Carnegie Mellon University
Language Technologies Institute

11-411/11-611 Natural Language Processing

Lemmatization

David R. Mortensen

September 13, 2022

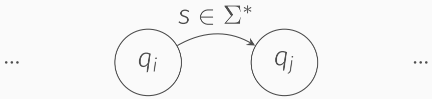
Language Technologies Institute

Introduction to Finite State Machines

A Finite State Automaton Is a 5-Tuple

$\langle Q, Q_0, F, \Sigma, \delta \rangle$

- A finite set of states Q
- A special start state $q_0 \in Q$
- A set of final states $F \subseteq Q$
- A finite alphabet Σ
- A state transition function δ: $Q \times \Sigma \rightarrow \mathcal{P}(Q)$ [$\mathcal{P}$ represents the POWERSET]


$$\begin{tikzcd} \dots & q_i \arrow[r, bend left, "s \in \Sigma^*"] & q_j & \dots \end{tikzcd}$$

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

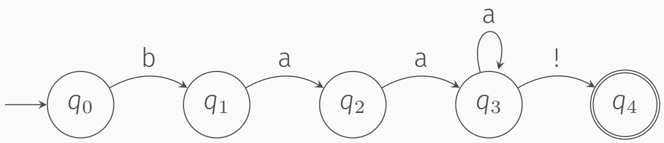
An FSA for the Sheep Language baa*!

Recognizes:

- baa!
- baaa!
- baaaa!

Rejects:

- ba
- ba!
- baaa


$$\begin{tikzcd} \arrow[r] & q_0 \arrow[r, bend left, "b"] & q_1 \arrow[r, bend left, "a"] & q_2 \arrow[r, bend left, "a"] & q_3 \arrow[loop above, "a"] \arrow[r, bend left, "!"] & q_4 \end{tikzcd}$$

A Metaphor for a Finite-State Automaton

- You have a “tape” with a string of symbols written on it
- You move through the FSA by following transitions with the next symbol on the tape
- When you follow such a transition, the symbol is “consumed”
- If consuming all of the symbols coincides with being at an accepting state, you win! (The FSA accepts the string).
- Otherwise, you lose! (The FSA rejects the string).

The Empty String Is Useful

Sometimes, it’s useful to have a transition that consumes no symbols. Such a transition can be labeled with ε (the empty string).

$$\varepsilon == ""$$

The empty string matches any string, anywhere. Consuming it consumes nothing.

Recognizing Morphology with Finite State Machines

Even in English, words can have complex morphotactic structure

Consider the derivational morphology of nouns:

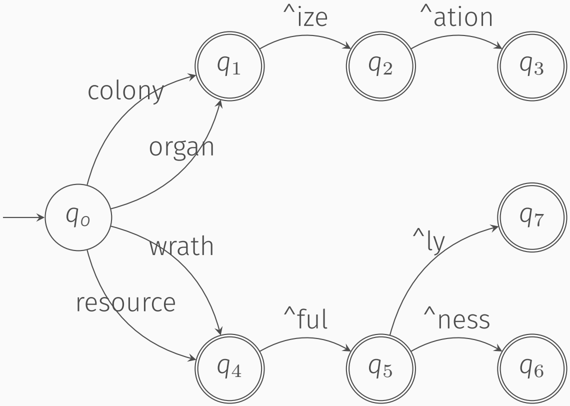
We have:

- colony
- colony^ize
- colony^ize^ation
- resource
- resource^ful
- resource^ful^ly
- resource^ful^ness

But not:

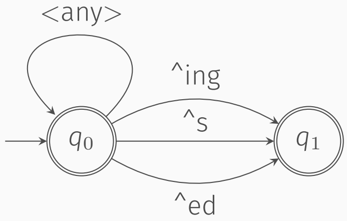
- resource^ly
- colony^ation
- colony^ful
- colony^ness
- resource^ize

Morphotactic Patterns Can Be Expressed as an FSA


$$\begin{tikzcd}
 & q_1 \arrow[r, bend left, "\text{\^{}ize}"] & q_2 \arrow[r, bend left, "\text{\^{}ation}"] & q_3 \\
\arrow[r] & q_o \arrow[u, bend left, "\text{colony}"] \arrow[u, bend right, "\text{organ}"'] \arrow[d, bend left, "\text{wrath}"] \arrow[d, bend right, "\text{resource}"'] & & q_7 \\
 & q_4 \arrow[r, bend left, "\text{\^{}ful}"] & q_5 \arrow[ur, bend left, "\text{\^{}ly}"] \arrow[r, bend left, "\text{\^{}ness}"] & q_6
\end{tikzcd}$$

