Natural Language Processing

Lecture 2: Words and Morphology

Learning Objectives

- **By the end of this lecture, you should know the following things:**
  - Words have internal structure
  - What a morpheme is
  - What a root is
  - What kinds of affixes and formal operations there are in morphology
  - The inflection/derivation distinction and why it's important in NLP
  - The types of languages that exist with respect to morphology (isolating, agglutinative, fusional, etc.)
  - The importance of morphology as a problem (and resource) in NLP
  - What lemmatization and stemming are
  - The finite-state paradigm for morphological analysis and lemmatization
- **By the end of this lecture, you should be able to do the following things:**
  - Find internal structure in words
  - Distinguish prefixes, suffixes, and infixes
  - Construct a simple FST for lemmatization

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
- We tell you about linguistics because we think you are unlikely to already know about it
- And we hope that, by the end of this lecture, you will have learned something you didn't know about words

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’

*Morphemes are minimal meaningful components of words.

Kind of Morphemes

- **Roots**
  - The central morphemes in words, which carry the main meaning
- **Affixes**
  - Prefixes
    - **pre**-nuptual, **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*

Inflection, Derivation, Lemmatization, and Stemming

- For many NLP tasks, it is desirable to remove inflectional morphology (which does not change the meaning or part of speech of words) but leave derivational morphology behind.
- For example, in information retrieval if you search for *repurpose* you probably also want to return *repurposing*, *repurposes*, and *repurposed* but probably not *purpose*.
- To do this, data may be normalized using **stemming** (which "chops off" inflectional affixes to yield a "stem") or **lemmatization** (which returns the "dictionary" form of a word).
- The second homework will be on lemmatization.

