

Natural Language Processing

Lecture 2: Words and Morphology

Linguistic Morphology

The shape of Words to Come

What? Linguistics?

- One common complaint we receive in this course goes something like the following:
I'm not a linguist, I'm a computer scientist! Why do you keep talking to me about linguistics?
- NLP is not just P; it's also NL
- Just as you would need to know something about biology in order to do computational biology, you need to know something about natural language to do NLP
- If you *were* linguists, we wouldn't have to talk much about natural language because you would already know about it

What is Morphology?

- Words are not atoms
 - They have internal structure
 - They are composed (to a first approximation) of **morphemes**
 - It is easy to forget this if you are working with English or Chinese, since they are simpler, morphologically speaking, than most languages.
 - But...
 - **mis-understand-ing-s**
 - 同志**们** tongzhi-**men** 'comrades'

Kind of Morphemes

- **Roots**

- The central morphemes in words, which carry the main meaning

- **Affixes**

- Prefixes
 - **pre**-nuptial, **ir**-regular
- Suffixes
 - determin-**ize**, iterat-**or**
- Infixes
 - Pennsyl-**f**kin**-vanian
- Circumfixes
 - **ge**-sammel-**t**

Nonconcatenative Morphology

- **Umlaut**

- foot : feet :: tooth : teeth

- **Ablaut**

- sing, sang, sung

- **Root-and-pattern or templatic morphology**

- Common in Arabic, Hebrew, and other Afroasiatic languages
- Roots made of consonants, into which vowels are shoved

- **Infixation**

- **Gr-um-adwet**

Functional Differences in Morphology

- **Inflectional morphology**

- Adds information to a word consistent with its context within a sentence
- Examples
 - Number (singular versus plural)
automaton → *automata*
 - *Walk* → *walks*
 - Case (nominative versus accusative versus...)
he, him, his, ...

- **Derivational morphology**

- Creates new words with new meanings (and often with new parts of speech)
- Examples
 - *parse* → *parser*
 - *repulse* → *repulsive*

Irregularity

- **Formal irregularity**

- Sometimes, inflectional marking differs depending on the root/base
 - *walk : walked : walked :: sing : sang : sung*

- **Semantic irregularity/unpredictability**

- The same derivational morpheme may have different meanings/functions depending on the base it attaches to
 - *a kind-ly old man*
 - **a slow-ly old man*

The Problem and Promise of Morphology

- Inflectional morphology (especially) makes instances of the same word appear to be different words
 - Problematic in information extraction, information retrieval
- Morphology encodes information that can be useful (or even essential) in NLP tasks
 - Machine translation
 - Natural language understanding
 - Semantic role labeling

Morphology in NLP

- The processing of morphology is largely a solved problem in NLP
- A rule-based solution to morphology: finite state methods
- Other solutions
 - Supervised, sequence-to-sequence models
 - Unsupervised models

Levels of Analysis

Level	hugging	panicked	foxes
Lexical form	hug +V +Prog	panic +V +Past	fox +N +Pl fox +V +Sg
Morphemic form (intermediate form)	hug^ing#	panic^ed#	fox^s#
Orthographic form (surface form)	hugging	panicked	foxes

- In morphological analysis, map from orthographic form to lexical form (using morphemic form as intermediate representation)
- In morphological generation, map from lexical form to orthographic form (using the morphemic form as intermediate representation)

Morphological Analysis and Generation: How?

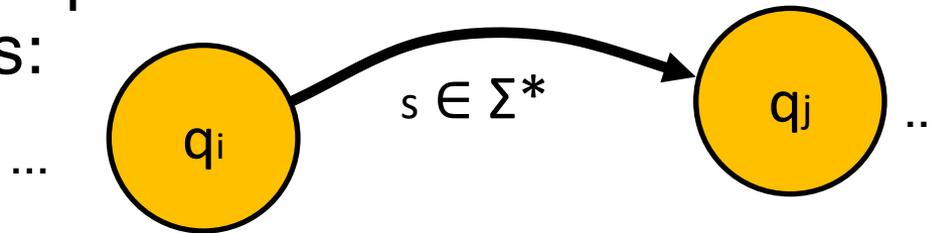
- Finite-state transducers (FSTs)
 - Define **regular relations** between strings
 - “foxes” \mathcal{R} “fox +V +3p +Sg +Pres”
 - “foxes” \mathcal{R} “fox +N +Pl”
 - Widely used in practice, not just for morphological analysis and generation, but also in speech applications, surface syntactic parsing, etc.
 - Once compiled, **run in linear time** (proportional to the length of the input)
- To understand FSTs, we will first learn about their simpler relative, the FSA or FSM
 - Should be familiar from theoretical computer science
 - FSAs can tell you whether a word is morphologically “well-formed” but cannot do analysis or generation

Finite State Automata

Accept them!

