Chapter 11
Computational Phonology

bidakupadotigolabubidakutupiropadotigolabutupirobidaku...
Word segmentation stimulus (Saffran et al., 1996a)

computational phonology

Recall from Ch. 7 that **phonology** is the area of linguistics that describes the systematic way that sounds are differently realized in different environments, and how this system of sounds is related to the rest of the grammar. This chapter introduces **computational phonology**, the use of computational models in phonological theory.

One focus of computational phonology is on computational models of phonological representation, and on how to use phonological models to map from surface phonological forms to underlying phonological representation. Models in (non-computational) phonological theory are generative; the goal of the model is to represent how an underlying form can generate a surface phonological form. In computation, we are generally more interested in the alternative problem of **phonological parsing**; going from surface form to underlying structure. One major tool for this task is the finite-state automaton, which is employed in two families of models: **finite-state phonology** and **optimality theory**.

A related kind of phonological parsing task is **syllabification**: the task of assigning syllable structure to sequences of phones. Besides its theoretical interest, syllabification turns out to be a useful practical tool in aspects of speech synthesis such as pronunciation dictionary design. We therefore summarize a few practical algorithms for syllabification.

Finally, we spend the remainder of the chapter on the key problem of how phonological and morphological representations can be learned.

11.1 Finite-State Phonology

Ch. 3 showed that spelling rules can be implemented by transducers. Phonological rules can be implemented as transducers in the same way; indeed the original work by Johnson (1972) and Kaplan and Kay (1981) on finite-state models was based on phonological rules rather than spelling rules. There are a number of different models of **computational phonology** that use finite automata in various ways to realize phonological rules. We will describe the **two-level morphology** of Koskenniemi (1983) first mentioned in Ch. 3. Let's begin with the intuition, by seeing the transducer in Fig. 11.1 which models the simplified flapping rule in (11.1):

(11.1) $$/t/ \rightarrow [dx] \;/\; \acute{V} \;\_\_\_\; V$$

The transducer in Fig. 11.1 accepts any string in which flaps occur in the correct

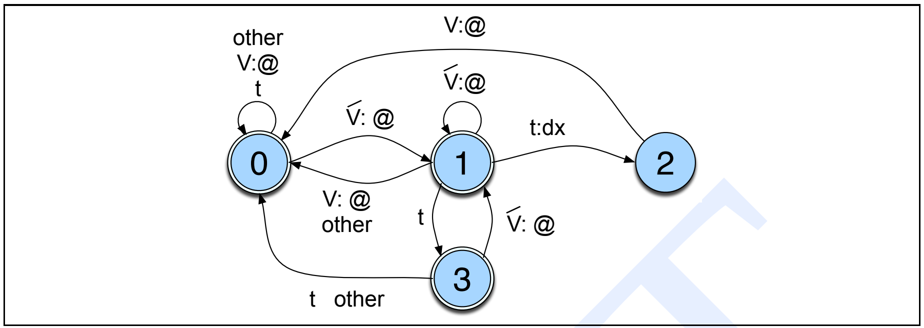


Figure 11.1 Transducer for English Flapping: ARPAbet “dx” indicates a flap, and the “other” symbol means “any feasible pair not used elsewhere in the transducer”. “@” means “any symbol not used elsewhere on any arc”.

places (after a stressed vowel, before an unstressed vowel), and rejects strings in which flapping doesn’t occur, or in which flapping occurs in the wrong environment.[1]

We’ve seen both transducers and rules before; the intuition of two-level morphology is to augment the rule notation to correspond more naturally to transducers. We motivate his idea by beginning with the notion of **rule ordering**. In a traditional phonological system, many different phonological rules apply between the lexical form and the surface form. Sometimes these rules interact; the output from one rule affects the input to another rule. One way to implement rule-interaction in a transducer system is to run transducers in a *cascade*. Consider, for example, the rules that are needed to deal with the phonological behavior of the English noun plural suffix *-s*. This suffix is pronounced [ix z] after the phones [s], [sh], [z], [zh], [ch], or [jh] (so *peaches* is pronounced [p iy ch ix z], and *faxes* is pronounced [f ae k s ix z]), [z] after voiced sounds (*pigs* is pronounced [p ih g z]), and [s] after unvoiced sounds (*cats* is pronounced [k ae t s]). We model this variation by writing phonological rules for the realization of the morpheme in different contexts. We first need to choose one of these three forms ([s], [z], [ix z]) as the “lexical” pronunciation of the suffix; we chose [z] only because it turns out to simplify rule writing. Next we write two phonological rules. One, similar to the E-insertion spelling rule of page 63, inserts an [ix] after a morpheme-final sibilant and before the plural morpheme [z]. The other makes sure that the -*s* suffix is properly realized as [s] after unvoiced consonants.

(11.2) $\epsilon \rightarrow$ ix / [+sibilant] ˆ ___ z #

(11.3) z $\rightarrow$ s / [-voice] ˆ ___ #

These two rules must be *ordered*; rule (11.2) must apply before (11.3). This is because the environment of (11.2) includes z, and the rule (11.3) changes z. Consider running both rules on the lexical form *fox* concatenated with the plural -*s*:

Lexical form:	f aa k ˆ z
(11.2) applies:	f aa k s ˆ ix z
(11.3) doesn’t apply:	f aa k sˆ ix z

[1] For pedagogical purposes, this example assumes (incorrectly) that the factors that influence flapping are purely phonetic and are non-stochastic.

bleeding

If the devoicing rule (11.3) was ordered first, we would get the wrong result. This situation, in which one rule destroys the environment for another, is called **bleeding**:[2]

Lexical form:	f aa k s ˆ z
(11.3) applies:	f aa k s ˆ s
(11.2) doesn't apply:	f aa k s ˆ s

As was suggested in Ch. 3, each of these rules can be represented by a transducer. Since the rules are ordered, the transducers would also need to be ordered. For example if they are placed in a **cascade**, the output of the first transducer would feed the input of the second transducer.

Many rules can be cascaded together this way. As Ch. 3 discussed, running a cascade, particularly one with many levels, can be unwieldy, and so transducer cascades are usually replaced with a single more complex transducer by **composing** the individual transducers.

Koskenniemi's method of **two-level morphology** that was sketchily introduced in Ch. 3 is another way to solve the problem of rule ordering. Koskenniemi (1983) observed that most phonological rules in a grammar are independent of one another; that feeding and bleeding relations between rules are not the norm.[3] Since this is the case, Koskenniemi proposed that phonological rules be run in parallel rather than in series. The cases where there is rule interaction (feeding or bleeding) we deal with by slightly modifying some rules. Koskenniemi's two-level rules can be thought of as a way of expressing **declarative constraints** on the well-formedness of the lexical-surface mapping.

Two-level rules also differ from traditional phonological rules by explicitly coding when they are obligatory or optional, by using four differing **rule operators**; the ⇔ rule corresponds to traditional **obligatory** phonological rules, while the ⇒ rule implements **optional rules**:

Rule type	**Interpretation**
a:b ⇐ c __ d	*a* is **always** realized as *b* in the context *c* __ *d*
a:b ⇒ c __ d	*a* may be realized as *b* **only** in the context *c* __ *d*
a:b ⇔ c __ d	*a* must be realized as *b* in context *c* __ *d* and nowhere else
a:b /⇐ c __ d	*a* is **never** realized as *b* in the context *c* __ *d*

The most important intuition of the two-level rules, and the mechanism that lets them avoid feeding and bleeding, is their ability to represent constraints on *two levels*. This is based on the use of the colon (":"), which was touched on very briefly in Ch. 3. The symbol *a:b* means a lexical *a* that maps to a surface *b*. Thus *a:b* ⇔ *:c* __ means *a* is realized as *b* after a **surface** c. By contrast *a:b* ⇔ *c:* __ means that *a* is realized as *b* after a **lexical** c. As discussed in Ch. 3, the symbol *c* with no colon is equivalent to *c:c* that means a lexical *c* which maps to a surface *c*.

[2] If we had chosen to represent the lexical pronunciation of *-s* as [s] rather than [z], we would have written the rule inversely to voice the *-s* after voiced sounds, but the rules would still need to be ordered; the ordering would simply flip.

[3] Feeding is a situation in which one rule creates the environment for another rule and so must be run beforehand.

Fig. 11.2 shows an intuition for how the two-level approach avoids ordering for the ix-insertion and z-devoicing rules. The idea is that the z-devoicing rule maps a *lexical* z-insertion to a *surface* s and the ix rule refers to the *lexical* z.

