6.863J Natural Language Processing Lecture 6: part-of-speech tagging to parsing

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

The Menu Bar

- Administrivia:
  - Schedule alert: Lab1 due next *today* Lab 2, posted Feb 24; due the Weds after this – March 5 (web only – can post pdf)
- *Agenda:*
- Finish up POS tagging – Brill method
- From tagging to parsing: from linear representations to hierarchical representations

Two approaches

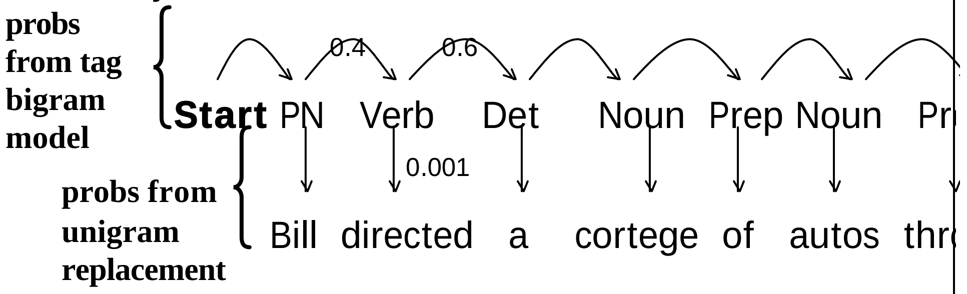
1. Noisy Channel Model (statistical) –
2. Deterministic baseline tagger composed with a cascade of fixup transducers

These two approaches will the guts of Lab 2

(lots of others: decision trees, ...)

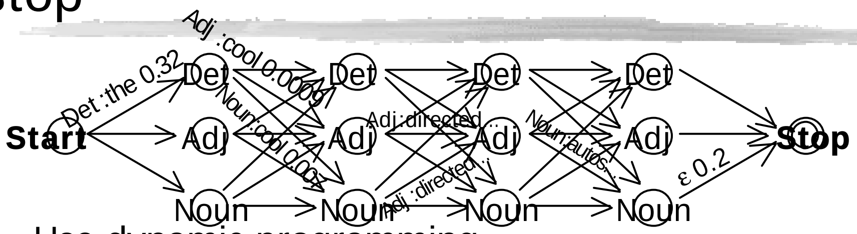
Summary

- We are modeling p(word seq, tag seq)
- The tags are hidden, but we see the words
- Is tag sequence X likely with these words?
- Noisy channel model is a “Hidden Markov Model”:



- Find X that maximizes probability product

Finding the best path from start to stop



- Use dynamic programming
- What is best path from Start to *each* node?
  - Work from left to right
  - Each node stores its best path from Start (as probability plus one backpointer)
- Special acyclic case of Dijkstra's shortest-path algorithm
- Faster if some arcs/states are absent

Method: Viterbi algorithm

- For each path reaching state s at step (word) t, we compute a path probability. We call the max of these viterbi(s,t)
- [Base step] Compute viterbi(0,0)=1
- [Induction step] Compute viterbi(s',t+1), assuming we know viterbi(s,t) for all s

Viterbi recursion

path-prob(s'|s,t) = viterbi(s,t) * a[s,s']

probability of path to s' through s	max path score * for state s at time t	transition probability $s \rightarrow s'$

$$\text{viterbi}(s',t+1) = \max_{s\, \in \text{STATES}} \text{path-prob}(s' \mid s,t)$$

Viterbi Method...

- This is *almost* correct...but again, we need to factor in the *unigram* prob of a state s' emitting a particular word *w* given an observation of that surface word *w*
- So the correct formula for the path prob to s' from s is:

$$\text{path-prob}(s'|s,t) = \underbrace{\text{viterbi}(s,t)}_{\text{Path prob so far to s}} * \underbrace{a[s,s']}_{\text{Bigram transition prob to state s'}} * \underbrace{b_{s'}(o_t)}_{\text{Unigram output prob at state s'}}$$

Finally...

- As before, we want to find the max path probability, over all states s:

$$\max_{s \in \text{STATES}} \text{path-prob}(s' \mid s,t)$$

Or as in your text...p. 179

```
function VITERBI(observations of len T,state-graph) returns best-path

  num-states ← NUM-OF-STATES(state-graph)
  Create a path probability matrix viterbi[num-states+2,T+2]
  viterbi[0,0] ← 1.0
  for each time step t from 0 to T do
    for each state s from 0 to num-states do
      for each transition s′ from s specified by state-graph
        ~~new-score ← viterbi[s, t] * a[s,s′] * b_s′(o_t)~~ Find the path probability
        if ((vite~~rbi[s′,t+1] = 0) || (new-score > viterbi[s′, t+1]~~))
          then                                    Find the max so far
              viterbi[s′, t+1] ← new-score
              back-pointer[s′, t+1] ← s
  Backtrace from highest probability state in the final column of viterbi[] and
return path
```

Two approaches

1. Noisy Channel Model (statistical) – what's that?? (we will have to learn some statistics)

2. Deterministic baseline tagger composed with a cascade of fixup transducers

These two approaches will the guts of Lab 2

(lots of others: decision trees, ...)

Fixup approach: Brill tagging (a kind of transformation-based learning)

Another FST Paradigm: Successive Fixups

- Like successive markups but *alter*
- Morphology
- Phonology
- Part-of-speech tagging
- **…**