Irregularity

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

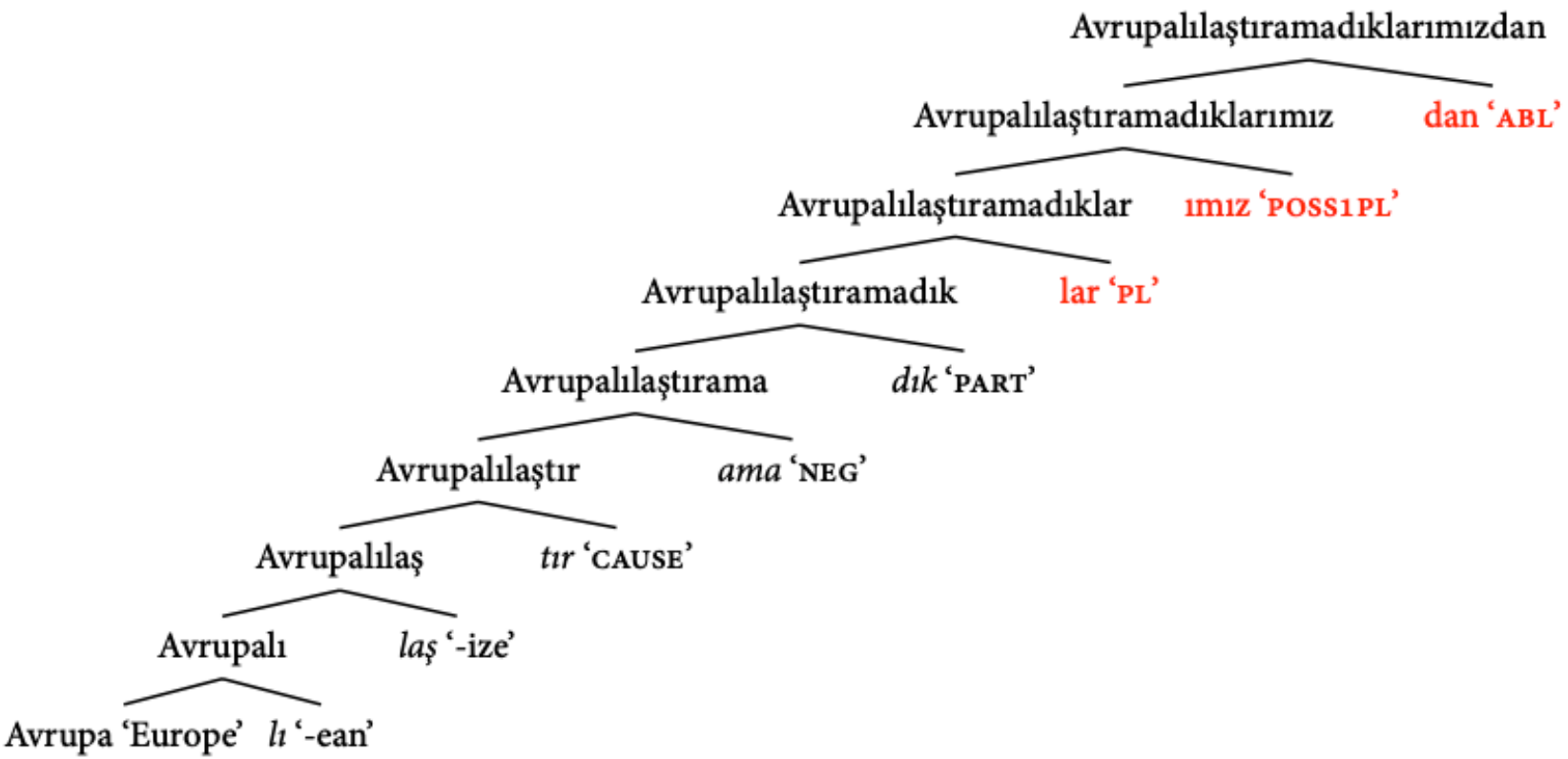
Morphological Typology

- Languages differ in how they do morphology and in how much morphology they have
  - **Isolating (or analytic**) languages like Chinese or English have very little inflectional morphology and are also not rich in derivation. Most words consist of a single morpheme.
  - **Agglutinative** languages like Turkish or Telugu have many affixes and can stack them one after another like beads on a string
  - **Fusional (or flexional)** languages like Spanish or German pack many inflectional meanings into single affixes, so that they are morphologically rich without "stacking" prefixes or suffixes
  - **Templatic** languages like Arabic or Amharic are a special kind of **fusional** languages that perform much of their morphological work by changes internal to the root.

Isolating Languages: Chinese

(1) 里斯 对 这 个 案件 的 调查
Lisi dui zhei ge anjian de diaocha
Lisi to this CLF case DE investigation
进行 了 一 个 小时
jinxing le yi ge xiaoshi
last ASP one CLF hour
'Lisi's investiagation of the case lasted an hour.'

Agglutinative Languages: Turkish



Fusional Languages: German

	Present	Perfect	Preterit
1SG	mache	gemacht	machte
2SG	machst	gemacht	machtest
3SG	macht	gemacht	machte
1PL	machen	gemacht	machten
2PL	macht	gemacht	machtet
3PL	machen	gemacht	machten

Templatic Languages: Arabic

	Perfect		Imperfect		Participle	
	Active	Passive	Active	Passive	Active	Passive
I	katab	kutib	ktub	ktab	kaatib	ktuub
II	kattab	kuttib	kattib	kattab	kattib	kattab
III	kaatab	kuutib	kaatib	kaatab	kaatib	kaatab
IV	ʔaktab	ʔuktib	ktib	ktab	ktib	ktab
V	takattab	tukuttib	takattab	takattab	takattib	takattab
VI	takaatab	tukuutib	takaatab	takaatab	takaatib	takaatab
VII	nkatab	nkutib	nkatib	nkatab	nkatib	nkatab
VIII	ktatab	ktutib	ktatib	ktatab	ktatib	ktatab
IX	ktab(a)b	ktab(i)b	ktab(i)b			
X	staktab	stuktib	staktib	staktab	staktib	staktab

Relevance of Morphological Typology to NLP

- If all languages were isolating, morphology would be solved
- Given a modest amount of data, a fairly naive ML model can learn simple morphological patterns without any human intervention
- However, **most languages are not isolating** and for most of these languages there is not enough data to "learn your way out" of the morphology problem by brute force
- Agglutinative, fusional, and templatic languages all present unique challenges for NLP tasks
- One exiting area of NLP research is extending existing NLP techniques to morphologically rich languages

