

6.863J Natural Language Processing

Lecture 2: Automata, Two-level phonology, & PC-Kimmo (the Hamlet lecture)

Instructor: Robert C. Berwick
berwick@ai.mit.edu



The Menu Bar

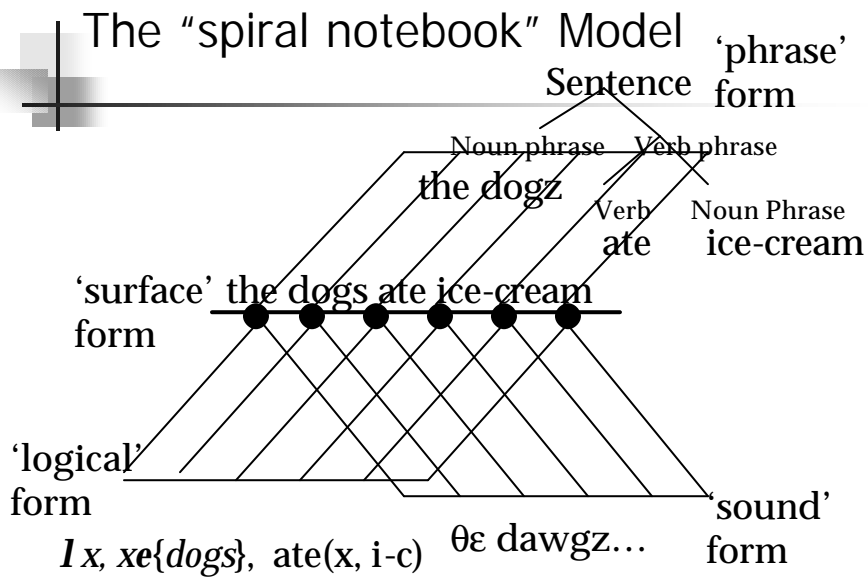
- Administrivia
 - web page: www.ai.mit.edu/courses/6.863/ now with Lecture 1, Lab1
 - Questionnaire posted (did you email it?)
 - Lab1: split into Lab1a (this time) Lab1b (next time)
- *What* and *How*: word processing, or computational morphology
- What's in a word: morphology
- Modeling morpho-phonology by finite-state devices
- Finite-state automata vs. finite state transducers
- Some examples from English
- PC-Kimmo & Laboratory 1:how-to

Levels of language

- Phonetics/phonology/morphology: what words (or subwords) are we dealing with?
- Syntax: What phrases are we dealing with? Which words modify one another?
- Semantics: What's the literal meaning?
- Pragmatics: What should you conclude from the fact that I said something? How should you react?

6.863J/9.611J SP03 Lecture 2

The "spiral notebook" Model



6.863J/9.611J SP03 Lecture 2

Start with words: they illustrate all the problems (and solutions) in NLP

- Parsing words
Cats → CAT + N(oun) + PL(ural)
- Used in:
 - Traditional NLP applications
 - Finding word boundaries (e.g., Latin, Chinese)
 - Text to speech (*boathouse*)
 - Document retrieval (example next slide)
- In particular, the problems of *parsing*, *ambiguity*, and *computational efficiency* (as well as the problems of *how people do it*)

6.863J/9.611J SP03 Lecture 2

Example from information retrieval

- Keyword retrieval: *marsupial* or *kangaroo* or *koala*
- Trying to form equivalence classes - ending not important
- Can try to do this without *extensive* knowledge, but then:
organization → organ European → Europe
generalization → generic noise → noisy

6.863J/9.611J SP03 Lecture 2

Morphology

- Morphology is the study of how *words* are *built up* from smaller *meaningful* units called *morphemes* (morph= shape; logos=word)
- Easy in English – what about other languages?

6.863J/9.611J SP03 Lecture 2

What about other languages?

Present indicative	Imperf	Imperf Indic	Future	Preterite	Present Subjun	Cond	Imp. Subj	F S
amo		amaba	amaré	amé	ame	amaría	amara	a
amas	ama	amabas	amarás	amaste	ames	amarías	amaras	a
	ames							
ama		amamba	amará	amó	ame	amaría	amara	a
amamos								
amáis	amad	amambais	amremos	amomos	amemos	amaríanos	amarais	a
	amáis							
aman		amamban	amarán	amaron	amen	amarían	amarain	a

How to love in Spanish...incomplete...you can finish it after Valentine's Day...

6.863J/9.611J SP03 Lecture 2

What about other languages?