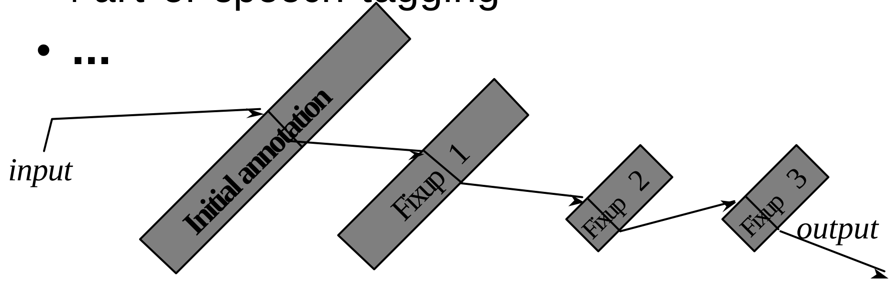
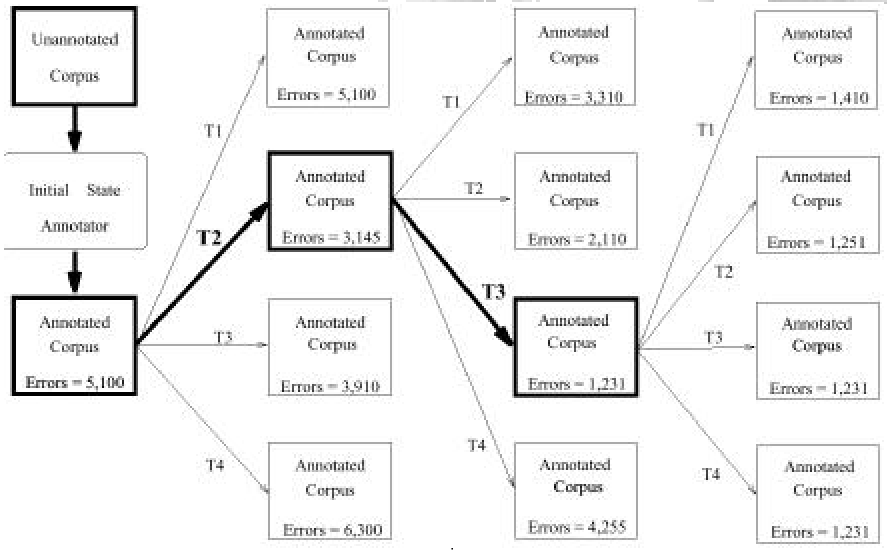
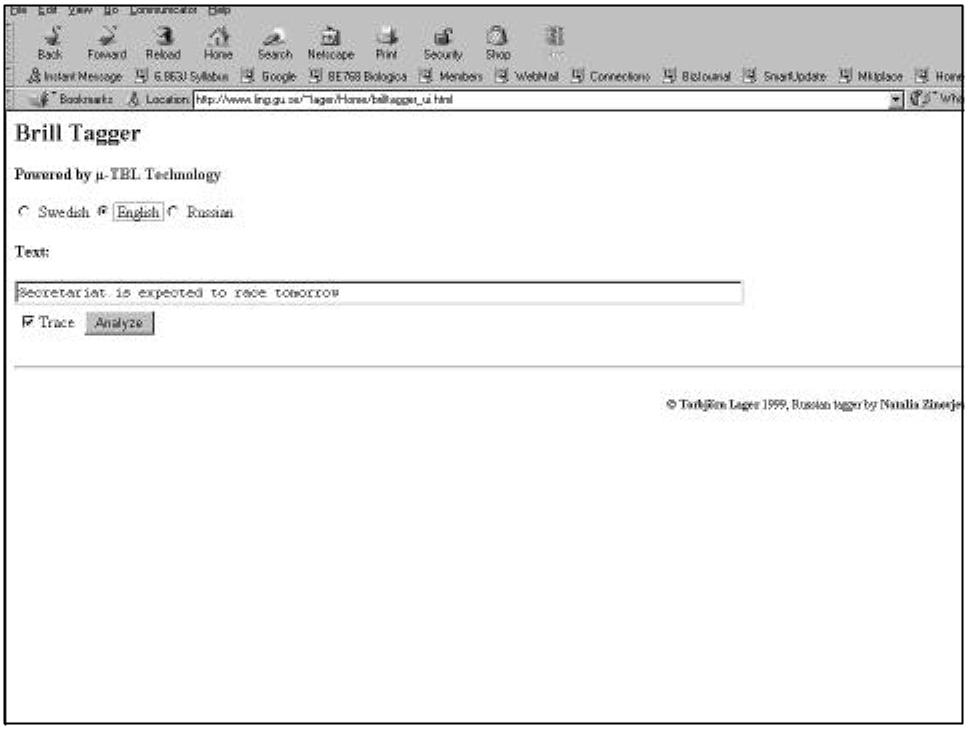


figure from Brill's thesis

Transformation-Based Tagging (Brill 1995)


Unannotated Corpus
Initial State Annotator
Annotated Corpus Errors = 5,100
T1
T2
T3
T4
Annotated Corpus Errors = 5,100
Annotated Corpus Errors = 3,145
Annotated Corpus Errors = 3,910
Annotated Corpus Errors = 6,300
Annotated Corpus Errors = 3,310
Annotated Corpus Errors = 2,110
Annotated Corpus Errors = 1,231
Annotated Corpus Errors = 4,255
Annotated Corpus Errors = 1,410
Annotated Corpus Errors = 1,251
Annotated Corpus Errors = 1,231
Annotated Corpus Errors = 1,231


File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop
Instant Message 6.863J Syllabus Google BE768 Biologica Members WebMail Connections BizJournal SmartUpdate Mktplace Home
Bookmarks Location

http://www.ling.gu.se/~lager/Home/brilltagger_ui.html

Brill Tagger

Powered by μ-TBL Technology

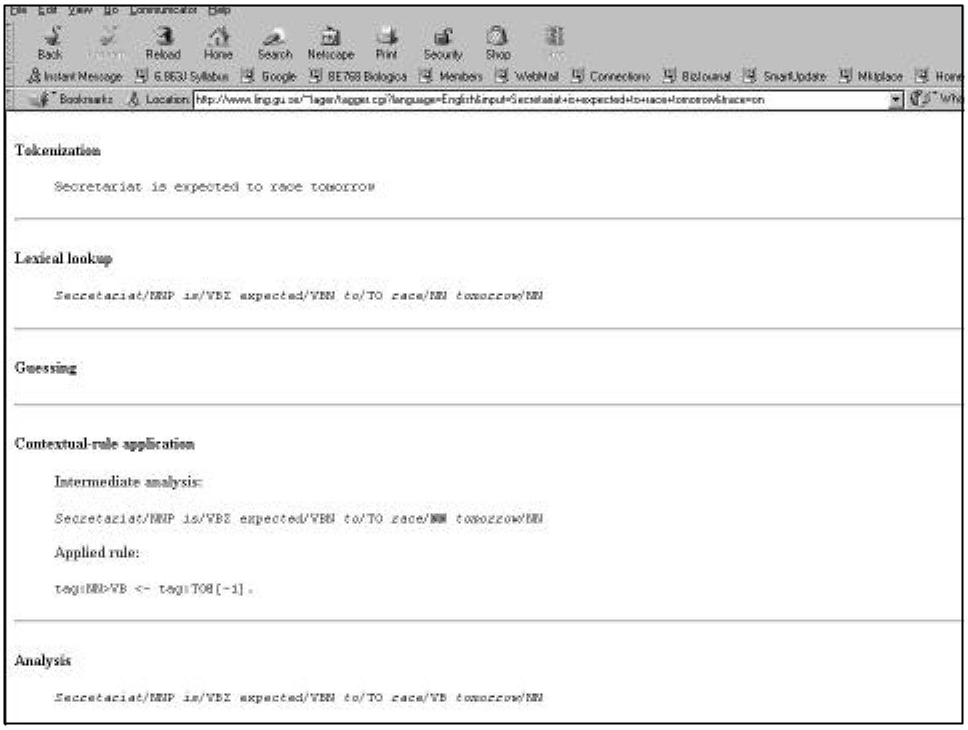
○ Swedish [✓ English] ○ Russian

Text:

```
Secretariat is expected to race tomorrow
```

[✓ Trace] Analyze

© Torbjörn Lager 1999, Russian tagger by Natalia Zinovje


File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop
Instant Message 6.863J Syllabus Google BE768 Biologica Members WebMail Connections BizJournal SmartUpdate Mktplace Home
Bookmarks Location

http://www.ling.gu.se/~lager/tagger.cgi?language=English&input=Secretariat+is+expected+to+race+tomorrow&trace=on

Tokenization

```
Secretariat is expected to race tomorrow
```

Lexical lookup

```
Secretariat/NNP is/VBZ expected/VBN to/TO race/NN tomorrow/NN
```

Guessing

Contextual-rule application

Intermediate analysis:

```
Secretariat/NNP is/VBZ expected/VBN to/TO race/NN tomorrow/NN
```