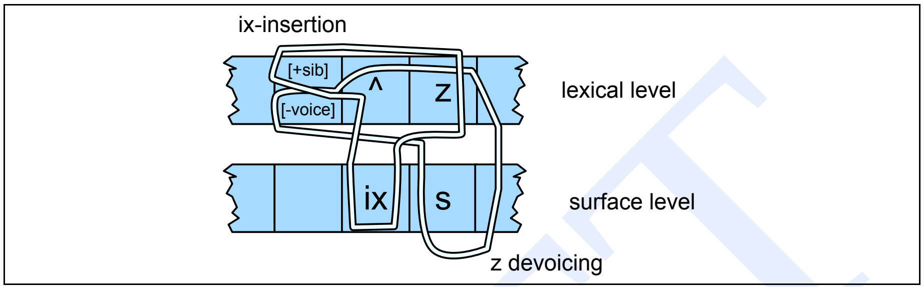


Figure 11.2 The constraints for the i-insertion and z-devoicing rules both refer to a *lexical* z, not a *surface* z.

The two-level rules that model this constraint are shown in (11.4) and (11.5):

(11.4) $\epsilon : \mathrm{ix} \Leftrightarrow \text{[+sibilant]: } \hat{}\ \underline{\quad}\ \text{z: \#}$

(11.5) $\mathrm{z} : \mathrm{s} \Leftrightarrow \text{[-voice]: } \hat{}\ \underline{\quad}\ \text{\#}$

As Ch. 3 discussed, there are compilation algorithms for creating automata from rules. Kaplan and Kay (1994) give the general derivation of these algorithms, and Antworth (1990) gives one that is specific to two-level rules. The automata corresponding to the two rules are shown in Fig. 11.3 and Fig. 11.4. Fig. 11.3 is based on Fig. 3.17 of Ch. 3; see page 64 for a reminder of how this automaton works. Note in Fig. 11.3 that the plural morpheme is represented by z:, indicating that the constraint is expressed about a lexical rather than surface z.

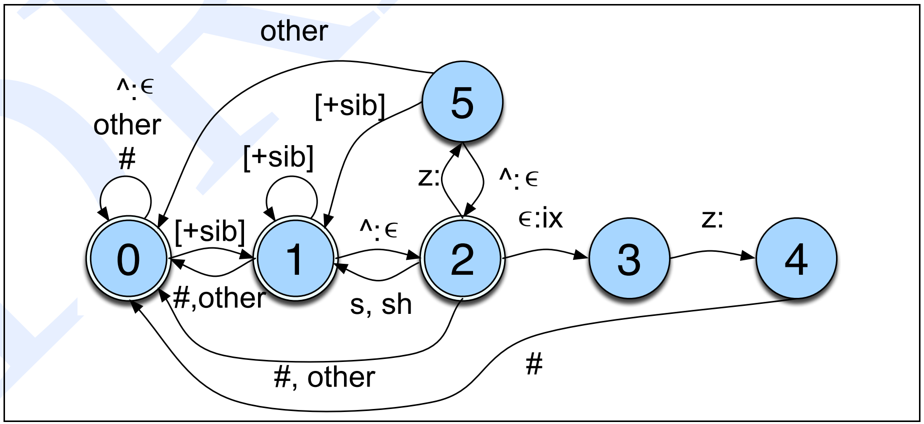


Figure 11.3 The transducer for the ix-insertion rule 11.2. The rule can be read *whenever a morpheme ends in a sibilant, and the following morpheme is word-final* z*, insert [ix].*

Fig. 11.5 shows the two automata run in parallel on the input [f aa k s ˆ z]. Note that both the automata assumes the default mapping ˆ:ϵ to remove the morpheme boundary, and that both automata end in an accepting state.

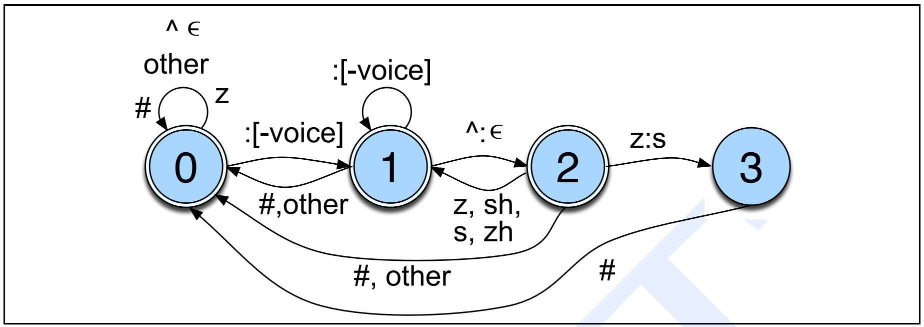


Figure 11.4 The transducer for the z-devoicing rule 11.3. This rule might be summarized *Devoice the morpheme z if it follows a morpheme-final voiceless consonant.*

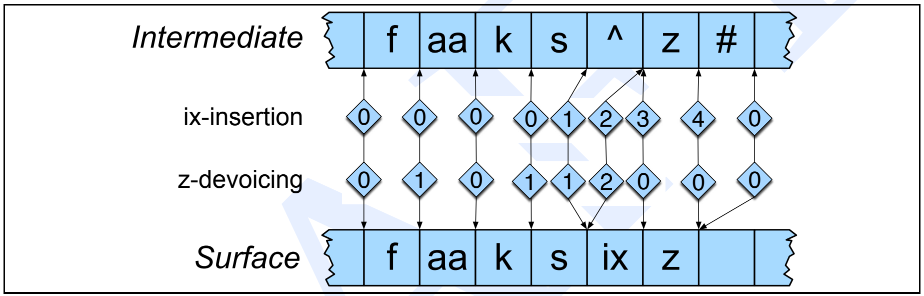


Figure 11.5 The transducer for the ix-insertion rule 11.2 and the z-devoicing rule 11.3 run in parallel.

11.2 Advanced Finite-State Phonology

11.2.1 Harmony

Finite-state models of phonology have also been applied to more sophisticated phonological and morphological phenomena. Let's consider a finite-state model of a well-known complex interaction of three phonological rules in the Yawelmani dialect of Yokuts, a Native American language spoken in California.[4]

vowel harmony

First, Yokuts (like many other languages including for example Turkish and Hungarian) has **vowel harmony**. Vowel harmony is a process in which a vowel changes its form to look like a neighboring vowel. In Yokuts, a suffix vowel changes its form to agree in backness and roundness with the preceding stem vowel. That is, a front vowel like /i/ will appear as a back vowel [u] if the stem vowel is /u/. This **Harmony** rule applies if the suffix and stem vowels are of the same height (e.g., /u/ and /i/ both high, /o/ and /a/ both low): [5]

[4] These rules were first drawn up in the traditional Chomsky and Halle (1968) format by Kisseberth (1969) following the field work of Newman (1944).

[5] Examples from Cole and Kisseberth (1995). Some parts of system such as vowel underspecification have been removed for pedagogical simplification (Archangeli, 1984).

	High Stem				Low Stem			
	Lexical		**Surface**	**Gloss**	**Lexical**		**Surface**	**Gloss**
Harmony	dub+hin	→	dubhun	“tangles”	bok’+al	→	bok’ol	“might eat”
No Harmony	xil+hin	→	xilhin	“leads by the hand”	xat’+al	→	xat’al	“might find”

The second relevant rule, **Lowering**, causes long high vowels to become low; /uː/ becomes [oː] and /iː/ becomes [eː], while the third rule, **Shortening**, shortens long vowels in closed syllables:

Lowering				Shortening		
ʔuːt’+it	→	ʔoːt’ut	“steal, passive aorist”	sːap+hin	→	saphin
miːk’+it	→	meːk’+it	“swallow, passive aorist”	suduːk+hin	→	sudokhun

The three Yokuts rules must be ordered, just as the ix-insertion and z-devoicing rules had to be ordered. Harmony must be ordered before Lowering because the /uː/ in the lexical form /ʔuːt’+it/ causes the /i/ to become [u] before it lowers in the surface form [ʔoːt’ut]. Lowering must be ordered before Shortening because the /uː/ in /suduːk+hin/ lowers to [o]; if it was ordered after shortening it would appear on the surface as [u].

The Yokuts data can be modeled either as a cascade of three rules in series, or in the two-level formalism as three rules in parallel; Fig. 11.6 shows the two architectures (Goldsmith, 1993; Lakoff, 1993; Karttunen, 1998). Just as in the two-level examples presented earlier, the rules work by referring sometimes to the lexical context, sometimes to the surface context; writing the rules is left as Exercise 4 for the reader.

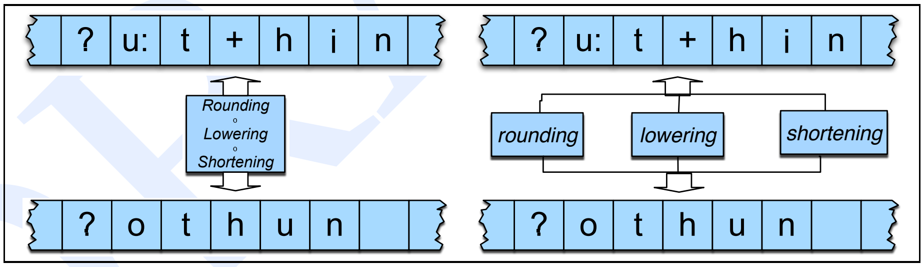


Figure 11.6 Combining the rounding, lowering, and shortening rules for Yawelmani Yokuts.

