An Efficient Implementation of Phonological Rules using Finite-State Transducers

I. Lee Hetherington

Spoken Language Systems Group
MIT Laboratory for Computer Science
Cambridge, MA 02139 USA
ilh@sls.lcs.mit.edu

Abstract

Context-dependent phonological rules are used to model the mapping from phonemes to their varied phonetic surface realizations. Others, most notably Kaplan and Kay, have described how to compile general context-dependent phonological rewrite rules into finite-state transducers. Such rules are very powerful, but their compilation is complex and can result in very large nondeterministic automata. In this paper we present a simplified rewrite rule system and a technique to efficiently compile such a system into finite-state transducers.

1. Introduction

In our SUMMIT speech recognition system, we have always used phonological rules to model pronunciation variability [1]. We use such rules to transform word phonemic baseforms to graphs of alternate pronunciations at the phonetic level. These rules account for many different phonological phenemona such as place assimilation, gemination, epenthetic silence insertion, alveolar stop flapping, and schwa deletion. These rules primarily depend on input-level phonemic context, but phonetic realizations may also depend on nearby realizations.

Kaplan and Kay's classic paper [2] provides a comprehensive treatment of phonological rewrite rules of the form

$$\phi \rightarrow \psi / \lambda \_\_ \rho, \qquad (1)$$

where ϕ is the target of replacement, ψ the replacement, λ the left context, and ρ the right context, with all four of these components being regular expressions. There are many variations presented, including left-to-right and right-to-left directional application, simultaneous application, optional and obligatory replacement, and the simultaneous application of rules in batches. Further, they show how various phonological rule systems can be mapped to their framework of regular relations.

Karttunen's replace operator [3] is equivalent to (1) and has varieties allowing the λ and ρ contexts to apply to the input or output level, as well as an operator called directed replacement [4] that maximally matches potentially overlapping ϕ instances contained in the input.

Both Kaplan and Kay and Karttunen provide mathematical formulations of how such rules can be implemented as a cascade of finite-state transductions. However, the compilation of rules to finite-state transducers using their formulations can be slow and involve large nondeterministic automata.

Mohri and Sproat [5] present an alternative formulation for left-to-right, obligatory rules that is computationally more efficient. However, *each* rule is still compiled into no less than five component transducers, and the transducers for individual rules are composed together to produce a transducer for a whole set of rules.

In this paper we present a phonological rule system and a technique to compile a set of rules into one or two finite-state transducers. Our rules represent a simplification to those of Kaplan and Kay and yet allow us to express much of the phonological variability we observe in interactions between real users and our conversational systems. In fact, many of our rules are a result of studying misrecognitions within our JUPITER weather information domain [6].

2. Rules

2.1. Rule Specification

Individual rules consist of four components: the target ϕ, the replacement ψ, the left context λ, and the right context ρ, and we write them in the following form:

$$\{\lambda_1, \ldots, \lambda_m\} \quad \phi \quad \{\rho_1, \ldots, \rho_n\} \quad \Rightarrow \quad \psi \quad ;$$

In contrast to the rules of Kaplan and Kay, in which all four components are regular expressions, we have

This research was supported by DARPA under Contract N66001-99-1-8904 monitored through the Naval Command, Control, and Ocean Surveillance Center.

made the following simplifications. For input alphabet Σ_i, we have $\phi \in \Sigma_i$, $\lambda \subseteq \Sigma_i$, and $\rho \subseteq \Sigma_i$, with an empty context specification $\lambda = \emptyset$ or $\rho = \emptyset$ indicating no contextual dependence.

The contexts λ and ρ are both expressed at the input level. This is analogous to Kaplan and Kay's simultaneous rules [2] and Karttunen's upward-oriented rules [3].

In addition we have added functionality to the replacement regular expression $\psi \subset \Sigma_o^*$, for output alphabet Σ_o, allowing for the expression of local surface-level constraints. This gives us some capability to refer to context on both the input and surface levels simultaneously. We will describe this in more detail in Section 2.3.

2.2. Rule Application

Conceptually, rules are applied at every position of the input simultaneously. There is no orientation toward one direction or another. Furthermore, for each input position, the *first* rule with matching ϕ, λ, and ρ expressions is applied. In our system, we require at least one rule to match every possible (ϕ, λ, ρ); this is generally achieved by specifying a final context-independent $\sigma \Rightarrow \sigma$ for every input symbol $\sigma \in \Sigma_i$. As formulated, rule application is deterministic, with exactly one rule firing for every input in context.

