systolic DFA construction described above.

2. Add the appropriate transition-weights to each of the systolic DFA constructed in (1) to keep track of the number of constraint violations in any sequence of transitions corresponding to a computation on a full form string.

   Each weight vector will be of length 1. Given the division of the transitions in the transition relation for each systolic DFA into δ_1, δ_2, and δ_3 in the systolic DFA construction given in part (7) of Theorem 4.5.4, let the following weight-vectors be assigned to the transitions in each of these sets (keeping in mind that transition-set δ_2 has been replaced with the transition-set δ'_2 defined above):

   δ_1: Assign each transition in this set the weight 0.

   δ'_2: Assign each transition (q, x, q') such that $q = [s, q_1, \ldots, q_c] \in Q_c$ and $q' = [s, \delta(s, x), \delta(q_1, x), \ldots, \delta(q_{c-1}, x)] \in Q_c$ the weight 0 if $\delta(q_c, x) \not\in F$ and the weight 1 if $\delta(q_c, x) \in F$.

   δ_3: Assign each transition in this set the weight $(|u| - c) + 2$.

   Observe that the weights assigned to δ'_2 indicate when a substring of length c of the given full form string has or has been accepted by the CDFA associated with the systolic DFA, and that the weights assigned to the transitions in δ_3 ensure that no given string that does not correspond to a full form string can have a summed number of constraint violations that is less than the summed number of constraint violations for any full form string.

3. Construct the lexical form DFA corresponding to the given lexical form string u and set the weight of each transition in this DFA to $\mathbf{0}^1$.

4. Intersect the transition-weighted systolic DFA in C' and the transition-weighted lexical form DFA constructed in steps (2) and (3) above using the priority weighted intersection operation described earlier in this section to preserve the ordering O of the constraint CDFA.

5. Apply Dijkstra's algorithm (as modified to operate relative to weight vectors) to the arc-weighted graph associated with the transition diagram for the transition-weighted intersection DFA constructed in (4) to determine the minimum-weight paths between the vertex corresponding to the start state and all vertices corresponding to final states. By the reasoning given earlier this section and the construction of the intersection DFA, these minimum-weight paths will correspond to the optimal full form strings for u relative to C under O.

Consider the time complexity of each step in turn.

- As the systolic DFA for each CDFA in C can be created in $O(|Q|^c(|\Sigma_h| + |\Sigma_v|))$ time, step (1) runs in $O(|Q|^c|C|(|\Sigma_h| + |\Sigma_v|))$ time.
- As step (2) operates on each transition in each systolic DFA in C' and are there $|C|$ such systolic DFA, step (2) runs in $O(|Q|^c|C|(|\Sigma_h| + |\Sigma_v|))$ time.
- As the lexical form DFA corresponding to a given lexical form string u consists of $|u|+1$ states linked by $|u|$ transitions, step (3) runs in $O(|u|)$ time.
- As each systolic DFA has at most $|Q|^c$ states and the given lexical form DFA has $|u| + 1 \leq 2|u|$ states, step (4) can be done in $O(|Q|^{c|C|}|u||C|(|\Sigma_h| + |\Sigma_v|)^2)$ time by the time complexity of pairwise priority weighted intersection given earlier in this section and reasoning analogous to that for DFA intersection in Section 2.2.3 which derives the time complexity of k-wise intersection from that for pairwise intersection. Note the addition of the $|C|$ term to account for the effect of the length of the weight vectors on the priority weighted intersection operation.
- As the graph associated with the transition diagram for the intersection DFA constructed in (4) has $|V| \leq (|Q|^c)^{|C|}(|u| + 1) \leq 2|Q|^{c|C|}|u|$ vertices and $|A| \leq (|Q|^c)^{|C|}(|u| + 1)(|\Sigma_h| + |\Sigma_v|)2|Q|^{c|C|}|u|(|\Sigma_h| + |\Sigma_v|)$ arcs and Dijkstra's algorithm runs in $O(|V| \log |V| + |A|)$ time, step (5) can be done in

$$\begin{aligned} O(|Q|^{|C|c}|u| \log |Q|^{|C|c}|u| &+ |Q|^{|C|c}|u|(|\Sigma_h| + |\Sigma_v|)) \\ &= O((|Q|^{|C|c}|u|)^2 + |Q|^{|C|c}|u|(|\Sigma_h| + |\Sigma_v|)) \\ &= O(|Q|^{2|C|c}|u|^2(|\Sigma_h| + |\Sigma_v|)) \end{aligned}$$

  time.

Given the above, the algorithm thus runs in $O(|Q|^{2|C|c}|u|^2|C|(|\Sigma_h| + |\Sigma_v|)^2)$ time, which is fixed-parameter tractable relative to $|C|$, c, and $|Q|$.

Proof of (8): As it is always the case that $c \leq |u|$, the result follows from (7) and Lemma 2.1.33.

Proofs of (9 – 11): The first parts of (9) and (10) and all of (11) follow from the W-hardness results for BDFAI established in parts (5 – 7) of Theorem 4.2.4, the reduction in Lemma 4.6.1 from BDFAI to OT-ENCODE in which $|C'| = |A|$, $c' = |u'| = 2k$, $|Q'| = |Q|$, $|\Sigma'_h| = |\Sigma| + 1$, and $|\Sigma'_v| = 1$, and Lemma 2.1.25. The second parts of (9) and (10) follow from the W-hardness results for BDFAI established in parts (5) and (6) of Theorem 4.2.4, the reduction in Lemma 4.6.2 from BDFAI to OT-ENCODE in which $|C'| = |A|$, $c' = |u'| = 2k$, $|Q'| = |Q|$, $|\Sigma'_h| = 1$, and $|\Sigma'_v| = |\Sigma| + 1$, and Lemma 2.1.25.

Proofs of (12 – 15): Follow from (1 – 4) and Lemma 2.1.35. ∎

Theorem 4.6.5

1. OT-DECODE *is* NP*-hard when* $|Q| = 2$ *and* $|\Sigma_v| = 1$.
2. OT-DECODE *is* NP*-hard when* $|\Sigma_h| = 2$ *and* $|\Sigma_v| = 1$.
3. $\langle c, |\Sigma_h| \rangle$-OT-DECODE *is in* FPT.
4. $\langle |s|, |\Sigma_h| \rangle$-OT-DECODE *is in* FPT.
5. $\langle |C|, c, |Q| \rangle$-OT-DECODE *is in* FPT.
6. $\langle |C|, |s|, |Q| \rangle$-OT-DECODE *is in* FPT.
7. $\langle |C|, c, |s|, |\Sigma_v|_1 \rangle$-OT-DECODE *is* $W[1]$*-hard.*
8. $\langle c, |s|, |Q|_2, |\Sigma_v|_1 \rangle$-OT-DECODE *is* $W[2]$*-hard.*
9. $\langle |C|, |\Sigma_h|_2, |\Sigma_v|_1 \rangle$-OT-DECODE *is* $W[t]$*-hard for all* $t \geq 1$.
10. $\langle |Q|_2, |\Sigma_v|_1 \rangle$-OT-DECODE $\notin XP$ *unless* $P = NP$.
11. $\langle |\Sigma_h|_2, |\Sigma_v|_1 \rangle$-OT-DECODE $\notin XP$ *unless* $P = NP$.

Proof:

Proof of (1): Follows from the NP-hardness of BDFAI when $|Q| = 2$ as established in part (1) of Theorem 4.2.4 and the reduction in Lemma 4.6.3 from BDFAI to OT-DECODE in which $|Q| = |Q'|$ and $|\Sigma'_v| = 1$.

Proof of (2): Follows from the NP-hardness of BDFAI when $|\Sigma| = 2$ as established in part (2) of Theorem 4.2.4 and the reduction in Lemma 4.6.3 from BDFAI to OT-DECODE in which $|\Sigma'_h| = |\Sigma|$ and $|\Sigma'_v| = 1$.

Proof of (3): The following algorithm is a modification of the construction given in part (5) of Theorem 4.5.4 which takes into account the constraints on full forms associated with the given surface form string s that are imposed by the lexicon. This will be done (as it was in part (3) of Theorem 4.5.5) by converting the full form graph into a DFA and intersecting that DFA with the lexicon DFA before performing the shortest-path computations. As the algorithm description given below will draw heavily on various terms and quantities defined in part (5) of Theorems 4.5.4 and 4.6.4, the reader should perhaps review that notation before reading the rest of this proof.

The steps in this algorithm for OT-DECODE are as follows:

1. Construct the full form graph for s by applying algorithm CFFG from Theorem 4.5.4.
2. Add the arc-weights described in part (5) of Theorem 4.6.4 to the full form graph constructed in (1) to keep track of the number of constraint violations in any path corresponding to a full form string.
3. Interpret the arc-weighted full form graph constructed in (2) as the transition diagram of a transition-weighted DFA $A = \langle Q, (\Sigma_h - \Sigma_{hU}) \cup (\Sigma_v - \Sigma_{vU}), \delta, s, F \rangle$ in which $Q = V$, δ is constructed from the graph and its labels, the start state is vertex s in the graph, and the final states are the vertices in $VC(s_{(|s|-c)+1})$.
4. Assign the weight-vector $\mathbf{0}^1$ to each transition in the lexicon DFA $DFA_{FF}(D)$
5. Intersect the DFA constructed in (3) with the lexicon DFA $DFA_{FF}(D)$ by applying the priority weighted intersection operation described earlier in this section.
6. Apply Dijkstra's algorithm (as modified to operate relative to weight vectors) to the arc-weighted graph associated with the transition diagram for the transition-weighted DFA constructed in (4) to determine the minimum-weight paths between the vertices corresponding to the start and final states in the DFA for that graph. By the reasoning given earlier this section and the construction of the DFA in (1) – (4) above, these minimum-weight paths will correspond to the optimal full form strings for s relative to C under O such that the lexical form strings associated with these full form strings are generated by D.

Note that max(VC, max(arc), $|V|$, and $|A|$ have the same values as they did in part (5) of Theorem 4.5.5:

- $\max(VC) = |\Sigma_h|^c$.
- $\max(arc) = |\Sigma_h|$.
- $|V| = \max(VC)|g| = |\Sigma_h|^c|s|$, where $|V|$ is the number of vertices in the full form graph constructed in (1) (and hence the number of states in the DFA constructed in (3)).
- $|A| = \max(VC)\max(arc)|g| = |\Sigma_h|^c|\Sigma_h||s| = |\Sigma_h|^{c+1}|s|$, where $|A|$ is the number of arcs in the full form graph constructed in (1) (and hence the number of transitions in the DFA constructed in (3)).

Consider now the running times of the individual steps in this algorithm:

- As algorithm CFFG runs in $O((\max(VC)|g|)^2 \max(arc)c)$ time, step (1) can be done in $O((|\Sigma_h|^c|s|)^2|\Sigma_h|c) = O(|\Sigma_h|^{2c+1}|s|^2c)$ time.
- As step (2) effectively examines each arc in the full form graph and can run potentially all CDFA in C on these strings at a cost of $O(c^2)$ time each, step (2) can be done in $O(|A||C|c^2) = O(|\Sigma_h|^{c+1}|C|c^2|s|)$ time.
- As step (3) effectively examines each arc and vertex in the full form graph, step (3) can be done in $O(|V| + |A|) = O(|\Sigma_h|^c|s| + |\Sigma_h|^{c+1}|s|) = O(|\Sigma_h|^{c+1}|s|)$ time.
- As step (4) effectively examines each transition in the lexicon DFA, step (3) can be done in $O(|Q_D|(|\Sigma_h| + |\Sigma_v|))$ time.
- As the lexicon DFA $DFA_{FF}(D)$ has $|Q_D|$ states and the DFA created in step (3) has $|V|$ states, step (5) can be done in $O(|V||Q_D|(|\Sigma_h| + |\Sigma_v|)|\Sigma_h|) = O(|\Sigma_h|^c|s||Q_D|$ $(|\Sigma_h| + |\Sigma_v|)^2)$ time by the time complexity of pairwise priority weighted intersection given earlier in this section and reasoning analogous to that for DFA intersection in Section 2.2.3 which derives the time complexity of k-wise intersection from that for pairwise intersection. Note that though each state in the lexicon DFA is the source of at most $(|\Sigma_h| + |\Sigma_v|)$ transitions, each state in the DFA created in step (3) is the source of at most $|\Sigma_h|$ transitions; hence, each vertex in the graph associated with the transition diagram of the intersection DFA of these two DFA can be the source of at most $|\Sigma_h|$ arcs. Thus, the graph associated with the transition diagram has $|V'| \leq |V||Q_D| = \max(VC)|g||Q_D| = |\Sigma_h|^c|s||Q_D|$ vertices and $|A'| \leq |V'||\Sigma_h| = |\Sigma_h|^c|s||Q_D||\Sigma_h| = |\Sigma_h|^{c+1}|s||Q_D|$ arcs.
- As Dijkstra's algorithm runs in $O(|V|\log|V| + |A|)$ time, step (6) can be done in $O(|V'|\log|V'| + |A'|) = O(\max(VC)|g|\log\max(VC)|g| + \max(VC)\max(arc)|g|)$ $= O((\max(VC)|g|)^2 + \max(VC)\max(arc)|g|) = O((\max(VC)|g|)^2\max(arc)) =$ $O((|\Sigma_h|^c|s|)^2|\Sigma_h|) = O(|\Sigma_h|^{2c+1}|s|^2)$ time.