The Problem and Promise of Morphology

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

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 (possibly using morphemic form as intermediate representation)
- In morphological generation, map from lexical form to orthographic form (possibly using the morphemic form as intermediate representation)
- **Older (finite state) methodology**: use intermediate representation
- **Newer (neural) methodology**: typically go "end-to-end"

Finite State Technologies

- FSAs and FSTs are old technology, but they are still widely used in industry
- They should up in some contemporary research, but are sometimes considered to be “niche”
- We teach you about them here because they are valuable tools for understanding morphology and morphological analysis as an NLP problem.

Morphological Analysis and Generation: How?

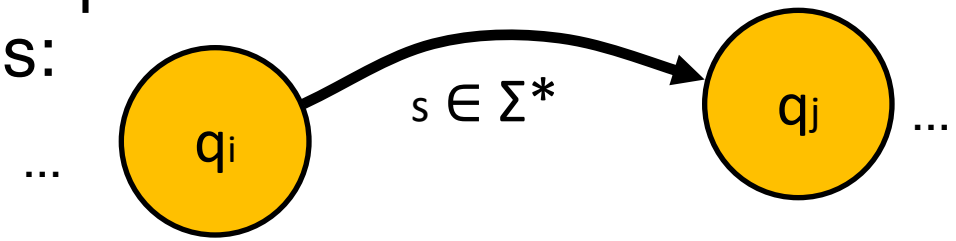
- Finite-state transducers (FSTs)
  - Define **regular relations** between strings
  - “foxes”$\Re$“fox +V +3p +Sg +Pres”
  - “foxes”$\Re$“fox +N +Pl”
  - Widely used in practice, not just for morphological analysis and generation, but also in speech applications, surface syntactic parsing, etc.
  - Once compiled, if *deterministic*, **run in linear time** (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!”d Example of an FSA

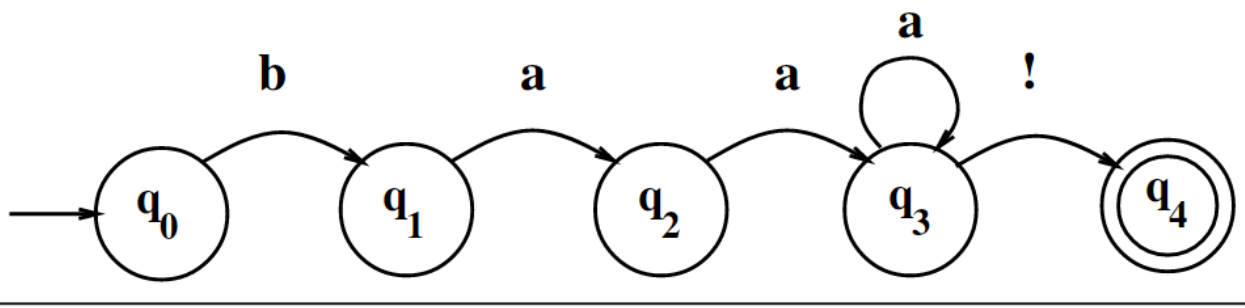


Figure 2.10 A finite-state automaton for talking sheep.

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 FSAs 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 known, 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