Out of Vocabulary Words

- Some roots/stems cannot be known in advance
- Consider the word *squigging*
- Stem guessing can be used to “guess” previously unknown stems
- Loops labeled with all symbols in the alphabet
- Results in ambiguity/non-determinism



Introduction to Finite State Transducers

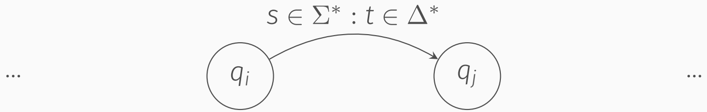
Finite State Transducers (FSTs) are Finite State Machines with Two Tapes

- FSTs have an input tape and an output tape
- Labels on transitions have two “sides”
    - Symbols from one are read from the input tape
    - Symbols from the other are written to the output tape

A Finite State Transducer Is a 7-Tuple

$\langle Q, q_0, F, \Sigma, \Gamma, \delta, \omega \rangle$

- A finite set of states Q
- A special start state $q_0 \in Q$
- A set of final states $F \subseteq Q$
- A finite input alphabet Σ
- A finite output alphabet Γ
- A state transition function δ: $Q \times \Sigma \cup \{\epsilon\} \rightarrow \mathcal{P}(Q)$
- An output function $\gamma : Q \times \Sigma \cup \{\epsilon\} \times Q \rightarrow \Gamma^*$



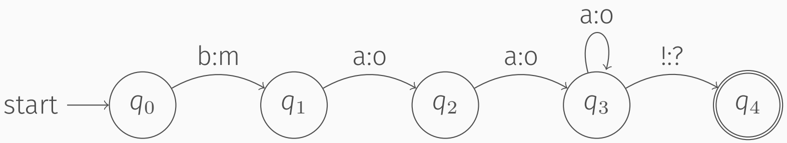
FSTs Encode a Mapping between Input Strings and Output Strings

- The mapping encoded by FSTs can be many-to-many
- An FST is **deterministic** iff for each state q there is at most one transition labeled with a given label. A transducer can be input deterministic (or subsequential) or output deterministic.
- A transducer is **functional** if each input string is transduced to a unique output string. There may be multiple paths, however, that contain this input and output string pair.

A Sheep-to-Cow Translator

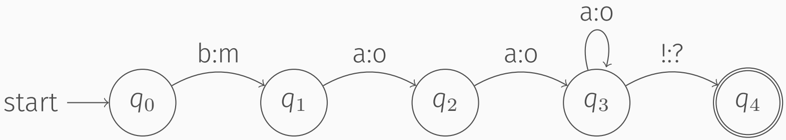
baa! → moo?
baaa! → mooo?
baaaa! → mooo?
ba! → (undefined)

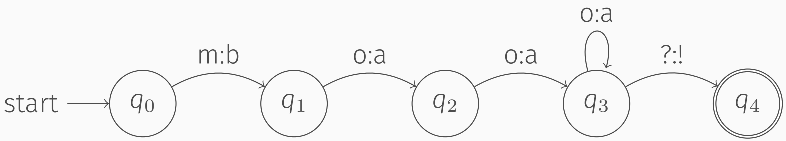
baa → (undefined)
baa? → (undefined)
moo? → (undefined)



FSTs Are Invertable

FSTs are invertable. Inversion switches the input and output symbols/strings on each transition. When inverted, this FST maps from “moo?” to “baa!” and from “moooo?” to “baaaa!”





FSTs Are Composable (Closed under Composition)

- When two FSTs are composed, they behave as if the output of the first was fed as input to the second.
- Compose “baa!” → “moo?” FST with an FST that capitalizes letters.
- “baaaaa!” → “MOOOOO?”

FSTs Are Closed Under Union

- FSTs represent sets of pairs (mappings).
- Taking the union of two FSTs gives you the union of these mappings
- Take the union of a “baa!” → “moo?” FST and a “moo?” → “baa!” FST (like those we discussed earlier).
- This new FST will map “baaaa!” → “moooo?” and “mooo?” → “baaa!”