Finite-State Automaton

- Q : a finite set of states
- $q_0 \in Q$: a special start state
- $F \subseteq Q$: a set of final states
- Σ : a finite alphabet
- Transitions:



- Encodes a **set** of strings that can be recognized by following paths from q_0 to some state in F .

A “baaaaa!” Example of an FSA

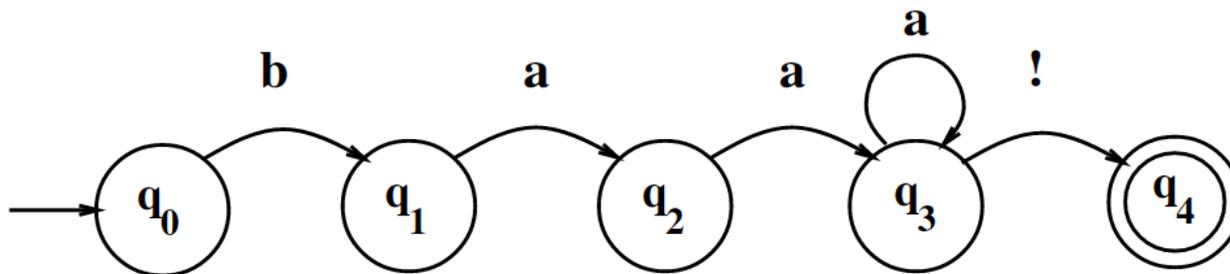


Figure 2.10 A finite-state automaton for talking sheep.

Don't Let Pedagogy Lead You Astray

- To teach about finite state machines, we often trace our way from state to state, consuming symbols from the input tape, until we reach the final state
- **While this is not wrong, it can lead to the wrong idea**
- What are we actually asking when we ask whether a FSM accepts a string? Is there a path through the network that...
 - Starts at the initial state
 - Consumes each of the symbols on the tape
 - Arrives at a final state, coincident with the end of the tape
- Think **depth-first search!**

Formal Languages

- A formal language is a set of strings, typically one that can be generated/recognized by an automaton
- A formal language is therefore potentially quite different from a natural language
- However, a lot of NLP and CL involves treating natural languages like formal languages
- The set of languages that can be recognized by **FSA**s are called **regular languages**
- **Conveniently, (most) natural language morphologies belong to the set of regular languages**

FSAs and Regular Expressions

- The set of languages that can be characterized by FSAs are called “regular” as in “**regular expression**”
- Regular expressions, as you may know, are a fairly convenient and standard way to represent something equivalent to a finite state machine
 - The equivalence is pretty intuitive (see the book)
 - There is also an elegant proof (not in the book)
- Note that “regular expression” implementations in programming languages like Perl and Python often go beyond true regular expressions

