From alternations to ordered rules: A system for learning Derivational Phonology

Marc Simpson

A Thesis
in
The Special Individualized Program

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Arts (Special Individualized Program) at
Concordia University
Montreal, Quebec, Canada

March 2010

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Marc Simpson

Entitled: From alternations to ordered rules:
A system for learning Derivational Phonology

and submitted in partial fulfillment of the requirements for the degree of

Master of Arts (Special Individualized Program)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the the final examining committee:

______________________________ Chair

______________________________ Examiner

______________________________ Examiner

______________________________ Supervisor

Approved by ______________________________
Chair of Department or Graduate Program Director

_____ 20 _ ______________________________
Dean of Faculty

Abstract

From alternations to ordered rules: A system for learning Derivational Phonology

Marc Simpson

This work presents a computational rule learner tasked with inferring underlying forms and ordered rules from phonological paradigms akin to those found in traditional pen and paper analyses. The scheme being proposed is a batch learner capable of analysing surface alternations and hypothesising ordered derivations compatible with them in order to create an explicit mapping from UR to SR.

We shall refer to both the competence of an idealised speaker-hearer (in keeping with traditional generative linguistic theory) and the conscious methods employed by the phonologist in the course of analysing data sets.

The fundamental axiom of this model is that the child has memorised the relevant surface forms (as they appear in the paradigm) alongside the appropriate semantic information in order to allow them to set up paradigmatic structures for the purpose of inferring both underlying forms and phonological rules simultaneously. The mapping from minimal pairs to underlying forms is the primary conduit to inferring the rules themselves.

A full code listing for the rule learner, written in Prolog, is available.

Contents

Chapter 1

Foundations

As a means of introducing the topic of automated phonological inference, it is worth returning to the foundational principles of phonology beginning with the theoretical motivation for our conception of *representations* and the *rules* that map between them. Having established such a foundation, we will then be able to theoretically and formally ground the computational model at the heart of this thesis.

1.1 Rejecting the null hypothesis

The context for any discussion of phonological grammar is the (common-sense) notion that the pronunciations of words are simply lexicalised—memorised—as they are acquired. While this notion fails to stand up to scrutiny and is, indeed, the position against which generative phonological theory is situated, it is worth clearly stating before proceeding with a discussion of approaches to *distributional regularity* and *productivity* in the phonological component of mental grammar.

A succinct but informative discussion of the *null hypothesis* can be found in one of the seminal texts of modern phonology, Kenstowicz & Kisseberth's *Generative Phonology*:

> There is no principle of English phonology which predicts that the word *cab* is composed of three sounds *k, æ, b*, appearing in this particular order. In order to know how to pronounce this word, one must simply wait to hear it pronounced by someone else who already knows its pronunciation. We thus might assume that once the native speaker hears a given word pronounced, he simply memorizes what sounds make up the word and in what order those sounds are pronounced. This information would be stored in the lexicon of the speaker's grammar. [. . .]
>
> Thus no phonological component of the grammar would be required: The syntactic rules, which specify the order of the morphemes in the sentence, and the lexicon, which specifies the pronunciation of each morpheme, would jointly produce a pronunciation for each sentence. Let us refer to this view of pronunciation as the **null hypothesis**. (Kenstowicz & Kisseberth, 1979, p. 26)

Of course, the past fifty years of research into the human language faculty directly contradict this null hypothesis—be it in the syntactic context where data from binding and movement provide compelling evidence for a generative aspect to human linguistic cognition, or in (morpho-)phonology where distributional regularities are observable (and analysable through the interaction of simple abstractions of sound-changing processes) and empirically verifiable through simple tests of linguistic competence (Berko, 1958). Hale and Reiss (2008) present four traditional arguments against the null hypothesis and for generativity as it relates to the phonological component of grammar. These are: distributional regularities, evidence from alternations, the *wug* test (mentioned above) and borrowing (loanwords).

Distributional regularities are the currency of phonology: they are the regular, tacit sound patterns found in a particular language. Take, for example, the case of flapping in North American English where *t* and *ɾ* are *allophones* in complementary distribution. When *t* appears underlyingly between two vowels where the second is unstressed, *t* surfaces as a flap. For example:

(1.1)

UR	atom	atom-ic
SR	a[ɾ]om	a[t]omic

Do we wish to say that the speaker has stored two distinct pronunciations (a[ɾ]om and a[t]omic) or should we offer a more principled explanation of this distinction—starting with the predictable application of this alternation? The answer chosen by generative phonologists—due to the compelling regularity of such patterns—is the latter. This case of allophony can be captured by a simple flapping rule,

$$\text{t} \rightarrow \text{ɾ} \ / \ \text{V} \ \_\_ \ \text{V}_{[-stress]}$$

which is accompanied by a prediction: if the environment for this rule is met in a different form (that is, a form in which *t* appears between two vowels where the second is unstressed), including those novel or borrowed, flapping should still be evidenced. Put differently: we expect this rule to apply every time its environment is met and that application is not contingent on the current list of forms stored in the speaker's lexicon.

1.1.1 Neutralisation

Inferring underlying representations from alternating surface forms can be tricky. Consider the following data set from a toy language:

(1.2)

	Surface Form	Gloss
a.	gap	a dog
b.	gaba	some dogs
c.	gab	a cat
d.	gaba	some cats

This set of forms exhibits neutralisation—forms that are stored differently underlyingly can be pronounced identically (evidenced by 'some dogs' vs. 'some cats'). The question raised for the phonologist analysing such a data set as well as for the child attempting to acquire the correct underlying forms is the following: how to figure out the correct underlying representation for 'dog' and 'cat'.

To help solve this problem, we shall propose the two principles provided in (1.3):

(1.3)

1. If a form does *not* alternate on the surface, infer that the underlying form is the same as the surface representation;
2. If a form *does* alternate on the surface, on the other hand, the underlying form should be selected from one of the surface alternants.
    - In order to select the correct surface alternant, reference must be made to other forms that have already been inferred.

To illustrate, let us apply these principles to an analysis of the form in (1.2). The first condition states that where a form fails to alternate, we can infer that to be the underlying form. Notice that whereas the second consonant in 'dog' exhibits a $p \sim b$ alternation the surface forms for 'cat' are consistent. Principle (1) tells us that the root for 'cat' should be inferred as *gab*. Turning to principle (2) and 'dog', it is clear that we need to pick between 'gap' and 'gab'. If we choose the same form, *gab*, for 'dog' we encounter a problem. In identical environments, identical underlying forms surface differently. This is logically inconsistent and results in an incorrect inference for 'dog'. If we instead select *gap*—a step still compatible with (2)—we no longer run into this problem; what remains to be shown are the rules governing the $p \sim b$ alternation itself.

Having chosen our underlying representations, we can propose phonological rules to generate the correct surface forms. Where there is no alternation, nothing needs to be accounted for and so we can turn immediately to the form for 'dog', *gap*. This yields the following situation,

(1.4)

	Underlying Form	Rule	Surface Form
a.	gap	–	gap
b.	gap-a	?	gab-a

A number of compatible rules can be proposed; for the sake of simplicity we shall offer just three:

1. /p/ → [+voice] / __ a

2. /p/ → [+voice] / a __

3. /p/ → [+voice] / a __ a

Since the underlying form has been inferred as *gap*, we need a rule to rewrite $p \rightarrow b$; here we have already gone one step further and proposed a featural analysis—p is voiced either (i) before a, (ii) after a or (iii) between as. We could extend this still further by claiming that the voicing environment is *intervocalic* and that specifying this environment as a is too specific. This is not a question we shall be dwelling on, however; the central concern here is sketching the problem of neutralisation, demonstrating a set of principles for proceeding with an analysis and outputting a set of possible rules. This is exactly the function of the rule learner that we shall be proposing below.

Given the above discussion it should be clear that the task of the theoretical linguist interested in acquisition of language and computation over linguistic space is to provide an appropriate account both for the primitives over which computation takes place and the processes that employ them. Our goal is to show that neutralisation problems like the one posed above can be solved by a system that operates according to the principles in (1.3).

1.2 Situating phonology biolinguistically

Once the null hypothesis has been rejected along the grounds outlined in (1.1), our next step has to be an inquiry into the ontology of phonological representation and transformation: what is the nature of stored forms and how do these relate to the distribution of sounds in a given language? What sort of model elegantly captures what we know about language?

1.2.1 Substance-free phonology

This project is underpinned by a commitment to a *substance-free* approach to phonology.

> We propose that it is useful to conceive of a grammar as a relationship between (a) a set of symbols (entities like features and variables; constituents like syllables, feet, NPs) and (b) a set of computations (operations whose operands are drawn from the set of symbols, such as concatenation and deletion). The issue of substance arises only with respect to the set of symbols, and for the sake of simplicity we restrict ourselves to the set of phonological primitives known as distinctive features and to the rep-

> resentations that can be defined as combinations of distinctive features. (Hale & Reiss, 2000, p. 157)

The implications of this methodology are as follows,

> The conclusion we wish to draw from the above examples and many others like them is that the best way to gain an understanding of the computational system of phonology is to assume that the substance of phonological entities is *never* relevant to how they are treated by the computational system, except in *arbitrary, stipulative* ways. . . . If our goal as generative linguists is to define the set of *computationally possible* human grammars, "universal tendencies" are irrelevant to that enterprise. We propose extending the Saussurean notion of the arbitrary nature of linguistic signs to the treatment of phonological representations by the phonological computational system. Phonology is not and should not be grounded in phonetics since the facts that phonetic grounding is meant to explain can be derived without reference to *phonology*. (Hale & Reiss, 2000, p. 157)

We shall return to this lack of phonetic grounding once our scheme has been presented further.

1.3 I-Language, Features and UG

By focusing on *I-Language* as our object of study—and bearing in mind the points already addressed regarding substance (or rather its lack) in phonological formalisms—we now need to concretise the fundamental theoretical principles at play before addressing the question of how we might go about instantiating these principles in a computational model for the purpose of testing and exploring their implications.

1.3.1 Parsing and assumptions

We assume that the child has the already parsed acoustic input into featural representations; the capacity to do so can be attributed to properties of UG whereby raw acoustic signals can be transduced and mapped onto a discrete selection of binary features.

As such, representations are supposed to be richly specified relative to the base feature set (formally: arbitrary in nature). Consequently our approach begins with surface forms that can be compared with respect to their feature specifications rather than necessitating a comparison first on a phonemic level.

1.4 Optimality Theory: Divergence

The dominant mode of phonological theorising and grammar construction since the mid-1990s has been Optimality Theory (OT) built on the original formulation in Prince & Smolensky (1993).

OT differs from Derivational Phonology in two important respects. Firstly, Optimality Theory favours parallel computation for the evaluation of surface forms. Secondly, the OT model of computation in both the parsing and generation phases is fundamentally *constraint*-driven rather than process-orientated.

1.5 Modelling

Before proceeding with a discussion of the software being presented in this thesis—which deals with both phonological acquisition and the nature of computation once this acquisition phase is complete—it is first necessary to understand what is meant by a computational model, how it relates to phonological formalism and what, exactly, the use of an implementation is with respect to developing phonological theory further. These questions hinge on how one goes about proceeding with an abstract and formal science of mind—in other words, the operating procedures of cognitive science itself. Are we required to make claims about the neural or psycholinguistic aspect when constructing a theory of phonology? Does learning need to be informed by developments in neuroscience?

By widening our linguistic perspective to include results from the field of vision, we can address these questions by referencing David Marr's work on the cognitive foundations of visual competence. This is one of the clearest discussions of the relationship between different levels of theory construction, analysis and implementation.

1.5.1 The world according to Marr

In his work on vision, Marr outlines three information-processing levels. This discussion is worth repeating in full for it serves to clarify the context in which building an automated rule learner falls—both in relation to theoretical work and recent trends in automating learning through the use of neural networks in Optimality Theory.

> We can summarize our discussion in something like the manner shown in [the table in (1.5)], which illustrates the different levels at which an information-processing device must be understood before one can be said to have understood it completely. At one extreme, the top level, is the abstract computational theory of the device, in which the performance of the device is characterized as a mapping from one kind of information to another, the abstract properties of this mapping are defined precisely, and its appropriateness and adequacy for the task at hand are demonstrated. In the center is the choice of representation for the input and output and

> the algorithm to be used to transform one into the other. And at the other extreme are the details of how the algorithm and representation are realized physically—the detailed computer architecture, so to speak. These three levels are coupled, but only loosely. The choice of an algorithm is influenced for example, by what it has to do and by the hardware in which it must run. But there is a wide choice available at each level, and the explication of each level involves issues that are rather independent of the other two. (Marr, 1982, p. 24–5)

The figure that Marr mentions is reproduced below as (1.5),

(1.5) Levels of analysis, from Marr (1982)

Computational theory	**Representation and algorithm**	**Hardware implementation**
What is the goal[1] of computation, why is it appropriate and what is the logic of the strategy by which it can be carried out?	How can this computational theory be implemented? In particular, what is the representation for the input and output, and what is the algorithm for the transformation?	How can representation and algorithm be realized physically?

The bulk of our project relates to the middle column in Marr's diagram—the representational stage. Building on a theory of computation already present in the theoretical literature, the task presented to the linguist writing a learning system such

[1] That is, 'performance of the device is characterized as a mapping from one kind of information to another, the abstract properties of this mapping are defined precisely and its appropriateness and adequacy for the task at hand are demonstrated', Ibid.

as this is that of finding a way to *algorithmically compute over the representational space described by the formalism.* This process is not unidirectional, however. The end goal of building a learning model is *reflexive*, to remark on the original computational theory—in attempting to find the most elegant algorithms and selecting the most appropriate data structures for phonological representations, we can revise our understanding (and propose new theories and processes) at the computational level itself.

This sort of approach does not say anything about the *hardware* layer—rather, it begins with the pseudo-code outlined by the computational theory and translates it into the specification for a *virtual machine.* Such reticence in this regard is, we think, theoretically justified—especially in light of the alternative approach which is currently dominant in the phonological field.

Optimality Theoretic models seek to unify all three columns, providing an account of phonological competence that encompasses hardware, algorithm and mapping abstractions as made clear by Smolensky (1999):

> Fundamental defining principles of OT, and their relation to connectionism. . .
>
> **a.** *Optimality.* The correct output representation is the one that maximizes Harmony.
>
> **b.** *Containment.* Competition for optimality is between outputs that include the given input. (Clamping the input units restricts the optimization in a network to those patterns including the input.)
>
> **c.** *Parallelism.* Harmony measures the degree of simultaneous satisfaction of constraints. (Connectionist optimization is parallel: the constraints encoded in the connections all apply simultaneously to a

potential output.)

d. *Interactionism.* The complexity of patterns of grammaticality comes not from individual constraints, which are relatively simple and general, but from the mutual interaction of multiple constraints. (Each connection in a network is a simple, general constraint on the coactivity of the units it connects; complex behavior emerges only from the interaction of many constraints.)

e. *Conflict.* Constraints conflict: it is typically impossible to simultaneously satisfy them all. (Positive and negative connections typically put conflicting pressures on a unit's activity.)

f. *Domination.* Constraint conflict is resolved via a notion of differential strength: stronger constraints prevail over weaker ones in cases of conflict.

g. *Minimal violability.* Correct outputs typically violate some constraints (because of e), but do so only to the minimal degree needed to satisfy stronger constraints.

h. *Learning requires determination of constraint strengths.* Acquiring the grammar of a particular language requires determining the relative strengths of constraints in the target language.

