Text Analytics

Lecture 2b:
morphology & finite-state transducers

Klinton Bicknell

(borrowing from: Dan Jurafsky and Jim Martin)

Words

- Finite-state methods are particularly useful in dealing with a lexicon
- Many devices, most with limited memory, need access to large lists of words
- And they need to perform fairly sophisticated tasks with those lists
- So we'll first talk about some facts about words and then come back to computational methods

English Morphology

- Morphology is the study of the ways that words are built up from smaller meaningful units called morphemes
- We can usefully divide morphemes into two classes
  - Stems: The core meaning-bearing units
  - Affixes: Bits and pieces that adhere to stems to change their meanings and grammatical functions

English Morphology

- We can further divide morphology up into two broad classes
  - Inflectional
  - Derivational

Word Classes

- By word class, we have in mind familiar notions like noun and verb
- We'll go into the gory details in Chapter 5
- Right now we're concerned with word classes because the way that stems and affixes combine is based to a large degree on the word class of the stem

Inflectional Morphology

- Inflectional morphology concerns the combination of stems and affixes where the resulting word:
  - Has the same word class as the original
  - Serves a grammatical/semantic purpose that is
    - Different from the original
    - But is nevertheless transparently related to the original

Nouns and Verbs in English

- Nouns are simple
  - Markers for plural and possessive
- Verbs are only slightly more complex
  - Markers appropriate to the tense of the verb

Regulars and Irregulars

- It is a little complicated by the fact that some words misbehave (refuse to follow the rules)
  - Mouse/mice, goose/geese, ox/oxen
  - Go/went, fly/flew
- The terms regular and irregular are used to refer to words that follow the rules and those that don't

Regular and Irregular Verbs

- Regulars…
  - Walk, walks, walking, walked, walked
- Irregulars
  - Eat, eats, eating, ate, eaten
  - Catch, catches, catching, caught, caught
  - Cut, cuts, cutting, cut, cut

Inflectional Morphology

- So inflectional morphology in English is fairly straightforward
- But is complicated by the fact that are irregularities

Derivational Morphology

- Derivational morphology is the messy stuff that no one ever taught you.
  - Quasi-systematicity
  - Irregular meaning change
  - Changes of word class

Derivational Examples

- Verbs and Adjectives to Nouns

-ation	computerize	computerization
-ee	appoint	appointee
-er	kill	killer
-ness	fuzzy	fuzziness

Derivational Examples

- Nouns and Verbs to Adjectives

-al	computation	computational
-able	embrace	embraceable
-less	clue	clueless

Example: *Compute*

- Many paths are possible…
- Start with compute
  - Computer -> computerize -> computerization
  - Computer -> computerize -> computerizable
- But not all paths/operations are equally good (allowable?)
  - Clue
    - Clue -> *clueable