Lexical: Paris+mut+nngau+juma+niraq+lauq+sima+nngit+junga
Surface: Pari mu nngau juma nira lauq sina nngit tunga

Paris = (root = Paris)
+mut = terminalis case ending
+nngau = go (verbalizer)
+juma = want
+niraq = declare (that)
+lauq = past
+sima = (added to -lauq- indicates "distant past")
+nngit = negative
+junga = 1st person sing. present indic (nonspecific)

Figure 2: Inuktitut: *Parimunngaujumaniralausimannngittunga* = "I never said I wanted to go to Paris"

6.863J/9.611J SP03 Lecture 2

What about other processes?

- Stem: core meaning unit (morpheme) of a word
- Affixes: bits and pieces that combine with the stem to modify its meaning and grammatical functions

Prefix: *un-*, *anti-*, etc.

Suffix: *-ity*, *-ation*, etc.

Infix:

Tagalog: *um+hinigi* → **humingi** (borrow)

Any infixes in 'nonexotic' language like English?

Here's one: *un-f*****-believable*

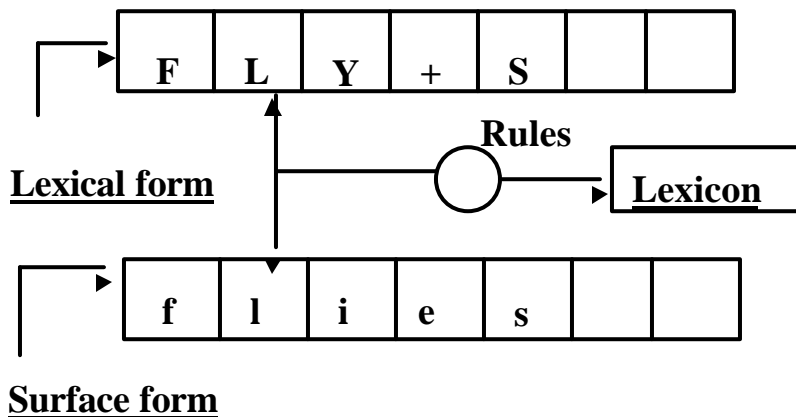
6.863J/9.611J SP03 Lecture 2

OK, now how do we deal with this computationally?

- What knowledge do we need?
- How is that knowledge put to use?
- What:
 - duckling*; *beer* (implies what K...?)
 - chase* + *ed* → *chased* (implies what K?)
 - breakable* + *un* → *unbreakable* ('prefix')
- How: a bit trickier, but clearly we are at least doing this kind of mapping...

6.863J/9.611J SP03 Lecture 2

Our goal: PC-Kimmo



6.863J/9.611J SP03 Lecture 2

Two parts to the “what”

1. Which units can glue to which others (roots and affixes) (or stems and affixes), eg,
2. What ‘spelling changes’ (orthographic changes) occur – like dropping the e in ‘chase + ed’

OK, let’s tackle these one at a time, but first consider a (losing) alternative...

6.863J/9.611J SP03 Lecture 2

KISS: A (very) large dictionary

1. Impractical: some languages associate a single meaning w/ a Sagan number of distinct surface forms (600 billion in Turkish)

German: *Leben+s+versicherung+gesellschaft+s+angestellter*
(life+ CmpAug+insurance+ CmpAug+company+CompAug
+employee)

Chinese compounding: about 3000 ‘words,’ combine to yield tens of thousands

2. Speakers don’t represent words as a list

Wug test (Berko, 1958)

Juvenate is rejected slower than *pertoire* (real prefix matters)

6.863J/9.611J SP03 Lecture 2

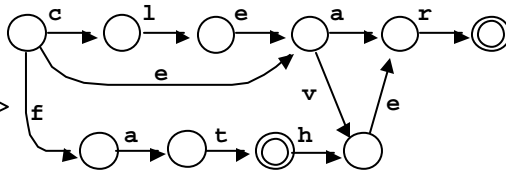
Representing possible roots + affixes as a finite-state automaton

Wordlist

clear
clever
ear
ever
fat
father

compile

Network



/usr/dict/words

25K words
206K chars

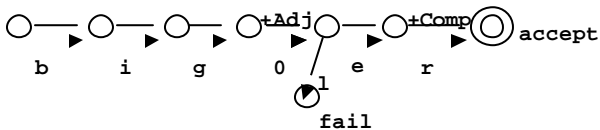
2 sec

FSM

17728 states,
37100 arcs

6.863J/9.611J SP03 Lecture 2

Now add in states to get possible combos, as well as features



This much is easy – a straightforward fsa
States = equivalence classes

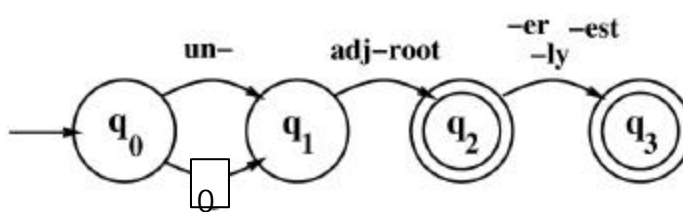
6.863J/9.611J SP03 Lecture 2

English morphology: what states do we need for the fsa?

