Computational Phonology – Part I: Foundations

Jeffrey Heinz*
University of Delaware

Abstract

Computational phonology approaches the study of sound patterns in the world's languages from a computational perspective. This article explains this perspective and its relevance to phonology. A restrictive, universal property of phonological patterns – they are regular – is established, and the hypothesis that they are subregular is presented. This article is intended primarily for phonologists who are curious about computational phonology, but do not have a rigorous background in mathematics or computation. However, it is also informative for readers with a background in computation and the basics of phonology, and who are curious about what computational analysis offers phonological theory.

1. What is Computational Phonology?

Computational phonology is formal phonology, and formal phonology is theoretical phonology. Computational phonology is not concerned with the implementation of phonological theories on computers (though that may be a byproduct of computational analysis). The primary concern of computational phonology is the content of the theory itself.

This article and *Computational Phonology – Part II: Grammars, Learning and the Future* present three important contributions of computational phonology. First, computational analysis of phonological formalisms reveals that the similarities between generative theories of phonology like Optimality Theory (OT) (Prince and Smolensky 2004) and the one found in *The Sound Pattern of English* (SPE) (Chomsky and Halle 1968) outweigh their differences. This is in part because computational analysis identifies exactly how different generative theories define and combine the individual factors that make up a language's phonology. It is also in part because of the second contribution: computational analysis reveals a restrictive, universal property of phonological patterns: they are REGULAR. What are regular patterns? A definition is given in (1).

(1) A pattern is regular if and only if (iff) it is possible to partition the set of logically possible words into *finitely* many blocks such that
  a. all words in any block either obey the pattern or all do not, and
  b. for any block, if it contains words w_1 and w_2 then, for all words v, there is a block which contains both w_1v and w_2v.

For example, consider the pattern given by words composed only of CV syllables. This pattern is regular. To see why, partition all logically possible words into three blocks: Block 1 contains all words which either start with V or which contain consecutive CC or VV sequences; Block 2 contains other words which end in V; and Block 3 contains the remaining words. In other words, logically possible words like VCV, CVVC, CVCC belong to Block 1, words like CVCV belongs to Block 2 and words like CVC belong to block 3. Clearly all words in Block 2 belong to the language and all words in Blocks 1 and 3 do not, satisfying condition (1-a). It is not hard to see that (1-b) is satisfied as well. To illustrate, consider that both CV and CVCV belong to Block 2 and notice that concatenating CCV to both of those strings results in strings that are in Block 1, whereas concatenating C to both strings results in strings that are in Block 3, whereas concateting CVCV to both strings results in strings that are in Block 2. Since the partition above satisfies conditions (1-a) and (1-b) and has finitely many blocks (there are three), this pattern is regular.[1] Additional examples of regular and non-regular patterns are given later.

The third important contribution is that computational analysis reveals that phonological patterns are SUBREGULAR. That is, there are even more restrictive properties shared by *all* phonological patterns no matter how diverse; in particular, the kinds of distinctions that phonological patterns make (the kinds of blocks) are sharply limited. Regular and subregular patterns are discussed in more detail in Section 2. Taken together, these contributions show computational phonology is identifying stronger and stronger universal properties of phonological patterns and identifying the necessary and sufficient conditions for a logically possible pattern to be a phonological one.

This article establishes the foundations of the theory of computation and the first major result of computational phonology: all phonological patterns are regular. Foundational aspects of the theory of computation include problems, algorithms, decidability, tractability, and formal language theory. All of these topics are interrelated, and their relevance to phonology is made clear.

In a brief review, it is impossible to cover every deserving topic. Notably absent from this article and Part II are computational analyses of nonlinear phonology (Bird and Ellison 1994; Eisner 1997; Gibbon 1987; Kornai 1994) and applications of computational phonology to speech technologies (e.g. Carson-Berndsen 1998).

The remainder of the introduction establishes the general principles that guide the remainder of the article. The first two are the scientific principle of factorizing complex systems into their component parts and the importance of restrictiveness and adequate expressivity to linguistic theories. The subsequent principles introduce the computational perspective: mathematical characterizations of patterns and the Zen-like importance of understanding problems more than their solutions.

Science and phonology. Phonological systems are complex systems, presumably because there are many interacting factors. The goal of any science when confronted with complex phenomena is to identify these factors, the principles underlying them, and their interaction. Figure 1 illustrates where F_i indicate individual factors and P_L is the phonology of a particular language L. Generative theories of phonology and morphophonology describe the whole phonology of a language in exactly this way: there are individual phonological generalizations (F_i) which interact in particular ways. There are several questions that are asked about such theories. Are the factors language-specific, language-universal, or both? What constrains the factors such that P_L and $P_{L'}$ are not arbitrarily different (under a central hypothesis within generative phonology that phonologies of the world's languages do not differ arbitrarily). What is the nature of the interaction (×) of phonological generalizations? Computational phonology helps us understand how different theories of phonology answer these questions. I refer to the structure of theories as shown in Figure 1 as the ARCHITECTURE of the theory.

$$F_1 \times F_2 \times \ldots \times F_n = P_L$$

Fig 1. Theories of phonology.