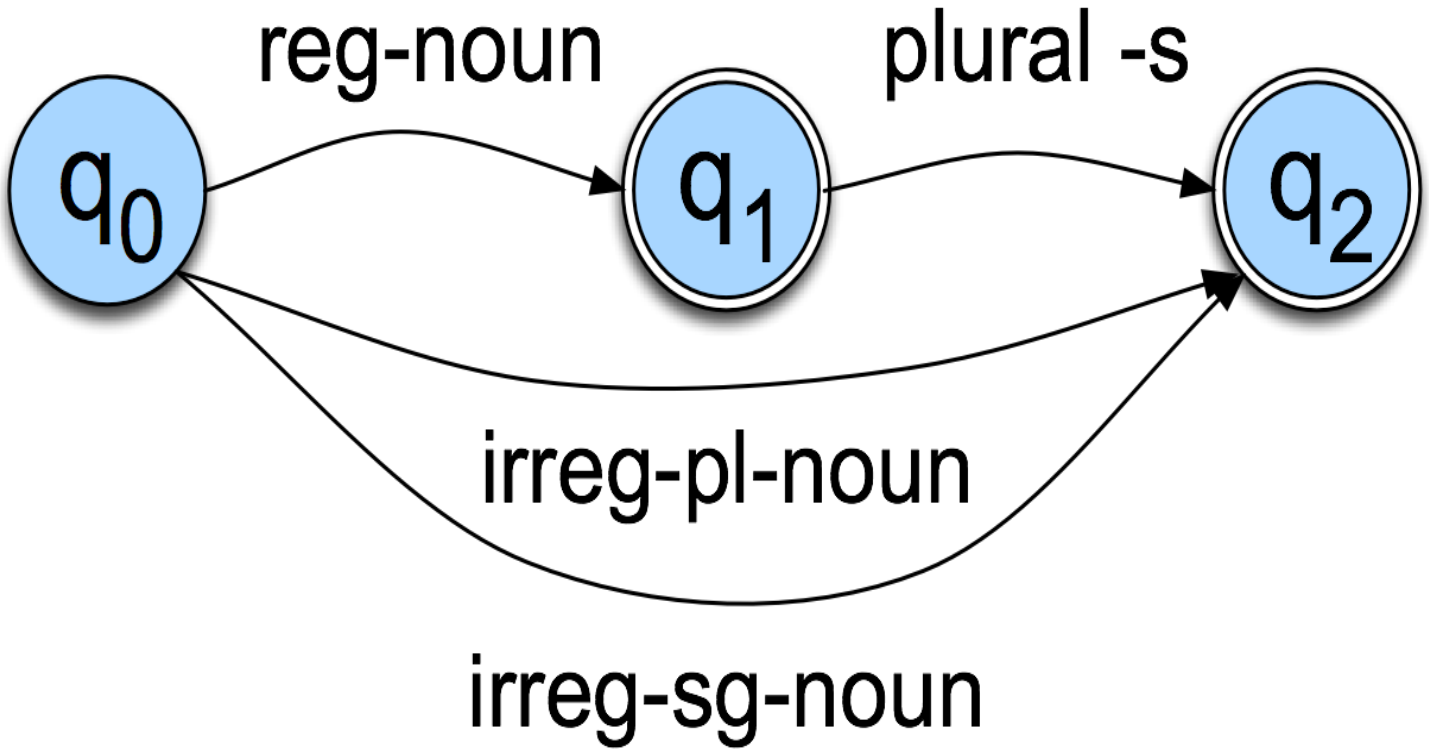
Morphology and FSAs

- We'd like to use the machinery provided by FSAs to capture these facts about morphology
  - Accept strings that are in the language
  - Reject strings that are not
  - And do so in a way that doesn't require us to in effect list all the words in the language

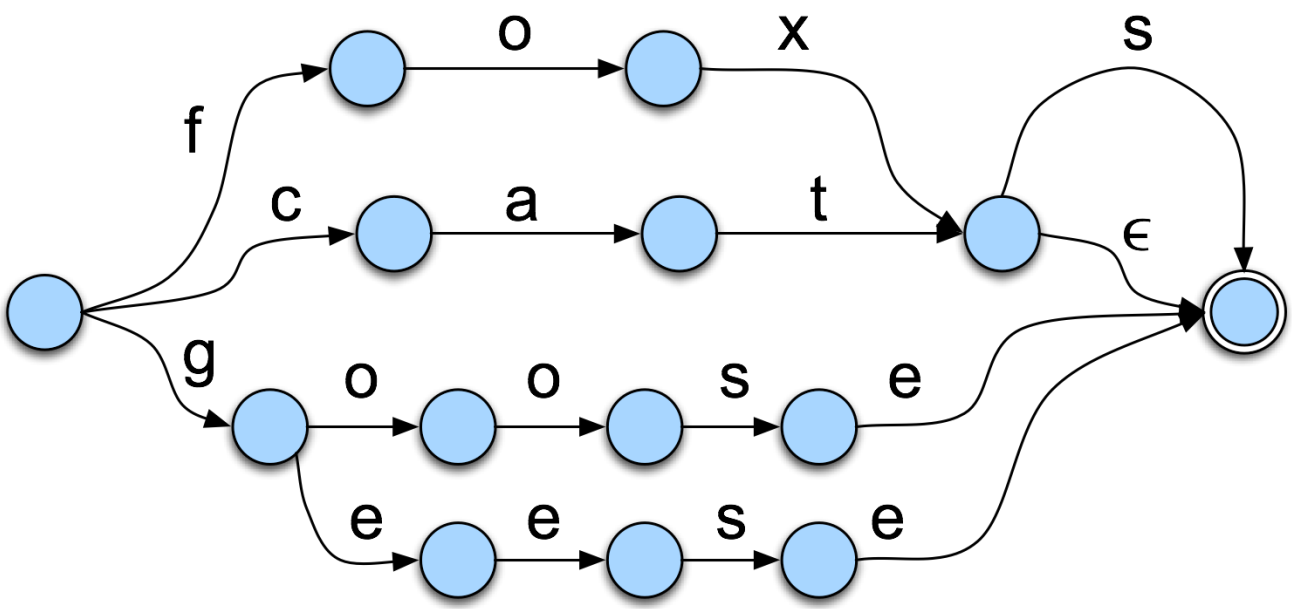
Start Simple

- Regular singular nouns are ok
- Regular plural nouns have an -s on the end
- Irregulars are ok as is