11.2.2 Templatic Morphology

Finite-state models of phonology and morphology have also been proposed for the templatic (non-concatenative) morphology (discussed on page 52) common in Semitic languages like Arabic, Hebrew, and Syriac.

There are a number of computational finite-state implementations of non-concatenative morphology. Many of them draw on the CV approach of McCarthy (1981), in which a word like /katab/ is represented by three separate morphemes; a **root morpheme** consisting of consonants (*ktb*), a **vocalic morpheme** consisting of vowels (*a*), and a CV pattern morpheme (sometimes called a **binyan** or a **CV skeleton**) (*CVCVC*). McCarthy represented this morphemes on three separate morphological **tiers** (Goldsmith, 1976).

tiers

An influential model by Kay (1987), for example, uses separate tapes for each of McCarthy's tiers. A high-level intuition of Kay's model is shown in Fig. 11.7, which shows his special transducer that reads four tapes instead of two.

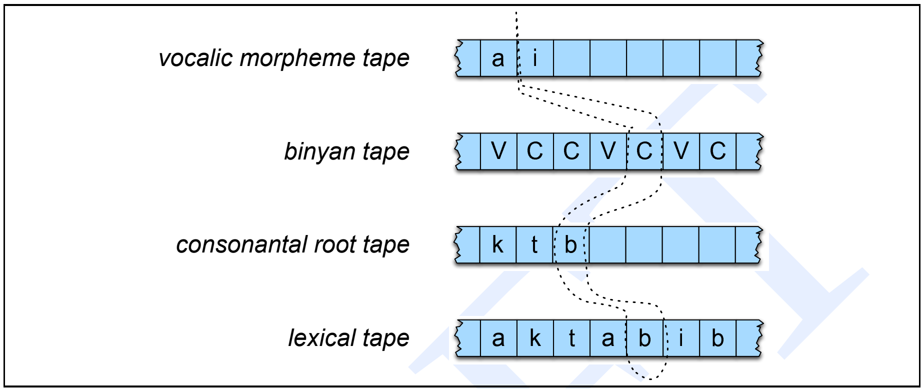


Figure 11.7 A finite-state model of templatic ("non-concatenative") morphology. Adapted from Kay (1987) and Sproat (1993).

The complication with such a multi-tape model is designing a machine which aligns the various strings on the tapes in the correct way; Kay proposed that the binyan tape could act as a sort of guide for alignment. Kay's intuition has led to a number of more fully worked out finite-state models of Semitic morphology; see the end of the chapter for details of these models, as well as alternatives based on new finite-state operations.

11.3 Computational Optimality Theory

Optimality Theory
OT

In a traditional phonological derivation, we are given an underlying lexical form and a surface form. The phonological system then consists of a sequence of rules which map the underlying form to the surface form. **Optimality Theory** (**OT**) (Prince and Smolensky, 1993) offers an alternative way of viewing phonological derivation, based on the metaphor of filtering rather than transforming. An OT model includes two functions (GEN and EVAL) and a set of ranked violable constraints (CON). Given an underlying form, the GEN function produces all imaginable surface forms, even those which couldn't possibly be a legal surface form for the input. The EVAL function then applies each constraint in CON to these surface forms in order of constraint rank. The surface form which best meets the constraints is chosen.

Let's briefly introduce OT, using some Yawlemani data, and then turn to the computational ramifications.[6] In addition to the interesting vowel harmony phenomena discussed above, Yawelmani has phonotactic constraints that rule out sequences of consonants; three consonants in a row (CCC) are not allowed to occur in a surface word. Sometimes, however, a word contains two consecutive morphemes such that the

[6] The following explication of OT via the Yawelmani example draws heavily from Archangeli (1997) and a lecture by Jennifer Cole at the 1999 LSA Linguistic Institute.

first one ends in two consonants and the second one starts with one consonant (or vice versa). What does the language do to solve this problem? It turns out that Yawelmani either deletes one of the consonants or inserts a vowel in between.

If a stem ends in a C, and its suffix starts with CC, the first C of the suffix is deleted ("+" here means a morpheme boundary):

(11.6) **C-deletion:** $C \rightarrow \epsilon \,/\, C + \underline{\quad} \, C$

For example, simplifying somewhat, the CCVC "passive consequent adjunctive" morpheme hneːl drops the initial C if the previous morpheme ends in a consonant. Thus after diyel "guard", we would get the form diyel-neːl-aw, "guard - passive consequent adjunctive - locative".

If a stem ends in CC and the suffix starts with C, the language instead inserts a vowel to break up the first two consonants:

(11.7) **V-insertion:** $\epsilon \rightarrow V \,/\, C \underline{\quad} C + C$

For example in i is inserted into the root ʔilk- "sing" when it is followed by the C-initial suffix -hin, "past", producing ʔilik-hin, "sang", but not when followed by a V-initial suffix like -en, "future" in ʔilken "will sing".

resyllabified

Kisseberth (1970) proposed that these two rules have the same function: avoiding three consonants in a row. Let's restate this in terms of syllable structure. It happens that Yawelmani syllables can only be of the form CVC or CV; complex onsets or complex codas i.e., with multiple consonants, aren't allowed. Since CVCC syllables aren't allowed on the surface, CVCC roots must be **resyllabified** when they appear on the surface. From the point of view of syllabification, then, these insertions and deletions all happen so as to allow Yawelmani words to be properly syllabified. Here's examples of resyllabifications with no change, with an insertion, and with a deletion:

Underlying Morphemes	**Surface Syllabification**	**Gloss**
ʔilk-en	ʔil.ken	"will sing"
ʔilk-hin	ʔi.lik.hin	"sang"
diyel-hnil-aw	di.yel.neː.law	"guard – passive cons. adjunctive – locative"

The intuition of Optimality Theory is to try to directly represent these kind of constraints on syllable structure directly, rather than using idiosyncratic insertion and deletion rules. One such constraint, *COMPLEX, says "No complex onsets or codas". Another class of constraints requires the surface form to be identical to (faithful to) the underlying form. Thus FAITHV says "Don't delete or insert vowels" and FAITHC says "Don't delete or insert consonants". Given an underlying form, the GEN function produces all possible surface forms (i.e., every possible insertion and deletion of segments with every possible syllabification) and they are ranked by the EVAL function using these (violable) constraints. The idea is that while in general insertion and deletion are dispreferred, in some languages and situations they are preferred over violating other constraints, such as those of syllable structure. Fig. 11.8 shows the architecture.

The EVAL function works by applying each constraint in ranked order to each candidate. Starting with the highest-ranked constraints, if one candidate either does

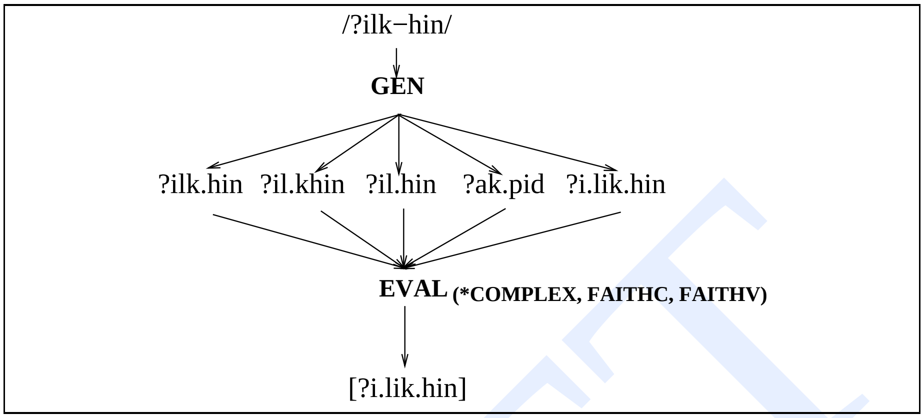


Figure 11.8 The architecture of a derivation in Optimality Theory (after Archangeli (1997)).

not violate no constraints or violates less of them than all the other candidates, that candidate is declared optimal. If two candidates tie (have the same highest ranked violation), then the next-highest ranked violation is considered. This evaluation is usually shown on a **tableau** (plural **tableaux**). The top left-hand cell shows the input, the constraints are listed in order of rank across the top row, and the possible outputs along the left-most column.[7] If a form violates a constraint, the relevant cell contains *****; a ***!** indicates the fatal violation which causes a candidate to be eliminated. Cells for constraints which are irrelevant (since a higher-level constraint is already violated) are shaded.

tableau

***

**!*

	/ʔilk-hin/	*COMPLEX	FAITHC	FAITHV
	ʔilk.hin	*!		
	ʔil.khin	*!		
	ʔil.hin		*!	
☞	ʔi.lik.hin			*
	ʔak.pid		*!	