Expressivity and restrictiveness. When comparing theories, the notions of restrictiveness and expressivity are paramount. Which theory is unnecessarily more powerful than the other, and which theory is inadequately expressive? The most restrictive theory which is minimally adequately expressive is tacitly assumed to be the most desirable. Theories that are inadequately expressive leave attested patterns unexplained. On the other hand, theories that are insufficiently restrictive leave the absence of unattestable patterns without explanation. A theory that anything is possible is not much of a theory. The theory of computation is a *universal* theory which concretely defines the dual notions of restrictiveness and expressivity.

Characterizing phonological patterns. It will be useful to be familiar with set-theoretic, relational, and functional characterizations of phonological generalizations. A set is a collection of elements. A relation is a set specifying which elements of one set are associated with which elements of another set. A function maps each element of one set (the domain) to at most one element of another set (the codomain).

Sets and relations can be described as functions. To illustrate, consider the phonological generalization that English post-consonantal, word-final coronal stops optionally delete (Coetzee 2004; Guy 1980; Guy and Boberg 1997). For example, *perfect* is sometimes pronounced [pɹ̩fɛk]. There are additional factors that condition the frequency of the rule's application, (such as the initial sound of the following word, if any) and a general discussion of frequency effects occurs in Part II. The rule in (2) captures the deletion process.

(2) $[+\text{coronal},-\text{continuant}] \rightarrow \emptyset / \text{C}\_\_\_\_\#$

Figure 2 shows a fragment of the relational and the functional characterization of (2).

Two things ought to be clear. The first is that it is impossible to write down the relational and functional characterizations of word-final deletion in their entirety because they are infinite in size. This follows not only from there not being a principled upper bound on the length of words, but also from the fact that even if there was, it would be a distinct generalization from the one about word-final deletion. In the same way that nothing in the rule in (2) limits its application to actual words in English, there is nothing in the rule that limits its application to words of certain lengths. The second is that (2) is a precise, finite characterization of both the relational and functional characterizations. Armed with this rule, we can decide which pairs of strings belong to the relational characterization, or equivalently, which pairs the functional characterization maps to 1.

Computational theories. This article emphasizes the importance of relational and functional characterizations of phonological generalizations because they are the focus of

	R		f	
wɛst	→	wɛs	(wɛst,wɛs)	→ 1
			(wɛst,wɛsk)	→ 0
			(wɛst,wɛ)	→ 0
	...		...	
pɪwɛst	→	pɪwɛs	(pɪwɛst,pɪwɛs)	→ 1
			(pɪwɛst,pɪwɛsk)	→ 0
			(pɪwɛst,pɪwɛ)	→ 0
	...		...	

Fig 2. Fragments of the relational and functional characterization of the rule in (2). The function *f* maps a pair (*x*,*y*) to 1 if and only if *x* → *y* belongs to *R*.

phonological inquiry, which aims to identify essential properties of phonological generalizations. Computational analysis is exactly the analysis which permits this. Barton et al. (1987, 96–7) explain:

> Scientific explanation of any complex biological information-processing system occurs at three levels: (1) a *computational theory*, which explains what is computed and why; (2) a *representation* for the input and output of the process and the *algorithm* for the transformation; and (3) the hardware *implementation*, or the device in which the representation and algorithm are physically realized… The competence theories of linguistics correspond to Marr's (1980) topmost level of computational theory–explaining what structures are computed and why, ignoring memory limitations, shifts of attention of interest, and errors.

Additionally, Marr (1980, 27) writes

> … an algorithm is likely to be understood more readily by understanding the nature of the problem being solved than by examining the mechanism (and hardware) in which it is embodied.

What does this mean? In the context of the optional word-final deletion generalization, it means understanding the relational and functional characterization is likely to be more important than the procedure implied by the rule in (2) for deciding whether a pair of strings exemplifies the generalization or not. Informally, the phonological generalization is the problem and the rule in (2) is one algorithm that solves it.

Organization. This article is organized as follows. Section 2 discuss foundations of theoretical computer science relevant to computational phonology. References are generally withheld and then given in a Further Reading section. Section 3 presents Kaplan and Kay's (1994) analysis which establishes that virtually all phonological patterns are regular. Section 4 concludes.

2. Foundations

This section reviews especially relevant foundational issues at the intersection of theoretical computer science, philosophy, and linguistics; in particular, the mathematical notions of PROBLEMS and ALGORITHMS, as well as their foundational properties such as DECIDABILITY, TRACTABILITY, and DETERMINISM. Readers familiar with these concepts may skip this section.

