Lecture 9

The Power of SPE and OT

We have come a long way since we took our first look at phonology. Starting from a very naive perspective that deliberately ignored most linguistic assumptions and only considered only surface-true generalizations, we kept tweaking our computational model and arrived at the conclusion that most phonological patterns fall within the very weak classes of strictly local and strictly piecewise languages. Suprasegmental patterns required factoring out via culminativity, which can be expressed with 1-threshold testable grammars given a suitable alphabet. The dependence on specific alphabets was worrying though, so that we took a closer look at how much power alphabet refinement can grant us. The answer was rather shocking, as even the strictly 2-local languages see an immense increase in expressivity when they are supplemented with a hidden alphabet. We saw that a hidden alphabet pushes our model all the way up to the regular languages, with no strong empirical evidence that this enormous power is ever needed in phonology.

These findings are particularly troubling because alphabet refinement amounts to positing a more abstract underlying structure, which is an ubiquitous strategy in phonology. This suggests that the standard theories of phonology might actually be too powerful. This claim is difficult to evaluate with our current tools, however, because linguistic theories of phonology do not simply distinguish between well-formed and ill-formed strings, but rather specify a mapping from underlying forms to surface forms. So rather than jumping to conclusions, we should try to give faithful formalizations of these theories. If generative capacity still turns out to be problematic, some reflecting is in order as to why linguists may feel the need for this extra power.

1 Formalizing Rewrite Rules

1.1 Rewrite Rules in SPE

For many decades the dominant theory of phonology was SPE (Chomsky and Halle 1968). According to SPE, phonology is a set of rewrite rules that map underlying representations to surface forms. A rewrite rule takes the form

$$\alpha \to \beta \mid \gamma\_\delta,$$

which means that a substring α that occurs between γ and δ is rewritten as β. For instance, $aa \to \varepsilon \mid b\_bb$ rewrites all instances of *baabb* as *bbb*. The part before the vertical bar, i.e. $\alpha \to \beta$, is the actual rewriting step, whereas γ_δ is the context

specification. A rule can apply from left-to-right, right-to-left, or in parallel to all licit target sites.

Example 9.1 Directionality of Rule Application

Consider the rule $a \rightarrow b \mid ab\_ba$, which turns a into b whenever it occurs between ab and ba (this is an abstracted version of the process of full, local assimilation). Depending on how this rule is applied, the string *abababababa* is mapped to different output strings.

directionality	output string
left-to-right	*abbbabbbaba*
right-to-left	*ababbbabbba*
parallel	*abbbbbbbbba*

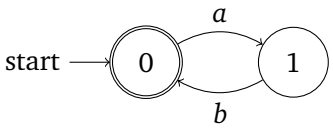
Multiple rules can be conflated into a single rule using round and curly brackets to indicate optionality and non-determinism, respectively. For example, $a \rightarrow \{b, c\} \mid c(d)c\_$ rewrites a as b or c iff it is preceded by cc or cdc. Finally, the collection of conflated rules is linearly ordered, determining in the sequence of rule applications.

As you can see, SPE features a dazzling array of technical tools, and that is actually just the tip of the iceberg. Segments are actually formalized as matrices of binary features, and sets of segments are represented via underspecified matrices. Many of the rewrite rules use this format to indicate the rewriting of segments as the change of feature values, and there is a lot of ancillary notation like α-values to keep rules as short as possible. Finally, segments and strings in the context specification can be subscripted with $^+$ such that w^+ stands for every string in the language w^+. It should be fairly obvious that we do not have the right tools to formalize all of these concepts. Let us start with the most basic notion, then: string rewriting.

1.2 Finite State Transducers

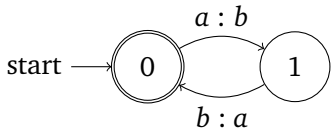
In the previous chapter we encountered finite state automata (FSAs) as another formalism for talking about strictly 2-local grammars with hidden alphabets. Just like these grammars, an automaton only decides the well-formedness of strings, it is not a mechanism for rewriting a string as another one. However, automata can easily be turned into such a mechanism.

Suppose that we want to map every string of the language $(ab)^*$ to its counterpart where the order of as and bs has been switched. The automaton for $(ab)^*$ is given below.



We want this automaton to also establish a connection between each $(ab)^n$ and $(ba)^n$. An easy way of doing this is to extend the automaton so that it operates on two strings at the same time. Each arc now gets a label $\sigma : \omega$, with the first component indicating

the symbol in the first string, and the second component the symbol in the second string. For the current example, we extend a to $a : b$ and b to $b : a$.