FSA for English Derivational Morphology

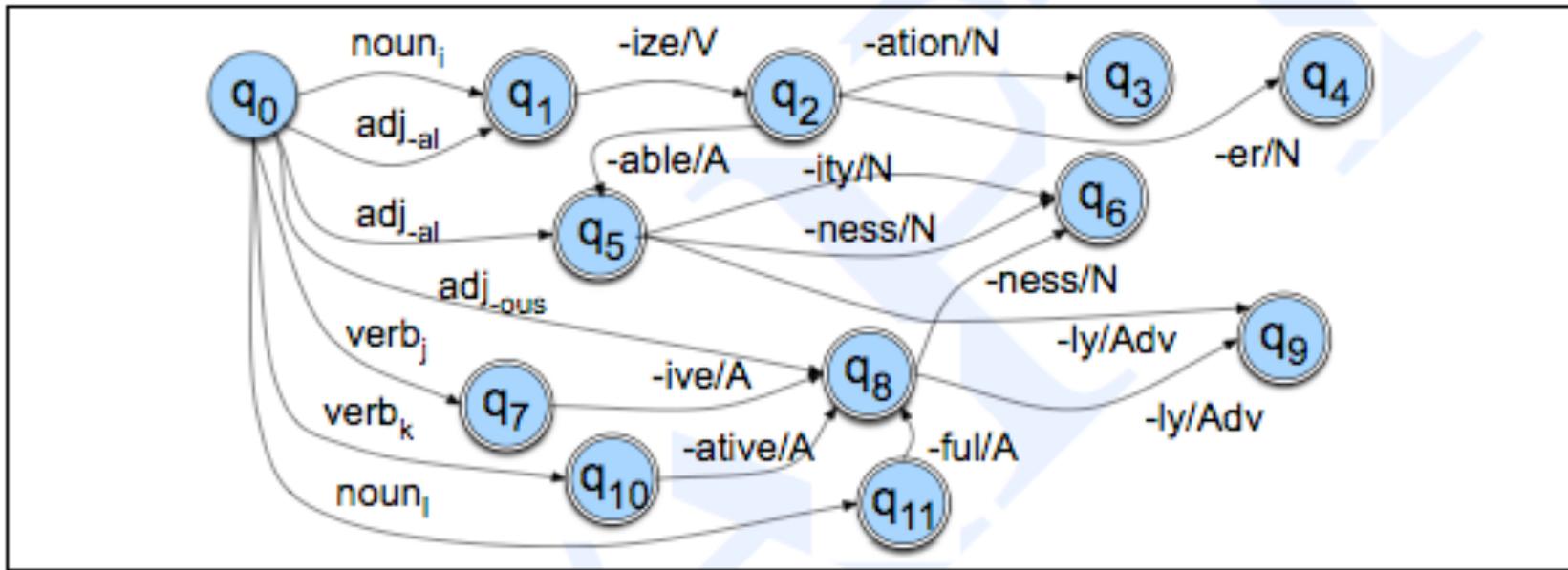


Figure 3.6 An FSA for another fragment of English derivational morphology.

Finite State Transducers

I am no longer accepting the things I cannot change; I am changing the things that I cannot accept

Morphological Parsing/Analysis

Input: a word

Output: the word's stem(s)/lemmas and features expressed by other morphemes.

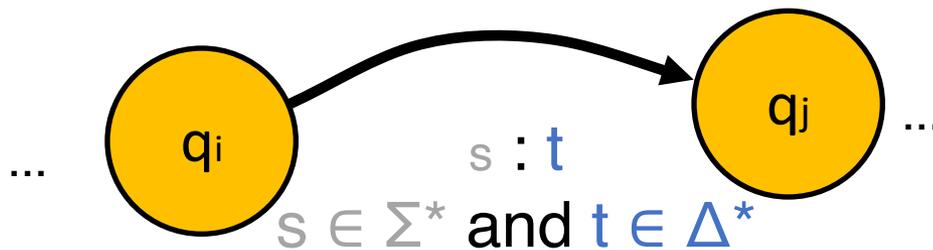
Example: geese → {goose +N +Pl}
gooses → {goose +V +3P +Sg}
dog → {dog +N +Sg, dog +V}
leaves → {leaf +N +Pl, leave +V +3P +Sg}

Three Solutions

1. Table
2. Trie
3. Finite-state transducer

Finite State Transducers

- Q : a finite set of states
- $q_0 \in Q$: a special start state
- $F \subseteq Q$: a set of final states
- Σ and Δ : two finite alphabets
- Transitions:



Turkish Example

uygarlaştıramadıklarımızdanmışsınızcasına

“(behaving) as if you are among those whom we were not able to civilize”

<i>uygar</i>	“civilized”
+ <i>laş</i>	“become”
+ <i>tır</i>	“cause to”
+ <i>ama</i>	“not able”
+ <i>dık</i>	past participle
+ <i>lar</i>	plural
+ <i>ımız</i>	first person plural possessive (“our”)
+ <i>dan</i>	second person plural (“y’all”)
+ <i>miş</i>	past
+ <i>sınız</i>	ablative case (“from/among”)
+ <i>casına</i>	finite verb → adverb (“as if”)