2.1. PROBLEMS AND ALGORITHMS

Problems. Informally, problems are questions with several parameters. When all parameters are set, an instance of the problem is obtained. For example, the phonotactic well-formedness problem is given below:

(3) For a given phonological grammar G and surface form w, is w well-formed?

In this example, there are two parameters: a grammar and a surface form. When these are set, we have a specific instance of the phonotactic well-formedness problem. An example is whether 'blick' is a well-formed word according to English phonology.

What is the instance space of the phonotactic well-formedness problem? Much depends on what is meant by the phrase *phonological grammar* in (3). For example, it is necessary to decide whether the phonological grammars are an ordered list of SPE-style rewrite rules

(Chomsky and Halle 1968) or an ordered list of OT constraints (Prince and Smolensky 2004) or something else.

Answers to problems are also from a well-defined set. For the phonotactic well- formedness problem, the answers can be categorical, i.e. either 'yes' or 'no' as above, or they could make more distinctions; e.g. by letting answers be real numbers.[2] There is no loss of generality by assuming the answers are limited to 'yes' or 'no' and so we make this assumption for the background exposition. Discussion of problems with real-numbered answers (e.g. problems relating to free variation) occurs in Part II.

Additional problems of interest to phonology include, but are not limited to, the ones in (4).

(4) Informal statements of phonolgical problems.
   a. For a given phonological grammar and underlying form, what is the surface form? (the generation problem)
   b. For a given phonological grammar and a surface form, what is the underlying form? (the recognition problem)
   c. Given the set of possible phonological grammars and an underlying form, what are the possible surface forms? (the typological problem)
   d. Given a finite set of surface generated by a particular phonological grammar, which phonological grammar generated them? (the phonotactic learning problem)
   e. Given a finite set of underlying forms paired with surface forms generated by particular phonological grammar, what phonological grammar generated them? (the alternation learning problem)
   f. Given a finite set of meanings paired with surface forms from a particular phonological grammar, what phonological grammar and lexicon generated them? (the phonological learning problem)

As already mentioned, much hinges on the the qualifier 'phonological' in the phrase 'phonological grammar'. This is because what counts as a phonological grammar defines the instance space of the problem. For example, virtually no one believes the correct answer to any of the learning problems is a grammar which just regurgitates the training data and fails to generalize to new forms. Such grammars exist in principle but are presumably not phonological.

Algorithms. Algorithms are step-by-step procedures which solve problems. An algorithm solves a problem if it can be applied to *any* instance of the problem and be *guaranteed* to output the correct answer.

Mathematically, the distinction between problems and algorithms is subtler. The functional characterization of a problem is given by a mapping of its instances to its answers. For example, consider the problem 'For all strings x,y, does the pair (x,y) exemplify word-final coronal stop deletion?' The function in Figure 2 maps all instances of this problem to its answers, where 1 means 'yes' and 0 means 'no'. Algorithms are particular implementations of such functions, provided such implementations exist. Henceforth, I will use the word *problem* interchangeably with *function*, and the word *algorithm* interchangeably with *procedure*.

Decidability. Problems are classified according to the inherent difficulty. Most logically possible problems do not admit any algorithm. Such problems are called UNDECIDABLE. DECIDABLE problems can be solved by algorithms which always yield an output. Without loss of generality, consider a variant of the generation problem (4-a). Instead of asking what surface form s an underlying form u maps to, consider the problem which asks whether (*u*,*s*) is a valid mapping, whose answers are 'yes' and 'no'. If this problem is decidable, then for every instance of this problem there is an algorithm which always (correctly) answers 'yes' or 'no'. SEMI-DECIDABLE problems are those for which there exists an algorithm which always outputs 'yes' on inputs whose answer is 'yes' (but it may never produce an output on inputs whose answer is 'no'). Semi-decidable problems are also called COMPUTABLE. Additional classification is discussed in Section 2.2.

Tractability. Decidable problems are classified into two types: TRACTABLE and INTRACTABLE. Problems are tractable iff there is a algorithm which implements this function in fewer than $f(n)$ steps where f is a polynomial function and n is the length of the input (this input is the problem instance in some string-based representation). Decidable problems which cannot be so bounded are intractable.

Determinism. Additionally, DETERMINISTIC algorithms are ones for which at every state there is a single, uniquely defined next action. NON-DETERMINISTIC algorithms are not deterministic so there is at least one state where either no action is defined (the algorithm 'hangs'), or where more than one action is defined (and what happens next is determined randomly). Non-deterministic algorithms are often described as 'unrealistic' because they amount to procedures which guess and then verify.[3]

The distinction between deterministic and non-deterministic algorithms is made because of the famous hypothesis that the class of problems P solvable by tractable deterministic algorithms are distinct from the class of problems NP solvable by tractable non-deterministic algorithms. P is obviously included within NP, but it is unknown whether P equals NP, and most computer scientists believe it does not.