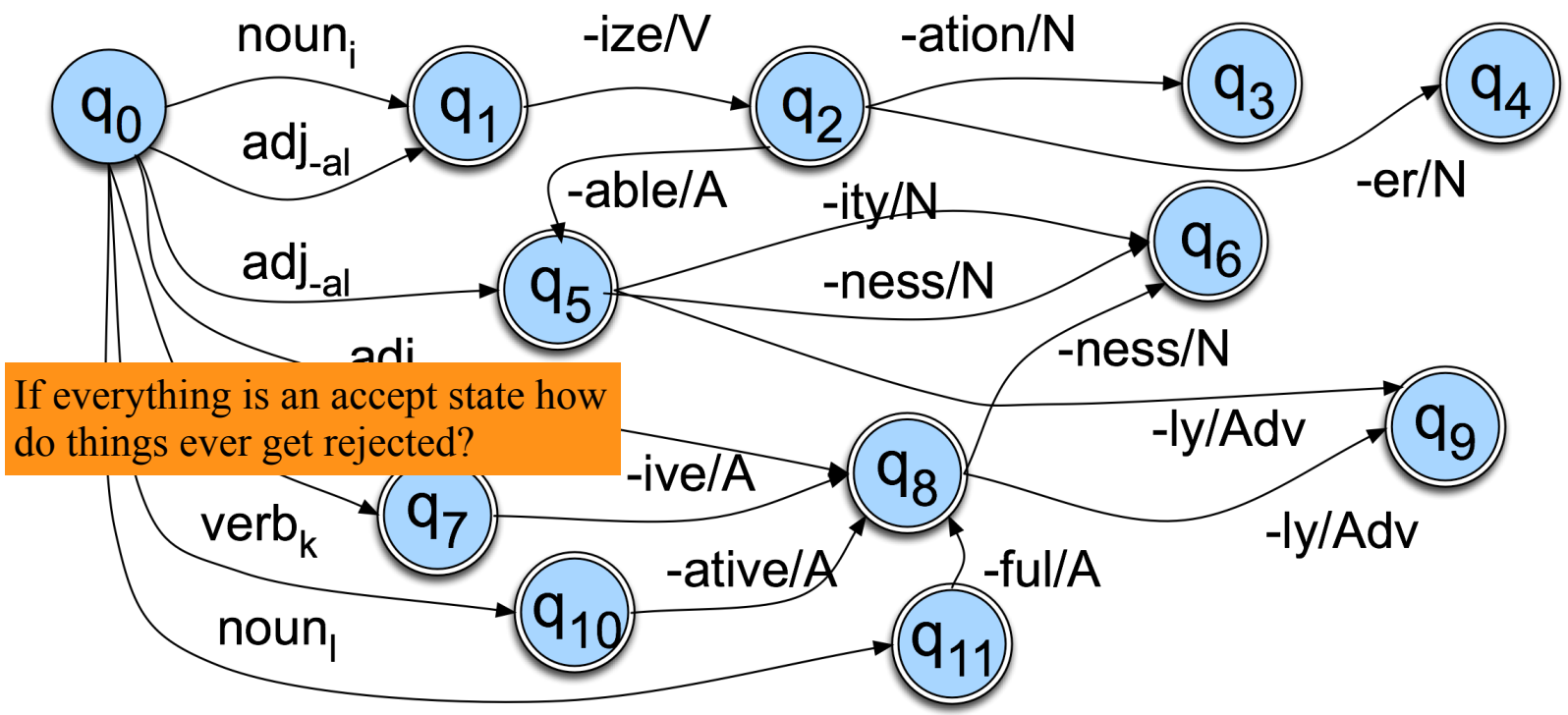
Simple Rules



Now Plug in the Words



Derivational Rules



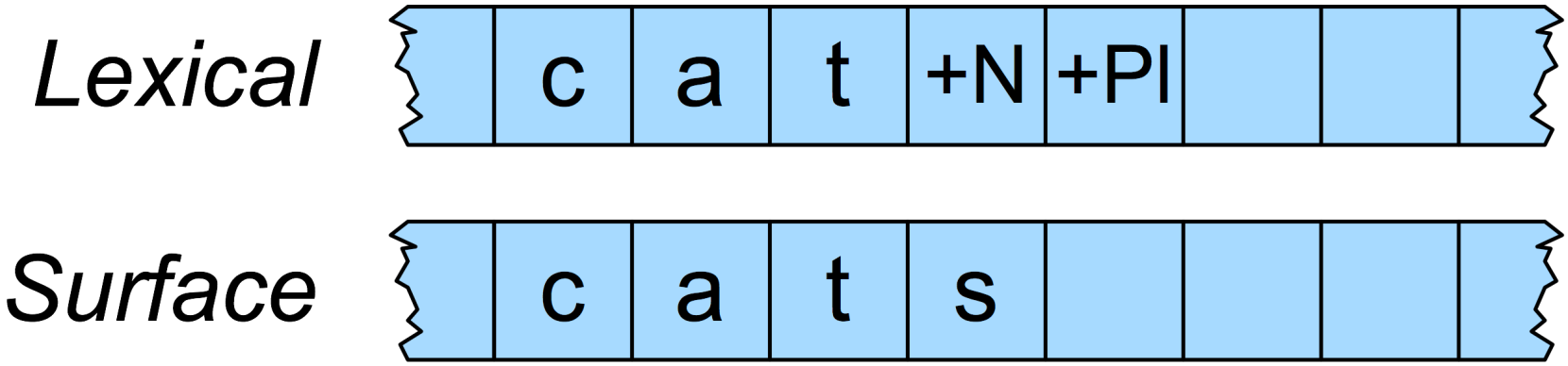
Parsing/Generation vs. Recognition

- We can now run strings through these machines to recognize strings in the language
- But recognition is usually not quite what we need
  - Often if we find some string in the language we might like to assign a structure to it (parsing)
  - Or we might have some structure and we want to produce a surface form for it (production/generation)
- Example
  - From "cats" to "cat +N +PL"

Finite State Transducers

- The simple story
  - Add another tape
  - Add extra symbols to the transitions
  - On one tape we read "cats", on the other we write "cat +N +PL"