Morphological Parsing with FSTs

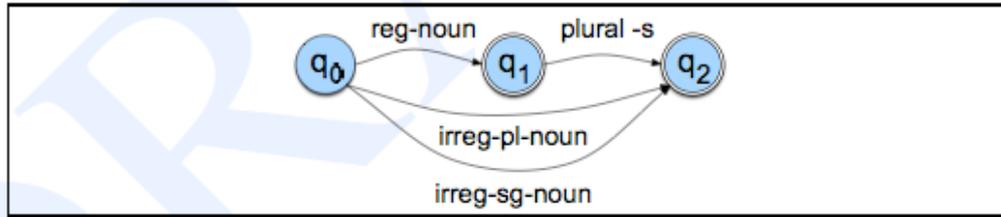


Figure 3.3 A finite-state automaton for English nominal inflection.

reg-noun	irreg-pl-noun	irreg-sg-noun	plural
fox	geese	goose	-s
cat	sheep	sheep	
aardvark	mice	mouse	

reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o : e o : e s e	goose
cat	sheep	sheep
aardvark	m o : i u : e s : c e	mouse

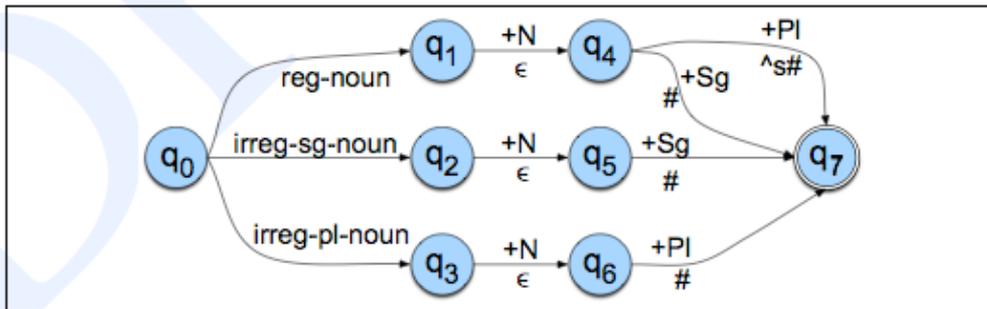


Figure 3.13 A schematic transducer for English nominal number inflection T_{num} . The symbols above each arc represent elements of the morphological parse in the lexical tape; the symbols below each arc represent the surface tape (or the intermediate tape, to be described later), using the morpheme-boundary symbol \wedge and word-boundary marker $\#$. The labels on the arcs leaving q_0 are schematic, and need to be expanded by individual words in the lexicon.

- Note “same symbol” shorthand.
- \wedge denotes a morpheme boundary.
- $\#$ denotes a word boundary.
- \wedge and $\#$ are not there automatically—they must be inserted.

Separation of concerns

- Typically, a morphological analyzer will be divided into (at least) two sections, each implemented with a separate FST:
 - Morphotactics
 - Allomorphic/orthographic rules
- Morphotactics
 - Maps between “zoch +N +Pl” and “zoch^s#”
 - Concatenates the “basic” form of morphemes together
 - Lemmas concatenated with affixes
 - Lemma can be “guessed”
- Allomorphic rules
 - Maps between output of morphotactics (intermediate or morphemic representation) and surface representation
 - “zoch^s#” \leftrightarrow “zoches”

English Spelling (Orthographic Rules)

Name	Description of Rule	Example
Consonant doubling	1-letter consonant doubled before <i>-ing/-ed</i>	beg/begging
E deletion	Silent e dropped before <i>-ing</i> and <i>-ed</i>	make/making
E insertion	e added after <i>-s,-z,-x,-ch,-sh</i> before <i>-s</i>	watch/watches
Y replacement	<i>-y</i> changes to <i>-ie</i> before <i>-s</i> , <i>-i</i> before <i>-ed</i>	try/tries
K insertion	verbs ending with <i>vowel + -c</i> add <i>-k</i>	panic/panicked