- As an example, consider adjectives
Big, bigger, biggest
Cool, cooler, coolest, coolly
Red, redder, reddest
Clear, clearer, clearest, clearly, unclear, unclearly
Happy, happier, happiest, happily
Unhappy, unhappier, unhappiest, unhappily
Real, unreal, silly

6.863J/9.611J SP03 Lecture 2

Will this fsa work?



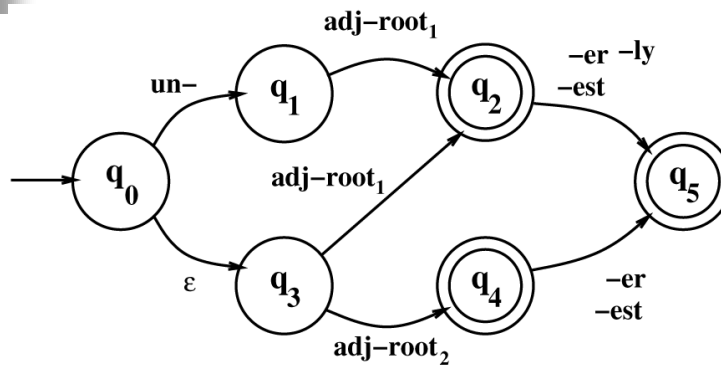
6.863J/9.611J SP03 Lecture 2

Ans: no!

- Accepts all adjectives above, but
- Also accepts *unbig*, *readly*, *realest*
- Common problem: overgeneration
- Solution?

6.863J/9.611J SP03 Lecture 2

Revised picture



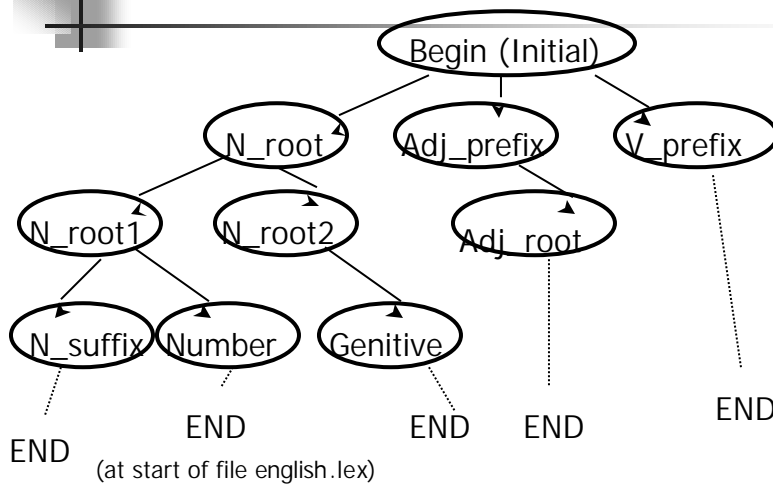
6.863J/9.611J SP03 Lecture 2

How does PC-Kimmo represent this?

Here's what the pc-kimmo fsa looks like – the fsa states are called 'alternation classes' or 'lexicons'

6.863J/9.611J SP03 Lecture 2

PC-Kimmo states for affix combos (portion) = lexicon tree



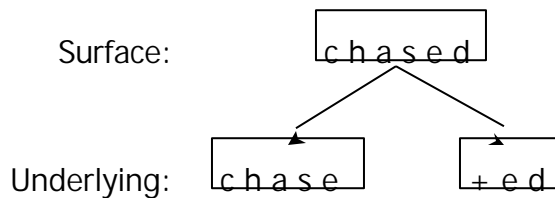
6.863J/9.611J SP03 Lecture 2

Next: what about the spelling changes? That's harder!

