

# Words and the Company they keep

# Motivation

- Environment:
  - mostly “not a full analysis (sentence/text parsing)”
- Tasks where “words & company” are important:
  - word sense disambiguation (MT, IR, TD, IE)
  - lexical entries: subdivision & definitions (lexicography)
  - language modeling (generalization, [kind of] smoothing)
  - word/phrase/term translation (MT, Multilingual IR)
  - NL generation (“natural” phrases) (Generation, MT)
  - parsing (lexically-based selectional preferences)

# Collocations

- Collocation
  - Firth: “word is characterized by the company it keeps”; collocations of a given word are statements of the habitual or customary places of that word.
  - non-compositionality of meaning
    - cannot be derived directly from its parts (heavy rain)
  - non-substitutability in context
    - for parts (red light)
  - non-modifiability (& non-transformability)
    - kick the ~~yellow~~ bucket; take exceptions ~~A~~ to

# Association and Co-occurrence; Terms

- Does not fall under “collocation”, but:
- Interesting just because it does often [rarely] appear together or in the same (or similar) context:
  - (doctors, nurses)
  - (hardware, software)
  - (gas, fuel)
  - (hammer, nail)
  - (communism, free speech)
- Terms:
  - need not be  $> 1$  word (notebook, washer)

# Collocations of Special Interest

- Idioms: really fixed phrases
  - kick the bucket, birds-of-a-feather, run for office
- Proper names: difficult to recognize even with lists
  - Tuesday (person's name), May, Winston Churchill, IBM, Inc.
- Numerical expressions
  - containing “ordinary” words
    - Monday Oct 04 1999, two thousand seven hundred fifty
- Phrasal verbs
  - Separable parts:
    - look up, take off

# Further Notions

- Synonymy: different form/word, same meaning:
  - notebook / laptop
- Antonymy: opposite meaning:
  - new/old, black/white, start/stop
- Homonymy: same form/word, different meaning:
  - “true” (random, unrelated): can (aux. verb / can of Coke)
  - related: polysemy; notebook, shift, grade, ...
- Other:
  - Hyperonymy/Hyponymy: general vs. special: vehicle/car
  - Meronymy/Holonymy: whole vs. part: body/leg

# How to Find Collocations?

- Frequency
  - plain
  - filtered
- Hypothesis testing
  - $t$  test
  - $\chi^2$  test
- Pointwise (“poor man’s”) Mutual Information
- (Average) Mutual Information

# Frequency

- Simple
  - Count n-grams; high frequency n-grams are candidates:
    - mostly function words
    - frequent names
- Filtered
  - Stop list: words/forms which (we think) cannot be a part of a collocation
    - a, the, and, or, but, not, ...
  - Part of Speech (possible collocation patterns)
    - A+N, N+N, N+of+N, ...



C(a,b)	a	b
80871	of	the
58841	in	th
26430	to	the
21842	on	the
21839	for	the
18568	and	the
16121	that	the
15630	at	the
15494	to	be
13899	in	a
13689	of	a
13361	by	the

C(a,b)	a	b
11487	New	York
7261	United	Sates
5412	Los	Angeles
3301	last	year
3191	Saudi	Arabia
2699	last	week
2514	vice	president
2378	Persian	Gulf
2161	San	Francisco
2106	Presiden	Bush
2001	Middle	East
1942	Saddan	Hussein

# Hypothesis Testing

- Hypothesis
  - something we test (against)
- Most often:
  - compare possibly interesting thing vs. “random” chance
  - “Null hypothesis”:
    - something occurs by chance (that’s what we suppose).
    - Assuming this, prove that the probability of the “real world” is then too low (typically  $< 0.05$ , also 0.005, 0.001)... therefore reject the null hypothesis (thus confirming “interesting” things are happening!)
    - Otherwise, it’s possible there is nothing interesting.

# Hypothesis Testing

- Conclusions are about  $H_0$ 
  - reject  $H_0$  in favour of  $H_1$
  - do not reject  $H_0$
- we never can conclude
  - reject  $H_1$ , or even
  - accept  $H_1$

# Results on an Hypothesis test

		Decision	
		Reject $H_0$	Don't reject
Truth	$H_0$	Type I Error	Right Decision
	$H_1$	Right Decision	Type II Error

# Significance level

- $P(\text{type I error}) = \text{significance level} = \alpha$

## *t* test (Student's *t* test)

- Significance of difference
  - compute “magic” number against normal distribution (mean  $\mu$ )
  - using real-world data: ( $\bar{x}$  real data mean,  $s^2$  variance,  $N$  size):
    - $t = (\bar{x} - \mu) / \sqrt{s^2 / N}$
  - find in tables (see MS, p. 609):
    - d.f. = degrees of freedom (parameters which are not determined by other parameters)
    - percentile level  $p = 0.05$  (or better)
  - the bigger  $t$ :
    - the better chances that there is the interesting feature we hope for (i.e. we can reject the null hypothesis)
    - $t$ : at least the value from the table(s)

## *t* test on words

- null hypothesis: independence
  - mean  $\mu$ :  $p(w_1) p(w_2)$
- data estimates:
  - $\hat{x}$  = MLE of joint probability from data
  - $s^2$  is  $p(1-p)$ , i.e. almost  $p$  for small  $p$ ;  $N$  is the data size
- Example: (d.f.  $\sim$  sample size)
  - ‘general term’ (homework corpus):  $c(\text{general}) = 108$ ,  $c(\text{term}) = 40$
  - $c(\text{general,term}) = 2$ ; expected  $p(\text{general})p(\text{term}) = 8.8\text{E-}8$
  - $t = (9.0\text{E-}6 - 8.8\text{E-}8) / (9.0\text{E-}6 / 221097)^{1/2} = 1.40$  (not  $> 2.576$ ) thus ‘general term’ is not a collocation with confidence 0.005
  - ‘true species’:  $(84/1779/9)$ :  $t = 2.774 > 2.576$  !!

## Pearson's Chi-square test

- $\chi^2$  test (general formula):  $\sum_{i,j} (O_{ij} - E_{ij})^2 / E_{ij}$ 
  - where  $O_{ij}/E_{ij}$  is the observed/expected count of events  $i, j$
- for two-outcomes-only events:

$w_{\text{right}} \setminus w_{\text{left}}$	= true	≠ true
= species	9	1,770
≠ species	75	219,243

$$\chi^2 = 221097(219243 \times 9 - 75 \times 1770)^2 / 1779 \times 84 \times 221013 \times 219318 = 103.39 > 7.88$$

(at .005 thus we can reject the independence assumption)



# Pointwise Mutual Information

- This is **NOT** the MI as defined in Information Theory
  - (IT: average of the following; not of **values**)

- ...but might be useful:

$$I'(a,b) = \log_2 (p(a,b) / p(a)p(b)) = \log_2 (p(a|b) / p(a))$$

- Example (same):

$$I'(\text{true,species}) = \log_2 (4.1\text{e-}5 / 3.8\text{e-}4 \times 8.0\text{e-}3) = 3.74$$

$$I'(\text{general,term}) = \log_2 (9.0\text{e-}6 / 1.8\text{e-}4 \times 4.9\text{e-}4) = 6.68$$

- measured in bits but it is difficult to give it an interpretation
- used for ranking ( $\sim$  the null hypothesis tests)

# Mutual Information and Word Classes

# The Problem

- Not enough data
  - **Language Modeling: we do not see “correct” n-grams**
    - solution so far: smoothing
  - **suppose we see:**
    - short homework, short assignment, simple homework
  - **but not:**
    - simple assignment
  - **What happens to our (bigram) LM?**
    - $p(\text{homework} | \text{simple}) = \text{high probability}$
    - $p(\text{assignment} | \text{simple}) = \text{low probability (smoothed with } p(\text{assignment}))$
- They should be much closer!

# Word Classes

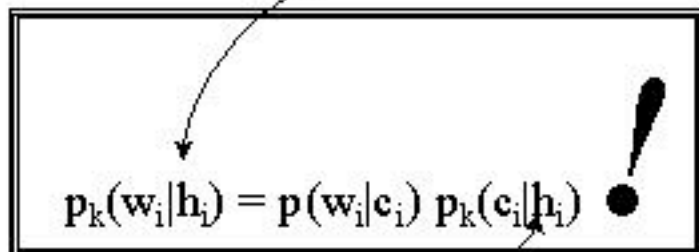
- Observation: similar words behave in a similar way
  - trigram LM:
    - in the ... (all nouns/adj);
    - catch a ... (all things which can be caught, incl. their accompanying adjectives);
  - trigram LM, conditioning:
    - a ... homework (any attribute of homework: short, simple, late, difficult),
    - ... the woods (any verb that has the woods as an object: walk, cut, save)
  - trigram LM: both:
    - a (short,long,difficult,...) (homework,assignment,task,job,...)

# Solution

- Use the Word Classes as the “reliability” measure
- Example: we see
  - short homework, short assignment, simple homework
  - but not:
    - simple assignment
  - Cluster into classes:
    - (short, simple) (homework, assignment)
      - covers “simple assignment”, too
- Gaining: realistic estimates for unseen n-grams
- Loosing: accuracy (level of detail) within classes

# The New Model

- Rewrite the n-gram LM using classes:
  - Was:  $[k = 1..n]$ 
    - $p_k(w_i|h_j) = c(h_i, w_i) / c(h_j)$  [history: (k-1) words]
  - Introduce classes:


$$p_k(w_i|h_j) = p(w_i|c_i) p_k(c_i|h_i) !$$

- history: classes, too: [for trigram:  $h_1 = c_{i-2}, c_{i-1}$ , bigram:  $h_1 = c_{i-1}$ ]
- Smoothing as usual
  - over  $p_k(w_i|h_j)$ , where each is defined as above (except uniform which stays at  $1/V$ )

# Training Data

- Suppose we already have a mapping:
  - $r: V \rightarrow C$  assigning each word its class ( $c_i = r(w_i)$ )
- Expand the training data:
  - $T = (w_1, w_2, \dots, w_{|T|})$  into
  - $T_C = (\langle w_1, r(w_1) \rangle, \langle w_2, r(w_2) \rangle, \dots, \langle w_{|T|}, r(w_{|T|}) \rangle)$
- Effectively, we have two streams of data:
  - word stream:  $w_1, w_2, \dots, w_{|T|}$
  - class stream:  $c_1, c_2, \dots, c_{|T|}$  (def. as  $c_i = r(w_i)$ )
- Expand Heldout, Test data too

## Training the New Model

- As expected, using ML estimates:
  - $p(w_i|c_i) = p(w_i|r(w_i)) = c(w_i) / c(r(w_i)) = c(w_i) / c(c_i)$ 
    - !!!  $c(w_i, c_i) = c(w_i)$  [since  $c_i$  determined by  $w_i$ ]
  - $p_k(c_i|h_i)$ :
    - $p_3(c_i|h_i) = p_3(c_i|c_{i-2}, c_{i-1}) = c(c_{i-2}, c_{i-1}, c_i) / c(c_{i-2}, c_{i-1})$
    - $p_2(c_i|h_i) = p_2(c_i|c_{i-1}) = c(c_{i-1}, c_i) / c(c_{i-1})$
    - $p_1(c_i|h_i) = p_1(c_i) = c(c_i) / |T|$
- Then smooth as usual
  - not the  $p(w_i|c_i)$  nor  $p_k(c_i|h_i)$  individually, but the  $p_k(w_i|h_i)$



# Classes: How To Get Them

- We supposed the classes are given
- Maybe there are in [human] dictionaries, but...
  - dictionaries are incomplete
  - dictionaries are unreliable
  - do not define classes as equivalence relation (overlap)
  - do not define classes suitable for LM
    - small, short... maybe; small and difficult?
- → we have to construct them from data (again...)