Such an extended automaton is called a *finite state transducer* (FST). We can view a transducer as

- verifying whether two strings are related (do both strings take the same path through the transducer?),
- building two strings in parallel (follow some path from an initial state to a final one and keep track of the first/second label of each arc), or
- rewriting the first string as the second one (follow the arcs through transducer that yield the first string and build the second string that is described by these arcs).

Obviously it is this third perspective that is of particular interest to us.

Example 9.2 Three Views of Finite State Transducers

Let us take a quick glance at the behavior of the FST above for the strings *abab* and *baba*. When viewed as a recognizer for a relation between strings, the transducer has to be able to find an identical state assignment for both strings. This is indeed the case.

0	1	0	1	0
	a	b	a	b
	b	a	b	a

But *abab* is not related to *babb*.

0	1	0	1	!
	a	b	a	b
	b	a	b	b

Similarly, the transducer can build *abab* and *baba* in parallel via the sequence of transitions $\langle 0, a : b, 1\rangle$, $\langle 1, b : a, 0\rangle$, $\langle 0, a : b, 1\rangle$, $\langle 1, b : a, 0\rangle$. And we can combine both views by first determining that *abab* is recognized via the sequence $\langle 0, a : b, 1\rangle$, $\langle 1, b : a, 0\rangle$, $\langle 0, a : b, 1\rangle$, $\langle 1, b : a, 0\rangle$, and that the string jointly described by the second components of the transitions is *baba*.

Formally, an FST is an FSA that has been extended with an output alphabet.

Definition 9.1 (Finite State Transducer). A *finite state transducer* (FST) is a 6-tuple $A := \langle \Sigma, \Omega, Q, I, F, \Delta\rangle$, where

- Σ is the input alphabet,

- Ω is the output alphabet,
- Q is a finite set of states,
- $I \subseteq Q$ is the set of *initial* states,
- $F \subseteq Q$ is the set of *final* states,
- $\Delta \subseteq Q \times \Sigma \times \Omega \times Q$ is a finite set of transition rules.

The FST is *deterministic* iff I is a singleton set and Δ contains no two $\langle p, a, b, q\rangle$ and $\langle p, a, c, r\rangle$ with $q \neq r$ or $b \neq c$. Otherwise it is non-deterministic.

Note that in contrast to non-deterministic automata, non-deterministic transducers cannot always be made deterministic. Characterizing the subclass of non-deterministic transducers that can be determinized is too advanced a topic for us.

An FST does not generate a language, i.e. a set of strings, but a binary relation, i.e. pairs of strings. Such a binary relation between strings is also called a *transduction*, and a transduction that can be computed by an FST is called a *finite state transduction* or a *rational relation*. Note that if we only consider the first component of each pair in the transduction we get the input language, whereas restricting our attention to the second component yields the output language.

So now we have a mechanism for rewriting strings. We do not know yet if it can handle the full range of SPE rewrite rules, but it does provide us with a formal basis that we can explore with our computational techniques.

1.3 Properties of Finite State Transducers

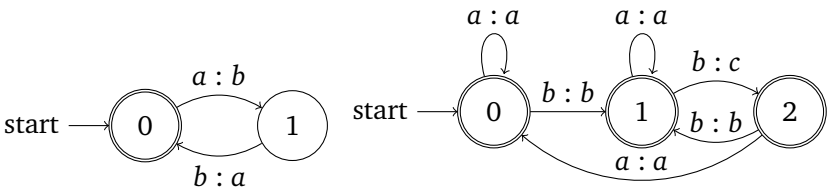
If we are to use FSTs as models of SPE rewrite rules, we will have to be able to combine FSTs, just like SPE combines multiple rewrite rules into a grammar. For SPE, this means in particular being able to use the output of one rewrite rule as the input of the next rewrite rule. So if rule R rewrites the string u as v, and then R' rewrites v as w, then a grammar consisting of those two rules rewrites u as v (assuming R applies before R').

We can do do something very similar with FSTs by constructing their *composition*. Given two FSTs that compute the relations R and R', respectively, their composition computes the relation $R \circ R' := \left\{\langle u, w\rangle \mid \langle u, v\rangle \in R \text{ and } \langle v, w\rangle \in R'\right\}$. The construction is very similar to the intersection of FSAs. Both automata are run in parallel, but whenever the first automaton takes an arc that is labeled $a : b$, the second automaton must take an arc whose first component is b.