- ✓ Which units can glue to which others (roots and affixes) (or stems and affixes)
2. What 'spelling changes' (orthographic changes) occur – like dropping the e in 'chase + ed'

6.863J/9.611J SP03 Lecture 2

Mapping between surface form & underlying form



But clearly this can go either way – given the underlying form, we can *generate* the surface form – so we really have a *relation* betw. surface & underlying form, viz.:

6.863J/9.611J SP03 Lecture 2

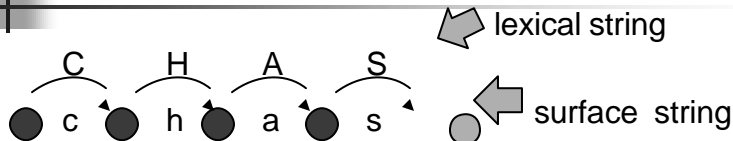
Conventional notation

Lexical (underlying) form: c h a s e + e d
Surface form: c h a s 0 0 e d

The 0's "line up" the lexical & surface strings
This immediately suggests a finite-state automaton
'solution' : an extension known as a
finite-state transducer

6.863J/9.611J SP03 Lecture 2

Finite-state transducers: a *pairing* between lexical/surface strings



- Or more carefully

6.863J/9.611J SP03 Lecture 2

Definition of finite-state automaton (fsa)

- A (*deterministic*) *finite-state automaton* (FSA) is a quintuple $(Q, \Sigma, \delta, q_0, F)$ where
 - Q is a *finite* set of states
 - Σ is a finite set of terminal symbols, the alphabet
 - $q_0 \in Q$ is the initial state
 - $F \subseteq Q$, the set of final states
 - δ is a function from $Q \times \Sigma \rightarrow Q$, the transition function

6.863J/9.611J SP03 Lecture 2

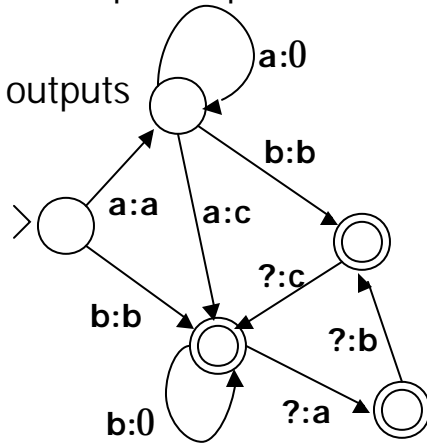
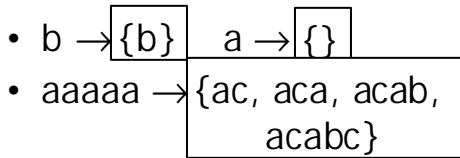
Definition of finite-state transducer

- state set Q
- initial state q_0
- set of final states F
- input alphabet S (also define Σ^*, Σ^+)
- output alphabet D
- transition function $\delta : Q \times \Sigma \rightarrow 2^Q$
- output function $\sigma : Q \times \Sigma \times Q \rightarrow D^*$

6.863J/9.611J SP03 Lecture 2

Regular relations on strings

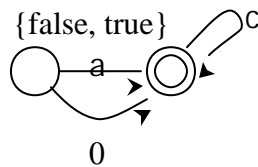
- *Relation*: like a function, but multiple outputs ok
- *Regular*: finite-state
- *Transducer*: automaton w/ outputs



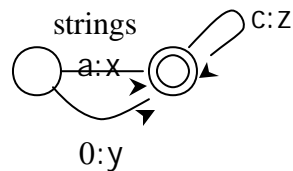
6.863J/9.611J SP03 Lecture 2

The difference between (familiar) fsa's and fst's: functions from...

Acceptors (FSAs)



Transducers (FSTs)



6.863J/9.611J SP03 Lecture 2

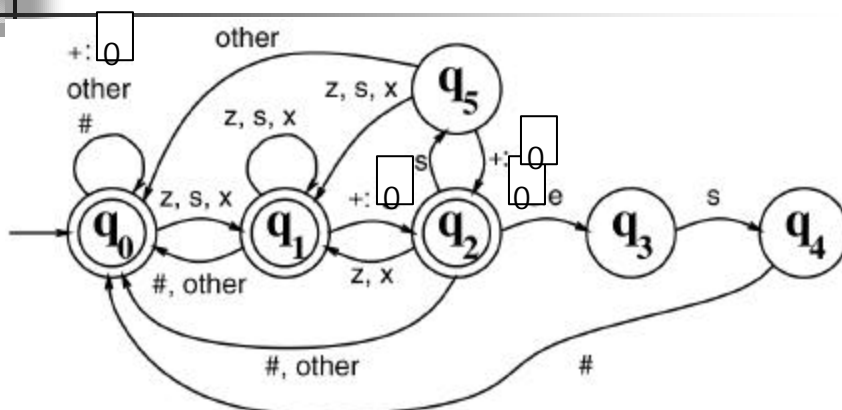
Defining an fst for a spelling-change rule

- Suggests all we need to do is build an fst for a spelling-change rule that 'matches' lexical and surface strings
- Example: fox+s, foxes; buzz+s, buzzes
- Rule: Insert e before non initial x,s,z
- Instantiation as an fst (using PC-Kimmo notation)

