6.863J Natural Language Processing
Lecture 4: My Fair Lady Lecture

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

My Fair Lady

- "Words, words, words, I'm so sick of words I get words all day through; First from him, now from you…" - Eliza Doolittle

The Menu Bar

- Administrivia

  Lecture 3 posted; Lab 1a (aka "component II") due today; Lab 1b, due next Monday
- Kimmo & Laboratory 1b: how-to
- Postmortem: Complexity of Kimmo/fst's – too weak? Too strong? What makes a good computational linguistics representation? A good linguistic representation? A good algorithm?
- Alternatives: morphology w/o a dictionary

Why do we care?

- We need to recover information about root + affixes even for simple IR (though this has been questioned)
- We need information for later analysis…

Morphology: why do we need it for language analysis?

- Inflectional Morphology:
  - Agreement-features (person, number, gender)
    - Examples: movies, blonde, actress
    - Irregular examples: appendices, geese
  - Case
    - Examples: he/him, who/whom
  - Comparatives and superlatives
    - Examples: happier/happiest
  - Tense
    - Examples: drive/drives/drove (-ed)/driven

Morphology

- Derivational Morphology
  - Nominalization
    - Examples: formalization, informant, informer, refusal, lossage
  - Deadjectivals
    - Examples: weaken, happiness, simplify, formalize, slowly, calm
  - Deverbals
    - Examples: *see nominalizations*, readable, employee
  - Denominals
    - Examples: formal, bridge, ski, cowardly, useful

Part of the English Tense System

	Present	Past	Future	Infinitive
(basic)	eats	ate	will eat	to eat
Perfect	has eaten	had eaten	will have eaten	to have eaten
progressive	is eating	was eating	will be eating	to be eating
Perfect+ progressive	has been eating	had been eating	will have been eating	to have been eating

Morphology and Semantics

- Suffixation
  - Examples:
    - *x* employ *y*
      - employee: picks out *y*
      - employer: picks out *x*
    - x read y
      - readable: picks out y
- Prefixation
  - Examples:
    - undo, redo, un-redo, encode, defrost, asymmetric, malformed, ill-formed, pro-Chomsky

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.

Laboratory 1b – remaining details

- What phenomena you're covering
- How to build spelling-change fsa's - details
- How to build morpheme automaton - details

The phenomena

- 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 & 2 morpheme patterns

The phenomena

Spelling changes:

1. *g-j mutation*
2. *z-c mutation*
3. *Pluralization*

Morpheme automaton:

Noun endings

Verb conjugation - 1 form

Phenomenon 2: z-c mutation

- *z-c mutation*

  $z \rightarrow c$ before front vowels, *z* otherwise

  cruzar (to cross); *cruzo, 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*]

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
- Thinking in terms of constraints (what is ruled out by the rule) is the most difficult 'mindset' to attain…

Build automaton for lex, surface pairs

- But what are the lexical pairs?
- Ah, your job!
- Trying pairings – not generally the infinitive, e.g.

  cruzar, cruzamos → legit pair?

  `cruzar`

  `cruzamos`

Look at the other pairs – what do you think the root is?

Writing rules

- **`cruzar/cruzamos  cruzar/cruce ?`**
- We can try a (tentative) lexical/surface pair, and from that extract the right spelling change
- Do it step by step: use the alignment to write down the 'straight-line' acceptance path:

  `cruz`

  `cruce`

Pad out length by using 0's (nulls) (why important)?

`cruz0` **`cruz0`**

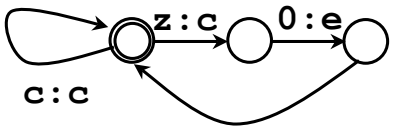
`cruce` **`cruzo`**

Outline context – hmm, perhaps we do need root?

Writing rules

From context to rule:

cruz0,cruce c:c, r:r, u:u, z:c, 0:e - accept



c:c
r:r,
u:u,…

cruz+ cruz+
cruce cruzo

But… is this the correct root?

Some format details

- For the automata: the **.rul** file:

```
ALPHABET
a ^ b c C d e < f g . . .
NULL 0
ANY @
BOUNDARY #
RULE "Default characters" 1 33
    a ^ c d e …  z
1:  a ^ c d e …  z   ; WHY Needed?

RULE "z goes to c" 3  4
    @ z  +  +
    @ c  e  o
1:  1 2  0  0
2.  ? 3  0  0
3.  1 2  1  1
RULE "PLURALIZE" n m
 <automaton table>
```

Instead of writing fst tables…

- You can use the program fst
- To run:
  build fst type rules in file spanish.fst, then

```
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

- *"z before high vowel to c"*

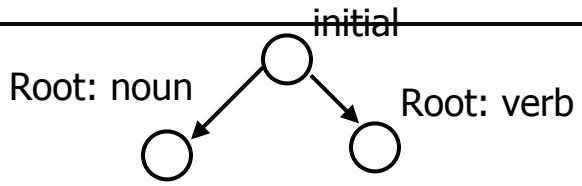
```
subset hivowel  e i
machine "ztoc"
state one
z:hivowel   two
c:c  one
z:z  one
others reject