One appeal of Optimality Theoretic derivations is that the constraints are presumed to be cross-linguistic generalizations. That is all languages are presumed to have some version of faithfulness, some preference for simple syllables, and so on. Languages differ in how they rank the constraints; thus English, presumably, ranks FAITHC higher than *COMPLEX. (How do we know this?)

11.3.1 Finite-State Transducer Models of Optimality Theory

Now that we've sketched the linguistic motivations for Optimality Theory, let's turn to the computational implications. We'll explore two: implementation of OT via finite-state models, and stochastic versions of OT.

[7] Although there are an infinite number of candidates, it is traditional to show only the ones which are 'close'; in the tableau below we have shown the output ʔak.pid just to make it clear that even very different surface forms are to be included.

Can a derivation in Optimality Theory be implemented by finite-state transducers? Frank and Satta (1998), following the foundational work of Ellison (1994), showed that (1) if GEN is a regular relation (for example assuming the input doesn't contain context-free trees of some sort), and (2) if the number of allowed violations of any constraint has some finite bound, then an OT derivation can be computed by finite-state means. This second constraint is relevant because of a property of OT that we haven't mentioned: if two candidates violate exactly the same number of constraints, the winning candidate is the one which has the smallest number of violations of the relevant constraint.

One way to implement OT as a finite-state system was worked out by Karttunen (1998), following the above-mentioned work and that of Hammond (1997). In Karttunen's model, GEN is implemented as a finite-state transducer which is given an underlying form and produces a set of candidate forms. For example for the syllabification example above, GEN would generate all strings that are variants of the input with consonant deletions or vowel insertions, and their syllabifications.

Each constraint is implemented as a filter transducer that lets pass only strings which meet the constraint. For legal strings, the transducer thus acts as the identity mapping. For example, *COMPLEX would be implemented via a transducer that mapped any input string to itself, unless the input string had two consonants in the onset or coda, in which case it would be mapped to null.

The constraints can then be placed in a cascade, in which higher-ranked constraints are simply run first, as suggested in Fig. 11.9.

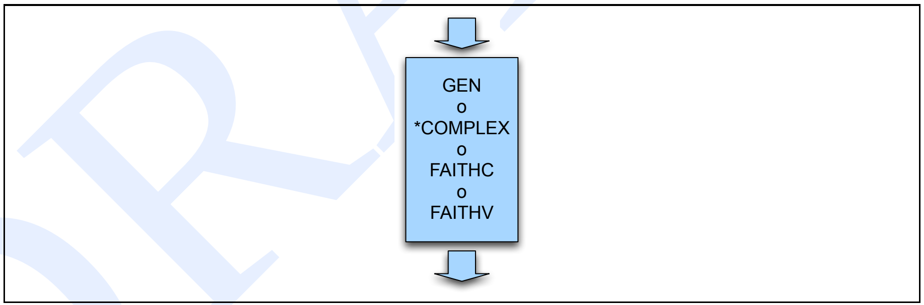


Figure 11.9 Version #1 ("merciless cascade") of Karttunen's finite-state cascade implementation of OT.

There is one crucial flaw with the cascade model in Fig. 11.9. Recall that the constraints-transducers filter out any candidate which violates a constraint. But in many derivations, including the proper derivation of ʔi.lik.hin, even the optimal form still violates a constraint. The cascade in Fig. 11.8 would incorrectly filter it out, leaving no surface form at all! Frank and Satta (1998) and Hammond (1997) both point out that it is essential to only enforce a constraint if it does not reduce the candidate set to zero. Karttunen (1998) formalizes this intuition with the **lenient composition** operator. Lenient composition is a combination of regular composition and an operation called **priority union**. The basic idea is that if any candidates meet the constraint these candidates will be passed through the filter as usual. If no output meets the constraint, lenient composition retains *all* of the candidates. Fig. 11.10 shows the general idea; the

lenient composition

interested reader should see Karttunen (1998) for the details.

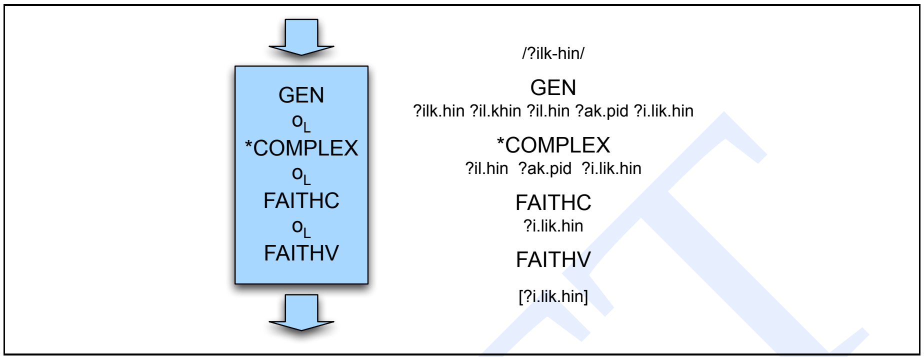


Figure 11.10 Version #2 ("lenient cascade") of Karttunen's finite-state cascade implementation of OT, showing a visualization of the candidate populations that would be passed through each FST constraint.

11.3.2 Stochastic Models of Optimality Theory

Classic OT was not designed to handle variation of the kind we saw in Sec. 7.3, since it assigns a single most-harmonic output for each input. Dealing with variation requires a more dynamic concept of constraint ranking. We mentioned in that section the variationist model in sociolinguistics, in which logistic regression is used to combine phonetic, contextual, and social factors to predict a probability of a particular phonetic variant. Part of this variationist intuition can be absorbed into an Optimality Theory framework through probabilistic augmentations.

Stochastic OT

One such augmentation is **Stochastic OT** (Boersma and Hayes, 2001). In Stochastic OT, instead of the constraints being rank-ordered, each constraint is associated with a value on a continuous scale. The continuous scale offers one thing a ranking cannot: the relative importance or weight of two constraints can be proportional to the distance between them. Fig. 11.11 shows a sketch of such a continuous scale.

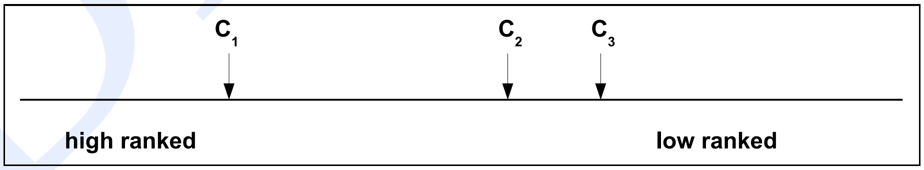


Figure 11.11 The Boersma and Hayes (2001) continuous scale in Stochastic OT.

How can the distance between constraints play a role in evaluation? Stochastic OT makes a further assumption about the values of constraints. Instead of each constraint having a fixed value as in Fig. 11.11, it has a Gaussian distribution of values centered on a fixed value, as in Fig. 11.12. At evaluation time, a value for the constraint is drawn (a **selection point**) with a probability defined by the mean and variance of the Gaussian associated with each constraint.

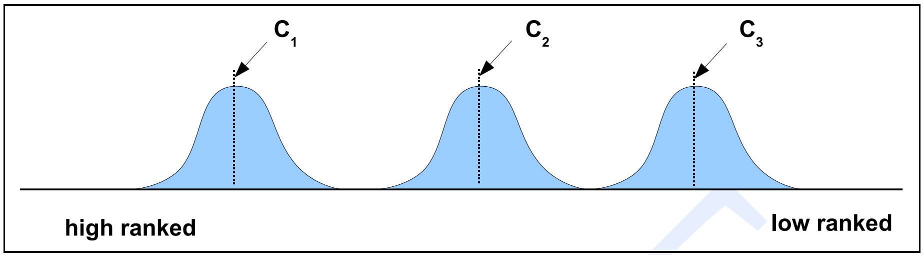


Figure 11.12 Three constraints in Stochastic OT which are strictly ranked; thus non-stochastic OT is a special case of Stochastic OT.

If the distribution for two constraints is far enough apart, as shown in Fig. 11.12 there will be little or no probability of the lower ranked constraint outranking the higher-ranked one. Thus Stochastic OT includes non-stochastic OT as a special case.

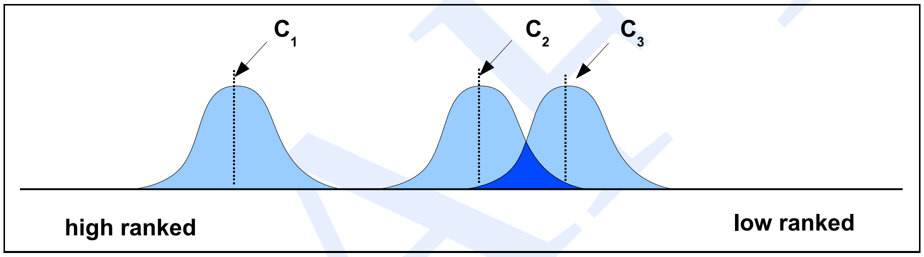


Figure 11.13 Three constraints in Stochastic OT in which C_3 will sometimes outrank C_2.