Applied rule:

```
tag:NN>VB <- tag:TO@[-1].
```

Analysis

```
Secretariat/NNP is/VBZ expected/VBN to/TO race/VB tomorrow/NN
```

Transformation based tagging

- Combines symbolic and stochastic approaches: uses machine learning to refine its tags, via several passes
- Analogy: painting a picture, use finer and finer brushes - start with broad brusch that covers a lot of the canvas, but colors areas that will have to be repainted. Next layer colors less, but also makes fewer mistakes, and so on.
- Similarly: tag using broadest (most general) rule; then an narrower rule, that changes a smaller number of tags, and so on. (We haven't said how the rules are learned)
- First we will see how the TBL rules are *applied*

Applying the rules

1. First label every word with its most-likely tag (as we saw, this gets 90% right...!) for example, in Brown corpus, *race* is most likely to be a Noun:

   P(NN|race)= 0.98

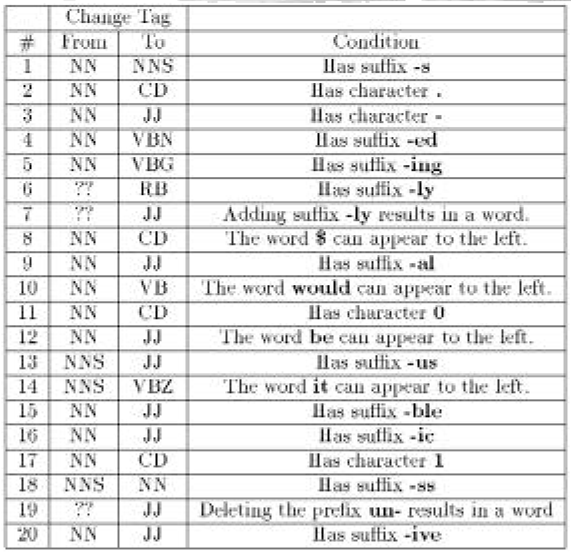
   P(VB|race)= 0.02
2. ...expected/VBZ to/T TO race / VB morrow/NN

   ...the/DT race/NN for/IN outer/JJ space/NN
3. Use transformational (learned) rules to change tags:

   Change **NN** *to* **VB** *when the previous tag is* **TO**

figure from Brill's thesis

Initial Tagging of OOV Words



#	Change Tag From	To	Condition
1	NN	NNS	Has suffix **-s**
2	NN	CD	Has character **.**
3	NN	JJ	Has character **-**
4	NN	VBN	Has suffix **-ed**
5	NN	VBG	Has suffix **-ing**
6	??	RB	Has suffix **-ly**
7	??	JJ	Adding suffix **-ly** results in a word.
8	NN	CD	The word **$** can appear to the left.
9	NN	JJ	Has suffix **-al**
10	NN	VB	The word **would** can appear to the left.
11	NN	CD	Has character **0**
12	NN	JJ	The word **be** can appear to the left.
13	NNS	JJ	Has suffix **-us**
14	NNS	VBZ	The word **it** can appear to the left.
15	NN	JJ	Has suffix **-ble**
16	NN	JJ	Has suffix **-ic**
17	NN	CD	Has character **1**
18	NNS	NN	Has suffix **-ss**
19	??	JJ	Deleting the prefix **un-** results in a word
20	NN	JJ	Has suffix **-ive**

(supervised) learning pudding - How?

- 3 stages

1. Start by labeling every word with most-likely tag
2. Then examine every possible transformation, and selects one that results in most improved tagging
3. Finally, re-tags data according to this rule
4. Repeat 1-3 until some stopping criterion (no new improvement, or small improvement)

- Output is ordered list of transformations that constitute a tagging procedure

How this works

- Set of possible ‘transforms’ is infinite, e.g., “transform NN to VB if the previous word was *MicrosoftWindoze* & word *braindead* occurs between 17 and 158 words before *that*”
- To limit: start with small set of abstracted transforms, or *templates*

Templates used: Change *a* to *b* when...

The preceding (following) word is tagged **z**.
The word two before (after) is tagged **z**.
One of the two preceding (following) words is tagged **z**.
One of the three preceding (following) words is tagged **z**.
The preceding word is tagged **z** and the following word is tagged **w**.
The preceding (following) word is tagged **z** and the word
two before (after) is tagged **w**.

Variables ***a, b, z, w,*** range over parts of speech

Method

1. Call `Get-best-transform` with list of potential templates; this calls
2. `Get-best-instance` which instantiates each template over all its variables (given specific values for where we are)
3. Try it out, see what score is (improvement over known tagged system -- supervised learning); pick best one locally

```
function TBL(corpus) returns transforms-queue
 INTIALIZE-WITH-MOST-LIKELY-TAGS(corpus)
 until end condition is met do
   templates ← GENERATE-POTENTIAL-RELEVANT-TEMPLATES
   best-transform ← GET-BEST-TRANSFORM(corpus, templates)
   APPLY-TRANSFORM(best-transform, corpus)
   ENQUEUE(best-transform-rule, transforms-queue)
 end
 return(transforms-queue)

 function GET-BEST-TRANSFORM(corpus, templates) returns transform
   for each template in templates
     (instance, score) ← GET-BEST-INSTANCE(corpus, template)
     if (score > best-transform.score) then best-transform ← (instance, score)
   return(best-transform)
```

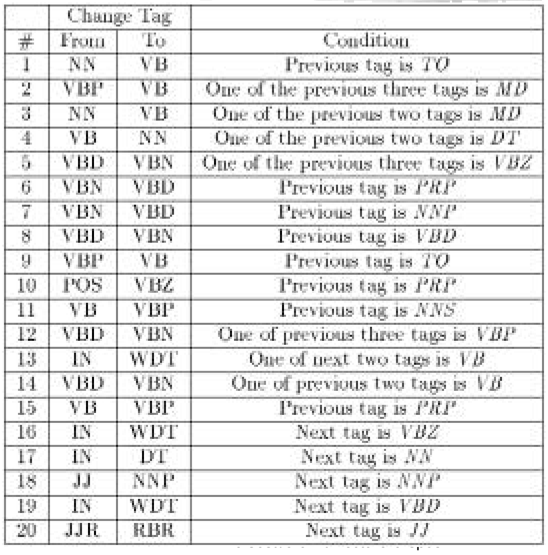
```
function GET-BEST-INSTANCE(corpus, template) returns transform
  for from-tag ← from tag−1 to tag−n do
   for to-tag ← from tag−1 to tag−n do
     for pos ← from 1 to corpus-size do
       if (correct-tag(pos) == to-tag && current-tag(pos) == from-tag)
            num-good-transforms(current-tag(pos−1))++
       elseif (correct-tag(pos)==from-tag && current-tag(pos)==from-tag)
             num-bad-transforms(current-tag(pos−1))++
     end
     best-Z ← ARGMAX_t(num-good-transforms(t) - num-bad-transforms(t))
      if(num-good-transforms(best-Z) - num-bad-transforms(best-Z)
                 > best-instance Z) then
        best-instance ← "Change tag from from-tag to to-tag
                              if previous tag is best-Z"
return(best-instance)

procedure APPLY-TRANSFORM(transform, corpus)
for pos ← from 1 to corpus-size do
  if (current-tag(pos)==best-rule-from)
       && (current-tag(pos−1)==best-rule-prev))
     current-tag(pos) = best-rule-to
```