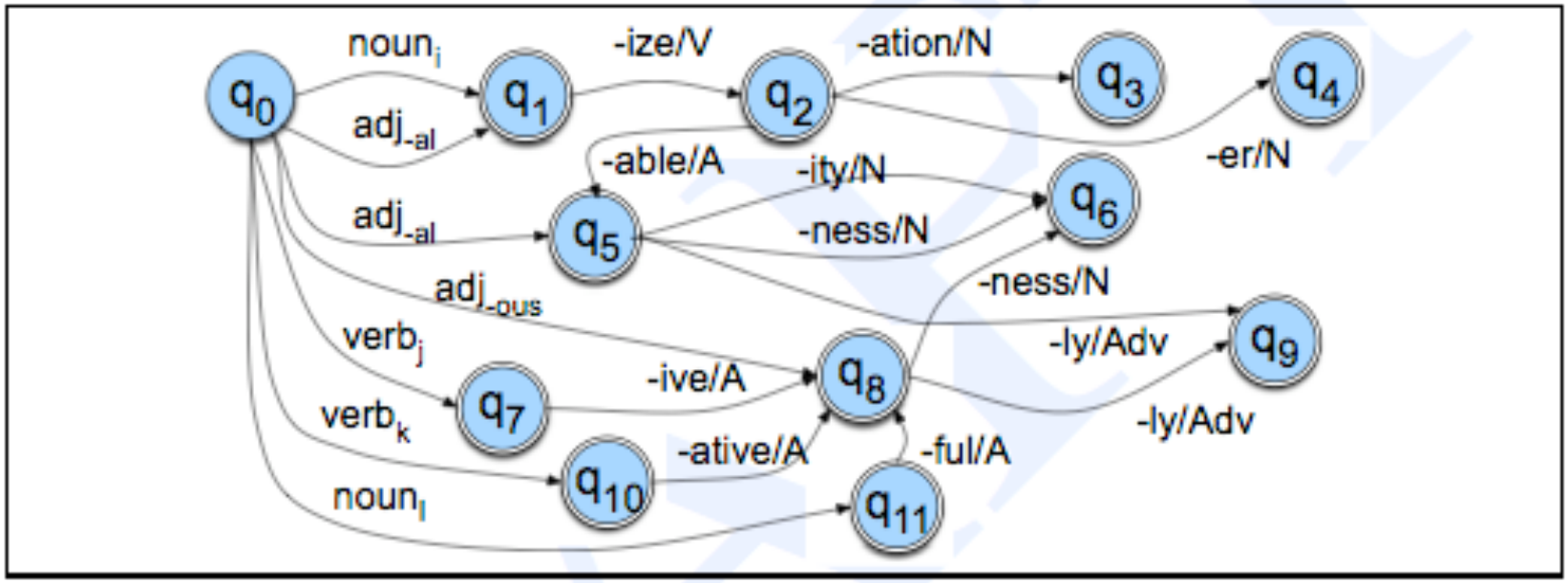


Figure 3.6 An FSA for another fragment of English derivational morphology.

Finite State Transducers

I am no longer accepting the things I cannot change.

Morphological Parsing/Analysis

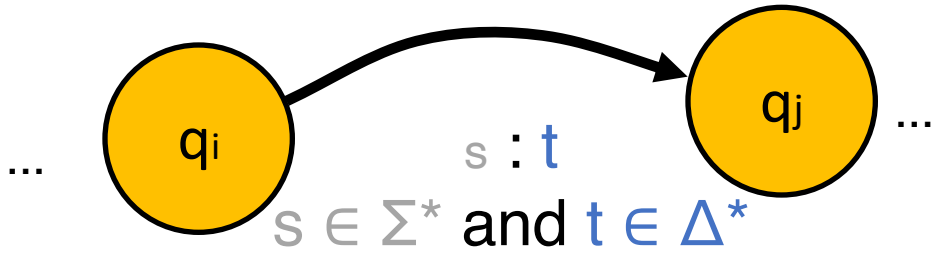
Input: a word

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

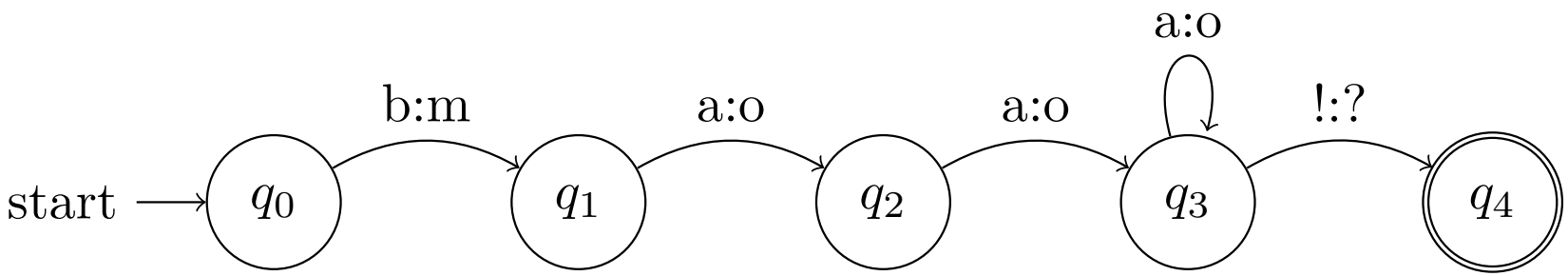
Example: geese → {goose +N +Pl}
gooses → {goose +V +3P +Sg}
dog → {dog +N +Sg, dog +V}
leaves → {leaf +N +Pl, leave +V +3P +Sg}