The interesting cases arise when two constraints in Stochastic OT overlap in their distribution, when there is some probability that a lower-ranked constraint will override a higher-ranked constraint. In Fig. 11.13, for example, constraint C_2 will generally outrank C_3 but occasionally outrank C_2. This allows Stochastic OT to model variation, since for the same underlying form differing selection points can cause different surface variants to be most highly ranked.

In addition to the advantage of modeling variation, Stochastic OT differs from non-stochastic OT in having a stochastic learning theory, which we will return to in Sec. 11.5.3.

We can see stochastic OT itself as a special case of the general linear models of Ch. 6.

11.4 Syllabification

Syllabification

Syllabification, the task of segmenting a sequence of phones into syllables, is important in a variety of speech applications. In speech synthesis, syllables are important in predicting prosodic factors like accent; the realization of a phone is also dependent on its position in the syllable (onset [l] is pronounced differently than coda [l]). In speech recognition syllabification has been used to build recognizers which represent pronunciations in terms of syllables rather than phones. Syllabification can help find errors in pronunciation dictionaries, by finding words that can't be syllabified, and can help annotate corpora with syllable boundaries for corpus linguistics research. Syllabification also plays an important role in theoretical generative phonology.

One reason syllabification is a difficult computational task is that there is no completely agreed-upon definition of syllable boundaries. Different on-line syllabified dictionaries (such as the CMU and the CELEX lexicons) sometimes choose different syllabifications. Indeed, as Ladefoged (1993) points out, sometimes it isn't even clear how many syllables a word has; some words (*meal*, *teal*, *seal*, *hire*, *fire*, *hour*) can be viewed either as having one syllable or two.

Like much work in speech and language processing, syllabifiers can be based on hand-written rules, or on machine learning from hand-labeled training sets. What kinds of knowledge can we use in designing either kind of syllabifier? One possible constraint is the **Maximum Onset** principle, which says that when a series of consonants occur word-medially before a vowel (VCCV), as many as possible (given the other constraints of the language) should be syllabified into the onset of the second syllable rather than the coda of the first syllable. Thus the Maximum Onset principle favors the syllabification V.CCV over the syllabifications VC.CV or VCC.V.

Maximum Onset

Another principle is to use the **sonority** of a sound, which is a measure of how perceptually salient, loud or vowel-like it is. There are various attempts to define a **sonority hierarchy**; in general, all things being equal, vowels are more sonorous than glides (w, y), which are more sonorous than liquids (l, r), followed by nasals (n, m, ng), fricatives (z, s, sh, zh, v, f th, dh), and stops. The sonority constraint on syllable structure says that the nucleus of the syllable must be the most sonorous phone in a sequence (the **sonority peak**), and that sonority decreases monotonically out from the nucleus (toward the coda and toward the onset). Thus in a syllable $C_1C_2VC_3C_4$, the nucleus V will be the most sonorous element, consonant C_2 will be more sonorous than C_1 and consonant C_3 will be more sonorant than consonant C_4.

sonority

sonority hierarchy

Goldwater and Johnson (2005) implement a simple rule-based language-independent classifier based only on maximum onset and sonority sequencing. Given a cluster of consonants between two syllable nuclei, sonority constrains the syllable boundary to be either just before or just after the consonant with the lowest sonority. Combining sonority with maximum onset, their parser predicts a syllable boundary just before the consonant with the lowest sonority. They show that this simple syllabifier correctly syllabifies 86-87% of multisyllabic words in English and German.

While this error rate is not unreasonable, and there is further linguistic and some psychological evidence that these principles play a role in syllable structure, both Maximum Onset and sonority sequencing seem to have exceptions. For example in the English syllable-initial clusters /sp st sk/ in words like *spell*, the less sonorous /p/ occurs between the more sonorous /s/ and the vowel, violating sonority sequencing (Blevins, 1995). Without some way to rule out onset clusters that are disallowed language-specifically like /kn/ in English, the combination of sonority sequencing plus maximum onset incorrectly predicts the syllabification of words like *weakness* to be *wea.kness* rather than *weak.ness*. Furthermore, other constraints seem to be important, including whether a syllable is stressed (stressed syllables tend to have more complex codas), the presence or absence of morphological boundaries, and even the spelling of the word (Titone and Connine, 1997; Treiman et al., 2002).

Achieving higher performance thus requires the use of these sorts of language-specific knowledge. The most commonly used rule-based syllabifier is based on the dissertation of Kahn (1976), available in an implementation by Fisher (1996). The Kahn algorithm makes use of language-specific information in the form of lists of allowable English initial clusters, allowable English final clusters, and 'universally bad' clusters. The algorithm takes strings of phones, together with other information like word boundaries and stress if they are available, and assigns syllable boundaries between the phones. Syllables are built up incrementally based on three rules, as sketched out in Fig. 11.14. Rule 1 forms nuclei at each syllabic segment, Rule 2a attaches onset consonants to the nucleus, and Rule 2b attaches coda consonants.[8] Rule 2a and 2b make use of lists of legal onset consonant sequences (including e.g. [b], [b l], [b r], [b y], [ch], [d], [d r], [d w], [d y], [dh], [f], [f l], [f r], [f y], [g], [g l], [g r], [g w], etc). and legal coda clusters. There are a very large number of coda consonant clusters in English; some of the longer (4-consonant) clusters include:

k s t s
k s th s

l f th s
l k t s
l t s t

m f s t
m p f t
m p s t

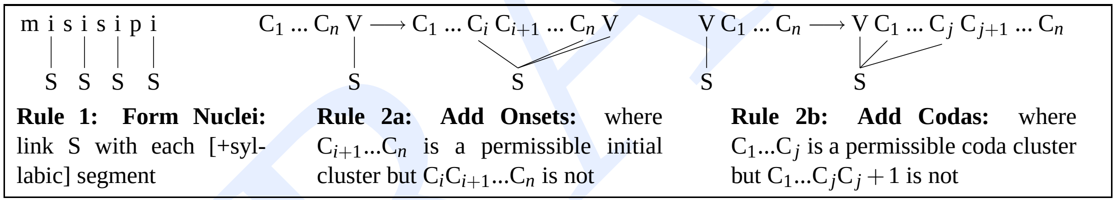
n d th s
n t s t
n t th s

n k s t
n k t s
n k th s

r k t s
r l d z
r m p th

r p t s
r s t s
r t s t

The algorithm also takes a parameter indicating how fast or casual the speech is; the faster or more informal the speech, the more resyllabification happens, based on further rules we haven't shown.


m i s i s i p i
S S S S
$C_1 ... C_n V \longrightarrow C_1 ... C_i C_{i+1} ... C_n V$
S
S
$V C_1 ... C_n \longrightarrow V C_1 ... C_j C_{j+1} ... C_n$
S
S

Rule 1: Form Nuclei: link S with each [+syllabic] segment

Rule 2a: Add Onsets: where $C_{i+1}...C_n$ is a permissible initial cluster but $C_iC_{i+1}...C_n$ is not

Rule 2b: Add Codas: where $C_1...C_j$ is a permissible coda cluster but $C_1...C_jC_j+1$ is not

Figure 11.14 First three syllabification rules of Kahn (1976). Rule 2b may not apply across word boundaries.

Instead of hand-written rules, we can apply a machine learning approach, using a hand-syllabified dictionary as a supervised training set. For example the CELEX syllabified lexicon discussed in Sec. 7.5 is often used this way, selecting some words as a training set, and reserving others as a dev-test and test set. Statistical classifiers can be used to predict syllabifications, including decision trees (van den Bosch, 1997), weighted finite-state transducers (Kiraz and Möbius, 1998), and probabilistic context-free grammars (Seneff et al., 1996; Müller, 2002, 2001; Goldwater and Johnson, 2005).

weighted FST

For example the Kiraz and Möbius (1998) algorithm is a weighted finite-state transducer which inserts a syllable boundary in a sequence of phones (akin to the morpheme-boundaries we saw in Ch. 3). A **weighted FST** (Pereira et al., 1994) is a simple augmentation of the finite transducer in which each arc is associated with a probability as well as a pair of symbols. The probability indicates how likely that path is to be taken; the probability on all the arcs leaving a node must sum to 1.

The syllabification automaton of Kiraz and Möbius (1998) is composed of three separate weighted transducers, one for onsets, one for nuclei, and one for codas, con-

[8] Note that the fact that Rule 2a precedes Rule 2b can be seen as an implementation of Maximum Onset.

catenated together into an FST that inserts a syllable marker after the end of the coda. Kiraz and Möbius (1998) compute path weights from frequencies in the training set; each path (for example the nucleus [iy]) of frequency f is assigned a weight of $1/f$. Another way to convert frequencies to costs is to use log probabilities. Fig. 11.15 shows a sample automaton, simplified from Kiraz and Möbius (1998). We have shown the weights only for some of the nuclei. The arcs for each possible onset, nucleus, and coda, are drawn from a language-dependent list like the one used in the Kahn algorithm above.