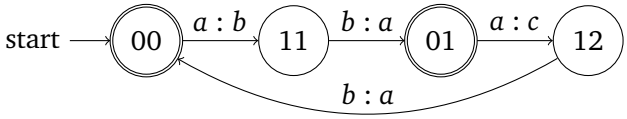
Composition Let $A := \langle Q_A, \Sigma, \Gamma, I_A, F_A, \Delta_A\rangle$ and $B := \langle Q_B, \Gamma, \Omega, I_B, F_B, \Delta_B\rangle$ be two FSTs. Their composition is $A \circ B := \langle Q_A \times Q_B, \Sigma, \Omega, I_A \times I_B, F_A \times F_B, \Delta\rangle$, where Δ is the smallest set containing all $\left\langle (q_a, q_b), \sigma, \omega, (q'_a, q'_b)\right\rangle$ such that $\left\langle q_a, \sigma, \gamma, q'_a\right\rangle \in \Delta_A$ and $\left\langle q_b, \gamma, \omega, q'_b\right\rangle \in \Delta_B$.

Example 9.3 Transducer Composition

Suppose that we want to combine our first example transducer with another transducer that replaces every second b by a c. So if both transducer are run in sequence, the string *abab* is no longer mapped to *baba* but rather *baca*. Each transducer is shown below.

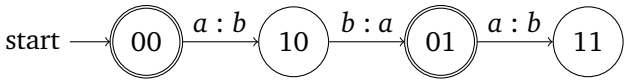


Following the procedure above, we obtain a transducer with 4 useful states.



It is easy to see that this transducer rewrites *abab* as *baca*.

Notice that composition is not commutative. Running the right transducer before the left one maps *abab* to nothing at all because *abac* is not in the domain of the first transducer. You can verify this by looking at their composition; in order to highlight the differences with the previous transducer, the states are still labeled such that their first component represents the state of the left transducer and the second one that of the right transducer.



Closure under composition entails another important property: the image of a regular language L under an FST T is always regular. This is implied by a few simple observations.

Why do we need FST composition for this result? That is to say, why isn't it enough to observe that every FST can be turned into an FSA by dropping the first component of every arc label?

1. If we drop the first component from each arc of the FST T, we get an FSA that defines a regular output language.

2. Every language L can be lifted to a transduction by looking at its identity function $\mathrm{id}(L) := \{\langle w, w\rangle \mid w \in L\}$. If L is regular, this transduction is obtained by taking an FSA for L (which must exist thanks to L being regular) and expanding each arc label a to $a : a$.

3. Since the class of FSTs is closed under composition, $\mathrm{id}(L) \circ T$ is an FST. Notice that the image of Σ^* under $\mathrm{id}(L) \circ T$ is exactly the image of L under T.

4. If Σ^* is the input language for an FST, then the output language consists of all strings that can be derived from some path through the FST. In other words, the output language is exactly the language of the FSA that is obtained by dropping the first component of each arc label.

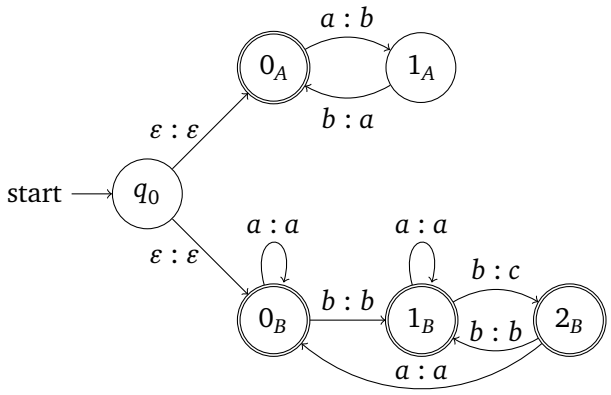
5. It follows that $T(L) = (\mathrm{id}(L) \circ T)(\Sigma^*)$ is a regular language.

Besides composition, it is also of interest to look at the Boolean closure properties.

Union We can just use the automaton construction that adds a single initial state from which ε-transitions take us to the initial states of the transducers. That is to say, $A \cup B := \langle Q_A \cup Q_B \cup \{q_0\}, \Sigma \cup \Gamma, \Gamma \cup \Omega, q_0, F_A \cup F_B, \Delta \rangle$, where $\Delta := \Delta_A \cup \Delta_B \cup \{\langle q_0, \varepsilon, \varepsilon, i \rangle \mid i \in I_A \cup I_B\}$. Keep in mind that the states of Q_A and Q_B must be renamed if the two sets are not disjoint.

Example 9.4 Transducer Union

The union of the two FSTs from the previous example is given below.



