Morphology parsing

Informatics 2A: Lecture 14

Shay Cohen

School of Informatics
University of Edinburgh
scohen@inf.ed.ac.uk

23 October 2015

1. Morphology parsing: the problem
2. Finite-state transducers
3. Finite-state transducers
4. FSTs for morphology parsing and generation

(This lecture is based on Jurafsky & Martin chapter 3, sections 1–7.)

Morphology in other languages

Morphology is the study of the *structure of words*

English has relatively impoverished morphology.

Languages with rich morphology: Turkish, Arabic, Hungarian, Korean, and many more (actually, English is rather unique in having relatively simple morphology)

For example, Turkish is an *agglutinative* language, and words are constructed by concatenating *morphemes* together without changing them much:
evlerinizden: "from your houses" (morphemes: ev-ler-iniz-den)

This lecture will mostly discuss how to build an English morphological analyser.

Morphology

Stems and affixes (prefixes, suffixes, infixes and circumfixes) combine together

Four methods to combine them:

- Inflection (stem + grammar affix) - word does not change its grammatical class (*walk* $\rightarrow$ *walking*)
- Derivation (stem + grammar affix) - word changes its grammatical form (*computerize* $\rightarrow$ *computerization*)
- Compounding (stems together) - *doghouse*
- Cliticization - *I've, we're, he's*

Morphology can be concatenative or non-concatenative (e.g. templatic morphology as in Arabic)

Morphological parsing: the problem

English has concatenative morphology. Words can be made up of a main stem (carrying the basic dictionary meaning) plus one or more affixes carrying grammatical information. E.g.:

Surface form:	cats	walking	smoothest
Lexical form:	cat+N+PL	walk+V+PresPart	smooth+Adj+Sup

Morphological parsing is the problem of extracting the lexical form from the surface form. (For speech processing, this includes identifying the word boundaries.)

We should take account of:

- Irregular forms (e.g. goose $\rightarrow$ geese)
- Systematic rules (e.g. 'e' inserted before suffix 's' after s,x,z,ch,sh: fox $\rightarrow$ foxes, watch $\rightarrow$ watches)

Why bother?

- Any NLP tasks involving grammatical parsing will typically involve morphology parsing as a prerequisite.
- Search engines: e.g. a search for 'fox' should return documents containing 'foxes', and vice versa.
- Even a humble task like spell checking can benefit: e.g. is 'walking' a possible word form?

But why not just list all derived forms separately in our wordlist (e.g. walk, walks, walked, walking)?

- Might be OK for English, but not for a morphologically rich language — e.g. in Turkish, can pile up to 10 suffixes on a verb stem, leading to 40,000 possible forms for some verbs!
- Even for English, morphological parsing makes adding/learning new words easier.
- In speech processing, word breaks aren't known in advance.

How expressive is morphology?

Morphemes are tacked together in a rather "regular" way.

This means that finite-state machines are a good way to model morphology. There is no need for "unbounded memory" to model it (there are no long range dependencies).

This is as opposed to syntax, the study of the order of words in a sentence, which we will learn about in another lecture

Parsing and generation

Parsing here means going from the surface to the lexical form. E.g. foxes $\rightarrow$ fox +N +PL.

Generation is the opposite process: fox +N +PL $\rightarrow$ foxes. It's helpful to consider these two processes together.

Either way, it's often useful to proceed via an intermediate form, corresponding to an analysis in terms of morphemes (= minimal meaningful units) before orthographic rules are applied.

Surface form:	foxes
Intermediate form:	fox ^ s #
Lexical form:	fox +N +PL

(^ means morpheme boundary, # means word boundary.)

N.B. The translation between surface and intermediate form is exactly the same if 'foxes' is a 3rd person singular verb!

Finite-state transducers

We can consider ϵ-NFAs (over an alphabet Σ) in which transitions may also (optionally) produce *output* symbols (over a possibly different alphabet Π).

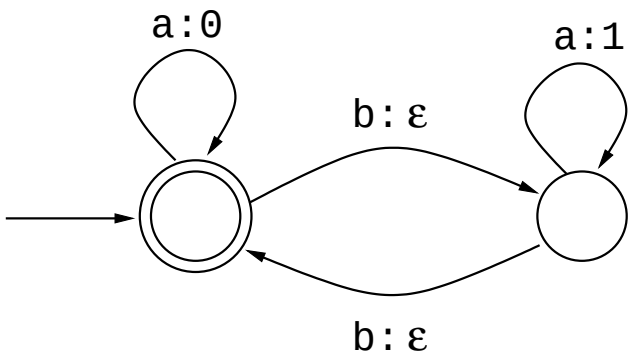
E.g. consider the following machine with input alphabet $\{a, b\}$ and output alphabet $\{0, 1\}$:

Such a thing is called a finite state transducer.
In effect, it specifies a (possibly multi-valued) translation from one regular language to another.