2.2. FORMAL LANGUAGE THEORY

Formal language theory forms another important chapter in theoretical computer science and linguistics and is closely related to the preceding discussion. This is because a language is a problem, in the mathematical sense described above. Assuming some representational scheme, each instance of a problem can be encoded as a string, and the question is whether that string maps to 'yes' or 'no'. Likewise, languages can be thought of as mapping strings to 'yes' if the string belongs to the language or to 'no' if it does not. This is called the MEMBERSHIP PROBLEM, and this function defines the language. It sounds strange to think of a language as a problem, but problems are just functions, and languages are functions too; after all, they are the product of a number of generalizations, which are also functions (Figure 1).

Languages. It is customary to fix a finite alphabet of symbols. Phonologists could take this to mean every possible IPA symbol (including diacritics), or as indivisible features along with symbols which allow groupings of those features to be interpreted. Prosodic markings (e.g. syllabic, super-, and sub-syllabic boundaries) can also be part of the alphabet. The alphabet can really be anything, so long as it is finite. There is no problem representing as much phonological structure as necessary (e.g. autosegmental tiers, foot boundaries) with strings, provided we know how to interpret the string.

Standardly, the symbol Σ denotes this alphabet and Σ^* the set of all logically possible finite strings writable with this alphabet, and a LANGUAGE is a subset of Σ^*. Phonologists are often interested in how underlying forms are related to surface forms. Without loss of generality, assume the same alphabet is used for underlying and surface forms. An ALTERNATION is then a subset of $\Sigma^* \times \Sigma^*$ (i.e. a subset of the set containing all logically possible pairs of strings).

Grammars. A grammar G is a finite description of a potentially infinite language L. Essentially, G is an algorithm which solves the ENUMERATION PROBLEM: What is the *n*th string of L? (A set is enumerable if and only if it is finite or can be put in one-to-one correspondence with the natural numbers.)

Languages for which the membership problem is semi-decidable are exactly those for which the enumeration problem is solvable. Thus, a theory of phonology which only asserts its patterns are semi-decidable is hardly a theory at all – because there are *no* restrictions on what constitutes a grammar other than the fact that it can be written down in a finite period of time.

The Chomsky Hierarchy. The Chomsky Hierarchy classifies languages according to their inherent expressivity. There are five major nested regions (5).

(5) finite $\subset$ regular $\subset$ context-free $\subset$ context-sensitive $\subset$ semi-decidable

The finite languages are also regular languages, which are also context-free, and so on (but not vice versa). Importantly, the choice of alphabet has zero impact on a language's place in the hierarchy. It does not matter whether phonological strings are coded with zeroes and ones or with thousands of symbols.

Every region goes by other names. For examples, languages which are semi-decidable are also called Type-0 and RECURSIVELY ENUMERABLE. These terms come from different formalisms, which were later realized to describe exactly the same set of languages. This is one reason why the Chomsky Hierarchy attracts so much attention. When independently-motivated formalisms converge to exactly the same regions, it suggests deeper principles are involved. This is exactly the case with regular patterns which are describable by regular expressions, formulae of Monadic-Second Order (MSO) logic, and finite-state machines (FSMs). Each of these formalisms essentially yield a finite partition of the space of all logically possible words in accordance with (1).

This classification is also independent of whether we consider languages as functions which map strings to 'yes' or 'no' or as functions mapping strings to real numbers. Languages which describe probability distributions over Σ^* are called STOCHASTIC LANGUAGES, and are typically described with probabilistic variants of the common grammatical formalisms used to define the regions in (5).

Since *any* pattern is a language in the sense above, a distinct advantage of the Chomsky Hierarchy is it allows for the comparison of patterns from different domains. Another advantage is that different grammatical formalisms within a domain (like phonology) can be compared in terms of their expressive power.

What kind of formal languages are phonological patterns? The consensus is that phonological patterns are regular. The arguments are discussed in Section 3. This hypothesis is made not only with respect to the functional characterizations of individual phonological generalizations, but also with respect to the product of these phonological generalizations as well. 'Being regular' is a property of phonological patterns at Marr's highest level, the computational level.

There are two important points to make in this regard. First, just because we can describe phonological patterns with FSMs, for example, does not mean FSMs are the correct description at the algorithmic level. The grammatical formalisms phonologists or psycholinguists employ may be exactly the right ones. I would argue, however, that it is vital for phonologists to relate their formalisms to the computational level, and to the widely adopted representations at this level, in order to be able to communicate the nature of phonological patterns effectively with people in other sciences, and to compare and contrast them with patterns in other domains.