## Creating the Word-to-Class Map

- We will talk about bigrams from now
- Bigram estimate:
  - $p_2(c_i|h_i) = p_2(c_i|c_{i-1}) = c(c_{i-1}, c_i) / c(c_{i-1}) = c(r(w_{i-1}), r(w_i)) / c(r(w_{i-1}))$
- Form of the model:
  - just raw bigram for now:
    - $P(T) = \prod_{i=1..|T|} p(w_i|r(w_i)) p_2(r(w_i)|r(w_{i-1}))$  ( $p_2(c_1|c_0) =_{df} p(c_1)$ )
- Maximize over  $r$  (given  $r \rightarrow$  fixed  $p, p_2$ ):
  - define objective  $L(r) = 1/|T| \sum_{i=1..|T|} \log(p(w_i|r(w_i)) p_2(r(w_i)|r(w_{i-1})))$
  - $r_{best} = \operatorname{argmax}_r L(r)$  ( $L(r) =$  norm. logprob of training data... as usual)

## Simplifying the Objective Function

- Start from  $L(r) = 1/|T| \sum_{i=1..|T|} \log(p(w_i|r(w_i)) p_2(r(w_i)|r(w_{i-1})))$ :
 
$$1/|T| \sum_{i=1..|T|} \log(p(w_i|r(w_i)) \underline{p(r(w_i))} p_2(r(w_i)|r(w_{i-1})) / \underline{p(r(w_i))}) =$$

$$1/|T| \sum_{i=1..|T|} \log(\underline{p(w_i, r(w_i))} p_2(r(w_i)|r(w_{i-1})) / \underline{p(r(w_i))}) =$$

$$1/|T| \sum_{i=1..|T|} \log(\underline{p(w_i)}) + 1/|T| \sum_{i=1..|T|} \log(p_2(r(w_i)|r(w_{i-1})) / \underline{p(r(w_i))}) =$$

$$-H(W) + 1/|T| \sum_{i=1..|T|} \log(p_2(r(w_i)|r(w_{i-1})) \underline{p(r(w_{i-1}))} / (\underline{p(r(w_{i-1}))} \underline{p(r(w_i))})) =$$

$$-H(W) + 1/|T| \sum_{i=1..|T|} \log(\underline{p(r(w_i), r(w_{i-1}))} / (\underline{p(r(w_{i-1}))} \underline{p(r(w_i))})) =$$

$$-H(W) + \sum_{d,e \in C} p(d,e) \log( p(d,e) / (p(d) p(e)) ) =$$

$$-H(W) + I(D,E)$$

(event E picks class adjacent (to the right) to the one picked by D)
- Since  $W$  does not depend on  $r$ , we ended up with  $I(D,E)$ .  
the need to maximize  $\rightarrow$

# Maximizing Mutual Information

(dependent on the mapping  $r$ )

- Result from previous foil:
  - Maximizing the probability of data amounts to maximizing  $I(D,E)$ , the mutual information of the adjacent classes.
- Good:
  - We know what a MI is, and we know how to maximize.
- Bad:
  - There is no way how to maximize over so many possible partitionings:  $|V|^{|V|}$  - no way to test them all.

# Training or Heldout?

- Training:
  - best  $I(D,E)$ : all words in a class of its own
    - will not give us anything new.
- Heldout: ok, but:
  - must smooth to test any possible partitioning (unfeasible):
    - using raw model: 0 probability of heldout (almost) guaranteed
      - will not be able to compare anything
  - some smoothing estimates? (to be explored...)
- Solution:
  - use training anyway, but only keep  $I(D,E)$  as large as possible

# The Greedy Algorithm

- Define merging operation on the mapping  $r: V \rightarrow C$ :
  - merge:  $R \times C \times C \rightarrow R' \times C'$ :  $(r,k,l) \rightarrow r',C'$  such that
  - $C' = \{C - \{k,l\} \cup \{m\}\}$  (throw out  $k$  and  $l$ , add new  $m \notin C$ )
  - $r'(w) = \dots m$  for  $w \in r_{\text{INV}}(\{k,l\})$ ,  
 $\dots r(w)$  otherwise.
- 1. Start with each word in its own class ( $C = V$ ),  $r = \text{id}$ .
- 2. Merge two classes  $k,l$  into one,  $m$ , such that
$$(k,l) = \text{argmax}_{k,l} I_{\text{merge}(r,k,l)}(D,E).$$
- 3. Set new  $(r,C) = \text{merge}(r,k,l)$ .
- 4. Repeat 2 and 3 until  $|C|$  reaches predetermined size.

# Word Classes in Applications

- Word Sense Disambiguation: context not seen [enough(-times)]
- Parsing: verb-subject, verb-object relations
- Speech recognition (acoustic model): need more instances of [rare(r)] sequences of phonemes
- Machine Translation: translation equivalent selection [for rare(r) words]

# Markov Models



# Review: Markov Process

- Bayes formula (chain rule):

$$P(W) = P(w_1, w_2, \dots, w_T) = \prod_{i=1..T} p(w_i | w_1, w_2, \dots, w_{i-n+1}, \dots, w_{i-1})$$

- n-gram language models:

- Markov process (chain) of the order n-1:

$$P(W) = P(w_1, w_2, \dots, w_T) = \prod_{i=1..T} p(w_i | w_{i-n+1}, w_{i-n+2}, \dots, w_{i-1})$$

Using just one distribution (Ex.: trigram model:  $p(w_i | w_{i-2}, w_{i-1})$ ):

Positions: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Words: My car **broke down** and within hours Bob's car **broke down** too .

$$p(, | \mathbf{broke\ down}) = p(w_5 | w_3, w_4) = p(w_{14} | w_{12}, w_{13})$$

*approximation*

# Markov Properties

- Generalize to any process (not just words/LM):
  - Sequence of random variables:  $X = (X_1, X_2, \dots, X_T)$
  - Sample space  $S$  (*states*), size  $N$ :  $S = \{s_0, s_1, s_2, \dots, s_N\}$

## 1. Limited History (Context, Horizon):

$$\forall i \in 1..T; P(X_i | X_1, \dots, X_{i-1}) = P(X_i | X_{i-1})$$

## 2. Time invariance (M.C. is stationary, homogeneous)

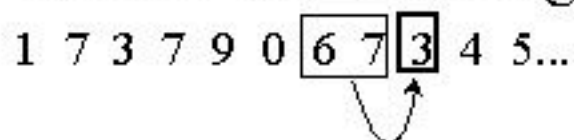
$$\forall i \in 1..T, \forall y, x \in S; P(X_i = y | X_{i-1} = x) = p(y|x)$$

1 7 3 7 9 0 6 7 3 4 5...

# Long History Possible

- What if we want trigrams:

1 7 3 7 9 0 6 7 3 4 5...



- Formally, use transformation:

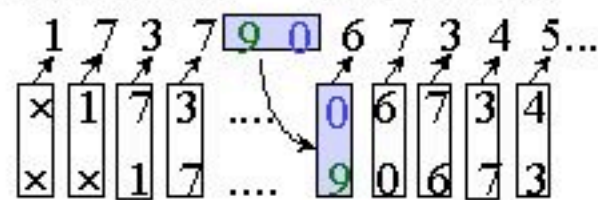
Define new variables  $Q_i$ , such that  $X_i = \{Q_{i-1}, Q_i\}$ :

Then

$$P(X_i | X_{i-1}) = P(Q_{i-1}, Q_i | Q_{i-2}, Q_{i-1}) = P(Q_i | Q_{i-2}, Q_{i-1})$$

Predicting ( $X_i$ ):

1 7 3 7 9 0 6 7 3 4 5...



History ( $X_{i-1} = \{Q_{i-2}, Q_{i-1}\}$ ):