Finite-state transducers

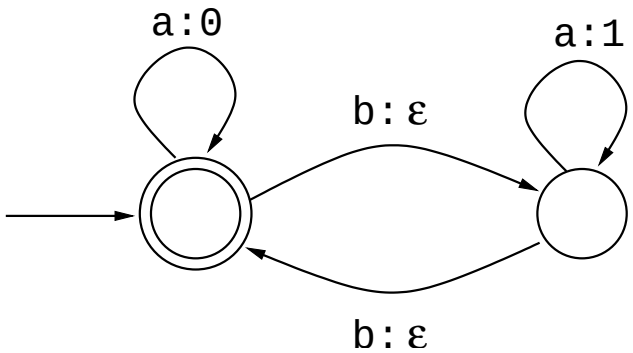
We can consider ϵ-NFAs (over an alphabet Σ) in which transitions may also (optionally) produce *output* symbols (over a possibly different alphabet Π).

E.g. consider the following machine with input alphabet $\{a, b\}$ and output alphabet $\{0, 1\}$:



Such a thing is called a finite state transducer.
In effect, it specifies a (possibly multi-valued) translation from one regular language to another.

Quick exercise



What output will this produce, given the input *aabaaabbbab*?

1. 001110
2. 001111
3. 0011101
4. More than one output is possible.

Formal definition

Formally, a finite state transducer T with inputs from Σ and outputs from Π consists of:

- sets Q, S, F as in ordinary NFAs,
- a transition relation $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Pi \cup \{\epsilon\}) \times Q$

From this, one can define a many-step transition relation $\hat{\Delta} \subseteq Q \times \Sigma^* \times \Pi^* \times Q$, where $(q, x, y, q') \in \hat{\Delta}$ means "starting from state q, the input string x can be translated into the output string y, ending up in state q'." (Details omitted.)

Note that a finite state transducer can be run in either direction! From T as above, we can obtain another transducer $\overline{T}$ just by swapping the roles of inputs and outputs.

Formally, a finite state transducer T with inputs from Σ and outputs from Π consists of:

- sets Q, S, F as in ordinary NFAs,
- a transition relation $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Pi \cup \{\epsilon\}) \times Q$

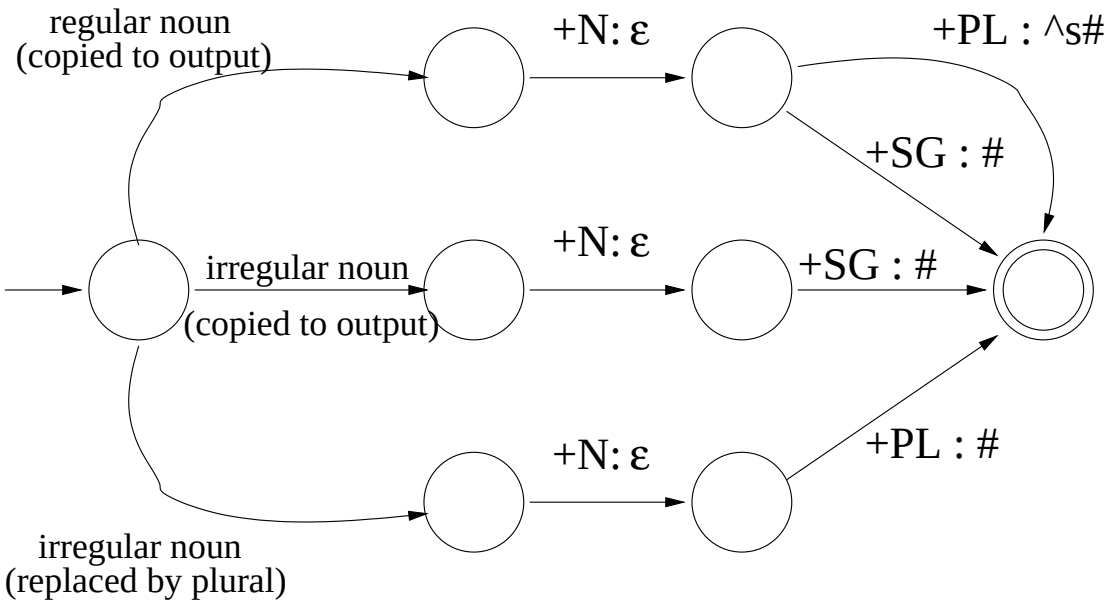
Reminder: Formally, an NFA with alphabet Σ consists of:

- A finite set Q of states.
- A transition relation $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$,
- A set $S \subseteq Q$ of possible starting states,
- A set $F \subseteq Q$ of accepting states.