(Smolensky, 1999, p. 597)

In OT, parallelist connectionist networks provide the bedrock over which higher level abstractions are computed: a connectionist network *is* the brain in these models. The representational and algorithmic level involves an approach to mapping the constraint space, ranking ordering techniques and surface form generation modules onto the hardware and as such makes claims both about the neurological nature of abstract phonological computations and how abstractions are learned. We say that these abstracts have been *reduced* to their neural correlates. The final computational level is the daily bread of OT phonological analysis, and is usually represented in the form of a *tableaux*. The presentation of a phonological alternation will have, for better or worse, a well defined relationship to both the algorithmic and hardware layer.

We instead adopt Marrian modularity; hypothesising that the relationship between linguistic computation and its algorithmic realisation in no way necessitates (or implies) an understanding of the underlying hardware. Put another way: neural nets may well be the best possible mechanism for modelling neural structure—this project, however, sits at a higher level of abstraction and as such has nothing to say about the brain or how the virtual machine relates to neural pathways. This naturally introduces the question of which approach is more desirable; from the brief presentation above it certainly seems that an approach encompassing all three levels is richer than the modularity we have adopted.

This is not so. Two considerations are important here. The first is that neural (connectionist) networks do not, in any empirical way, model the brain. Instead these neural network models rely on an *analogy* between simple data structures (labelled as 'neurons') and neurological structure. There is no doubt that these networks are oversimplifications—but oversimplifications are acceptable in the development of a science and working with simplified models can be productive. The real problem is this: any attempt to build a neural analogy into a theory of abstract linguistic computation ignores the simple fact that *we do not yet understand the mapping from abstract reasoning to neural matter.* While the Optimality Theorist can claim that, starting from simple foundations, he can model a phonological grammar, this is still contingent on the arbitrariness of the connectionist platform itself.

Modularity affords us the possibility to understand one domain well using a set of well-defined primitives; later, these primitives can be analysed and refactored into functionality that interfaces with lower levels of the cognitive architecture in the Language Faculty.

1.6 Why Derivational Phonology?

It is not the purpose of this thesis to directly address the debate between Derivational and Optimality Theoretic approaches to phonology and come out in favour of one or the other—such a task is too large even for a work of this length that is wholly dedicated to the subject. Our task is more modest; in this section we situates the debate between serialist and parallelist approaches to representing the Language Faculty—and phonology in particular—in order to justify and contextualise the learning task that we have set for ourselves.

In his *Manifesto*, Vaux (2007) addresses the (supposed) theoretical advances in OT models by revisiting the original motivating arguments and assessing whether their validity is well justified. The key concern is well summarised in the following passage:

> I am not aware of any serious attempt by an optimologist to explicitly examine or falsify a DP [Derivational Phonology] analysis. This is not surprising, given that none of the points in (1) [empirical results, generality of scope, parsimony, markedness, connectionism, factorial typology, conspiracies, MSCs, problems with rules and levels, gradient wellformedness, backcopying, . . . learnability] actually poses a legitimate problem for DP. Space constraints prevent me from discussing all of the points in (1) here; in what follows I focus on those that are mentioned most frequently in

> the OT literature.
>
> (Vaux, 2007, p. 2–3)

Vaux goes on to argue for serialist models and against OT, assessing the viability of each model as it relates to the list of categories outlined at the beginning of the paper *and* by re-examining the viability of a subset of these categories (e.g., typology, markedness) as integral parts of phonology theory.

Perhaps most the most salient topic in the manifesto is the comparison of acquisition and generalisation in each theory. Vaux argues the following:

> **1.4. Acquisition as generalization formation**
>
> Finally, OT misses the fact that grammar construction is driven by the extraction of generalizations from the data to which the learner is exposed. These generalizations are encoded directly in rules and inviolable constraints, whereas OT is forced to simulate their effects via complicated constraint rankings, which in turn can only be arrived at after comparing the outputs of an equally complicated array of competing rankings. In this sense the learning strategy employed in DP is formally simpler than what is required in OT, and more insightfully captures our intuitions concerning the nature of the acquisition process.
>
> (Vaux, 2007, p. 6)

While the meaning of *formal simplicity* is ambiguous (computational complexity or generative capacity?), the essence of the point—that DP models have an intuitively simple acquisition model involving learning well-factored implications and combining them to yield ordered derivations—is important for what shall follow. From this simple model, phonological effects such as opacity come free with no need for additional computational machinery as in OT. This is not to say that OT is undesirable simply because it is complex; rather, we are suggesting that certain phenomena might be better explained in a derivational context if they are contingent on other well motivated properties of that model.

As such: if we accept Vaux's style of argumentation and focus solely on the acquisition problem for serialist models we could, in theory, see excellent returns on our investment as our implementation would, by definition, deal with problematic cases such as opacity *gratis*.

Chapter 2

Learning Phonology: Previous Approaches

2.1 Just *what* are we trying to learn?

Put baldly, the task of our learner is to infer underlying forms and ordered rules from phonological paradigms akin to those found in traditional pen and paper analyses. As such, the scheme that we are proposing can be considered a *batch learner* capable of analysing surface alternations and hypothesising ordered derivations compatible with them in order to create an explicit mapping from UR to SR.

This approach, then, shall refer to both the competence of an idealised speaker-hearer (in keeping with traditional generative linguistic theory) and the conscious methods employed by the phonologist in the course of analysing data sets. Ambiguity for the learner will mean that a linguist applying the same methodology will have to rely on other heuristics for coming to a satisfactory solution and that a speaker-hearer employing the method being outlined will either be unable to make a satisfactory inference or apply methods outside of the purview of our scheme.

In order for these tasks to be coherent, then, the fundamental axiom of our model is that the child has memorised the relevant surface forms—forms heard by the child and stored as sequences of phonological segments—along with the appropriate semantic information in order to allow them to set up paradigmatic structures for the purpose of inferring both underlying forms and phonological rules simultaneously. Indeed, the mapping from minimal pairs to underlying forms is the primary conduit to inferring the rules themselves. Furthermore, the speaker-hearer must have the capacity to structure these forms in such a way that minimal differences can be analysed and the rules that account for variations can be inferred. We shall term the structure in which variant surface forms are situated the *paradigm*.

A further stipulation of our model, in keeping with the Derivational tradition, is that the output of such an analysis—ordered rules—is the final state of the phonology, not a precondition for building constraint rankings in an Optimality Theoretic framework. As we shall see below in the discussion of the Albright & Hayes' (1999) approach, this distinguishes our model from contemporary efforts to employ rules as an intermediate phase in an eventual transition to ranked constraints. As outlined in the first chapter, we wish to render phonological competence explicit in a Derivational light, paying particular attention to complex phenomena that Optimality models still struggle to compute—in particular, opacity. It is not the purpose of this thesis to argue for Derivational phonology and against the OT framework; rather, we demonstrate the results that can be obtained in the Derivational context by applying a simple set of heuristics for analysis and inference. As such, where Albright and Hayes use rules as they are an effective learning strategy only to abandon them due to a commitment to OT, we believe that discrete rules inferred in the course of learning are an end in themselves.

We hope to achieve all of the above goals with an elegant and simple model capable of contributing to further research into explanatorily adequate models of human competence and language acquisition.

2.2 Phonotactics, stress, underlying forms

2.2.1 Bobrow & Fraser (1968). A Phonological Rule Tester

Bobrow and Fraser's scheme is of historical interest not for its learning capabilities but rather for being one of the first attempts to computationally model the *generative* phonological system of SPE (Chomsky & Halle, 1968). Especially noteworthy is a point that they make regarding the multiplicity of rule types in the Derivational model:

> We distinguish three types of phonological rules within the system: a simple rule, an insertion rule, and a string rule. It is convenient to think of each rule as consisting of a left-hand side (LHS), which specifies the condition on the substring to be altered; a right-hand side (RHS), which specifies the change to be made; and a context, which specifies the environment in which the substring matched by the LHS must be located. (Bobrow & Fraser, 1968, p. 768)

While this distinction points to an important fact about generative capacity—that deletion and insertion are qualitatively different from feature changing rules in that they (i) change the length of the representation and (ii) manipulate feature matrices as a whole rather than operating over a subset of featural specifications—it seems more convenient simply to view phonological rules as string re-writing systems.

Even so, the point made above by Bobrow and Fraser carries over to our scheme in which deletion and insertion have to be treated specially—and require preprocessing to address problems of correspondence.

2.2.2 Albright & Hayes (1999): An Automated Learner

Albright and Hayes (1999, hereafter AH) open their paper with a discussion of the relationship between the computational implementation of their phonological learner and the role of the linguist:

> In this approach, phonological analyses are not invented by linguists, but must be learned by algorithm, from a representative set of input data. The linguist is responsible for the algorithm, and the algorithm models the language through the grammar it learns. In this way, every analysis produced would address learnability directly, since the algorithm that generates the analysis in effect forms the learnability theory for that analysis. (Albright & Hayes, 1999, p. 2)

We find ourselves in accord with this statement; the natural question, then, is what the underlying axioms of their model are and whether they are appropriate for a theory of acquisition.

The AH model is predominantly *morphological*; as such, much of the content of their paper cannot be readily translated into the tasks that face our learning problem. Default mappings, confidence metrics (through *reliability* criteria) and hypothesis growth all play central roles in their scheme. Phonology enters the picture through morphological guesswork:

> A crucial part of our Learner is that it attempts to "bootstrap" the phonology and the morphology off of one another. Tentative guesses about morphology spawn guesses about phonology. Then, with some phonology in place, the performance of the morphological system is improved. Our specific strategy is to take highly trustable mappings, and try applying them

> to *all* stems meeting their structural description. For many stems this obtains the right answer, but for others the answer is wrong. The reason, very often, is that the system needs to take phonology into account. (Albright & Hayes, 1999, p. 6)

Well-formedness in the AH model

The main problem for this approach (according to Albright and Hayes) is knowing why certain sequences are ill-formed. They provide two possible solutions:

1. Scanning the 'language in general' for sequences exhibited by incorrect outputs;
    - Absence of a sequence—in conjunction with evidence of a 'repair' process—is a good indicator of ill-formedness.
2. Learning phonotactic well-formedness through an independent module and feeding this information into the initial state of the bootstrapping learner.

(Albright & Hayes, 1999, p. 6)

From our vantage point, there are two problems with these solutions. The first is that a language-wide scan is necessary for ascertaining ill-formedness; far more desirable would be a model that employs local processes to account for learning *incrementally*. Models that propose multiple sweeps over large quantities of data in order to ascertain local processes might make sense from a computational perspective (i.e., data mining) but are suspicious when proposed as models of human linguistic competence. This is related to a rather confusing point that Albright and Hayes make, namely:

> Our view is that it would be worthwhile for at least a subset of phonological theorists to model speakers instead of languages.

> (Albright & Hayes, 1999, p. 2)

It is unclear exactly what is meant by *languages* in this context. It seems that Albright and Hayes are employing the set theoretic definition of language (as a set of strings)—or perhaps they are referring to E-language and data mining of corpora. Whatever the case, this sort of statement is vacuous once one recognises that the object of study for theoretical linguistics is I-language: in proposing a computational learner for phonology, one seeks to model the competence of speaker-hearers. This is the only coherent way of understanding what it means for a theoretician to model language. As Chomsky clarifies,

> Taking knowledge of language to be a cognitive state, we might construe the "language" as an abstract object, the "object of knowledge", an abstract system of rules and principles (or whatever turns out to be correct) that is an image of the generative procedure, the I-language, represented in the mind and ultimately in the brain in now-unknown "more elementary" mechanisms.
>
> (Chomsky, 2000, p. 73)

If we wish to understand the patterns and regularities in the phonology (and more broadly, across other linguistic systems—syntax, semantics, etc.) it makes little sense to focus on string sets and corpora when the generative aspect of language use is situated in the mind of the speaker-hearer; it is an object of knowledge.

The second problem (again, from our perspective) is a reliance on phonotactics to aid the learning process. We would rather demonstrate a learning model that relies on the acquisition of forms as strings of segments that are then *locally* compared to yield rules through observed alternation.

Other Issues

A further point of difference from our approach is the exemplar theoretic foundations of the AH model. Albright and Hayes explicitly state that

> Our Learner does not reduce stems to a single underlying form; rather, it rationalizes the paradigm by finding how (and to what extent) paradigm members can be projected from one another. Speakers are assumed to memorize not underlying forms, but rather at least enough surface forms of each stem so that missing paradigm elements can be projected from what is already known (Burzio 1996; Hayes, in press). (Albright & Hayes, 1999, p. 7)

We do not employ such projection between paradigm members, nor the assumption that speakers fail to memorise underlying forms. Rather, the batch learning model under consideration here begins with a glossed set of paradigms and takes the learner from surface forms to underlying forms. The rules inferred in the course of analysis can then be directly employed for the generation task yielding phonologically conditioned surface representations from entries in the speaker-hearer's lexicon.

2.2.3 Tesar and Smolensky (2000): Learnability in OT

Tesar and Smolensky (2000) (hereafter, T&S) provide a learning model for stress rather than segmental alternation. Given the Optimality Theoretic paradigm, the hope is that it can be generalised to the ranking of other OT constraints too, hence this discrete and specific focus.

A goal of the T&S learner is to constrain the possible search space and prevent combinatorial search among $N!$ possible grammars (given N constraints). They argue that…

> . . . to achieve meaningful assurance of learnability from our grammatical theory, we must seek evidence that the theory provides the space of possible grammars with the *kind of structure* that learning can effectively exploit. (Tesar & Smolensky, 2000, p. 3)

Tesar and Smolensky make frequent reference to the link between metrical acquisition and Principles and Parameters referencing, among other work, the *triggering learning algorithm* of Gibson and Wexler (1994). This is indicative of the divergence of our approach from theirs. Firstly, acquisition of phonological rules clearly differs from establishing a constraint ranking as per Optimality Theory's methodology. Secondly, we consider stress acquisition (and stress computation) to require additional computational machinery and draw a strict demarcation between the acquisition of underlying forms and phonological rules on the one hand and stress rules on the other. (See for example Idsardi, 1992.) Finally, we have no need to refer to parametric properties of Universal Grammar in order to establish the phonological rules and processes at work in language-specific grammars. Rather, we assume a universal feature inventory and discrete, well-defined cross-linguistic operations for mapping between underlying forms and surface forms.