Hence the algorithm for OT-DECODE described above runs in $O(|\Sigma_h|^{2c+1}|C|c^2|s|^2|Q_D|$ $(\Sigma_h| + |\Sigma_v|)^2)$ time, which is fixed-parameter tractable relative to c and and $|\Sigma_h|$.

Proof of (4): As it is always the case that $c \leq |s|$, the result follows from (3) and Lemma 2.1.33.

Proof of (5): The required algorithm makes the following modifications to the algorithm given in part (7) of Theorem 4.6.4:

- Change all steps to deal with the surface form string s instead of the lexical form string u.
- Insert a step $(3')$ between steps (3) and (4) that sets the weight of each transition in the lexicon DFA to $\mathbf{0}^1$.

- Add the transition-weighted lexicon DFA created in step ($3'$) to the set of DFA that are intersected in step (4).

These changes have the following effects on the running time of the algorithm:

- As step ($3'$) must examine all transitions in the lexicon DFA, step ($3'$) runs in $O(|Q_D|(|\Sigma_h| + |\Sigma_v|))$ time.
- As each systolic DFA has at most $|Q|^c$ states, the given lexical form DFA has $|s| + 1 \leq 2|s|$ states, and the lexicon DFA has $|Q_D|$ states, step (4) is now done in $O(|Q|^{c|C|}|s||Q_D||C|(|\Sigma_h| + |\Sigma_v|)^2)$ time.
- As the graph associated with the transition diagram for the intersection DFA constructed in (4) has $|V| \leq (|Q|^c)^{|C|}(|s| + 1)|Q_D| \leq 2|Q|^{c|C|}|s||Q_D|$ vertices and $|A| \leq (|Q|^c)^{|C|}(|s| + 1)|Q_D|(|\Sigma_h| + |\Sigma_v|) \leq 2|Q|^{c|C|}|s||Q_D|(|\Sigma_h| + |\Sigma_v|)$ arcs and Dijkstra's algorithm runs in $O(|V|\log|V| + |A|)$ time, step (5) is now done in $O(|Q|^{|C|c}|s||Q_D|\log|Q|^{|C|c}|s||Q_D| + |Q|^{|C|c}|s||Q_D|(|\Sigma_h| + |\Sigma_v|)) = O(|Q|^{2|C|c}|s|^2|Q_D|^2 (|\Sigma_h| + |\Sigma_v|))$ time.

Given the above and the running times for steps (1) – (3) as originally derived in part (5) of Theorem 4.6.4, the algorithm thus runs in $O(|Q|^{2|C|c}|C||s|^2|Q_D|(|\Sigma_h| + |\Sigma_v|)^2)$ time, which is fixed-parameter tractable relative to $|C|$, c, and $|Q|$.

Proof of (6): As it is always the case that $c \leq |s|$, the result follows from (5) and Lemma 2.1.33.

Proofs of (7 – 9): Follow from the W-hardness results for BDFAI established in parts (5 – 7) of Theorem 4.2.4, the reduction in Lemma 4.6.3 from BDFAI to OT-DECODE in which $|C'| = |A|$, $|s'| = c' = 2k$, $|Q'| = |Q|$, $|\Sigma'_h| = 1$, and $|\Sigma'_v| = |\Sigma| + 1$, and Lemma 2.1.25.

Proofs of (10) and (11): Follow from (1) and (2) and Lemma 2.1.35. ∎

4.6.3 Implications

All parameterized complexity results for problems OT-ENCODE and OT-DECODE that are either stated or implicit in the lemmas and theorems given in the previous section are shown in Tables 4.12 and 4.13. Consider the implications of these results for each problem in turn:

- OT-ENCODE: The sources of polynomial-time intractability are $\{|P|, c, |Q|\}$ and $\{c, |\Sigma_h|, |\Sigma_v|\}$. The mechanisms associated with these sources are the constraint set (specifically the intersection DFA formed from the systolic DFA associated with all constraint CDFA) and the set of possible context-size strings. All of the aspects in these sources are defining properties of an Optimality-Theoretic system, and hence none of them can be eliminated to reduce the complexity of the problem. However, by the same reasoning as given in Section 4.5.3 for problem DP-ENCODE, the FPT algorithm underlying the latter source should efficient in practice and may run much faster than the original analysis suggests.

- OT-DECODE: The sources of polynomial-time intractability are $\{|C|, c, |Q|\}$ and $\{c, |\Sigma_h|\}$. The mechanisms associated with these sources are the constraint set (specifically the intersection DFA formed from the systolic DFA associated with all constraint CDFA) and the set of possible context-size strings with respect to a fully-specified visible component. As in OT-ENCODE, all of the aspects in these sources are defining properties of an Optimality-Theoretic system, and hence none of them can be eliminated to reduce the complexity of the problem; however, for the same reasons given above, the FPT algorithm underlying the latter source should be efficient in practice (though, as the whole hidden component is unspecified and the whole visible component is specified in every given surface form, the original analysis may not be bettered by considering the numbers of unspecified visible and hidden elements in the given surface form as aspects). As is the case for the other automaton-based phonological theories examined in this chapter, part (5) of Theorem 4.6.5 shows that the lexicon can be treated as just another constraint. Further analyses of the role of the lexicon in the polynomial-time intractability of OT-DECODE may profit from techniques that characterize the sets encoded in constraints (see discussion in Section 4.2).

Under the simplified autosegmental representation used here, the relationship between Declarative Phonology and Optimality Theory is very close — the proofs of NP- and W-hardness are essentially identical as are the algorithms modulo the substitution of priority weighted DFA intersection for DFA intersection and shortest path computation for depth-first search. This is so because Declarative Phonology is essentially a special case of Optimality Theory in which optimal full forms violate none of the given constraints, and are hence optimal under arbitrary constraint orderings. When the results derived in the previous section are compared with those derived in Sections 4.3.2, 4.4.2, and 4.5.2 for FST-based rule systems, the KIMMO system and Declarative Phonology, it becomes obvious that the hardness proofs and the FPT algorithms are remarkably similar in all four theories and seem oblivious to the differences in the formulations of these theories. As will be discussed further in Section 4.7, this suggests some interesting computational characteristics that may be common to all theories of phonological processing.

There are three immediate consequences of the results derived above:

1. Unless $FPT = W[2]$, there is no general algorithm for OT-ENCODE whose non-polynomial time-complexity is purely a function of the maximum constraint context-size c, cf. the polynomial-time algorithm given in [Tes95] for generating optimal full forms from given lexical forms relative to a restricted type of representation when $c = 3$. This suggests that unreasonable computational power may be encoded in constraints that have unbounded-size contexts, e.g. ALIGN, and that the use of such constraints may have to be severely curtailed to give efficient

implementations of Optimality-Theoretic systems (possibly along the lines suggested in [Eis97a, Eis97b]).

2. Unless $P = NP$, there is no general polynomial-time algorithm for OT-ENCODE when the set of candidate full form strings associated with a given lexical form string is restricted to being a regular language generated by some FSA, cf. [Ell94, Tes95].

3. Unless $P = NP$, there is no general polynomial-time algorithm for OT-ENCODE when no insertions or deletions are allowed in the lexical form, cf. the allegedly efficient algorithm given in [Ham97] for generating optimal syllabifications of given unsyllabified lexical forms when no insertions or deletions are allowed.

Note that these results do not preclude polynomial-time algorithms for other restricted cases of encoding problems such as those considered by Tesar and Hammond, but it does make the existence of such algorithms for general Optimality-Theoretic systems unlikely.

Eisner's NP-hardness result [Eis97a] implies the second consequence listed above and may even be seen as building a very strong case for the first consequence when $c = 1$. Technical merits aside, Eisner's work is also of interest for the manner in which his approach to using computational complexity to analyze linguistic theories (which seems to be typical of other such work done to date) differs from that used in this thesis. As such, his work provides examples for various points made in Chapter 3 of this thesis.

The most obvious difference is in the way in which Eisner interprets NP-hardness results. In particular, his NP-hardness reduction creates instances of Optimality-Theoretic systems whose associated DFA are such that the number of states in these DFA is lower-bounded by an exponential function in the number of tiers in these instances. Eisner concludes from this that his problem is " ... NP-hard on the number of tiers ... " [Eis97a, page 6]. It is not clear exactly what Eisner means by this. He does not establish that the number of tiers is crucial to the NP-hardness of his formulation of the encoding problem because there may be other reductions that establish NP-hardness without requiring an unbounded number of tiers. Neither does he establish that the number of tiers is responsible for this NP-hardness in the sense of having an associated algorithm whose non-polynomial time complexity time is purely a function of the number of tiers as he only shows that the size of the DFA (let alone the time required to construct it) is *lower* bounded (instead of upper bounded) relative to an exponential function in the number of tiers. In any case, Eisner's comment is a classic example of how NP-hardness results can be misinterpreted, and should be viewed as yet another cautionary tale which implicitly advocates the use of both the techniques and conceptual framework of parameterized complexity analysis.

A much more subtle and meaningful difference involves the style in which computational analyses of linguistic theories is carried out. The approach underlying [Eis97a, Eis97b] is to start with a restricted version of Optimality Theory and establish the descriptive adequacy and computational properties of this restricted formalism before (if ever) considering more general versions of the theory. This is diametrically opposed to the approach advocated in this thesis which starts with a very general and abstract version of a phonological theory such as Optimality Theory and establishes the computational properties of the set of possible restricted versions of this theory relative to a given set of restrictions of interest before (if ever) considering these restricted versions and their descriptive adequacy in detail. Informally, Eisner's approach is "bottom up" (in that work starts on the specific and progresses to the general) and systematic parameterized complexity analysis is "top down" (in that work starts on the general and progresses to the specific). Both approaches have their advantages and disadvantages. The bottom up approach has the advantage of allowing immediate detailed investigation of a particular formulation of a theory (at the expense of possibly missing more appropriate formulations in the rush to get detailed investigation started); the top-down approach has the advantage of first establishing the spectrum of available formulations and their computational properties to allow the choice of most appropriate formulation before detailed investigation begins (at the expense of possibly wasting a lot of time and energy ruling out inappropriate formulations). Though the latter approach is advocated in this thesis on the grounds that it seems to be a more systematic way of deriving good linguistic theories, the choice of which approach to use when analyzing linguistic theories in practice is ultimately a function of an investigator's resources and goals.

In the matter of insertions and deletions, Optimality Theory is like Declarative Phonology in that, ultimately, there is no deletion; material can only be added to a given lexical and surface form to create a full form that is consistent with that given form. As OT-ENCODE and OT-DECODE are special cases of encoding and decoding problems in Optimality Theory that allow insertion, all NP- and W-hardness results derived above still hold for these problems. To say anything more requires a more explicit model of how such insertion might take place in the simplified representation used here. Consider the model given for Declarative Phonology in Section 4.5.3, in which hidden-visible symbol pairs may be inserted within or around given lexical or surface forms and these given forms specify where such insertion can occur. Using techniques described in Section 4.4.3, if insertions are allowed, certain W-hardness results hold in more restricted cases or relative to higher levels of the W-hierarchy. It is also possible that certain parameterized problems that were previously known to have FPT algorithms may be shown W-hard. The full extent of these changes will not be addressed here. For now, simply observe that the FPT algorithms will continue to work by the reasoning given in Section 4.5.3.