state two
+:e three
+:0 reject
…
others one
```

Design of morpheme machine

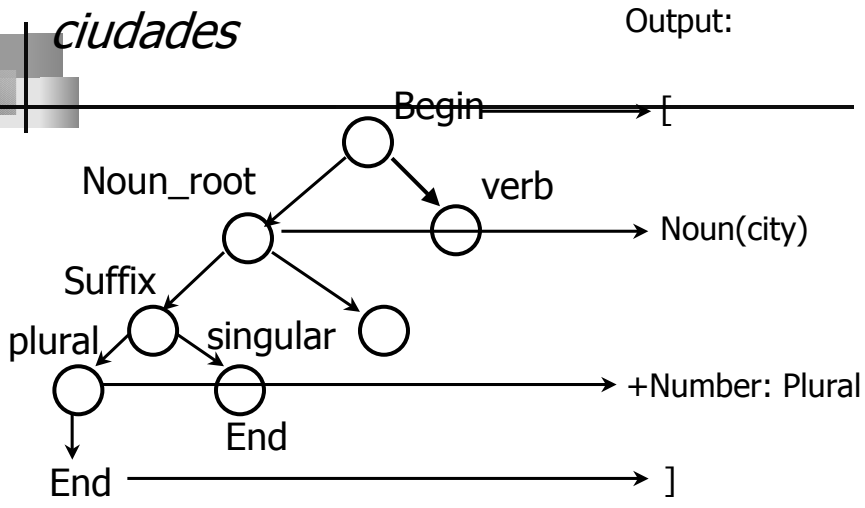
- One big fsa, that handles two phenomena: plurals and verb endings

Automaton design for lexicon



Q: what do we need to add to noun sequence?

The morpheme tree: Adding plurals - *ciudades*



Final output: [Noun(city)+Number: Plural]

The lexicon – take 2

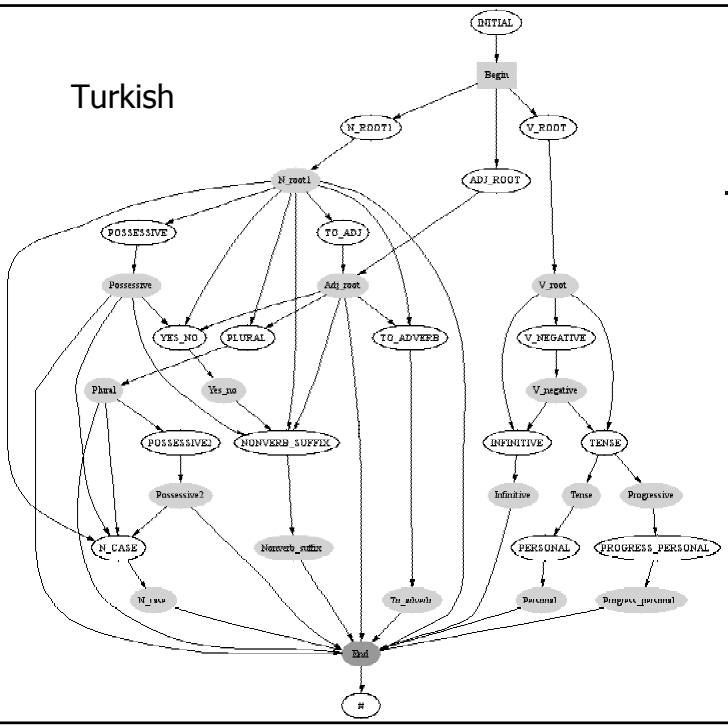
- You will deal with two types of 'endings'

1. Noun endings: plural suffix **`+s`**
2. Verb endings: verb stem + tense markers

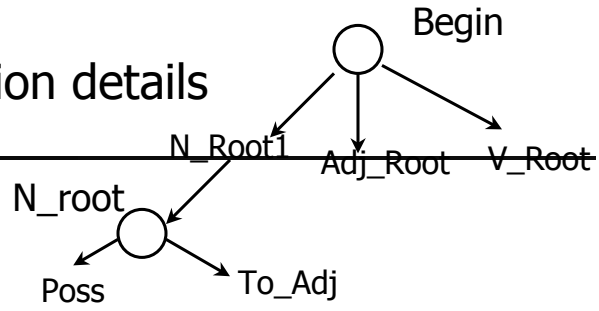
   Simplest: infinitive marker **`+ar, +er, +ir`**

   See table in lab file for details: 5 x 3 table for Present tense; ditto for Subjunctive tense ("*I might….*")

Turkish



Specification 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

…

Specifying transition arcs

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

Transitions from

State	next-state	Gloss (output)
N_root1:		
kol	N_root	Noun('arm')
kitab	N_root	Noun('book')

The End

End:

0 # " "

Final output is concatenation of all the outputs along the path, eg,:

[Noun(arm)+plural]

Kimmo: its use and abuse or: Post-mortem

- Criteria to evaluate: scientific, engineering
- Scientific: is this a sufficient representation to cover the linguistic possibilities?
- Is this a necessary representation: does it appropriately represent space of possibilities? (all and only the natural morphophonological rule systems)