The E Insertion Rule as a FST

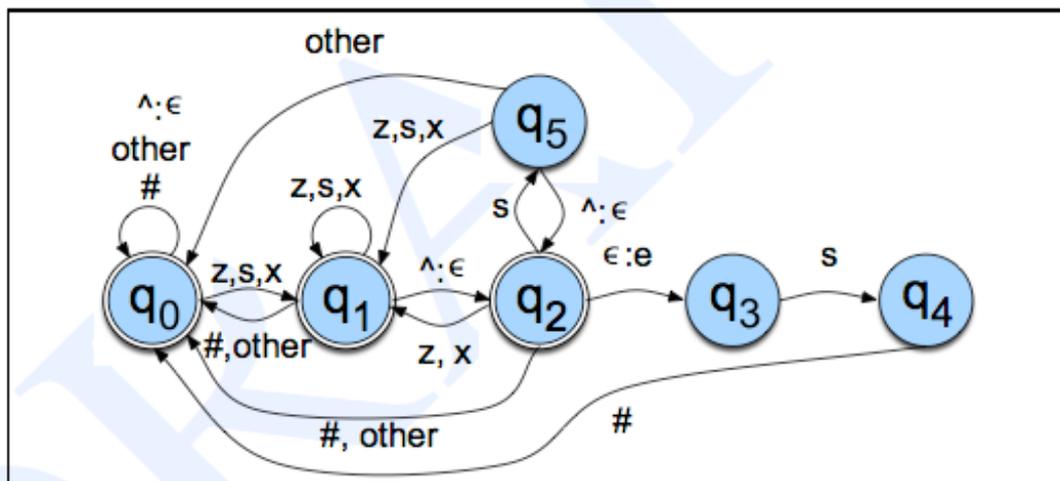


Figure 3.17 The transducer for the E-insertion rule of (3.4), extended from a similar transducer in Antworth (1990). We additionally need to delete the # symbol from the surface string; this can be done either by interpreting the symbol # as the pair #:ε, or by postprocessing the output to remove word boundaries.

$$\varepsilon \rightarrow e / \left\{ \begin{array}{c} s \\ x \\ z \end{array} \right\} \hat{\quad} \text{---} s\#$$

FST in Theory, Rule in Practice

- There are a number of FST toolkits (XFST, HFST, Foma, etc.) that allow you to compile rewrite rules into FSTs
- Rather than manually constructing an FST to handle orthographic alternations, you would be more likely to write rules in a notation similar to the rule on the preceding slide.
- Cascades of such rules can then be compiled into an FST and composed with other FSTs
- For your homework, you will construct FSTs directly, using some code to make the process tractable.