Intersection Given two transducers, we can construct their intersection using the exact algorithm for intersection of automata. However, it is not guaranteed that the transduction computed by this new FST is the intersection of the transductions computed by the original two FSTs. This is witnessed by the following counterexample: let R and R' be the regular relations $\{\langle a^n, b^n c^* \rangle \mid n \geq 1\}$ and $\{\langle a^n, b^* c^n \rangle \mid n \geq 1\}$, respectively. Their intersection is $R \cap R' := \{\langle a^n, b^n c^n \rangle \mid n \geq 1\}$. It is well-known that $b^n c^n$ is not regular (a fact we cannot prove yet), but since the output language of an FST is always regular, $R \cap R'$ cannot be a finite state transduction.

(Relative) Complement Non-closure under intersection and closure under union jointly imply non-closure under (relative) complement via De Morgan's law $A \cap B = \overline{\overline{A} \cup \overline{B}}$.

Theorem 9.2. The class of finite state transductions is closed under composition and union, but not intersection or (relative) complement. The class of regular language is closed under finite state transductions. ⌟

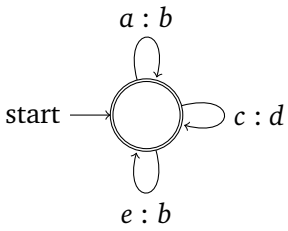
Closure under intersection does hold for a proper subclass of finite state transductions, though, namely those that are computed by *ε-free* FSTs: no arc label has ε as its first component ("do not insert new nodes") or as its second component ("do not delete any nodes"). Such transductions only relate strings of equal length, which is why they're

called *equal length relations*. An equal length relation can be viewed not just as binary relation, i.e. a set over pairs of strings, but also as sets of strings of pairs of symbols. That is to say, the pair $\langle abba, baab\rangle$ can be viewed as the string $\langle a,b\rangle\langle b,a\rangle\langle b,a\rangle\langle a,b\rangle$ instead. It is fairly easy to prove that equal length relations are regular languages and thus inherit all their closure properties, including closure under intersection and (relative) complement.

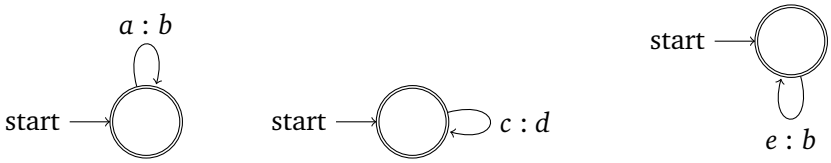
2 The Power of SPE

2.1 SPE Generates All Regular Languages

We already know that every regular language is a projection of some strictly 2-local language. A projection simply replaces every element in the input alphabet Σ by some element of Ω, irrespective of its surrounding. As Σ and Ω are finite, we can view a projection as a single-state FST with a loop labeled $a : b$ iff the projection maps a to b.



But this is just the intersection of three ε-free transducers.



Each one of these transducers corresponds to a rewrite rule $a \to b$, so an SPE grammar that consists only of the three rewrite rules for the respective FSTs above computes the original projection (careful: we have to ensure Σ and Ω are disjoint, otherwise a rewrite rule may accidentally rewrite some of the output symbols of one of the previous rewrite rules).

Example 9.5 Rewrite Rules for a Projection

Suppose we have a projection π between $\Sigma := \{a,b,c\}$ and $\Omega := \{c,d\}$ that is given by the following table:

input	output
a	c
b	c
c	d

This projection maps the string *abbacc* to *ccccdd*.

When converting this projection into an SPE grammar, we first have to make the two alphabets disjoint. This is accomplished by a rewrite rule that rewrite every $\sigma \in \Sigma$ by σ_i.

$$\sigma \rightarrow \sigma_i$$

Now all we have to do is write a rewrite rule for every row in the table.

$$a_i \rightarrow c$$
$$b_i \rightarrow c$$
$$c_i \rightarrow d$$

The table below shows how *abbacc* is rewritten as *ccccdd* by running one rule after another.

rule	**output**
input	*abbacc*
$\sigma \rightarrow \sigma_i$	$a_i b_i b_i a_i c_i c_i$
$a_i \rightarrow c$	$c b_i b_i c c_i c_i$
$b_i \rightarrow c$	$cccc_i c_i$
$c_i \rightarrow d$	*ccccdd*

If we hadn't rendered the two alphabets disjoint, the output string would have looked very different.

rule	**output**
input	*abbacc*
$a \rightarrow c$	*cbbccc*
$b \rightarrow c$	*cccccc*
$c \rightarrow d$	*dddddd*

This procedure works for every projection over arbitrary alphabets Σ and Ω, which means that SPE, coupled with a strictly 2-local language of underlying forms, generates all regular languages.

Lemma 9.3. SPE can compute every projection between two arbitrary alphabets. ⌟

Some phonologists might object, though, that using a strictly 2-local language for the set of underlying representations is rather generous and that this set is not as restricted. The *richness of the base* assumption, for instance, contends that every element of Σ^* is a well-formed underlying representation. But this objection is moot since SPE can generate all regular languages even with a rich base.

