Lecture 3: The end of the word

Instructor: Robert C. Berwick
berwick@ai.mit.edu
The Menu Bar

• Administrivia
  Lecture 2 posted; Lab 1b(“component III”) out today
  • Finite-state transducer model: the wrapup
  • Kimmo & Laboratory 1b: how-to
  • Complexity of fst’s – too weak? Too strong?
    What makes a good computational linguistics representation? A good algorithm?
The Three Ideas of Two-Level Morphology

- Rules are symbol-to-symbol constraints that are applied in parallel, not sequentially like rewrite rules, via sets of transducers.
- The constraints can refer to the lexical context, to the surface context or to both contexts at the same time.
- Lexical lookup and morphological analysis are performed in tandem.
Word of the day

- Turkish word:
  uygarlas,tiramadiklarimizdanmis,sinizcasina

= 
uygar+las,+tir+ama+dik+lar+imiz+dan+mis,+siniz+casina

(behaving) as if you are among those whom we could not cause to become civilized
Two fs machines in tandem
Declarative filtering

- **ALL** possible string pairs come in
- Spelling change fst’s only allow valid spelling pairs through the door (doesn’t care about what a valid morpheme is): $g:g \ 0:o \backslash s:s$
- Morpheme tree allows only valid root + affix sequences through the door (doesn’t care about spelling): $b\ e\ e\ r \leftrightarrow b\ e\ +\ e\ r$
- **Combination** admits only permissible lexical:surface word forms
Conceptually:...

define L1 {talk} | {walk} | {work};

define L2 "+Vb":0 | "+V3s":s | "+Vpr":{ing} | "+Vpst":{ed} ;
Machines

Set of parallel of two-level rules compiled into finite-state automata interpreted as transducers Koskenniemi ‘83
In practice

- Bidirectional: generation or analysis
- Compact and fast
- Commercially available for about 20 languages including English, German, Dutch, French, Italian, Spanish, Portuguese, Finnish, Russian, Turkish, Japanese, ...
- Research systems for many other languages, including Arabic, Malay, ...

Finite-state transducer

canonical form

inflected form

6.863J/9.611J SP04 Lecture 3
Ambiguities

- être + IndP + SG + P1
- suivre + IndP + SG + P1
- suivre + IndP + SG + P2
- suivre + Imp + SG + P2
- payer + IndP + SG + P1

"upper language"

French Transducer

suis

"lower language"

paie
paye
Insert ‘e’ before non-initial z, s, x ("epenthesis")
Successful pairing of foxes, fox+s
Can a transducer do this?

- igpay atinlay is unfay
- define Cons [ b ];
  define Vowel [a | e | i | o | u ];
  define Ltr [ Cons | Vowel ];
  define Limit [" " | "\t" | .#. ];

- Suppose just one consonant, e.g, be
- Strategy?
What about rule ordering?
Rule Conflicts

Resolution by underspecification:

\[ k:0 \mid k:v \iff \text{Vowel } _ \text{Vowel} \]

\[ k:v \iff \text{u } _ \text{u} \]

Exception

General rule

makun

ma un

pukun

puvun

6.863J/9.611J SP04 Lecture 3
Constraints on both sides

N:m correspondence requires a following p on the lexical side.
In this context, all other possible realization of a lexical N are prohibited.

p:m correspondence requires a preceding m on the surface side.
In this context, all other possible realization of a lexical p are prohibited.
Parallel application – how?
Sequential Application

N \rightarrow m / _ p

p \rightarrow m / m _
Machine Rule 1 ("N goes to m")

Rule 1: N→m | __ p
Machine Rule 2 ("p goes to m")

Rule 2: \( p \rightarrow m \mid m \_\_ \)
Sequential Application in Detail

\[
\begin{align*}
\text{k a N p a n} \\
0 & 0 & 0 & 2 & 0 & 0 & 0 \\
\downarrow & & & & & & \\
& \text{k a m p a n} \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\downarrow & & & & & & \\
& \text{k a m m a n}
\end{align*}
\]
When is this possible?

- Rule 1
- Rule 2
- Rule 3
- Rule 4

Single FST

input

output

input
OK then...

- To apply all 5 rules at once, we can simply intersect them all, right? (So given character pair has to pass all the spelling checks at each point)

  - Right?
  - Wrong!
Relations

Ordered Set: members are ordered.

Ordered Pair: \(<A,B>\) vs. \(<B,A>\)

Relation: set whose members are ordered pairs
• (lexical, surface)
Relations

An Infinite Relation:

Identity Relation: `<“fly”, “fly”>`
What does transducer define?

- Lexical form is a finite-state language (a regular language)
- Surface form is a regular language
- Transducer pairs (lexical form, surface form) = regular relation (“rational relation:”)
- What are the properties of rational relations? Is this the same as what Kimmo does?
Closure properties of FSTs not the same as FSAs!