The point that T&S make in the above quotation, however, applies equally to Derivational and Optimality Theoretic models: ranking of constraints and ordering of rules both require an $N!$ search space. This has been presented as a fundamental problem for Derivational Phonology (and even an argument against it); we disagree. Since the rules posited in Derivational models are language-specific, N will typically be much smaller than in OT. In section (4.7.1) we shall propose a system involving incremental checks of inferred rules as well as a final attempt to *permute* the resultant ordered set as much as possible providing all possible plausible orderings.

2.2.4 Tesar (2006): Learning from Paradigmatic Information

A good discussion of the importance of the paradigm in phonological learning is provided by Tesar (2006) where paradigmatic learning is contrasted with approaches dominated by phonotactics—

> Paradigmatic information is of interest here because of its role in phonological learning. It stands in addition to purely phonotactic learning, in which each word is treated as isolated and monolithic. Phonotactic learning has been characterized as a stage in which the learner has no awareness of word-internal morphological structure, and no knowledge of shared morphemes across words (Hayes 2004, Prince and Tesar 2004). In other words, phonotactic learning does not make use of paradigmatic information. Phonotactic information, under this characterization, consists of the observed inventory of surface word forms. (Tesar, 2006, p. 293)

Furthermore,

> Paradigmatic information is necessary for learning; there are aspects of phonological systems which are not revealed through phonotactic information alone. (Tesar, 2006, p. 293)

The necessity of non-phonotactic information in the learning task is demonstrated through the presentation of two toy languages with identical phonotactics but very different phonologies. These languages are differentiated by their default stress—the first defaulting left, the second right (Tesar, 2006, p. 295). In presenting his scheme, Tesar draws attention to the necessity of breaking information into subsets of the overall domain in order to construct appropriate learning algorithms—and that the choice of subsets involves a compromise between informational redundancy and computational efficiency. Regarding the content of the sets, he writes,

> If the learner is going to focus processing on data subsets, it will want to construct subsets that group together forms that are inter-related in revealing ways. . . . If the learner processes a data subset by trying many different possible underlying forms for each of the morphemes in the data subset, then the required computational effort can be expected to increase significantly for data subsets with larger numbers of morphemes (other things being equal) (Tesar, 2006, p. 296).

As such, there is a trade-off in this model—one that we might paint along an axis of aptness vs. efficiency[1]. An example of an (undesirable) imbalance would be concentrating on individual tokens; processing can be undertaken efficiently but too little information is provided to the learner. Instead we ought to focus on the paradigm where enough context is provided for the learning task to be initiated. It is difficult to see how this tension could be related back to the acquisition process itself; where Tesar seems to allow for two situations—efficient but questionable inference and inefficient but rigourous learning—we would rather rely on a well defined process that steps through all provided data and makes inferences according to well defined principles.

As indicated in (§2.1), our starting point is the paradigm—we do not make use of phonotactics in the task of acquiring a phonological grammar. Put differently, our model attempts to learn phonologies without referencing phonotactics. Tesar, by contrast, *does* rely on phonotactic information to aid in the learning task:

> While phonotactic information is insufficient to determine the entire grammar, it can determine parts of the grammar. Phonotactic learning here

[1] Tesar does not use the term 'aptness' in this paper, though it seems appropriate given the content of his discussion.

> involves treating each word in isolation, as if it were monomorphemic. [. . .] The phonotactic inventory of [a language] determines some ranking relations. . . . [These] are necessary to map the observed forms to themselves. (Tesar, 2006, p. 297–8)

Our research question is more fundamental: given surface alternations, how successfully can underlying forms be learned?

Unset Features in Underlying Forms

A further difference from the Tesar model has to do with feature specifications in the course of learning. Tesar employs a special symbol—'?'—to denote 'a feature that as *not yet* been set by the learner' (Tesar, 2006, p. 299). This is not a means for representing underspecified segments; rather, it acts as a placeholder for later specification inferred in the course of analysis.

Our alternation approach—sketched in some detail in (§4.3.1)—instead finds *derivation paths* between segments exhibiting alternation. As such, nothing is added to the lexicon without a coherent featural specification. While this might seem like a trivial difference, it in fact exhibits a very different approach to rule and underlying form acquisition; Tesar treats the lexicon as a store for ongoing inference, whereas we provide no such structure. By contrast, we encode data to be learned in the paradigm itself and store inferred underlying forms at the end of analysis. This final store we might term the 'lexicon', though in our model it is localised to a particular paradigm and merely summarises the results of the acquisition process.

2.3 Encoding the logic of the analyst

2.3.1 Surface alternation as entry point: for child, for phonologist

As outlined in the foregoing discussion—in particular of Tesar's model (2006)—we employ paradigmatic information as an input to the learner; from the paradigm, morphological alternations evidenced in surface forms can be extracted to infer phonological rules and hypothesise serial derivations. Recall that...

> ... [p]aradigmatic information is information requiring knowledge of morphological identity across words. It consists of the phonological consequences of knowing that a morpheme must have a single phonological underlying form, even if it surfaces differently in different words.
>
> (Tesar, 2006, Abstract)

In employing paradigms for inference, we are in accord with Tesar's observation that phonotactics are not appropriate for conducting batch phonological acquisition—and that it is the phonological consequences of morphological identity that yield ordered rules.

We are now ready to lay the foundations for a model that determines morphological identity from alternating forms, thereby bootstrapping a phonological grammar. Inference hinges on a single condition: *underlying forms must be unified according to gloss* (see §4.1.2). That is to say—if two underlying forms for *cat* are found in the course of pairwise analysis, the learner cannot choose different forms for each—given that their glosses are identical (*cat* = *cat*), URs must be too. This identity is guaranteed through *unification* which provides a powerful context in which to situate rule acquisition and also points to a specific declarative programming model for implementing this approach.

Chapter 3

Theoretical Preliminaries

3.1 What is a feature?

3.1.1 Binary Features

This system assumes binary features for defining segments, though we admit '∅' as a legal sign for the purpose of *un*specifying a feature. This is discussed in (§3.2.1).

3.1.2 Features without (phonetic) substance

An important remark at this point has to do with the ontological status of the features and segments that are employed in the demonstrations to follow. For want of a better word, we provide *toy* inventories—in so far as they are qualitatively limited for expository purposes[1].

Given our discussion of substance-free phonology in (1.2.1), however, it is clear that in a very fundamental sense these inventories are *not* toys at all. This relates to the essential arbitrariness of the feature-set with regard to the actual phonological

[1] It is worth remarking at this juncture that the arbitrariness of the selected features has no integral connection with the nature of analysis, merely the content of inferred rules and representations. As such we would gain little by providing a richer inventory and likely end up confusing the reader.

computation—and of special interest to us, phonological inference. Since…

> …the substance of phonological entities is *never* relevant to how they are treated by the computational system, except in *arbitrary, stipulative* ways (Hale & Reiss, 2000, p. 157) …

…a demonstration of the mechanics of alternation and derivation—while related to an inventory—are generalisable to the computational properties of the grammar itself. In other words, features are treated as first-class objects integral to phonological computation—i.e., phonological computation and inference operates over feature space, not unanalysed segments. Neither the process of transduction nor the observation of minimal differences relies on anything other than feature bundle comparison, unification and computation of identity—irrespective of the phonetic correlates of the features involved.

3.2 What is a representation?

A representation is a sequence of feature matrices bounded by # markers, symbols drawn from the meta-language that are mapped with an identity relation. Below we explain the structure of representations and the segments used to build them.

3.2.1 Implicit Precedence

We assume that linear representations are ordered sets of segments where segments are drawn from an inventory of featural units composed of sign (where $sign \in \{\varnothing, +, -\}$) and attribute ($attr \in Attributes$, defined below).

(3.1) $Attributes = \{anterior, strident, sonorant, voice, nasal, lateral, continuant,$
$consonantal, distributed, delayed, dorsal, coronal, labial,$

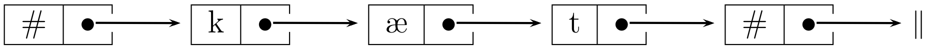
$high, low, back, round\}$

Segments are sets whose members are sign-attribute pairs; underspecification is possible through the omission of sign-attribute pairs from the set representing a segment. The pair $U = (\varnothing, Attr)$ is necessary for *un*specifying a feature. The attribute set is the only representation of feature attributes—we do not employ feature geometric structure anywhere in our model.

Precedence is an implicit property of representations; ordered sets are implemented computationally as *linked lists*, a near ubiquitous data structure in which each element (or CONS cell) contains a value slot (in which we place the feature matrix) and a pointer slot which indicates a successor cell[2]. The final element in a representation points to the null set (depicted here as '‖'); this is used in the majority of predicates as the *terminating condition* for recursive function application.

To illustrate, consider a simple representation like /#kæt#/ which would be represented as follows (note that feature matrices have been notated in IPA for expository purposes):

(3.2)

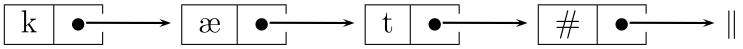


Following the traditional depiction of linked lists in computer science, we depict CONS cells as boxed records where the FIRST slot contains a segment and the REST slot points to the following cell. This clarification is worthwhile as, although we only encode precedence implicitly, the *data structure* upon which representations are built shares with Raimy an (underlying) explicit coding of precedence links (Raimy, 2000). Given that these pointers are set during sequential construction of the representation, though, we simply treat the chain of pointers as implicitly encoding precedence

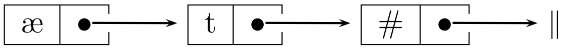
[2] In Lisp terminology, the value slot is known as CAR and the next pointer CDR. If we need to refer to these slots, we shall instead adopt the more modern (and language neutral) terms FIRST and REST.

and make use of standard list manipulation techniques for affixation and morphological decomposition. Stepping through a representation involves following the REST pointers which return the tail of the list:

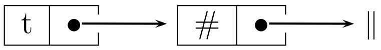
(3.3)



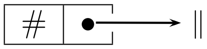
(3.4)



(3.5)



(3.6)



(3.7) ‖

As mentioned above, we can step recursively through representations and treat (3.7) as the terminating condition—when the head of the representation is '‖', there are no more segments to process.

3.3 What is a rule?

Phonological rules are string rewrite systems that map between strings of input symbols drawn from a well defined vocabulary. An ideal mechanism for performing this mapping is the *transducer* which consumes its input one symbol at a time and writes output to a parallel location. Due to the convenience with which we can implement input-output mapping between segments with finite-state approaches, we adopt transducers in our model with a crucial caveat: we are *not* making theoretical claims as to the finite-state nature of phonology as a whole. We clarify this point—using transducers for engineering a solution but not for making a formal claim about the generative capacity of the phonological module—below.

3.3.1 Finite State Transduction

Formal phonology—both as a computational and theoretical discipline—has attached itself to the idea of employing finite-state automata wherever possible. In this respect phonology is differentiated from other grammatical modules (syntax, semantics) as the latter cannot employ finite state technology effectively. This result has been known to hold for syntax since (Chomsky, 1957).

Research into the finite-stateness of phonology has its foundation in work by C. D. Johnson regarding the possibility of representing phonological rewrite rules as *regular relations* that map between input and output alphabets (Johnson, 1972). Defined as 'a finite-state language in which the expressions are composed of symbol pairs rather than single symbols, a mapping of one regular set to another one' (Karttunen, 1993), regular relations embody a radical simplification of the computational machinery (finite state transducers) and weak grammatical complexity (regular as opposed to context-sensitive grammar) at work in phonological derivations.

For this result to hold, various restrictions need to be imposed upon the phonological component—notably, the rejection of cyclicity in favour of iterative, directional rule application—but the basic approach appears to be an attractive one for reasons of elegance and simplicity.

More importantly, finite state transduction offers a very real possibility of boosting the efficiency of computation when implementing formal models. This realisation emerged a decade later when Kaplan and Kay (1994) discovered Johnson's (1972) work and employed the isomorphism between regular relations and finite state transducers to suggest that the phonology *as a whole* could be represented as a sequence of transducers—which could then compiled into a single finite-state transducer encompassing the entire ordered rule system (Kaplan & Kay, 1994).

While this adoption may seem desirable at first blush, it ignores at least two important points. The first is that it is unclear whether we can simply characterise phonology as finite state to the exclusion of the rest of the language faculty. It has long been known that natural language syntax exceeds the generative capacity of context-free (and hence regular, finite-state) grammars. To claim a theoretical advance by demonstrating that the phonology can be modelled using finite-state transduction then becomes clouded by the more general admission that the language faculty is not, in itself, finite-state. This argument can be bypassed by an appeal to modularity and a research programme that seeks to constrain every module as far as possible. The motivation for this is, however, unclear. From a theoretical and cognitivist standpoint, is is uncertain what it means to establish the weak generative capacity of a given module. Chomsky made this point over forty years ago, stating that…

> …one can construct hierarchies of grammatical theories in terms of weak and strong generative capacity, but it is important to bear in mind that these hierarchies do *not* necessarily correspond to what is probably the empirically most significant dimension of increasing power of linguistic theory. (Chomsky, 1965, p. 62)

Chomsky's discussion suggests that weak generative capacity should play practically no role in determining the power of a linguistic theory.

> It might conceivably turn out that [a] theory is extremely powerful (perhaps even universal, that is, equivalent in generative capacity to the theory of Turing machines) along the dimension of weak generative capacity, and even along the dimension of strong generative capacity. It will not necessarily follow that it is very powerful (and hence to be discounted) in the

> dimension which is ultimately of real empirical significance. (Chomsky, 1965, p. 62)

Berwick and Weinberg (1982) clarify this notion of *empirical significance* by breaking it down into two related problems: *Relevant Range* and *Implementation*.