nonlexicalized rules learned by TBL tagger

	Change tags			
#	From	To	Condition	Example
1	NN	VB	Previous tag is TO	to/TO race/NN $\rightarrow$ VB
2	VBP	VB	One of the previous 3 tags is MD	might/MD vanish/VBP $\rightarrow$ VB
3	NN	VB	One of the previous 2 tags is MD	might/MD not reply/NN $\rightarrow$ VB
4	VB	NN	One of the previous 2 tags is DT	
5	VBD	VBN	One of the previous 3 tags is VBZ	

figure from Brill's thesis

Transformations Learned



	Change Tag		
#	From	To	Condition
1	NN	VB	Previous tag is *TO*
2	VBP	VB	One of the previous three tags is *MD*
3	NN	VB	One of the previous two tags is *MD*
4	VB	NN	One of the previous two tags is *DT*
5	VBD	VBN	One of the previous three tags is *VBZ*
6	VBN	VBD	Previous tag is *PRP*
7	VBN	VBD	Previous tag is *NNP*
8	VBD	VBN	Previous tag is *VBD*
9	VBP	VB	Previous tag is *TO*
10	POS	VBZ	Previous tag is *PRP*
11	VB	VBP	Previous tag is *NNS*
12	VBD	VBN	One of previous three tags is *VBP*
13	IN	WDT	One of next two tags is *VB*
14	VBD	VBN	One of previous two tags is *VB*
15	VB	VBP	Previous tag is *PRP*
16	IN	WDT	Next tag is *VBZ*
17	IN	DT	Next tag is *NN*
18	JJ	NNP	Next tag is *NNP*
19	IN	WDT	Next tag is *VBD*
20	JJR	RBR	Next tag is *JJ*

BaselineTag*
NN @→ VB / / TO _
VBP @→ VB / / ... _
etc.

Compose this cascade of FSTs.

Get a big FST that does the initial tagging and the sequence of fixups "all at once."

Error analysis: what's hard for taggers

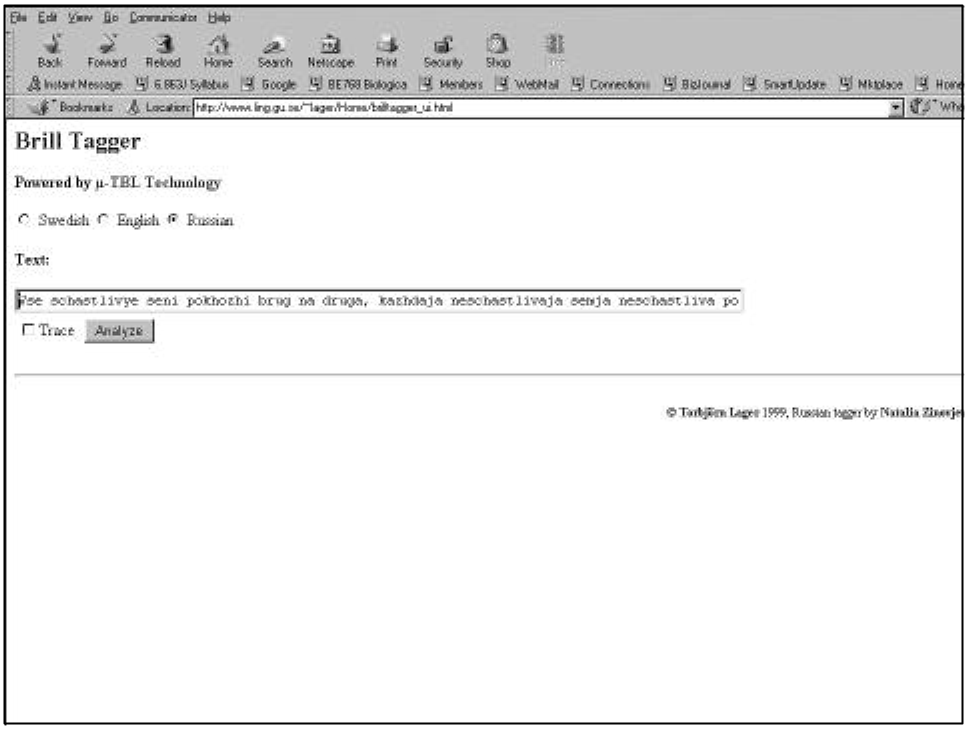
- Common errors (> 4%)
  - NN vs .NNP (proper vs. other nouns) vs. JJ (adjective): hard to distinguish prenominally; important to distinguish esp. for information extraction
  - RP vs. RB vs IN: all can appear in sequences immed. after verb
  - VBD vs. VBN vs. JJ: distinguish past tense, past participles (*raced* vs. *was raced* vs. *the out raced horse*)

What's hard

- Unknown words
  - Order 0 idea: equally likely over all parts of speech
  - Better idea: same distribution as 'Things seen once' estimator of 'things never seen' - theory for this done by Turing (again!)
  - *Hapax legomenon*
  - Assume distribution of unknown words is like this
  - But most powerful methods make use of how word is spelled
- See file in the course tagging dir on this

Or unknown language

- Vse schastlivye sen'i pokhozhi brug na druga, kazhdaja neschastlivaja sem'ja neschastliva po-svoemu


File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop
Instant Message 6.863J Syllabus Google BE768 Biologica Members WebMail Connections BizJournal SmartUpdate Mktplace
Bookmarks Location: http://www.ling.gu.se/~lager/Home/brilltagger_ui.html

Brill Tagger

Powered by µ-TBL Technology

Swedish English [✓ Russian]

Text:

Vse schastlivye semi pokhozhi brug na druga, kazhdaja neschastlivaja semja neschastliva po

☐ Trace Analyze

© Torbjörn Lager 1999, Russian tagger by Natalia Zinovje

Most powerful unknown word detectors

- 3 inflectional endings (*-ed, -s, -ing*); 32 derivational endings (*-ion,* etc.); capitalization; hyphenation
- More generally: should use morphological analysis! (and some kind of machine learning approach)
- How hard is this? We don’t know - we actually don’t know how children do this, either (they make mistakes)

Laboratory 2

- Goals:

1. Use both HMM and Brill taggers
2. Find errors that both make, relative to genre
3. Compare performance – use of kappa & ‘confusion matrix’
4. All the slings & arrows of corpora – use Wall Street Journal excerpts, as well as ‘switchboard’ corpus

Brown/Upenn corpus tags

J. text, p. 297 Fig 8.6
1M words
60K tag counts

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	“to”	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential ‘there’	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	*$*
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	*#*
PDT	Predeterminer	*all, both*	“	Left quote	(‘ or “)
POS	Possessive ending	*’s*	”	Right quote	(’ or ”)
PP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PP$	Possessive pronoun	*your, one’s*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

Coda on kids

C: “Mommy, nobody don’t like me”

A: No, say, “nobody likes me”

C: Nobody don’t likes me

A: Say, “nobody likes me”

C: Nobody don’t likes me
[7 repetitions]

C: Oh! Nobody don’t like me!