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:



Translating from Assertive Sheep to Quizzical Cow



Turkish Example

uygarlaştıramadıklarımızdanmışsınızcasına

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

uygar	"civilized"
+*laş*	"become"
+*tır*	"cause to"
+*ama*	"not able"
+*dık*	past participle
+*lar*	plural
+*ımız*	first person plural possessive ("our")
+*dan*	second person plural ("y'all")
+*mış*	past
+*sınız*	ablative case ("from/among")
+*casına*	finite verb → adverb ("as if")

Morphological Parsing with FSTs

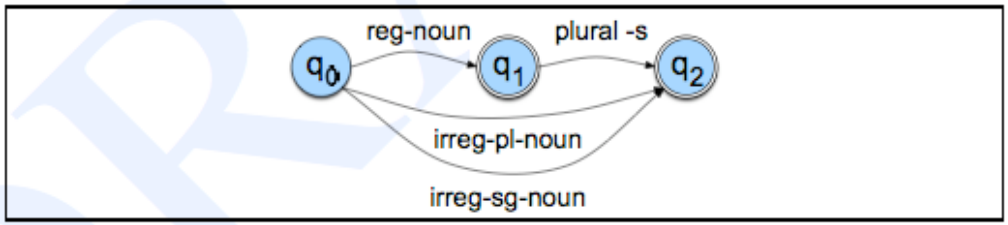


Figure 3.3 A finite-state automaton for English nominal inflection.

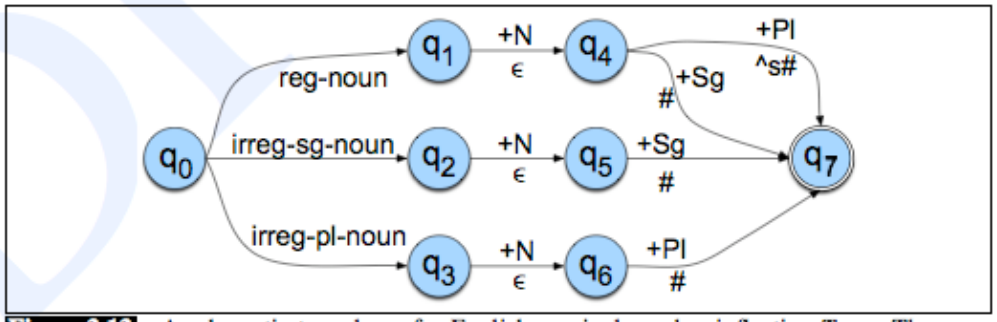


Figure 3.13 A schematic transducer for English nominal number inflection T_{num}. The symbols above each arc represent elements of the morphological parse in the lexical tape; the symbols below each arc represent the surface tape (or the intermediate tape, to be described later), using the morpheme-boundary symbol ^ 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.