Combining FSTs

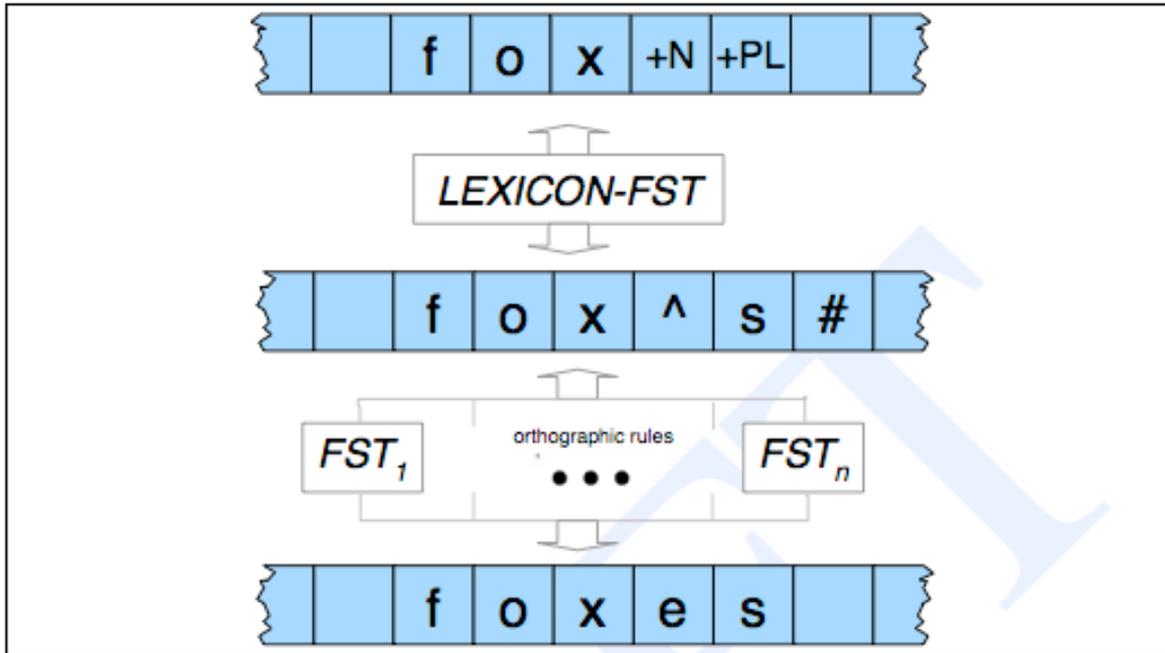


Figure 3.19 Generating or parsing with FST lexicon and rules

parse



generate

Operations on FSTs

- There are a number of operations that can be performed on FSTs:
 - **composition:** Given transducers T and S , there exists a transducer $T \circ S$ such that $x[T \circ S]z$ iff $x[T]y$ and $y[S]z$; effectively equivalent to feeding an input to T , collecting the output from T , feeding this output to S and collecting the output from S .
 - **concatenation:** Given transducers T and S , there exists a transducer $T \cdot S$ such that $x_1x_2[T \cdot S]y_1y_2$ and $x_1[T]y_1$ and $x_2[S]y_2$.
 - **Kleene closure:** Given a transducer T , there exists a transducer T^* such that $\epsilon[T^*]\epsilon$ and if $w[T^*]y$ and $x[T]z$ then $wx[T^*]yz$; $x[T^*]y$ only holds if one of these two conditions holds.
 - **union:** Given transducers T and S , there exists a transducer $T \cup S$ such that $x[T \cup S]y$ iff $x[T]y$ or $x[S]y$.
 - **intersection:** Given transducers T and S , there exists a transducer $T \cap S$ such that $x[T \cap S]y$ iff $x[T]y$ and $x[S]y$. **FSTs are not closed under intersection.**

Algorithm 1 Weighted-Composition(T_1, T_2)

```
1:  $Q \leftarrow I_1 \times I_2$ 
2:  $\mathcal{K} \leftarrow I_1 \times I_2$ 
3: while  $\mathcal{K} \neq \emptyset$  do
4:    $q = (q_1, q_2) \leftarrow \text{Head}(\mathcal{K})$ 
5:    $\text{Dequeue}(\mathcal{K})$ 
6:   if  $q \in I_1 \times I_2$  then
7:      $I \leftarrow I \cup \{q\}$ 
8:      $\lambda(q) \leftarrow \lambda_1(q_1) \otimes \lambda_2(q_2)$ 
9:     if  $q \in F_1 \times F_2$  then
10:       $F \leftarrow F \cup \{q\}$ 
11:       $p(q) \leftarrow p_1(q_1) \otimes p_2(q_2)$ 
12:      for each  $(e_1, e_2) \in E[q_1] \times E[q_2]$  such that  $o[e_1] = i[e_2]$  do
13:        if  $(q') = (n[e_1], n[e_2] \notin Q)$  then
14:           $Q \leftarrow Q \cup \{q'\}$ 
15:           $\text{Enqueue}(\mathcal{K}, q')$ 
16:           $E \leftarrow E \uplus \{(q, i[e_1], o[e_2], w[e_1] \otimes w[e_2], q')\}$ 
17: return  $T$ 
```

Algorithm complexity	
Time	$O(V_1 \cdot V_2 \cdot D_1 \cdot (\log D_2 + M_2))$
Space	$O(V_1 \cdot V_2 \cdot V_1 \cdot M_2)$
$V_i = \# \text{states}$, $D_i = \max \text{ out-degree}$, $M_i = \max \text{ multiplicity of } i^{\text{th}} \text{ WFST}$. www.OpenFST.org	