Second, while 'being regular' may be a necessary property of phonological generalizations, it is almost certainly not a sufficient one. Many regular languages describe unnatural phonological patterns. For example, imagine the logically possible language in which words are well-formed only if they contain an even number of vowels, regardless of their order. Words like *bbb,baba, bbaa, aab, bbaabbb* are all well-formed according to this pattern, unlike words like *a, ababa, bab, bababab*. This pattern is a regular language,[4] though most phonologists would agree natural languages do not contain bonafide phonological generalizations of this sort (nor could natural languages do so).

What this means is that phonological patterns are almost certainly subregular; that is, they occupy some area strictly smaller than the regular languages.

(6) Hypothesis: Phonology $\subset$ Regular

There has been little research to date determining the exact nature of the subregular region phonology occupies. The discussion of some promising results are postponed until Part II because they are relatively recent and point the direction for future research.

2.3. FURTHER READING

Garey and Johnson (1979) and Papadimitriou (1994) provide excellent introductions to computational complexity theory. The first chapters of these books introduce several important concepts and are highly recommended. Sipser (1997) is also very good. Rogers (1967) explains the theory of recursive functions.

Harrison (1978) provides an introduction to formal language theory and the Chomsky Hierarchy. Salomaa (1973) and Thomas (1997) are more technical but provide an algebraic and logical perspectives, respectively.

Beesley and Kartunnen (2003) provides an excellent introduction in the context of morpho-phonology to regular sets, relations, and FSMs, as well as providing a useful suite of software for morpho-phonological analysis. Hopcroft et al. (2001) is another excellent source on regular patterns and FSMs in a non-linguistic context.

Rogers and Pullum (forthcoming); Rogers et al. (2009) and Rogers and Hauser (2010) provide excellent, accessible introductions to subregular language classes, though McNaughton and Papert's (1971) original treatment and analysis is irreplaceable.

Manning and Schütze (1999) and Jurafsky and Martin (2008) also address many of these topics specifically for readers interested in computational linguistics. Kracht (2003) and Kornai (2007) offer rigorous mathematical treatments.

3. SPE-style Phonology

Architecture. In the *Sound Pattern of English*, Chomsky and Halle (1968) present a theory of phonology where individual context-sensitive rules represent individual phonological generalizations which interact by their ordering. I refer to this theory as SPE-STYLE PHONOLOGY. Recalling Figure 1, the individual factors in the phonology of a language are given by these rules, and their interaction ($\times$) is given by their ordering.

One important consequence of the ordering of rules is that the later application of a rule may obscure the earlier application of another. As a consequence, a central claim of

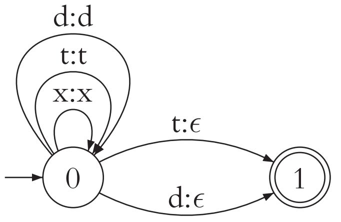

Fig 3. A finite-state machine (FSM) representation of the rule in (2). The symbol *x* is an abbreviation meaning of any symbol in the alphabet other than [t, d]. The symbol ϵ is the empty string. Labels on transitions *a:b* mean *a* is rewritten as *b*. Initial states are marked with an incoming arrow with no source and final states with double peripheries. Hence State 0 is a non-final, initial state and State 1 is a final, non-initial state.

this theory is that there are bonafide phonological generalizations that may not be 'surface-true' in the whole phonology of the language (Kiparsky 1973).

Expressivity. Johnson (1972) and Kaplan and Kay (1994) independently recognized that despite the context-sensitive rewrite rules, the functional characterizations of SPE-style phonologies are *regular*.

For example, Figure 3 shows a deterministic finite-state representation of the rule in Figure 2. This finite-state machine is a TRANSDUCER. Transducers are grammars that describe alternations (i.e. relations) in the following way. Paths along the transitions of the machine correspond to pairs of strings. Paths that begin in initial states and end in final states represent pairs of strings that are in the relation. For example, the string *wɛst* is related to *wɛs* because there is a path through the machine that starts in state 0 and ends in state 1: (w:w), (ɛ:ɛ), (s:s), (t:ϵ). (ϵ is the empty string so *t:ϵ* means that [t] is rewritten with nothing.) Similarly, *wɛst* is not related to *wɛst* because no such path exists: 0 is not a final state and the only path to 1 erases word final [t]s by writing them as ϵ. It is easy to see that the FSM in Figure 3 captures the (infinite) functional characterization of the rule in Figure 2.