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

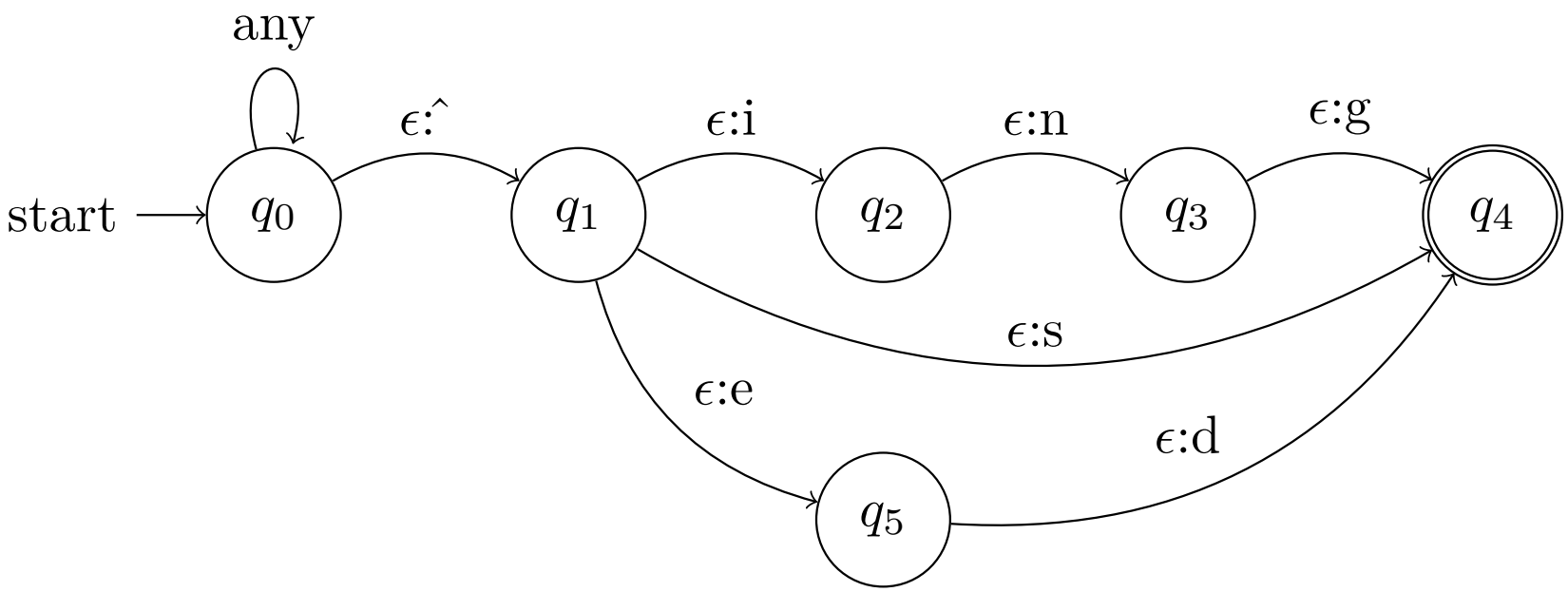
reg-noun	irreg-pl-noun	irreg-sg-noun
fox	g o:e o:e s e	goose
cat	sheep	sheep
aardvark	m o:i u:ϵ s:c e	mouse

- Note "same symbol" shorthand.
- ^ denotes a morpheme boundary.
- # denotes a word boundary.
- ^ and # are not there automatically—they must be inserted.

Separation of concerns

- Typically, a finite state 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 morphtactics (intermediate or morphemic representation) and surface representation
  - “zoch^s#” <-> “zoches"

Generating Inflected forms of English Verbs from Lemmas



English Spelling (Orthographic Rules)