- Engineering/computational: what is its computational power? Is it strong enough? Is it too strong?
- How well does it work in practice?
- Is there an alternative?

Outline: the Use and Abuse of Kimmo

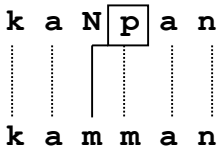
Kimmo: what is it good for?

- How we return features for parsing
- What can it do? – A longer example of rule ordering
- What can't it do
- What's its computational complexity?
- Morphology w/o a dictionary? The Porter algorithm
- Learning morphology – Goldsmith
- On to pr's and stat. Lang.

Criterion 1: linguistic adequacy

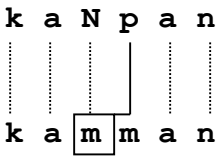
- Is Kimmo sufficient?
- Classic rule systems: ordered sets of rewrite rules
- Can Kimmo do these? (Kimmo rules are unordered)

Constraints on both sides



N:m correspondence requires a following p on the lexical side.

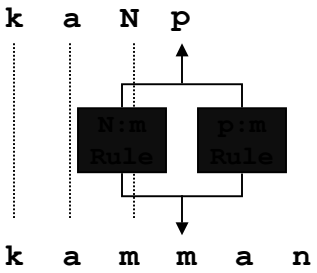
In this context, all other possible realization of a lexical N are prohibited.



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

k a N p a n

N -> m / _ p

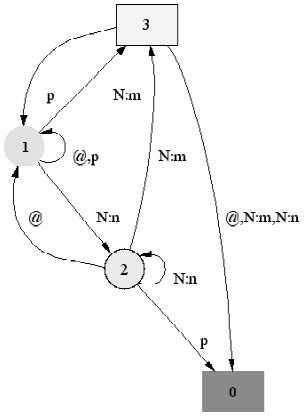
k a m p a n

p -> m / m _

k a m m a n

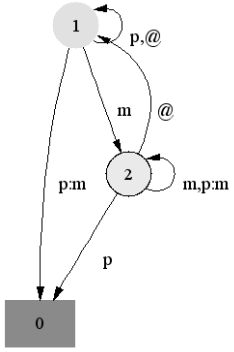
Machine Rule 1 ("N goes to m")

Rule 1: N→m | __ p

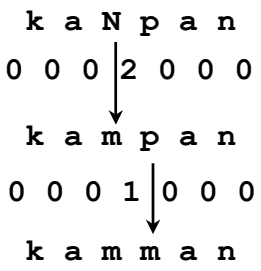


Machine Rule 2 ("p goes to m")

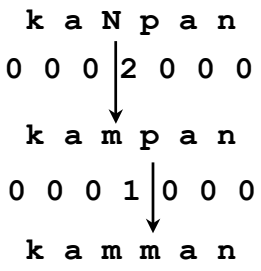
Rule 2: p→ m | m___

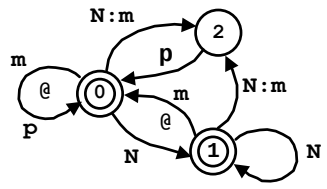