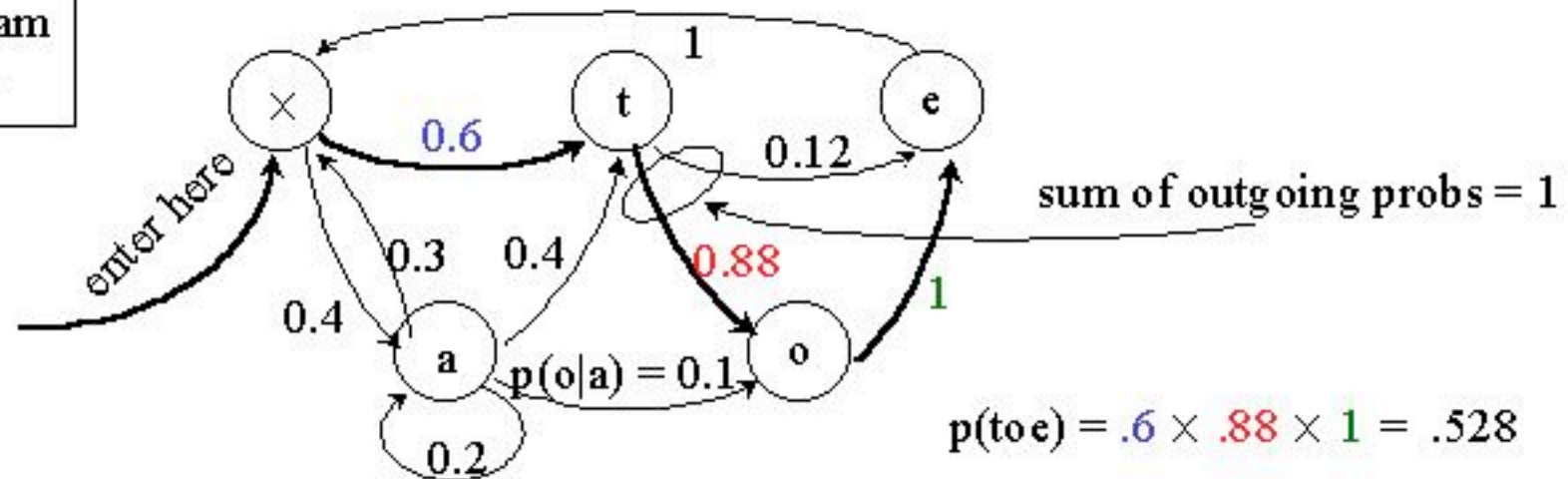
x 1 7 3 ... 0 6 7 3 4

x x 1 7 ... 9 0 6 7 3

# Graph Representation: State Diagram

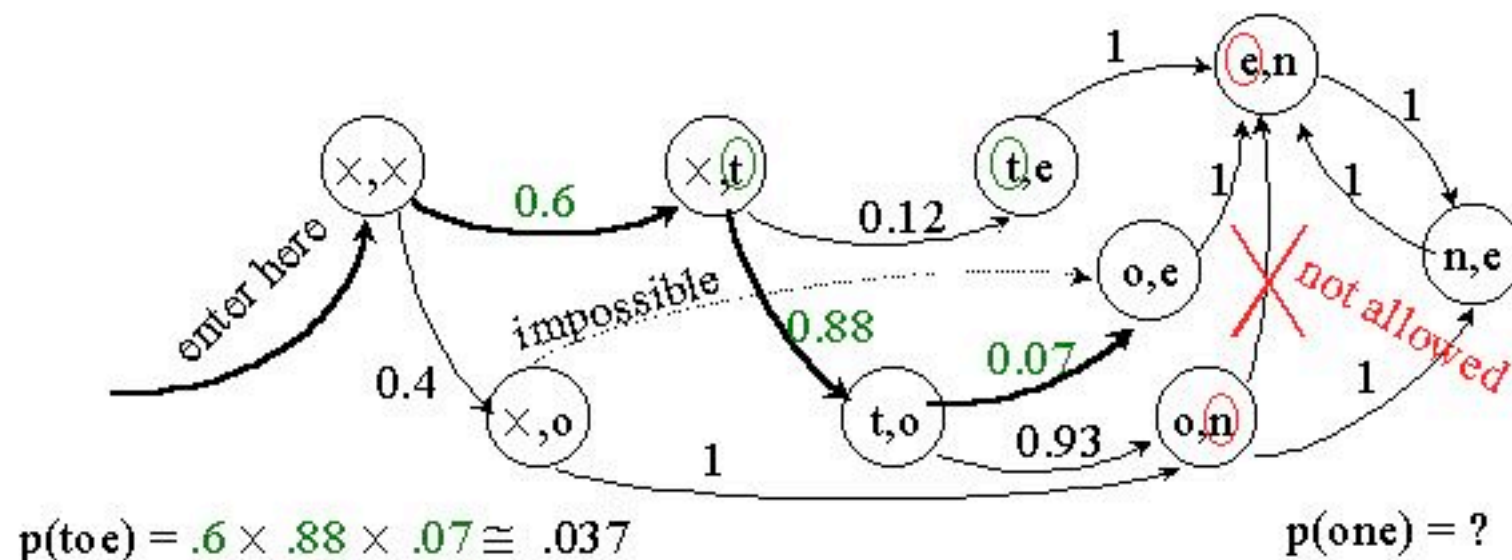
- $S = \{s_0, s_1, s_2, \dots, s_N\}$ : states
- Distribution  $P(X_i | X_{i-1})$ :
  - transitions (as arcs) with probabilities attached to them:

Bigram  
case:

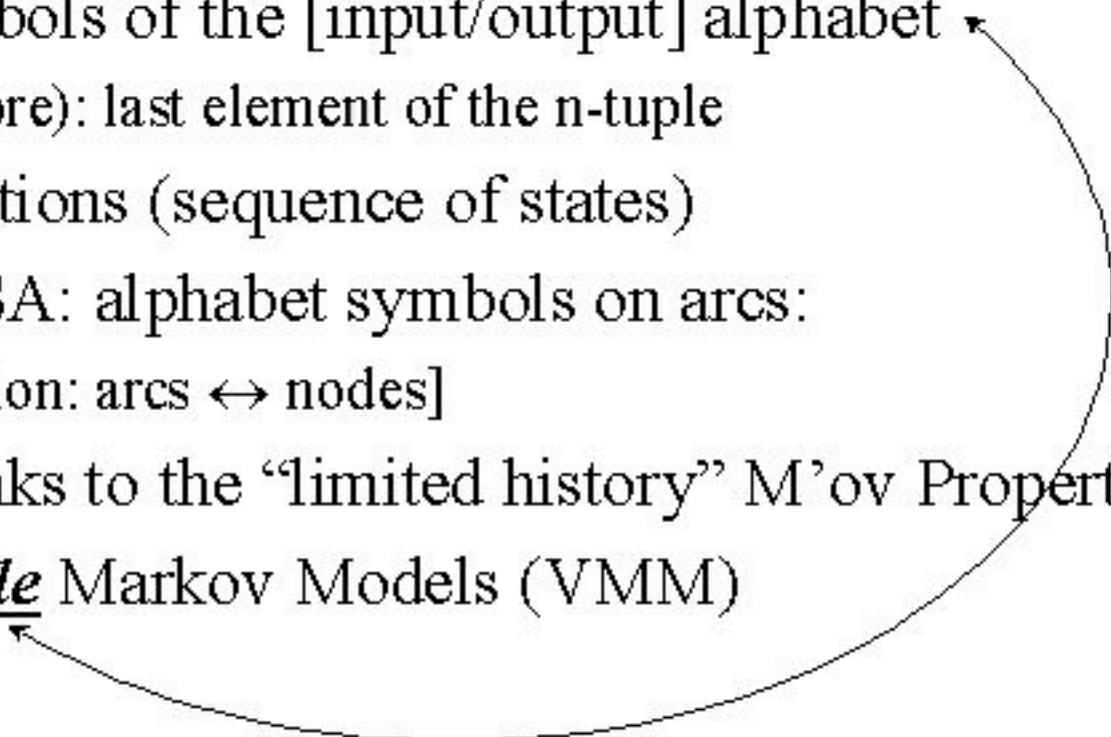


# The Trigram Case

- $S = \{s_0, s_1, s_2, \dots, s_N\}$ : states: pairs  $s_i = (x, y)$
- Distribution  $P(X_i | X_{i-1})$ : (r.v.  $X$ : generates pairs  $s_i$ )

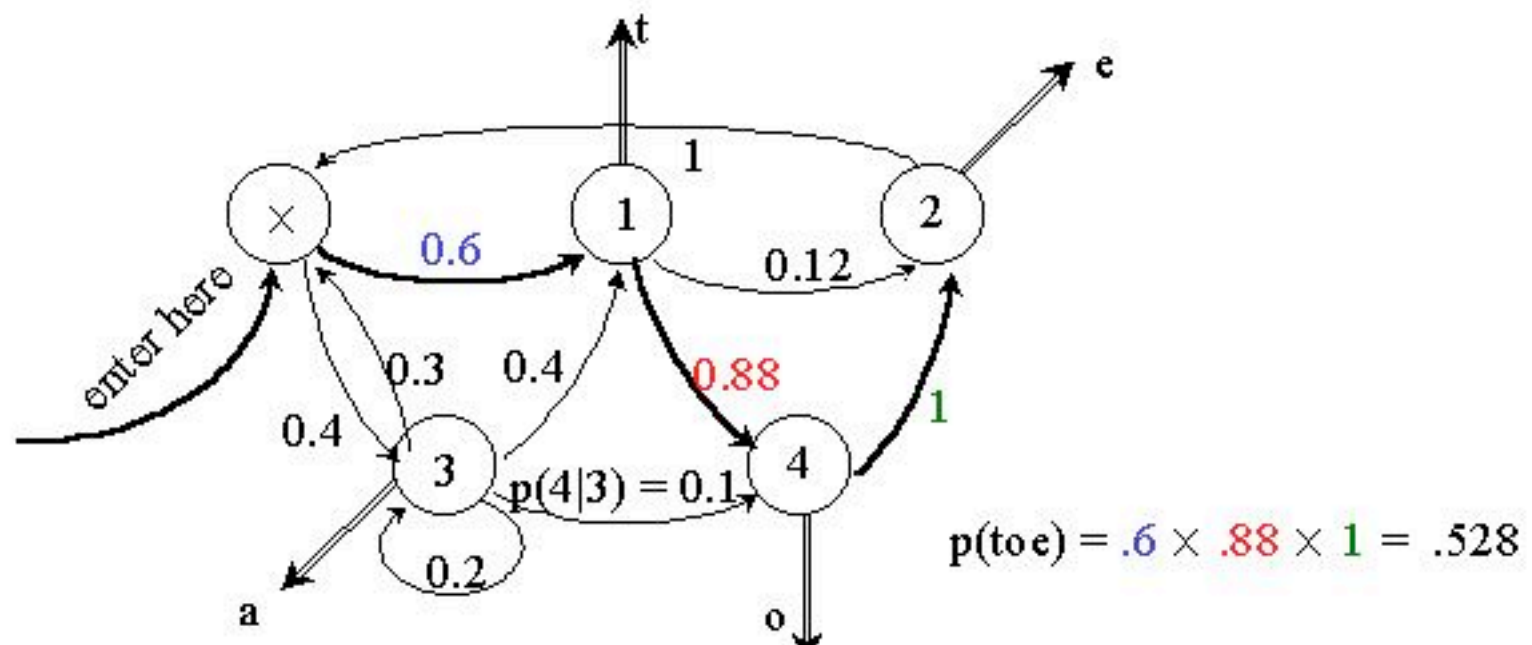


# Finite State Automaton

- States  $\sim$  symbols of the [input/output] alphabet
    - pairs (or more): last element of the n-tuple
  - Arcs  $\sim$  transitions (sequence of states)
  - [Classical FSA: alphabet symbols on arcs:
    - transformation: arcs  $\leftrightarrow$  nodes]
  - Possible thanks to the “limited history” Markov Property
  - So far: Visible Markov Models (VMM)
- 