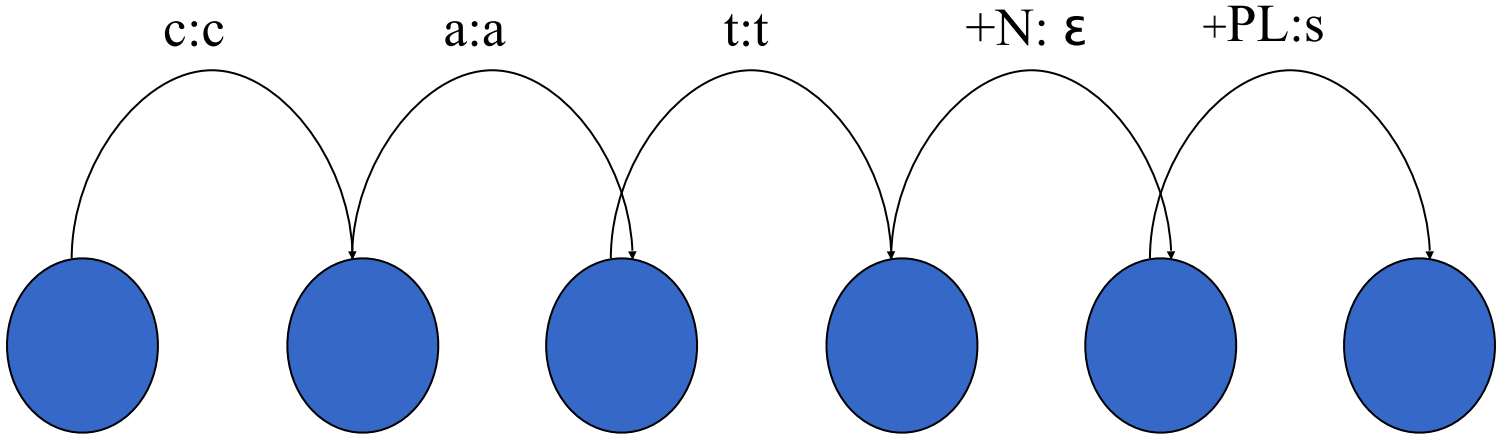
FSTs



Applications

- The kind of parsing we're talking about is normally called morphological analysis
- It can either be
  - An important stand-alone component of many applications (spelling correction, information retrieval)
  - Or simply a link in a chain of further linguistic analysis

Transitions



- c:c means read a c on one tape and write a c on the other
- +N:ε means read a +N symbol on one tape and write nothing on the other
- +PL:s means read +PL and write an s

Typical Uses

- Typically, we'll read from one tape using the first symbol on the machine transitions (just as in a simple FSA).
- And we'll write to the second tape using the other symbols on the transitions.

Ambiguity

- Recall that in non-deterministic recognition multiple paths through a machine may lead to an accept state.
  - Didn't matter which path was actually traversed
- In FSTs the path to an accept state does matter since different paths represent different parses and different outputs will result

Ambiguity

- What's the right parse (segmentation) for
  - Unionizable
  - Union-ize-able
  - Un-ion-ize-able
- Each represents a valid path through the derivational morphology machine.

Ambiguity

- There are a number of ways to deal with this problem
  - Simply take the first output found
  - Find all the possible outputs (all paths) and return them all (without choosing)
  - Bias the search so that only one or a few likely paths are explored

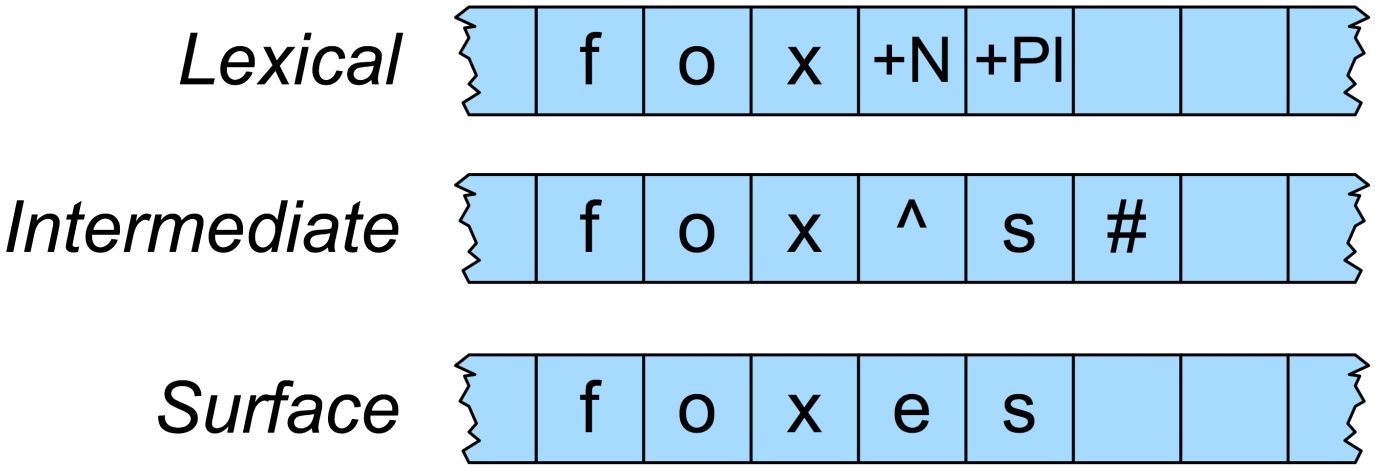
The Gory Details

- Of course, its not as easy as
  - “cat +N +PL” <-> “cats”
- As we saw earlier there are geese, mice and oxen
- But there are also a whole host of spelling/ pronunciation changes that go along with inflectional changes
  - Cats vs Dogs
  - Fox and Foxes