- NOT closed under intersection
  why? Example: \((a:b)*(0:c)* \cap (0:b)*(a:c)*\)
  Intuition: 1\(^{st}\) makes equal a’s, b’s; 2\(^{nd}\), equal a’s & c’s
  So output is: < , >, <a, bc>, <aa, bbcc>, ...

- NOT possible to make FTNs deterministic in general either...
  why? Consider following FST:
Inherently nondeterministic FTN

Diagram:

- Nodes: 0, 1, 2, 3
- Edges:
  - 0 → 1: x:a
  - 1 → 0: x:a
  - 1 → 3: a:a
  - 3 → 1: b:b
  - 2 → 3: b:b
  - 3 → 2: b:b

6.863J/9.611J SP04 Lecture 3
Composition

Composition is an operation on two relations.

Composition of the two relations \(<x,y>\) and \(<y,z>\) yields \(<x, z>\)

Example: \(<\text{“cat”}, \text{“chat”}>\) with \(<\text{“chat”}, \text{“Katze”}>\) gives \(<\text{“cat”}, \text{“Katze”}>\)
Composition

- $A \circ C \circ B$  The relation $C$ such that if $A$ maps $x$ to $y$ and $B$ maps $y$ to $z$, $C$ maps $x$ to $z$. 
Composition

\[ \text{Composition} \]

\[ \begin{array}{cccccc}
& c & a & t & & \\
c & h & a & t & & \\
& c & h & a & t & \rightarrow \\
& \text{K} & h & a & t & z & e
\end{array} \]
Composition

Merging the two networks
Composition

The Composition of the Networks

What is this reminiscent of?
Composition

FTNs **ARE** closed under composition
What are the implications?

- FTNs inherently require backup if simulated (in the worst case) – Kimmo at least NP-hard (proof later on)
- Empty elements cause computational complexity (unless restricted – equal length condition)
- Composition can save us, but then rule ordering must be watched carefully
Sequential Application in Detail

\[
\begin{align*}
\text{k a N p a n} \\
0 & 0 & 0 & 2 & 0 & 0 & 0 \\
\text{k a m p a n} \\
0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\text{k a m m a n}
\end{align*}
\]
Let’s see an example - English races’

Recognizing surface form "races’".
0 (r.r) --> (1 1 1 2 1 1)
      EP G Y EL I

1   (a.a) --> (1 1 4 1 2 1)
      EP G Y EL I

2   (c.c) --> (1 2 16 2 11 1)

3   (e.o) --> (1 1 16 1 12 1)
      EP G Y EL I

4   Entry |race| ends --> new lexicon N, config (1 1 16 1 12 1)
      EP G Y EL I

- *note* that each of the 5 automata are running “in parallel” (what is the first one?)
Backup search

**Problem:** $e$ is paired with 0 (null)...

(which is wrong - it’s guessing that the form is “racing” - has stuck in an empty (zero) character after $c$ but before $e$) - *elision* automaton has 2 choices

This is *nondeterminism* in action (or inaction)!

5
Entry /0 ends --> new lexicon C1, config (1 1 16 1 12 1)  
EP G Y EL I

6
Entry /0 is word-final --> path rejected (leftover input).
5
(+.0) --> (1 1 16 1 13 1)  
EP G Y EL I

6
Nothing to do.
5
(+.e) --> automaton Epenthesis blocks from state 1.
4
Entry |race| ends --> new lexicon P3, config (1 1 16 1 12 1)  
EP G Y EL I
22 steps later

3  (e.e) --> (1 1 16 1 14 1)
    EP G Y EL I

4  Entry |race| ends --> new lexicon N, (1 1 16 1 14 1)
    E   G Y EL I

5  Entry /0 ends --> new lexicon C1, config (1 1 16 1 14 1)

6  Entry /0 is word-final --> rejected (leftover input)

7  Entry +/s ends --> new lexicon C2, (1 4 16 2 1 1)

8  Entry /0 is word-final --> rejected (leftover input)

9  End --> lexical form ("race+s’") (N PL GEN)
Laboratory 1b

Goals:
- How to use Kimmo to analyze another language (Spanish), as example “front end”
- Build automata for some simple Spanish morphological/phonemic rules (that interact)
- Build lexicon
- Learn what is hard and what is easy about this
- Recognize all and only the words in spanish.rec; Generate all the surface forms

Resources:
- Lab1b pdf file link from web page
- File of all the surface words to parse/reject (covering the phenomena) spanish.rec, also linked from web page
- Pykimmo, pcimmo & documentation
- Program to `compile’ rules into automata: fst
What you must turn in (via URL)

1. A description of *how your system operates*
2. URL ptrs to your `.lex` and `.rul` files
   span.lex
   span.rul
3. A log of a recognition run on the file
   `spanish.rec` which is linked on the web page & also at toplevel on course locker
4. Discussion of what you built/why
5. You must answer 3 questions:
The questions