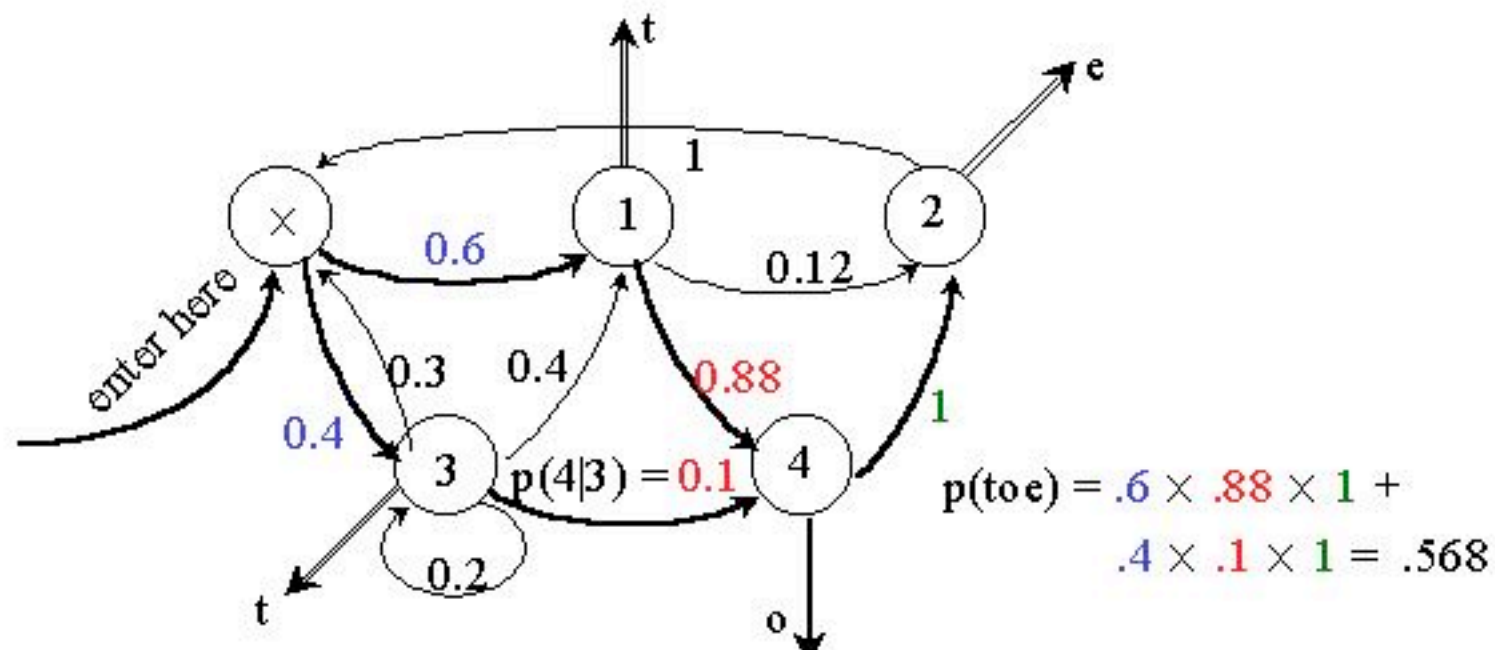
# Hidden Markov Models

- The simplest HMM: states generate [observable] output (using the “data” alphabet) but remain “invisible”:



# Added Flexibility

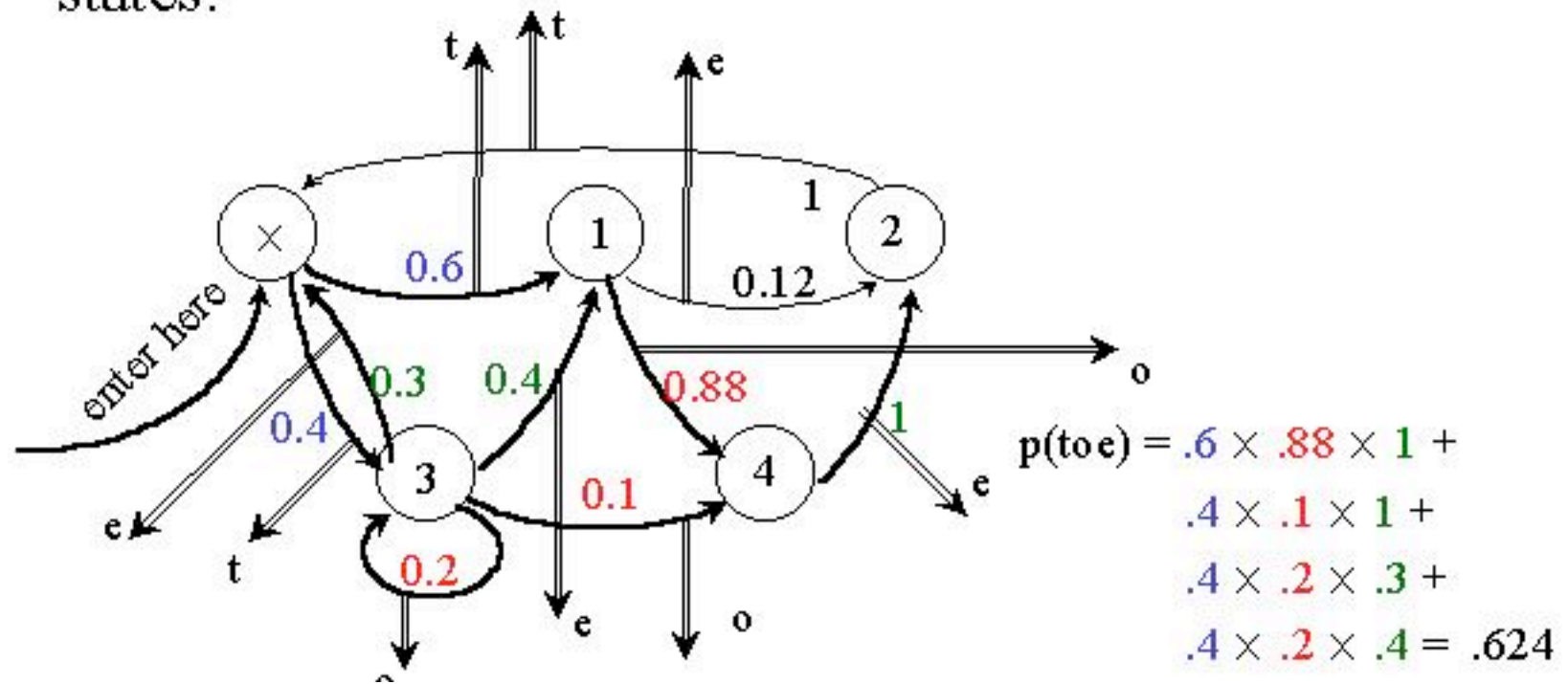
- So far, no change; but different states may generate the same output (why not?):





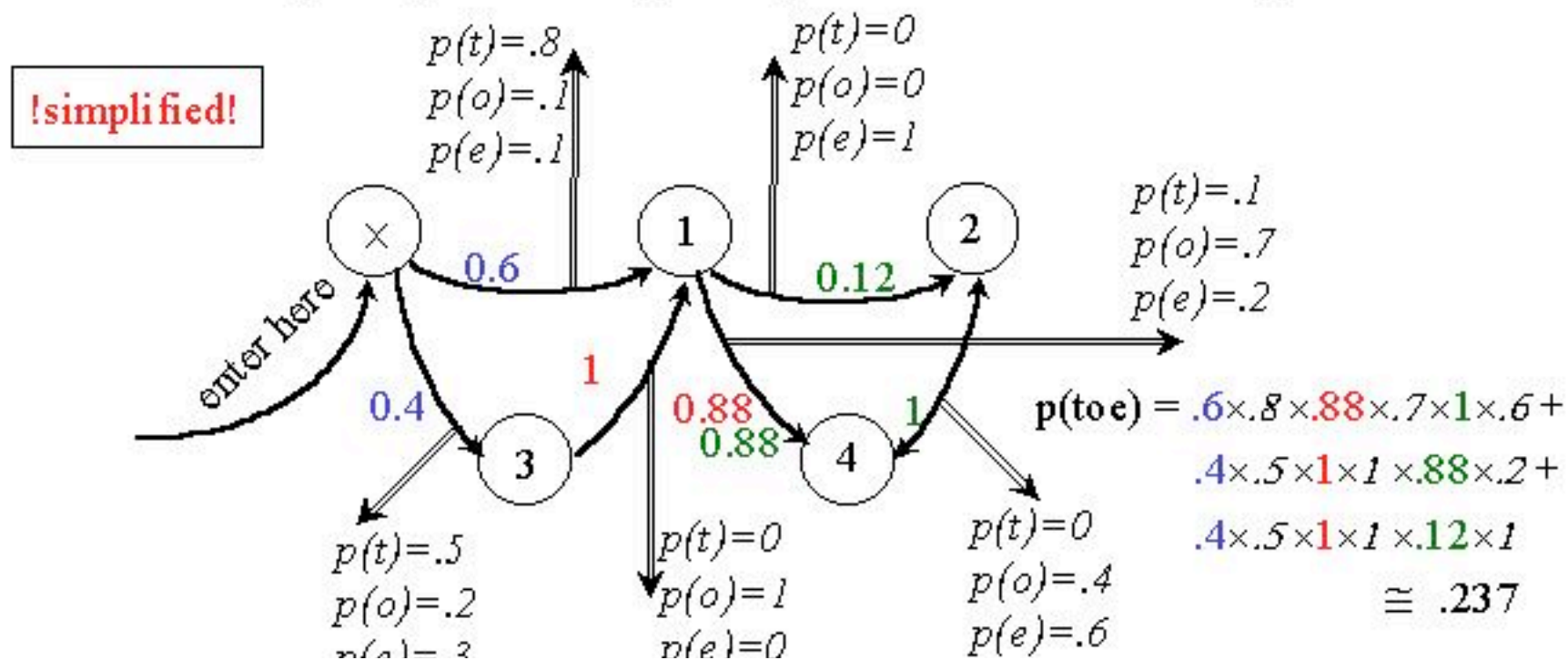
# Output from Arcs...