The only assumption required to obtain the conclusion that phonologies are regular is that SPE-style rewrite rules are forbidden from applying to their 'own output'. By 'own output', Kaplan and Kay do not mean the rule cannot reapply to any part of the output string; they mean something much narrower: rules cannot reapply within the specific part of the string that they have rewritten (i.e. the locus of structural change). Rules can reapply to their own output, provided the part of the string currently targeted by the rule is not properly contained in what was already rewritten. Kaplan and Kay employ this condition in order to prohibit reapplication of the kind of rule in (7) which would otherwise result in non-regular patterns.

(7) $\emptyset \rightarrow$ ab / ______

If this epenthesis rule could reapply within its own locus of application, then for example, the string *ki* would be related with *abki*, *aabbki*, *aaabbbki* and so on, and the relation obtained is not regular because the pattern a^nb^n is properly context-free (Chomsky 1956).[5]

Kaplan and Kay's analysis is especially insightful because it is *constructive*. In other words, they show, under the one assumption mentioned, how each SPE-style rewrite rule describes a particular regular relation as well as how to construct the finite-state transducer which accepts exactly that relation. Then they show how the interaction of those rules via their ordering define new regular relations using the COMPOSITION operator (○).

Regular relations are closed under composition. This means that the relation obtained by composing two regular relations is also regular. In FSM terms, it means the composition of two finite-state transducers is also a finite-state transducer. Thus for any two distinct phonological generalizations F_1 and F_2, their composition $F_1 \circ F_2$ can also be interpreted as a phonological generalization. Thus the phonology of the whole language P_L can also be thought of as a single (though complicated) phonological rule. It follows, conversely, that there are many ways to decompose P_L into a sequence of ordered rules (Karttunen 1993). Not only do these facts mitigate criticisms of SPE rule-ordering on the grounds that intermediate forms are not meaningful, but it also solves the recognition problem (4–b) because transducers are bidirectional.

Another revealing aspect of their analysis is their characterization of optional rules. Instead of letting such rules apply optionally, the optionality is built into the rule itself. For example, they would characterize the optional rule of word-final stop deletion (2) with the relational and functional characterizations in Figure 4. A non-deterministic FSM representing those generalizations is shown in Figure (5). It is not clear what empirical evidence could ever distinguish between the generalizations in Figure 2 and Figure 4.

Not only do Kaplan and Kay show that SPE-style phonologies generate regular relations, they show that every regular relation can be described with a SPE-style phonology. In other words the expressivity of the SPE-style phonology is exactly the regular relations under the assumption that rules do not reapply within the loci of their structural changes.

The question of empirical adequacy of SPE-style phonology can now be addressed. Since SPE-style grammars have been used to describe virtually all known phonological processes such as local assimilation and dissimilation, stress assignment, deletions, epenthesis, metathesis, vowel and consonantal harmony and disharmony, the conclusion is that all of these processes are regular, as is the product of their interactions, provided none can reapply within the loci of their structural changes.

Cyclic application of rules (Chomsky and Halle 1968, p. 60) is a specific proposal which in principle permits rules to apply within the loci of their structural changes. Consequently, a theory which permits cyclic application cannot be regular. Kaplan and Kay (p. 365) write

> The cycle has been a major source of controversy ever since it was first proposed by Chomsky and Halle (1968), and many of the phenomena that motivated it can also be given non-cyclic descriptions. Even for cases where a non-recursive, iterative account has not yet emerged, there may be restrictions on the mode of reapplication that limit the formal power of the grammar without reducing its empirical or explanatory coverage.

	R		f		
wεst	→	wεst	(wεst,wεst)	→	1
wεst	→	wεs	(wεst,wεs)	→	1
			(wεst,wεsk)	→	0
			(wεst,wε)	→	0
	...		...		
pɪwεst	→	pɪwεst	(pɪwεst,pɪwεst)	→	1
pɪwεst	→	pɪwεs	(pɪwεst,pɪwεs)	→	1
			(pɪwεst,pɪwεsk)	→	0
			(pɪwεst,pɪwε)	→	0
	...		...		

Fig 4. Fragments of the relation and functional characterization of optional post-consonantal, word-final coronal stops deletion in English.

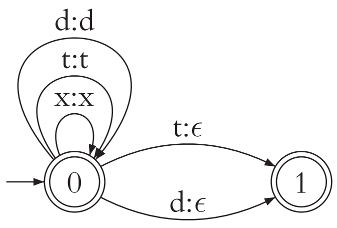

Fig 5. A finite-state machine (FSM) representation of the optional rule in (2) with optionality built-in. Symbols and notation as in Figure 3.

In other words, an empirically adequate, cyclic theory of phonology which prohibits rules from reapplying within the loci of their structural changes may be possible. To my knowledge, this has never been followed up. Until a clear case of a phonological rule necessarily reapplying within the loci of its structural change appears, the stronger, more restrictive hypothesis that phonological processes are regular appears well-supported by the impressive empirical coverage of non-cyclic SPE-phonological grammars and Kaplan and Kay's careful computational analysis.