<u>f</u>	o	x	0	e	s	#	surface
F	O	X	+	0	S	#	lexical

6.863J/9.611J SP03 Lecture 2

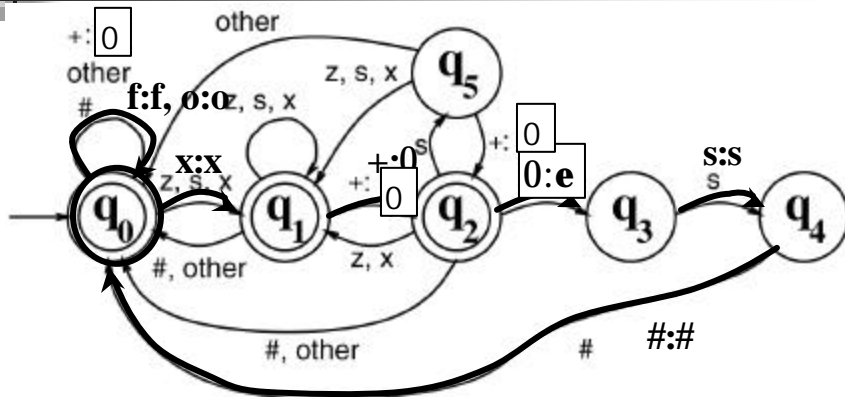
Insert 'e' before non-initial z, s, x ("epenthesis")



<u>f</u>	o	x	0	e	s	#	surface
F	O	X	+	0	S	#	lexical

6.863J/9.611J SP03 Lecture 2

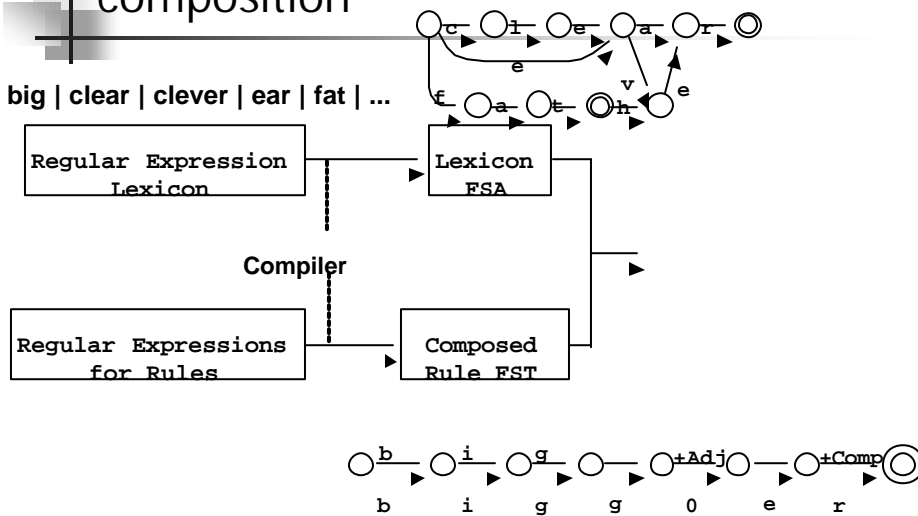
Successful pairing of foxes, fox+s



f	o	x	0	e	s	#	surface
F	O	X	+	0	S	#	lexical

6.863J/9.611J SP03 Lecture 2

Now we *combine* the fst for the rules and the fsa for the lexicon by composition



6.863J/9.611J SP03 Lecture 2

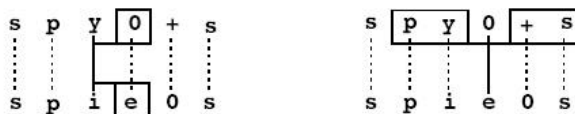
So we're done, no?

- ✓ Which units can glue to which others (roots and affixes) (or stems and affixes)
- ✓ What 'spelling changes' (orthographic changes) occur – like dropping the e in 'chase + ed'

6.863J/9.611J SP03 Lecture 2

So, we're done, right?

- Not so fast...!!!!
- Sometimes, more than 1 spelling change rule applies. Example: spy+s, spies: y
- y goes to *i* before an inserted *e* (compare, "spying")
- *e* inserted at affix +s



6.863J/9.611J SP03 Lecture 2

Simultaneous rules

- All we gotta do is write one fst for each of the spelling change rules we can think of, no?
- Since fsa's are *closed* under intersection, we can apply all the rules simultaneously... can we?
- No! Fst's cannot, in general, be intersected... (but, they can, under certain conditions...)