Consider now what these results have to say about the various search problems associated with Optimality Theory systems. First, note the following relationships between OT-ENCODE and OT-DECODE and their associated search problems:

- Any instance of OT-ENCODE can be solved by a single call to $\text{ENC}_{\mathbf{OT}}$; moreover, any instance of $\text{ENC}_{\mathbf{OT}}$ can be solved by the FPT algorithms given in the previous section for OT-ENCODE.

- Any instance of OT-DECODE can be solved by a single call to $\text{DEC}_{\mathbf{OT}}$; moreover, any instance of $\text{DEC}_{\mathbf{OT}}$ can be solved by the FPT algorithms given in the previous section for OT-DECODE.

- Any instance of OT-ENCODE in which the lexical form contains a fully specified surface form can be solved by a single call to $\text{CHK}_{\mathbf{OT}}$; moreover, any instance of $\text{CHK}_{\mathbf{OT}}$ can be solved by the FPT algorithms given in the previous section for OT-ENCODE. Problem OT-ENCODE is NP-hard under the restriction in the former by the reductions given in Lemmas 4.6.1 and 4.6.2. Note that in the case of the latter, the FPT algorithms must use the given surface form s to restrict the shortest-path computations in the graph and intersection DFA constructions so that derived full forms are consistent with both u and s.

Hence, modulo various conjectures, $\text{ENC}_{\mathbf{OT}}$, $\text{DEC}_{\mathbf{OT}}$, and $\text{CHK}_{\mathbf{OT}}$ do not have polynomial-time algorithms and have the same sources of polynomial-time intractability as their corresponding decision problems. This is the same situation as with FST-based rule systems that operate by FST composition and Declarative Phonology; possible reasons for why this is so will be discussed in Section 4.7.

The results derived above are relevant to implementations of Optimality-Theoretic systems to the extent that they suggest (albeit relative to a small set of aspects) what the sources of polynomial-time intractability in these systems may and may not be. These results also have implications for the computational complexity of learning constraint-rankings in Optimality-Theoretic systems from examples. Polynomial-time algorithms are known for the case where the examples are optimal full forms [TS96]. An open question concerns the more realistic case where the examples are surface forms [Tes96, Tes97a, Tes97b]. This situation is modeled by the following search problem:

- $\text{LRN}_{\mathbf{OT}}(g, u, s)$: Given a surface form s and an Optimality-Theoretic system g without an ordering C on the set of constraints C, return any ordering O of C such that there is a surface form f generated by Gen from some lexical form u that is optimal relative to C under O and has s as its surface form, and $\perp$ otherwise.

Note that in the instances of OT-ENCODE created in the reduction in Lemma 4.6.1, $u = s$, u is the only possible lexical form of length $|s|$ (and hence the only possible lexical form for s), and either all orderings of C work or none do; as any of these instances can be solved by a single call to $\text{LRN}_{\mathbf{OT}}$, $\text{LRN}_{\mathbf{OT}}$ does not have a polynomial-time algorithm and has the same sources of polynomial-time intractability as OT-ENCODE (modulo various conjectures).

The above suggests many directions for future research. Several of the more intriguing directions are:

1. Characterize the computational complexity of OT-ENCODE and OT-DECODE in terms of trade-offs between various aspects of the individual CDFA being intersected, e.g., structure of the sets of strings encoded by the CDFA.
2. Redo the analyses given in this section with an additional aspect denoting the maximum number of violations any constraint may have on a candidate full form string.

3. Formalize and investigate a model of Optimality Theory with a more realistic finite-state model of *Gen*, e.g., a FST that relates lexical and full forms [FS97, Kar98].

As with the other automaton-based phonological theories examined in this thesis, the research in (1) may benefit from formulating the constraints implicit in the various types of finite-state automata used in the formulation of Optimality Theory above in logic along the lines suggested in Section 4.2. The research in (2) would complement that given in [FS97, Kar98] which considers the generative capacity of FST-based implementations of Optimality Theory that allow only a fixed number of constraint violations. Finally, the research in (3) would be part of a larger investigation of FST formulations of Optimality Theory [FS97, Kar98].

Table 4.12: The Parameterized Complexity of the OT-Encode Problem.

Parameter	Alphabet Sizes ($\|\Sigma_h\|$,$\|\Sigma_v\|$)			
	(Unb,Unb)	(Unb,Prm)	(Prm,Unb)	(Prm,Prm)
–	NP-hard	$\notin XP$	$\notin XP$	$\notin XP$
$\|C\|$	$W[t]$-hard	$W[t]$-hard	$W[t]$-hard	$W[t]$-hard
c	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$\|u\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$\|Q\|$	$\notin XP$	$\notin XP$	$\notin XP$	???
$\|C\|, c$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$\|C\|, \|u\|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$\|C\|, \|Q\|$	???	???	???	???
$c, \|u\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$c, \|Q\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$\|u\|, \|Q\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$\|C\|, c, \|u\|$	$W[1]$-hard	$W[1]$-hard	$W[1]$-hard	FPT
$\|C\|, c, \|Q\|$	FPT	FPT	FPT	FPT
$\|C\|, \|u\|, \|Q\|$	FPT	FPT	FPT	FPT
$c, \|u\|, \|Q\|$	$W[2]$-hard	$W[2]$-hard	$W[2]$-hard	FPT
$\|C\|, c, \|u\|, \|Q\|$	FPT	FPT	FPT	FPT

Table 4.13: The Parameterized Complexity of the OT-Decode Problem.

Parameter	Alphabet Sizes ($\|\Sigma_h\|$,$\|\Sigma_v\|$)			
	(Unb,Unb)	(Unb,Prm)	(Prm,Unb)	(Prm,Prm)
–	NP-hard	$\notin XP$	$\notin XP$	$\notin XP$
$\|C\|$	$W[t]$-hard	$W[t]$-hard	$W[t]$-hard	$W[t]$-hard
c	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$\|s\|$	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$\|Q\|$	$\notin XP$	$\notin XP$	???	???
$\|C\|, c$	$W[1]$-hard	$W[1]$-hard	FPT	FPT
$\|C\|, \|s\|$	$W[1]$-hard	$W[1]$-hard	FPT	FPT
$\|C\|, \|Q\|$	???	???	???	???
$c, \|s\|$	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$c, \|Q\|$	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$\|s\|, \|Q\|$	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$\|C\|, c, \|s\|$	$W[1]$-hard	$W[1]$-hard	FPT	FPT
$\|C\|, c, \|Q\|$	FPT	FPT	FPT	FPT
$\|C\|, \|s\|, \|Q\|$	FPT	FPT	FPT	FPT
$c, \|s\|, \|Q\|$	$W[2]$-hard	$W[2]$-hard	FPT	FPT
$\|C\|, c, \|s\|, \|Q\|$	FPT	FPT	FPT	FPT

4.7 Some Final Thoughts on the Computational Complexity of Phonological Processing

In this chapter, a number of complexity-theoretic results have been derived for various problems associated with five phonological theories. To the extent that these problems have been formulated correctly relative to the guidelines and caveats given in Chapter 3, these results have implications for their associated theories. Some of these implications for particular theories have already been discussed. This section will focus on the implications of the results derived in this chapter for phonological processing in general. As such, it consists of descriptions of some patterns in the derived results and some speculations about the factors responsible for these patterns and what they may mean for phonological processing in general.

The most obvious pattern is the limited number of types of mechanisms underlying observed sources of polynomial-time intractability. These sources are listed in Tables 4.14 and 4.15, and seem to fall into the following two broad categories:

1. The amount and composition of "hidden" structure in the representation proposed by the phonological theory, i.e., structure that is not dependent on and hence can be manipulated independently of given lexical or surface forms (**Computational Space**).
2. The complexity of the mapping relating lexical and surface forms as implemented by mechanisms within the phonological theory (**Complexity of Mapping**).

Each of these types of sources will be discussed in more detail below.

Consider the first of these types of sources. The role of hidden structure has previously been acknowledged in the discussions of all phonological theories examined in this chapter. This hidden structure has manifested itself in these theories in different ways:

- Intermediate forms in the derivation (Simplified Segmental Grammars and FST-based rule systems).
- The surface form (in the encoding problem) or the lexical form (in the decoding problems) (the KIMMO system).
- The underspecified portions of the hidden and visible components in the lexical form (in the encoding problem) or the underspecified hidden component in the surface form (in the decoding problem) (Declarative Phonology and Optimality Theory).

What becomes obvious when all theories are considered together is that when one is interested in what aspects of this hidden structure are sources of polynomial-time intractability, the precise manner in which the hidden structure manifests itself, e.g., the number of intermediate forms in a rule-based derivation, is not as important as the overall amount and composition of this structure, e.g., the number of segment-strings of a given size relative

Phonological Theory	Sources of Polynomial-Time Intractability	
	Complexity of Mapping	Computational Space
Simplified Segmental Grammars	$\{\lvert R\rvert_?\}$	$\{\lvert u\rvert, \lvert f\rvert, \lvert v\rvert\}$
FST-Based Rule Systems	$\{\lvert A\rvert, \lvert Q\rvert\}$	$\{\lvert u\rvert, \lvert\Sigma\rvert\}$
The KIMMO System	$\{\lvert A\rvert, \lvert Q\rvert\}$	$\{\lvert u\rvert, \lvert\Sigma_s\rvert\}$
Declarative Phonology	$\{\lvert P\rvert, c, \lvert Q\rvert\}$	$\{c, \lvert\Sigma_h\rvert, \lvert\Sigma_v\rvert\}$
Optimality Theory	$\{\lvert C\rvert, c, \lvert Q\rvert\}$	$\{c, \lvert\Sigma_h\rvert, \lvert\Sigma_v\rvert\}$

Table 4.14: Sources of Polynomial-Time Intractability in the Encoding Decision Problems Associated With Phonological Theories Examined in This Thesis. See the appropriate section in this chapter for definitions of the aspects in the sources given for a particular phonological theory.

Phonological Theory	Sources of Polynomial-Time Intractability	
	Complexity of Mapping	Computational Space
Simplified Segmental Grammars	$\{\lvert R\rvert, \lvert s\rvert\}$	$\{\lvert s\rvert, \lvert f\rvert, \lvert v\rvert\}$
FST-Based Rule Systems	$\{\lvert A\rvert, \lvert Q\rvert\}$	$\{\lvert s\rvert, \lvert\Sigma\rvert\}$
The KIMMO System	$\{\lvert A\rvert, \lvert Q\rvert\}$	$\{\lvert s\rvert, \lvert\Sigma_u\rvert\}$
Declarative Phonology	$\{\lvert P\rvert, c, \lvert Q\rvert\}$	$\{c, \lvert\Sigma_h\rvert\}$
Optimality Theory	$\{\lvert C\rvert, c, \lvert Q\rvert\}$	$\{c, \lvert\Sigma_h\rvert\}$

Table 4.15: Sources of Polynomial-Time Intractability in the Decoding Decision Problems Associated With Phonological Theories Examined in This Thesis. See the appropriate section in this chapter for definitions of the aspects in the sources given for a particular phonological theory.

to a particular set of (possibly multi-valued) features. One immediate consequence of this is that it shows that certain claims in the literature that particular linguistic theories are better on computational grounds simply because they have fewer levels of representation are misguided (see papers in [Gol93] and references).

All of the reductions given in this thesis exploit hidden structure for encoding instances of NP-hard problems; hence, having a rich hidden structure is one of the keys to NP-hardness in phonological processing. However, it is not by itself enough to guarantee NP-hardness; this hidden structure must also be adequately accessible to mechanisms within these theories. This is shown particularly well by the analyses done for Declarative Phonology and Optimality Theory, in which the context-size of the constraints (and hence the amount of hidden structure that any constraint can act upon) is shown to be part of a source of polynomial-time intractability. The investigation of other aspects besides context-size that constrain the access of mechanisms to representations in phonological theories is an interesting topic for future research.