- Added flexibility: Generate output from arcs, not states:



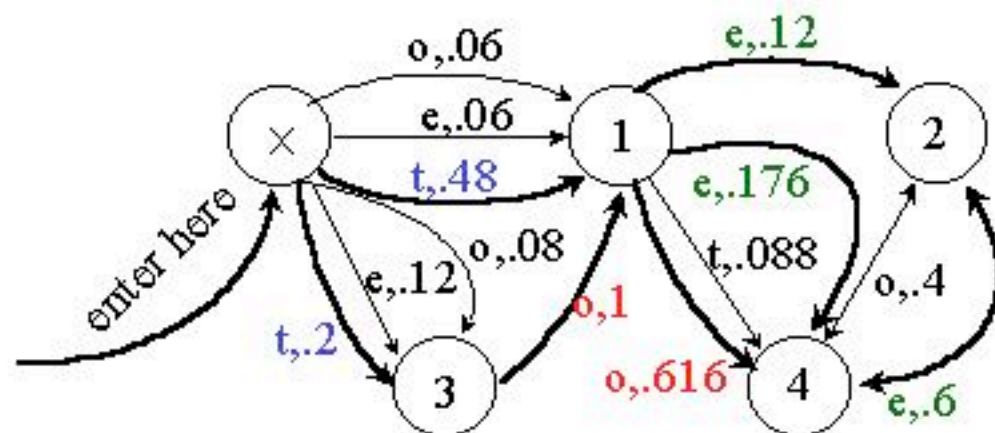
## ... and Finally, Add Output Probabilities

- Maximum flexibility: [Unigram] distribution (sample space: output alphabet) at each output arc:



## Slightly Different View

- Allow for multiple arcs from  $s_i \rightarrow s_j$ , mark them by output symbols, get rid of output distributions:



$$\begin{aligned}
 p(\text{toe}) &= .48 \times .616 \times .6 + \\
 &\quad .2 \times 1 \times .176 + \\
 &\quad .2 \times 1 \times .12 \cong .237
 \end{aligned}$$

In the future, we will use the view more convenient for the problem at hand.

# Formalization

- HMM (the most general case):
  - five-tuple  $(S, s_0, Y, P_S, P_Y)$ , where:
    - $S = \{s_0, s_1, s_2, \dots, s_T\}$  is the set of states,  $s_0$  is the initial state,
    - $Y = \{y_1, y_2, \dots, y_V\}$  is the output alphabet,
    - $P_S(s_j | s_i)$  is the set of prob. distributions of transitions,
      - size of  $P_S$ :  $|S|^2$ .
    - $P_Y(y_k | s_i, s_j)$  is the set of output (emission) probability distributions.
      - size of  $P_Y$ :  $|S|^2 \times |Y|$
- Example:
  - $S = \{x, 1, 2, 3, 4\}$ ,  $s_0 = x$
  - $Y = \{t, o, e\}$

# Formalization - Example

- Example (for graph, see foils 11,12):

- $S = \{x, 1, 2, 3, 4\}$ ,  $s_0 = x$

- $Y = \{e, o, t\}$

- $P_S$ :

	x	1	2	3	4
x	0	.6	0	.4	0
1	0	0	.12	0	.88
2	0	0	0	0	1
3	0	1	0	0	0
4	0	0	1	0	0

→  $\Sigma = 1$

$P_Y$ :

	e	x	1	2	3	4
o		x	1	2	3	4
t		x	1	2	3	4
x		.8		.5		.2
1			0		.1	
2					0	
3		0				
4			0			

↗  $\Sigma = 1$

# Using the HMM

- The generation algorithm (of limited value :-)):
  1. Start in  $s = s_0$ .
  2. Move from  $s$  to  $s'$  with probability  $P_S(s'|s)$ .
  3. Output (emit) symbol  $y_k$  with probability  $P_S(y_k|s,s')$ .
  4. Repeat from step 2 (until somebody says enough).
- More interesting usage:
  - Given an output sequence  $Y = \{y_1, y_2, \dots, y_k\}$ , compute its probability.
  - Given an output sequence  $Y = \{y_1, y_2, \dots, y_k\}$ , compute the most likely sequence of states which has generated it.
  - ...plus variations: e.g.,  $n$  best state sequences

# HMM algorithms: Trellis and Viterbi

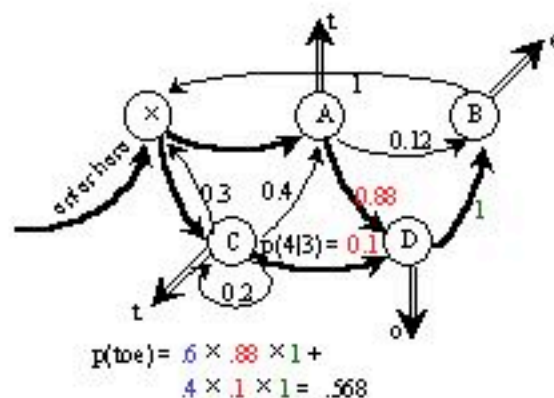
# HMM: The Two Tasks

- HMM (the general case):
  - five-tuple  $(S, S_0, Y, P_S, P_Y)$ , where:
    - $S = \{s_1, s_2, \dots, s_T\}$  is the set of states,  $S_0$  is the initial state,
    - $Y = \{y_1, y_2, \dots, y_V\}$  is the output alphabet,
    - $P_S(s_j | s_i)$  is the set of prob. distributions of transitions,
    - $P_Y(y_k | s_i, s_j)$  is the set of output (emission) probability distributions.
- Given an HMM & an output sequence  $Y = \{y_1, y_2, \dots, y_k\}$ :
  - (Task 1) compute the probability of  $Y$ ;
  - (Task 2) compute the most likely sequence of states which has generated  $Y$ .



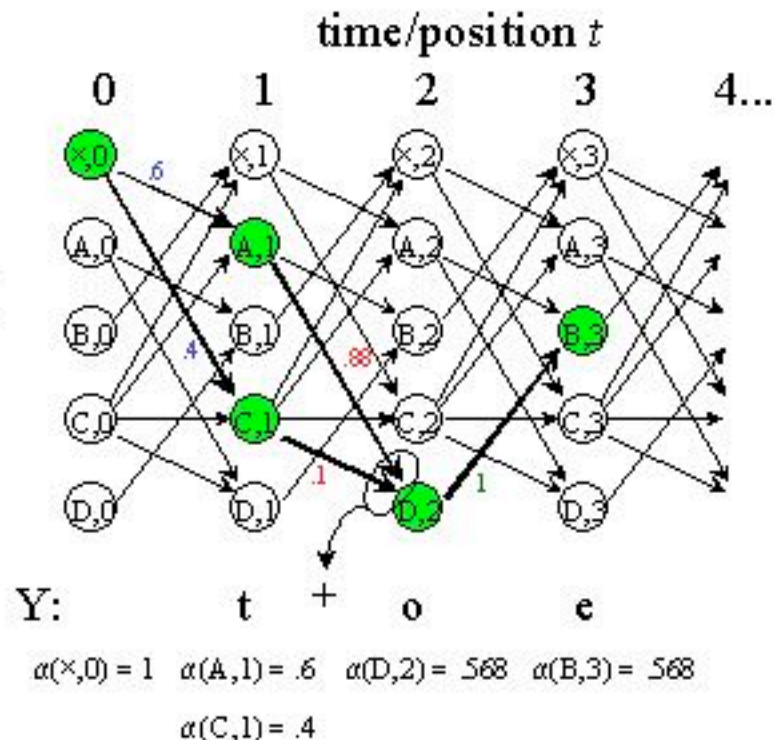
# Trellis - Deterministic Output

HMM:



Trellis:

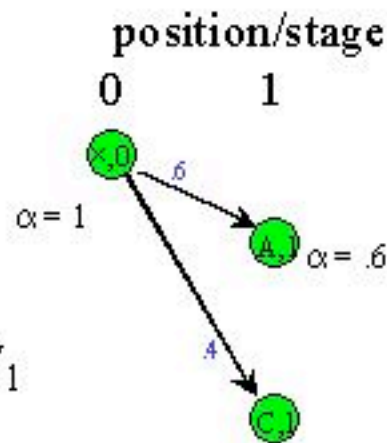
“rollout”



- trellis state: (HMM state, position)
- each state: holds one number (prob):  $\alpha$
- probability of Y:  $\sum \alpha$  in the last state

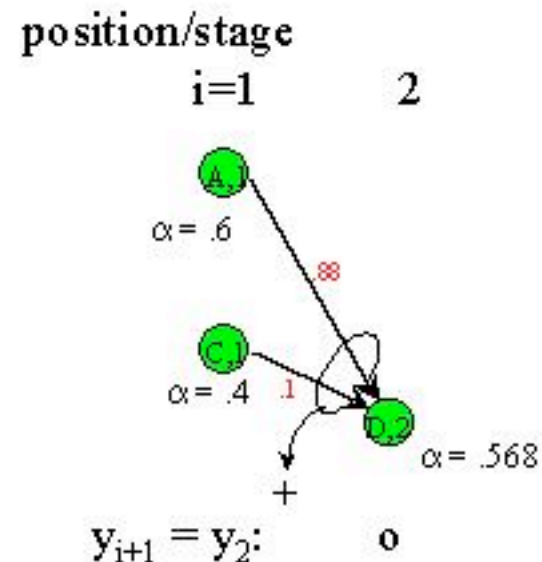
# Creating the Trellis: The Start

- Start in the start state ( $\times$ ),
  - set its  $\alpha(\times, 0)$  to 1.
- Create the first stage:
  - get the first “output” symbol  $y_1$
  - create the first stage (column)
  - but only those trellis states which generate  $y_1$
  - set their  $\alpha(state, 1)$  to the  $P_S(state|\times) \underbrace{\alpha(\times, 0)}_1$
- ...and forget about the  $0$ -th stage



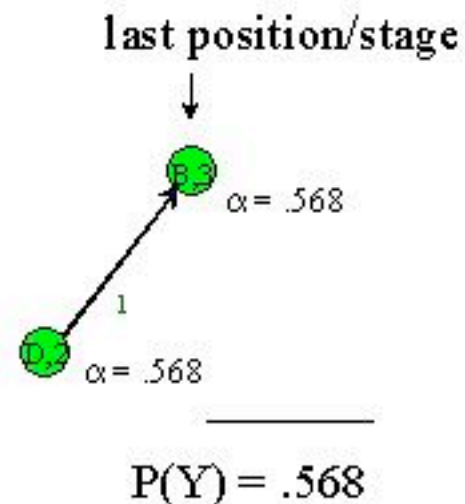
# Trellis: The Next Step

- Suppose we are in stage  $i$
- Creating the next stage:
  - create all trellis states in the next stage which generate  $y_{i+1}$ , but only those reachable from any of the stage- $i$  states
  - set their  $\alpha(state, i+1)$  to:
 
$$P_S(state|prev.state) \times \alpha(prev.state, i)$$
 (add up all such numbers on arcs going to a common trellis state)
  - ...and forget about stage  $i$



# Trellis: The Last Step

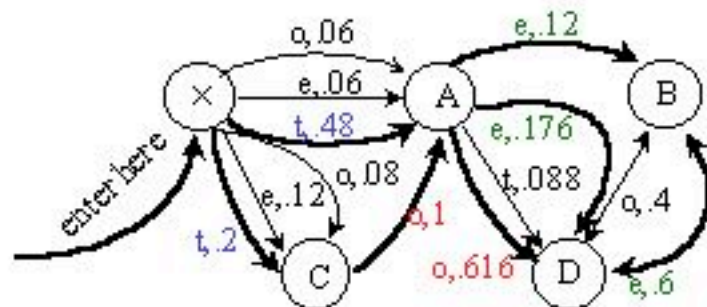
- Continue until “output” exhausted
  - $|Y| = 3$ : until stage 3
- Add together all the  $\alpha(state, |Y|)$
- That’s the  $\underline{P(Y)}$ .
- Observation (pleasant):
  - memory usage max:  $2|S|$
  - multiplications max:  $|S|^2|Y|$



# Trellis: The General Case (still, bigrams)

- Start as usual:
  - start state ( $\times$ ), set its  $\alpha(\times, \theta)$  to 1.