Weighted Finite State Transducers

- FSTs are a special case of wFSTs (Weighted Finite State Transducers)
- wFSTs have a weight associated with each transition
- This means that, for the same input, there can be multiple outputs distinguished by weight
- These weights must be from a SEMIRING

Semirings are Mathemtical Objects from Abstract Algebra

A semiring is a set $\mathbb{K}$, two operators $\bigoplus$ and $\bigotimes$, and two elements $\bar{0}$ and $\bar{1}$:

- $\bigoplus$, an associative, commutative operator with $\bar{0}$ as its identity element.
  - $a \bigoplus (b \bigoplus c) = (a \bigoplus b) \bigoplus c$ (commutative)
  - $a \bigoplus b = b \bigoplus a$ (associative)
  - $a \bigoplus \bar{0} = \bar{0} \bigoplus a = a$
- $\bigotimes$, an associative operator with $\bar{1}$ as its identity element and $\bar{0}$ as its annihilator
  - $a \bigotimes (b \bigotimes c) = (a \bigotimes b) \bigotimes c$ (associative)
  - $a \bigotimes \bar{1} = \bar{1} \bigotimes a = a$
  - $a \bigotimes \bar{0} = \bar{0} \bigotimes a = \bar{0}$
- $\bigotimes$ distributes over $\bigoplus$:
  - $a \bigotimes (b \bigoplus c) = (a \bigotimes b) \bigoplus (a \bigotimes c)$
  - $(a \bigoplus b) \bigotimes c = (a \bigotimes c) \bigoplus (b \bigotimes c)$

Some Common Semirings

Here are some examples of semirings (adapted from Mehryar Mohri)

Semiring	Set	$\bigoplus$	$\bigotimes$	$\bar{0}$	$\bar{1}$
Boolean	$\{false, true\}$	$\vee$	$\wedge$	$false$	$true$
Probability	$\mathbb{R}_+$	$+$	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\bigoplus_{log}$	$+$	$+\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	$+$	$+\infty$	0

with $\bigoplus_{log}$ defined by: $x \bigoplus_{log} y = -log(e^{-x} + e^{-y})$

The Operators Correspond to Computations over wFSTs

sum	to compute the weight of a sequence (sum of the weights of the paths labeled with that sequence)
multiply	to compute the weight of a path (product of the weights of the constituent transitions)

These operations allow us to compute the weight of an input-output pair and thus “rank” outputs. Suppose you have to following paths (input symbol:output symbol/weight):

1. a:b/1 $\rightarrow$ b:c/1 $\rightarrow$ c:d/1
2. a:b/1 $\rightarrow$ b:ϵ/2 $\rightarrow$ c:d/1
3. ϵ:b/3 $\rightarrow$ a:ϵ/1 $\rightarrow$ b:c/1 $\rightarrow$ c:d/1

Then, in the tropical semiring, $path_1 = 1 \bigotimes 1 \bigotimes 1 = 3$, $path_2 = 1 \bigotimes 2 \bigotimes 1 = 4$, and $path_3 = 3 \bigotimes 1 \bigotimes 1 \bigotimes 1 \bigotimes 1 = 6$. The weight for abc:bcd ($path_1$ and $path_3$) is $3 \bigoplus 6 = 3$

Preferences Can Be Encoded through Weights

Sometimes you want to allow multiple paths through a wFST, but you want to favor paths with specific properties.

- In a lemmatization task, you might want to *allow* outputs in which an orthographic rule does not apply, but *favor* outputs in which it does
- The weight/cost for the first condition can be made higher than the weight/cost for the second
- Normally, these weights would be learned empirically
- In Homework 2, you will engineer them manually

Applying Finite State Transducers to Allomorphy

Morphological Analysis and Generation Mean Mapping Between Lexical Form and Surface Form

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

Spelling Rules Are a Kind of Allomorphy

- Spelling rules (a kind of allomorphic rules) can be easily represented and implemented with FSTs
- Because you can use inversion, you can map either from the surface form to the morphemic form or vis versa, then invert if you want to transduce in the other direction

There Are Several Spelling Rules in English

Name	Description	Examples
consonant doubling	single consonants (one letter) doubled before -ing and -ed	beg/begging pad/padded trap/trapping
e-deletion	silent e dropped before -ing and -ed	confide/confiding save/saved
e-insertion	e added after -s, -z, -x, -ch, and -sh when before -s	fizz/fizzes fish/fishes
y-replacement	-y is replaced by -ie before -s and by -i before -ed	try/tries/tried bury/buries/buried
k-insertion	verbs ending with a vowel + -c add -k before -ing and -ed.	panic/panicked picnic/picnicking