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

The E Insertion Rule as a FST

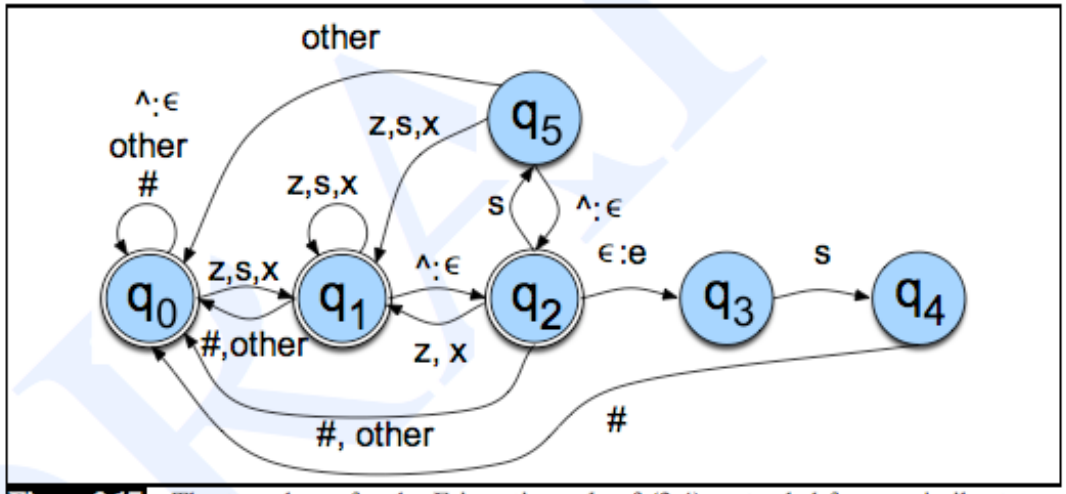


Figure 3.17 The transducer for the E-insertion rule of (3.4), extended from a similar transducer in Antworth (1990). We additionally need to delete the # symbol from the surface string; this can be done either by interpreting the symbol # as the pair #:ϵ, or by postprocessing the output to remove word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

$$\varepsilon \rightarrow \mathrm{e} / \left\{ \begin{array}{c} \mathrm{s} \\ \mathrm{x} \\ \mathrm{z} \end{array} \right\} \hat{} \_\_ \ \mathrm{s}\#$$

Combining FSTs

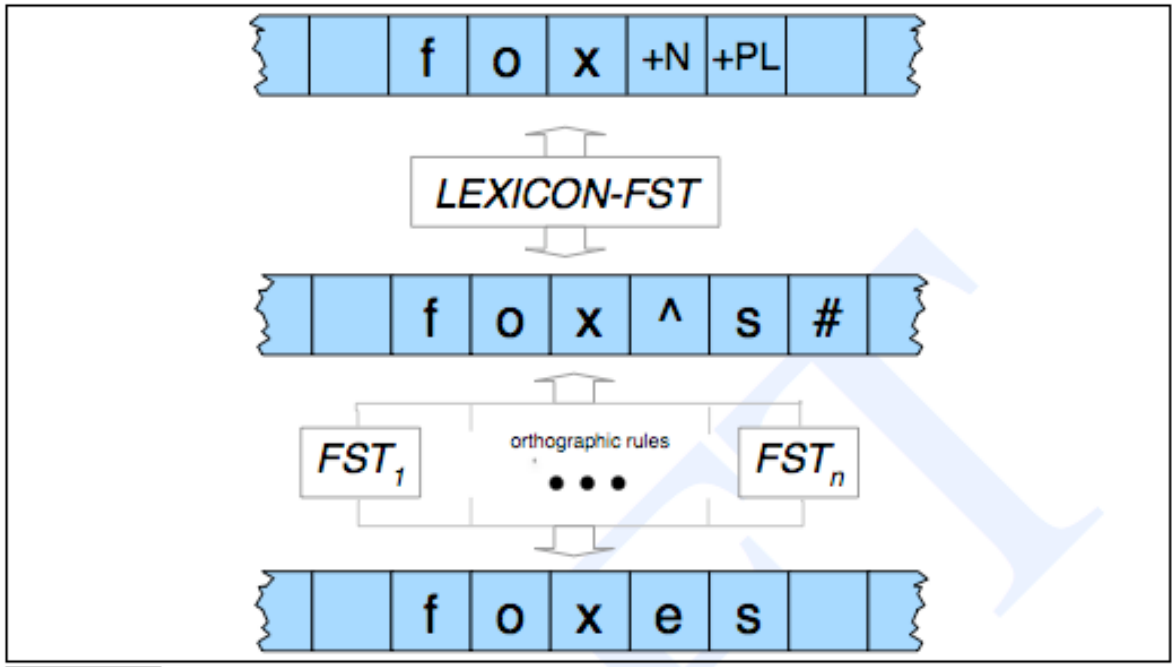


Figure 3.19 Generating or parsing with FST lexicon and rules

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 $y[S]z$; *e*ffectively 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)

