Morphology parsing

Grundläggande textanalys: Lecture 3

Course given by
Marie Dubremetz
`marie.dubremetz@lingfil.uu.se`

At:
Uppsala University
Department of Linguistic and Philology
Acknowledgement to:
School of Informatics
University of Edinburgh

April 2015

Sources

These slides (from slide 3 to slide 18), slightly modified, are borrowed from **John Longley** from the **school of Informatics (University of Edinburgh)** with his kind authorization.

1. Morphology parsing: the problem
2. FSTs for morphology parsing and generation

(This lecture is taken almost directly from Jurafsky and Martin [2009] chapter 3, sections 1–7.)

Which pre-processing today?

The steps of pre-pocessing (can) include:

- Normalization of encoding, format etc.
- Cleaning
- Word normalization (language variations, sms etc.)
- Tokenization and sentence segmentation [cf previous course]
- Lemmatization and Morpho-analysis $\Leftarrow$
- Stemming $\Leftarrow$
- Parsing

[$\Leftarrow$ Today's class]

Morphological parsing: the problem

In many languages, words can be made up of a main lemma (carrying the basic dictionary meaning) plus one or more affixes carrying grammatical information. E.g. in English:

Surface form:	cats	walking	flickor [in Swedish]
Lexical form:	cat+N+PL	walk+V+PresPart	flicka+Undef+PL

Morphological parsing is the problem of extracting the lexical form from the surface form.

Should take account of:

- Irregular forms (e.g. goose → geese)
- Systematic rules (e.g. ‘e’ inserted before suffix ‘s’ after s,x,z,ch,sh: fox → foxes, watch → watches)

Why bother?

- NLP tasks involving meaning extraction will often involve morphology parsing.
- 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 new words easier (e.g. 'tweet').
- Morphology parsing is just more interesting than brute listing!

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 ˆ s #
Lexical form:	fox +N +PL

(ˆ 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!

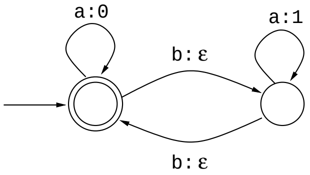
Finite-state transducer (Etymology)

Finite: an FST "finite" because it has a finite number of states and a limited memory. Note: Finite number of states does not mean finite number of possible input strings!
State: an FST is a machine constituted of states and transitions. Its behaviour is lead by a word given as an input: the transducer transits from a state to another by following defined transitions each time it reads a new letter.
Transducer: from the latin trans- 'across' + ducere 'lead'. The transducer is literally the machine that leads the transition/transformation of an input string into another string.

Finite-state transducers

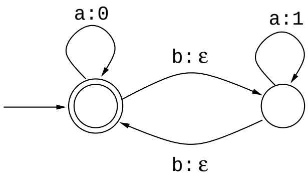
We can consider ϵ-NFAs (over an alphabet Σ) in which transitions may also (optionally) produce *output* symbols (over a possibly different alphabet Π).

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.

Clicker exercise



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

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 $\Delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Pi \cup \{\epsilon\}) \times Q$

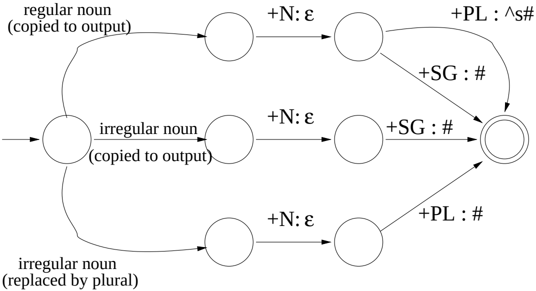
Example of transition relation:
(q1,a,b,q2),(q2,c,d,q3)

From T as above, we can obtain another transducer $\overline{T}$ just by swapping the roles of inputs and outputs.

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.

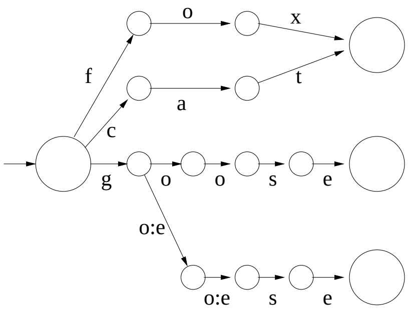
We can do this with a transducer of the following schematic form:



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 2: From intermediate to surface form

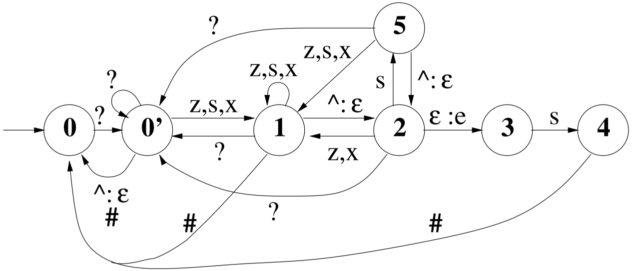
To convert a sequence of morphemes to surface form, we apply a number of orthographic rules such as the following.