The Relevant Range problem has 'quasi-biological import'[3] (which is to say it is biolinguistically motivated) and hinges on the mathematical relationship between the time complexity of different formalisms and the size of the data submitted for parsing. Berwick and Weinberg point out that in a syntactic setting, the length of the input sentence actually impacts the efficiency of the parsing algorithm; if a sentence exceeds a certain length, arguments from the standpoint of algorithmic efficiency and grammatical economy may come into play. If, however, the input length is shorter than this critical length, the economic choice is far from clear. As such,

> . . . an argument based on algorithmic superiority is only valid if we add the assumptions that: (1) sentences of the break-point length or greater actually occur in practice; (2) it actually matters that one procedure can parse a single sentence 11 words long in half the time of another—presumably for reasons of expressive power; and (3) the language faculty has been "shaped" by natural selection primarily on the basis of the selectional advantage conferred by more efficient sentence processing (leaving aside the question of whether or not this is indeed the primary "role" of the language faculty). However, it is actually difficult to see under what conditions this alleged parsing advantage could arise in practice. Not only are we forced to envisage a case where the speedier parsing of a long sentence matters, and matters in some selectional sense, but also this difficult-to-parse

[3] (Berwick & Weinberg, 1982, p. 180)

> sentence can have no two-sentence expressive substitute (for otherwise, it would come under the functional umbrella of the "slower" exponential procedure as well)
>
> Thus, the distinction between, say, cubic time and exponential time [parsing] procedures is possibly of no import in biological practice, depending upon the range of sentence lengths that actually mattered in the evolutionary "design" of language...
>
> (Berwick & Weinberg, 1982, p. 180–1)

This discussion gives a clear indication of why we should be wary of constraining our formalism through an appeal to formal language theory alone, defining economy in the absence of actual biolinguistically motivated theory. Furthermore, it may turn out that more complex formalisms are in fact more easily learnable—contrary to received wisdom of the discipline:

> Thus, although a set of strings may be perfectly well describable (in the weak generative sense) by a system of low expressive power, it may actually be advantageous in terms of parsing efficiency to capture the structure of that set by a more powerful formalism. The reason is simply that if in one formalism parsing time is some linear function of the length of the input and the size of the grammar (i.e. is proportional to $k \times |G|n$), and if one can move to, say, a context-free formalism and reduce the size of the grammar exponentially, then the price of using the n^3 context-free parsing algorithm could be well worth it: a reduction in the size of the grammar could more than make up for the increase due to the exponent change from n to n^3. Note that this advantage of succinctness is quite different from the usual linguistic claim that a more compact grammar

> is more easily learnable; we are claiming that it is possible that a more compact grammar, expressed by a more powerful formal system, is more efficiently processed as well.
>
> (Berwick & Weinberg, 1982, p. 184)

With this said about the *relevant range* problem, Berwick and Weinberg discuss the problem of implementation and associating computational organisation with cognitive structure:

> Since different representational formats can make for quite significant differences in parsing efficiency in the case of context-free parsing, it seems reasonable to conclude that the proper practical evaluation of an algorithm is a sophisticated task. It requires careful attention to alternative data structures and the underlying organization of the computer that has been assumed. In the cognitive domain the task is even more difficult, since the attendant computational assumptions are more likely to be lacking independent support. For example, if the primitive parallel operations demanded by the most efficient of the Graham, Harrison, and Ruzzo techniques have no analogue in cognitive machinery, then we cannot exploit the efficiency gains of this method. In short, we again discover that we must have a theory of implementation and some specific knowledge of the computational capabilities of the brain.
>
> (Berwick & Weinberg, 1982, p. 186)

This argument coincides with our discussion of Marr presented in (1.5.1); the empirical significance of a particular grammatical formalism cannot be determined on the basis of mathematical string sets; rather, it has to relate to cognitive structure, a theory of acquisition and a theory of the representational space over which computation takes place.

While the above exposition of empirical significance is compelling from a biolinguistic perspective, the opposing position—extolled by many in the field—still holds strong currency today. John Coleman, for example, claims that...

> ...the continued preponderance of all manner of transformational rules in generative phonology suggests that phonologists, on the whole, are unaware of the formal problems that led syntacticians to give up transformational rules, or that they regard phonology to be governed by different considerations from syntax (see Bromberger & Halle, 1989)
>
> (Coleman, 2005, p. 84)

While it may be true that there has been a lack of formal explicitness in phonological theory, it would seem that exclusively focusing on weak generative capacity is misguided and, as pointed out by Chomsky, not necessarily informative or useful in any empirical sense.

A final—and revealing—remark comes from Johnson himself who recognised the potential for translating phonological rules into finite state machines. He notes that

> [o]ne could, for example, propose that phonological rules be finite transducers in the literal sense. No one would take such a suggestion seriously because of the linguistic inappropriateness of the formulations it would require. (Johnson, 1972, p. 58)

Here we agree with Johnson; for our purposes, finite-state models are convenient from an implementation standpoint but strictly do *not* reflect our attitude towards the ontological status of phonological rules themselves.

3.3.2 Why are FSTs desirable at all, then?

Given the simplicity of their specification, finite state transducers make for a conveniently compact representation of rules in computational memory. To focus on their simplicity seems misguided as the infrastructure required for their induction (in our system) simply negates the theoretical attractiveness of simple devices from theoretical standpoint.

3.3.3 Features good, orthography bad

The theoretical plausibility of finite state transduction is often obscured by demonstrations of these devices for the purposes of orthographic morphological analysis (see, for example, Gildea & Jurafsky, 1995).

While we can point to the formal equivalence between transducers that employ orthographic alphabets (or phonemic ones) and those that are built with featural vocabularies, the former class of machines are incapable of expressing a fundamental principle of phonological rules: the notion of natural classes. This is addressed in detail in (§4.5.1).

At the simplest level, we can think of natural classes in terms of feature intersection capturing generalisations about the distribution of patterns in the phonology of a given language. The mechanisms responsible for inferring and applying rules need to be able to refer to commonalities in rule environments and foci.

3.3.4 Single feature changes

A final theoretical consideration is the nature of the structural changes licensed in individual phonological rewrite rules. Here we follow Chomsky (1967) in constraining the size of the change matrix to a single feature:

> Principle 6: Two successive lines of a derivation can differ by at most one feature specification.
>
> (Chomsky, 1967, p. 125)

This means that alternations exhibiting multiple feature changes in surface alternants of a single morpheme—for example *pag*∼*pax*—need to be broken down into their individual parts. The resulting individual rules can then either be adjacent in the derivation or separated by an arbitrary number of intermediate rules.

Consider Chomsky's original discussion of converting /k/ to [c]:

> (31) A segment which becomes a palatal becomes strident.
>
> [. . .]
>
> If we omit the reference to stridency, the rule converting /k/ to [c] is:
>
> (32) $\text{k} \rightarrow \begin{bmatrix} -\text{grave} \\ +\text{diffuse} \end{bmatrix}$
>
> (Chomsky, 1967, p. 125)

Given Principle 6, this rule needs to be interpreted as 'comprising two steps: the first of these converts /k/ to the corresponding [−grave] segment[4] [k,]; and the second step introduces the feature [+diffuse][5]' (Ibid, our footnotes). Why would we want to do this? Chomsky argues that by breaking down this composite rule (which we shall term a *process*) into two discrete steps, a separate rule can be interleaved that converts palatals into stridents. As such, the result of the application of [−grave] will yield [k̲] which can then be converted to [č]. This has proven very useful in finding compatible orderings during rule inference (see §4.3.1).

[4] [−back]

[5] [+high]

Chapter 4

Implementation

4.1 Why Prolog?

Our learner has been built from the ground up in the Prolog, a programming language that…

> …can be considered primarily as a theorem-proving system [Rob65] whose common application consists of asking queries in the context of a universe defined in clausal form. (Boizumault, 1993, p. 30)

In the case of phonological rule acquisition, the theorems to be proved are the rules inferred in the course of analysis. Prolog is particularly well suited to this task due to its in built support for *unification*—in particular, the learner can *backtrack* when it makes an erroneous inference or is asked to re-analyse a data set.

The primary top-level input to the inference engine is what we shall term a phonological *paradigm*; this is in fact composed of *pairs* of surface forms that are analysed in the course of inference. In this sense, the term serves something of a double meaning referring both to the paradigms being compared and the global data set over which this computation takes place. We adopt the latter.

In order to find underlying forms and phonological rules, a *paradigm query* is decomposed and rules are sought that map between the pairs in the input. In order to explain this process clearly it is first necessary to understand the basic mechanics of a Prolog programme—its facts, rules and the method by which we query the database.

4.1.1 Facts and Rules

At the heart of Prolog lie three main concepts: facts, rules and queries.

> The simplest kind of statement is called a fact. Facts are a means of stating that a relation holds between objects. (Sterling & Shapiro, 1994, p. 11)

To be more specific: facts define the *primitives* of our model—for our purposes, these primitives should correspond with properties of Universal Grammar. These primitives provide the foundation upon which phonological inference is computed by the learner. Given our primitives, we can define *rules* which...

> ...enabl[e] us to define new relationships in terms of existing relationships. (Sterling & Shapiro, 1994, p. 18)

4.1.2 Unification

Unification is fundamental to the method of binding variables in Prolog—and hence its ability to perform complex inferences. Unification is directly related to identity, subsumption and substitution and can be adequately introduced with a simple symbolic example—

> Informally, unification is the process of making terms identical by means of certain substitutions. For example, the terms $f(a, y, z)$ and $f(x, b, z)$,

> with a, b constants and x, y, z variables, can be made identical by applying to them the substitution $\{x/a, y/b\}$: both sides then become $f(a, b, z)$. But the substitution $\{x/a, y/b, z/a\}$ also makes these two terms identical. Such substitutions are called unifiers. (Apt, 1997, p. 24)

We can formalise this notion by following Shieber and examining unification as it applies to feature structures,

> In formal terms, we define the *unification* of two feature structures D' and D'' as the most general feature structure D, such that $D' \sqsubseteq D$ and $D'' \sqsubseteq D$. We note this $D = D' \sqcup D''$. (Shieber, 1986)

where '$\sqsubseteq$' is the subsumption operator. One structure A subsumes another B if and only if it has less information than B (or: if and only if A is more general than B).

While Shieber is referring to constraint-based unification models of grammar such as LFG and HPSG the principle applies equally well to the unification of feature matrices in Derivational Phonology—this can be employed for predicating on the presence or absence of a specific feature as well as finding a specification in order to flip its *sign* specification—the key criterion being its *substitutability*.

To illustrate the utility of this method, consider Boizumault's schematisation of unification of terms in Prolog:

> Given: two terms t_1 and t_2.
>
> Result: a pair $(bool, \sigma)$ such that:
>
> - $bool = true$ if and only if the two terms are unifiable.
> - if $bool = true$ then σ is the most general unifier for t_1 and t_2.

(Boizumault, 1993, p. 16)

A commonly used introductory example found in the majority of Prolog textbooks involves *parenthood* and exhibits the above principles very clearly. First, let us define a few facts about the world—as mentioned above, these can be considered *primitives* of the logical world over which inference is to take place:

Elementary Facts

(4.1)

```
male(john).
male(matthew).

female(sally).
female(zoe).

parent(john, zoe).
parent(sally, zoe).

parent(john, matthew).
parent(sally, matthew).
```

Unification becomes relevant when we pair these facts with the following rules:

Elementary Rules

(4.2)

```
father(X, Y) :-
  parent(X, Y),
  male(X).

mother(X, Y) :-
  parent(X, Y),
  female(X).

```

```
child(X, Y) :-
  parent(Y, X).
```

The rules in (4.2) can be read as follows:

- X is a father of Y if X is a parent of Y and X is male. Put differently: a father is a *male parent* of a child.
- X is a mother of Y if X is a parent of Y and X is female. Or: a mother is a *female parent* of a child.
- X is a child of Y if Y is a parent of X. This rule defines the inversion of the parent relationship—a rule built on the given facts.

These rules allow us to *query* the world:

Querying

(4.3)

```
?- father(X, Y).
X = john,
Y = zoe ;
X = john,
Y = matthew ;
false.

?- father(X, zoe).
X = john ;
false.
```

Each set of bindings preceding a semi-colon in the above example is indicative of a possible unification with the facts given in (4.1)—when interacting with Prolog we hit the $\boxed{;}$ key to see whether anything else can satisfy our query. In the first query, `father(X, Y)`, we ask Prolog to unify the terms X and Y with *facts of the world* such that the query is true. Here we get two positive results before `false` is returned—meaning that there are *only two* possible bindings. In the second query, `father(X, zoe)`, the same process applies but this time, since we have handed the system Zoe’s name as a literal (rather than a variable to be bound like Y), Prolog has to satisfy X such that X is the father of Zoe. As such, the only binding for X is `john`. Since Zoe only has one father, any attempt to resatisfy X by attempting a different unification with the facts results in `false`.

4.1.3 Summary: Backtracking

Of central import for any learner is the property of *backtracking*: namely, the ability to return to the last known *true* state and attempting to re-satisfy a query by following a different path of logical inference. This generally involves rebinding variables and employing a different rule (or: set of rules) to arrive at a satisfactory answer.

In that Prolog provides an inbuilt backtracking mechanism, it is a salient choice of platform on which to build a system to infer underlying forms and phonological rules. Where there are multiple possible analyses, the system can backtrack—hypothesising different URs in the process—and hence arrive at alternative analyses for the relevant rules operating in a paradigm.

By way of example, and in anticipation of explanations to come, we shall take a look at an elementary paradigm which can be satisfied by more than one analysis:

(4.4)

Simple Paradigm Demo

```
surface_morpheme(demoLanguage, [g,a,b], ’dog’).
surface_morpheme(demoLanguage, [g,a,p], ’dog’).
surface_morpheme(demoLanguage, [g,a,b], ’cat’).
surface_morpheme(demoLanguage, [a], ’PL’).
```

```

simpleDemo(Rules) :-
 paradigm(
 demoLanguage,
 [
  '#+gab+a#', dog,
  '#+gap+#' , dog,
  '#+gab+a#', cat,
  '#+gab+#' , cat
 ], _, _, _, Rules).

runSimpleDemo :-
 findall(Rules,
  (simpleDemo(Rules), format("Possibility:~n "),
            print_fst_rules(Rules)), _).
```

The first four lines in this example are used to define surface alternants and their associated glosses in `demoLanguage`. Next, we define a one-place predicate that unifies for `Rules` by analysing a `demoLanguage` paradigm. The reason that we refer to `demoLanguage` is to enable morpheme lookup during analysis. The paradigm that we're analysing contains four surface forms which will be analysed a pair at a time: *gaba* ∼ *gap* and *gaba* ∼ *gab*. The only information that we wish to draw from this analysis, in this example, is the ordered ruleset which will be unified with `Rules`, the sixth argument to `paradigm(...)`. In this way, `Rules` will be bound and unified with the `simpleDemo(...)` argument.