Multi-Tape Machines

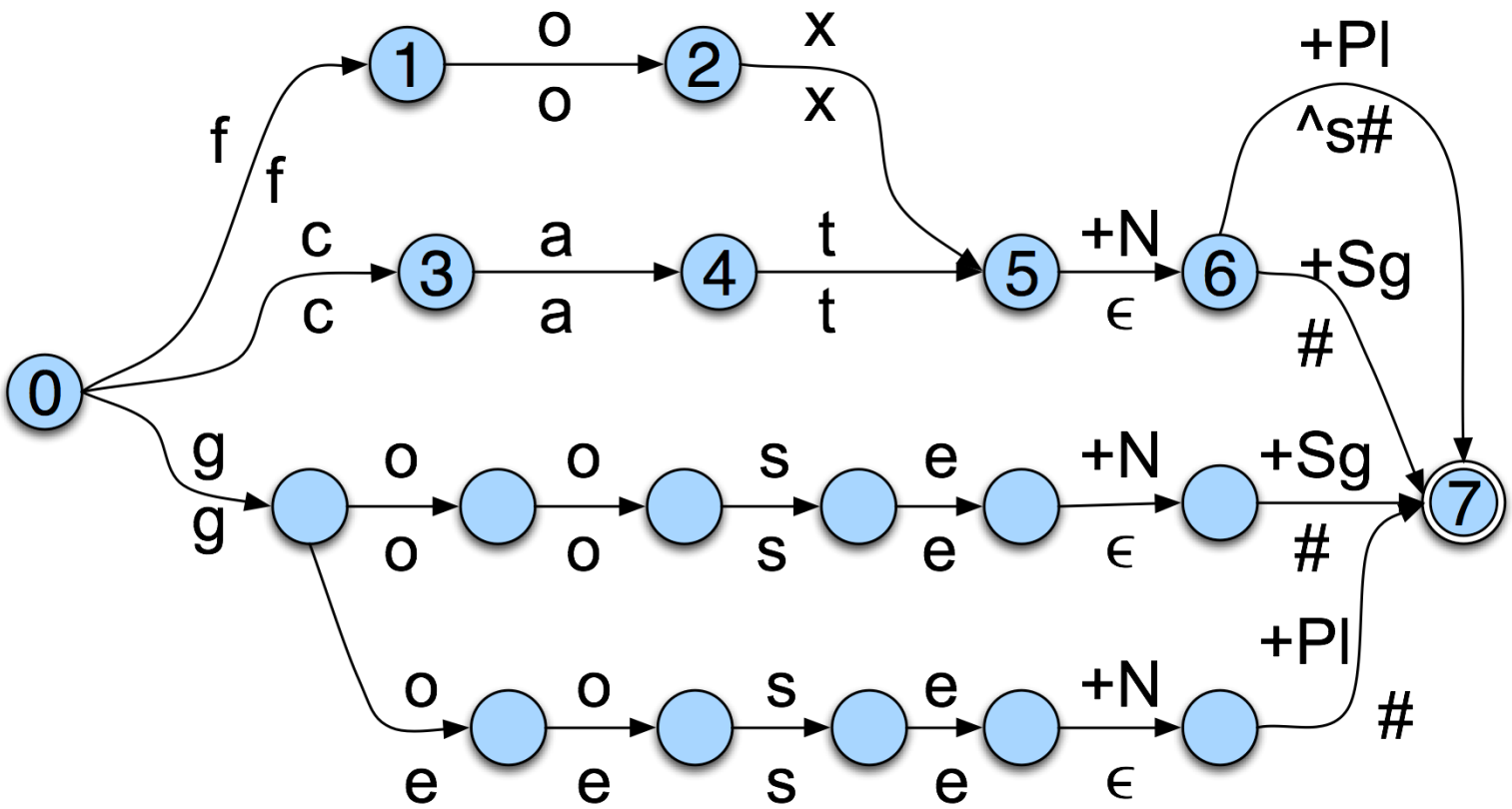
- To deal with these complications, we will add more tapes and use the output of one tape machine as the input to the next
- So to handle irregular spelling changes we'll add intermediate tapes with intermediate symbols