Richness of the base is an invention of OT and was never entertained during the reign of SPE. However, for practical applications that use rewrite rules in conjunction with a lexicon it is actually more efficient to turn the lexicon into a transduction as described here so that it can be composed with the rewrite rules.

Recall that every language can be turned into a transduction by looking at its identity function, so the strictly 2-local language L itself is just a finite state transduction $\mathrm{id}(L)$. Therefore we can assume that the set of underlying forms includes every element of Σ^*, which is then restricted to L via the transduction $\mathrm{id}(L)$. As long as $\mathrm{id}(L)$ can

be translated into a rewrite rule, SPE can restrict the set of underlying forms to this language and then compute its projection, yielding a regular output language.

Lemma 9.4. For every strictly 2-local language L over Σ, there is a sequence of SPE rewrite rules that generates the image of Σ^* under $\mathrm{id}(L)$. ⌟

Proof. We will use rewrite rules to construct a filter that eliminates all strings of Σ^* that contain a bigram g_i that is not a member of 2-grams(w) for any $w \in L$. The first rewrite rule inserts a special symbol $*$ at the beginning of all such strings:

$$\varepsilon \rightarrow * \mid \_(\Sigma^+) \begin{Bmatrix} g_1 \\ \vdots \\ g_n \end{Bmatrix}$$

Since this is a parallel, mandatory rewrite rule, every ungrammatical string now starts with $*$. All we have to do is delete all symbols that occur to the left of a $*$, as well as $*$ itself.

$$\sigma \rightarrow \varepsilon \mid *(\Sigma^+)\_, \text{ for every } \sigma \in \Sigma$$
$$* \rightarrow \varepsilon$$

An SPE grammar that applies these rules from left to right is guaranteed to generate all members of L, and only those. □

Corollary 9.5. Every regular language is generated by some SPE grammar. ⌟

2.2 SPE Generates Only Regular Languages

We just proved that SPE can generate every regular language, but can it generate even more complex languages, languages that are not regular? There are two answers to this question. If we go by the definition of what an SPE grammar looks like, then SPE is shockingly powerful: every recursively enumerable (≈ computable) language is generated by some SPE grammar. That is about as powerful as it gets, and it is due to SPE using both context-sensitive rewriting and deletion of input segments. If every such SPE grammar where a possible natural language phonology, phonological dependencies could involve center embedding, crossing dependences, arbitrary copying, and restrictions that hold only if a word encodes a theorem of first-order logic. So from this perspective, SPE overgenerates to a ludicrous degree and utterly fails to make distinctions between natural and unnatural dependencies.

This view of SPE is at odds with how phonologists think of SPE. While it is true that SPE is considered too powerful, it is commonly assumed to be mostly in the right ballpark. And if we look at the analyses in the literature, it seems unlikely that any of them could be used to generate any of the patterns listed above. This is an important insight that was first pointed out by Johnson (1972) and later formalized by Kaplan and Kay (1994): SPE *as used by linguists* generates only regular languages.

The restriction that phonologists follow without even being aware of it is that the material rewritten by a rule may not be operated on by the very same rule. More precisely, suppose that we have a rewrite rule R that rewrites the substrings $w_{i,j}$ spanning from position i to j in R's input string w. Let $w^R_{i,j}$ be the substring of the output of R that corresponds to $w_{i,j}$. Then R may not rewrite any material of $w^R_{i,j}$.

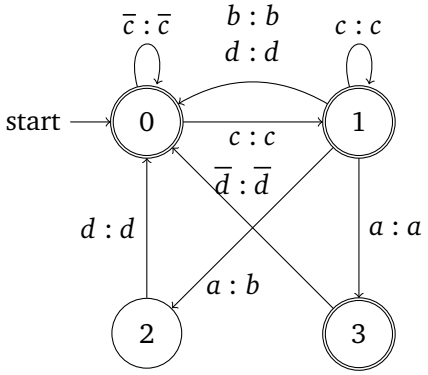
Intuitively, Johnson simply realized that we may think of a rewrite rule as a scanner window that moves through the string from left to right and rewrites material. If the window rewrites $w_{i,j}$ as $w^R_{i,j}$, then $w^R_{i,j}$ winds up immediately to the left of the window and thus cannot be operated on again.