Generally, for rules with the same target ϕ, rules are ordered from more to less specific λ and ρ contexts. For example, applying the ordered rules

$$\begin{array}{ccccc} \{a\} & a & \{a\} & \Rightarrow & a_1 \quad ; \\ \{\} & a & \{a\} & \Rightarrow & a_2 \quad ; \\ \{a\} & a & \{\} & \Rightarrow & a_3 \quad ; \\ \{\} & a & \{\} & \Rightarrow & a_4 \quad ; \end{array}$$

to $a{\cdot}a{\cdot}a$ would produce $a_2{\cdot}a_1{\cdot}a_3$.

Our rules are *batch* rules in that they are all applied simultaneously across all input positions in one batch. As such, the ϕ, λ, and ρ all refer to the input and not the output of previously applied rules; there are no previously applied rules. If it is desired to have rules be sequential in the sense that some rules will operate on the output of previous rules, then the rules must be divided into separate batches, and these batches applied sequentially.

Furthermore, our rules are obligatory, meaning that if a rule matches, ψ *will* replace ϕ. It is trivial to make a rule behave as if it were optional by including ϕ within ψ. This is trivial because $\phi \in \Sigma_i$, a single symbol and not a regular expression.

2.3. Surface-Level Constraints

While λ and ρ express required context at the input level only, we do have the capability to enforce context constraints at the surface or output level through the use of auxiliary symbols contained at the beginning or ending of the replacement ψ. In fact, there are two mechanisms available: one referring directly to adjacent surface symbols and one referring to user-specified connection symbols. We will demonstrate both mechanisms with examples.

Consider the following set of interrelated rules:

$$\begin{array}{ccccl} \{\} & d & \{y\} & \Rightarrow & dcl\ (d \mid jh) \quad ; \\ \{\} & t & \{y\} & \Rightarrow & tcl\ (t \mid ch) \quad ; \\ \{d,\ t\} & y & \{\} & \Rightarrow & y \mid <\{ch,\ jh\} \quad ; \end{array}$$

Together, these rules could model the palatalization of the stops in "would you" and "hit you" by mapping a sequence $d{\cdot}y$ to $dcl{\cdot}jh$, as well as the non-palatalized $dcl{\cdot}d{\cdot}y$. Here, $<\{ch,\ jh\}$ denotes an explicit reference to either ch or jh on the surface to the left.

We can use $<\{\cdot\}$ and $\{\cdot\}>$ to specify sets of required left and right surface-level symbols, respectively. The surface context specification itself is not part of the final replacement, but as we will see later is used internally to enforce constraints.

A second mechanism is available for expressing surface-level constraints that is somewhat more powerful. We can specify a pair of auxiliary symbols that we can use in the ψ expressions of different rules to enforce particular connections at the surface level. For example, consider the following set of rules:

$$\begin{array}{ccccl} \{\} & s & \{t\} & \Rightarrow & s \mid sr\ r\$ \quad ; \\ \{s\} & t & \{r\} & \Rightarrow & t \mid \$r\ t \quad ; \end{array}$$

If we have previously specified that *r$* is connected to *$r* (a retroflex flag of sorts, right- and left-hand versions, respectively), then together these rules say that $s{\cdot}t{\cdot}r$ can be realized as $s{\cdot}t{\cdot}r$ or $sr{\cdot}t{\cdot}r$, where sr could represent an s in a retroflex environment. This allows somewhat longer-distance surface constraints. Here, the input s can "look" beyond the input and surface t to see the r to the right. However, in practice we might instead do the following:

$$\begin{array}{ccccl} \{\} & s & \{t\} & \Rightarrow & s \mid sr\ \{tr\}> \quad ; \\ \{s\} & t & \{r\} & \Rightarrow & t \mid tr \quad ; \end{array}$$

to produce $s{\cdot}t{\cdot}r$ or $sr{\cdot}tr{\cdot}r$. In this case, because the surface tr encodes the presence of the r, the rule for s can refer to it with $\{\cdot\}>$ notation. Still, we often find auxiliary connecting symbols useful because they can be mnemonic and because they can represent whole sets of surface labels compactly.