Multi-Level Tape Machines



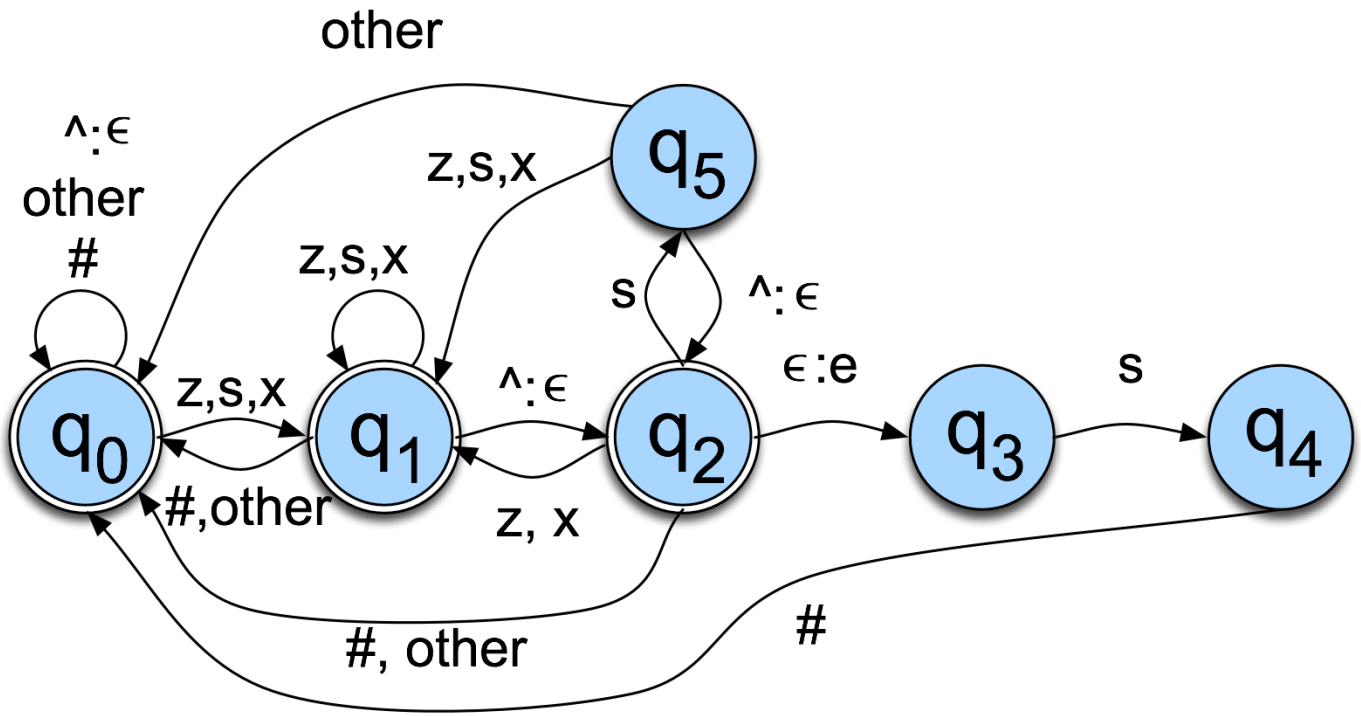
- We use one machine to transduce between the lexical and the intermediate level, and another to handle the spelling changes to the surface tape

Lexical to Intermediate Level

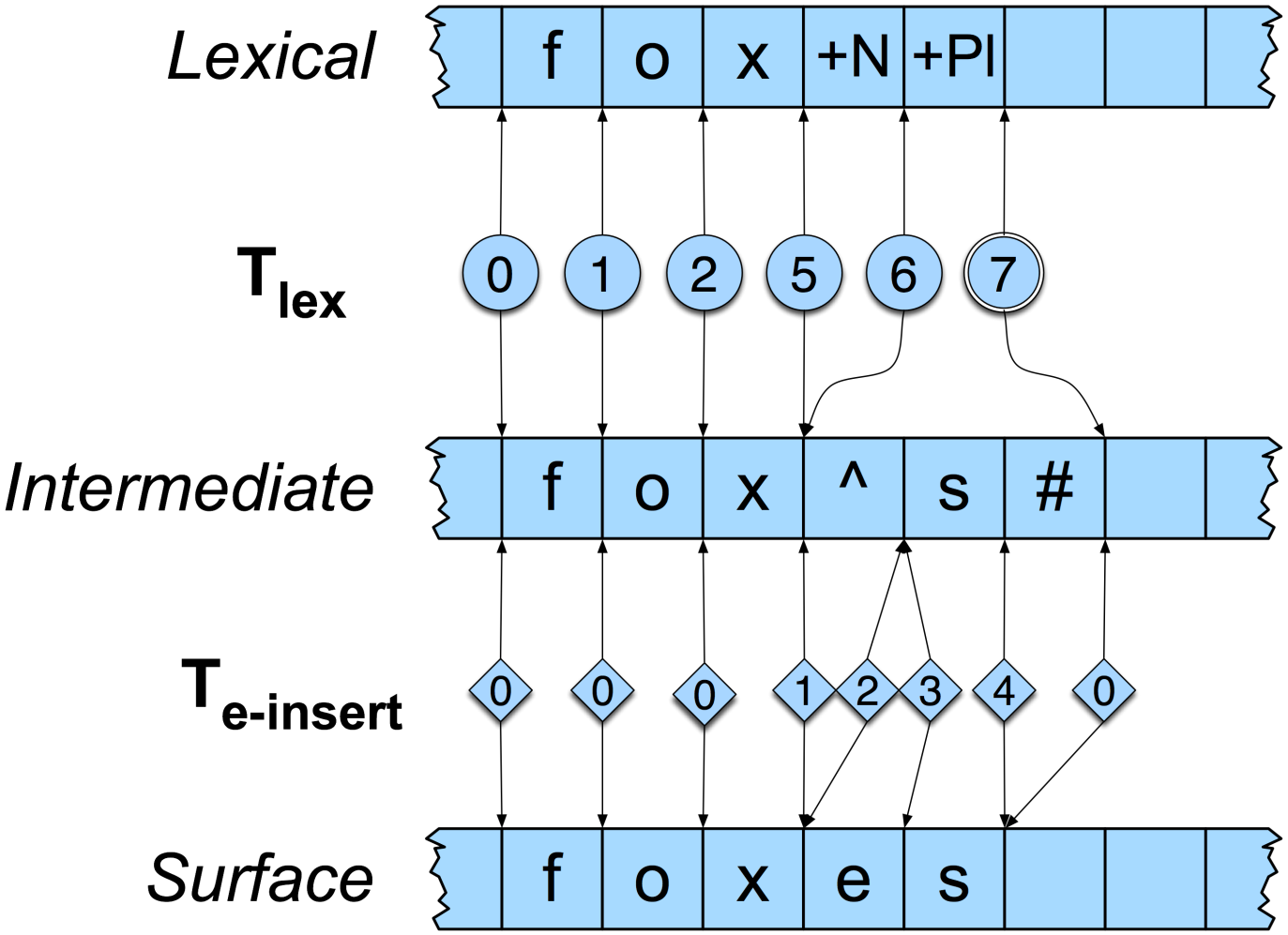


Intermediate to Surface

- The add an "e" rule as in fox^s# <-> foxes#



Foxes


Lexical
f o x +N +Pl
T lex
0 1 2 5 6 7
Intermediate
f o x ^ s #
T e-insert
0 0 0 1 2 3 4 0
Surface
f o x e s