Example 9.6 Recursive and Non-Recursive Rewrite Rule Application

Consider the rewrite rule $\varepsilon \rightarrow ab|\_b$, which inserts ab in front of b. With Johnson's restriction, this rule rewrites the string ab as $a(ab)^+b$. Without it, on the other hand, it also produces strings of the form a^nb^n, among others. See the table for an illustration, with the application domain highlighted in red.

recursive	non-recursive
ab	ab
$aabb$	$aabb$
$aaabbb$	$aababb$
$aaaabbbb$	$aababbab$

A rule that rewrites a single symbol and applies in parallel at all possible rewriting sites is fairly easy to translate into an FST. Suppose the rewrite rule in question is $a \rightarrow b \mid c\_d$. Then this rule can be thought of as a transducer that rewrites every symbol by itself except for as that are preceded by a c and followed by a d. The FST is depicted below, where $\overline{\sigma}$ is a placeholder for every $\sigma \in \Sigma \setminus \{\sigma\}$.



More sophisticated parallel rules are generalizations of this template, whereas left-to-right and right-to-left application of rules requires special diacritic symbols and is a lot more complicated. Nonetheless Kaplan and Kay (1994) showed successfully that all three types of rule application can be modeled with FSTs. The transduction carried out by a given SPE grammar with ordered rules $R_1 \cdots R_n$ thus is the composition of the FSTs corresponding to these rules. As FSTs are closed under composition, every SPE grammar can be represented by a single FST. And since FSTs generate regular languages, it follows that SPE generates only regular languages.

Theorem 9.6. SPE generates exactly the class of regular languages. ⌟

3 Comparing SPE and OT

3.1 A Formal Definition of OT

In the mid 90s, a new theory became the dominant paradigm in phonological research: Optimality Theory (OT; Prince and Smolensky 2004). OT uses violable ("soft"), ranked constraints in place of SPE's rewrite rules. It has been claimed that this limits its generative capacity in comparison to SPE and also allows it to state generalizations that could not be captured with SPE. We will see that at least the first claim is false.

First, the *generator* GEN maps every input string to its possible *output candidates*, which is usually taken to be every string that can be formed over the same alphabet. Since richness of the base is a standard assumption in OT, the generator computes the transduction $\Sigma^* \times \Sigma^*$. In addition, each constraint c restricts the range of a given transduction τ such that if $i\tau := \{o \mid \langle i, o\rangle \in \tau\}$ is the set of output candidates for input i, then c only keeps those $o \in i\tau$ that are *optimal* with respect to c. Optimality holds of o iff there is no $o' \in i\tau$ such that $c(o') < c(o)$, where $c(o)$ is the number of violations o incurs with respect to c. For example, the constraint $^*ab$ penalizes as being followed by b and is violated 2 times in *abab*, but 3 times in *ababab*. We denote the result of restricting transduction τ via c by $\tau \restriction c$. The transduction computed by an OT grammar with constraints $c_1, \ldots, c_n$, with c_i higher ranked than c_j for all $i < j$, is GEN $\restriction c_1 \restriction \cdots \restriction c_n$.

Example 9.7 A Small OT Grammar

Suppose that we have an OT grammar with two constraints, one being $^*ab$, the other one $2a!$, which requires at least two as to occur in the string. What are the optimal output candidates for *ab* under this grammar? This is easily determined via an *OT tableaux*.

ab	$^*ab$	$2a!$
ε		**
a		*
b		**
aa		
ab	*	*
ba		*
bb		**
⋮		
$aaaa$		
⋮		
$aabb$	*	
⋮		
$abab$	**	
⋮		

As you can see, the optimal output candidates are exactly those strings that contain at least 2 as and not a single instance of *ab*. Note that these optimal output candidates

do not violate either constraint, so it does not matter how we rank the two constraints.

Now suppose that we also have a third constraint ID that punishes any deviation from the input string. Then the ranking of constraints does make a difference. For example, if ID is the highest ranked constraint, *ab* is the only optimal output candidate for *ab*. If it the lowest ranked constraint, then the sole optimal output candidate is *aa*.

ab	ID	$^*ab$	2*a*!
ε	**		**
a	*		*
b	*		**
aa	*		
ab		*	*
ba	**		*
bb	*		**
⋮			
aaaa	***		
⋮			
aabb	**	*	
⋮			
abab	**	**	
⋮			

ab	$^*ab$	2*a*!	ID
ε		**	**
a		*	*
b		**	*
aa			*
ab	*	*	
ba		*	**
bb		**	*
⋮			
aaaa			***
⋮			
aabb	*		**
⋮			
abab	**		**
⋮			

The constraints used in the previous example fall into two distinct classes. The first two, $^*ab$ and 2*a*!, do not take the input into account. They are called *markedness constraints*. The constraint ID, on the other hand, is a *faithfulness constraint*, as it requires the output to deviate as little as possible from the input.