Stage 1: From lexical to intermediate form

Consider the problem of translating a lexical form like 'fox+N+PL' into an intermediate form like 'fox ^ s # ', taking account of irregular forms like goose/geese.

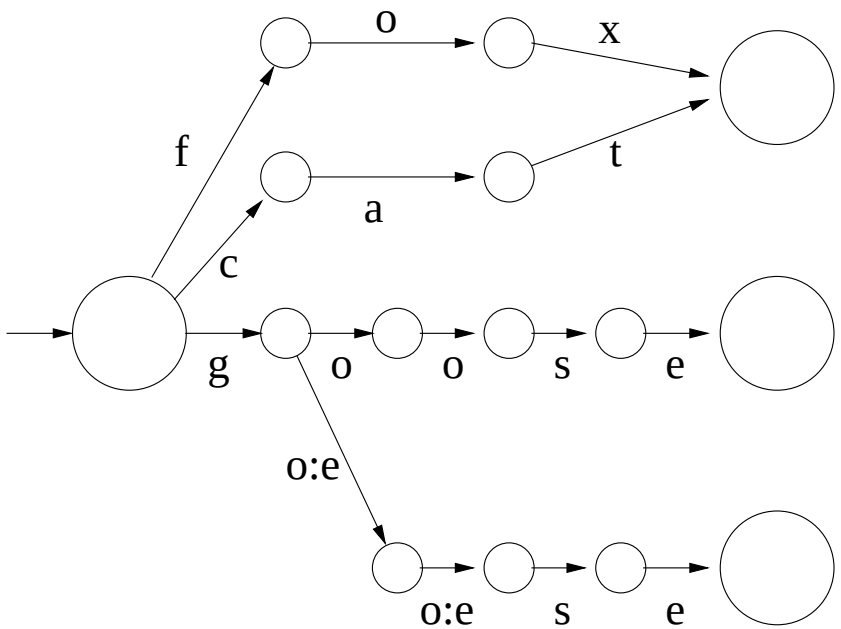
We can do this with a transducer of the following schematic form:



We treat each of +N, +SG, +PL as a single symbol.
The 'transition' labelled $+PL : \hat{}s\#$ abbreviates three transitions:
$+PL : \hat{}, \quad \epsilon : s, \quad \epsilon : \#$.