- 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,ˆ,#.
(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. 'exˆserviceˆmen#' to be translated to 'exservicemen'.

Putting it all together

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

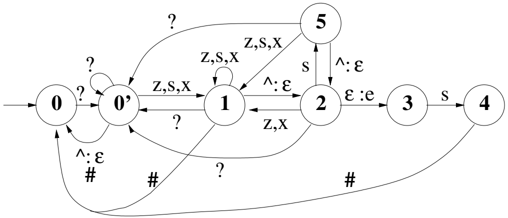
To go from lexical to surface form, use 'Stage 1' transducer followed by a bunch of orthographic rule transducers like the above.

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.

Clicker exercise 2

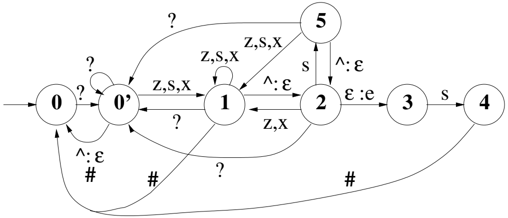


Apply this backwards to translate from surface to int. form.

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 ^).

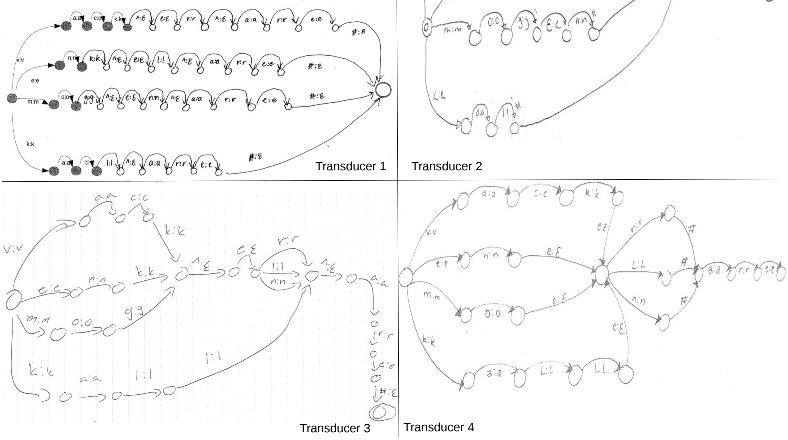
Exercise

In the next slide you will see a problem and 4 transducers that are supposed to solve it. Only 2 transducers correctly answer this problem. Can you find them?

Exercise

The majority of Swedish adjectives form the comparative by adding -are to the uninflected positive form of the adjective(e.g. kall, kallare). Those adjectives that end with -el, -en or -er in the positive, drop the -e- before the comparative ending(e.g.,enkel, enklare).

Create the morphological transducer that apply those spelling rules to the intermediate form kall^are#, enk^el^are#, mog^en^are#, vack^er^are# and return as an output the surface realization (e.g. enk^el^are#->enklare). Below is the beginning of your transducer, complete the figure. (Have a look at chapter 3.2 of J&M part on FST).



Lexicon-Free FSTs: The Porter Stemmer

Imagine a list of documents that contain the words "dances", "dance", "danced"
And a user looking for a document about "dancing".
If we do not apply any transformation to those words the machine cannot match them together.
Before the 80's we used lexicons: heavy, hard to develop.

Lexicon-Free FSTs: The Porter Stemmer

Imagine a list of documents that contain the words "dances", "dance", "danced"

And a user looking for a document about "dancing".

If we do not apply any transformation to those words the machine cannot match them together.

Before the 80's we used lexicons: heavy, hard to develop.

Until came the Porter algorithm:

In a set of rules written beneath each other, only one is obeyed, and this will be the one with the longest matching S1 for the given word. For example, with

SSES → SS
IES → I
SS → SS
S →

(here the conditions are all null) CARESSES maps to CARESS since SSES is the longest match for S1. Equally CARESS maps to CARESS (S1='SS') and CARES to CARE (S1='S').