$\alpha = 1$

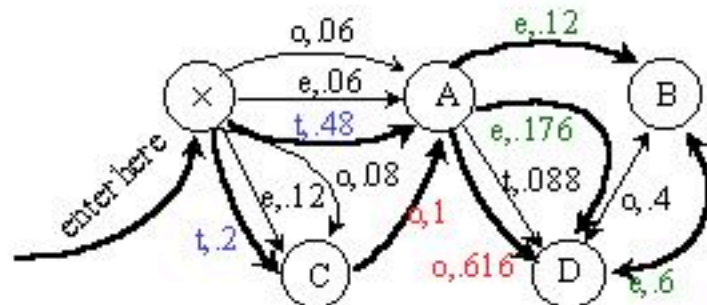
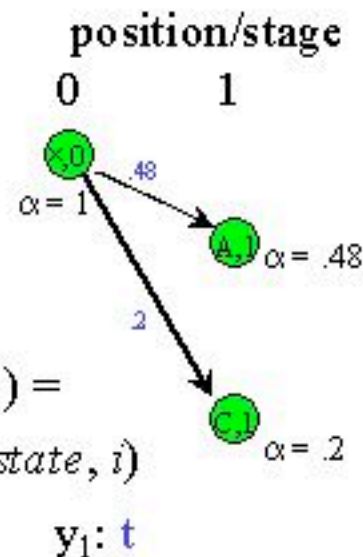


$$\begin{aligned}
 p(\text{toe}) &= .48 \times .616 \times .6 + \\
 &\quad .2 \times 1 \times .176 + \\
 &\quad .2 \times 1 \times .12 \cong .237
 \end{aligned}$$

# General Trellis: The Next Step

- We are in stage  $i$  :
  - Generate the next stage  $i+1$  as before (except now arcs generate output, thus use only those arcs marked by the output symbol  $y_{i+1}$ )

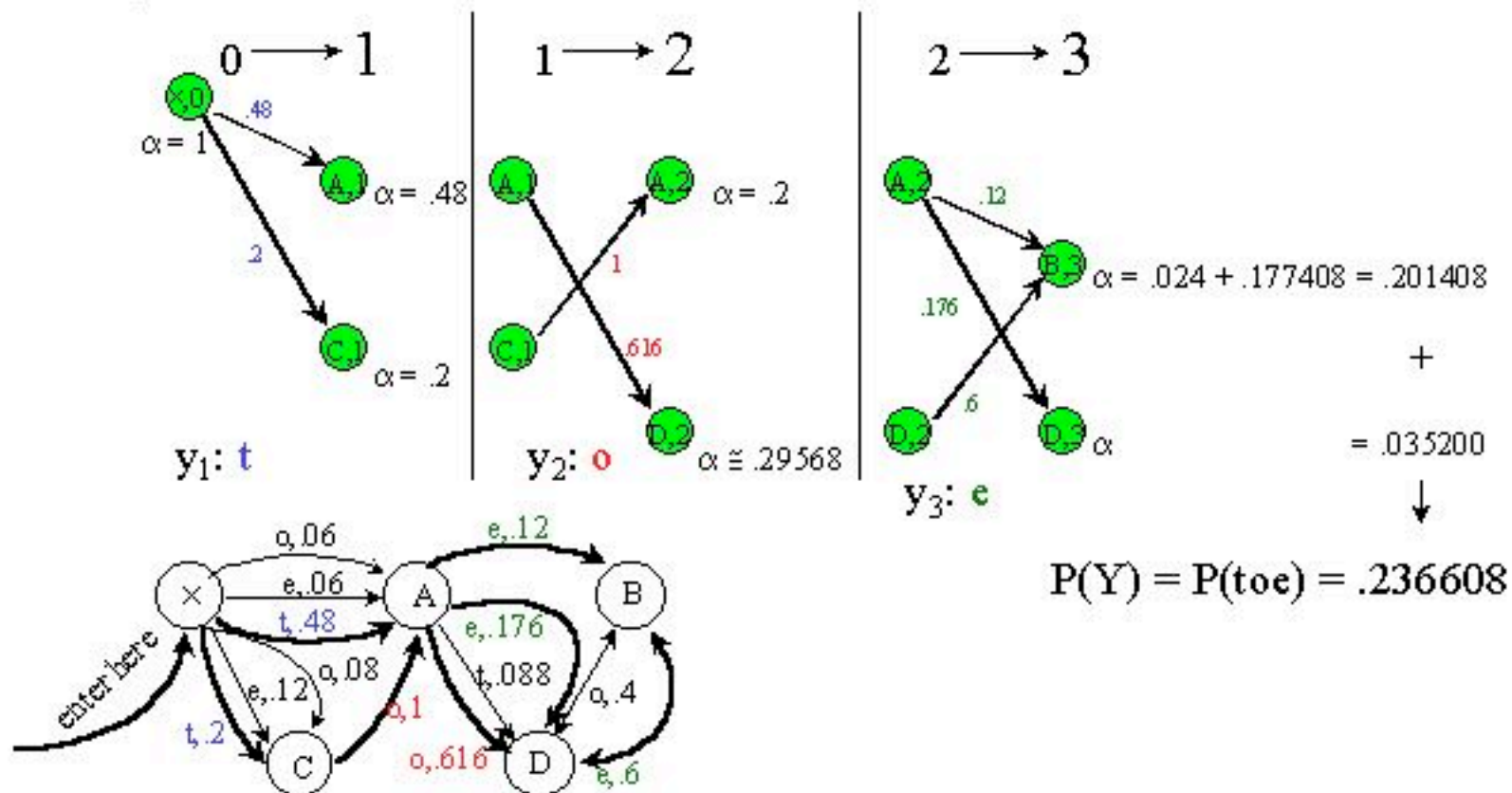
- For each generated *state*, compute  $\alpha(\text{state}, i+1) = \sum_{\text{incoming arcs}} P_Y(y_{i+1} | \text{state}, \text{prev.state}) \times \alpha(\text{prev.state}, i)$



...and forget about stage  $i$  as usual.

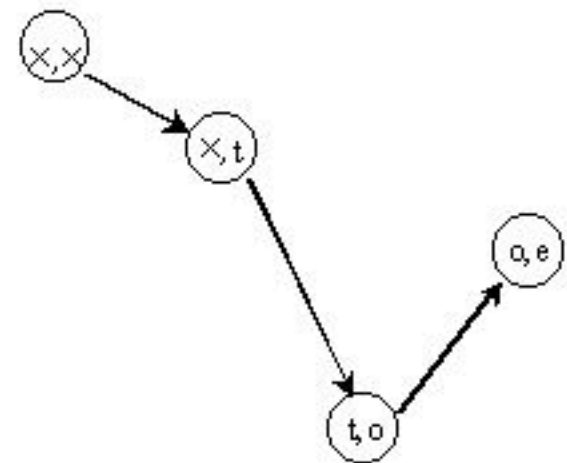
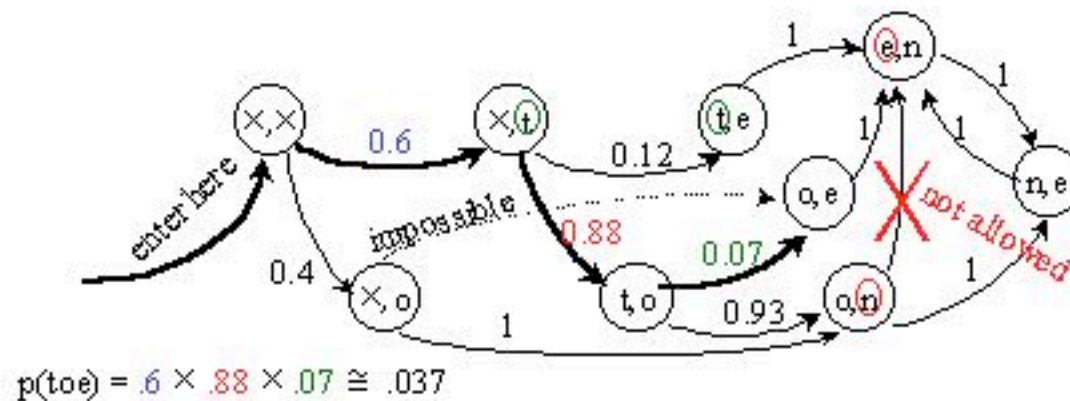
# Trellis: The Complete Example

Stage:



# The Case of Trigrams

- Like before, but:
  - states correspond to bigrams,
  - output function always emits the second output symbol of the pair (state) to which the arc goes:



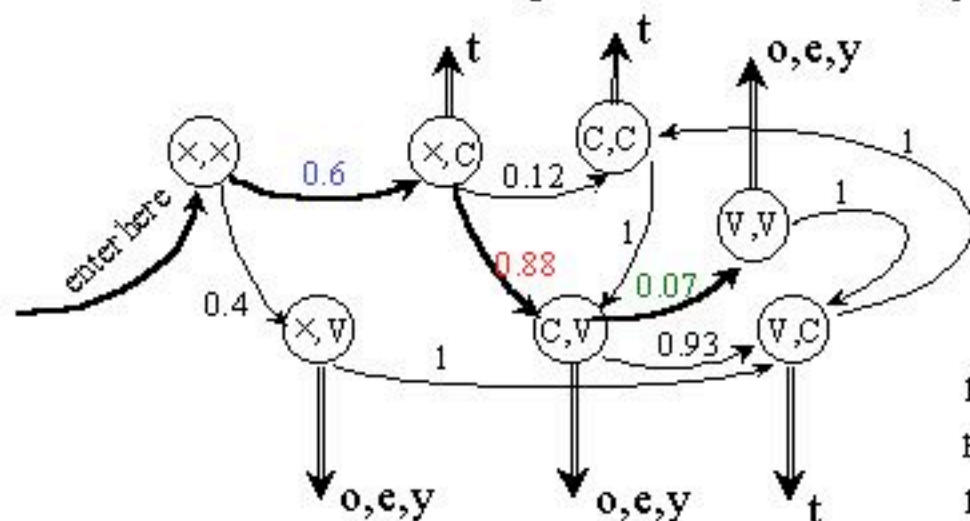
Multiple paths not possible → trellis not really needed