Consider now the second of these types of sources. Encoding and decoding problems associated with the phonological theories examined in this thesis are NP-hard under a number of different types of mechanisms for implementing the mapping between lexical and surface forms. What is surprising is that this NP-hardness does not seem to depend on such aspects as the number or size of individual mapping-mechanisms or even the type of mapping mechanism, e.g., rule or constraint, but rather on the size and complexity of the combination of these mechanisms that actually implements the mapping between surface and lexical forms. This suggests that a truly useful investigation of the sources of polynomial-time intractability in phonological theories in terms of the mechanisms in those theories should focus on the nature of the mappings between representations encoded in individual mechanisms and the combination of these mechanisms. A useful framework for doing this is to view all mechanisms, be they rules, constraints, or lexicons, as relations between sets of representations and to view mappings between lexical and surface forms as compositions of these relations [Kar98]. In this framework, a rule is a relation between the set of all representations and the set of all representations resulting from the application of the rule, a constraint is an identity relation on the set of all representations accepted by the constraint, and a lexicon is an identity relation on all representations that can be generated by the lexicon. A recasting of the phonological theories examined in this thesis into this framework is given in Figure 4.12. One immediate consequence of this recasting is that it graphically shows the similarity recently noted by Karttunen [Kar98] of Optimality Theory to older constraint- and rule-based theories, and suggests a natural progression in the computational power required by various phonological theories, in that constraint-based systems are special cases of rule-base systems which are in turn special cases of Optimality-Theoretic systems.

All of the reductions given in this thesis exploit mapping mechanisms to operate on the instances of NP-hard problems that are encoded in the hidden structure in the representations; hence, having a mapping mechanism that can encode a rich variety of relations between representations is another key to NP-hardness in phonological processing. As noted in Section 4.1.3, there can be interesting tradeoffs between the richness of various types of mechanisms, e.g., a rich lexicon can compensate for a constrained rule-set and vice versa. Note that richly expressive mapping mechanisms are not in themselves enough to guarantee NP-hardness; as was pointed out above, these mechanisms must also have adequate access to the representations. One powerful constraint on this access is limited context-size; another is to limit the mapping mechanism itself (rather than the representation) such that the mapping mechanism can only access a limited number of representations (see part (3) of Theorem 4.1.18). An investigation of aspects that characterize the richness of the relations encoded in individual phonological mechanisms and their composition as a whole (possibly by formulating these relations in logic along the lines suggested in Section 4.2) is an interesting topic for future research.

The observations above derived relative to decision problems can also be used to explain the observed patterns of computational complexity in the various search problems associated with phonological theories that were examined in this thesis. These patterns are summarized in Table 4.16. The patterns for each problem can be explained as follows:

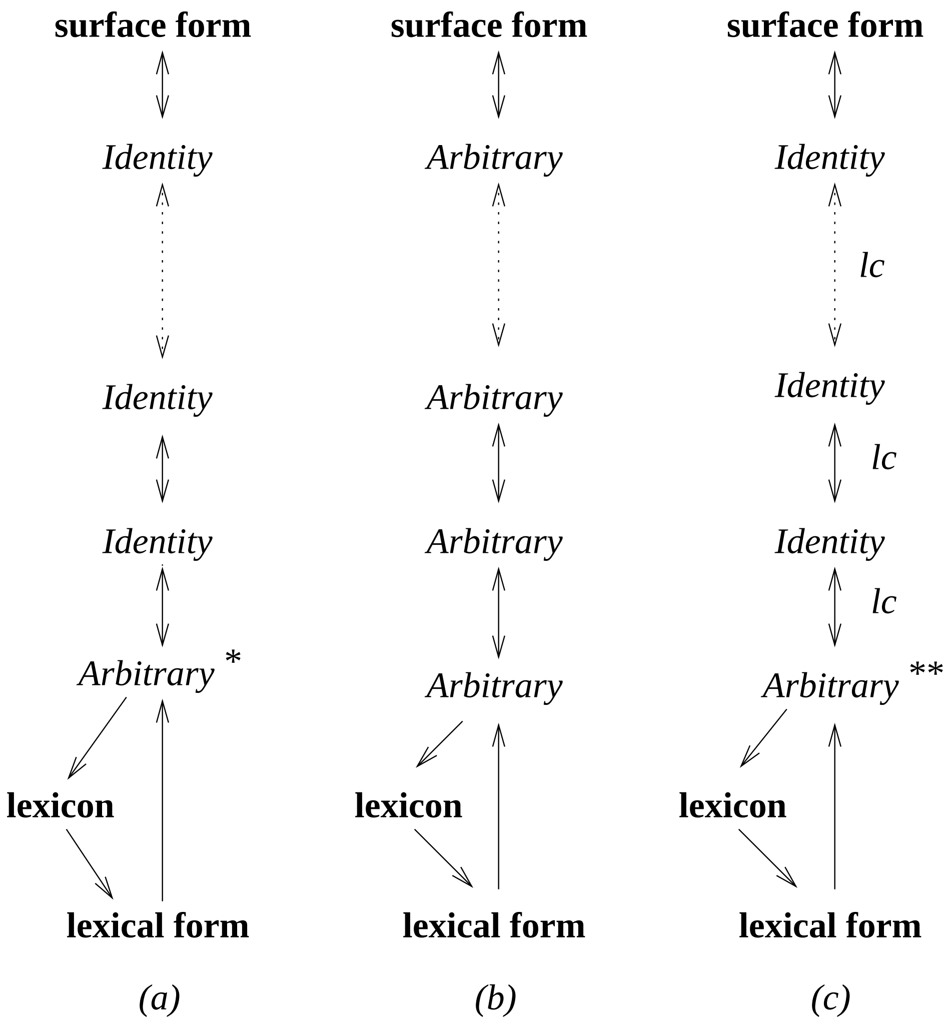


Figure 4.12: Phonological Theories as Compositions of Representation-Relations. The above shows various reformulations of phonological theories examined in this thesis as compositions of representation-relations. Italicized terms x represent representation-relations of type x. An arrow between two representation-relations indicates composition; the arrow's direction indicates which way representations can move through the composition. a) The reformulation of constraint-based theories such as Declarative Phonology and KIMMO (where the representation in KIMMO is interpreted as a lexical-surface form pair). The starred (*) relation generates all fully-specified representations consistent with the lexical form. b) The reformulation of rule-based theories like FST-based rule systems and Simplified Segmental Grammars. c) The reformulation of Optimality-Theoretic systems which allow a bounded number of violations per constraint (following that described in [Kar98]). In this scheme, relations denoting the sets of representations that violate particular constraints some fixed number of times are combined by the lenient composition (lc) operation, which essentially allows the composition to ignore those relations that would eliminate all candidate surface forms. The doubly starred (**) relation corresponds to Gen.

Phonological Theory	Search Problem		
	ENC	DEC	CHK
Simplified Segmental Grammars		$\surd$	$\surd$
FST-Based Rule Systems	$\surd$	$\surd$	$\surd$
The KIMMO System	$\surd$	$\surd$	
Declarative Phonology	$\surd$	$\surd$	$\surd$
Optimality Theory	$\surd$	$\surd$	$\surd$

Table 4.16: The Computational Complexity of Search Problems Associated With Phonological Theories Examined in This Thesis. For phonological theory $\mathbf{X} \in \{$**SSG, FST. KIM, DP, OT**$\}$, the abbreviations ENC, DEC, and CHK above stand for the search problems $ENC_{\mathbf{X}}$, $DEC_{\mathbf{X}}$, and $CHK_{\mathbf{X}}$ defined in the introduction to Chapter 4. A check-mark ($\surd$) in the box corresponding to a particular search problem associated with some phonological theory indicates that this search problem is not solvable in polynomial time unless $P = NP$.

- ENC: The computational complexity of the encoding search problem depends both on the richness of the mapping mechanism and the computational complexity of accessing forms produced by applying this mechanism. In the case of Simplified Segmental Grammars, the NP-hardness of SSG-ENCODE establishes that the mapping mechanism is sufficiently rich; this suggests that the polynomial-time solvability of $ENC_{\mathbf{SSG}}$ is a result of being able to access arbitrary rather than particular surface forms
associated with a given lexical form.
- DEC: The computational complexity of the decoding search problem depends on the combined richness of the lexicon and the mapping mechanism.
- CHK: The computational complexity of the checking search problem depends on whether representations are the sum of lexical and surface forms, i.e., if there is any hidden structure in the representation that is not contained in the union of the structure in the lexical and surface forms. If there is no such hidden structure (as is the case with the KIMMO system), then the problem can be solved in polynomial time. If there is such hidden structure (as is the case in all other phonological theories examined in this thesis), the problem is not solvable in polynomial time unless $P = NP$.

An area of research of particular interest in light of the polynomial-time solvability of $ENC_{\mathbf{SSG}}$ is the formulation and analysis of search problems that access the sets of forms produced by the application of mapping mechanisms to given forms in manners different than those examined here, e.g., enumeration of all forms vs. extraction of an arbitrary form.

A final caveat is perhaps in order. It is very tempting to continue extrapolating from the formal results given here to progressively grander albeit more informal conclusions. For instance, the observation that there are very few distinct FPT algorithms relative to the aspects considered for the problems examined in this thesis could be construed as suggesting an interesting and somewhat constrained structure underlying phonological processing in general. Evocative though such extrapolations may be, they are no substitute for (and may indeed be a hindrance to formulating) testable hypotheses for future research. It must also be remembered that all interpretations of results given above are tentative (both because of the incompleteness of the intractability maps derived here and the dependence of these interpretations on unproven conjectures about the separateness of P and NP as well as the levels of the W hierarchy), and that it is inevitable that many of these interpretations will have to be revised or discarded in light of future results. That being said, it does seem certain that both the conceptual framework and techniques of systematic parameterized complexity analysis will continue to be of use in deriving and interpreting such results.

Chapter 5

Conclusions

In this thesis, systematic parameterized complexity analysis has been formally defined as the systematic application of techniques developed within the theory of parameterized computational complexity [DF95a, DF95b, DF99], and it has been discussed why this type of analysis is better than classical complexity-theoretic analyses such as NP-completeness at establishing the sources of polynomial-time intractability in computational problems. This discussion has been illustrated by systematic parameterized analyses of the encoding and decoding problems associated with five theories of phonological processing in natural languages. A summary of the results derived for these theories is given in the introduction to this thesis; for details, please see the appropriate **Analysis** and **Implications** sections in Chapter 4 and the general discussion of the implications of all results derived in this thesis given in Section 4.7.

Of all the implications discussed in this thesis, one stands out above the rest: namely, that the NP-hardness of the encoding and decoding problems associated with a phonological theory seems to be independent of many of the gross aspects of the processing architecture postulated by that theory. That is, it doesn't seem to matter whether phonological mechanisms are rules or constraints, whether these mechanisms are or are not finite-state, whether these mechanisms are ordered or unordered, or whether the system has one, two, or many levels of representation — to the extent that the theories examined in this thesis are representative of the spectrum of phonological theories embodying combinations of these alternatives, the encoding and decoding problems associated with all such theories seem to be NP-hard in general. This will not and should not have any effect on the work of linguists who use these theories to describe languages, as such computational issues are not a consideration in creating such descriptions. However, the above should be important to computational linguists, as it should lay to rest debates over and efforts to create phonological theories whose associated operations are efficient and focus efforts on the far more interesting and useful question of why phonological processing seems to be so computationally difficult.

Many directions for future research have been proposed in this thesis. Future research particular to specific phonological theories is described following the analyses for these theories in Chapter 4; some directions for future research that apply to all theories are given below. The focus of such research should be on removing the "in general" qualifier in the preceding paragraph. Though phonological processing may indeed be NP-hard in general, there may be aspects latent in phonological theories whose restriction may yet make such processing efficient in practice. As already discussed in Chapter 4, aspects of particular interest are those that capture the richness of the representation-relations implicit in individual phonological mechanisms. Some other potential sources of useful aspects are as follows:

- A very important restriction on phonological theories that has not been addressed at all in this thesis (or for that matter in the literature on the computational analysis of phonological theories) is the maximum number of solutions that can be associated with a given input — that is, the maximum number of surface (lexical) forms that can be associated with a given lexical (surface) form in the search version of an encoding (decoding) problem. Typically, this number will be very small; however, in many instances of phonological systems created by reductions in this thesis, this number is either zero or very large. It would be interesting to see which (if any) NP- and W-hardness results will still hold for the theories examined in this thesis if this number is bounded. This research may be aided by employing parameterized analogs of various complexity classes developed to study problems with bounded numbers of solutions, e.g., UP, $FewP$, $SpanP$ (see [JoD90, Sections 4.1 and 4.2] and references). For now, note that under fairly broad conditions, the problems examined in this thesis for Simplified Segmental Grammars, FST-based rule systems, and Optimality Theory remain NP-hard if the number of solutions is bounded to one in both encoding and decoding problems (this already holds for Simplified Segmental Grammars and FST-based rule systems by the reductions given in this thesis; the appropriate modifications to the reductions for Optimality Theory involve the addition of new constraints to C which select the lexicographically smallest candidate full form).