In the rules below, examples of their application, successful or otherwise, are given on the right in lower case. The algorithm now follows:

Step 1a

SSES → SS		caresses	→ caress
IES → I		ponies	→ poni
		ties	→ ti
SS → SS		caress	→ caress
S →		cats	→ cat

Step 1b

(m>0)	EED → EE	feed	→ feed
		agreed	→ agree
(*v*)	ED →	plastered	→ plaster
		bled	→ bled
(*v*)	ING →	motoring	→ motor
		sing	→ sing

If the second or third of the rules in Step 1b is successful, the following is done:

AT → ATE	conflat(ed)	→ conflate
BL → BLE	troubl(ing)	→ trouble
IZ → IZE	siz(ed)	→ size
(*d and not (*L or *S or *Z))		
→ single letter	hopp(ing)	→ hop
	tann(ed)	→ tan
	fall(ing)	→ fall
	hiss(ing)	→ hiss
	fizz(ed)	→ fizz
(m=1 and *o) → E	fail(ing)	→ fail
	fil(ing)	→ file

The rule to map to a single letter causes the removal of one of the double letter pair. The –E is put back on –AT, –BL and –IZ, so that the suffixes –ATE, –BLE and –IZE can be recognised later. This E may be removed in step 4.

Step 1c

(*v*) Y → I	happy	→ happi
	sky	→ sky

Step 1 deals with plurals and past participles. The subsequent steps are much more straightforward.

Step 2

(m>0)	ATIONAL	→ ATE	relational	→ relate
(m>0)	TIONAL	→ TION	conditional	→ condition
			rational	→ rational
(m>0)	ENCI	→ ENCE	valenci	→ valence
(m>0)	ANCI	→ ANCE	hesitanci	→ hesitance
(m>0)	IZER	→ IZE	digitizer	→ digitize
(m>0)	ABLI	→ ABLE	conformabli	→ conformable
(m>0)	ALLI	→ AL	radicalli	→ radical
(m>0)	ENTLI	→ ENT	differentli	→ different
(m>0)	ELI	→ E	vileli	→ vile
(m>0)	OUSLI	→ OUS	analogousli	→ analogous
(m>0)	IZATION	→ IZE	vietnamization	→ vietnamize
(m>0)	ATION	→ ATE	predication	→ predicate
(m>0)	ATOR	→ ATE	operator	→ operate
(m>0)	ALISM	→ AL	feudalism	→ feudal
(m>0)	IVENESS	→ IVE	decisiveness	→ decisive
(m>0)	FULNESS	→ FUL	hopefulness	→ hopeful
(m>0)	OUSNESS	→ OUS	callousness	→ callous
(m>0)	ALITI	→ AL	formaliti	→ formal
(m>0)	IVITI	→ IVE	sensitiviti	→ sensitive
(m>0)	BILITI	→ BLE	sensibiliti	→ sensible

The test for the string S1 can be made fast by doing a program switch on the penultimate letter of the word being tested. This gives a fairly even breakdown of the possible values of the string S1. It will be seen in fact that the S1-strings in step 2 are presented here in the alphabetical order of their penultimate letter. Similar techniques may be applied in the other steps.

Step 3

(m>0)	ICATE	→ IC	triplicate	→ triplic
(m>0)	ATIVE	→	formative	→ form
(m>0)	ALIZE	→ AL	formalize	→ formal
(m>0)	ICITI	→ IC	electriciti	→ electric
(m>0)	ICAL	→ IC	electrical	→ electric
(m>0)	FUL	→	hopeful	→ hope
(m>0)	NESS	→	goodness	→ good

Step 4

(m>1)	AL	→	revival	→ reviv
(m>1)	ANCE	→	allowance	→ allow
(m>1)	ENCE	→	inference	→ infer
(m>1)	ER	→	airliner	→ airlin
(m>1)	IC	→	gyroscopic	→ gyroscop
(m>1)	ABLE	→	adjustable	→ adjust
(m>1)	IBLE	→	defensible	→ defens

Lexicon-Free FSTs: The Porter Stemmer

The most famous stemmer algorithm is the Porter algorithm Like morpho-analyzers, stemmers can be seen as cascaded transducers but it has no lexicon.

Porter algorithm example

For words like: falling, attaching, sing, hopping etc.

Step 1:

1. If the word has more than one syllab and end with 'ing':
2. ▶ Remove 'ing' and apply the second step

Step 2:

1. If word finishes by a double consonant (except L S Z):
2. ▶ Transform it into a single letter

Porter algorithm example

For words like: falling, attaching, sing, hopping etc.

Step 1:

1. If the word has more than one syllab and end with ‘ing’:
2. ▶ Remove ’ing’ and apply the second step

Step 2:

1. If word finishes by a double consonant (except L S Z):
2. ▶ Transform it into a single letter

falling → fall
attaching → attach
sing → sing
hopping → hop

Porter algorithm limits and advantages

Will be wrong on irregularities:
something → someth
But:

- Very simple algorithm
- Useful for IR

References

Daniel Jurafsky and James H Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, volume 163 of *Prentice Hall Series in Artificial Intelligence*. Prentice Hall, 2009.

Martin F. Porter. *An Algorithm for Suffix Stripping Program*. 1980.

Chapter 3 for the second edition.

Problem with automaton?

Read chapter 2.

Play with this:

`http://automatonsimulator.com/`