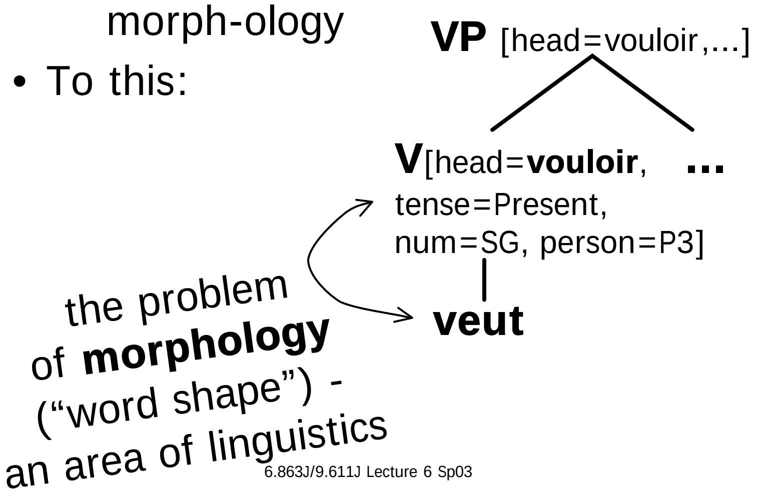
Parsing words - review

- We are mapping between surface, underlying forms
- Sometimes, information is ‘invisible’ (I.e., erased *e,* or an underlying/surface 0)
- There is ambiguity (more than one parse)

From lines to hierarchical respresentions...

- From this:

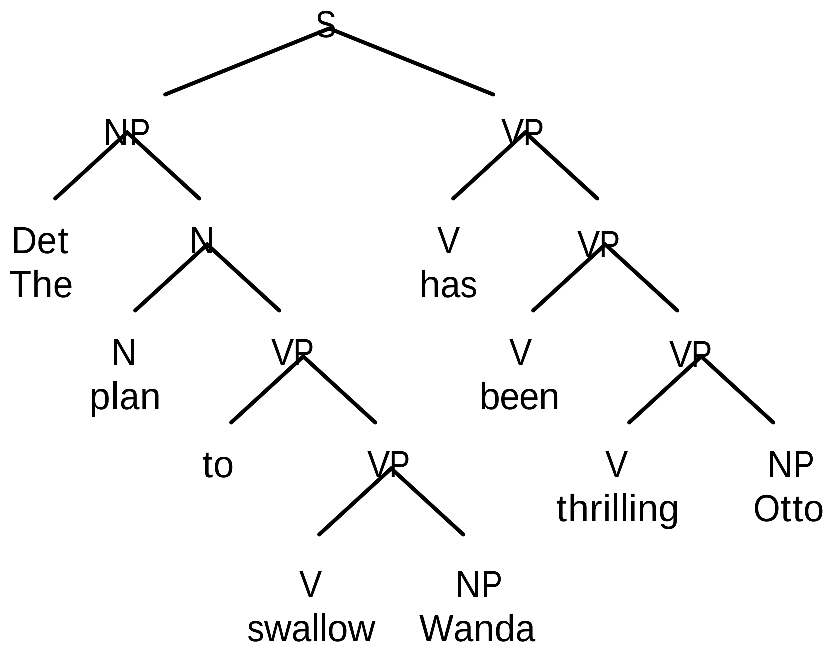
  morph-ology
- To this:


VP [head=vouloir,...]
V[head=**vouloir**, tense=Present, num=SG, person=P3]
...
veut
the problem of **morphology** (“word shape”) - an area of linguistics

What can’t linear relations represent?

- wine dark sea $\rightarrow$ (wine (dark sea)) *or* ((wine dark) sea) ?
- deep blue sky
- Can fsa’s *represent* this?
- Not really: algebraically, *defined* as being associative (doesn’t matter about concatenation order)

So, from linear relations... to hierarchies

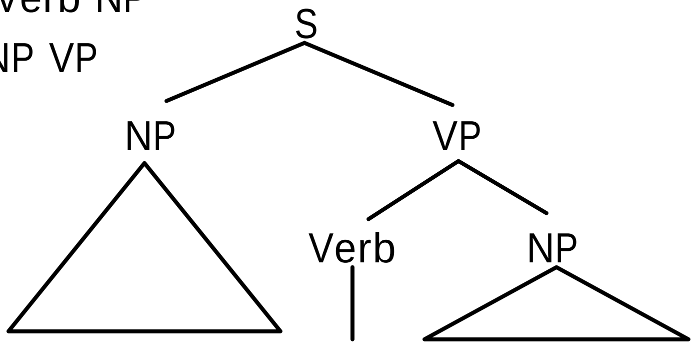


Examples

Verb → thrills

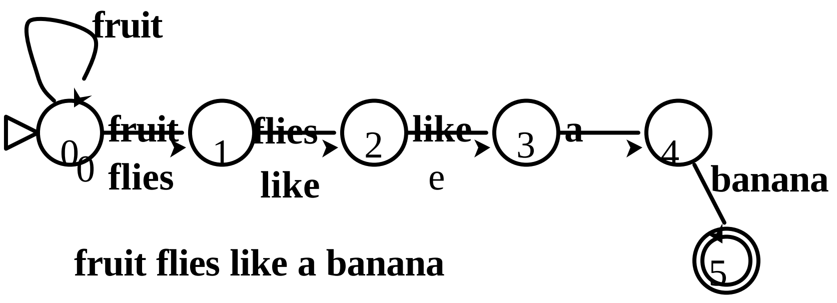
VP→ Verb NP

S → NP VP



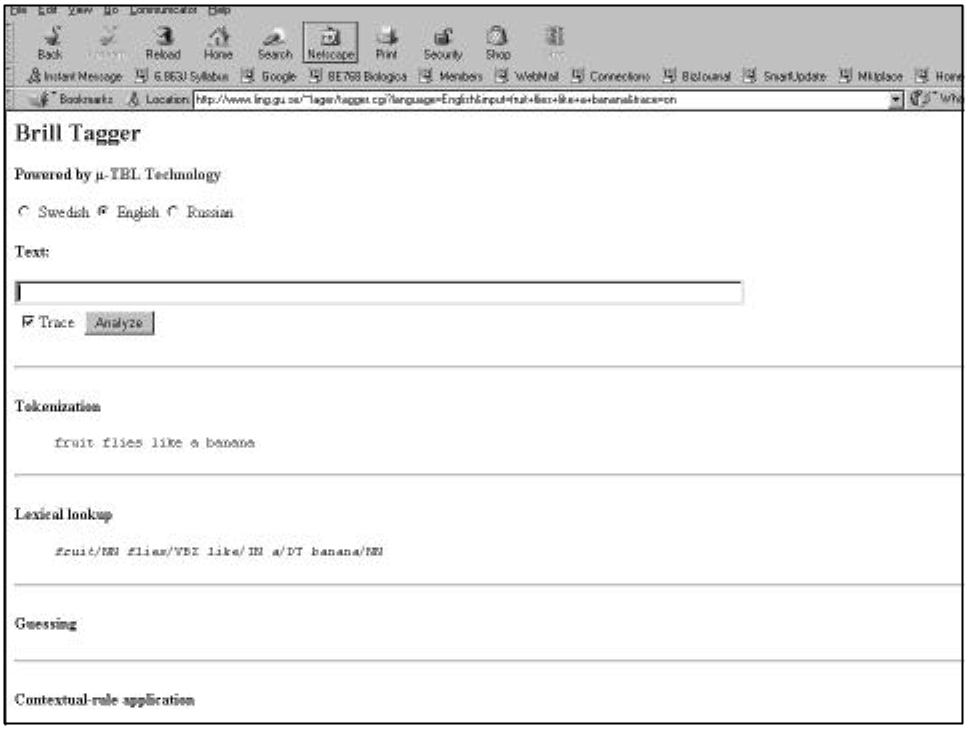
A roller coaster thrills every teenager

Parsing for fsa's: keep track of what 'next state' we could be in at each step



fruit flies like a banana

NB: ***ambiguity*** **= > 1 path through network**
= > 1 sequence of states ('*parses*')
= > 1 'syntactic rep' = >1 'meaning'



http://www.ling.gu.se/~lager/tagger.cgi?language=English&input=fruit+flies+like+a+banana&trace=on