Calling the above query will yield the following set of possible analyses:

Possible Analyses

(4.5)

```
# swipl -f learner.pl -t runSimpleDemo

```

```
Possibility:
  /p/ [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / __a
Possibility:
  /p/ [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / __a#
Possibility:
  /p/ [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / #ga__a
Possibility:
  /p/ [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / #ga__a#
Possibility:
  /p/ [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / a__a
Possibility:
  /p/ [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / a__a#
Possibility:
  /p/ [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / ga__a
Possibility:
  /p/ [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / ga__a#
```

The above output demonstrates the analyses compatible with this paradigm. The focus segment, */p/*, is a second-class object—the rule focus is actually the feature matrix $\{-cor, -dors, -son, -cont, +lab, -del, -vc, +cons\}$. For our purposes, though, it may be more convenient to consider an abbreviated form of this output, provided in (4.6),

(4.6)

- /p/ → [+voice] / __a
- /p/ → [+voice] / __a#
- /p/ → [+voice] / #ga__a
- /p/ → [+voice] / #ga__a#
- /p/ → [+voice] / a__a

- /p/ → [+voice] / a__a#
- /p/ → [+voice] / ga__a
- /p/ → [+voice] / ga__a#

Only $p \rightarrow b$ is hypothesised for the alternating ‘dog’ form as a $b \rightarrow p$ rule would behave incorrectly by overgenerating for ‘cat’. Consider any devoicing rule that has /*b*/ as its focus. Since ‘cat’ does not alternate, only one underlying form will be hypothesised. Now we need to ensure that devoicing applies only to the ‘dog’ form but, problematically, ‘dog’ and ‘cat’ would have the same UR (since, by supposition, the rule we are looking for devoices). If both ‘dog’ and ‘cat’ are underlyingly identical and devoicing has applied in the surface form *gap*, then it should necessarily have applied to the suffixless ‘cat’ form—but does not. In identical environments we have *gab* and *gap*; clearly no devoicing rule could have applied here.

4.2 Representation

4.2.1 Features

A feature is simply defined as a Prolog *fact*. The attribute name is stored as an *atom* and is, unsurprisingly, arbitrary. A simple inventory, used for our examples, follows:

(4.7) Feature Inventory

```
feature(anterior).
feature(strident).
feature(sonorant).
feature(voice).
feature(nasal).
feature(lateral).
feature(continuant).
feature(consonantal).
feature(distributed).
feature(delayed).
feature(dorsal).
feature(coronal).
feature(labial).
feature(high).
feature(low).
feature(back).
feature(round).
```

Definition of a feature's *sign* follows the same basic principle making for the following concise definitions:

(4.8)

Sign Definition

```
sign(+).
sign(-).
sign(0).
```

Finally, as discussed in section (3.2.1), a feature specification is a tuple—namely, a pair—built from a *sign* and *attribute*. As such, we define a phonological *unit* (i.e., the basic unit of specification) as follows:

(4.9)

Feature Inventory

```
unit(X,Y) :- sign(X), feature(Y).
```

This states that a pair (X, Y) can be used to build a phonological unit so long as X is a *sign* (i.e., $X \in \{\varnothing, +, -\}$) and Y is a feature attribute as defined in (4.7). As mentioned in (3.2.1), $X = \varnothing$ is used for despecifying units in a feature matrix—not as a means of encoding underspecification.

4.2.2 Segments

Segments are (linked) lists of feature specifications. Underspecification can be handled through the omission of a specification (rather than a ternary specification slot).

Below is an example of a segment definition:

(4.10)

Segment /a/

```
segment(a,    [ unit(-,high)         ,   unit(-,back)   ,
                unit(+,low)          ,   unit(-,round)  ,
                unit(+,sonorant)     ,   unit(+,voice)  ,
                unit(-,consonantal)                       ] ).
```

Here we bind ‘a’ to a feature matrix. This is used for mapping between paradigm inputs and the featural representations that underlie them.

It is taken as axiomatic that segments may be *underspecified* for certain features. This seems to be a sensible allowance given the utility of underspecified segments in accounting for vowel harmony cross-linguistically (Inkelas, 1994).

4.2.3 Representations

Representations are lists of feature bundles. In section (3.2.1) we discussed the notion of implicit precedence in phonological representations. There we employed segment names in the FIRST box of each CONS cell. For conveniently mapping between IPA input and lists of matrices, we can leverage facts of the form `segment(X,Y)` by *mapping* to a new list, unifying X with a segment in the input and writing out Y in its stead:

(4.11)

Mapping from segment names to matrices

```
as_matrices([],[]).
as_matrices([X|Xs], [Y|Ys]) :- segment(X,Y), as_matrices(Xs,Ys).
```

The first line is the terminating condition—the empty list maps to itself—while the second line maps an IPA segment name (X) to its definition (Y) then recurses by calling `as_matrices/2` on the remainder of the representation.

4.2.4 Rules

Sterling and Shapiro (1994) provide a useful starting point for approaching finite state computation in Prolog with a simple implementation of a finite state machine (FSM), provided below.

```
accept(Xs) ←
    The string represented by the list Xs is accepted by the NDFA
    defined by initial/1, delta/3, and final/1.

 accept(Xs)          ←   initial(Q), accept (Xs, Q).

 accept([X|Xs], Q)   ←   delta(Q, X, Q1), accept(Xs, Q1).

 accept([], Q)       ←   final(Q).
```

(Sterling & Shapiro, 1994, p. 320)

The left arrow (←) is equivalent to a Prolog operator that we have already encountered, '`:-`'. We can read the three rules as follows:

1. `accept(Xs)` is true if there is some initial state Q and `accept(Xs, Q)` is true.

    - This rule takes a list of symbols (Xs), tries to locate a fact defining an initial state Q and queries the two-place version of (`accept/2`). *recognised.*

2. `accept([X|Xs], Q)` is true if there is a transition (defined by `delta/3`) from the current state, Q to some new state, Q_1, that recognises the first symbol in the input list, X.

- If this is true, the clause recurses by calling `accept(Xs, Q1)`—i.e., does this FSM accept the *rest* of the tokens starting from state Q_1?

3. `accept([], Q)` is true if and only if Q is the final state.

    - The input symbol list has been exhausted; this clause states that this is only acceptable if the FSM has been left in its final state.

These three rules provide all the necessary machinery for building a recogniser. All that remains is to define the transitions (using the `delta` predicate) and initial and start states. For example, the following machine will accept $(ab)*$:

(4.12)

FSM for (ab)*

```
initial(q0).
final(q0).
delta(q0,a,q1).
delta(ql,b,q0).
```

(Defined in Sterling & Shapiro, 1994, p. 321)

We can easily extend this scheme to provide a transduction mechanism by altering the definitions of `accept` to expect an additional *output tape*, onto which rewritten symbols are mapped, and amending our transition deltas to define which mapping to affect at each state.

(4.13)

From FSM to FST

```
accept(Xs,Ys)              :- initial(Q), accept(Xs, Ys, Q).
accept([X|Xs],[Y|Ys],Q)    :- delta(Q,X,Y,Q1), accept(Xs,Ys,Q1).
accept([], [], Q)          :- final(Q).

initial(q0).
```

```
final(q0).
delta(q0,a,'A',q1).
delta(q1,b,'B',q0).
```

This transducer accepts the same inputs as the FSM given in (4.12) but additionally maps $a \rightarrow A$ and $b \rightarrow B$ in the course of recognition:

Running the transducer

(4.14)

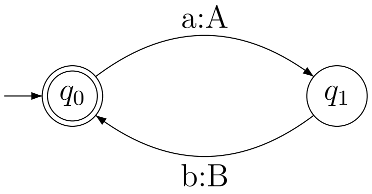
```
?- accept([a,b,a,b], Output).
Output = ['A', 'B', 'A', 'B'].

?- accept([a,b,a], Output).
false.

?- accept([a,b,a,b,a,b], Output).
Output = ['A', 'B', 'A', 'B', 'A', 'B'].
```

We can represent this transducer as the graph in (4.15),

(4.15)



4.3 Deriving one segment from another

We shall begin this discussion by considering a simple raising problem, $a \rightarrow e$ (where a is an italicised form of 'a', an open front unrounded vowel). Let us assume the following featural mapping:

$$(4.16)\quad \begin{bmatrix} -\text{high} \\ -\text{back} \\ +\text{low} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} \rightarrow \begin{bmatrix} -\text{high} \\ -\text{back} \\ -\text{low} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix}$$

Here, the feature matrices differ with respect to their specification of $[\alpha\text{low}]$. We can frame the problem as follows: given a, which featural specifications will yield e? Our first task is isolating the features that differ. This can be done by subtracting the first matrix from the second to find the featural units not present in the first. We call this process `derive`.

$$(4.17)\quad \begin{bmatrix} -\text{high} \\ -\text{back} \\ -\text{low} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} - \begin{bmatrix} -\text{high} \\ -\text{back} \\ +\text{low} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} = [-\text{low}]$$

A first draft of `derive` is presented below,

(4.18)

derive/3: first draft

```
derive(S1,S2,Specifications) :-
  segment(S1, Bundle1),
  segment(S2, Bundle2),
  subtract(Bundle2, Bundle1, Specifications).
```

As the parameter list indicates, `derive` expects three arguments: the first two are symbolic names for segments—in this case, *a* and *e*. The third argument is bound by subtracting the intersection of these bundles from the matrix for *e*. The example in (4.17) is thus queried as follows,

(4.19)

Querying derive/3

```
?- derive(a, e, Specifications).
Specifications = [unit(-, low)]
```

Now that we have the specifications present in the second bundle but absent from the first (in this case, just the one), we apply these specifications to *a* by calling the `specifyMatrix` predicate, notated below by the ⋓ operator.

(4.20)
$$[-\text{low}] \Cup \begin{bmatrix} -\text{high} \\ -\text{back} \\ \boxed{+}\ \ \text{low} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} = \begin{bmatrix} -\text{high} \\ -\text{back} \\ \boxed{-}\ \ \text{low} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix}$$

It is important to point out that we are *not* employing standard unification here—by definition, two segments that differ by one or more feature cannot be unified. Rather, `specifyMatrix` calls `specify` for each of the units in the set handed to it.

(4.21) A sketch of `specify`

1. If a matrix already contains the attribute being specified, alter its *sign* specification such that the featural unit can be unified. This process can be seen in (4.20).

2. If a matrix is missing the specification of the featural unit being specified (i.e., it is underspecified with respect to that feature), the specification is unified (which amounts to union) with the matrix.

The `specify` predicate is identical with what Kaplan (1995) calls ‘priority union’ and similar to the ‘default unification’ of Lascarides et. al (1999). We shall explain with reference Kaplan’s presentation:

> For two f-structures [for us, feature matrices] A and B, ‘A/B’ is their priority union, perhaps read as A given B, or A in the context of B. It is the set of pairs $\langle s, v\rangle$ such that v is equal to the value of the attribute s in the f-structure A, if s is in the domain of A, otherwise the value of s in the f-structure B. the operator *gives priority to the values in A but anything that A doesn’t include gets filled in for B.*
>
> (Kaplan, 1995, p. 365, our italics)

To illustrate this, Kaplan offers the example reproduced here as (4.22),

$$(4.22)\quad A = \begin{bmatrix} q & r \\ s & t \\ u & v \end{bmatrix} \qquad B = \begin{bmatrix} q & m \\ s & t \\ p & l \end{bmatrix} \qquad C = \begin{bmatrix} q & r \\ s & t \\ u & v \\ p & l \end{bmatrix}$$

The second column of each matrix is analogous to our phonological co-efficients, the first column to feature attributes. Concluding his explanation, Kaplan clarifies that

> A/B gets $\langle q, r\rangle$ from A, ignoring what would be the inconsistent [coefficient] value of q in B, not even noticing it. $\langle s, t\rangle$ is common to both so A/B includes that. It also has $\langle u, v\rangle$ from A and $\langle p, l\rangle$ from B.

> The basic idea is that values found in A override the values found in B, and B supplies the defaults.
>
> (Kaplan, 1995, p. 365)

We shall not adopt the oblique (‘/’) notation for `specify`, opting instead for ‘$\uplus$’ to represent priority union.

With the theoretical and formal meaning of the `specify` predicate established, let us return to the details of its operation.

To show the second interpretation of `specify`—union in the case of absent specification—let us assume that we wish to map $a' \rightarrow e$ where a' is underspecified with respect to [αlow]. A matrix for a' is provided in (4.23),

$$(4.23)\quad \begin{bmatrix} -\text{high} \\ -\text{back} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix}$$

As in (4.17), we subtract the bundle for a' from e to find the features that need to be specified for a'.

$$(4.24)\quad \begin{bmatrix} -\text{high} \\ -\text{back} \\ -\text{low} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} - \begin{bmatrix} -\text{high} \\ -\text{back} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} = [-\text{low}]$$

Note that the absence of [αlow] from a' doesn't impact the resultant matrix—from the perspective of finding features in need of specification, underspecification and sign difference are equivalent. Either a feature needs to be specified due to it being absent or it needs to be re-specified due to a sign change.

Calling $[-\text{low}] \Cup a$ is now equivalent to $[-\text{low}] \cup a$:

$$(4.25)\quad [-\text{low}] \Cup \begin{bmatrix} -\text{high} \\ -\text{back} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} = \begin{bmatrix} -\text{high} \\ -\text{back} \\ \boxed{-\text{low}} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} \equiv$$

$$[-\text{low}] \cup \begin{bmatrix} -\text{high} \\ -\text{back} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} = \begin{bmatrix} -\text{high} \\ -\text{back} \\ \boxed{-\text{low}} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix}$$

The approach presented so far poses a problem, however, when attempting to map $e \rightarrow a'$:

$$(4.26)\quad \begin{bmatrix} -\text{high} \\ -\text{back} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} - \begin{bmatrix} -\text{high} \\ -\text{back} \\ -\text{low} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} = \varnothing$$

The problem is the following: while $a' \cap e$ is equivalent to $e \cap a'$, subtracting the larger matrix from a' results in the empty list—read as 'nothing to specify'. This would suggest that $a' = e$ (i.e., nothing needs specifying to map $e \to a'$) which is clearly not the case. Such a result would be encountered regularly in the course of analysing a paradigm; by erroneously telling the learner that nothing needs specifying, segments will be conflated and inference will run awry.

This issue results from `derive` having no knowledge of underspecification in its analysis. To fix it, we need to step through an additional stage in which we check for features that are specified in the source (focus) bundle but are absent from the target.

(4.27) derive/3: final draft

```
derive(S1,S2,Specifications) :-
  segment(S1, Bundle1),
  segment(S2, Bundle2),
  missing_specifications(Bundle2, Bundle1, Absent),
  subtract(Bundle2, Bundle1, Specs),
  append(Specs, Absent, Specifications).
```

Now `derive` calls an additional predicate, `missing_specifications`, to see which features are specified in `Bundle1` but are absent from `Bundle2`, building an intermediate set. We union this set with the result of the subtraction to yield a final specifications list. To express `missing_specifications` in set theoretic terms requires two mapping operations—one from a domain of phonological units (defined as $\langle sign, attribute\rangle$ pairs) to a set of attributes and the second from a set of attributes back into a codomain of $\langle sign, attribute\rangle$ where $sign = \varnothing$:

(4.28)
$$\begin{array}{lll} f: & \langle S, A\rangle & \rightarrow & A \\ g: & A & \rightarrow & \langle \varnothing, A\rangle \end{array}$$

Specifications missing from `Bundle2` can now be found by employing the intersection and subtraction method in *attribute* space. Units in the `Absent` set will be given a *sign* value of $\varnothing$, indicating that they should be unspecified in order to arrive at the correct target segment.