3. Compilation to Transducers

We construct the transducer P representing the set of rules by factoring it into three component transducers:

$$P = I \circ R \circ S.$$

I enforces input contextual constraints, λ_i and ρ_i, and outputs the particular rule i to apply to each input symbol (see Section 3.1). R performs replacement $\phi_i \Rightarrow \psi_i$ for all i (see Section 3.2). Finally, S enforces surface contextual constraints (see Section 3.3).

3.1. Input-Constrained Rule Selection: I

The role of I is to select, for each input symbol, the first rule that matches its immediate left and right context. Constructing a deterministic left-to-right I would, in general, require I to introduce a one-symbol delay so that it can observe the right context ρ. However, if we factor I as $I = I_r \circ I_l$, we can deterministically apply I_r right to left and I_l left to right, without introducing any undesirable delay. In this factorization, we have I_r map each input symbol σ to the subset of σ's rules compatible with the right input context. Then, we have I_l map each such rule subset to the first rule compatible with the left input context.

We construct the reverse of I_r, $\mathrm{rev}(I_r)$, to be applied right to left, as follows. For input alphabet Σ_i, define $\Sigma'_i = \Sigma_i \cup \{\epsilon\}$. For each $\sigma \in \Sigma'_i$, create a state labeled σ and make it final. Make the state labeled ϵ the initial state. For each $\sigma_1 \in \Sigma'_i$, $\sigma_2 \in \Sigma_i$, add a transition from state σ_1 to σ_2 labeled $\sigma_2{:}r^{\triangleright}(\sigma_2, \sigma_1)$, where $r^{\triangleright}(\sigma_2, \sigma_1)$ denotes an auxiliary label representing the subset of rules for σ_2 consistent with a σ_1 to the right. $\mathrm{rev}(I_r)$ is deterministic by construction.

We construct I_l as follows. For each state $\sigma \in \Sigma'_i$, create a state labeled σ and make it final. Make the state labeled ϵ the initial state. For each $\sigma_1 \in \Sigma'_i$, $\sigma_2 \in \Sigma_i$, $\sigma_3 \in \Sigma'_i$, for each unique subset $r^{\triangleright}(\sigma_2, \sigma_3)$ add a transition from state σ_1 to state σ_2 labeled $r^{\triangleright}(\sigma_2, \sigma_3){:}r^{\triangleleft}(\sigma_1, r^{\triangleright}(\sigma_2, \sigma_3))$, where $r^{\triangleleft}(\sigma_1, r^{\triangleright}(\sigma_2, \sigma_3))$ denotes an auxiliary label representing the first rule for σ_2 in subset $r^{\triangleright}(\sigma_2, \sigma_3)$ consistent with a left context of σ_1. I_l is deterministic by construction.

See Figure 1 for an example set of rules, Figure 2 for the corresponding I_r, and Figure 3 for the corresponding I_l. In the figures, an auxiliary label such as b{2,3} denotes the rule subset constraining the second and third rules for b, and a label such as b[2] denotes the second rule for b.

3.2. Replacement: R

Construction of the replacement transducer R is straightforward because input and surface contextual constraints are handled by the other transducers. Thus, all R has to do is map each rule label ϕ_i to its replacement ψ_i. Thus the rule replacement transducer

$$R = \left(\textstyle\bigcup_i \phi_i \times \psi_i\right)^*,$$

the union of all individual replacements $\phi_i \times \psi_i$ with closure. We can construct each of these replacements by concatenating $\phi_i \times \{\epsilon\} = \phi_i{:}\,\epsilon$ with $\{\epsilon\} \times \psi_i$, where $\{\epsilon\} \times \psi_i$ is the transducer representing ψ_i on its output side with ϵ on its input side.

When compiling regular expressions into replacements ψ_i, we transform surface constraints as follows:

$$\begin{aligned} <\{\sigma_1, \ldots, \sigma_n\} &\rightarrow s^{\triangleleft}_{\sigma_1} \cup \cdots \cup s^{\triangleleft}_{\sigma_n}, \\ \{\sigma_1, \ldots, \sigma_n\}> &\rightarrow s^{\triangleright}_{\sigma_1} \cup \cdots \cup s^{\triangleright}_{\sigma_n}, \\ \$\alpha &\rightarrow c^{\triangleleft}_{\alpha}, \text{ and} \\ \alpha\$ &\rightarrow c^{\triangleright}_{\alpha}, \end{aligned}$$

where $s^{\triangleleft}_{\sigma}$ and $s^{\triangleright}_{\sigma}$ denote auxiliary symbols representing a required surface σ to the left and right, respectively, and $c^{\triangleleft}_{\alpha}$ and $c^{\triangleright}_{\alpha}$ denote auxiliary symbols representing left and right user-specified connections, respectively. The constraints on the placement of these auxiliary symbols will be enforced by S, described below. For the purposes of construction of R, we can disregard such constraints. In general, R is not deterministic because a relation $\phi_i \times \psi_i$ can be one to many.