# Trigrams with Classes

- More interesting:
  - n-gram class LM:  $p(w_i | w_{i-2}, w_{i-1}) = p(w_i | c_i) p(c_i | c_{i-2}, c_{i-1})$
  - states are pairs of classes  $(c_{i-1}, c_i)$ , and emit “words”:

(letters in our example)



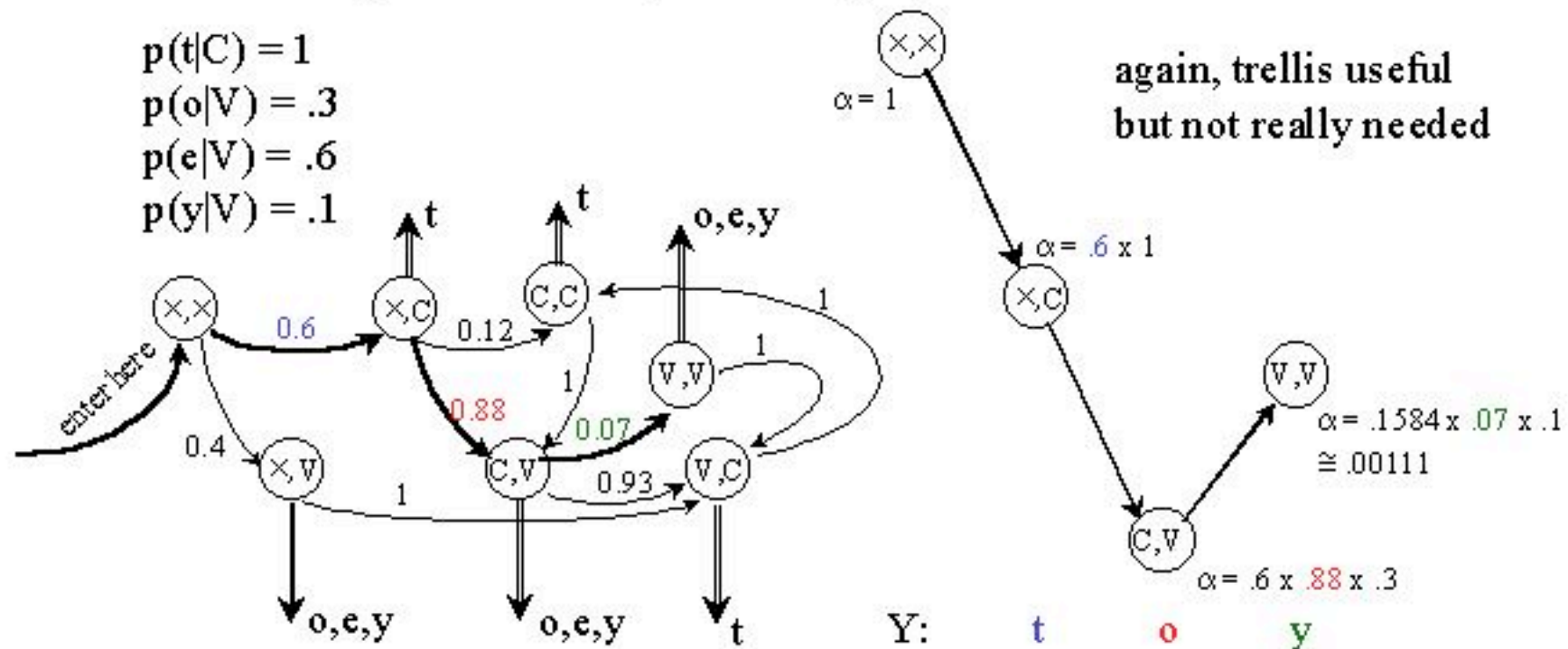
$p(t|C) = 1$       usual,  
 $p(o|V) = .3$       non-  
 $p(e|V) = .6$       overlapping  
 $p(y|V) = .1$       classes

$$\begin{aligned}
 p(\text{toe}) &= .6 \times 1 \times .88 \times .3 \times .07 \times .6 \cong .00665 \\
 p(\text{teo}) &= .6 \times 1 \times .88 \times .6 \times .07 \times .3 \cong .00665 \\
 p(\text{toy}) &= .6 \times 1 \times .88 \times .3 \times .07 \times .1 \cong .00111 \\
 p(\text{ttv}) &= .6 \times 1 \times .12 \times 1 \times 1 \times .1 \cong .0072
 \end{aligned}$$

# Class Trigrams: the Trellis

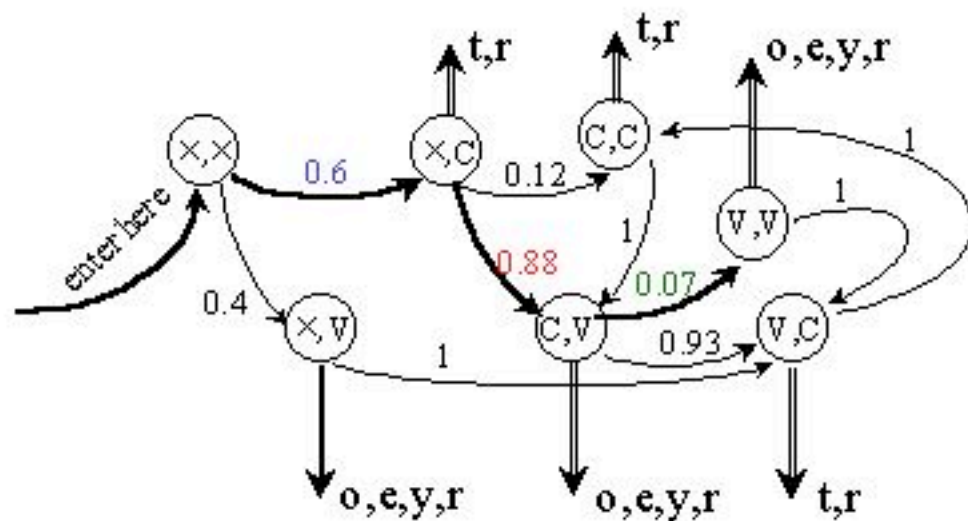
- Trellis generation (Y = "toy"):

$$\begin{aligned}
 p(t|C) &= 1 \\
 p(o|V) &= .3 \\
 p(e|V) &= .6 \\
 p(y|V) &= .1
 \end{aligned}$$



# Overlapping Classes

- Imagine that classes may overlap
  - e.g. 'r' is sometimes vowel sometimes consonant, belongs to V as well as C:



$$\begin{aligned}
 p(t|C) &= .3 \\
 p(r|C) &= .7 \\
 p(o|V) &= .1 \\
 p(e|V) &= .3 \\
 p(y|V) &= .4 \\
 p(r|V) &= .2
 \end{aligned}$$

$$p(\text{try}) = ?$$

# Overlapping Classes: Trellis Example

$$p(t|C) = .3$$

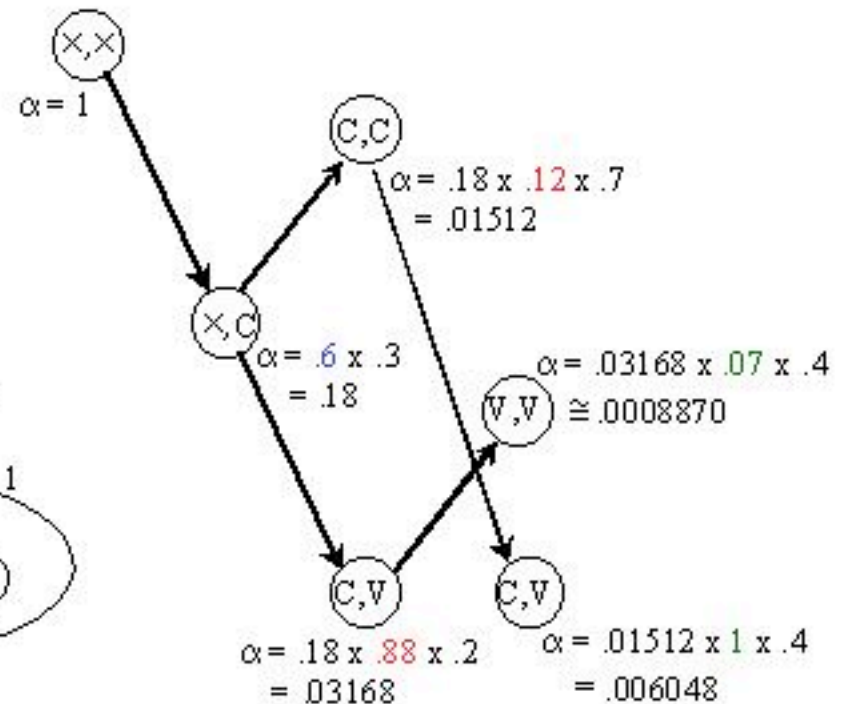
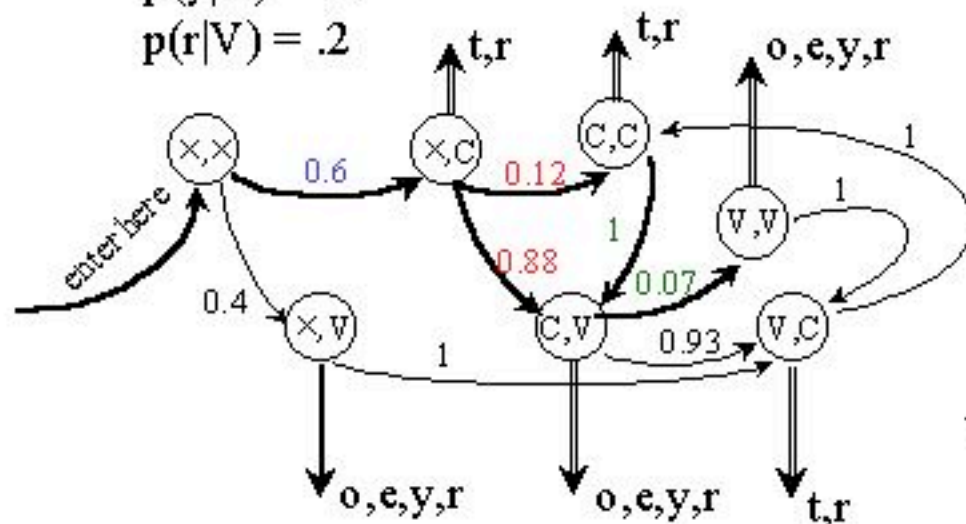
$$p(r|C) = .7$$

$$p(o|V) = .1$$

$$p(e|V) = .3$$

$$p(y|V) = .4$$

$$p(r|V) = .2$$



Y: **t** **r** **y**  $p(Y) = \underline{.006935}$

## Trellis: Remarks

- So far, we went left to right (computing  $\alpha$ )
- Same result: going right to left (computing  $\beta$ )
  - supposed we know where to start (finite data)
- In fact, we might start in the middle going left and right
- Important for parameter estimation  
(Forward-Backward Algorithm alias Baum-Welch)
- Implementation issues:
  - scaling/normalizing probabilities, to avoid too small numbers  
& addition problems with many transitions