Definition 9.7 (OT Constraints). An OT constraint is a function $\Sigma^* \times \Sigma^* \times \mathbb{N}$ that maps input-output pairs to natural numbers. A constraint c is a *markedness* constraint iff there are no distinct inputs i and j such that $c(i,o) \neq c(j,o)$. Otherwise it is a *faithfulness* constraint.

Definition 9.8. An OT grammar is a pair $O := \langle C, < \rangle$, where $C := \{c_1, \ldots, c_n\}$ is a finite set of OT constraints and $<$ a linear order over C. The grammar O computes the transduction $\tau := \textsc{Gen} \restriction c_1 \restriction \cdots \restriction c_n$, where

- $c_i < c_{i+1}$ for all $1 \leq i < n$, and
- $R \restriction c_j := \big\{ \langle i,o \rangle \in R \mid \neg \exists o' \big[\langle i,o' \rangle \in R \wedge c_j(i,o') < c_j(i,o) \big] \big\}$.

The language generated by O is $O(L) := \bigcup_{i \in \Sigma^*} i\tau$.

3.2 Finite-State OT

It is obvious that the power of an OT grammar depends on its constraints. Without restrictions on what counts as a valid OT constraint, every recursively enumerable language can be generated by an OT grammar. Recall that the same is true for SPE if we do not block rewrite rules from rewriting their own output. In the case of SPE, phonologists intuitively followed that restriction, and likewise the set of commonly entertained OT constraints follows certain restrictions. For instance, there is no constraint in the literature that is satisfied by only those strings whose length is prime, or the Gödel number of a theorem of first-order logic. Nor has anybody proposed anything like a constraint that requires the same number of as and bs to occur in a string. In the following, we will show that the generative capacity of OT is exactly that of SPE as long as the constraints obey certain properties. We start out with a simple fragment of OT (Frank and Satta 1998; Karttunen 1998), which is subsequently extended to a more faithful formalization (Jäger 2002).

Categorical constraints Suppose that all constraints are markedness constraint and each string in Σ^* violates it at most once. In other words, each constraint c defines a language $L(c)$ of well-formed strings. If this language is regular, then the OT grammar generates a regular language.

Theorem 9.9. Let $O := \langle C, < \rangle$ be an OT grammar such that every $c \in C$ is a markedness constraint that defines a regular language. Then τ is a finite state transduction and $O(L)$ is regular. ⌟

Proof. The regularity of $O(L)$ follows immediately from the fact that Σ^* is regular and regular languages are closed under finite state transductions. The following is an inductive proof that τ is a finite-state transduction.

The base case is trivial, as $\textsc{Gen} := \Sigma^* \times \Sigma^*$ is obviously finite-state. Suppose, then, that R is a finite-state transduction. In this case, we have

$$R \downharpoonright c_i := \begin{cases} R \circ \mathrm{id}(L(c_i)) & \text{if } L(c_i) \neq \emptyset \\ R & \text{otherwise.} \end{cases}$$

For every regular language L, one can test whether it is empty. So we can remove from O all c_i that define an empty language, yielding a compacted grammar O' that computes the same transduction as O. Since $\mathrm{id}(L)$ is a finite-state transduction if L is regular, and since finite-state transductions are closed under composition, the transduction computed by O' — and thus O — is finite-state. □

Testing for emptiness can be omitted by using a more complicated definition instead, see Karttunen (1998) for details.

Note that every regular language L (except the empty language $\emptyset$) can be generated by such a restricted version of OT: we simply limit the set of constraints to a single c with $L(c) = L$. Since the empty language is of no interest to linguists, we can already conclude that OT is at least as powerful as SPE.

FST constraints Virtually no constraint in the literature satisfies the requirement that every string violates it at most once. Fortunately, though, the result can be extended to constraints that have no such upper bound.

Every markedness constraint c defines a relation R_c over Σ^* such that $\langle o, o' \rangle \in R$ iff $c(o) < c(o')$ (by definition we can ignore the input i for markedness constraints).

So the optimal outputs are exactly those for which there is no o' such that $\langle o', o\rangle \in R_c$. Now suppose that c is the i-th constraint of the OT grammar O, and that $\tau := \textsc{Gen} \restriction c_1 \restriction \cdots \restriction c_{i-1}$. Then we can relativize R_{c_i} to τ by removing all rankings that pertain to an output candidate that has already been eliminated: $rel_c^\tau := R_c \cap (\tau^{-1} \circ \tau)$. The relation $\tau^{-1} \circ \tau$ holds between outputs o and o' iff they are competing candidates for some input i. In other words, τ contains both $\langle i, o\rangle$ and $\langle i, o'\rangle$.