6.863J/9.611J SP03 Lecture 2

The classical problem

- Traditional phonological grammars consisted of a *cascade* of general rewrite rules, in the form:
 $x \rightarrow y / \phi _ \gamma$
- If a symbol x is *rewritten* as a symbol y , then afterwards x is no longer available to other rules
- *Order* of rules is important
- Note this system is *Turing complete* – can simulate general steps of any computation.. So, gulp, how do we cram them into finite-state devices...?

6.863J/9.611J SP03 Lecture 2

Example from English ("gemination")

underlying

quiz + s



Rule A: s -> es after z

intermediate

quiz + es



Rule B: z doubles before
Suffix beginning with
vowel

surface

quizzes

6.863J/9.611J SP03 Lecture 2

What's the difference?

- FSA isomorphic to *regular languages* (sets of strings)
- FST isomorphic to *regular relations*, or *sets of pairs of strings*
- Like FSAs, closed under union, *but unlike* FSAs, FSTs are *not* closed under complementation, intersection, or set difference

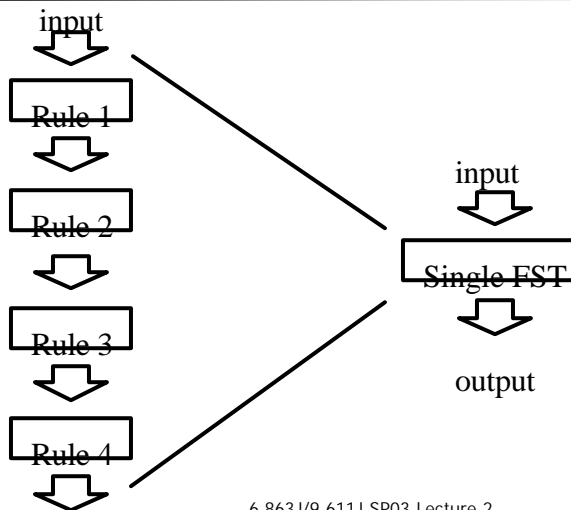
6.863J/9.611J SP03 Lecture 2

But this is a problem...

- How do we know which order of rules?
- A transducer merely computes a static *regular relation*, and is therefore inherently reversible – so equally viable for analysis or synthesis
- The constraints are *declarative*
- Since the rules describe such *relations*, in general, more than one possible answer – which do we pick? (Inverting the order becomes hard)
- This blocked matters until C. Johnson recalled a theorem of Schutzenberger [1961] viz.,

6.863J/9.611J SP03 Lecture 2

When is this possible?



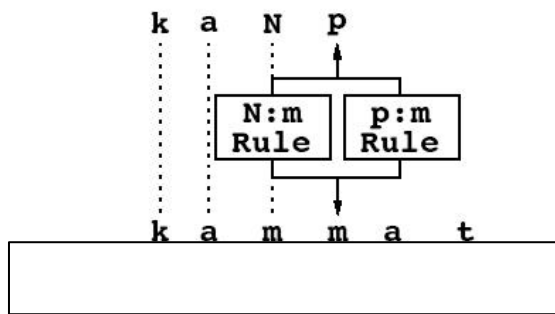
6.863J/9.611J SP03 Lecture 2

Schutzenberger's condition on closure of fst's

- The relations described by the individual transducers add up to a regular relation (I.e., a single transducer) when considered as a whole *if*
- The transducers act in lockstep: each character pair is seen simultaneously by all transducers, and they must all "agree" before the next character pair is considered
- No transducer can make a move on one string while keeping the other one in place unless all the other transducers do the same

6.863J/9.611J SP03 Lecture 2

Simultaneous read heads



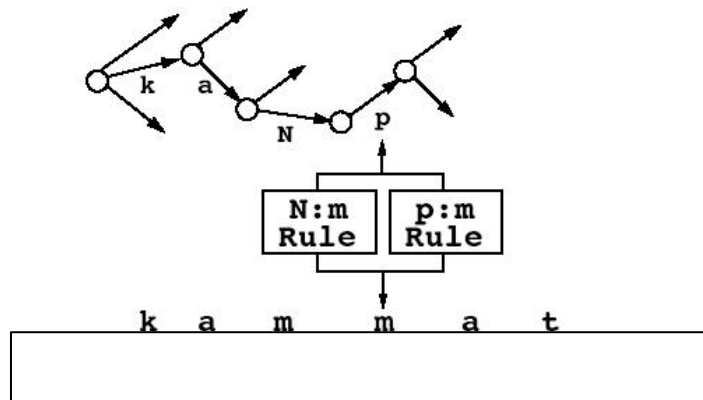
6.863J/9.611J SP03 Lecture 2

The condition

- For FSTs to act in lockstep, any 0 transitions must be synchronized – that is, the lexical/surface pairing must be *equal length*
- S. called this an equal length relation
- Under this condition, fst's *can* be intersected – PC-Kimmo program simulates this intersection, via simultaneous “read heads”