(4.29) Step 1:

$$\begin{bmatrix} -\text{high} \\ -\text{back} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} - \begin{bmatrix} -\text{high} \\ -\text{back} \\ -\text{low} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} = \varnothing$$

Step 2:

$$\texttt{missing\_specifications}\left(\begin{bmatrix}-\text{high}\\-\text{back}\\-\text{round}\\+\text{sonorant}\\+\text{voice}\\-\text{consonantal}\end{bmatrix},\begin{bmatrix}-\text{high}\\-\text{back}\\-\text{low}\\-\text{round}\\+\text{sonorant}\\+\text{voice}\\-\text{consonantal}\end{bmatrix}\right)=$$

$[\varnothing\text{low}]$

Result:

$\varnothing \cup [\varnothing\text{low}] = [\varnothing\text{low}]$

We now need to refine our definition of `specify` to correctly interpret a ∅ sign:

(4.30) Revised sketch of `specify`

1. If a matrix already contains the attribute being specified, either:
    - Remove the target unit if the *sign* of the item being specified is ∅ (demonstrated below in 4.31);
    - Alter its *sign* specification such that the featural unit can be unified. This process can be seen in (4.20).
2. If a matrix is missing the specification of the featural unit being specified (i.e., it is underspecified with respect to that feature), the specification is unioned with the matrix.

Mapping from $e \rightarrow a'$ therefore yields the following, (4.31) $$[\varnothing\text{low}] \Cup \begin{bmatrix} -\text{high} \\ -\text{back} \\ \boxed{-\text{low}} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix} = \begin{bmatrix} -\text{high} \\ -\text{back} \\ -\text{round} \\ +\text{sonorant} \\ +\text{voice} \\ -\text{consonantal} \end{bmatrix}$$

It is important that we can assure segmental identity in this way for the purposes of unifying hypotheses for different forms in a phonological paradigm. Specifying a feature with a null sign is equivalent to despecification. In this case, '∅' is a symbol drawn from the metalanguage that removes attributes from the target bundle. Such an operation exceeds the power of feature-filling unification models and can be justified by the necessity of feature-changing phonological rules (Reiss, 2003).

Finally, here are the definitions for `specify` (we have left out insertion and deletion for the time being),

(4.32)

specify/3: first draft

```
specify([], X, X).

specify(unit(+,F), [unit(-,F)|Units], Segment) :-
  Segment = [unit(+,F)|Units].

specify(unit(-,F), [unit(+,F)|Units], Segment) :-
  Segment = [unit(-,F)|Units].

specify(unit(0,F), [unit(_,F)|Units], Units) :- !.

specify(unit(S,F), [unit(S,F)|Units], [unit(S,F)|Units]).
```

```
12
13 specify(U, [X|Units], [X|Segment]) :-
14   specify(U, Units, Segment).
15
16 specify(Unit, [], [Unit]).
```

The first line indicates that attempting to specify nothing (the empty set, ∅) results in identity of focus and target. The second and third clauses deal with sign flipping when they reach a unit with the same attribute. The fourth clause on the ninth line indicates that if we find the specified attribute we should remove it from the set. The fifth clause indicates that if both sign and attribute match a specification on the focus, this specification should carry over to the target. Finally, specifying a unit on an empty set (a matrix with no specifications) yields a matrix with that unit.

4.3.1 changeTo: mapping derive

The preceding discussion dealt with single segment changes, irrespective of how many features are involved in the act of specification. The next level of abstraction requires an expansion of this functionality across this *input string* as whole, building an *ordered set* of derivation paths, one for each segment. Where a segment exhibits identity, this path will be of length zero—the null set, ∅. This task is achieved by the `changeTo` and `change` predicates.

(4.33) The changeTo/4 Predicate

```
changeTo([],[],[],[]).
changeTo([S1|Rep1],[S2|Rep2],[P|Xs],[U|Ys]) :-
  derive(S1,S2,X),
  U = S1,
  permutation(X, P),
  changeTo(Rep1,Rep2,Xs,Ys).
```

The first definition of `changeTo` is its the terminating condition—it is true that the null set can be changed into the null set with a null set of changes where the underlying form of the input is also null. This is vacuously true. The second definition is the core rule: we leverage `derive` to recursively infer a *derivation set* that can map between strings. First, we find the *derivation path* from S_1 to S_2 and temporarily bind this as X. S_1 is stored as U—the underlying segment—and a permutation of X is stored as the path for this segment. Finally, `changeTo` recurses with the remaining segments in the input and output strings.

The role of permutation is best evidenced by an example of querying `changeTo`. Consider the query in (4.34):

(4.34)

Exploring changeTo/4

```
?- changeTo([a,g],[a,x],Changes, Underlying).
Changes = [[], [unit(+, continuant), unit(+, delayed), unit(-, voice)]],
Underlying = [a, g] ;

Changes = [[], [unit(+, delayed), unit(+, continuant), unit(-, voice)]],
Underlying = [a, g] ;

Changes = [[], [unit(+, delayed), unit(-, voice), unit(+, continuant)]],
Underlying = [a, g] ;

Changes = [[], [unit(+, continuant), unit(-, voice), unit(+, delayed)]],
Underlying = [a, g] ;

Changes = [[], [unit(-, voice), unit(+, continuant), unit(+, delayed)]],
Underlying = [a, g] ;

Changes = [[], [unit(-, voice), unit(+, delayed), unit(+, continuant)]],
Underlying = [a, g] ;
false.
```

If we consider the second term in `Changes` to be an *unordered* set, all results can be considered equivalent; permutations would be irrelevant. Importantly, though, these derivation paths are being permuted as *ordered* sets. The first binding states that mapping $a \rightarrow a$ can be achieved with an empty derivation path (no changes need to be made to arrive at the target) and that $g \rightarrow x$ can be achieved by the sequences given in (4.35, 4.36, etc.).

(4.35) $g \rightarrow [+continuant] \rightarrow [+delayed] \rightarrow [-voice] = x$

(4.36) $g \rightarrow [+delayed] \rightarrow [+continuant] \rightarrow [-voice] = x$

By hypothesising all possible orders in which these features can be specified we can make the ordering task much easier. The adjacency of [+delayed] and [+continuant] in (4.36) needn't manifest in an ordered derivation—by permuting at this stage, we are simply providing higher levels of inference with information for performing the ordering operation.

4.3.2 `change`: abstracting `changeTo`

The `change` predicate essentially leverages `changeTo` after preprocessing for correspondence and alignment. Specially demarcated substrings of the inputs are fed through `changeTo` and then recombined with the appropriate prefix and suffix information as a post-processing step.

Discussing the definition of this predicate is unnecessary for comprehending the functionality of the learner as whole; what *is* necessary, however, is a brief discussion of alignment and correspondence concerns that are crucial for inference to function correctly.

4.4 Alignment

4.4.1 Correspondence

Knowledge of correspondence is fundamental for building an accurate learning system. First, consider the following chart for the surface alternation $\#tag\# \sim \#tak\#$:

$$(4.37)\quad \begin{array}{ccccc} \# & \mathrm{t} & \mathrm{a} & \mathrm{g} & \# \\ \Updownarrow & \Updownarrow & \Updownarrow & \Updownarrow & \Updownarrow \\ \# & \mathrm{t} & \mathrm{a} & \mathrm{k} & \# \end{array}$$

Here there is a one-to-one correspondence between segments and we note that $g \sim k$. This case is obviously contrived: there is no reason to focus on the voicing difference between /g/ and /k/ as they fall in the same environment and are therefore not actually alternating. Let us therefore extend (4.37) to a more plausible comparison, the surface alternation $\#taga\# \sim \#tak\#$ where $-a$ is a suffix. A first approximation is given in (4.38):

$$(4.38)\quad \begin{array}{cccccc} \# & \mathrm{t} & \mathrm{a} & \mathrm{g} & \mathrm{a} & \# \\ \Updownarrow & \Updownarrow & \Updownarrow & \Updownarrow & \Updownarrow & \\ \# & \mathrm{t} & \mathrm{a} & \mathrm{k} & \# & \end{array}$$

What this (erroneous) chart shows is that in the absence of additional information—or an algorithm—the suffix of the first form will correspond with the word boundary symbol of the second. This in turn yields the absurd statement $a \sim \#$. The simplest solution to the problem is to frame the substring of inputs that should be compared. In other words, as in (4.38), we are interested in the alternation in the stem; the suffix is irrelevant to *locating* the alternation, though it will of course be indispensable for building possible rule environments at a later stage of analysis.

In order to frame material for correspondence comparison, we employ '+' symbols in the input. These essentially demarcate the window in which correspondence should occur.

(4.39)
$$\begin{array}{cccccccc} \# & + & \mathrm{t} & \mathrm{a} & \mathrm{g} & + & \mathrm{a} & \# \\ & & \Updownarrow & \Updownarrow & \Updownarrow & & & \\ \# & + & \mathrm{t} & \mathrm{a} & \mathrm{k} & + & \# & \end{array}$$

From this correspondence we localise alternations and infer that $g \sim k$.

4.4.2 Realignment

In cases of deletion and insertion, the above correspondence is still problematic. Consider $\#pat\# \sim \#paa\#$ where $-a$ is a suffix:

(4.40)
$$\begin{array}{ccccccc} \# & + & \mathrm{p} & \mathrm{a} & \mathrm{t} & + & \# \\ & & \Updownarrow & \Updownarrow & \Updownarrow & & \\ \# & + & \mathrm{p} & \mathrm{a} & + & \mathrm{a} & \# \end{array}$$

Even with our + markers, alignment is incorrect; now we have $t \sim +$ which is both incorrect and exhibits a mixing of symbol types—segment with boundary marker. The solution to this problem involves the inference of a null segment, ϵ, to correspond with the t:

(4.41)
$$\begin{array}{cccccccc} \# & + & \mathrm{p} & \mathrm{a} & \mathrm{t} & + & \# & \\ & & \Updownarrow & \Updownarrow & \Updownarrow & & & \\ \# & + & \mathrm{p} & \mathrm{a} & \epsilon & + & \mathrm{a} & \# \end{array}$$

This is the task of `realign/3`, shown in (4.42), a predicate that inserts ϵ in one of the two (framed) forms when input lengths differ.

Segment Realignment Predicates

(4.42)

```
realign([], [], []).
realign([], [_|_], [0]).
realign([X|Stem1], [Y|Stem2], [0|Align1]) :-
  X \= Y, realign([X|Stem1], Stem2, Align1).
realign([X|Stem1], [X|Stem2], [X|Align1]) :-
  realign(Stem1, Stem2, Align1).

adjust_alignment(Stem1, Stem2, Stem1, Align2) :-
  length(Stem1, L1), length(Stem2, L2),
  L1 > L2, !, realign(Stem2, Stem1, Align2).

adjust_alignment(Stem1, Stem2, Align1, Stem2) :-
  length(Stem1, L1), length(Stem2, L2),
  L1 < L2, !, realign(Stem1, Stem2, Align1).

adjust_alignment(Stem1, Stem2, Stem1, Stem2).
```

4.5 Rules

Rules are inferred in two steps—the first involves building lists for foci, change and environment; the second involves a translation of these lists into transducers for further evaluation.

4.5.1 FSTs

Finite-state transducers are built dynamically from rule structures for two explicit purposes:

1. They make rule composition convenient;

2. They provide a means for efficiently and effectively testing inferred rules across a paradigm as the last step in satisfying a query.

(4.43) A Finite-State Transducer is a 6-tuple $(\Sigma_1, \Sigma_2, Q, i, F, E)$ such that:

- Σ_1 is a finite alphabet, namely the input alphabet
- Σ_2 is a finite alphabet, namely the output alphabet
- Q is a finite set of states
- $i \in Q$ is the initial state
- $F \subseteq Q$ is the set of final states
- $E \subseteq Q \times \Sigma_1^* \times \Sigma_2^* \times Q$ is the set of edges

(Drawn from Roche & Shabes, 1997, p. 14)

Defining inferable transducers requires a few minor modifications to the code outlined in (4.13). The core predicates are listed in (4.44),

(4.44)

Transducers

```
transduce(FST,Xs,Ys) :-
  initial(FST,Q), transduce(FST, Xs, Ys, Q).

transduce(FST,[X|Xs],[Y|Ys],Q) :-
  delta(FST,Q,X,Y,Q1), transduce(FST,Xs,Ys,Q1).

```

```
transduce(FST, [], [], Q) :-
  final(FST,Q), !.
```

The `transduce/3` predicate provides a toplevel input; it looks up the initial state, Q for the transducer in question—information provided by unifying with FST—and calls `transduce/4` to map a string of Xs to Ys. This latter predicate, `transduce/4`, tries to find a transition—via `delta/5`—that, given the current state Q, defines a mapping from X to Y. If such an arc exists, the machine will be placed into a new state, Q_1, and the operation recurses.

Transduction terminates when three conditions hold:

1. The input tape is empty,

2. The output tape is empty,

3. The machine is in a state Q that is a member of the final states.

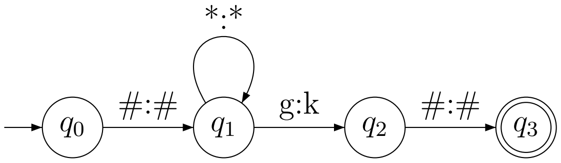
The predicates in (4.44) only provide the architecture for transduction; in order to see them operating we need to define a new machine on their basis. Given that we wish to have transducers inferred, we avoid hard-coding facts about them *in general* (unlike the example in (4.13)), instead marking the component parts of an FST *dynamic*.

(4.45) Making *dynamic* transducers

```
:- dynamic initial/2.
:- dynamic final/2.
:- dynamic delta/5.
```

The goal of this approach is to produce rules akin to the depiction in (4.46), a transducer graph representing the phonological rule $g \rightarrow k\ /\_\_\#$.

(4.46)



This transducer begins by matching the word-boundary symbol on the input tape and mapping it to itself, consuming the symbol in the process. We shall call the identity mapping *self-mapping*. If the first symbol on the input tape is *not* the word-boundary symbol, recognition fails and the transducer rejects its input. If on the other hand the mapping succeeds, the machine changes state: $q_0 \rightarrow q_1$. In q_1 the '*' symbol indicates a wildcard match—any segment will map to itself (identity holds). When this transition is followed, no state change occurs. If the entirety of the input tape is exhausted while still on q_1, mapping will fail as q_1 is not a member of the set of final states. If the machine encounters a 'g' on the input tape, it has the option of mapping it to 'k' and in so doing changes state again, this time $q_1 \rightarrow q_2$. State q_2 requires that the very next symbol be a word boundary symbol which is again self-mapped to reach the final state, q_3. In the event that the symbol following 'g' is *not* the word-boundary symbol, recognition will fail and the $g \rightarrow k$ mapping will effectively not have applied.