Definition 9.10 (Rational constraint). A constraint c is *rational* with respect to transduction τ iff there is a finite state transduction S such that $S \cap \tau = rel_c^\tau$.

Intuitively, a constraint c is rational if we can find a finite state transduction S that describes the same ranking over the set of output candidates produced by τ. So the finite state transduction does not have to fully replicate c, the two only need to agree on the remaining candidates. Given such an S, the set of suboptimal candidates is simply $\text{ran}(R \circ S) := \{o \mid \langle i, o\rangle \in R \circ S\}$. The relative complement $\text{ran}(R) \setminus \text{ran}(R \circ S)$ thus is the set of optimal output candidates. Notice that this is guaranteed to be a regular language thanks to the closure properties of finite state transductions and regular languages. Consequently, the identity function over this set is a finite state transduction, so it follows that

$$R \restriction c_i := R \circ \text{id}(\text{ran}(R) \setminus \text{ran}(R \circ S_i))$$

is a finite state transduction, too.

Theorem 9.11. Let $O := \langle C, <\rangle$ be an OT grammar such that every $c_i \in C$ is a rational markedness constraint with respect to $\textsc{Gen} \restriction c_1 \restriction \cdots \restriction c_{i-1}$. Then τ is a finite state transduction and $O(L)$ is regular. ⌟

Two things should be kept in mind. First, if c_i can be defined in terms of a finite state transducer, then it is always a rational constraint since one valid choice for S in this case is the transitive closure of c_i.

Example 9.8 A Rational OT Constraint

The constraint $^*ab$ can be modeled by a transducer that non-deterministically inserts

- a b after an a, or
- $a\,b$ at arbitrary points.

For instance a can be rewritten as ab, aba, or $abaab$, but not as ba. Hence we have $a < ab$, $a < aba$, and $a < abaab$, but $a \not< ba$ (note that $aba < abaab$, too).

More importantly, the generalized lenient composition gives the intended result only if optimality is a global property that is independent of the choice of input.

Definition 9.12 (Global Optimality). Given an OT grammar O, optimality is *global* iff it holds for all o that $\langle i, o\rangle \in \tau$ and $\langle j, o\rangle \in \textsc{Gen}$ implies $\langle j, o\rangle \in \tau$.

Global optimality requires for every output candidate that is optimal for input i that it is also optimal for every other input j that is an output candidate for. This is the case as long as $\text{GEN} = \Sigma^* \times \Sigma^*$ and all constraints are markedness constraints, but not if the grammar also contains faithfulness constraints. So this approach still cannot handle faithfulness constraints.

Faithfulness constraints Riggle (2004) develops a very different formalization in terms of *weighted FSTs*. These are FSTs where every arc also has a weight attached it. Riggle uses these weights to encode constraint violations: whenever a string takes a specific path, it may incur a violation of a specific constraint that corresponds to the weight of the arc. Unfortunately we do not have the time to discuss this approach in greater detail.

4 Insights about SPE and OT

We have seen that both SPE and OT can generate every regular language, even if we consider only very weak fragments of the formalisms. For SPE, it suffices to have a strictly 2-local set of underlying forms and a few simple rewrite rules that compute the desired projection. If one wants to also eliminate the set of underlying representations, one needs three additional rules that act as a filter on Σ^*. For OT, a single markedness constraint is sufficient to generate every regular language. A realistic OT grammar with output and faithfulness constraints might even generate non-regular languages, disproving the common assumption that OT is more restricted than SPE. At any rate, both formalisms are more powerful than what is needed for phonology. In fact, they are so powerful that the language class they describe cannot be learned in the limit from positive text, and there is no obvious way to limit their expressivity.

Crucially, though, our notion of power focuses on the generated languages. But it could be argued that a phonological theory must go beyond describing the correct output forms and also has to specify the correct mappings from underlying forms to surface forms. After all, the knowledge that ['Ra:t] is a well-formed string of German is rather useless if the speaker cannot map the word to the lexical entry /Ra:d/ 'wheel'. Maybe a theory that is expressive enough to describe all these mappings cannot do without a certain amount of power that also allows it to generate arbitrary regular languages. That seems rather odd — why, then, is the range of attested natural language phonologies such a narrowly restricted subset of the regular languages? But this scenario cannot be ruled out as too little is currently known about the computational properties of the mappings from underlying forms to surface forms. Fortunately this subject has gotten a lot more attention in recent years, see e.g. Chandlee (2014). Incomplete as the picture may be at this point, these first forays already provide tentative evidence that SPE and OT overgenerate even with respect to the mappings and that their increased power does not allow for significantly more succinct descriptions of the empirical phenomena.