$Q \leftarrow I_1 \times I_2$
$\mathcal{K} \leftarrow I_1 \times I_2$
while $\mathcal{K} \neq \emptyset$ **do**
  $q = (q_1, q_2) \leftarrow Head(\mathcal{K})$
  $Dequeue(\mathcal{K})$
  if $q \in I_1 \times I_2$ **then**
    $I \leftarrow I \cup \{q\}$
    $\lambda(q) \leftarrow \lambda_1(q_1) \otimes \lambda_2(q_2)$
  if $q \in F_1 \times F_2$ **then**
    $F \leftarrow F \cup \{q\}$
    $p(q) \leftarrow p_1(q_1) \otimes p_2(q_2)$
  for each $(e_1, e_2) \in E[q_1] \times E[q_2]$ such that $o[e_1] = i[e_2]$ **do**
    if $(q') = (n[e_1], n[e_2] \notin Q)$ **then**
      $Q \leftarrow Q \cup \{q'\}$
      $Enqueue(\mathcal{K}, q')$
    $E \leftarrow E \uplus \{(q, i[e_1], o[e_2], w[e_1] \otimes w[e_2], q')\}$
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=#states, D_i=max out-degree, M_i=max multiplicity of i^{th} WFST. *www.OpenFST.org*

FST Operations

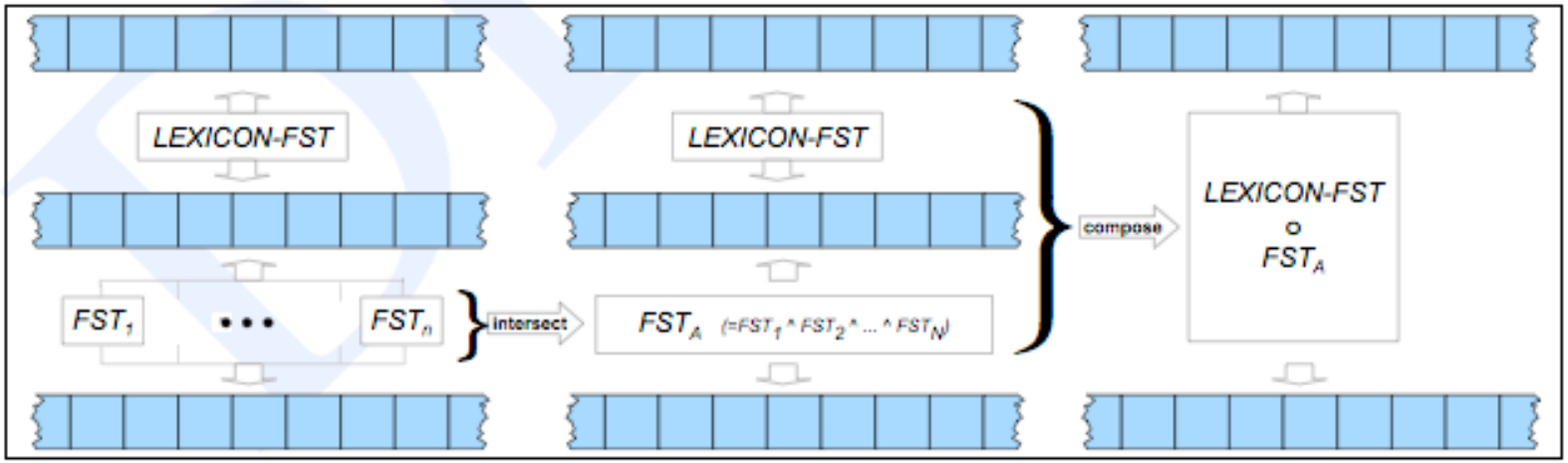


Figure 3.21 Intersection and composition of transducers.

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
- There is a lot of current research on sequence-to-sequence (seq2seq, encoder-decoder) models of morphological analysis and generation
- These have many of the advantages of FSTs, but do not require hand-engineering rules
- They do require a significant amount of labeled data for training, though, and they are harder to debug than FSTs

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
noces	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?