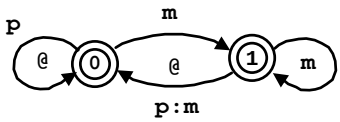
Sequential Application in Detail



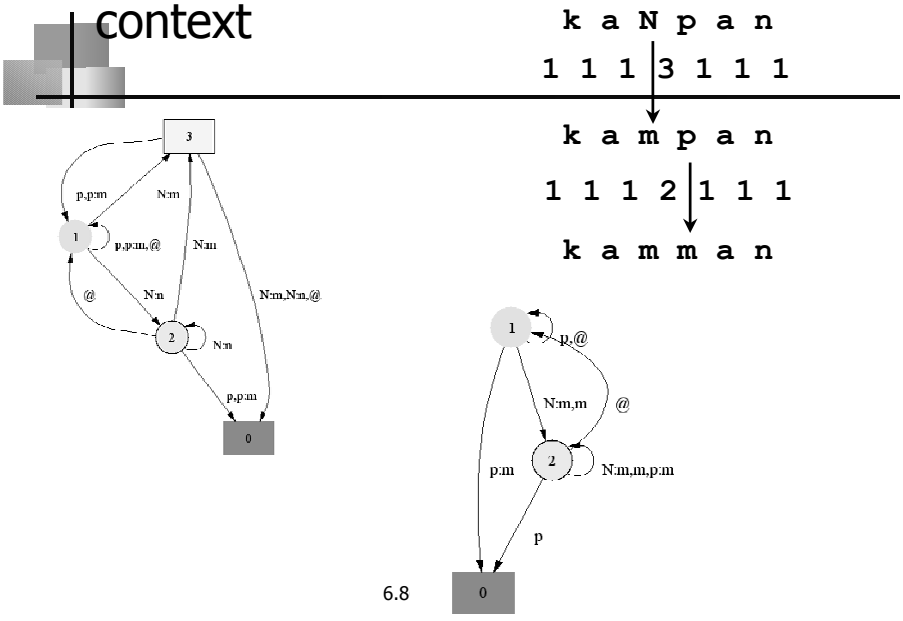
Sequential Application in Detail







Rules take into acct each other's context



What constraint do we need for this 'parallel' approach to work?

- Machines must act in lockstep (sequentially locked) – o.w., won't be looking at the same character at the same time
- "Equal length" constraint:
- Pad out lexical, surface strings s.t. they are of the same length (we'll see why in a moment)
- Example: consider our 4 ordered rule case…

4 ordered rules – classic case

- LR: a+ked re+ked a+sin re+sin
  SR: akseyd reseyd assayn rezayn
- Rule 1: Duplication- Cons → Cons Cons | æ + _
- Rule 2: s to z - s → z | _ Vowel
- Rule 3: k to s - k → s | Vowel _ Vowel
- Rule 4: Vowel shift i → ay, e → ey, …
- Q: can we do this in Kimmo??

4 ordered rules

- LR: a+ked re+ked a+sin re+sin
  SR: akseyd reseyd assayn rezayn

Pad out so LR and SR of equal length, also noting +:0 correspondence

a+ked	**re+ked**	**a+sin**	**re+sin**
aksed	**re0sed**	**assin**	**re0zin**

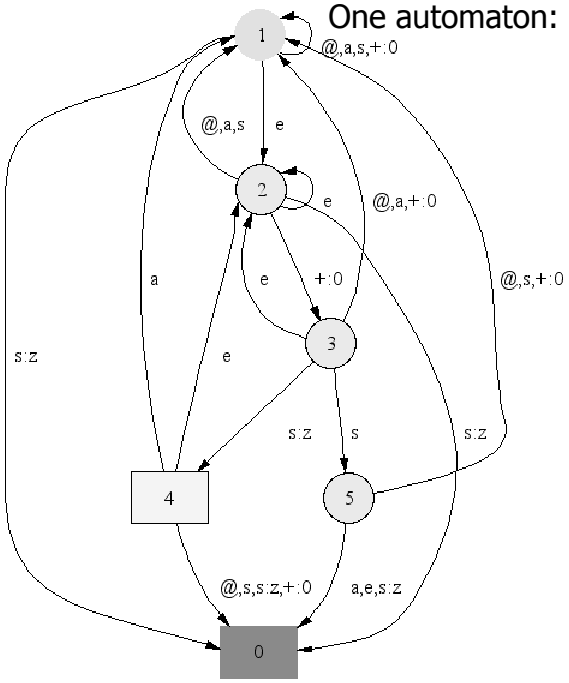
Extract contexts

a+ked	re+ked	a+sin	re+sin
aksed	re0sed	assin	re0zin

Rule: +:k ⇔ a:a __ k:s
+:s ⇔ a:a __ s:s

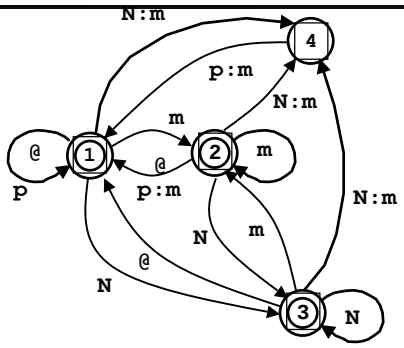
Rule: k:s ⇔ e +:0 __ V | +:k __ V
+:s ⇔ a:a __ s:s

One automaton: s-to-z

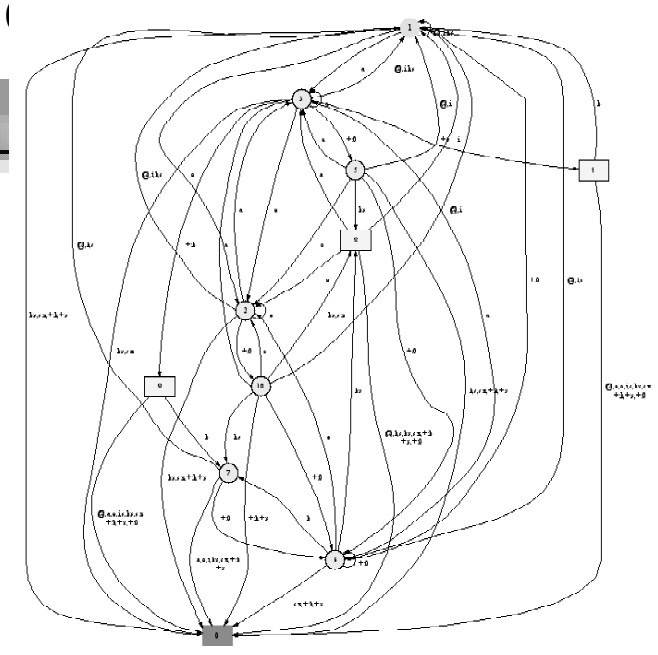


Method 2 - Retain order: Composition