- All problems examined in this thesis assume that the mapping mechanisms, e.g., constraints, rules, and lexicon, are part of the problem instance. However, if these mechanisms are fixed in advance, all aspects associated with these mechanisms become constants, and many of the FPT algorithms derived in this thesis for these problems that are based on these aspects can be used to solve these problems in polynomial time. This process of fixing the mapping mechanism in advance allows these mechanisms to be "pre-compiled", which in turn allows the computational complexity associated with these mechanisms to be removed from the problem.

  The practicality of pre-compilation is discussed at length in [BBR87, Section 2.3]. Though pre-compilation seems like a good idea and it reflects the biological reality that people already know a language before they use it to communicate, it can be shown that any such pre-compilation process cannot be computable in polynomial time unless $P = NP$ (otherwise, one could solve the NP-hard encoding and decoding problems considered in this thesis by composing such pre-compilation processes with the polynomial-time algorithms that use the compiled mechanisms). The big question is, in what aspects and manners relative to those aspects can the non-polynomial time algorithmic behavior of the pre-compilation process be expressed? It would be interesting to know the spectrum of options for such pre-compilation. The FPT algorithms in this thesis embody several types of pre-compilation, e.g., finite-state intersection and composition. Future research should apply techniques developed specifically to create FPT algorithms to see if other less obvious types of pre-compilation exist. This research may be aided by complexity-theoretic techniques developed within the knowledge database literature for assessing the pre-compilability of computational problems [CD98, CDLS96, CDLS97].

It might also be of interest to examine natural language data itself rather than phonological theories to see if there are aspects of this data whose values are bounded in practice and have associated FPT algorithms. A good starting point for such research would be the surface-form data for Finnish examined in [KC88] which an implementation of the KIMMO system analyzed into lexical forms in time almost linear in the length of the given surface forms, cf. the NP-hardness of KIM-DECODE and the $W[2]$-hardness of $\langle |s| \rangle$-KIM-DECODE as established in Theorem 4.4.4.

Looking back, it is interesting that many of the questions that are asked about the computational complexity of phonological processing are most naturally formulated and answered within the framework of parameterized complexity analysis. As discussed in Section 2.1.3 and Chapter 3, this is perhaps not overly surprising as parameterized computational complexity theory is arguably one of the most "realistic" theories of computational complexity proposed to date, in the sense of acknowledging and accommodating both the realities of problem solvability and the needs of algorithm users and designers. It is my hope that the research described in this thesis will both encourage systematic parameterized complexity analyses of other NP-hard problems and serve as an example of how such analyses should be carried out and interpreted.

Bibliography

[And94] Stephen Anderson. 1994. Parsing morphology: "Factoring" words. In Eric S. Ristad, ed., *Language Computations*, pages 167–183. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 17. American Mathematical Society, Providence, RI.

[Ant90] E.L. Antworth. 1990. *PC-KIMMO: A two-level processor for morphological analysis.* Occasional Publications in Academic Computing no. 16. Summer Institute of Linguistics, Dallas, TX.

[Bae91] Ricardo A. Baeza-Yates. 1991. Searching Subsequences. *Theoretical Computer Science*, 78, 363–376.

[Bar86] G. Edward Barton. 1986. Computational complexity in two-level morphology. In *Proceedings of the 24th Conference of the Association for Computational Linguistics*, pages 53–59.

[BBR87] G. Edward Barton, Robert C. Berwick, and Eric S. Ristad. 1987. *Computational Complexity and Natural Language.* MIT Press, Cambridge, MA.

[Brs79] Jean Berstel. 1979. *Rational Transductions and Context-Free Languages.* B.G. Tuebner, Stuttgart.

[Ber84] Robert C. Berwick. 1984. Strong Generative Capacity, Weak Generative Capacity, and Modern Linguistic Theories. *Computational Linguistics*, 10(3–4), 189–202.

[Ber85] Robert C. Berwick. 1985. *The Acquisition of Syntactic Knowledge.* MIT Press, Cambridge, MA.

[Ber89] Robert C. Berwick. 1989. Natural Language, Computational Complexity, and Generative Capacity. *Computers and Artificial Intelligence*, 8(5), 423–441.

[BF95] Robert C. Berwick and Sandiway Fong. 1995. A quarter century of computation with transformational grammar. In Jennifer Cole, Georgia M. Green, and Jerry L. Morgan, eds., *Linguistics and Computation*, pages 103–143. CSLI Lecture Notes no. 52. CSLI Publications, Stanford, CA.

[BW83] Robert C. Berwick and Amy Weinberg. 1983. Parsing efficiency, computational complexity, and the evaluation of grammatical theories. *Linguistic Inquiry*, 13, 165–191.

[BW84] Robert C. Berwick and Amy Weinberg. 1984. *The Grammatical Basis of Linguistic Performance.* MIT Press, Cambridge, MA.

[Bir95] Steven Bird. 1995. *Computational Phonology: A Constraint-Based Approach.* Cambridge University Press.

[BE94] Steven Bird and T. Mark Ellison. 1994. One-Level Phonology: Autosegmental Representations and Rules as Finite Automata. *Computational Linguistics*, 20(1), 55–90.

[BDF+95] Hans L. Bodlaender, Rod G. Downey, Michael R. Fellows, Michael T. Hallett, and H. Todd Wareham. 1995. Parameterized Complexity Analysis in Computational Biology. *Computer Applications in the Biosciences*, 11(1), 49–57.

[BDFW95] Hans L. Bodlaender, Rod G. Downey, Michael R. Fellows, and H. Todd Wareham. 1995. The Parameterized Complexity of Sequence Alignment and Consensus. *Theoretical Computer Science*, 147(1–2), 31–54.

[BS90] Ravi B. Boppana and Michael Sipser. 1990. The Complexity of Finite Functions. In Jan van Leeuwen, ed., *Handbook of Theoretical Computer Science: Volume A*, pages 757–804. Elsevier, Amsterdam.

[Bre95] Diane Brentari. 1995. Sign Language Phonology: ASL. In John Goldsmith, ed., *Handbook of Phonology*, pages 615–639. Basil Blackwell, Cambridge. MA.

[CD98] Marco Cadoli and Francesco M. Donini. 1998. A Survey on Knowledge Compilation. Manuscript.

[CDLS96] Marco Cadoli, Francesco M. Donini, Paolo Liberatore, and Marco Schaerf. 1996. Feasibility and Unfeasibility of Off-line Processing. In *Proceedings of the Fourth Israeli Symposium on Theory of Computing and Systems (ISTCS-96).* Lecture Notes in Computer Science. Springer-Verlag, Berlin.

[CDLS97] Marco Cadoli, Francesco M. Donini, Paolo Liberatore, and Marco Schaerf. 1997. Preprocessing of intractable problems. Technical report DIS 24-97, Dipartimento di Informatica e Sistemistica, Universitá di Roma "La Sapienza".

[CCDF94] Liming Cai, Jianer Chen, Rod G. Downey, and Michael R. Fellows. 1994. On the structure of parameterized problems in NP. In *STACS'94: Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science*, pages 509–520. Lecture Notes in Computer Science. Springer-Verlag, Berlin.

[Ces96] Marco Cesati. 1996. Structural Aspects of Parameterized Complexity. Ph.D. thesis, Università degli Studi di Roma "La Sapienza".

[CI97] Marco Cesati and Miriam Di Ianni. 1997. Parameterized Parallel Complexity. Technical Report TR97-006, Electronic Colloquium on Computational Complexity.

[CW95] Marco Cesati and H. Todd Wareham. 1995. Parameterized Complexity Analysis in Robot Motion Planning. In *Proceedings of the 25th IEEE International Conference on Systems, Man, and Cybernetics: Volume 1*, pages 880–885. IEEE Press, Los Alamitos, CA.

[Cho57] Noam Chomsky. 1957. *Syntactic Structures.* Janua Linguarum nr. 4. Mouton, 's-Gravehage.

[CH68] Noam Chomsky and Morris Halle. 1968. *The Sound Pattern of English.* Harper and Row, New York.

[Col95] John Coleman. 1995. Phonology and computational linguistics – a personal overview In Jennifer Cole, Georgia M. Green and Jerry L. Morgan, eds., *Linguistics and Computation*, pages 223–254. CSLI Lecture Notes no. 52. CSLI Publications, Stanford, CA.

[CL92] John Coleman and John Local. 1992. The "No Crossing" Constraint in Autosegmental Phonology. *Linguistics and Philosophy*, 14, 295–338.

[Coo71] Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing (STOC'71)*, pages 151–158. ACM Press, New York, NY.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. 1990. *Introduction to Algorithms.* MIT Press, Cambridge, MA.

[Cre95] Nadia Creignou. 1995. A Dichotomy Theorem for Maximum Generalized Satisfiability Problems. *Journal of Computer and System Sciences*, 51, 511–522.

[DeF91] Richard A. Demers and Ann K. Farmer. 1991. *A Linguistic Workbook.* Second edition. MIT Press, Cambridge, MA.

[DF93] Rod G. Downey and Michael R. Fellows. 1993. Fixed-parameter tractability and completeness III. Some structural aspects of the W-hierarchy. In Klaus Ambos-Spies, Steve Homer, and Uwe Schöning, eds., *Complexity Theory*, pages 166–191. Cambridge University Press.

[DF95a] Rod G. Downey and Michael R. Fellows. 1995a. Fixed-parameter tractability and completeness I. Basic results. *SIAM Journal on Computing*, 24(4), 873–921.

[DF95b] Rod G. Downey and Michael R. Fellows. 1995b. Fixed-parameter tractability and completeness II. On completeness for $W[1]$. *Theoretical Computer Science*, 141, 109–131.

[DF95c] Rod G. Downey and Michael R. Fellows. 1995c. Parameterized computational feasibility. In Peter Clote and Jeffrey B. Remmel, eds., *Feasible Mathematics II*, pages 219–244. Birkhäuser, Boston, MA.

[DF99] Rod G. Downey and Michael R. Fellows. 1999. *Parameterized Complexity.* Springer-Verlag, Berlin.

[DFK+94] Rod G. Downey, Michael R. Fellows, Bruce M. Kapron, Michael T. Hallett, and H. Todd Wareham. 1994. Parameterized Complexity of Some Problems in Logic and Linguistics (Extended Abstract). In A. Nerode and Yuri V. Matiyasevich, eds., *Logical Foundations of Computer Science*, pages 89–101. Lecture Notes in Computer Science no. 813. Springer-Verlag, Berlin.

[DFS98] Rod G. Downey, Michael R. Fellows, and Ulrike Stege. 1998. Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability. In F. Roberts, J. Nesetril, and J. Kratochvil, eds., *The Future of Discrete Mathematics: Proceedings of the First DIMACS-DIMATIA Workshop, Prague, 1997.* AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, Providence, RI.

[Eis97a] Jason Eisner. 1997a. Efficient Generation in Primitive Optimality Theory. Technical Report ROA-206-0797, Rutgers Optimality Archive.

[Eis97b] Jason Eisner. 1997b. What Constraints Should OT Allow? Technical Report ROA-204-0797, Rutgers Optimality Archive.

[Ell94] T. Mark Ellison 1994. Phonological Derivation in Optimality Theory. In *Proceedings of the Fifteenth International Conference on Computational Linguistics (COLING'94)*, pages 1007–1013.

[Ell95] T. Mark Ellison 1995. OT, Finite-State Representations, and Procedurality. Manuscript.

[Ell96] T. Mark Ellison 1996. The Universal Constraint Set: Convention not Fact. Manuscript.

[Eva98] Patricia A. Evans. 1998. Algorithms and Complexity for Annotated Sequence Analysis. Ph.D. thesis in preparation, Department of Computer Science, University of Victoria.

[FV93] Tomás Feder and Moshe Y. Vardi. 1993. Monotone Monadic SNP and Constraint Satisfaction. In *Proceedings of the 25th Annual ACM Symposium on the Theory of Computing (STOC'93)*, pages 612–622. ACM Press, New York, NY.

[Fel97] Michael R. Fellows. 1997. Personal communication.