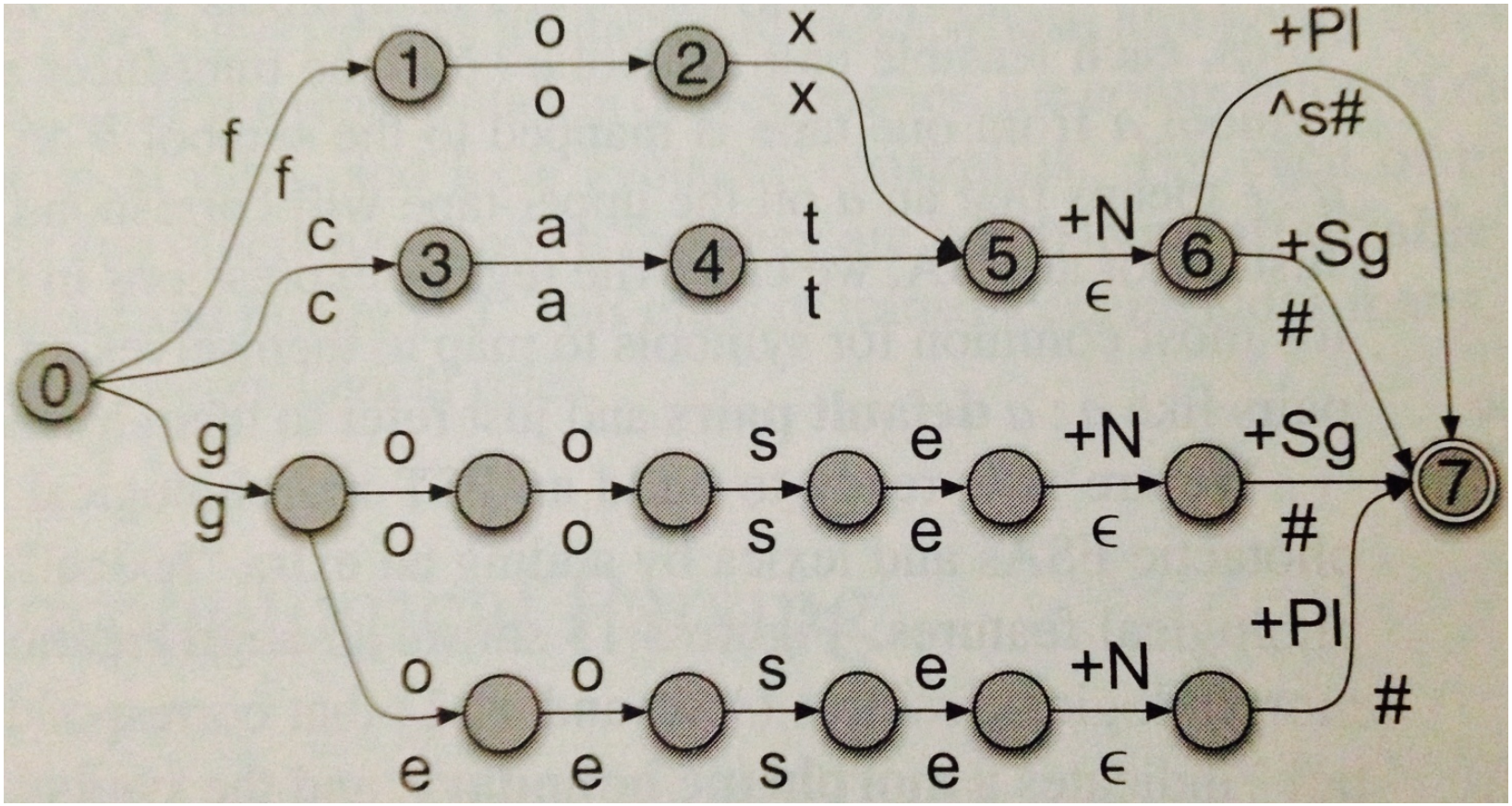
The Stage 1 transducer fleshed out

The left hand part of the preceding diagram is an abbreviation for something like this (only a small sample shown):



Here, for simplicity, a single label u abbreviates $u : u$.

Stage 1 in full



Stage 2: From intermediate to surface form

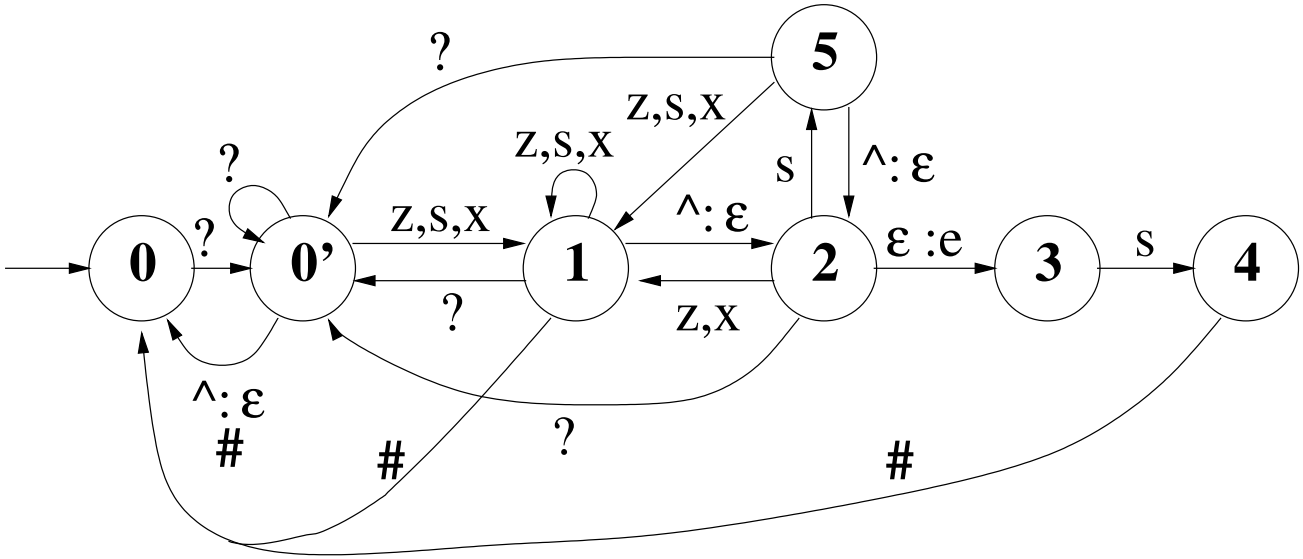
To convert a sequence of morphemes to surface form, we apply a number of orthographic rules such as the following.