```
k a N p a n
1 1 1 4 1 1 1
k a m m a n
```



FTNs ARE closed under composition



Is Kimmo sufficient?

- Ideally: *yes, if* locally, purely concatenative phenomena (obviously, because fsa's)
- FSAs are based *purely* on an associative concatenation operation over strings (i.e., ((a+b)+c) ≡(a+(b+c)) where + denotes concatenation
- Antidisestablishmentarianism
- Turkish word: uygarlas,tiramadiklarimizdanmis,sinizcasina =

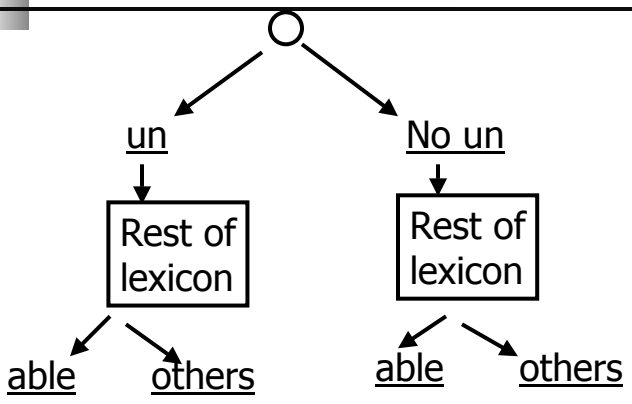
  uygar+las,+tir+ama+dik+lar+imiz+dan+mis,+siniz+casina

(behaving) as if you are among those whom we could not cause to become civilized

Is Kimmo sufficient?

- So, this lets us think what the system *might not* be good for… let's look at English first….
  - There seem to be some kinds of 'long distance' constraints…
  - Prefix/suffix links: only some prefixes tied to some suffixes
    - Un---------able
    - Undoable, uncanny, ?uncannyable, unthinkable, thinkable, readable, unreadable, unkind, *unkindable
  - So, we have to 'keep track' that the *un* is first or *not* – what does lexicon look like?

Lexicon must be (grotesquely) duplicated



Similar example of 'long distance' constraint

- French elision: le, la: l'arbe; l'homme
- Always put in front, elided if noun/adj begins w/ a vowel
  - However, blocked if noun is plural: *l'arbes, les arbes

This kind of duplication is a litmus test of something wrong

- Duplication: no relation between the two lexicons, but we know they're identical
- Principle *AWP*
- We will see this again and again
- Usually means we haven't carved (factored) the knowledge at the right 'joints'
- Solution? Usually more powerful machinery 'overlay' representations

Not *all* long distance effects are a barrier…

- Phenomena: *Vowel harmony*
  - yourgun + sInIz $\rightarrow$ yorgunsunuz
  - Round vowels assimilate to round vowels; back vowels to back, etc. - all the way from left to right
- Can Kimmo do it? What would be your model? Suppose harmony is right to left?

What about nonconcatenative L's?

- Semitic languages, eg, Arabic
- Intercalated consonants and vowels
- Root: k t b

Cons 'tier' C C C ⟶ CVCVC ("katab")
Vocalization: V V

Can we do this in Kimmo? (or in a linear system generally?)

Another example: Tagalog

Root	CV+root	Gloss
pili	pipili	'choose'
tahi	tatahi	'sew'
kuha	kukuha	'take'

What's going on? How to do in Kimmo?
What would you propose?

Need extensions

- Add multiple intersections to interdigitate: CCC^VV $\rightarrow$ CVCVC then go on from there…
- In general – more powerful machine
- Not yet completely explored

And finally…

- Is morphology really linear?
- Un[care –less] [uncare-less] – not really associative (cf 'dark blue sky')
- Somehow, we haven't captured possibly hierarchical structure – instead, shoehorned in

Verdict: is Kimmo sufficient?

- Not unless we add some hacks – in this sense, it is too weak
- OK, onto question 2

Is Kimmo necessary? Why the 'equal length' constraint?

- Zeroes (null elts) must be limited or else…
- Unlimited expansion $\rightarrow$ no longer a finite state (regular) i/o relation (in fact, Turing complete)
- (Thm: if input-output relation is not bounded by any size of the input, then it could run arbitrarily long…)
- Can no longer guarantee that you can represent this as a new FTN (or more…)
- Hints that power here is not necessary

Is Kimmo necessary?

- Does it explain why many non-human systems never occur (ruling them out)
- Or does it overshoot?

- Ans: it seems to overshoot, in at least 2 ways
- Overshoots detected by computational analysis

The power of Kimmo – part 1

- More powerful than well-known grammars in linguistics (and computational linguistics)
- We can use kimmo to 'count' – but natural languages don't do this…
- (Recall: we can use Kimmo to output a language with one counting relation: $a^n b^n$ – not a finite-state language)
- But we can do more… nothing stops us from producing a language with *m* counting relations, e.g, for any *n*, $\{(x, (cx)^n) \mid x \in \{a^* b^*\}\}$, e.g., for n=3, cababcababcabab, cbbbcbbbcbbb…

Kimmo admits more than context-free languages

- Fact: context-free languages can never define more than one counting dependency