[FS97] Robert Frank and Giorgio Satta. 1997. Optimality Theory and the Generative Complexity of Constraint Violability. Technical Report ROA-228-1197, Rutgers Optimality Archive.

[GJ79] Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco.

[Gaz85] Gerald Gazdar. 1985. Computationally Relevant Properties of Natural Languages and Their Grammars. *New Generation Computing*, 3, 273–306.

[Gol76] John Goldsmith. 1976. Autosegmental Phonology. Ph.D. thesis, MIT.

[Gol90] John Goldsmith. 1990. *Autosegmental and Metrical Phonology.* Basil Blackwell, Cambridge, MA.

[Gol93] John Goldsmith, ed. 1993. *The Last Phonological Rule: Reflections on Constraints and Derivations.* The University of Chicago Press.

[Hal96] Michael T. Hallett. 1996. An Integrated Complexity Analysis of Problems from Computational Biology. Ph.D. thesis, Department of Computer Science, University of Victoria.

[Ham97] Michael Hammond. 1997. Parsing syllables: modeling OT computationally. Technical Report ROA-222-1097, Rutgers Optimality Archive.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA.

[JoC72] C. Douglas Johnson. 1972. *Formal Aspects of Phonological Description.* Mouton, The Hague.

[JoD85] David S. Johnson. 1985. The NP-Completeness Column: An Ongoing Guide (15th edition). *Journal of Algorithms*, 6, 291–305.

[JoD90] David S. Johnson. 1990. A Catalog of Complexity Classes. In Jan van Leeuwen, ed., *Handbook of Theoretical Computer Science: Volume A*, pages 67–161. Elsevier, Amsterdam.

[JoM92] Mark Johnson. 1992. The Complexity of Inducing a Rule from Data. In Jonathan Mead, ed., *Proceedings of the Eleventh West Coast Conference on Formal Linguistics (WCCFL XI)*, pages 289–297. CSLI Publications, Stanford, CA.

[KK94] Ronald M. Kaplan and Martin Kay. 1994. Regular Models of Phonological Rule Systems. *Computational Linguistics*, 20(3), 331–378.

[Kar83] Lauri Karttunen. 1983. KIMMO: A general morphological processor. *Texas Linguistics Forum*, 22, 165–186.

[Kar93] Lauri Karttunen. 1993. Finite-State Constraints. In John Goldsmith, ed., *The Last Phonological Rule: Reflections on Constraints and Derivations*, pages 173–194. The University of Chicago Press.

[Kar98] Lauri Karttunen. 1998. The Proper Treatment of Optimality in Computational Phonology. Technical Report ROA-258-0498, Rutgers Optimality Archive.

[KCGS96] Lauri Karttunen, Jean-Pierre Chanod, Gregory Grefenstette, and Anne Schiller. 1996. Regular Expressions for Language Engineering. *Natural Language Engineering*, 2(4), 305–328.

[Kay87] Martin Kay. 1987. Nonconcatenative finite-state morphology. In *Proceedings of the Third Meeting of the European Chapter of the Association for Computational Linguistics*, pages 2–10.

[Ken94] Michael J. Kenstowicz. 1994. *Phonology in Generative Grammar.* Basil Blackwell, Cambridge, MA.

[Kor95] András Kornai. 1995. *Formal Phonology.* Garland Publishing, New York.

[Kos83] Kimmo Koskenniemi. 1983. Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production. Ph.D. thesis, University of Helsinki.

[KC88] Kimmo Koskenniemi and Kenneth W. Church. 1988. Complexity, Two-Level Morphology, and Finnish. In the *Proceedings of the 12th International Conference on Computational Linguistics (COLING'88)*, pages 335–340. John von Neumann Society for Computing Sciences, Budapest.

[LP93] Darlene Lacharité and Carole Paradis. 1993. The Emergence of Constraints in Generative Phonology and a Comparison of Three Current Constraint-Based Models. *Canadian Journal of Linguistics*, 38(2), 127–163.

[Lap97] Eric Laporte 1997. Rational Transductions for Phonetic Conversion and Phonology. In Emmanuel Roche and Yves Schabes, eds., *Finite-State Language Processing*, pages 407–429. MIT Press, Cambridge, MA.

[LP81] Harry R. Lewis and Christos H. Papdimitriou. 1981. *Elements of the Theory of Computation.* Prentice Hall, Englewood Cliffs, NJ.

[Lov73] Julie B. Lovins. 1973. Loanwords and the Phonological Structure of Japanese. Ph.D. thesis, University of Chicago. Reproduced by the Indiana University Linguistics Club.

[Mac92] Alan K. Mackworth. 1992. Constraint Satisfaction. In Stuart C. Shapiro, ed., *Encyclopedia of Artificial Intelligence*, pages 285–293. Second edition. John Wiley and Sons, New York, NY.

[Mai78] David Maier. 1978. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25(2), 322–336.

[MaR95] Alexis Manaster Ramer. 1995. Book Review: Ristad, *The Language Complexity Game*. *Computational Linguistics*, 21(1), 124–131.

[MP93] John McCarthy and Alan Prince. 1993. Prosodic Morphology I. Constraint Interaction and Satisfaction. Technical Report RuCCS TR-3, Rutgers University Center for Cognitive Science.

[Moh97] Mehryar Mohri. 1997. On the Use of Sequential Transducers in Natural Language Processing. In Emmanuel Roche and Yves Schabes, eds., *Finite-State Language Processing*, pages 355–382. MIT Press, Cambridge, MA.

[NMK94] J. Thomas Ngo, Joe Marks, and Martin Karplus. 1994. Computational Complexity, Protein Structure Prediction, and the Levinthal Paradox. In K. Merz and S. Le Grand, eds., *The Protein Folding Problem and Tertiary Structure Prediction*, pages 433–506. Birkhäuser, Boston, MA.

[OSB94] Naomi Oreskes, Kristen Schrader-Frechette, and Kenneth Belitz. 1994. Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences. *Science*, 263, 641–646.

[Pap94] Christos H. Papdimitriou. 1994. *Computational Complexity.* Addison-Wesley, Reading, MA.

[PR97] Fernando C.N. Pereira and Michael D. Riley. 1997. Speech Recognition by Composition of Weighted Finite Automata. In Emmanuel Roche and Yves Schabes, eds., *Finite-State Language Processing*, pages 431–453. MIT Press, Cambridge, MA.

[Per90] Dominique Perrin. 1990. Finite Automata. In Jan van Leeuwen, ed., *Handbook of Theoretical Computer Science: Volume B*, pages 1–57. Elsevier, Amsterdam.

[PS93] Alan Prince and Paul Smolensky. 1993. Optimality Theory: Constraint Interaction in Generative Grammar. Technical Report RuCCS TR-2, Rutgers University Center for Cognitive Science.

[Ram97] Rajeev Raman. 1997. Recent Results on the Single-Source Shortest Paths Problem. *SIGACT News*, 28(2), 81–87.

[Rei87] John H. Reif 1987. Complexity of the Generalized Mover's Problem. In J.T. Schwartz, Micha Sharir, and John E. Hopcroft, eds., *Planning, Geometry, and Complexity of Robot Motion*, pages 267–281. Ablex Publishing Corporation, Norwood, NJ.

[Ris90] Eric S. Ristad. 1990. Computational Structure of Human Language. Ph.D. thesis, Department of Electrical Engineering and Computer Science, MIT.

[Ris93a] Eric S. Ristad. 1993a. *The Language Complexity Game.* MIT Press, Cambridge, MA.

[Ris93b] Eric S. Ristad. 1993b. Complexity of the Simplified Segmental Phonology. Technical Report CS-TR-388-92 (revised May 1993), Department of Computer Science, Princeton University.

[Ris94] Eric S. Ristad. 1994. Complexity of morpheme acquisition. In Eric S. Ristad, ed., *Language Computations*, pages 185–198. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 17. American Mathematical Society, Providence, RI.

[Rit92] Graeme D. Ritchie. 1992. Languages Generated by Two-Level Morphological Rules. *Computational Linguistics*, 18(1), 41–59.

[RRBP92] Graeme D. Ritchie, Graham J. Russell, Alan W. Black, and Stephen G. Pullman. 1992. *Computational Morphology: Practical Mechanisms for the English Lexicon.* MIT Press, Cambridge, MA.

[RS97a] Emmanuel Roche and Yves Schabes, eds. 1997a. *Finite-State Language Processing.* MIT Press, Cambridge, MA.

[RS97b] Emmanuel Roche and Yves Schabes. 1997b. Introduction. In Emmanuel Roche and Yves Schabes, eds., *Finite-State Language Processing*, pages 1–66. MIT Press, Cambridge, MA.

[Rou87] William Rounds. 1987. Book Review: Barton, Berwick, and Ristad, *Computational Complexity and Natural Language. Computational Linguistics*, 13(3–4), 354–356.

[Rou91] William Rounds. 1991. The Relevance of Computational Complexity Theory to Natural Language Processing. In Peter Sells, Stuart Shieber, and Thomas Wasow, eds., *Foundational Issues in Natural Language Processing*, pages 9–29. MIT Press, Cambridge, MA.

[Sch78] Thomas J. Schaefer. 1978. The Complexity of Satisfiability Problems. In *Proceedings of the 10th Annual ACM Symposium on the Theory of Computing (STOC'78)*, pages 216–226. ACM Press, New York, NY.

[SB78] Sanford A. Schane and Birgitte Bendixen. 1978. *Workbook in Generative Phonology.* Prentice Hall, Englewood Cliffs, NJ.

[Sco92] James M. Scobbie. 1992. Towards Declarative Phonology. In Steven Bird, ed., *Declarative Perspectives in Phonology*, pages 1–27. Edinburgh Working Papers in Cognitive Science, Volume No. 7. University of Edinburgh.

[Sco93] James M. Scobbie. 1993. Issues in Constraint Violation and Conflict. In T. Mark Ellison and James M. Scobbie, eds., *Computational Phonology*, pages 37–54. Edinburgh Working Papers in Cognitive Science, Volume No. 8. University of Edinburgh.

[SCB96] James M. Scobbie, John S. Coleman, and Steven Bird. 1996. Key Aspects of Declarative Phonology. In J. Durand and B. Laks, eds. *Current Trends in Phonology: Models and Methods: Volume 2*, pages 685–709. European Studies Research Institute, University of Salford.

[Sip92] Michael Sipser. 1992. The History and Status of the P versus NP Question. In *Proceedings of the 24th Annual ACM Symposium on the Theory of Computing (STOC'92)*, pages 603–618. ACM Press, New York, NY.

[Spr92] Richard Sproat. 1992. *Morphology and Computation.* MIT Press, Cambridge, MA.

[SF99] Ulrike Stege and Micheal R. Fellows. 1999. An Improved Fixed-Parameter Tractable Algorithm for Vertex Cover. Technical Report no. 318, Department of Computer Science, ETH Zurich.

[Str94] Howard Straubing. 1994. *Finite Automata, Formal Logic, and Circuit Complexity.* Birkhäuser, Boston, MA.

[Tap97] Pasi Tapanainen. 1997. Applying a Finite-State Intersection Grammar. In Emmanuel Roche and Yves Schabes, eds., *Finite-State Language Processing*, pages 311–327. MIT Press, Cambridge, MA.

[Tes95] Bruce B. Tesar. 1995. Computational Optimality Theory. Ph.D. thesis, Department of Computer Science, University of Colorado.

[Tes96] Bruce B. Tesar. 1996. An Iterative Strategy for Language Learning. Handout for talk presented at the BCN Workshop on Conflicting Constraints, BCN Groningen, University of Groningen, the Netherlands.

[Tes97a] Bruce B. Tesar. 1997a. An Iterative Strategy for Learning Stress in Optimality Theory. In Elizabeth Hughes, Mary Hughes, and Annabell Greenhill, eds., *Proceedings of the 21st Annual Boston University Conference on Language Development*, pages 615–626. Cascadilla Press; Somerville, MA.

[Tes97b] Bruce B. Tesar. 1997b. Multi-Recursive Constraint Demotion. Technical Report ROA-197-0597, Rutgers Optimality Archive.

[TS96] Bruce B. Tesar and Paul Smolensky. 1996. Learnability in Optimality Theory (long version). Technical Report JHU-CogSci-96-3, Department of Cognitive Science, John Hopkins University. Also available as Technical Report ROA-156-1196, Rutgers Optimality Archive.

[Tso90] John K. Tsotsos. 1990. Analyzing vision at the complexity level. *Behavioral and Brain Science*, 13, 423–469.