- E-insertion: Insert e after s,z,x,ch,sh before a word-final morpheme -s. (fox $\rightarrow$ foxes)
- E-deletion: Delete e before a suffix beginning with e,i. (love $\rightarrow$ loving)
- Consonant doubling: Single consonants b,s,g,k,l,m,n,p,r,s,t,v are doubled before suffix -ed or -ing. (beg $\rightarrow$ begged)

We shall consider a simplified form of E-insertion, ignoring ch,sh.

(Note that this rule is oblivious to whether -s is a plural noun suffix or a 3rd person verb suffix.)

A transducer for E-insertion (adapted from J+M)



Here ? may stand for any symbol except z,s,x,^,#.
(Treat # as a 'visible space character'.)

At a morpheme boundary following z,s,x, we arrive in State 2. If the ensuing input sequence is s#, our only option is to go via states 3 and 4. Note that there's no #-transition out of State 5.

State 5 allows e.g. 'ex^service^men#' to be translated to 'exservicemen'.

Putting it all together

FSTs can be cascaded: output from one can be input to another.

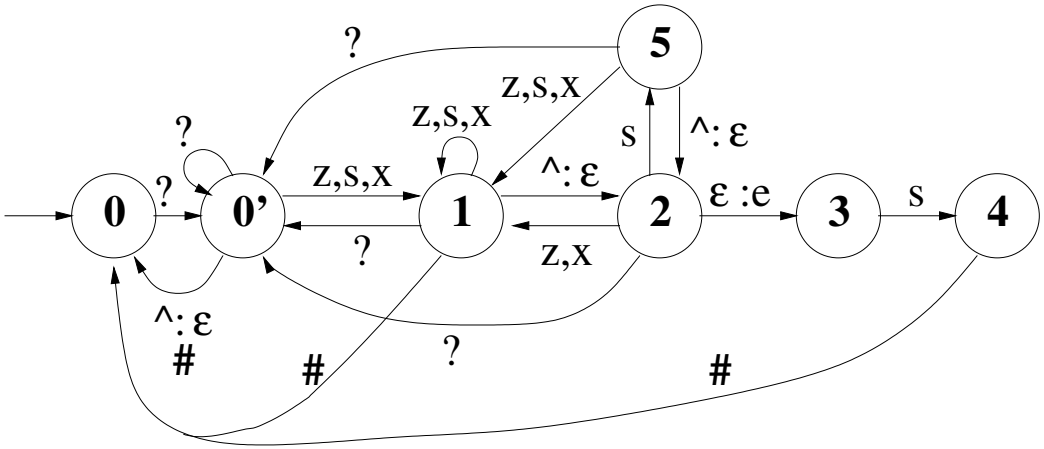
To go from lexical to surface form, use 'Stage 1' transducer followed by a bunch of orthographic rule transducers like the above. (Made more efficient by back-end compilation into one single transducer.)

The results of this generation process are typically deterministic (each lexical form gives a unique surface form), even though our transducers make use of non-determinism along the way.

Running the same cascade backwards lets us do parsing (surface to lexical form). Because of ambiguity, this process is frequently non-deterministic: e.g. 'foxes' might be analysed as fox+N+PL or fox+V+Pres+3SG.

Such ambiguities are not resolved by morphological parsing itself: left to a later processing stage.