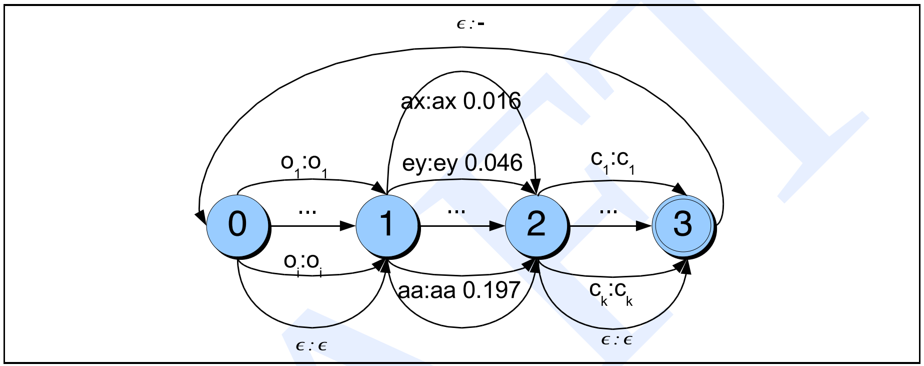


Figure 11.15 Syllabifier automaton, showing onset (o), coda (c), and nucleus arcs. Costs on each arc shown only for some sample nucleus arcs. The syllable boundary marker '-' is inserted after every non-final syllable. eps stands for ϵ. Simplified from Kiraz and Möbius (1998).

The automaton shown in Fig. 11.15 can be used to map from an input sequence like the phonetic representation of *weakness* [w iy k n eh s] into an output sequence that includes the syllabification marker like "-": [w iy k - n eh s]. If there are multiple possible legal syllabifications of a word, the Viterbi algorithm is used to choose the most likely path through the FST, and hence the most probable segmentation. For example, the German word *Fenster*, "window", has three possible syllabifications: [fɛns-tɐ] <74>, [fɛn-stɐ] <75>, and [fɛnst-ɐ] <87> (with costs shown in angle brackets). Their syllabifier correctly chooses the lowest cost syllabification fɛns-tɐ, based on the frequencies of onsets and codas from the training set. Note that since morphological boundaries also are important for syllabification, the Kiraz and Möbius (1998) syllabification transducer can be placed after a morphological parsing transducer, so that syllabification can be influenced by morphological structure.

More recent syllabifiers based on probabilistic context-free grammars (PCFGs) can model more complex hierarchical probabilistic dependencies between syllables (Seneff et al., 1996; Müller, 2002, 2001; Goldwater and Johnson, 2005). Together with other machine learning approaches like van den Bosch (1997), modern statistical syllabification approaches have a word accuracy of around 97–98% correct, and probabilistic model of syllable structure have also been shown to predict human judgments of the acceptability of nonsense words (Coleman and Pierrehumbert, 1997).

There are a number of other directions in syllabification. One is the use of unsupervised machine learning algorithms (Ellison, 1992; Müller et al., 2000; Goldwater and Johnson, 2005) Another is the use of other cues for syllabification such as allophonic details from a narrow phonetic transcription (Church, 1983).

11.5 Learning Phonology & Morphology

Machine learning of phonological structures is an active research area in computational phonology above and beyond the induction of syllable structure discussed in the previous section. Supervised learning work is based on a training set that is explicitly labeled for the phonological (or morphological) structure to be induced. Unsupervised work attempts to induce phonological or morphological structure without labeled training data. Let's look at three representative areas of learning: learning of phonological rules, learning of morphological rules, and learning of OT constraint rankings

11.5.1 Learning Phonological Rules

In this section we briefly summarize some early literature in learning phonological rules, generally couched either in terms of finite-state models of two-level phonology or classic Chomsky-Halle rules.

Johnson (1984) gives one of the first computational algorithms for phonological rule induction. His algorithm works for rules of the form

(11.8) $a \rightarrow b/C$

where C is the feature matrix of the segments around a. Johnson's algorithm sets up a system of constraint equations which C must satisfy, by considering both the positive contexts, i.e., all the contexts C_i in which a b occurs on the surface, as well as all the negative contexts C_j in which an a occurs on the surface. Touretzky et al. (1990) extended Johnsons work in various ways, including dealing with epenthesis and deletion rules.

The algorithm of Gildea and Jurafsky (1996) was designed to induce transducers representing two-level rules of the type we have discussed earlier. Gildea and Jurafsky's supervised algorithm was trained on pairs of underlying and surface forms. For example, they attempted to learn the rule of English flapping, (focusing only on the phonetic context and ignoring social and other factors). The training set thus consisted of underlying/surface pairs, either with an underlying /t/ and surface flap [dx], or an underlying /t/ and surface [t], as follows:

flapping			non-flapping		
butter	/b ah t axr/	→ [b ah dx axr]	*stop*	/s t aa p/	→ [s t aa p]
meter	/m iy t axr/	→ [m iy dx axr]	*cat*	/k ae t/	→ [k ae t]

The algorithm was based on OSTIA (Oncina et al., 1993), a general learning algorithm for the **subsequential transducers** defined on page 59. Gildea and Jurafsky showed that by itself, the OSTIA algorithm was too general to learn phonological transducers, even given a large corpus of underlying-form/surface-form pairs. For example, given 25,000 underlying/surface pairs like the examples above, the algorithm ended up with the huge and incorrect automaton in Fig. 11.16(a). Gildea and Jurafsky then augmented the domain-independent OSTIA system with learning biases which are specific to natural language phonology. For example they added a **Faithfulness** bias that underlying segments tend to be realized similarly on the surface (i.e. that all things being equal, an underlying /p/ was likely to emerge as a surface [p]). They did this by starting OSTIA with the underlying and surface strings aligned using Levenshtein distance. They also added knowledge about phonetic features (vowel versus consonant, reduced versus non-reduced vowel, etc). Together, adding these biases enabled OSTIA to learn the automaton in Fig. 11.16(b), as well as correct automatons for other phonological rules like German consonant devoicing.

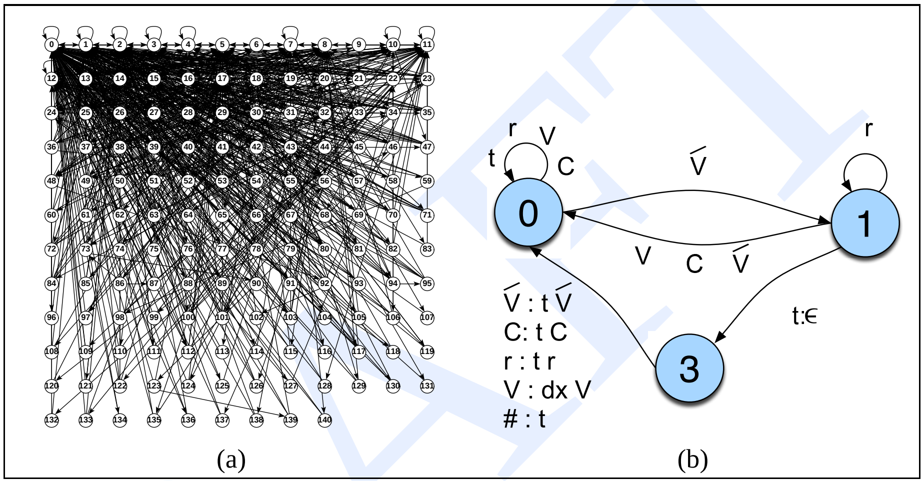


Figure 11.16 Induction of a flapping rule transducer (after Gildea and Jurafsky (1996)). The transducer in (a) is the initial attempt at learning. The transducer in (b) is the correct transducer induced after a faithfulness bias.

This phonological learning experiment illustrates that successful learning requires two components: a model which fits some empirical data and some prior knowledge or biases about the structure of the model.

Recent work on learning has focused either on morphological learning, or on ranking of OT constraints rather than on the induction of rules and constraints, and will be discussed in the next two sections.

11.5.2 Learning Morphology

We discussed in Ch. 3 the use of finite-state transducers for morphological parsing. In general, these morphological parsers are built by hand and have relatively high accuracy, although there has also been some work on supervised machine learning of morphological parsers (van den Bosch, 1997). Recent work, however, has focused on unsupervised ways to automatically bootstrap morphological structure. The unsupervised (or weakly supervised) learning problem has practical applications, since there are many languages for which a hand-built morphological parser, or a morphological segmented training corpus, does not yet exist. In addition, the learnability of linguistic structure is a much-discussed scientific topic in linguistics; unsupervised morphological learning may help us understand what makes language learning possible.

trie

Approaches to unsupervised morphology induction have employed a wide variety of heuristics or cues to a proper morphological parse. Early approaches were all essentially segmentation-based; given a corpus of words they attempted to segment each word into a stem and an affix using various unsupervised heuristics. For example the earliest work hypothesized morpheme boundaries at the point in a word where there is large uncertainty about the following letters (Harris, 1954, 1988; Hafer and Weiss, 1974). For example, Fig. 11.17 shows a **trie**[9] which stores the words *car*, *care*, *cars*, *cares*, *cared*, etc. Note that there are certain nodes in the tree in Fig. 11.17 that have a wide branching factor (after *car* and after *care*). If we think of the task of predicting the next letter giving the path in the trie so far, we can say that these points have a high conditional entropy; there are many possible continuations.[10] While this is a useful heuristic, it is not sufficient; in this example we would need a way to rule out the morpheme *car* as well as *care* being part of the word *careful*; this requires a complex set of thresholds.