Labels on Transitions for Different Processes

Input:Output	Description
y:i	substitute ”i” for ”y”
e:ε	delete ”e”
ε:e	insert ”e”

Implementing Lemmatization with Finite State Transducers

The Lemmatization Task

Lemmatization

- **Input** A word(form), e.g., *feet*, *cacti*, or *flurries*
- **Output** The lemma (the "dictionary form" of a word), e.g., `foot`, `cactus`, or `flurry`

In Morphological Analysis and Generation, the Top Level Is the Lexical Form

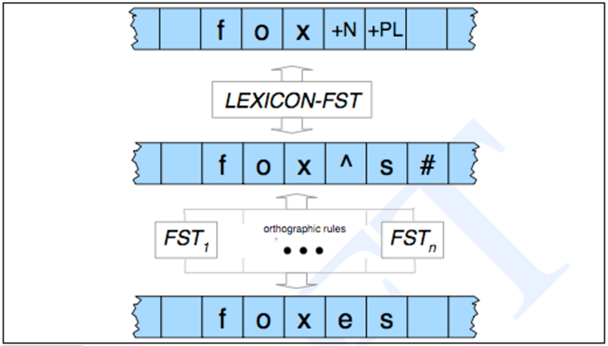


Figure 3.19 Generating or parsing with FST lexicon and rules

In Lemmatization, the Top Level Is the Lemma

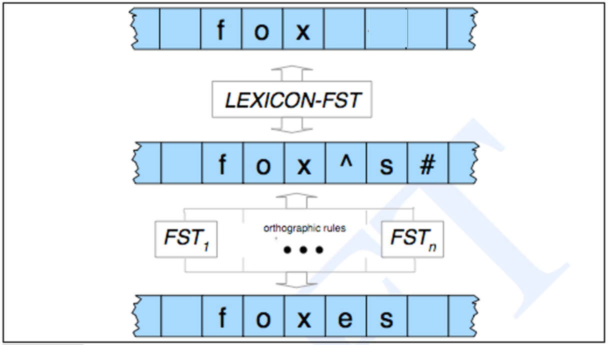
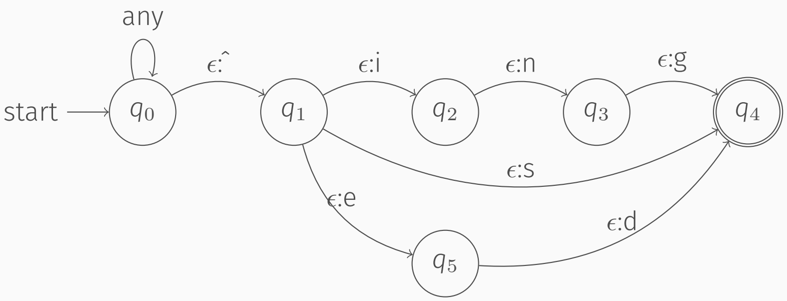


Figure 3.19 Generating or parsing with FST lexicon and rules

Simplified Morphotactic FST with Guesser


$$\begin{tikzcd}
\text{start} \arrow[r] & q_0 \arrow[loop above, "\text{any}"] \arrow[r, bend left, "\epsilon:\hat{}"] & q_1 \arrow[r, bend left, "\epsilon:i"] \arrow[rrr, bend right, "\epsilon:s"] \arrow[dr, bend right, "\epsilon:e"] & q_2 \arrow[r, bend left, "\epsilon:n"] & q_3 \arrow[r, bend left, "\epsilon:g"] & q_4 \\
 & & & q_5 \arrow[urr, bend right, "\epsilon:d"] & &
\end{tikzcd}$$

Building a Lemmatizer Can Be Seen as a Five-Step Process

- Write and compile a morphotactic FST (with guesser and lexicon) that generates inflected forms from lemmas
- Write and compile FSTs for allomorphic rules (spelling rules) that change morphemic forms into surface forms
- Combine allomorphic FSTs (using composition, intersection, etc.) to form allomorphic FST
- Compose morphotactic FST with allomorphic FST
- Invert resulting FST

Combine the Spelling Rules, then Compose It with the Morphotactic FST (Lexicon FST)

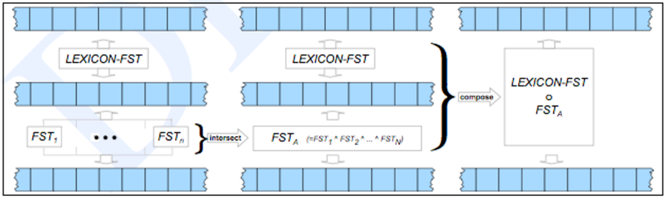


Figure 3.21 Intersection and composition of transducers.