- (Intuition: they use a stack for this – can only push and pop to match)
- So Kimmo is more powerful than this!

(still, might be ok – can parse these in cubic time)

- How powerful is it?
- Conjecture: as powerful as all the context-sensitive languages (even given limited erasing)

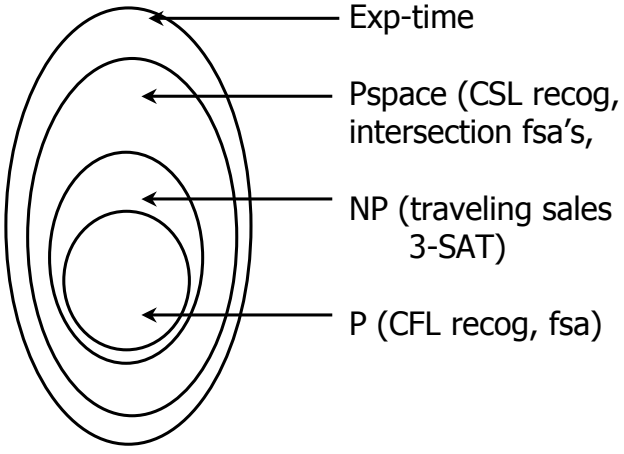
Complexity of Kimmo word recognition

- All these finite-state devices, working in parallel
- There is backup
- Is it intrinsic to the system? Or eradicable? Or, doesn't matter in practice?

Litmus test #2 – computational complexity of Kimmo – word parsing is intractable!

- Kimmo Recognition Problem (KRP):
  Given a language defined by an arbitrary (finite) Kimmo dictionary (lexical automata) and a finite set of Kimmo rules, how long in the worst case will it take to recognize whether a form is or is not in the language?
- Kimmo recognition problem is NP-hard
- As hard as any other problem solvable by a nondeterminstic Turing machine in polynomial time
- No known det polytime (eg, cubic) algorithm for NP-hard problems…

Complexity hierarchy



Parsing words with Kimmo is computationally intractable

- Intuition: what if the characters on the surface don't give any clues as to what 'features' they ought to have underlyingly? (e.g., whether a Noun or a Verb, as in *police police police*)
- This seems awfully close to the famous 3-SAT problem: is there an assignment of T(rue), F(alse) to the literals of an arbitrary Boolean formula in 3-conjunctive normal form s.t. the formula evaluates to *true*?
- In fact, we can simulate this problem using Kimmo

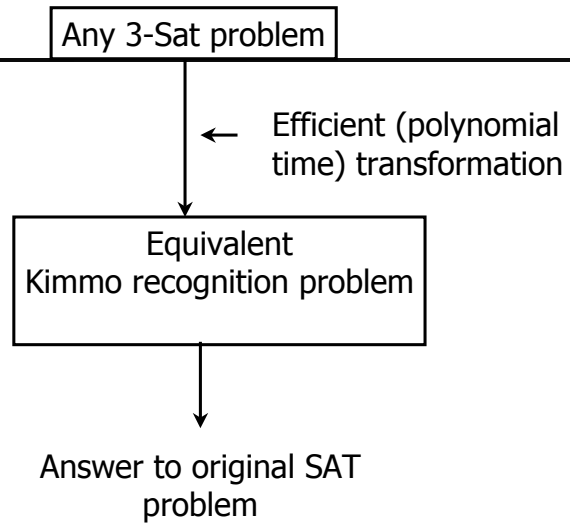
3-Sat (3-satisfiability) is NP-complete

- *Given* (arb) 3-Sat formula, e.g.,
- There is no known deterministic Turing machine that can figure out quickly (in polynomial time) whether there is an assignment of *true* or *false* to literals *x,y, z* in order to make the formula evaluates to true just by inspecting the local surface string
- We could guess this in polynomial time – i.e., Nondeterministic Polynomial, or NP time (time measured in length of the formula)

Reduction of 3-Sat to Kimmo recognition problem

- For every 3-Sat problem, we can find, in polynomial time, a corresponding Kimmo word recognition problem where there's a valid word if the 3-Sat problem was satisfiable
- If Kimmo recognition could be done in deterministic polynomial time (P) then so could 3-SAT

Reduction

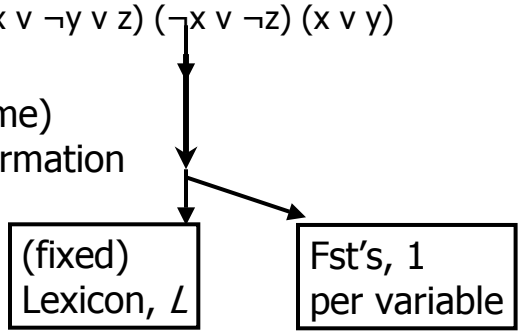


The reduction:

Given: arbitrary 3-SAT problem instance, e.g.,

(x v ¬y v z) (¬x v ¬z) (x v y)

Fast
(polytime)
transformation



word∈L if Sat instance satisfiable

If we could solve Kimmo recognition easily,
Then we could solve 3-Sat easily

Two components to 3-Sat

- The fact that an *x* that has a truth assignment in one place, must have the same truth assignment everywhere - what morphological process is that like?
- The fact that every triple must have at least 1 'T' underlyingly (so that the triple is true) - what morphological process is that like?

How the reduction works

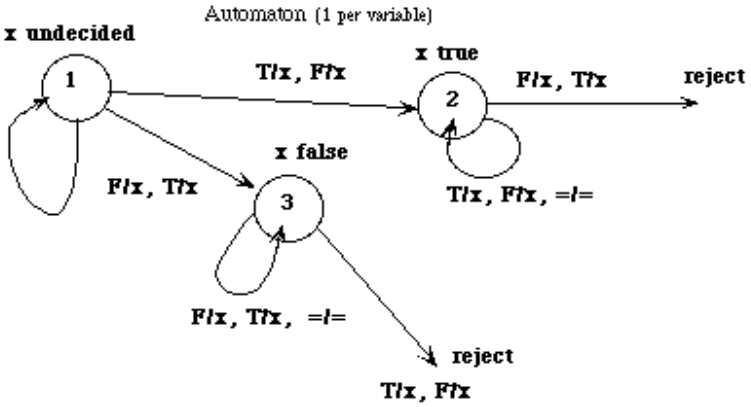
- Given arbitrary 3-sat formula ϕ, e.g.,
  (x v ¬y v z) (¬x v ¬z) (x v y)
- Represent in the form, a 'word':
  x-yz,-xz,xy

- For each variable x, we have an 'assignment machine' that ensures that x is mapped to T or F throughout the whole formula
- We have one machine (and a fixed dictionary) to checks each disjunction to make sure that at least one disjunct is true in every conjunct

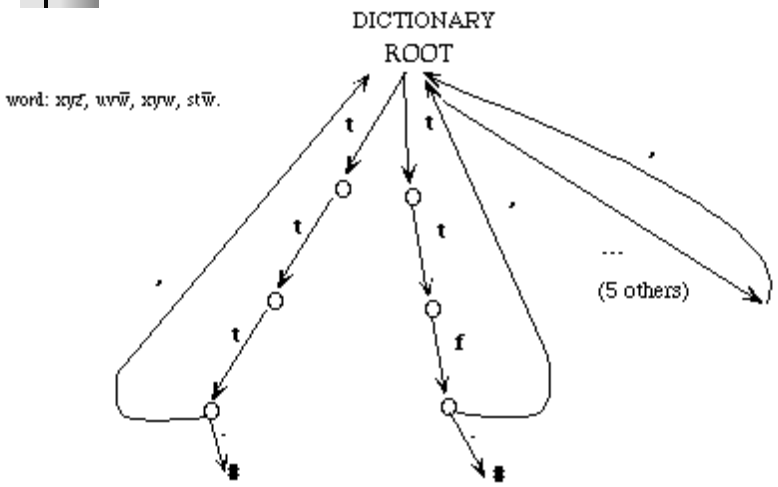
Two components

- Agreement: vowel harmony (if round at some point, round everywhere)
- Ambiguity: we can't tell what the underlying value of *x* is from the surface, but if there's at least one "t" per 'part of word', then we can spell out this constraint in dictionary
- Note that words (like Sat formulas) must be arbitrarily long… (pas de probleme)
- Dictionary is fixed…
- # of Vowel harmony processes corresponds to # of distinct literals

Reduce until done: assignment consistency



Reduce until done – formula must eval to *true*



What are the implications?

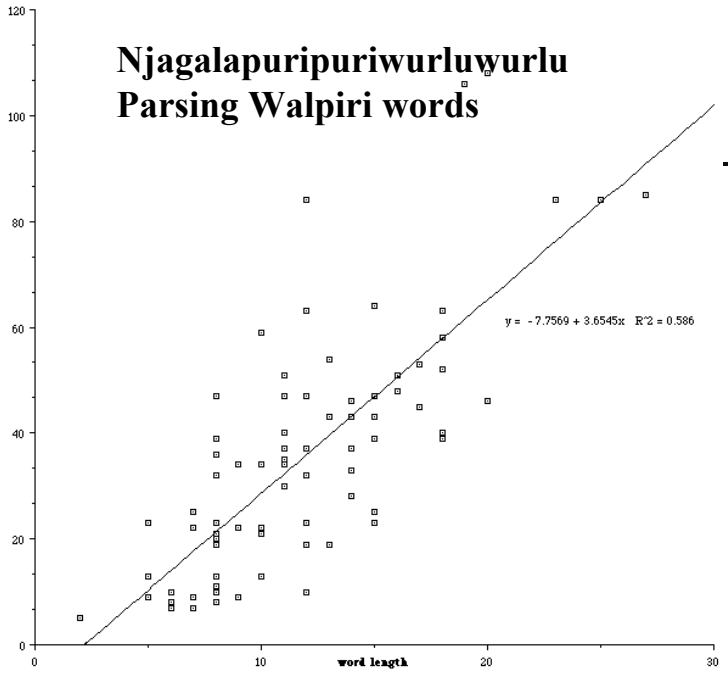