To allow for failed mappings to truly correlate with inapplicable rules, we require a fall through condition that allows inputs to be self-mapped across the board. If, however, a mapping *is* possible, we want to ensure that it has to apply and that we cannot employ the fall through. As such, upon reaching the final state of the mapping transducer we block the possibility of trying the identity rule with the cut operator, '!'. This states that once the transduction has succeeded, backtracking is no longer possible. The `transducer/4` predicate is therefore abstracted as `run_transducer/3`, listed in (4.47):

(4.47) Running a transducer indirectly

```
run_transducer(FST, Input, Output) :-
  initial(FST, Q),
  transduce(FST, Input, Output, Q), !.

run_transducer(_, Input, Input).
```

An objection might be raised that states q_0 and q_1 don't have correlates in the phonological rule which they are supposed to be encoding. This is only true in a very superficial sense: notice that the first transition requires that the input begin with a boundary symbol—this is simply a way of requiring well-formed input strings. The loop transition on q_1, by contrast, is actually implicit in the $g \rightarrow k$ rule itself. Its absence (with or without the inclusion of q_0) would require the first symbol on the input tape to be g—which is not what we want at all. Rather, the rule as written translates to the following: *ignore everything before a g in the input and when you locate one, rewrite it as k when preceding #.* Ignoring material before g is the job of the wildcard mapping on q_1.

A more pressing concern with the above discussion should be over the choice of symbols (or indeed, their type) on the input and output tapes—namely the use of orthography in representing a velar obstruent (see §3.3.3). Nothing especially theoretical is captured by this transducer: no reference is made to the features comprising g, nor to the fact that this mapping represents the process of word final *devoicing*. Let us therefore begin by proposing a simple but feature-theoretic version of the original rule,

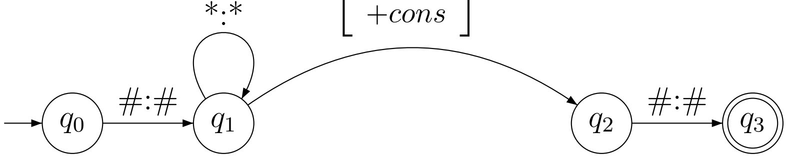
(4.48)
$$\begin{bmatrix} -\mathit{coronal} \\ +\mathit{dorsal} \\ -\mathit{sonorant} \\ -\mathit{continuant} \\ -\mathit{labial} \\ -\mathit{delayed} \\ +\mathit{voice} \\ +\mathit{consonantal} \end{bmatrix} \rightarrow \begin{bmatrix} -\mathit{voice} \end{bmatrix} \; / \; \rule{2em}{0.4pt}\#$$

Our choice of features for encoding g is unimportant here—relevant, though, is that our transduction model should be capable of representing rules that transcend the orthographic. Unsurprisingly, it can. To do so, the transition rule for $g : k$ must do two things:

1. Match the symbol at the head of the input tape to a feature bundle (using the subset relation);
2. Employ the $\Cup$ operator to force a specification of [+voice].

$$X{:}Y \leftarrow (\begin{bmatrix} -\mathit{cor} \\ +\mathit{dor} \\ -\mathit{son} \\ -\mathit{cont} \\ -\mathit{lab} \\ -\mathit{del} \\ +\mathit{vc} \\ +\mathit{cons} \end{bmatrix} \subseteq X) \wedge (Y \sqcup (X \Cup [+\mathit{voice}]))$$

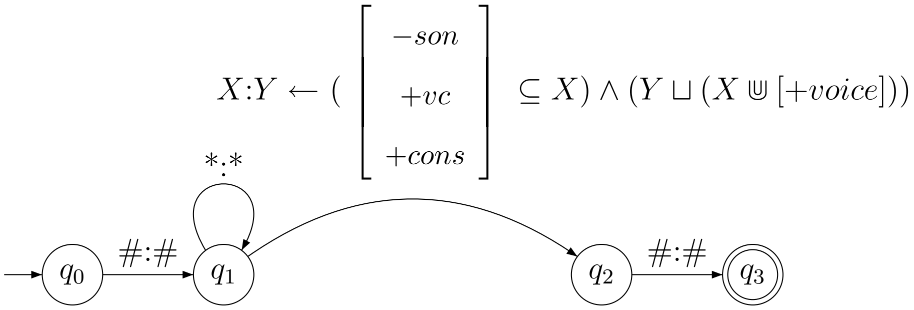
(4.49)



In this rather clumsy depiction, we see that X maps to Y so long as the subset relation holds for X; if it does, Y is the result of calling `specifyMatrix/3` on X.

A nice result of this approach is that the transducer can be generalised in a way not possible for (4.46): by taking a subset of the matrix for g, we can adapt the FST such that it is triggered for all voiced obstruents:

(4.50)


$X{:}Y \leftarrow (\begin{bmatrix} -son \\ +vc \\ +cons \end{bmatrix} \subseteq X) \wedge (Y \sqcup (X \Cup [+voice]))$
$*{:}*$
$\#{:}\#$
q_0
q_1
q_2
$\#{:}\#$
q_3

This approach therefore provides ample flexibility for capturing natural classes in rule foci.

4.6 Glossing

Glossing tells the learner which alternating surface forms should be derived from the same underlying form—through the unification of a form's gloss, the engine can hypothesise an appropriate UR.

The most direct way of explaining the necessity of glossing comes from neutralisation cases such as the one supplied in (4.51) below.

(4.51)

Surface Form	Gloss
gap	a dog
gaba	the dog
gap	a cat
gapa	the cat

Given that both ‘dog’ and ‘cat’ can surface as *gap* we need to ensure that in accounting for the alternation *gap* ∼ *gaba*, ‘dog’ is inferred to be underlyingly *gab* (with word-final devoicing) and differentiated from the form for ‘cat’. In the absence of glossing we could still infer that the only plausible rule is $b \rightarrow p$ but there would be no way of associating this with a particular form; furthermore, building more complex derivations would be impossible.

4.6.1 Surface Morphemes—`surface_morpheme/3`

The engine provides a simple predicate, `surface_morpheme`, for defining facts about surface forms and their glosses. The predicate is *dynamic*, meaning that facts can be defined outside of the inference engine’s core codebase. Its definition is trivial, provided in (4.52):

(4.52) Dynamic gloss predicate

```
:- dynamic(surface_morpheme/3).
```

This dynamic fact is employed indirectly by another predicate, `underlying_identity`, for ensuring that underlying forms are unified for a given gloss.

4.7 The Paradigm

Having described the building blocks of our system we can now turn to its toplevel input, the phonological paradigm. For us, a paradigm consists of a list of surface forms paired with their glosses. These glosses correspond with a substring of the representation bounded by + markers.

Consider the following toy language in (4.53):

(4.53) Language A

```
surface_morpheme(langA, [g,a,b], 'dog').
surface_morpheme(langA, [g,a,p], 'dog').
surface_morpheme(langA, [a], 'the').
surface_morpheme(langA, [i], 'PL').

langA :- paradigm(
          langA,
          [
            '#+gap+#' , dog,
            '#+gab+a#', dog,
            '#+gap+#' , dog,
            '#+gab+i#', dog
          ], Underlying, Surface, Lexicon, Rules),
          lang_info(Underlying, Surface, Lexicon, Rules).
```

Our definition begins with a preamble listing glosses for surface morphemes[1] independent of the context in which they are found. It is assumed by this layer of the system, then, that morphological information can be extracted from phonologically unanalysed surface forms prior to rule inference. This assumption might well prove to be untenable but it is not difficult to conceive of a 'preprocessor' step in which the information explicitly provided by `surface_morpheme` facts could be inferred automatically.

For now, we can read the `surface_morpheme/3` definitions as follows:

- In *langA*, the noun *dog* can surface as /gab/;
- In *langA*, the noun *dog* can also surface as /gap/;

[1] Ideally, this step would be automated—for clarity, we have left such morphological analysis to the linguist/programmer employing this system.

- In *langA*, the determiner *the* surfaces as /a/;
- In *langA*, the plural marker surfaces as /i/.

Next, a zero place predicate, `langA`, is defined (while terms unify within the rule body, `langA` has no arguments); this is a rule that evaluates to true so long as four variables can be bound by `paradigm`: `Underlying`, `Surface`, `Lexicon` and `Rules`. For convenience we conjoin a final term to print these variables in a well-formatted (and hence human-readable) fashion.

A query over this paradigm, therefore, should yield an analysis that provides a single underlying form for 'dog' given that it surfaces with two different phonological forms. Furthermore, since this is the only morpheme that exhibits surface alternation, nothing needs to be said for the determiner or plural marker; as such, we don't provide them as inputs for analysis by marking them in the input. (It is trivial to do so, but the resulting output contains redundant mapping information.)

In order to obtain an analysis, we simply feed `langA` in as a query:

Querying Language A

(4.54)

```
?- langA.
Rules
-----
 /p/  [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / __a
 /p/  [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / __i
Mappings
--------
  #gap#              => #gap#
  #gapa#             => #gaba#
  #gap#              => #gap#
  #gapi#             => #gabi#
Lexicon
```

```
-------
  gap      => 'dog'

[...]
```

The Prolog engine has returned the simplest possible analysis of the paradigm, suggesting that two rules are involved in accounting for the surface alternation $p \sim b$. Our 'lexicon' indicates that the underlying form for the only alternant provided in the paradigm, 'dog', is /gap/. Note that the absence of /i/ and /a/ is a consequence of those morphemes not being subjected to paradigmatic analysis—our lexicon merely collects the forms that we are interested in accounting for phonologically. Here, the alternation is explained through two rules, both with a focus of p and structural change of [+voice]: The two inferred rules are:

$$\begin{bmatrix} -coronal \\ -dorsal \\ -sonorant \\ -continuant \\ +labial \\ -delayed \\ -voice \\ +consonantal \end{bmatrix} \rightarrow \begin{bmatrix} +voice \end{bmatrix} \; / \; \underline{\hspace{2em}} \; \text{a} \tag{4.55}$$

(4.56)
$$\begin{bmatrix} -coronal \\ -dorsal \\ -sonorant \\ -continuant \\ +labial \\ -delayed \\ -voice \\ +consonantal \end{bmatrix} \rightarrow \begin{bmatrix} +voice \end{bmatrix} \;/\; \underline{\qquad}\,\mathrm{i}$$

These rules are *ordered*, although here rule interaction is irrelevant to the analysis.

The ellipsis on line 16 of (4.54) has been added to indicate that this solution is only one of many. Hitting $\boxed{;}$ immediately yields an analysis involving the opposite rule ordering, as in (4.57):

(4.57)

Querying Language A: Backtrack 1

```
Rules
-----
 /p/  [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / __i
 /p/  [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / __a
Mappings
--------
  #gap#               => #gap#
  #gapa#              => #gaba#
  #gap#               => #gap#
  #gapi#              => #gabi#
Lexicon
-------
  gap       => 'dog'
```

Ordering is irrelevant to this paradigm and the system correctly recognises this fact. Subsequent re-analyses involve adjustments to the rule environments posited for the rules given in (4.55–4.56) such as in (4.58),

(4.58)

Querying Language A: Backtrack 2

```
Rules
-----
 /p/  [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / __a
 /p/  [ -cor -dors -son -cont +lab -del -vc +cons ] --> [ +vc ] / __i#
Mappings
--------
  #gap#             => #gap#
  #gapa#            => #gaba#
  #gap#             => #gap#
  #gapi#            => #gabi#
Lexicon
-------
  gap      => 'dog'
```

before positing the opposite direction of mapping, $b \rightarrow p$ (4.59):

(4.59)

Querying Language A: Reverse Mapping

```
Rules
-----
 /b/  [ -cor -dors -son -cont +lab -del +vc +cons ] --> [ -vc ] / __#
Mappings
--------
  #gab#             => #gap#
  #gaba#            => #gaba#
  #gab#             => #gap#
  #gabi#            => #gabi#
```

```
Lexicon
-------
  gab      => ’dog’
```

Again, the simplest (smallest) environment is hypothesised first, followed by all possible environments.

4.7.1 Incremental Checks

A final stage of the learning process worth mentioning is the incremental checking that takes place in the course of inferring underlying forms and derivations. Ordered rules from the current hypothesis are regularly checked against the entire paradigm such that bad analyses are ruled out as early as possible and backtracking can be triggered. This occurs in the course of paradigm analysis, a process that is split into multiple levels and handed to sub-predicates (`paradigm4`, `paradigm3`, `paradigm2`, `paradigm1`).

Checking works very simply: for each entry in the paradigm, underlying forms and rules are inferred. Let us assume that there are three pairs (six forms) provided for comparison. Once the first pair has been compared, a set of rules (transducers) will have been inferred. These transducers are then tested against the next pair in the paradigm—if they make incorrect predictions they are illicit and backtracking to the first pair occurs. We now attempt to find a new rule that can account for these alternations and proceed once more to the second pair. Permutation (re-ordering) is one possible way of satisfying the conditions placed on the paradigm by transducer checks and in this way we get feeding and bleeding orders.

4.8 Further Work, Current Problems

The model presented in this chapter demonstrates the type of learning that can be achieved using surface alternation as the learner's only criterion for inferring rules and underlying forms. In discussing Tesar's work on learning from paradigmatic information (2006), we ruled out phonotactic information as a possible guide for our learner for reasons of parsimony. One possible amendment to the learning model we present above would be to compare its analyses with that of a phonotactically driven system and to use empirical tests to ascertain where phonotactic information produces more coherent analyses. Such comparison would be helpful in narrowing our focus to those problematic cases and addressing more directly why phonotactics has been chosen as integral to phonological models.

A further source of implementation and inquiry could investigate the phenomenon of 'free rides' much discussed in the (OT) literature. McCarthy (2004) presents the problem as follows,

> When alternation data tell the learner that some surface [B]s are derived from underlying /A/s, the learner will under certain conditions generalize by deriving all [B]s, even nonalternating ones, from /A/s. An adequate learning theory must therefore incorporate a procedure that allows nonalternating [B]s to take a "free ride" on the /A/ → [B] unfaithful map.
>
> (McCarthy, 2004, Abstract)

If free rides play a strong role in the acquisition process, it would seem wise to incorporate such tendencies of over-generalisation into a phonological learning model. Since we have attempted the simplest possible model (both in terms of first principles and implementational considerations), free rides play no explicit role in our model. It would be very interesting to explore first the degree of over-generalisation currently evidenced by our system and then experiment with different degrees of generalisation in hypothesising underlying forms. This would provide an elegant means for integrating recent work in Optimality learning with serialist frameworks such as the one presented in this paper.