Stemming (Lemmatization on the Cheap)

Stemming Is a Poor Man's Morphology

- Stemming—reducing wordform to substrings such that each inflected form of a word yields the same string
- Good for many quick-and-dirty NLP and information retrieval applications
- Reasonably good stemmers in a variety of languages are available at `https://snowballstem.org/`

Stemming Is for Applications Where the “Lemma” Doesn’t Have to Be a Real Word/Morpheme

WORD	PORTER STEMMER
no	no
noah	noah
nob	nob
nobility	nobil
nobis	nobi
noble	nobl
nobleman	nobleman
noblemen	noblemen
nobleness	nobl
nobler	nobl
nobles	nobl

WORD	PORTER STEMMER
noblest	noblest
nobly	nobli
nobody	nobodi
noces	noce
nod	nod
nodded	nod
nodding	nod
noddle	noddl
noddles	noddl
noddy	noddi
nods	nod

Stemmers Can Be Seen as a Cascade of Composed FSTs

- Stemmers operate as a sequence of deterministic string rewrite rules
- Typically modify the ends of words
- Like regex replacements
- These can be implemented as a composed (cascaded) sequence of FSTs

Stemming Is Hard to Get Right

WORD	PORTER STEMMER
no	no
noah	noah
nob	nob
nobility	nobil
nobis	nobi
noble	nobl
nobleman	nobleman
noblemen	noblemen
nobleness	nobl
nobler	nobl
nobles	nobl

noble, *nobler*, *nobleness*, and *nobility* are all different lexemes, but *noble*, *nobler*, and *nobleness* all get represented as `nobl` whereas *nobility* gets represented as *nobil*

Existing Stemmers Are Available for Many Languages

Snowball Stemmers

- English
- French
- Spanish
- Catalan
- Portuguese
- Italian
- Romanian
- German
- Dutch
- Swedish
- Norwegian
- Danish
- Russian
- Finnish
- Basque

WP Morphological Analysis and Generation

Word-and-Paradigm Morphology Puts Words Front and Center

Suppose we viewed the relationship between words not in terms of shared substrings but in terms of graphs or tables.

Word-and-Paradigm morphology, which is **central** to current computational approaches to inflection, does just that.

$$\text{RUN: \{PRESENT, SINGULAR, 3RD\}} \leftrightarrow \textit{runs}$$

Each label is the header of a table row or column or edges in a graph.

Applying Word-and-Paradigm Morphology to NLP

- WP can be used for both analysis and generation
- Analysis, using encoder-decoder model
    - Encode words as real valued vectors
    - Decode them as a lemma and a sequence of features
- Generation, using encoder-decoder model
    - Encode lemma and feature sequence as real valued vectors
    - Decode them as words

Tools and Resources for Morphological Analysis and Generation

Interest in NLP Tools for Morphology Is Growing

- As interest in non-English NLP grows, more tools are resources for morphology become available
    - Data resources (Unimorph, UD)
    - Stemmers
    - Analyzers

UniMorph Is a Growing Resource for Morphological Tasks

Unimorph is a collection of lexicons from 141 languages annotated according to a specific morphological schema

- Structure: each wordform on a line, words separated by blank lines
- On each line: stem, wordform, inflectional features (tab delimited)

Unimorph can be used to:

- Train neural morphological analyzers using LSTMs or Transformers
- Validate rule based (e.g. FST-based) morphological analyzers

Only useful for inflectional morphology; no derivation. Words are out of context.

For Word in Context (Syntactic and Otherwise) there is UD

- Universal Dependencies is a collection of dependency treebanks (will talk about in more detail during syntax module)
- Provides Unimorph-like analyses of inflectional morphology for each word in a corpus

There Are a Wide Variety of Morphological Analyzers

- There are many morphological analyzers for many languages
- Most of these are not in a standard collection or format
- However, Universal Dependency parsers like UDPipe and Stanza can provide analyses for some languages (those present in UD)
- These are freely available and relatively easy to use

Questions?