An FSA defines a set of strings.

It is primarily used to accept and reject strings.

Sheep Language
Instead of defining sets of strings, FSTs define sets of PAIRS of strings. A set of pairs in set theory is called a relation.

FSTs define relations on strings. Not all string relations can be captured by FSTs (as we’ll see). The set that can be captured is called the regular relations.
Given a string that consist of an even number of a’s followed by an even number of b’s, it will swap the a’s and b’s.
1. Two alphabets in the sheep language transducer:

   Upper  \{a, b, !\}
   Lower  \{1, 2, !\}

2. In the A and B swapper, the upper and lower alphabets are the same.

3. An FST also has two languages, an upper and lower language. In the sheep language transducer, the upper language is $baa^+!$, and the lower language is $21^+!$.

4. The FST defines a correspondence between members of the lower and upper alphabets, so there’s also a set of **feasible pairs**, giving the possible pairings of upper-alphabet symbols with lower-alphabet symbols (\{a:1, b:2, !:!\}).
An FSA with two tapes

We call the two tapes the upper and lower tapes.

1. As **recognizers**, FSTs read both tapes and check to see that the strings on the two tapes stand in the relation described by the machine.

2. As **analyzers**, they read the lower tape and find all possible upper strings that stand in the relation described by the machine. In our terms they find all analyses of a surface string.

3. As **generators**, they read the upper tape and find all possible lower strings that stand in the relation described by the machine. In our terms they find all possible realizations of an underlying string.
AS with FSAs, FSTs can be deterministic or non-deterministic.
$\epsilon$:a An "a" on the lower tape that does not correspond to anything on the upper tape (insert an "a" on the lower tape).

a:$\epsilon$ The "a" on the upper tape does not correspond to anything on the lower tape. (omit the "a" on the lower tape)

$\epsilon$: $\epsilon$ Jump to another state without reading either tape.

Any of these types of transitions is quite problematic if it loops from state $i$ to state $i$. Why?
Closure properties

From Jurafsky and Martin, Ch 2, we learned FSAs are closed under:

1. Union
2. Intersection
3. Concatenation
4. Complementation
5. Reversal

FSTs have only a subset of these closure properties:

1. Union
2. Concatenation
L_1 = \{\langle a^n, b^* c^n \rangle \mid n > 0\} \quad \text{A relation that pairs } n \text{ underlying } a \text{’s with any number of } b \text{’s followed by exactly } n \text{ c } \text{’s.}

L_2 = \{\langle a^n, b^n c^* \rangle \mid n > 0\} \quad \text{A relation that pairs } n \text{ underlying } a \text{’s with exactly } n \text{ b } \text{’s followed by any number of c } \text{’s.}

L_1 \cap L_2 = \{\langle a^n, b^n c^n \rangle \mid n > 0\}

The lower language is not regular, so the relation cannot be regular.
## Chomsky Hierarchy

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Automaton</th>
<th>Rule Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Recursive enumerable</td>
<td>Turing</td>
<td>Any rewrite rule</td>
</tr>
<tr>
<td>1</td>
<td>Context sensitive</td>
<td>LBND Turing</td>
<td>A → B  C___  D</td>
</tr>
<tr>
<td>2</td>
<td>Context-Free</td>
<td>ND PDA</td>
<td>A → BC . . .</td>
</tr>
<tr>
<td>3</td>
<td>Regular</td>
<td>FSA</td>
<td>A → aB, A → Ba</td>
</tr>
</tbody>
</table>

### Examples

- \((ab)^n\) Regular
- \(a^n b^n\) Context-free
- \(XX^R\) Mirror-language: Context-free
- \(a^n b^n c^n\) Context-sensitive
Phonological/morphological rewrite rules

A → B C___ D

ε → ab\ ___ b

Applied to input ab, generates $a^n b^n$.