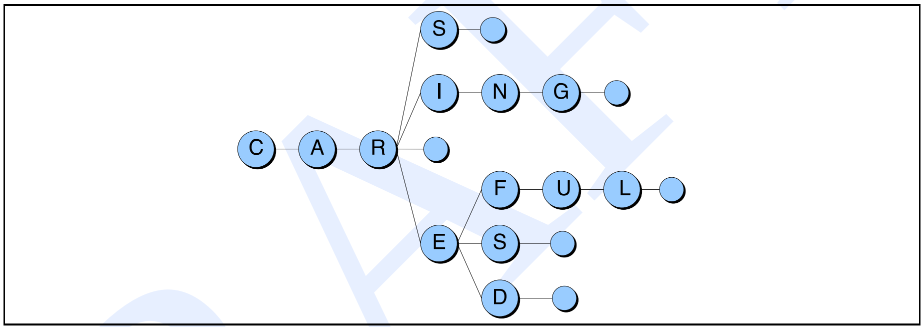


Figure 11.17 Example of a letter trie. A Harris style algorithm would insert morpheme boundaries after *car* and *care*. After Schone and Jurafsky (2000).

minimum description length

MDL

Another class of segmentation-based approaches to morphology induction focuses on globally optimizing a single criterion for the whole grammar, the criterion of **minimum description length**, or **MDL**. The MDL principle is widely used in language learning, and we will see it again in grammar induction in Ch. 14. The idea is that we are trying to learn the optimal probabilistic model of some data. Given any proposed model, we can assign a likelihood to the entire data set. We can also use the proposed model to assign a compressed length to this data (with probabilistic models we can use the intuition that the compressed length of the data is related to the entropy, which we can estimate from the log probability). We can also assign a length to the proposed model itself. The MDL principle says to choose the model for which the sum of the data length and the model length is the smallest. The principle is often viewed from a Bayesian perspective; If we are attempting to learn the best model $\hat{M}$ out of all models M for some data D which has the maximum a posteriori probability $P(M|D)$, we can

[9] A **trie** is a tree structure used for storing strings, in which a string is represented as a path from the root to a leaf. Each non-terminal node in the tree thus stores a prefix of a string; every common prefix is thus represented by a node. The word **trie** comes from *re***trie***val* and is pronounced either [t r iy] or [t r ay].

[10] Interestingly, this idea of placing boundaries at regions of low predictability has been shown to be used by infants for word segmentation (Saffran et al., 1996b).

use Bayes Rule to express the best model $\hat{M}$ as:

$$\hat{M} = argmax_M P(M|D) = argmax_M \frac{P(D|M)P(M)}{P(D)} = argmax_M P(D|M)P(M) \quad (11.9)$$

Thus the best model is the one which maximizes two terms: the likelihood of the data $P(D|M)$ and the prior of the model $P(M)$. The MDL principle can be viewed as saying that the prior term on the model should be related to the length of the model.

MDL approaches to segmentation induction were first proposed by de Marcken (1996) and Brent (1999), as well as Kazakov (1997); let's summarize from a more recent instantiation by Goldsmith (2001). The MDL intuition can be seen from the schematic example in Fig. 11.18 inspired by Goldsmith.

$$\left\{\begin{array}{l}\text{cooked cooks cooking}\\ \text{played plays playing}\\ \text{boiled boils boiling}\end{array}\right\} \qquad\qquad \left\{\begin{array}{l}\text{cook}\\ \text{play}\\ \text{boil}\end{array}\right\} \left\{\begin{array}{l}\text{ed}\\ \text{s}\\ \text{ing}\end{array}\right\}$$

(a) Word list with no structure
Total letter count: 54

(b) Word list with morphological structure
Total letter count: 18 letters

Figure 11.18 Naive version of MDL, showing the reduction in the description length of a lexicon with morphological structure; adapted from Goldsmith (2001).

signatures

As we see in Fig. 11.18, using morphological structure makes it possible to represent a lexicon with far fewer letters. Of course this example doesn't represent the true complexity of morphological representations, since in reality not every word is combinable with every affix. One way to represent slightly more complexity is to use **signatures**. A signature is a list of suffixes that can appear with a particular stem. Here are some sample signatures from Goldsmith (2001):

Signature	Example
NULL.ed.ing.s	remain remained remaining remains
NULL.s	cow cows
e.ed.es.ing	notice noticed notices noticing

The Goldsmith (2001) version of MDL considers all possible segmentations of every word into a stem and a suffix. It then chooses the set of segmentations for the whole corpus that jointly minimize the compressed length of the corpus and the length of the model. The length of the model is the sum of the lengths of the affixes, the stems, and the signatures. The length of the corpus is computed by using the model to assign a probability to the corpus and using this probably to compute the cross-entropy of the corpus given the model.

While approaches based solely on stem and affix statistics like MDL have been quite successful in morphological learning, they do have a number of limitations. For example Schone and Jurafsky (2000, 2001) noted in an error analysis that MDL sometimes segments valid affixes inappropriately (such as segmenting the word *ally* to *all+y*), or fails to segment valid but non-productive affixes (missing the relationship between *dirt* and *dirty*). They argued that such problems stemmed from a lack of semantic or syntactic knowledge, and showed how to use relatively simple semantic features to address them. The Schone and Jurafsky (2000) algorithm uses a trie to come up with “pairs of potential morphological variants”, (PPMVs) words which differ only in potential affixes. For each pair, they compute the semantic similarity between the words, using the Latent Semantic Analysis (LSA) algorithm of Ch. 23. LSA is an unsupervised model of word similarity which is induced directly from the distributions of word in context. Schone and Jurafsky (2000) showed that using the semantic similarity alone was at least as good a predictor of morphological structure as MDL. The table below shows the LSA-based similarity between PPMVs; in this example the similarity is high only for words that are morphologically related.

PPMV	Score	PPMV	Score	PPMV	Score	PPMV	Score
ally/allies	6.5	dirty/dirt	2.4	car/cares	-0.14	car/cared	-.096
car/cars	5.6	rating/rate	0.97	car/caring	-0.71	ally/all	-1.3

Schone and Jurafsky (2001) extended the algorithm to learn prefixes and circumfixes, and incorporated other useful features, including syntactic and other effects of neighboring word context (Jacquemin, 1997), and the Levenshtein distance between the PPMVs (Gaussier, 1999).

The algorithms we have mentioned so far have focused on the problem of learning regular morphology. Yarowsky and Wicentowski (2000) focused on the more complex problem of learning irregular morphology. Their idea was to probabilistically align an inflected form (such as English *took* or Spanish *juegan*) with each potential stem (such as English *take* or Spanish *jugar*). The result of their alignment-based algorithm was a inflection-root mapping, with both an optional stem change and a suffix, as shown in the following table:

English				Spanish			
root	**inflection**	**stem change**	**suffix**	**root**	**inflection**	**stem change**	**suffix**
take	took	ake→ook	+ϵ	jugar	juega	gar→eg	+a
take	taking	e→ ϵ	+ing	jugar	jugamos	ar→ ϵ	+amos
skip	skipped	ϵ →p	+ed	tener	tienen	ener→ien	+en

The Yarowsky and Wicentowski (2000) algorithm requires somewhat more information than the algorithms for inducing regular morphology. In particular it assumes knowledge of the regular inflectional affixes of the language and a list of open class stems; both are things that might be induced by the MDL or other algorithms mentioned above. Given an inflected form, the Yarowsky and Wicentowski (2000) algorithm uses various knowledge sources to weight the potential stem, including the relative frequency of the inflected form and potential stem, the similarity in lexical context, and the Levenshtein distance between them.

11.5.3 Learning in Optimality Theory

Let’s conclude with a brief sketch of work which addresses the learning problem in Optimality Theory. Most work on OT learning has assumed that the constraints are already given, and the task is to learn the ranking. Two algorithms for learning rankings have been worked out in some detail; the **constraint demotion** algorithm of Tesar and Smolensky (2000) and the **Gradual Learning Algorithm** of Boersma and Hayes (2001).

Constraint Demotion

The **Constraint Demotion** algorithm makes two assumptions: that we know all the possible OT constraints of the language, and that each surface form is annotated with its complete parse and underlying form. The intuition of the algorithm is that each of these surface observations gives us implicit evidence about the constraint ranking.