Another challenge to Kaplan and Kay's findings comes from reduplication. But reduplication is arguably a morphological, and not a phonological, process (Inkelas and Zoll 2005; Roark and Sproat 2007).

In sum, Kaplan and Kay's work establish that regular relations adequately cover virtually all phonological processes, stated both as individual generalizations and as the whole phonology.

Complexity of the generation problem. It is well-known that the generation problem for finite state transducers is linear in the length of the input string. It follows that the generation problem (4-a) for a given SPE-style grammar is also linear in the length of the underlying form. This follows regardless of whether the grammar is represented as an ordered list of individual transducers (recalling Figure 1, $F_1 \times F_2 \ldots F_n$) or, equivalently, as a single transducer (P_L). The latter case follows trivially, and the former case follows because, under composition (i.e. when $\times = \circ$), the output of one transducer becomes the input to the next. Hence the surface form can be computed on the order of $n|u|$ time steps.

4. Conclusion

The theory of computation is relevant to phonology. It provides a universal theory of expressivity and restrictiveness. When applied to phonological patterns, Kaplan and Kay established that SPE-style grammars describe exactly the regular class of languages. Since SPE-style grammars are descriptively adequate for virtually all known phonological patterns, the hypothesis that all phonological patterns are regular is well-supported.

Part II shows how this fact leads to another: that different grammatical formalisms employed by phonologists are much more similar than generally realized because, in part, they all describe regular relations.

Finally, it is almost certainly the case that not all regular patterns are phonological ones. Thus while 'being regular' may be a necessary property of phonological patterns, it is not a sufficient one. Thus, computational analysis points to a further hypothesis: all phonological patterns are subregular. Part II also points the way for this future research by introducing subregular language classes as they relate to phonological patterns.

Short Biography

Jeffrey Heinz's research is located at the intersection of theoretical linguistics (specifically phonology), theoretical computer science, computational linguistics, grammatical inference and cognitive science. He has authored or co-authored articles in these areas for the journals Phonology, Linguistic Inquiry, and the Journal of Child Language; chapters in the forthcoming Blackwell Companion to Phonology and Cambridge Handbook of Developmental Linguistics; and papers in the proceedings of the annual meeting of the Association for Computational Linguistics and of the International Colloquium on Grammatical Inference. His current research centers on the hypotheses that phonology is subregular and that phonological learning is modular. He holds a BS in Mathematics and a BA in Linguistics from the University of Marlyand, College Park, and a PhD in Linguistics from the University of California, Los Angeles. Jeffrey Heinz is currently an assistant professor at the University of Delaware.

Notes

* Correspondence address: Jeffrey Heinz, Department of Linguistics and Cognitive Science, University of Delaware, 42 E. Delaware Ave, Newark, DE 19716, USA. E-mail: heinz@udel.edu

[1] In fact, the only essential aspect of the definition of 'being regular' in (1) is the word 'finitely'. This is because for *any* logically possible pattern, there is a partition of the set of all logically possible words which satisfies (1-a) and (1-b). It's just that for most logically possible patterns, this partition is infinite.

[2] Technically, computable real numbers.

[3] Non-deterministic algorithms guess which sequence of actions to follow and then at the end of the process, a 'yes' output verifies the choices made, but a 'no' output means another guess ought to be explored. See Garey and Johnson (1979, 29) and Papadimitriou (1994, 45) for more discussion.

[4] To see this, consider a partition of Σ^* into two blocks: Block 1 contains all and only those words with an even number of vowels and Block 2 contains all and only those words with an odd number of vowels. Clearly (1-a) is satisfied. Also, it is not hard to verify that for any two strings u,v in the same block, there is no string x that can be concatenated to them to yield strings ux,vx belonging to different blocks (1-b).

[5] To illustrate, consider a partition of Σ^* into two blocks: Block 1 contains words a^nb^n for all n and Block 2 contains all other words. Words *aab* and *b* both belong to Block 2, but concatenation of *b* to these yields a word in Block 1 (*aabb*) and a word in Block 2 (*bb*), violating (1-b). In fact, no *finite* partition of Σ^* satisfies both (1-a) and (1-b).

Works Cited

Barton, G. Edward, Robert Berwick, and Eric Ristad. 1987. Computational complexity and natural language. Cambridge, MA: MIT Press.

Beesley, Kenneth, and Lauri Kartunnen. 2003. Finite state morphology. Stanford, CA: CSLI Publications.

Bird, Steven, and Mark Ellison. 1994. One-level phonology. Computational Linguistics 20. 55–90.

Carson-Berndsen, Julie. 1998. Time map phonology: finite state models and event logics in speech recognition. the Netherlands: Kluwer Academic Publishers.