Brill Tagger

Powered by μ-TBL Technology

○ Swedish [✓ English] ○ Russian

Text:

[✓ Trace] Analyze

Tokenization

```
fruit flies like a banana
```

Lexical lookup

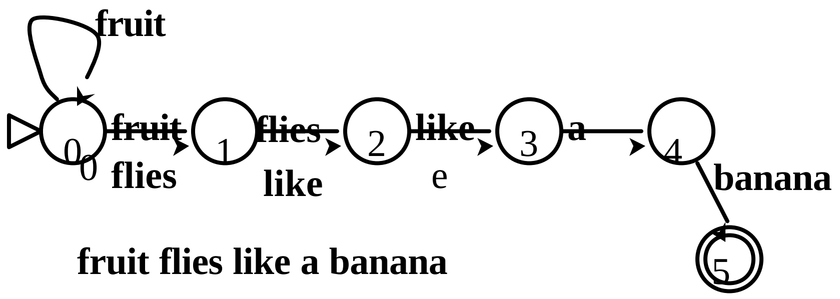
```
fruit/NN flies/VBZ like/IN a/DT banana/NN
```

Guessing

Contextual-rule application

FSA Terminology

- Transition function: next state unique = deterministic fsa
- Transition relation: > 1 next state = nondeterministic fsa



fruit flies like a banana

Methods for parsing

- How do we handle ambiguity?
- Methods:
  1. Backtrack
  2. Convert to deterministic machine (ndfsa $\rightarrow$ dfsa): *offline* compilation
  3. Pursue all paths in parallel: *online* computation ("state set" method)
  4. Use lookahead
- – We will use all these methods for more complex machines/language representations

FSA terminology

- Input alphabet,Σ; transition mapping, δ; finite set of states, Q; start state q_0; set of final states, q_f
- $\delta(q, s) \rightarrow q'$
- Transition function: next state unique = deterministic fsa
- Transition relation: > 1 next state = nondeterministic fsa

State-set method: simulate a nondeterministic fsa

- Compute all the possible next states the machine can be in at a step = *state-set*
- Denote this by S_i = set of states machine can be in after analyzing *i* tokens
- Algorithm has 3 parts: (1) *Initialize;* (2) *Loop;* (3) *Final state?*
- *Initialize:* S_0 denotes initial set of states we're in, before we start parsing, that is, q_0
- *Loop:* We must compute S_i, given S_{i-1}
- *Final?:* S_f = set of states machine is in after reading all tokens; we want to test if there is a final state in there

State-set parsing

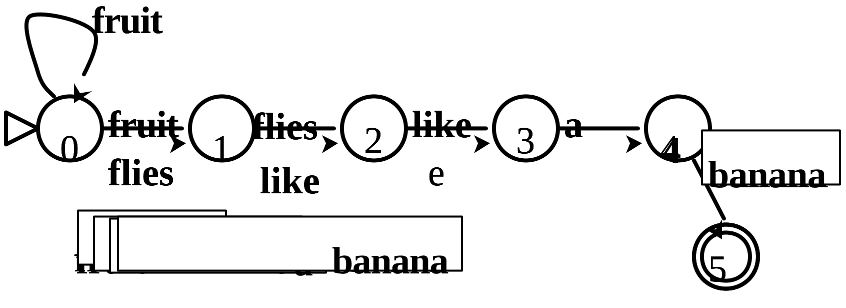
Initialize:	Compute initial state set, S_0
	1. $S_0 \leftarrow q_0$
	2. $S_0 \leftarrow \varepsilon$–closure($S_0$)
Loop:	Compute S_i from S_{i-1}
	1. For each word w_i, i=1,2,...,n
	2. $S_i \leftarrow \bigcup_{q \in S_{i-1}} d(q, w_i)$
	3. $S_i \leftarrow \varepsilon$–closure($S_i$)
	4. if $S_i = \varnothing$ then halt & reject else continue
Final:	Accept/reject
	1. If $q_f \in S_n$ then accept else reject

What's the minimal data structure we need for this?

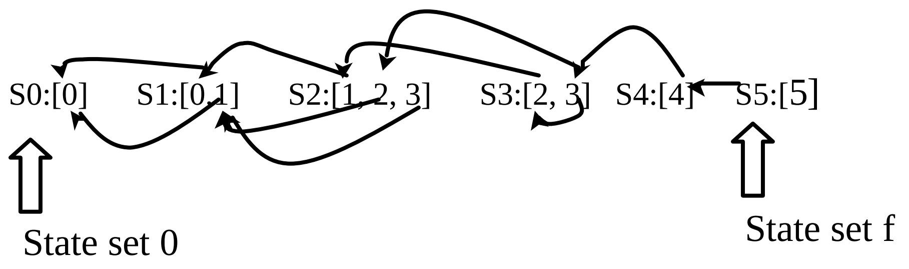
- [*S, i*] where *S* = denotes *set of* states we could be in; *i* denotes current point we're at in sentence
- As we'll see, we can use this *same* representation for parsing w/ more complex networks (grammars) - we just need to add *one* new piece of information for *state names*
- In network form $\textcircled{q_i} \xrightarrow{a} \textcircled{q_k} \xrightarrow{b}$
- In rule form:

$q_i \rightarrow t \bullet \beta\ q_f$ *where* t= some token of the input, and β = remainder (so 'dot' represents *how far we have traveled)*

Example



Use backpointers to keep track of the different paths (parses):



When is it better to convert at compile time vs. run time? (for fsa)

- *Run time*: compute next state set on the fly
- *Compile time*: do it once and for all
- When would this difference show up in natural languages (if at all)?

Where do the fsa states come from?

- States are <u>*equivalence classes*</u> of words (tokens) under the operation of *substitution*

- Linguistic formulation (Wells, 1947, pp. 81-82): “A word *A* belongs to the class determined by the environment ____*X* if *AX* is either an utterance or occurs as a part of some utterance” (*distributional* analysis)

- This turns out to be algebraically correct
- Can be formalized - the notion of *syntactic equivalence*

X-files: fragments from an alien language

1. Kerry lost the election
2. Gore will lose the election
3. Gore could lose the election
4. Gore should lose the election
5. Gore did lose the election
6. Gore could have lost the election
7. Gore should have lost the election
8. Gore will have lost the election
9. Gore could have been losing the election
10. Gore should have been losing the election
11. Gore will have been losing the election
12. Gore has lost the election

More X-files

14. Bush lost the election
15. Bush will lose the election
16. Bush could lose the election
17. Bush should lose the election
18. Bush did lose the election
19. Bush could have lost the election
20. Bush should have lost the election
21. Bush will have lost the election
22. Bush could have been losing the election
23. Bush should have been losing the election
24. Bush will have been losing the election
25. Bush has lost the election

Formally...