6.863J/9.611J SP03 Lecture 2

Plus lexicon – lexical forms always constrained by the path we're following through the lexicon tree



6.863J/9.611J SP03 Lecture 2

And that's PC-Kimmo, folks... or "Two-level morphology"

- A lexicon tree (a fsa to represent the lexicon)
- A set of (declarative) lexical/underlying relations, represented as a set of fst's that address *both* lexical and surface forms
- For English, roughly 5 rules does most of the work (you've seen 2 already) – 11 rules for a "full scale" system with 20,000 lexical entries (note that this typically achieves a 100-fold compression for English)
- The only remaining business is to tidy up the actual format PC-KIMMO uses for writing fst tables (which is quite bizarre)

6.863J/9.611J SP03 Lecture 2

Spelling change rules

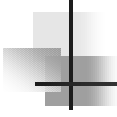
Name	Description	Example
Consonant Doubling (gemination, G)	1-letter consonant doubled before <i>-ing/ed</i>	beg/begging
E deletion (elision, EL)	Silent <i>e</i> dropped before <i>-ing, -ed</i>	make/making
E insertion (epenthesis, EP)	<i>e</i> added after <i>-s, -z, -ch, -sh</i> before <i>-s</i>	fox/foxes
Y replacement (Y)	<i>-y</i> changes to <i>-ie</i> before <i>-ed</i>	try/tries
I spelling (I)	<i>I</i> goes to <i>y</i> before vowels	lie/lying

6.863J/9.611J SP03 Lecture 2



How do we write these in PC-Kimmo?

6.863J/9.611J SP03 Lecture 2



PC-Kimmo 2-level Rules

- Rules look very similar to phonological rewrite rules, but their semantics is entirely different
- 2-level rules are completely declarative. No derivation; no ordering
- Rules are in effect modal statements about how a form can, must, or must not be realized

6.863J/9.611J SP03 Lecture 2

Form & Semantics of 2-level Rules

- Basic form is
L:S OP lc ... rc:
- Lexical L pairs with surface S in (optional) left, right context lc, rc. OP is one of
 - => Only but not always,
 - <= Always but not only
 - <=> Always and only
 - /<= Never
- lc and rc are 2-level i.e. can address lexical and surface strings

6.863J/9.611J SP03 Lecture 2

a:b => l_r

- If the symbol pair a:b appears, it must be in context l_r
- If the symbol pair a:b appears outside the context l_r, FAIL

lar lar lbr ~~xy~~
lbr lar lbr ~~xy~~

6.863J/9.611J SP03 Lecture 2

Example: epenthesis

; LR: fox+0s kiss+0s church+0s spy+0s

; SR: fox0es kiss0es church0es spi0e

(note: we NEED the + to mark the end of the root 'fox' – we can't just have fox0s paired with fox0es)

RULE "3 Epenthesis, 0:e => [Csib|ch|sh|y:i] +:0__s [+:0|#]" 7 9

6.863J/9.611J SP03 Lecture 2

$a:b \leq l_r$

- If lexical a appears in context l_r , then it must be realized as surface b
- If lexical a appears in context l_r , if it is realized as anything other than surface b , FAIL

~~lar lar lbr xay~~
~~lbr lar lbr xby~~

6.863J/9.611J SP03 Lecture 2

Y-I spelling

; y:i-spelling
; LR: spy+s happy+ly spot0+y+ness
; SR: spies happi0ly spott0i0ness

RULE "5 y:i-spelling, y:i <= :C__+:0 ~[i!]" 4 7

6.863J/9.611J SP03 Lecture 2

$a:b \Leftrightarrow l_r$

- If the symbol pair $a:b$ appears, it must be in context l_r
- If lexical a appears in context l_r , then it must be realized as surface b
- If the symbol pair $a:b$ appears outside the context l_r , FAIL
- If lexical a appears in context l_r , if it is realized as anything other than surface b , FAIL

~~lar lar lbr xay~~
~~lbr yar lbr xby~~

6.863J/9.611J SP03 Lecture 2

Possessives with 's'

; s-deletion

; LR: cat+s+'s fox+s+'s

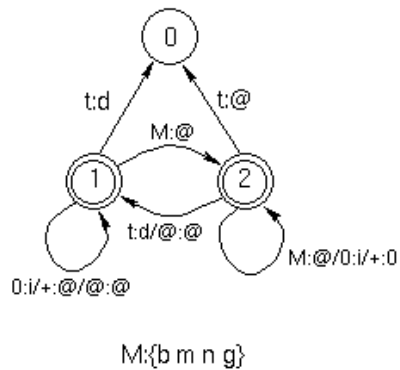
; SR: cat0s0'0 foxes0'0

RULE "7 s-deletion, s:0 <=> +:0 (0:e) s +:0 ' ___"