FST Operations

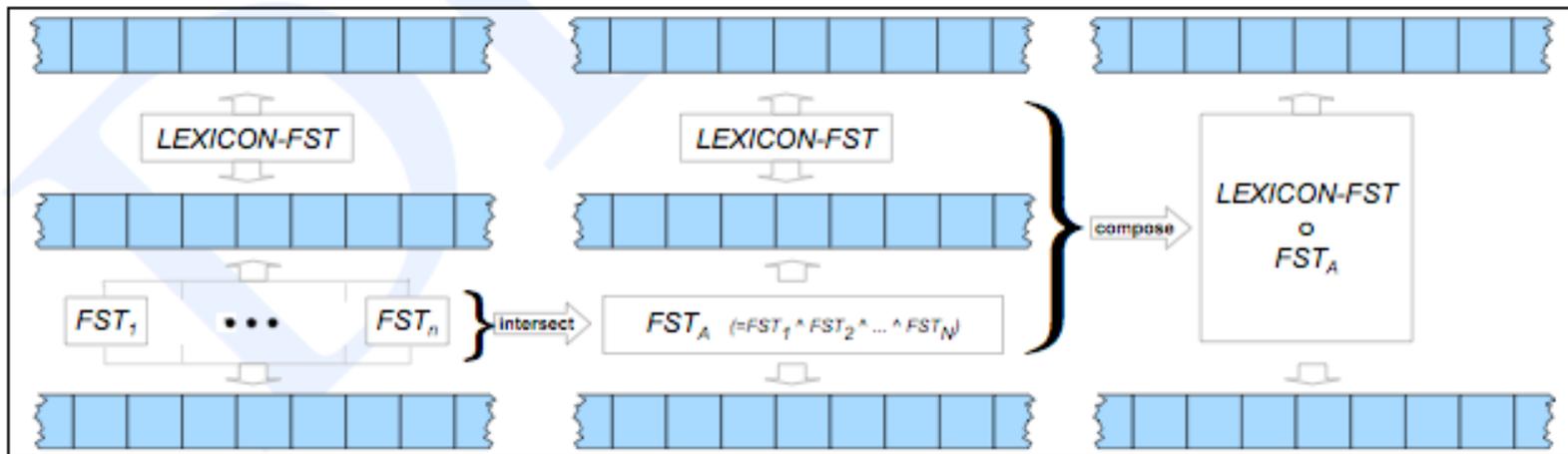


Figure 3.21 Intersection and composition of transducers.

A Word to the Wise

- You will be asked to create **FSTs** in a homework assignment and on an exam
- Sometimes, you will need to draw multiple **FSTs** and then combine them using **FST** operations
- The most common of these is **composition**
- If you catch yourself saying “**The output of FST A is the input to FST B,**” stop yourself and instead say “**Compose FST A with FST B**” or simply “ **$A \circ B$** ”

ML and Morphology

- Morphology is one area where—in practice—you may want to use hand-engineered rules rather than machine learning
- ML solutions for morphology do exist, including interesting unsupervised methods
- However, unsupervised methods typically give you only the parse of the word into morphemes (prefixes, root, suffixes) rather than lemmas and inflectional features, which may not be suitable for some applications

Some Notes on Homework 2

- Instead of analysis, you will do lemmatization: *squig*, *squigging*, *squigs*, and *squigged* should all map to *squig*
- You will implement delemmatization for in-vocabulary words (FST₁)
 - Make a transducer from lemma to surface form for each surface form-lemma pair (programmatically)
 - Take the union of these transducers
- You will implement morphotactics for OOV (out-of-vocabulary) words (FST₂)
 - Make a transducer for each affix, from lemma to morphemic form
 - Take the union of these transducers
- You will implement orthographic rules for OOV words (FST₃)
 - Make a transducer for each rule, from intermediate/morphemic form
 - Compose them, together with a post-processing FST that removes word and morpheme boundaries
- You will compose FST₂ and FST₃, then take the union of the resulting FST and FST₁
- This FST will perform delemmatization; invert it to lemmatize

STEMMING → STEM

Stemming (“Poor Man’s Morphology”)

Input: a word

Output: the word’s stem (approximately)

Examples from the Porter stemmer:

- sses → -ss
- ies → i
- ss → s

no	no
noah	noah
nob	nob
nobility	nobil
nobis	nobi
noble	nobl
nobleman	nobleman
noblemen	noblemen
nobleness	nobl
nobler	nobler
nobles	nobl
noblesse	nobless
noblest	noblest
nobly	nobli
nobody	nobodi
noce	noce
nod	nod
nodded	nod
nodding	nod
noddle	noddl
noddles	noddl
noddy	noddi
nods	nod

Tokenization

Tokenization

Input: raw text

Output: sequence of **tokens** normalized for easier processing.

“Tokenization is easy, they said!
Just split on whitespace, they
said!”*

*Provided you're working in English so words are (mostly)
whitespace-delimited, but even then...

The Challenge

Dr. Mortensen said tokenization of English is “harder than you’ve thought.” When in New York, he paid \$12.00 a day for lunch and wondered what it would be like to work for AT&T or Google, Inc.

Finite State Tokenization

- How can finite state techniques be used to tokenize text?
- Why might they be useful?
- Can you think of other potential tokenization techniques?