See Figure 4 for the minimized R corresponding to the rules in Figure 1. The surface constraint auxiliary symbols used in the transducer are $s^{\triangleright}_{\mathrm{B2}} =$ B2> and $s^{\triangleleft}_{\mathrm{A2}} =$ <A2.

3.3. Enforcing Surface Constraints: S

The surface constraint transducer S enforces surface constraints and erases any auxiliary symbols in $\{s^{\triangleleft}_{\sigma}, s^{\triangleright}_{\sigma}, c^{\triangleleft}_{\alpha}, c^{\triangleright}_{\alpha}\}$ while passing through symbols σ of the output alphabet Σ_o.

We can construct transducer S as

$$S = (S_1 \cup S_2 \cup S_3)^*,$$

where the individual components are

$$\begin{aligned} S_1 &= \textstyle\bigcup_{\sigma} (s^{\triangleright}_{\sigma}{:}\,\epsilon)^* \cdot \sigma{:}\,\sigma \cdot (s^{\triangleleft}_{\sigma}{:}\,\epsilon)^*, \\ S_2 &= \textstyle\bigcup_{\sigma_1,\sigma_2} \sigma_1{:}\,\sigma_1 \cdot (s^{\triangleright}_{\sigma_2}{:}\,\epsilon \;\cup\; s^{\triangleleft}_{\sigma_1}{:}\,\epsilon)^* \cdot \sigma_2{:}\,\sigma_2, \\ S_3 &= \textstyle\bigcup_{\alpha} c^{\triangleright}_{\alpha}{:}\,\epsilon \cdot c^{\triangleleft}_{\alpha}{:}\,\epsilon. \end{aligned}$$

The first component, S_1, enforces left-only or right-only surface constraints between consecutive output symbols. The second component, S_2, enforces mixed left and right surface constraints, e.g., as in the rules

$$\begin{array}{cccccc} \{\} & a & \{b\} & \Rightarrow & a_1 \mid a_2\ \{b_2\}> & ; \\ \{a\} & b & \{\} & \Rightarrow & b_1 \mid <\{a_2\}\ b_2 & ; \end{array}$$

Finally, the third component, S_3, enforces constraints for the user-specified connection symbols. All three components enforce surface constraints, erase auxiliary symbols, and pass Σ^*_o sequences unchanged. As constructed, S is non-deterministic, but it can be determinized.

See Figure 5 for the minimized S corresponding to the rules of Figure 1. Only those $s^{\triangleleft}_{\sigma}$ and $s^{\triangleright}_{\sigma}$ used in R are present.

3.4. Applying Rules

We now have all the components to compute the right-to-left component of the rules $P_r = \mathrm{rev}(I_r)$ and the left-to-right component $P_l = I_l \circ R \circ S$. We apply rules to a given input X as follows:

$$X \circ P = \mathrm{rev}(\mathrm{rev}(X) \circ P_r) \circ P_l.$$

If it is desired to have a single, left-to-right transducer to implement a set of rules, it can be constructed as $P = I_r \circ I_l \circ R \circ S$. I_r could be left non-deterministic so as to not introduce a one symbol delay, or it could be determinized and thus introduce delay.

4. SUMMIT's Rules

Within our SUMMIT speech recognition system, we currently use 230 phonological rules for English, 34 of the trivial form $\sigma \Rightarrow \sigma$, to map from baseform pronunciations to a variety of allowable phonetic realizations. We have 63 "phonemic" input symbols, with variations including deletable stop releases, flappable stops, and silence. The rules output 71 "phonetic" output symbols. Compiling these rules takes 2.0s on an 866MHz Pentium III and produces a minimized P_r with 20 states and 1,218 transitions and a P_l with 134 states and 13,642 transitions. Minimized P has 294 states and 21,009 transitions.

5. Conclusion and Future Work

We have presented a system for expressing phonological rules and a formulation for converting a batch of such rules to one or two finite-state transducers. Such rules allow for both input-level and surface-level constraints. The simplified form of ϕ, λ, and ρ allow for particularly efficient rule compilation involving significantly fewer intermediate transducers compared to [2, 3, 5]. Altogether, we find rule compilation and application is more than 100 times faster than the non-transducer-based technique we previously used.