Chapter 5

Examples

5.1 Deletion

Problems with deletion—in particular, correspondence—were covered in (§4.4.1). Here we provide a simple paradigm to demonstrate the learner's analysis. The input to the learner is the following paradigm:

(5.1)

	Surface Form	Gloss
a.	to	dog
b.	tolgi	the dog
c.	to	hat
d.	togi	the hat

From the above we can see that -*gi* is the suffix for the definite article and neutralisation occurs on the indeterminate forms for ‘dog’ and ‘hat’. The paradigm can be transcribed as follows:

Deletion Example

```
surface_morpheme(deletionLanguage, [t,o]  , ’dog’).
surface_morpheme(deletionLanguage, [t,o,l], ’dog’).
surface_morpheme(deletionLanguage, [t,o]  , 'hat').
surface_morpheme(deletionLanguage, [t,o]  , 'hat').
surface_morpheme(deletionLanguage, [g,i]  , 'the').

deletionLanguage :- paradigm(
          deletionLanguage,
          [
            '#+to+#' , dog,
            '#+tol+gi#', dog,
            '#+to+#' , hat,
            '#+to+gi#', hat
          ], Underlying, Surface, Lexicon, Rules),
          lang_info(Underlying, Surface, Lexicon, Rules).

goDeletionLanguage :- findall(_, deletionLanguage, _).
```

Submitting this paradigm to he leaner results in the following analysis:

Deletion Analysis

```

Rules
-----
 /0/  NULL --> [ +cor -dors -son -cont -lab +ant -nas +lat +vc +cons ] / __g
Mappings
---------
  #to#               => #to#
  #togi#             => #tolgi#
  #to#               => #to#
  #togi#             => #togi#
Lexicon
-------
  to       => 'hat'
  to       => 'dog'

Rules
-----
 /0/  NULL --> [ +cor -dors -son -cont -lab +ant -nas +lat +vc +cons ] / __gi
Mappings
--------
  #to#               => #to#
  #togi#             => #tolgi#
  #to#               => #to#
  #togi#             => #togi#
Lexicon
-------
  to       => 'hat'
  to       => 'dog'

Rules
-----
 /0/  NULL --> [ +cor -dors -son -cont -lab +ant -nas +lat +vc +cons ] / o__g
Mappings
--------
  #to#               => #to#
  #togi#             => #tolgi#
  #to#               => #to#
  #togi#             => #togi#
Lexicon
-------
  to       => 'hat'
  to       => 'dog'

Rules
```

```
-----
 /0/  NULL --> [ +cor -dors -son -cont -lab +ant -nas +lat +vc +cons ] / o__gi
Mappings
--------
  #to#               => #to#
  #togi#             => #tolgi#
  #to#               => #to#
  #togi#             => #togi#
Lexicon
-------
  to       => ’hat’
  to       => ’dog’

Rules
-----
 /0/  NULL --> [ +cor -dors -son -cont -lab +ant -nas +lat +vc +cons ] / to__g
Mappings
--------
  #to#               => #to#
  #togi#             => #tolgi#
  #to#               => #to#
  #togi#             => #togi#
Lexicon
-------
  to       => ’hat’
  to       => ’dog’

Rules
-----
 /0/  NULL --> [ +cor -dors -son -cont -lab +ant -nas +lat +vc +cons ] / to__gi
Mappings
```

```
--------
  #to#              => #to#
  #togi#            => #tolgi#
  #to#              => #to#
  #togi#            => #togi#
Lexicon
-------
  to       => 'hat'
  to       => 'dog'

[...]

Rules
-----
 /l/  [ +cor -dors -son -cont -lab +ant -nas +lat +vc +cons ] --> 0 / __#
Mappings
--------
  #tol#             => #to#
  #tolgi#           => #tolgi#
  #to#              => #to#
  #togi#            => #togi#
Lexicon
-------
  to       => 'hat'
  tol      => 'dog'

Rules
-----
 /l/  [ +cor -dors -son -cont -lab +ant -nas +lat +vc +cons ] --> 0 / o__#
Mappings
--------
```

```
  #tol#              => #to#
  #tolgi#            => #tolgi#
  #to#               => #to#
  #togi#             => #togi#
Lexicon
-------
  to        => 'hat'
  tol       => 'dog'

Rules
-----
 /l/  [ +cor -dors -son -cont -lab +ant -nas +lat +vc +cons ] --> 0 / to__#
Mappings
--------
  #tol#              => #to#
  #tolgi#            => #tolgi#
  #to#               => #to#
  #togi#             => #togi#
Lexicon
-------
  to        => 'hat'
  tol       => 'dog'

```

5.2 Ordering Example

This toy language—based loosely on Lamba—illustrates a crucial ordering when rule environments are local. (Only when the left and right environments are made very specific is a difference in ordering possible.) The input for our analysis is as follows:

(5.2)

	Surface Form	Gloss
a.	fiswa	the dog
b.	fiʃika	some dogs
c.	fiʃila	my dog
d.	fisana	your dog
e.	tulwa	the cat
f.	tulana	your cat
g.	tulika	some cats
h.	tulila	my cat
i.	tesela	my desk

We transcribe the above paradigm as follows, repeating some rows to focus on either stem or suffix alternation:

Ordering Example

```
surface_morpheme(orderingExample, [f,i,ʃ], ’dog’).
surface_morpheme(orderingExample, [f,i,s], ’dog’).
surface_morpheme(orderingExample, [t,u,l], ’cat’).
surface_morpheme(orderingExample, [t,e,s], ’desk’).
surface_morpheme(orderingExample, [i,k,a], ’PL’).
surface_morpheme(orderingExample, [i,l,a], ’my’).
surface_morpheme(orderingExample, [e,l,a], ’my’).
surface_morpheme(orderingExample, [a,n,a], ’your’).
surface_morpheme(orderingExample, [w,a], ’the’).

orderingExample :- paradigm(
          orderingExample,
          [
            ’#+fis+wa#’, dog,
            ’#+fiʃ+ika#’, dog,
            '#+fiʃ+ila#', dog,
            '#+fis+ana#' , dog,
            '#+tul+wa#', cat,
            '#+tul+ana#', cat,
            '#+tul+ika#', cat,
            '#+tul+ila#', cat,
            '#tul+ana+#', your,
            '#fis+ana+#', your,
            '#tes+ela+#', my,
            '#tul+ila+#', my,
            '#tul+ila+#', my,
            '#fiʃ+ila+#', my
          ], Underlying, Surface, Lexicon, Rules),
          lang_info(Underlying, Surface, Lexicon, Rules).

goOrderingExample :- findall(_, orderingExample, _).
```

Querying with the toplevel `goOrderingExample` yields the following (note that θ has been rendered /T/ due to limitations in unicode representation):

Ordering Analysis

```
Rules
-----
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / es__
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / __i
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / __i
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => 'my'
  ana      => 'your'
  tul      => 'cat'
  fis      => 'dog'

Rules
-----
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / es__l
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / __i
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / __i
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => ’my’
  ana      => ’your’
  tul      => ’cat’
  fis      => ’dog’

Rules
-----
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / es__la
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / __i
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / __i
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => ’my’
  ana      => ’your’
  tul      => ’cat’
  fis      => ’dog’

Rules
-----
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / es__
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => 'my'
  ana      => 'your'
  tul      => 'cat'
  fis      => 'dog'

Rules
-----
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / es__
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
```

```
  ila      => 'my'
  ana      => 'your'
  tul      => 'cat'
  fis      => 'dog'

Rules
-----
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / es__
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => 'my'
  ana      => 'your'
  tul      => 'cat'
  fis      => ’dog’

Rules
-----
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / es__l
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => ’my’
  ana      => ’your’
  tul      => ’cat’
  fis      => ’dog’

Rules
```

```
-----
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / es__l
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => 'my'
  ana      => 'your'
  tul      => 'cat'
  fis      => 'dog'

Rules
-----
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / es__l
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila       => 'my'
  ana       => 'your'
  tul       => 'cat'
  fis       => 'dog'

Rules
-----
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / es__la
Mappings
--------
```

```
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => 'my'
  ana      => 'your'
  tul      => 'cat'
  fis      => 'dog'

Rules
-----
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / es__la
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => 'my'
  ana      => 'your'
  tul      => 'cat'
  fis      => 'dog'

Rules
-----
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / es__la
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => ’my’
  ana      => ’your’
  tul      => ’cat’
  fis      => ’dog’

Rules
-----
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / s__
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => 'my'
  ana      => 'your'
  tul      => 'cat'
  fis      => 'dog'

Rules
-----
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / s__
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => ’my’
  ana      => ’your’
  tul      => ’cat’
  fis      => ’dog’

Rules
-----
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / s__l
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
```

```
-------
  ila      => ’my’
  ana      => ’your’
  tul      => ’cat’
  fis      => ’dog’

Rules
-----
 /s/  [ +cor -dors -son +cont -lab -dist +del +ant -vc +cons ] --> [ +dist ] / fi__i
 /i/  [ +hi -bk -lo -rd +son +vc -cons ] --> [ -hi ] / s__l
 /T/  [ +cor -dors -son +cont -lab +dist +del +ant -vc +cons ] --> [ -ant ] / fi__i
Mappings
--------
  #fiswa#            => #fiswa#
  #fisika#           => #fiʃika#
  #fisila#           => #fiʃila#
  #fisana#           => #fisana#
  #tulwa#            => #tulwa#
  #tulana#           => #tulana#
  #tulika#           => #tulika#
  #tulila#           => #tulila#
  #tulana#           => #tulana#
  #fisana#           => #fisana#
  #tesila#           => #tesela#
  #tulila#           => #tulila#
  #tulila#           => #tulila#
  #fisila#           => #fiʃila#
Lexicon
-------
  ila      => ’my’
  ana      => ’your’
  tul      => ’cat’
  fis      => ’dog’

[...]
```

Bibliography

Albright, A. & B. Hayes (1999). An automated learner for phonology and morphology. *Dept. of Linguistics, UCLA.* .

Apt, K. R. (1997). *From logic programming to Prolog.* London: Prentice Hall.

Berko, J. (1958). The child's learning of English morphology. *Word* **14**:150–177.

Berwick, Robert C. & Amy S. Weinberg (1982). Parsing efficiency, computational complexity, and the evaluation of grammatical theories. *Linguistic Inquiry* **13**:165–191.

Bobrow, D. G. & J. B. Fraser (1968). A phonological rule tester. *Commun. ACM* **11**:766–772.

Boizumault, P. (1993). *The implementation of Prolog.* New Jersey: Princeton University Press.

Bromberger, Sylvain & Morris Halle (1989). Why phonology is different. *Linguistic Inquiry* **20**:51–70.

Chomsky, N. (1967). Some general properties of phonological rules. *Language* **43**:102–128.

Chomsky, Noam (1957). *Syntactic structures*. The Hague: Mouton.

Chomsky, Noam (1965). *Aspects of the Theory of Syntax*. Cambridge, MA: MIT Press.

Chomsky, Noam (2000). *New Horizons in the Study of Language and Mind*. New York: Cambridge University Press.

Chomsky, Noam & Morris Halle (1968). *Sound Pattern of English*. New York: Harper and Row.

Coleman, J. (2005). *Phonological representations: their names, forms and powers*. Cambridge: Cambridge University Press.

Gibson, Edward & Kenneth Wexler (1994). Triggers. *Linguistic Inquiry* **25**:407–454.

Gildea, Daniel & Daniel Jurafsky (1995). Automatic induction of finite state transducers for simple phonological rules. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*. Morristown, NJ, USA, 9–15.

Hale, M. & C. Reiss (2000). "Substance abuse" and "dysfunctionalism": Current trends in phonology. *Linguistic Inquiry* **31**:157–169.

Hale, M. & C. Reiss (2008). *The phonological enterprise*. New York, NY: Oxford University Press.

Idsardi, William James (1992). *The computation of prosody*. Ph.D. thesis, M.I.T., Cambridge, MA, USA.

Inkelas, Sharon (1994). The consequences of optimization for underspecification. *ROA #41, Rutgers Optimality Archive* .

Johnson, C. Douglas (1972). *Formal aspects of phonological description*, vol. no. 3. The Hague: Mouton.

Kaplan, R. M. (1995). Three seductions of computational psycholinguistics. In M Dalrymple, R.M. Kaplan, J.T. Maxwell III & A. Zaenen (eds.), *Formal Issues in Lexical-Functional Grammar*, vol. 47 of *CSLI Lecture Notes Series*. Stanford, CA: CSLI Publications.

Kaplan, R. M. & M. Kay (1994). Regular models of phonological rule systems. *Computational linguistics* **20**:378.

Karttunen, L. (1993). Finite-state constraints. In *The last phonological rule: Reflections on constraints and derivations*. Chicago: University of Chicago Press, 173–194.

Kenstowicz, Michael & Charles Kisseberth (1979). *Generative Phonology: description and theory*. New York, NY: Academic Press.

Lascarides, Alex, Ted Briscoe, Nicholas Asher & Ann Copestake (1999). Order independent and persistent typed default unification. *Linguistics and Philosophy* **19**:1–89.

Marr, D. (1982). *Vision: a computational investigation into the human representation and processing of visual information*. W. H. Freeman, San Francisco.

McCarthy, J. J. (2004). Taking a free ride in morphophonemic learning. Ms. Umass. *Catalan Journal of Linguistics* **4**.

Prince, A. & P. Smolensky (1993). Optimality theory: Constraint interaction in generative grammar. *ROA-537, Rutgers Optimality Archive* .

Raimy, Eric (2000). Remarks on backcopying. *Linguistic Inquiry* **31**:541–552.

Reiss, Charles (2003). Deriving the feature-filling/feature-changing contrast: An application to hungarian vowel harmony. *Linguistic Inquiry* **34**:199–224.

Roche, Emmanuel & Yves Shabes (eds.) (1997). *Finite-State Language Processing.* Cambridge, MA, USA: MIT Press.

Shieber, S. M. (1986). *An introduction to unification-based approaches to grammar.* CSLI Lecture Notes 4. Stanford, CA: CSLI.

Smolensky, Paul (1999). Grammar-based connectionist approaches to language. *Cognitive Science* **23**:589–613.

Sterling, L. & E. Shapiro (1994). *The Art of Prolog.* Cambridge, MA: MIT.

Tesar, B. (2006). Learning from paradigmatic information. In *Proceedings—NELS*, vol. 36. Citeseer, 619.

Tesar, B. & P. Smolensky (2000). *Learnability in optimality theory.* Cambridge, MA: MIT.

Vaux, Bert (2007). Why the phonological component must be serial and rule-based. *Ms., Cambridge University* .