- FTNs inherently require backup if simulated (in the worst case) – Kimmo at least NP-hard (proof later on)
- Empty elements cause computational complexity (unless restricted – equal length condition) – true in all areas of linguistics
- Composition can save us, but then rule ordering must be watched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

Implications

- Do we need a machine powerful enough to represent intractable problems?
- No evidence for unbounded # of counting dependencies or harmony processes…
- Performance? Or do we need something this powerful??

Why should we care?

- This is *typical* of a combination of 'agreement and ambiguity' that trickles through all of natural language
- The agreement part – like Turkish vowel harmony
- The ambiguity part – like the *police police police* example
- Suggests that speed won't come from the formalism *all by itself*



Words are fine – but we need more

Paradigmatic example for NLP

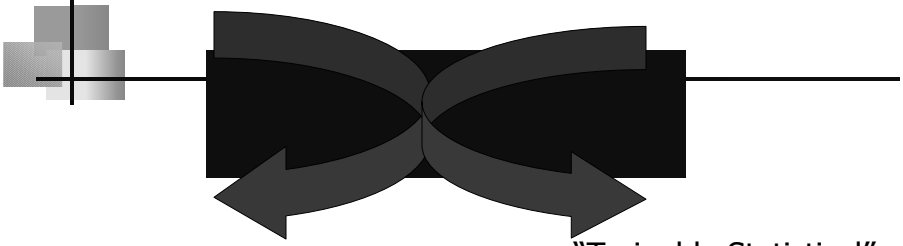
- Morphophonemic parsing
- Given surface form, recover underlying form:

Two ways

- Generative model – concatenate then fix up joints
  - stop + -ing = stopping, fly + s = flies
  - Use a cascade of transducers to handle all the fixups
- Probabilistic model - some constraints on morpheme sequences using prob of one character appearing before/after another

  prob(ing | stop) vs. prob(ly| stop)

The Great Divide in NLP:



"Knowledge Engineering" approach
Rules built by hand w/ K of Language
"Text understanding"

"Trainable Statistical" Approach
Rules inferred from lots of data ("corpora")
"Information retrieval"

the red pill or the blue pill?

What if we don't have a dictionary?

- Don't use one
- Learn one from data

Method 1: don't use a dictionary

- Best known method – Porter stemming (Porter, 1980)
- http://www.tartarus.org/~martin/PorterStemmer/ http://snowball.tartarus.org/
  - For English
  - Most widely used system
  - Manually written rules
  - 5 stage approach to extracting roots
  - Considers suffixes only
  - May produce non-word roots

Porter output

Sample Output (English):

consigned	consign	knack	knack
consignment	consign	knackeries	knackeri
consolation	consol	knaves	knavish
consolatory	consolatori	knavish	knavish
consolidate	consolid	knif	knif
consolidating	consolid	knife	knife
consoling	consol	knew	knew

Why?

Algorithmic stemmers can be fast (and lean):

E.g.: 1 Million words in 6 seconds on 500 MHz PC

- It is more efficient not to use a dictionary (don't have to maintain it if things change).
- It is better to ignore irregular forms (exceptions) than to complicate the algorithm (not much lost in practice).

Output - German

aufeinander	aufeinand	kategorie	kategori
auferlegen	auferleg	kategorien	kategori
auferlegt	auferlegt	kater	kat
auferlegten	auferlegt	katers	kat
auferstanden	auferstand	katze	katz
auferstehen	auferstand	katzen	katz
aufersteht	aufersteht	kätzchen	katzch

Method

Porter Stemmers use simple algorithms to determine which affixes to strip in which order and when to apply repair strategies.

Input	Strip -ed Affix	Repair
hoped	hop	hope (add -e if word is short)
hopped	hopp	hop (delete one if doubled)

Samples of the algorithms are accessible via the Web and can be programmed in any language.

Advantage: easy to see understand, easy to implement.

Stemming: Methods

- Dictionary approach not enough
  - Example: (Porter, 1991)
    - routed -> route/rout
      - At Waterloo, Napoleon's forces were routed
      - The cars were routed off the highway
    - Here, the (inflected) verb form is polysemous

Stemming: Errors

- Understemming: failure to merge
  - Adhere/adhesion
- Overstemming: incorrect merge
  - Probe/probable
    - Claim: -*able* irregular suffix, root: *probare* (Lat.)
- Mis-stemming: removing a non-suffix (Porter, 1991)
  - reply -> rep

Stemming: Interaction

- Interacts with noun compounding:
  - Example:
    - operating systems
    - negative polarity items
  - For IR, compounds need to be identified first…

Stemming: Porter Algorithm

- Rule format:
  - (condition on stem) suffix_1 -> suffix_2
  - In case of conflict, prefer longest suffix match
- "Measure" of a word is *m* in:
  - $(C)\ (VC)^m\ (V)$
  - C = sequence of one or more consonants