6.863J/9.611J SP03 Lecture 2

Example: Japanese past tense

• *Voicing*: t:d <=> <b m n g>: (+:0) (0:i) ___



6.863J/9.611J SP03 Lecture 2

a:b <= /l_r

- Lexical a is never realized as b in context l_r
- If lexical a is realized as b in the context l_r, FAIL

~~lar~~ lar lbr xay
~~lbr~~ lar lbr xby

6.863J/9.611J SP03 Lecture 2

Gemination (consonant doubling)

; {C} = {b,d,f,g,l,m,n,p,r,s,t}

RULE "16 Gemination, 0:0 /<= `:0 C* V {C}__+:0 [V|y:]" 5 16

6.863J/9.611J SP03 Lecture 2

2-Level Rule Semantics: summary

<code>a:b <=> l _ r;</code>	lar lar lbr xay lbr lar lbr xby	lexical surface
<code>a:b <= l _ r;</code>	lar lar lbr xay lbr lar lbr xby	
<code>a:b => l _ r;</code>	lar lar lbr xay lbr lar lbr xby	
<code>a:b /<= l _ r;</code>	lar lar lbr xay lbr lar lbr xby	

6.863J/9.611J SP03 Lecture 2

Automata Notation (.rul file)

- What were those funny 2 numbers at the end of the 'rewrite' notation?
- They specify the rows and columns of the corresponding automaton
- I'll show you one, but it's like Halloween 6 – a nightmare you don't want to remember
- We have a nicer way of writing them...
- OK, here goes...

6.863J/9.611J SP03 Lecture 2

Shudder...

RULE "16 Geminata, 0:0 /<= `:0 C* V {C}___+:0 [V|y:]" 5 16
` V y b d f g l m n p r s t + @
0 V @ b d f g l m n p r s t 0 @
1: 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
2: 2 4 2 2 2 2 2 2 2 2 2 2 2 2 1 2
3: 2 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
4: 2 1 1 5 5 5 5 5 5 5 5 5 5 5 1 1
5: 2 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1

6.863J/9.611J SP03 Lecture 2

Limits?

- Can PC-KIMMO do INFIXES?

Infix:

Tagalog: *um+hinigi* → ***humingi*** (borrow)

Any infixes in 'nonexotic' language like English?

Here's one: *un-f*****-believable*

6.863J/9.611J SP03 Lecture 2

Summary: what have we learned so far?

- FSTs can model many morphophonological systems - esp. concatenative (linear) phonology
- You can compose and parallelize the FSTs
- Nulls cause nondeterminism - why can't we get rid of nondeterminism like in FSAs
- What can this machine do?
- What can't it do?
- How complex can it be? (computational complexity in official sense)
- How complex is it in practice?
- Example from Warlpiri

6.863J/9.611J SP03 Lecture 2

Lab 1: PC-kimmo warmup

Login to Athena *SUN* workstation

```
Athena> attach 6.863
```

```
Athena> cd /mit/6.863/pckimmo-old
```

```
Athena> pckimmo
```

```
PC-Kimmo> take english
```

```
PC-Kimmo> recognize flies
```

```
  `fly+s fly+PL
```

```
...
```

```
PC-Kimmo> generate fly+s
```

```
  flies
```

```
PC-Kimmo> set tracing on
```

```
PC-Kimmo> quit
```

6.863J/9.611J SP03 Lecture 2

An example – try it yourself

6.863J/9.611J SP03 Lecture 2

Outfoxed? Off to the races...

Trace of an example *races'*

- The machine has to dive down many paths...

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.0) --> (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

6.863J/9.611J SP03 Lecture 2

More to go...

Problem: *e* was 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
```

6.863J/9.611J SP03 Lecture 2

And still more maze of twisty passages, all alike...it's going to try all the sublexicons w/ this bad guess..

6.863J/9.611J SP03 Lecture 2

Winding paths...after 22 steps...

```
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)
5      (+.0) --> (1 1 16 1 15 1)
6      (s.s) --> (1 4 16 2 1 1)
7      Entry +/s ends--> new lexicon C2, (1 4 16 2 1 1)
8      Entry /0 is word-final -->rejected(leftover input)
8      ('.') --> (1 1 16 1 1 1)
9      End --> lexical form ("race+s'" (N PL GEN))
```

6.863J/9.611J SP03 Lecture 2

The End

6.863J/9.611J SP03 Lecture 2