[Tso93] John K. Tsotsos. 1993. The Role of Computational Complexity in Perceptual Theory. In Sergio C. Masin, ed., *Foundations of Perceptual Theory*, pages 261–296. North-Holland, Amsterdam.

[Tso95] John K. Tsotsos. 1995. Behaviorist intelligence and the scaling problem. *Artificial Intelligence*, 75. 135–160.

[Wal96] Markus Walther. 1996. OT-SIMPLE: A construction-kit approach to Optimality Theory implementation. Technical Report ROA-152-1096, Rutgers Optimality Archive.

[War96a] H. Todd Wareham. 1996a. The Role of Parameterized Computational Complexity Theory in Cognitive Modeling. In AAAI-96 Workshop Working Notes: *Computational Cognitive Modeling: Source of the Power*.

[War96b] H. Todd Wareham. 1996b. The Computational Complexity of Phonological Derivation in Optimality Theory. Poster, Workshop on Conflicting Constraints, BCN Groningen, Groningen, The Netherlands.

[WC80] Kenneth Wexler and Peter W. Cullicover. 1980. *Formal Principles of Language Acquisition*. MIT Press, Cambridge, MA.

[XFST] Xerox Finite State Tool. URL: http://www.xrce.xerox.com/research/mltt/fsSoft/docs/fst-97/xfst97.html.

Appendix A

Problem Index

This appendix gives the page numbers on which various decision and search problems used in this thesis are first defined. The formal definitions of decision and search problems are given at the beginning of Section 2.1.1.

Decision Problems:

Page

Decision Problems (Cont'd):

Page

Search Problems:

Page

Appendix B

The Parameterized Complexity of Simplified Segmental Grammars with Segment Deletion Rules

This appendix contains the proofs of several results given in [DFK+94] concerning problems SSG-ENCODE and SSG-DECODE when segment deletion rules are allowed. Let these problems be denoted by SSG(D)-ENCODE and SSG(D)-DECODE , respectively.

The main result is derived via a reduction from the problem below. Recall that in boolean expressions, logical AND can be represented by the product sign ($\times$) and logical OR is represented by the addition sign ($+$). A boolean expression X is said to be **t-normalized**, $t \geq 2$, if X is in product-of-sums-of-products ... -of-sums form (P_1-of-S_2-of-P_3- ... -of-S_t) if t is even, and in product-of-sums-of-products ... -of-products form (P_1-of-S_2-of-P_3- ... -of-P_t) if t is odd. For example, boolean expressions in conjunctive normal form, e.g., $(a+b)\times(c+d)$, are 2-normalized. Furthermore, define the **weight** of a truth assignment to a set of boolean variables as the number of variables assigned the value T.

WEIGHTED t-NORMALIZED SATISFIABILITY (WtNSAT)
Input: A t-normalized boolean expression X and a positive integer k.
Question: Does X have a satisfying truth assignment of weight k?

Lemma B.1 *WtNSAT* $\leq_m$ SSG(D)-ENCODE

Proof: Note that in a t-normalized boolean expression, there are t levels corresponding to evaluation of either AND or OR expressions, and that within each level, the evaluation of individual expressions can be done independently. Both of these properties are exploited in the reduction described below.

The following parameters of t-normalized boolean expressions will be used below. For a t-normalized boolean expression X, let $C_{i,j}$, $i \geq 1$, be the j-th clause on level i of X, and $C_{0,j}$ be the j-th variable encountered in reading X from left to right. For all $C_{0,j}$, define $var(C_{0,j})$ as the number of the variable $C_{0,j}$ under some numbering of the variables

$C_{i,j}$	Clause	$sub(C_{i,j})$	$\#sub(C_{i,j})$	$len(C_{i,j})$	$start(C_{i,j})$
$C_{0,1}$	a	$\emptyset$	0	1	1
$C_{0,2}$	$\overline{b}$	$\emptyset$	0	1	2
$C_{0,3}$	c	$\emptyset$	0	1	3
$C_{0,4}$	$\overline{a}$	$\emptyset$	0	1	4
$C_{0,5}$	c	$\emptyset$	0	1	5
$C_{0,6}$	$\overline{b}$	$\emptyset$	0	1	6
$C_{0,7}$	c	$\emptyset$	0	1	7
$C_{0,8}$	a	$\emptyset$	0	1	8
$C_{0,9}$	b	$\emptyset$	0	1	9
$C_{0,10}$	b	$\emptyset$	0	1	10
$C_{0,11}$	c	$\emptyset$	0	1	11
$C_{1,1}$	$(a \times \overline{b} \times c)$	$C_{0,1}, C_{0,2}, C_{0,3}$	3	3	1
$C_{1,2}$	$(\overline{a} \times c)$	$C_{0,4}, C_{0,5}$	2	2	4
$C_{1,3}$	$(\overline{b} \times c)$	$C_{0,6}, C_{0,7}$	2	2	6
$C_{1,4}$	$(a \times b)$	$C_{0,8}, C_{0,9}$	2	2	8
$C_{1,5}$	$(b \times c)$	$C_{0,10}, C_{0,11}$	2	2	10
$C_{2,1}$	$((a \times \overline{b} \times c) + (\overline{a} \times c) + (\overline{b} \times c))$	$C_{1,1}, C_{1,2}, C_{1,3}$	3	7	1
$C_{2,2}$	$((a \times b) + (b \times c))$	$C_{1,4}, C_{1,5}$	2	4	8
$C_{3,1}$	Whole Formula	$C_{2,1}, C_{2,2}$	2	11	1

Table B.1: Sample Values of Various Parameters Defined in Lemma B.1. These parameter values are computed relative to the 3-normalized Boolean Expression $(((a \times \overline{b} \times c) + (\overline{a} \times c) + (\overline{b} \times c)) \times ((a \times b) + (b \times c)))$. See main text for explanation of abbreviations.

of X, and define $not(C_{0,j})$ as $+$ ($-$) if this variable is (not) negated in $C_{0,j}$. Define $sub(C_{i,j})$ as the subclauses on level $i-1$ that are joined together to make clause $C_{i,j}$, $\#(sub(C_{i,j})$ as the number of subclauses in $sub(C_{i,j})$, and $len(C_{i,j})$ as follows:

$$len(C_{i,j}) = \begin{cases} \sum_{c \in sub(C_{i,j})} len(c) & i \geq 1 \\ 1 & i = 0 \end{cases} \tag{B.1}$$

Finally, define $start(C_{i,j}) = 1 + \sum_{1 \leq k < j} len(C_{i,k})$. The values of these parameters for the various clauses of the 3-normalized boolean expression $(((a \times \overline{b} \times c) + (\overline{a} \times c) + (\overline{b} \times c)) \times ((a \times b) + (b \times c)))$ are given in Table B.1.

Given an instance $\langle X, k\rangle$ of WtNSAT defined on n boolean variables, construct the following instance $\langle g' = \langle F', D', R', c'_p, C', f'_C\rangle, u', s'\rangle$ of SSG(D)-Encode: Let F' consist of the union of the sets of features in the following table, in which $1 \leq i \leq n$, $1 \leq j \leq k$, $0 \leq l \leq t$, and $1 \leq m \leq t$:

Denote a segment with feature-value pairs $[f_1 \ \ v_1], [f_2 \ \ v_2], \ldots, [f_k \ \ v_k]$ as $\{[f_1 \ \ v_1], [f_2 \ \ v_2], \ldots, [f_k \ \ v_k]\}$. Let $u' = u_1u_2u_3$ such that it is composed of three substrings that are distinguished by their values for feature **SEG**. The makeup of each of these substrings is as follows:

Feature	Values
SEG	$\{A, V, E\}$
VAR	$\{1, 2, \ldots, n\}$
NOT	$\{+, -\}$
$\{$**SV**$i\}$	$\{+, -\}$
$\{$**SC**$j\}$	$\{+, -\}$
$\{$**R**$l\}$	$\{T, F\}$
$\{$**SS**$m\}$	$\{+, -\}$
$\{$**EL**$m\}$	$\{1, 2, \ldots, len(C_{t,1})\}$

1. Substring u_1 ([**SEG** A]: accumulator-segment) consists of a single segment that is assigned the features $\{$[**SV**i $-$],[**SC**j $-$]$\}$, $1 \leq i \leq n$, $1 \leq j \leq k$. This substring acts as a result-accumulator which, in conjunction with substring u_2, ensures that exactly k variables in expression X are set to T.

2. Substring u_2 ([**SEG** V]: variable-segments) consists of n segments that are assigned the features $\{$[**VAR** i],[**R**0 F]$\}$, $1 \leq i \leq n$.

3. Substring u_3 ([**SEG** E]: expression-segments) consists of $len(C_{t,1})$ segments, one segment for each occurrence of a variable in X, that encode the various clauses in X and the hierarchical relations of these clauses within X. The default features assigned to each segment j, $1 \leq j \leq len(C_{t,1})$, for a t-normalized boolean expression are $\{$[**VAR** $var(C_{0,j})$], [**NOT** $not(C_{0,j})$], [**R**0 F], [**SS**i $-$],[**R**i x_i]$\}$, $1 \leq i \leq t$ and $x_i = T$ ($x_i = F$) if $(t+i)$ is even (odd). For a particular expression X, each clause $C_{i,j}$ is encoded by an additional associated set of features in expression-segments $start(C_{i,j})$ through $start(C_{i,j}) + (len(C_{i,j}) - 1)$, namely [**EL**$i$ j] (segments $start(C_{i,j})$ through $start(C_{i,j}) + (len(C_{i,j}) - 1)$) and [**SS**$i$ $+$] (segment $start(C_{i,j})$).

   A note is in order concerning the factor $(t + i)$ used to set the default values of feature **R**i. As the lowest type of clause in a t-normalized boolean expression is a product if t is odd and a sum if t is even, the clause-type for the ith level depends on t as well as i. The reader can verify that factor $(t + i)$ correctly identifies the type of clauses on level i of a t-normalized expression expression, i.e., level i consists of product (sum) clauses if $(t + i)$ is even (odd). This factor is also used in rule-set 3(a) described below to ensure the correct application of product or sum logic during the evaluation of a given boolean expression.

The segment string u produced by these feature assignments for the 3-normalized boolean expression $(((a \times \overline{b} \times c) + (\overline{a} \times c) + (\overline{b} \times c)) \times ((a \times b) + (b \times c)))$ is given in Figure B.1. Let $D' = \{u'\}$, $c_p = \max(n, \max_{i=1}^{k} \sum_{C_{i,j} \in X} \#sub(C_{i,j}) - 1)$, and $s' = \{$[**SEG** A],[**SV1** $+$],[**SV2** $+$], ..., [**SV**n $+$],[**SC1** $+$],[**SC2** $+$],...,[**SC**k $+$]$\}$ $\{$[**SEG** E],[**VAR** 1],[**R0** T],[**NOT** $+$],[**EL1 1**],[**SS1** $+$], [**R1** T],[**EL2 1**],[**SS2** $+$], [**R2** T],...,[**EL**t **1**],[**SS**t $+$],[**R**t T] $\}$. The rule-sequence R has the following structure:

1. Choose a subset of the variables in X of cardinality k and set all occurrences of these variables in expression X to T.

(a) $2k$ m.e. rule-sets, i.e. k pairs of m.e. rule-sets, in which the first rule-set of pair i, $1 \leq i \leq k$, consists of $(n-1)$ optional rules that change all occurrences of features $\{[\mathbf{VAR}\ j],[\mathbf{R}0\ F]\}$ to $\{[\mathbf{VAR}\ j],\ [\mathbf{R}0\ T]\}$, $1 \leq j \leq (n-1)$ and an obligatory rule that changes all occurrences of features $\{[\mathbf{VAR}\ n], [\mathbf{R}0\ F]\}$ to $\{[\mathbf{VAR}\ n],\ [\mathbf{R}0\ T]\}$, and the second rule-set of the pair consists of n obligatory rules that assign features $\{[\mathbf{SV}j\ +],[\mathbf{SC}i\ +]\}$, $1 \leq j \leq n$, to the accumulator-segment if feature $\mathbf{SV}j$ of the accumulator-segment has value "$-$" and feature $\mathbf{R}0$ of variable-segment j has value T.

(b) An obligatory rule that deletes all variable-segments.

(c) An obligatory rule that sets the value of all **VAR** features to 1.