Quick exercise 2

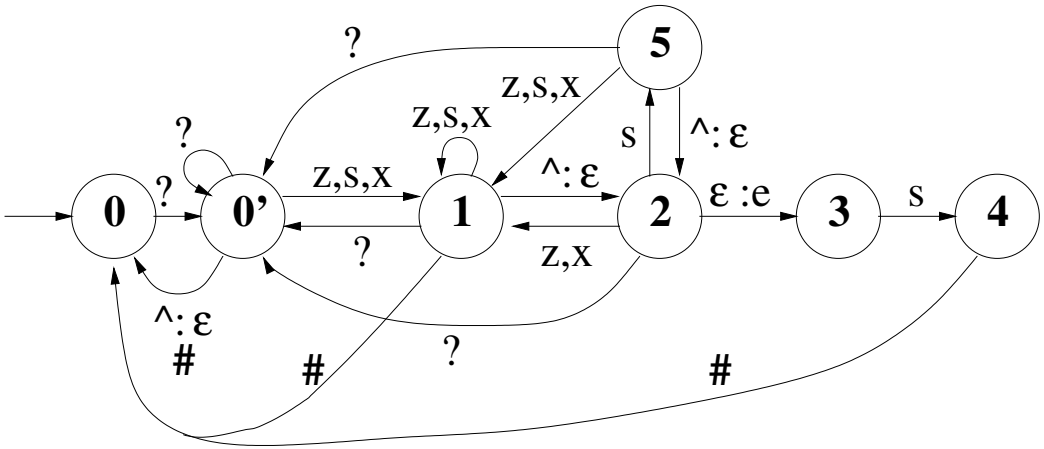


Apply this backwards to translate from surface to int. form.

Starting from state 0, how many sequences of transitions are compatible with the input string 'asses' ?

1. 1
2. 2
3. 3
4. 4
5. More than 4

Solution



On the input string 'asses', 10 transition sequences are possible!

- $0 \xrightarrow{a} 0' \xrightarrow{s} 1 \xrightarrow{s} 1 \xrightarrow{\epsilon} 2 \xrightarrow{e} 3 \xrightarrow{s} 4$, output ass^s
- $0 \xrightarrow{a} 0' \xrightarrow{s} 1 \xrightarrow{s} 1 \xrightarrow{\epsilon} 2 \xrightarrow{e} 0' \xrightarrow{s} 1$, output ass^es
- $0 \xrightarrow{a} 0' \xrightarrow{s} 1 \xrightarrow{s} 1 \xrightarrow{e} 0' \xrightarrow{s} 1$, output asses
- $0 \xrightarrow{a} 0' \xrightarrow{s} 1 \xrightarrow{\epsilon} 2 \xrightarrow{s} 5 \xrightarrow{\epsilon} 2 \xrightarrow{e} 3 \xrightarrow{s} 4$, output as^s^s
- $0 \xrightarrow{a} 0' \xrightarrow{s} 1 \xrightarrow{\epsilon} 2 \xrightarrow{s} 5 \xrightarrow{\epsilon} 2 \xrightarrow{e} 0' \xrightarrow{s} 1$, output as^s^es
- $0 \xrightarrow{a} 0' \xrightarrow{s} 1 \xrightarrow{\epsilon} 2 \xrightarrow{s} 5 \xrightarrow{e} 0' \xrightarrow{s} 1$, output as^ses
- Four of these can also be followed by $1 \xrightarrow{\epsilon} 2$ (output ^).

The Porter Stemmer

Lexicon can be quite large with finite state transducers

Sometimes need to extract the stem in a very efficient fashion (such as in IR)

The Porter stemmer: a lexicon-free method for getting the stem of a given word

ATIONAL $\rightarrow$ ATE (e.g., relation $\rightarrow$ relate)
ING $\rightarrow \epsilon$ if stem comtains a vowel (e.g. motoring $\rightarrow$ motor)
SSES $\rightarrow$ SS (e.g., grasses $\rightarrow$ grass)

Makes errors:

organization $\rightarrow$ organ
doing $\rightarrow$ doe
numerical $\rightarrow$ numerous
policy $\rightarrow$ police

Current methods for morphological analysis

A vibrant area of study

Mostly done by *learning from data*, just like many other NLP problems. Two main paradigms: unsupervised morphological parsing and supervised one.

NLP solvers are not perfect! They can make mistakes. Sometimes ambiguity can't even be resolved. BUT, for English, morphological analysis is highly accurate. With other languages, there is still a long way to go.

Finite state transducers in NLP

One of the basic tools that is used for many applications

- Speech recognition
- Machine translation
- Part-of-speech tagging
- ... and many more

Next class

Part-of-speech tagging:

- What are parts of speech?
- What are they useful for?
- Zipf's law and the ambiguity of POS tagging
- One problem NLP solves really well (... for English)