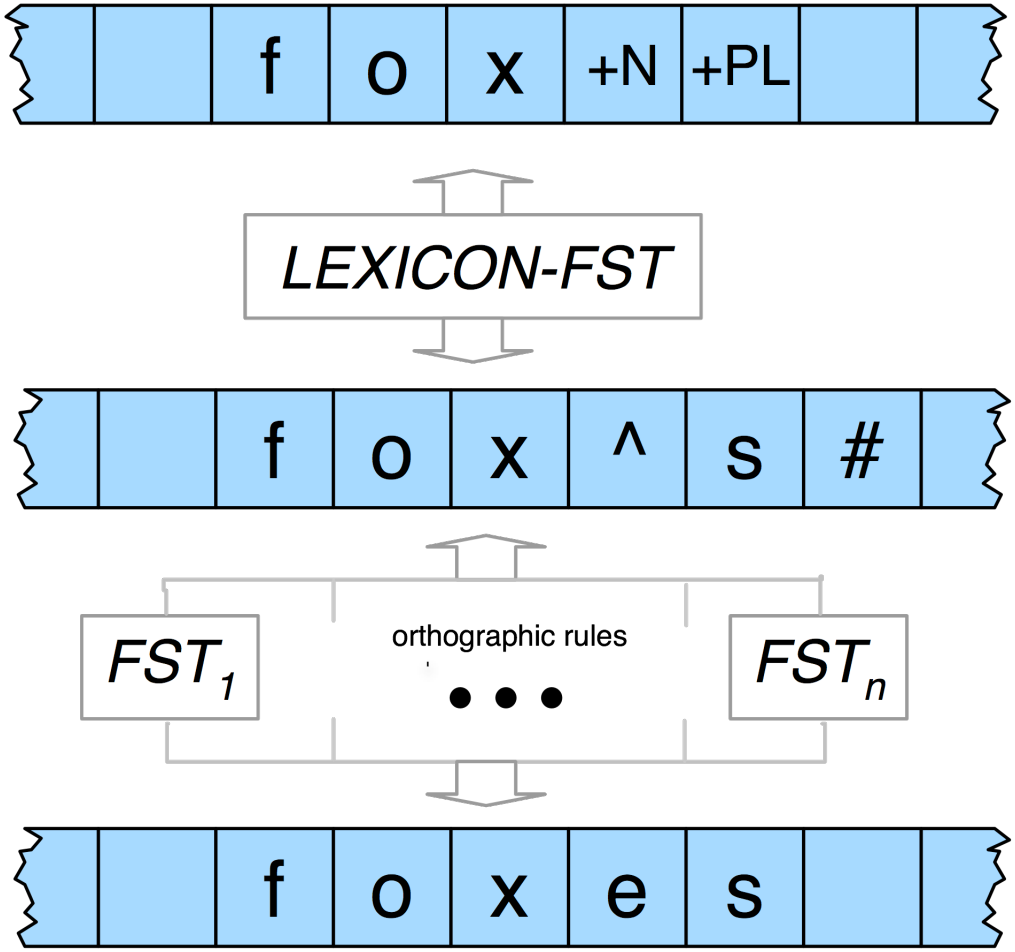
Note

- A key feature of this machine is that it doesn’t do anything to inputs to which it doesn’t apply.
- Meaning that they are written out unchanged to the output tape.

Overall Scheme

- We now have one FST that has explicit information about the lexicon (actual words, their spelling, facts about word classes and regularity).
  - Lexical level to intermediate forms
- We have a larger set of machines that capture orthographic/spelling rules.
  - Intermediate forms to surface forms