- Definition. A *binary relation* between sets *A, B,* is a subset (possibly empty) of $A \times B$
- Definition. Strings *k,r* are *left-substitutable* in a language *L,* if, for all strings *w* defined over Σ^*, $kw \in L$ iff $rw \in L$
- Fact. Left-substitutability is an equivalence relation (reflexive, transitive, symmetric)
- Definition. An equivalence relation over Σ is *finite rank* if it divides Σ into finitely many equivalence classes
- Definition. A binary relation *R* is called *right-invariant* if, for all $p,r \in \Sigma^*$, $pRr \Rightarrow pwRrw$

And formally...

- Fact. A right-invariant relation *R* is an equivalence relation
- <u>Theorem</u> (Myhill-Nerode, 1956)

<u>Theorem</u> (Myhill-Nerode, 1956).

- Let $L \subseteq \Sigma^*$. Then the following 3 propositions are equivalent:

1. *L* is generated (accepted) by some finite-state automaton (finite transition network);

2. *L* is the union of certain equivalence classes of a right-invariant equivalence relation of finite rank

3. Let the equivalence relation *R* be defined as follows: *xRy* iff *x* and *y* are left-substitutable in *L.* Then this relation *R* is of finite-rank and is right-invariant [this is Wells' definition]

Finite # of bins = finite state

- Gives easy way to show what is *not* finite-state
- Eg, $a^n cb^n$, *for all n> 0*
- Proof by contradiction.

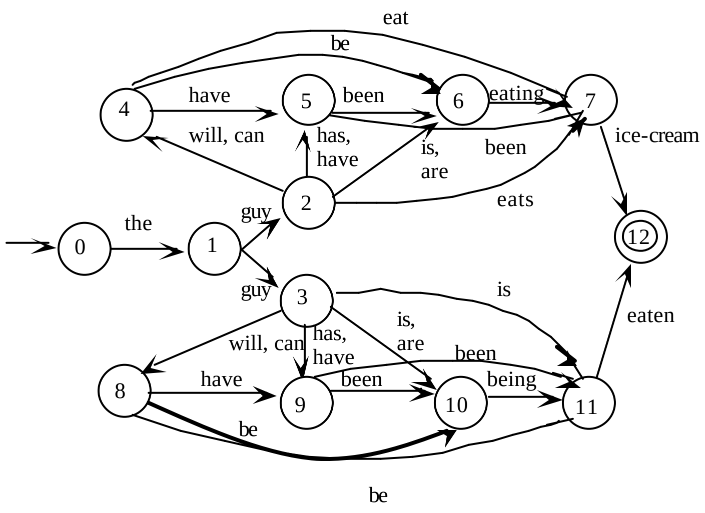
Suppose there was such an FSA. By the theorem, this FSA is of finite rank, and classifies all strings in Σ^* into one of a finite number of classes.

By the pigeonhole principle, there must exist some string a^i s.t. a^j with $j \neq i$ is in the same equivalence class as a^i . But then the fsa must recognize *both* a^i c a^j and a^i c a^i , a contradiction

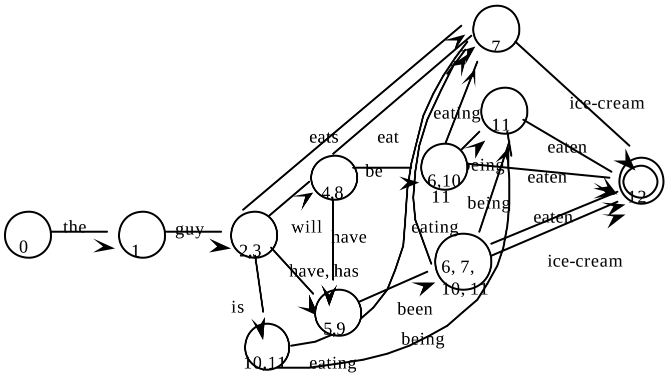
Why not fsa's forever?

- Can't yield the right *set of strings*= <u>weak generative capacity</u> (antiantimissle…)
- Can't yield the right *set of structures* = <u>strong generative capacity</u> (*dark blue sky*)
- How do these failures show up?

A more complex fsa

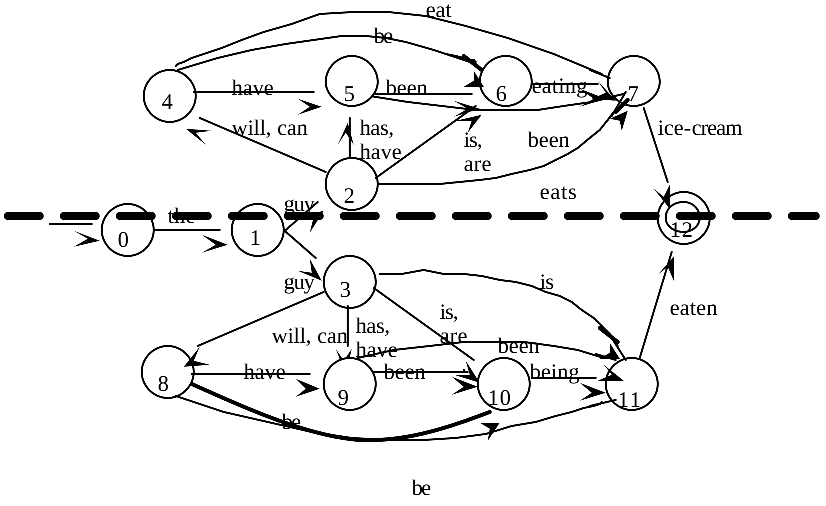


Conversion to deterministic machine



What are we missing here?

We are missing the symmetry

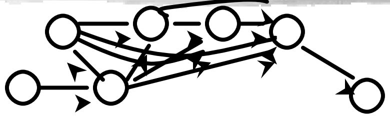


Having a poor representation...

- Shows up in having duplicated states (with no other connection to each other)
- System would be ‘just as complex’= have the same size (what is size of automaton?) even if the network were *not* symmetric
- So we have failed to capture this regularity & the network *could be compressed*
- How?

Compressability reveals rendundancy (pattern)that we have missed

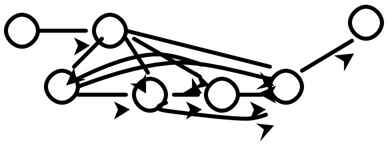
Active:



+

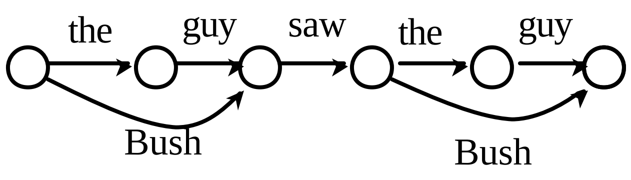
Rule that flips network=

Passive:



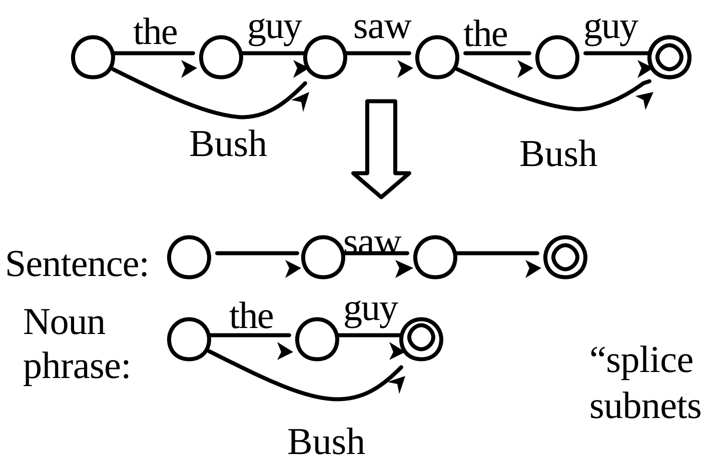
Aka “transformational grammar”

But it’s worse than that... more redundancy even so



So, obvious programming approach: use a *subroutine*

Subnetworks as subroutines, to compress the description

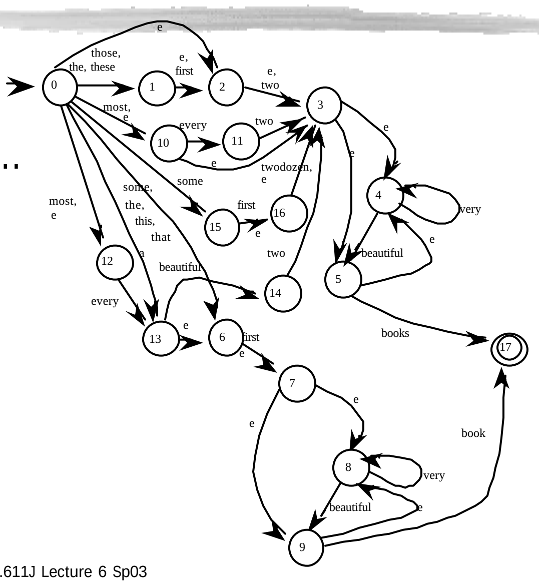


“splice out” common subnets

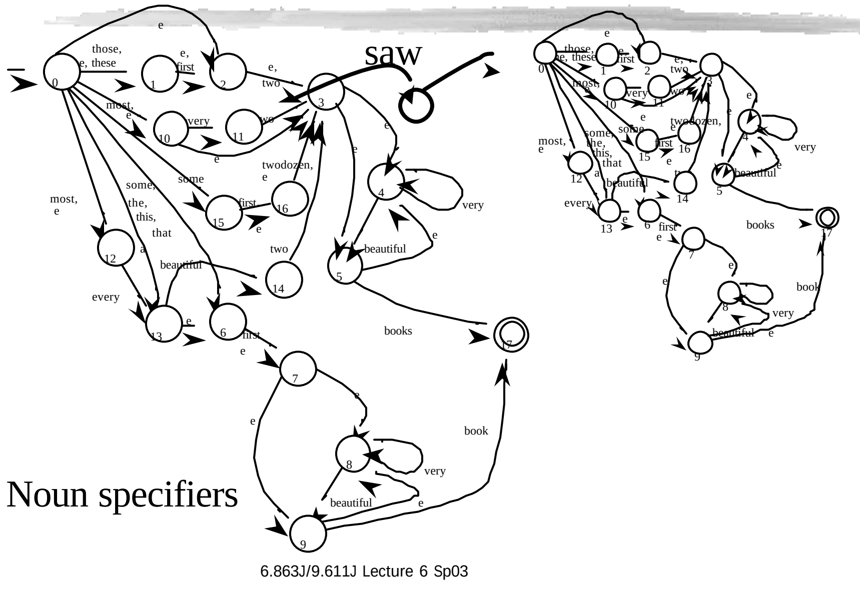
Could be worse...

Could be raining...

Noun "specifiers"



It could be even worse...



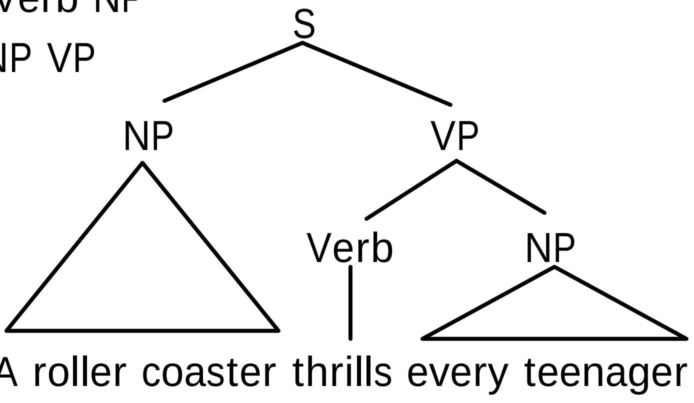
Noun specifiers

Examples

Verb → thrills

VP→ Verb NP

S → NP VP



The notion of a *common* subnetwork

- Equivalent to the notion of a *phrase*
- A Noun Phrase (NP)
- Defined by substitution class of a *sequence* of words (aka "a *constituent*") - extension beyond substitution of single words
- A phrase iff we can interchangeably substitute that sequence of words *regardless of context*
- So also gives us the notion of a *context-free grammar (CFG)*

Constituents, aka phrases

- Building blocks that are units of words concatenated together
- Why?
- Ans:

1. They *act together* (i.e., behave alike under operations) - what operations?
2. Succinctness
3. (Apparently) nonadjacent constraints

The deepest lesson

- Claim: *all* apparently nonadjacent relationships in languge can be reduced to *adjacent* ones via projection to a new level of representation
- (In one sense, vacuous; in another, deep)
- Example: Subject-Verb agreement (agreement generally)
- Example: so-called *wh*-movement

Gaps ("deep" grammar!)

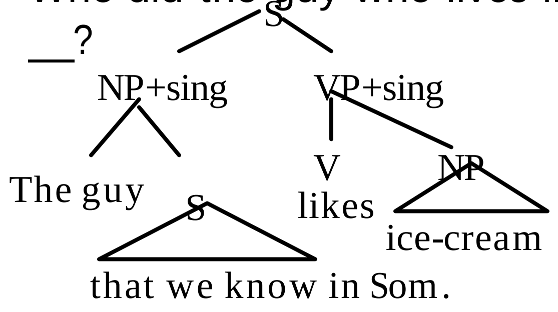
- Pretend "kiss" is a pure transitive verb.
- Is "the president kissed" grammatical?
  - If so, what type of phrase is it?

- the sandwich that
- I wonder what
- What else has

the president kissed e
Sally said the president kissed e
Sally consumed the pickle with e
Sally consumed e with the pickle

Examples

- The guy that we know in Somerville likes ice-cream
- Who did the guy who lives in Somerville see __?



The deep reason why

- Machinery of the mind: based only on concatenation of adjacent elements - *not* on ‘counting’ eg., “take the 7th element & move it...”
- Runs through all of linguistic representations (stress, metrical patterns, phonology, syntax, ...)
- Strong constraint on *what* we have to represent

Constituents

- Basic ‘is-a’ relation
- Act as ‘whole units’ -
  - *I want this student to solve the problem*
  - *?? Student, I want this to solve the problem*
  - *This student, I want to solve the problem*
- Sometimes, we don’t see whole constituents...book titles (claimed as objection to constituency):
  - *Sometimes a Great Notion*
  - *The Fire Next Time*
- Why might that be?