Chomsky, Noam. 1956. Three models for the description of language. IRE Transactions on Information Theory 113124. IT-2.

——, and Morris Halle. 1968. The sound pattern of English. New York: Harper and Row.

Coetzee, Andries. 2004. What it means to be a loser: non-optimal candidates in Optimality Theory. Doctoral dissertation, University of Massachusetts, Amherst.

Eisner, Jason. 1997. Efficient generation in primitive Optimality Theory. Proceedings of the 35th annual ACL and 8th EACL, 313–20. Madrid: Association for Computational Linguistics.

Garey, M. R., and D. S. Johnson. 1979. Computers and intractability: a guide to the theory of NP-completeness. New York: W. H. Freeman.

Gibbon, Daffyd. 1987. Finite state processing of tone languages. Proceedings of the European Association for Computational Linguistics, 291–7. Copenhagen: Association for Computational Linguistics.

Guy, G. R. 1980. Variation in the group and the individual: the case of final stop deletion. Locating language in time and space, ed. by William Labov, 1–36. New York: Academic Press.

——, and C. Boberg. 1997. Inherent variability and the obligatory contour principle. Language Variation and Change 9. 149–64.

Harrison, Michael A. 1978. Introduction to formal language theory. Reading, MA: Addison-Wesley Publishing Company.

Hopcroft, John, Rajeev Motwani, and Jeffrey Ullman. 2001. Introduction to automata theory, languages, and computation. Boston, MA: Addison-Wesley.

Inkelas, Sharon, and Cheryl Zoll. 2005. Reduplication: doubling in morphology. Cambridge: Cambridge University Press.

Johnson, C. Douglas. 1972. Formal aspects of phonological description. The Hague: Mouton.

Jurafsky, Daniel, and James Martin. 2008. Speech and language processing: an introduction to natural language processing, speech recognition, and computational linguistics. 2nd ed. Upper Saddle River, NJ: Prentice-Hall.

Kaplan, Ronald, and Martin Kay. 1994. Regular models of phonological rule systems. Computational Linguistics 20. 331–78.

Karttunen, Lauri. 1993. Finite-state constraints. The last phonological rule: reflections on constraints and derivations, ed. by John Goldsmith, 173–94. Chicago: University of Chicago Press.

Kiparsky, Paul. 1973. Abstractness, opacity and global rules. Three dimensions of linguistic theory, ed. by O. Fujimura and O. Fujimura, 57–86. Tokyo: TEC.

Kornai, Andras. 1994. Formal phonology. New York: Garland.

Kornai, András. 2007. Mathematical linguistics. Advanced Information and Knowledge Processing. London: Springer Verlag.

Kracht, Marcus. 2003. The mathematics of language. Berlin: Mouton de Gruyter.

Manning, Christopher, and Hinrich Schütze. 1999. Foundations of statistical natural language processing. Cambridge, MA: MIT Press.

Marr, David. 1980. Vision. Cambridge, MA: W.H. Freeman and Company.

McNaughton, Robert, and Seymour Papert. 1971. Counter-free automata. Cambridge, MA: MIT Press.

Papadimitriou, Christon. 1994. Computational complexity. Reading, MA: Addison Wesley.

Prince, Alan, and Paul Smolensky. 2004. Optimality theory: constraint interaction in generative grammar. Malden, MA: Blackwell Publishing.

Roark, Brian, and Richard Sproat. 2007. Computational approaches to morphology and syntax. Oxford: Oxford University Press.

Rogers, Hartley. 1967. Theory of recursive functions and effective computability. New York: McGraw Hill Book Company.

Rogers, James, and Marc Hauser. 2010. The use of formal languages in artificial language learning: a proposal for distinguishing the differences between human and nonhuman animal learners. Recursion and human language, ed. by Harry van der Hulst, Chap. 12, 213–32. Berlin, Germany: De Gruyter Mouton.

——, and Geoffrey Pullum. Forthcoming. Aural pattern recognition experiments and the subregular hierarchy. Journal of Logic, Language and Information. Paper presented at the 10th meeting of the Association for Mathematics of Language in 2007, Los Angeles, CA. <http://www.cs.earlham.edu/~jrogers/mol10.pdf>.

——, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. 2009. On languages piecewise testable in the strict sense. Proceedings of the 10th and 11th Biennial Conference on The Mathematics of Language (MOL'07/09), ed. by C. Ebert, G. Jäger, and J. Michaelis, 255–65. Berlin, Heidelberg: Springer-Verlag.

Salomaa, Arto. 1973. Formal languages. New York: Academic Press.

Sipser, Michael. 1997. Introduction to the theory of computation. Boston, MA: PWS Publishing Company.

Thomas, Wolfgang. 1997. Languages, automata, and logic. Vol. 3, Chap. 7, 389–455. New York: Springer.