It is very straightforward to add weights to the rules to produce weighted finite-state transducers. We plan to do this as well as automatically train such weights from a large corpus of paired phonetic and orthographic forced recognition paths. We also plan to address the issue of dynamically adding words to the system vocabulary while respecting the inter-word context dependency of the phonological rules.

6. Acknowledgements

Thanks to T. J. Hazen, who composed most of the phonological rules currently used within our systems and to Victor Zue, Jim Glass, and Stephanie Seneff for providing valuable feedback on this paper.

7. References

[1] V. Zue, J. Glass, D. Goodine, M. Phillips, and S. Seneff, "The SUMMIT speech recognition system: Phonological modelling and lexical access," in *Proc. Intl. Conf. on Acoustics, Speech, and Signal Processing*, Albuquerque, Apr. 1990, pp. 49–52.

[2] R. M. Kaplan and M. Kay, "Regular models of phonological rule systems," *Computational Linguistics*, vol. 20, no. 3, pp. 331–378, Sept. 1994.

[3] L. Karttunen, "The replace operator," in *Finite-State Language Processing*, E. Roche and Y. Schabes, Eds., pp. 117–147. MIT Press, Cambridge, Mass., 1997.

[4] L. Karttunen, "Directed replacement," in *Proc. 34th Mtg. of the Assoc. for Computational Linguistics*, Santa Cruz, June 1996, pp. 108–115.

[5] M. Mohri and R. Sproat, "An efficient compiler for weighted rewrite rules," in *Proc. 34th Mtg. of the Assoc. for Computational Linguistics*, Santa Cruz, June 1996, pp. 231–238.

[6] V. Zue, S. Seneff, J. R. Glass, J. Polifroni, C. Pao, T. J. Hazen, and L. Hetherington, "JUPITER: A telephone-based conversational interface for weather information," *IEEE Transactions on Speech and Audio Processing*, vol. 8, no. 1, pp. 85–96, Jan. 2000.

$$\begin{array}{cccccl}
\{b\} & a & \{\} & \Rightarrow & A \mid A2\ \{B2\}> & ; \\
\{\} & a & \{\} & \Rightarrow & A & ; \\
\{\} & b & \{a\} & \Rightarrow & B & ; \\
\{a\} & b & \{\} & \Rightarrow & B \mid <\{A2\}\ B2 & ; \\
\{\} & b & \{\} & \Rightarrow & B & ;
\end{array}$$

Figure 1: Example rules for demonstrating the construction of rev(I_r) of Figure 2, I_l of Figure 3, R of Figure 4, and S of Figure 5.

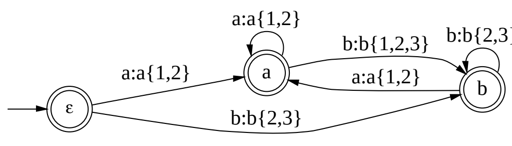

Figure 2: Example rev(I_r), mapping from input symbols to rule subsets compatible with right context, applied right-to-left.

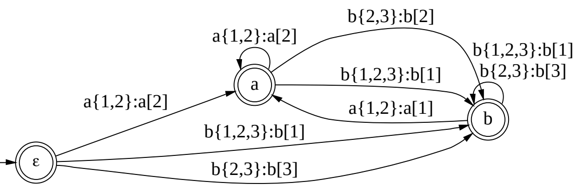

Figure 3: Example I_l, mapping from rule subsets to the first rule compatible with left context.

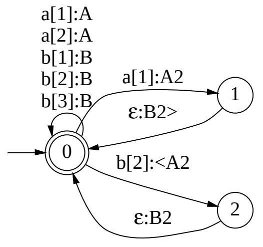

Figure 4: Example R performing rule replacement.

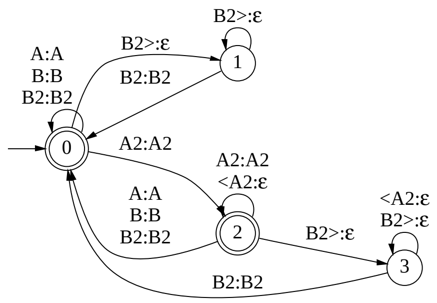

Figure 5: Example S enforcing all surface constraints.