(d) An obligatory rule that sets values of all features $\mathbf{SV}i$, $1 \leq i \leq n$, in the accumulator-segment to $+$.

The first rule-set of each pair i in (a) non-deterministically chooses a variable v to set to T, and the second rule-set in the pair checks that the variable has not already been chosen, i.e. $[\mathbf{SV}v\ +]$. Call the first type of rule-set the ith **variable-selection rule-set** and the second type of rule-set the ith **variable-check rule-set**.

2. Negate variable-values appropriately.

(a) An obligatory rule that changes all occurrences of features $\{[\mathbf{R}0\ F],[\mathbf{NOT}\ +]\}$ to $\{[\mathbf{R}0\ T],[\mathbf{NOT}\ -]\}$.

(b) An obligatory rule that changes all occurrences of features $\{[\mathbf{R}0\ T],[\mathbf{NOT}\ +]\}$ to $\{[\mathbf{R}0\ F],[\mathbf{NOT}\ -]\}$.

(c) An obligatory rule that sets the values of all **NOT** features to $+$.

3. For each level of X, evaluate every AND / OR expression.

The following three rule-sets perform these computations for level i, $1 \leq i \leq t$.

(a) An m.e. rule-set consisting of $p = \max_{C_{i,j} \in X} \#sub(C_{i,j})$ obligatory rules, in which rule l, $1 \leq l \leq p$, sets feature $\mathbf{R}i$ of the first segment encoding clause $C_{i,j}$, i.e. the leftmost segment with feature $[\mathbf{EL}i\ j]$, to F (T) if segment l of $C_{i,j}$ has feature $[\mathbf{R}(i-1)\ F]$ ($[\mathbf{R}(i-1)\ T]$) and $(t+i)$ is even (odd). For example, if $l = 3$ and $(t+i)$ is odd, i.e., the clauses on level i are sums (ORs), such a rule would have the form

$$\begin{bmatrix} \mathbf{EL}i & \alpha \\ \mathbf{SS}i & + \\ \mathbf{R}i & F \end{bmatrix} \rightarrow \begin{bmatrix} \mathbf{EL}i & \alpha \\ \mathbf{SS}i & + \\ \mathbf{R}i & T \end{bmatrix} \ / \ \underline{\qquad} \begin{bmatrix} \mathbf{EL}i & \alpha \\ \mathbf{SS}i & - \end{bmatrix} \begin{bmatrix} \mathbf{EL}i & \alpha \\ \mathbf{SS}i & - \\ \mathbf{R}(i-1) & T \end{bmatrix}$$

Note that by simultaneous rule application, each such rule l will evaluate each clause in level i relative to the result of its lth subclause; moreover, by the structuring of the feature **SS**i values to mark clause boundaries, no evaluation of a clause can access the results of another clause's subclause, and the rules will work correctly if clauses with different numbers of subclauses exist on the same level.

(b) An obligatory rule that deletes all segments with the features {[**EL**i α],[**SS**i $-$]}.

(c) An obligatory rule that sets the values of all features **R**$(i-1)$, **EL**i, and **SS**i to T, 1, and $+$, respectively.

Rule-set (a) simulates AND or OR clause evaluations. Note that the **R**i features in segment string u were originally encoded to default values for their respective clause-type, e.g. T for AND, F for OR. These default feature-values allow the first rule-set above to perform AND and OR computations by looking only at the results of individual subclauses in turn, i.e., change result of an AND-clause from default of T to F if any of its subclauses evaluates to F and change result of an OR-clause from default of F to T if any of its subclauses evaluates to T. Rule-set (b) deletes all but the first of the segments of each clause at level i, leaving the remaining segment to pass the evaluated result on to level $i+1$.

To complete this construction, let C' be the single control string described above and f'_C associate this control string with every member of D'^{+}. This construction can be done in time polynomial in the given instance of WtNSAT.

Consider the following proof of the correctness of this construction. By the structure of R', any application of R to u' will produce a two-segment string of the form {[**SEG** A],[**SV1** $+$],[**SV2** $+$], ...,[**SV**n $+$],[**SC1** α_1],[**SC2** α_2],...,[**SC**k α_k]} {[**SEG** E],[**VAR** 1],[**R0** T],[**NOT** $+$], [**EL1 1**],[**SS1** $+$],[**R1** T], [**EL2 1**],[**SS2** $+$], [**R2** T], ... [**EL**t **1**],[**SS**t $+$],[**R**t β]}. By the structure of the variable-check rule-sets in rule-set 1(a), feature **SC**i can have value $+$ if and only if a variable was set to T by the ith variable-selection rule-set that was not set to T by any previous variable-selection rule-set. As each variable-selection rule-set can set exactly one variable to T, each feature **SC**i, $1 \leq i \leq k$, can have value $+$ in the produced string if and only if exactly k variables in X were set to T. By the structure of rule-set 3(a), a product- (sum-) clause at level i can only have the feature-value pair [**R**i T] in its first segment if all (at least one) of its subclauses evaluated to T, i.e., had the feature-value pair [**R**$(i-1)$ T] in their first segments. Hence, feature **R**t in the produced string has value T if and only if each clause encoded in every level of the given t-normalized boolean expression was satisfied by the values assigned to the variables at level 0 for feature **R**0 – that is, if and only if the variables of the encoded expression X have a satisfying truth assignment. Hence, $g'(u') = s'$ if and only if there exists an assignment of k distinct variables to T that satisfies expression X, and the given instance of WtNSAT has a solution if and only if the constructed instance of SSG(D)-Encode has a solution.

Note that in the constructed instance of SSG(D)-Encode, $|R'| = 2kn + 6 + 2t + \sum_{C_{i,j} \in X} \#sub(C_{i,j})$, $|R'_?| = (n-1)k$, $|R'_{m.e.}| = 2k + 3t + 6$, $\#(R'_{m.e.}) = \max(n,p)$, $c' = \max(n+1,p)$, $|u'| = len(C_{t,1}) + n + 1$, $|s'| = 2$, $|f'| = 4t + n + 4$, and $|v'| = \max(n, \max_{i=1}^{k} \sum_{C_{i,j} \in X} 1)$, where $p = \max_{C_{i,j} \in X} \#sub(C_{i,j})$. ∎

Lemma B.2 *WtNSAT* $\leq_m$ SSG(D)-DECODE

Proof: Given an instance $\langle X, k\rangle$ of $\langle k\rangle$-WtNSAT defined on n boolean variables, construct an instance $\langle g' = \langle F', D', R', c'_p, C', f'_C\rangle, s'\rangle$ of SSG(D)-DECODE identical to that in Lemma B.1. Note that as no rule in R can delete an accumulator segment, i.e., a segment with the feature-value pair [**SEG** A], then any underlying form from D'^+ that can produce s' relative to R' must have exactly the same number of accumulator segments as s'. As s' has one accumulator segment, the only member of D'^+ that can possibly produce s' is u'. Hence, the given instance of WTNSAT has a solution if and only if and only if $g'(u') = s'$, and this reduction is correct for the same reasons as the reduction in Lemma B.1.

Note that in the constructed instance of SSG(D)-DECODE, all aspects have same values as for the constructed instance of SSG(D)-ENCODE in Lemma B.1. ∎

Theorem B.3 $\langle |R_{m.e.}|, |s|_2\rangle$-SSG(D)-ENCODE *and* $\langle |R_{m.e.}|, |s|_2\rangle$-SSG(D)-DECODE *are* $W[t]$*-hard for all* $t \geq 2$.

Proof: The results follow from the $W[t]$-completeness of $\langle k\rangle$-WtNSAT for $t \geq 2$ [DF92, Theorem 4.1], the reductions in Lemmas B.1 and B.2 from WtNSAT to SSG(D)-ENCODE and SSG(D)-DECODE, respectively, in which $|R_{m.e.}| = 2k + 3t + 6$ and $|s| = 2$, and Lemma 2.1.25. ∎

1	2	3	4
[**SEG** A]	[**SEG** V]	[**SEG** V]	[**SEG** V]
[**SV1** $-$]	[**VAR** 1]	[**VAR** 2]	[**VAR** 3]
[**SV2** $-$]	[**R**0 F]	[**R**0 F]	[**R**0 F]
[**SV3** $-$]			
[**SC1** $-$]			
[**SC2** $-$]			

$(((a$ $\times$	$\overline{b}$ $\times$	$c)$ $+$	$(\overline{a}$ $\times$	$c)$ $+$	$(\overline{b}$ $\times$	$\overline{c}))$
5	6	7	8	9	10	11
[**SEG** E]	[**SEG** E]	[**SEG** E]	[**SEG** E]	[**SEG** E]	[**SEG** E]	[**SEG** E]
[**VAR** 1]	[**VAR** 2]	[**VAR** 3]	[**VAR** 1]	[**VAR** 3]	[**VAR** 2]	[**VAR** 3]
[**R**0 F]	[**R**0 F]	[**R**0 F]	[**R**0 F]	[**R**0 F]	[**R**0 F]	[**R**0 F]
[**NOT** $-$]	[**NOT** $+$]	[**NOT** $-$]	[**NOT** $+$]	[**NOT** $-$]	[**NOT** $+$]	[**NOT** $+$]
[**EL1** 1]	[**EL1** 1]	[**EL1** 1]	[**EL1** 2]	[**EL1** 2]	[**EL1** 3]	[**EL1** 3]
[**SS1** $+$]	[**SS1** $-$]	[**SS1** $-$]	[**SS1** $+$]	[**SS1** $-$]	[**SS1** $+$]	[**SS1** $-$]
[**R1** T]	[**R1** T]	[**R1** T]	[**R1** T]	[**R1** T]	[**R1** T]	[**R1** T]
[**EL2** 1]	[**EL2** 1]	[**EL2** 1]	[**EL2** 1]	[**EL2** 1]	[**EL2** 1]	[**EL2** 1]
[**SS2** $+$]	[**SS2** $-$]	[**SS2** $-$]	[**SS2** $-$]	[**SS2** $-$]	[**SS2** $-$]	[**SS2** $-$]
[**R2** F]	[**R2** F]	[**R2** F]	[**R2** F]	[**R2** F]	[**R2** F]	[**R2** F]
[**EL3** 1]	[**EL3** 1]	[**EL3** 1]	[**EL3** 1]	[**EL3** 1]	[**EL3** 1]	[**EL3** 1]
[**SS3** $+$]	[**SS3** $-$]	[**SS3** $-$]	[**SS3** $-$]	[**SS3** $-$]	[**SS3** $-$]	[**SS3** $-$]
[**R3** T]	[**R3** T]	[**R3** T]	[**R3** T]	[**R3** T]	[**R3** T]	[**R3** T]

$\times$

$((a$ $\times$	$b)$ $+$	$(b$ $\times$	$c)))$
12	13	14	15
[**SEG** E]	[**SEG** E]	[**SEG** E]	[**SEG** E]
[**VAR** 1]	[**VAR** 2]	[**VAR** 2]	[**VAR** 3]
[**R**0 F]	[**R**0 F]	[**R**0 F]	[**R**0 F]
[**NOT** $-$]	[**NOT** $-$]	[**NOT** $-$]	[**NOT** $-$]
[**EL1** 4]	[**EL1** 4]	[**EL1** 5]	[**EL1** 5]
[**SS1** $+$]	[**SS1** $-$]	[**SS1** $+$]	[**SS1** $-$]
[**R1** T]	[**R1** T]	[**R1** T]	[**R1** T]
[**EL2** 2]	[**EL2** 2]	[**EL2** 2]	[**EL2** 2]
[**SS2** $+$]	[**SS2** $-$]	[**SS2** $-$]	[**SS2** $-$]
[**R2** F]	[**R2** F]	[**R2** F]	[**R2** F]
[**EL3** 1]	[**EL3** 1]	[**EL3** 1]	[**EL3** 1]
[**SS3** $-$]	[**SS3** $-$]	[**SS3** $-$]	[**SS3** $-$]
[**R3** T]	[**R3** T]	[**R3** T]	[**R3** T]

Figure B.1: A Lexical String Produced by the Reduction in Lemma B.1. The formula encoded above is the 3-normalized boolean expression $(((a \times \overline{b} \times c) + (\overline{a} \times c) + (\overline{b} \times c)) \times ((a \times b) + (b \times c)))$ under the variable-encoding $var(a) = 1$, $var(b) = 2$, and $var(c) = 3$.