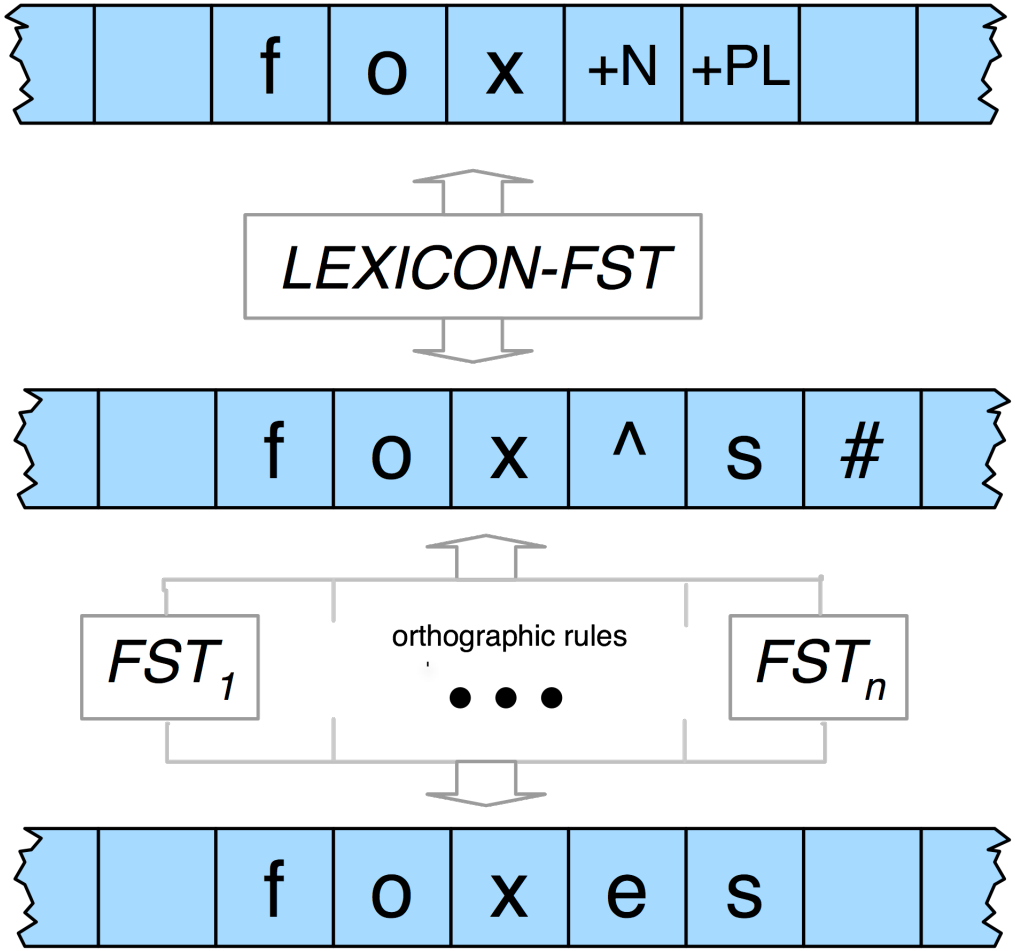
Overall Scheme



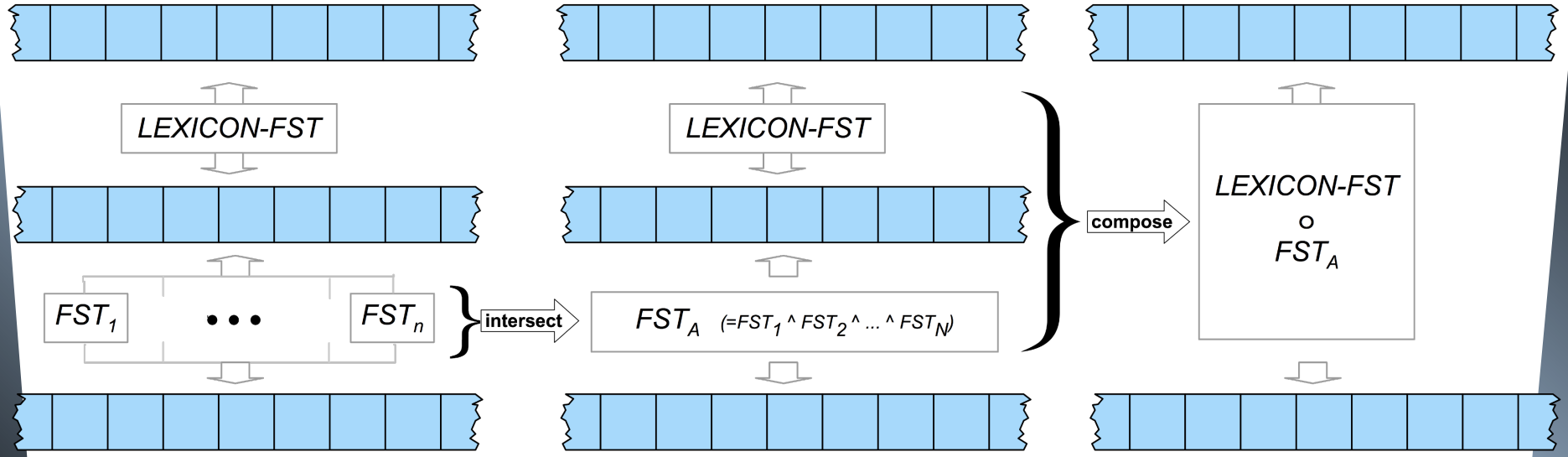
Cascades

- This is an architecture that we'll see again and again
  - Overall processing is divided up into distinct rewrite steps
  - The output of one layer serves as the input to the next
  - The intermediate tapes may or may not wind up being useful in their own right

Overall Plan



Final Scheme



Composition

1. Create a set of new states that correspond to each pair of states from the original machines (New states are called (x,y), where x is a state from M1, and y is a state from M2)
2. Create a new FST transition table for the new machine according to the following intuition…

Composition

- There should be a transition between two states in the new machine if it's the case that the output for a transition from a state from M1, is the same as the input to a transition from M2 or…

Composition

- $\delta_3((x_a,y_a), i:o) = (x_b,y_b)$ iff
  - There exists c such that
  - $\delta_1(x_a, i:c) = x_b$ AND
  - $\delta_2(y_a, c:o) = y_b$

CoreNLP: docs

- java: http://nlp.stanford.edu/software/corenlp.shtml
- python wrappers
  - https://github.com/brendano/stanford_corenlp_pywrapper
  - others listed at bottom of corenlp's java page
- tokenizer: http://nlp.stanford.edu/software/tokenizer.shtml