  - V = sequence of one or more vowels
  - Examples:
    - *tree* $C(VC)^0V$
    - *troubles* $C(VC)^2$

Stemming: Porter Algorithm

- Step 1a: remove plural suffixation
  - SSES -> SS (caresses)
  - IES -> I (ponies)
  - SS -> SS (caress)
  - S -> (cats)
- Step 1b: remove verbal inflection
  - (m>0) EED -> EE (agreed, feed)
  - (*v*) ED -> (plastered, bled)
  - (*v*) ING -> (motoring, sing)

Stemming: Porter Algorithm

- Step 1b: (contd. for -*ed* and -*ing* rules)
  - AT -> ATE (conflat*ed*)
  - BL -> BLE (troubl*ed*)
  - IZ -> IZE (siz*ed*)
  - (*doubled c & ¬(*L v *S v *Z)) -> single c (hopp*ing*, hiss*ing*, fall*ing*, fizz*ing*)
  - (m=1 & *cvc) -> E (fil*ing*, fail*ing*, slow*ing*)
- Step 1c: Y and I
  - (*v*) Y -> I (happy, sky)

Stemming: Porter Algorithm

- Step 2: Peel one suffix off for multiple suffixes
  - (m>0) ATIONAL -> ATE (relational)
  - (m>0) TIONAL -> TION (conditional, rational)
  - (m>0) ENCI -> ENCE (valenci)
  - (m>0) ANCI -> ANCE (hesitanci)
  - (m>0) IZER -> IZE (digitizer)
  - (m>0) ABLI -> ABLE (conformabli) - *able* (step 4)
  - …
  - (m>0) IZATION -> IZE (vietnamization)
  - (m>0) ATION -> ATE (predication)
  - …
  - (m>0) IVITI -> IVE (sensitiviti)

Stemming: Porter Algorithm

- Step 3
  - (m>0) ICATE -> IC (triplicate)
  - (m>0) ATIVE -> (formative)
  - (m>0) ALIZE -> AL (formalize)
  - (m>0) ICITI -> IC (electriciti)
  - (m>0) ICAL -> IC (electrical, chemical)
  - (m>0) FUL -> (hopeful)
  - (m>0) NESS -> (goodness)

Stemming: Porter Algorithm

- Step 4: Delete last suffix
  - (m>1) AL -> (revival) - *revive*, see step 5
  - (m>1) ANCE -> (allowance, dance)
  - (m>1) ENCE -> (inference, fence)
  - (m>1) ER -> (airliner, employer)
  - (m>1) IC -> (gyroscopic, electric)
  - (m>1) ABLE -> (adjustable, mov(e)able)
  - (m>1) IBLE -> (defensible,bible)
  - (m>1) ANT -> (irritant,ant)
  - (m>1) EMENT -> (replacement)
  - (m>1) MENT -> (adjustment)
  - …

Stemming: Porter Algorithm

- Step 5a: remove *e*
  - (m>1) E -> (probate, rate)
  - (m>1 & ¬*cvc) E -> (cease)
- Step 5b: *//* reduction
  - (m>1 & *LL) -> L (controll*er*, roll)

Stemming: Porter Algorithm

- Misses (understemming)
  - Unaffected:
    - *agreement* $(VC)^1VCC$ - step 4 (m>1)
    - *adhesion*
  - Irregular morphology:
    - drove, geese
- Overstemming
  - *relativity* - step 2
- Mis-stemming
  - *wander* $C(VC)^1VC$

Basic Morphology

Basic Affix Typology (don't seem to need more):

- i-suffix: inflectional suffix

  English: *cheer+ed = cheered, fit+ed = fitted, love+ed = loved*

- d-suffix: derivational suffix, changes word type

  English: *walk(V)+er = walker(N), happy(A)+ness=happiness(N)*

- a-suffix: attached suffix (enclitics).

  Italian *mandargli=mandare+gli* = to send + to him

Algorithmic Method

General Strategy:

- Normal order of suffixes seems to be *d, i, a.*
- Remove from right in order *a, i, d.*
- Generally remove all the *a* and *i* suffixes, sometimes leave the *d* one.

Types of Errors

• Conflation:	reply, rep.	rep
• Overstemming:	wander	wand
	news	new
• Misstemming:	relativity	relative
• Understemming:	knavish	knavish

Algorithmic Method

Strategy for German:

- Leave prefixes alone because they can change meaning.
- Put everything in small caps.
- Get rid of *ge-.*
- Get rid of *i* type: *e, em, en, ern, er, es, s, est,* (e.g, *armes* > *arm*)
- Get rid of *d* type*: end, ung, ig, ik, isch, lich, heit, keit*

Information Retrieval

Does stemming indeed improve IR?

- No: Harman (1991), Krovetz (1993)
- Possibly: Krovetz (1995)
  Depends on type of text, and the assumption is that once one moves beyond English, the difference will prove significant.

Crosslinguistic Applicability

• Can this type of stemming be applied to all languages?

— Not to Chinese, for example (doesn't need it).

• Do all languages have the same kind of morphology?

— No. Stemming assumes basically agglutinative morphology. This is not true crosslinguistically (but the algorithms seem to work pretty well within Indo-

Two ways of looking at language & the Great Divide

- Text understanding vs. Information Retrieval (IR)
- Info retrieval example: name extraction; how does Google correct "Britney Speers"