Given the underlying form, we can use the GEN algorithm to implicitly form the set of competitors. Now we can construct a set of pairs consisting of the correct observed grammatical form and each competitor. The learner must find a constraint ranking that prefers the observed learning *winner* over each (non-observed) competitor *loser*. Because the set of constraints is given, we can use the standard OT parsing architecture to determine for each winner or loser exactly which constraints they violate.

For example, consider the learning algorithm that has observed Candidate 1, but whose current constraint ranking prefers Candidate 2, as follows (this example and the following tables are modified from Boersma and Hayes (2001)):

	/underlying form/	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
	Candidate 1 (learning observation)	*!	**	*		*			*
☞	Candidate 2 (learner's output)		*	*	*		*		*

Given a set of such *winner/loser* pairs, the Constraint Demotion algorithm needs to demote each constraint that is violated by the winner Candidate 2, until the observed form (Candidate 1) is preferred. The algorithm first cancels any marks due to violations that are identical between the two candidates:

	/underlying form/	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
	Candidate 1 (learning observation)	*!	*~~*~~	~~*~~		*			~~*~~
☞	Candidate 2 (learner's output)		~~*~~	~~*~~	*		*		~~*~~

These constraints are pushed down in the hierarchy until they are dominated by the constraints violated by the loser. The algorithm divides constraints into **strata**, and tries to find a lower strata to move the constraints into. Here's shows a simplification of this intuition, as C_1 and C_2 get moved below C_8.

	/underlying form/	C_3	C_4	C_5	C_6	C_7	C_8	C_1	C_2
☞	Candidate 1 (learning observation)				*			*	*
	Candidate 2 (learner's output)		*!		*				

Gradual Learning Algorithm

The **Gradual Learning Algorithm** (GLA) of (Boersma and Hayes, 2001) is a generalization of Constraint Demotion that learns constraint rankings in Stochastic Optimality Theory. Since OT is a special case of Stochastic OT, the algorithm also implicitly learns OT rankings. It generalizes Constraint Demotion by being able to learn from cases of free variation. Recall from Sec. 11.3 that in Stochastic OT each constraint is associated with a **ranking value** on a continuous scale. The ranking value is defined as the mean of the Gaussian distribution that constitutes the constraint. The goal of the GLA is to assign a ranking value for each constraint. The algorithm is a simple extension to the Constraint Demotion algorithm, and follows exactly the same steps until the final step. Inside of demoting constraints to a lower strata, the ranking value of each constraint violated by the learning observation (Candidate 1) is decreased slightly, and the ranking value of each constraint violated by the learner's output (Candidate 2) is increased slightly, as shown below:

	/underlying form/	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
	Candidate 1 (learning observation)	∗!→	∗ →			∗ →			
☞	Candidate 2 (learner's output)				← ∗		← ∗		

11.6 Summary

This chapter has introduced many of the important concepts of phonetics and computational phonology.

- **Transducers** can be used to model phonological rules just as they were used in Ch. 3 to model spelling rules. **Two-level morphology** is a theory of morphology/phonology which models phonological rules as finite-state **well-formedness constraints** on the mapping between lexical and surface form.
- **Optimality theory** is a theory of phonological well-formedness; there are computational implementations, and relationships to transducers.
- Computational models exist for **syllabification**, inserting syllable boundaries in phone strings.
- There are numerous algorithms for learning phonological and morphological rules, both supervised and unsupervised.

Bibliographical and Historical Notes

Computational phonology is a fairly recent field. The idea that phonological rules could be modeled as regular relations dates to Johnson (1972), who showed that any phonological system that didn't allow rules to apply to their own output (i.e., systems that did not have recursive rules) could be modeled with regular relations (or finite-state transducers). Virtually all phonological rules that had been formulated at the time had this property (except some rules with integral-valued features, like early stress and tone rules). Johnson's insight unfortunately did not attract the attention of the community, and was independently discovered by Ronald Kaplan and Martin Kay; see Ch. 3 for the rest of the history of two-level morphology. Karttunen (1993) gives a tutorial introduction to two-level morphology that includes more of the advanced details than we were able to present here, and the definitive text on finite-state morphology is Beesley and Karttunen (2003). Other FSA models of phonology include Bird and Ellison (1994).

Earlier computational finite-state models that deal with templatic morphology in languages like Arabic include Kataja and Koskenniemi (1988), Kornai (1991), Bird and Ellison (1994), and Beesley (1996). Extensions of the Kay (1987) model include Kiraz (1997, 2000, 2001). Recent models based on extensions to the finite-state calculus include Beesley and Karttunen (2000).

Optimality theory was developed by Prince and Smolensky and circulated as a technical report (Prince and Smolensky, 1993) until its publication more than a decade later (Prince and Smolensky, 2004). A selection from the extensive finite-state literature in OT includes Eisner (1997, 2000b, 2002b), Gerdemann and van Noord (2000), and Riggle (2005).

Recent work on phonological learning has focused on some new areas. One is learning **phonotactic constraints** on the allowable word-internal sequences in the language, including probabilistic (Coleman and Pierrehumbert, 1997; Frisch et al., 2000; Bailey and Hahn, 2001; Hayes and Wilson, 2007; Albright, 2007) as well as non-probabilistic phonotactic constraints (Hayes, 2004; Prince and Tesar, 2004; Tesar and Prince, 2007). A related task is the learning of **underlying forms** and phonological alternations given the observed surface forms and the set of constraints. Many of the unsupervised algorithms for learning underlying forms are based on a constraint satisfaction approach, in which sets of possible underlying forms are proposed by examining alternating surface forms, and then iteratively ruling out possible underlying forms (Tesar and Prince, 2007; Alderete et al., 2005; Tesar, 2006a, 2006b). The recent unsupervised Maximum Likelihood Learning of Lexicons and Grammars (MLG) model of Jarosz (2006, 2008) learns underlying forms and constraint rankings given surface forms in a probabilistic version of OT using the Expectation-Maximization (EM) algorithm described in Ch. 6.

Harmonic Grammar

harmony

Indeed, in addition to this probabilistic model of Jarosz (2008), as well as the Stochastic OT described earlier in the chapter, much recent work in computational phonology has focused on models with weighted constraints, including **Harmonic Grammar** and **Maximum Entropy Models**. For example **Harmonic Grammar** is an extension to Optimality Theory (or more properly is the theory that Optimality Theory originally grew out of) in which optimality for a form is defined as maximal **harmony**. Harmony is defined by the sum of weighted constraints (Smolensky and Legendre, 2006). In using sums of weight rather than OT-style rankings, Harmony Theory resembles the log-linear models of Ch. 6. Recent computational work include the application to OT of Maximum Entropy Models (Goldwater and Johnson, 2003) and the Harmonic Grammar related models of Pater et al. (2007) and Pater (2007).

Word segmentation is one of the earliest problems in computational linguistics, and models date back to Harris (1954). Among the many modern models are Bayesian ones like Brent (1999) and Goldwater et al. (2006). The word segmentation problem is important also in computational developmental psycholinguistics; for representative recent work see Christiansen et al. (1998), Kuhl et al. (2003), Thiessen and Saffran (2004) and Thiessen et al. (2005). Recent work on morphology induction includes Baroni et al. (2002), Clark (2002), and Albright and Hayes (2003).

Readers with further interest in phonology should consult phonology textbooks like

Odden (2005) and Kager (2000).

Exercises

11.1 Build an automaton for rule (11.3).

Canadian raising

11.2 One difference between one dialect of Canadian English and most dialects of American English is called **Canadian raising**. Bromberger and Halle (1989) note that some Canadian dialects of English raise /aɪ/ to [ʌɪ] and /aʊ/ to [ʌʊ] in stressed position before a voiceless consonant. A simplified version of the rule dealing only with /aɪ/ can be stated as:

(11.10)

$$/\text{aɪ}/ \rightarrow [\text{ʌɪ}] \,/\, \underline{\quad\;} \begin{bmatrix} C \\ -\mathit{voice} \end{bmatrix}$$

This rule has an interesting interaction with the flapping rule. In some Canadian dialects the word *rider* and *writer* are pronounced differently: *rider* is pronounced [ɾaɪɾɚ] while *writer* is pronounced [ɾʌɪɾɚ]. Write a two-level rule and an automaton for both the raising rule and the flapping rule which correctly models this distinction. You may make simplifying assumptions as needed.

11.3 Write the lexical entry for the pronunciation of the English past tense (preterite) suffix *-d*, and the two level-rules that express the difference in its pronunciation depending on the previous context. Don't worry about the spelling rules. (Hint: make sure you correctly handle the pronunciation of the past tenses of the words *add*, *pat*, *bake*, and *bag*.)

11.4 Write two-level rules for the Yawelmani Yokuts phenomena of Harmony, Shortening, and Lowering introduced on page 369. Make sure your rules are capable of running in parallel.