**Johnson’s 1972 insight**

The rule generates a regular language **only if** it is allowed to reapply endlessly in the same position. But phonologists never had this interpretation in mind. Rules always drift rightward in string after every application.
### The drifting application regime produces the regular language $a(ab)^n b$

<table>
<thead>
<tr>
<th>in-place</th>
<th>Drift right</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \uparrow b$</td>
<td>$a \uparrow b$</td>
</tr>
<tr>
<td>$aa \uparrow bb$</td>
<td>$aab \uparrow b$</td>
</tr>
<tr>
<td>$aaa \uparrow bbb$</td>
<td>$aabab \uparrow b$</td>
</tr>
<tr>
<td>$aaaa \uparrow bbbb$</td>
<td>$aababab \uparrow b$</td>
</tr>
</tbody>
</table>

**Johnson’s result**

Johnson showed that if the rules were only used under the drifting application regime, the languages produced were regular. (Also see Karttunen 1993)
Rules describe (regular) relations

A → B C D
CAD → CBD

<table>
<thead>
<tr>
<th>In</th>
<th>Out</th>
<th>Yes/No</th>
</tr>
</thead>
<tbody>
<tr>
<td>cad</td>
<td>cbd</td>
<td>yes</td>
</tr>
<tr>
<td>cad</td>
<td>cad</td>
<td>no</td>
</tr>
<tr>
<td>cae</td>
<td>cae</td>
<td>yes</td>
</tr>
<tr>
<td>cae</td>
<td>cbe</td>
<td>yes</td>
</tr>
</tbody>
</table>

Approximate rule translation

\[ a \rightarrow b \backslash L \_\_ R \]

\[ \text{Id}(\Sigma^*) \ (\text{Opt}(\text{Id}(L) \ a \times b \ \text{ID}(R)))^* \]

Kaplan and Kay 1994
Rules are just FSTs

Consequence

Rules can be compiled into FSTs. Rule FSTs can be combined together into bigger FSTs that enforce all the rules. The rule applications can be ordered (linguists have shown this is necessary).
E-insertion revisited

\[ \epsilon \rightarrow \emptyset \hat{\epsilon} : \epsilon \quad \_\_\_ \quad S \quad \# \]
Concatenation

Does not enforce spelling rules.
Composition of relations

Example
accurate       inaccurate
definite      indefinite
competent    incompetent
patient        impatient
mature       immature

Prefix will be *iN*.

Order dependency

\[
\begin{align*}
R1. \quad & N \rightarrow m \setminus \{b, p\} \\
R2. \quad & N \rightarrow n
\end{align*}
\]

What we need is \( R1 \circ R2 \)
## Composition of Transducers II

### Composition of two relations

<table>
<thead>
<tr>
<th>Relation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Apfel:alma</td>
</tr>
<tr>
<td>T2</td>
<td>alma:apple</td>
</tr>
<tr>
<td>T2(\circ)T1</td>
<td>Apfel:apple</td>
</tr>
<tr>
<td>R1</td>
<td>iNpatient:impatient</td>
</tr>
<tr>
<td>R1</td>
<td>iNdefinite:iNdefinite</td>
</tr>
<tr>
<td>R2</td>
<td>iNdefinite:indefinite</td>
</tr>
<tr>
<td>R2(\circ)R1</td>
<td>iNpatient:impatient</td>
</tr>
<tr>
<td>R2(\circ)R1</td>
<td>iNdefinite:indefinite</td>
</tr>
</tbody>
</table>

In \(R2\circ R1\), in effect, the lower tape of \(R1\) becomes the upper tape of \(R2\) (\(R1\) output \(R1\) becomes \(R2\) input).
Combining concatenated lexicon & spelling rules

We **compose** the entire concatenated lexicon with the spelling rules. Concatenated *fox+s* becomes the input to e-insertion, emerging as *fox+es*. This is why the spelling rules have to allow all strings not falling in a rule’s domain to pass through unchanged.
1. Each part of the lexicon starts out as a transducer.

2. We **concatenate** the prefix, root, and suffix transducers. In practice there are different transducers for different parts of speech (noun, verb, adjective, noun suffix, verb suffix, adjective suffix), and different morphological classes (regular and irregular verbs).

3. In theory, we **compose** the concatenated lexicon with the spelling rule transducer (composition of all spelling rules). In practice, this gets very big, and we apply the rules in parallel when possible. (Regular relations are closed under serial composition: Kaplan & Kay 1994)

4. Implementation of FSTs being used in this class (Karttunen 2000).
*Formal aspects of phonological description.*  
The Hague: Mouton.  
University of California at Berkeley Ph Dissertation 1970.

Regular models of phonological rule systems.  

*Finite-State Morphology: Xerox Tools and Techniques.*  
Cambridge University Press.

Finite-state constraints.  
University of Chicago Press.