- What is your name?
- What is your quest?
- What...
The phenomena under study

- You are given the orthography, including some special characters to stand for the accented ones á, é, ó, ü, ñ; and some underlying characters you may find essential, such as J, C, Z.
  - Wise to proceed by first building the automata (rul) file; then the lexicon(s) - because you can test the rules without any lexicon by generation of a surface form
  - The automata can be built (roughly) by considering each phenomenon separately
  - 3 kinds of phenomena
The phenomena

1. *g-j mutation*
2. *z-c mutation*
3. *pluralization*
4. *Noun endings*
5. *Verb conjugation - 1 form*
Phenomenon 1: g-j mutation

- **g-j mutation**
  - $g \rightarrow j$ before a back vowel
    - *coger* (catch, infinitive); *cojo*, *coges*, *coge*, *cogemos*, *cogen*, *coja* (NOTE: *coger* is NOT the lexical underlying form!!!)
  - But some verbs not subject to this (exceptions!)
    - *llegar* (arrive); *llego*, *llegan*, *pagar* (pay); *pago*, *pagan*
  - *Don't accept* *llejo*, *lleja*, *cogo*, *coga* (the words don’t come marked with * on their sleeves, of course!)
  - Hint: can use the lexical (underlying) character J to solve (but there are other ways to do it)
How to build Kimmo systems

How to build lexicons using morpheme states and the actual lexical entries
How to build automata for spelling changes
Format for .lex file - 2 parts

1. Lexicon: Morpheme classes – name all the states, some transitions
   (1 or more blank lines)
2. Lexicon entries: transitions between the states
   (Recall that we consider only the underlying form combinations here – stems + affixes, not spelling change rules on the surface)
Example: lexicon design

Phenomena: Nouns and Verbs take different endings

Answer:

Different *morpheme states* for Nouns and Verbs
Example surface (s) underlying (u) pairs tell us what to do

- **ciudad**
  ciudad [N(city)]

- **ciudades**
  ciudad+s [N(city)pl]
Automaton design for lexicon

Q: what do we need to add to noun alternation?
Adding plural

start

noun

plural

End

verb

singular

End
How do we build spelling change automata?
Example: look at phenomenon, then see first how to describe

- What is the left and right context of the change?
- Write it as a declarative constraint
- Remember that you can use both the surface and the lexical characters to admit or to rule out a possibility
Phenomenon 2: z-c mutation

• **z-c mutation**
  
z → c before front vowels, z otherwise
  
cruzar (to cross); crujo, cruzas, cruza, cruzamos, cruzan, cruce

• If s causes a front vowel (e.g., e) to surface, then the rule still applies:
  
lápiz, lápices (pencil, pencils) [ l^piz, l^pices]
What’s the automaton got to do?

- Start
- KeepZ
- ChangeToC
- Reject
Now add the arcs...
Phenomenon 3: pluralization

- Adding *s* to a noun that ends in a consonant forces a surface *e* to appear:
  - *ciudad* (city); *ciudades*
- This can interact with other rules, e.g., *z-c-mutation*:
  - *lápiz*, *lápices*
- Nouns ending in a vowel are not subject to this rule: *bota*, *botas*
The lexicon – take 2

- Add a gloss at the very end of the process, so as to return the feature list and ‘translation’, e.g., *venzo* [1p sg pres indic conquer, defeat] (first person, singular, present tense, indicative)
  
  - We’ll show *how* to add this in a moment
  
  - You will deal with two types of ‘endings’
  1. Noun endings: plural suffix +*s*
  2. Verb endings: verb stem + tense markers
    
    Simplest: infinitive marker +*ar, +er, +ir*

    See table in pdf file for details: 5 x 3 table for Present tense; ditto for Subjunctive tense ("*I might....*")
Instead of writing fst tables...

- You can use the program fst
- To run:
  build fst type rules in file spanish.fst, then
  
  \texttt{fst -o ~yourpath/spanish.rul ~yourpath/spanish.fst}

- Also script to print fst files to dot, for ps viewing
- Format for fst rules:
FST rules

• “b after a vowel turns to a”

subset vowel a e

machine “bintoa”

state foo

vowel:vowel bar

b:b foo

c:c foo

d:d foo

others reject

rejecting state bar

b:e foo

b:b reject

others foo
Design of morpheme machine

- One big fsa
- Like this...
Specifying details

- List states – in cyan – followed by sets of transition labels (possible outgoing arcs)

Begin: N_Root1 Adj_Root V_Root

N_root: Poss To_adj

---

- For each arc: List transitions & next states, and output

N_root1:

Kol N_root Noun[‘arm’]
Kitab N_root Noun[‘book’]

...
The End

End:
0    #    ""