# The Viterbi Algorithm

- Solving the task of finding the most likely sequence of states which generated the observed data
- i.e., finding

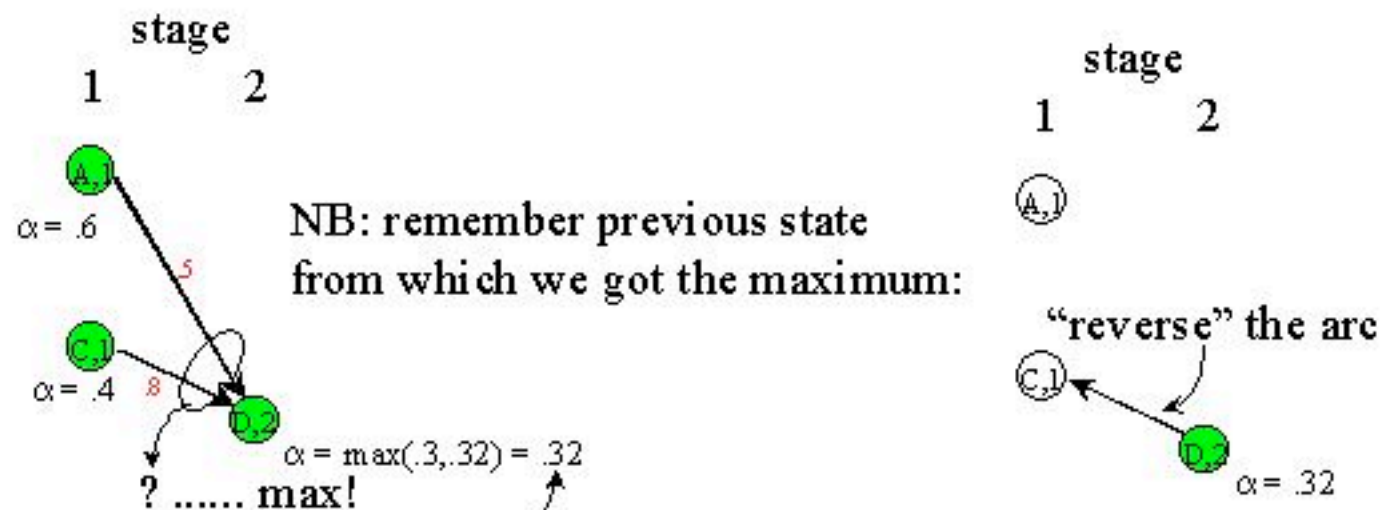
$$S_{\text{best}} = \operatorname{argmax}_S P(S|Y)$$

which is equal to (Y is constant and thus P(Y) is fixed):

$$\begin{aligned} S_{\text{best}} &= \operatorname{argmax}_S P(S, Y) = \\ &= \operatorname{argmax}_S P(s_0, s_1, s_2, \dots, s_k, y_1, y_2, \dots, y_k) = \\ &= \operatorname{argmax}_S \prod_{i=1..k} p(y_i | s_i, s_{i-1}) p(s_i | s_{i-1}) \end{aligned}$$

# The Crucial Observation

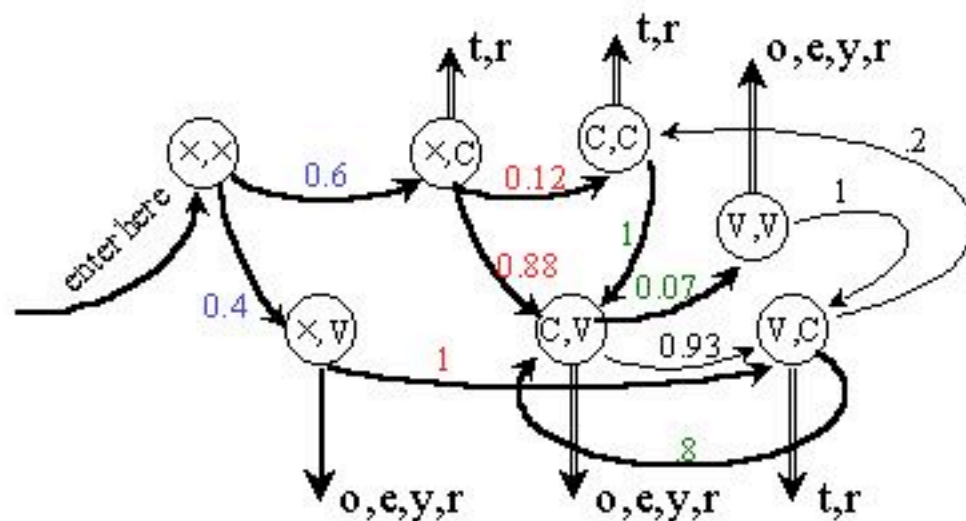
- Imagine the trellis build as before (but do not compute the  $\alpha$ s yet; assume they are o.k.); stage  $i$ :



this is certainly the “backwards” maximum to (D,2)... but  
it cannot change even whenever we go forward (M. Property: Limited History)

# Viterbi Example

- 'r' classification (C or V?, sequence?):



$p(t|C) = .3$   
 $p(r|C) = .7$   
 $p(o|V) = .1$   
 $p(e|V) = .3$   
 $p(y|V) = .4$   
 $p(r|V) = .2$

$\operatorname{argmax}_{XYZ} p(rry|XYZ) = ?$

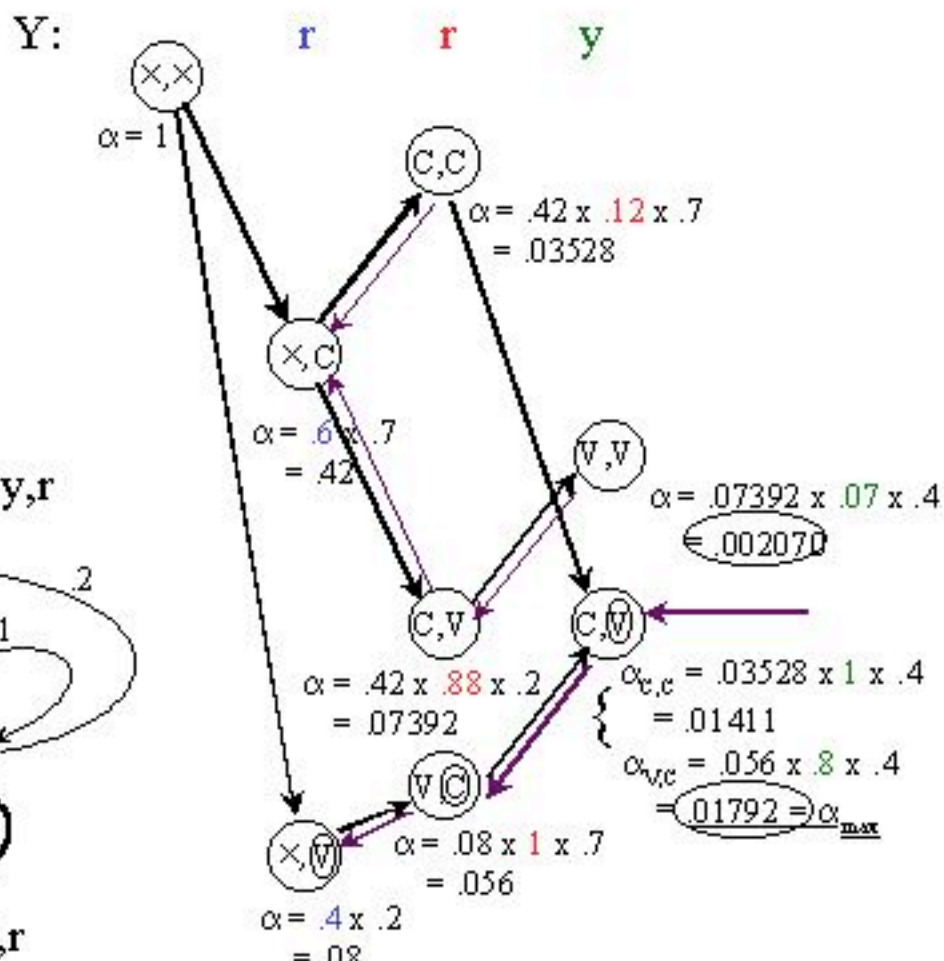
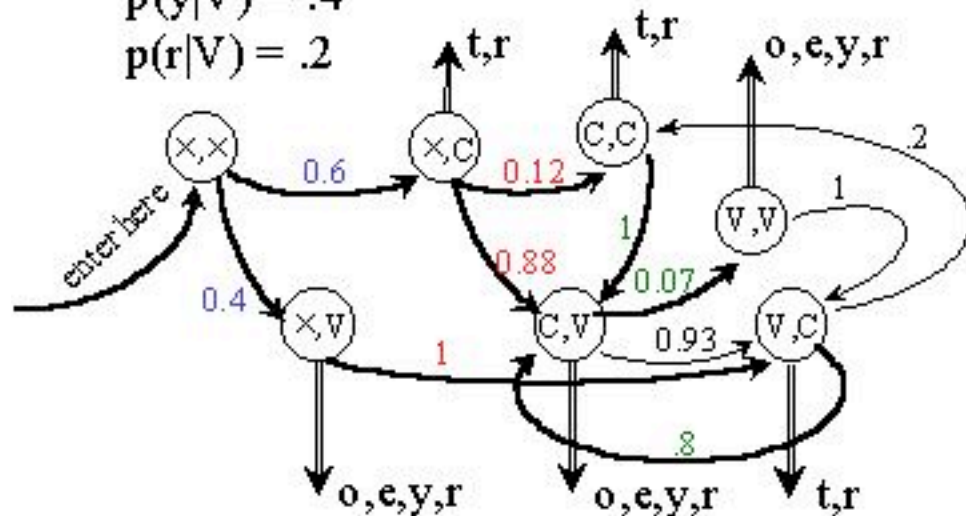
Possible state seq.:  $(x,v)(v,c)(c,v)[vcv]$ ,  $(x,c)(c,c)(c,v)[ccv]$ ,  $(x,c)(c,v)(v,v)[cvv]$



# Viterbi Computation

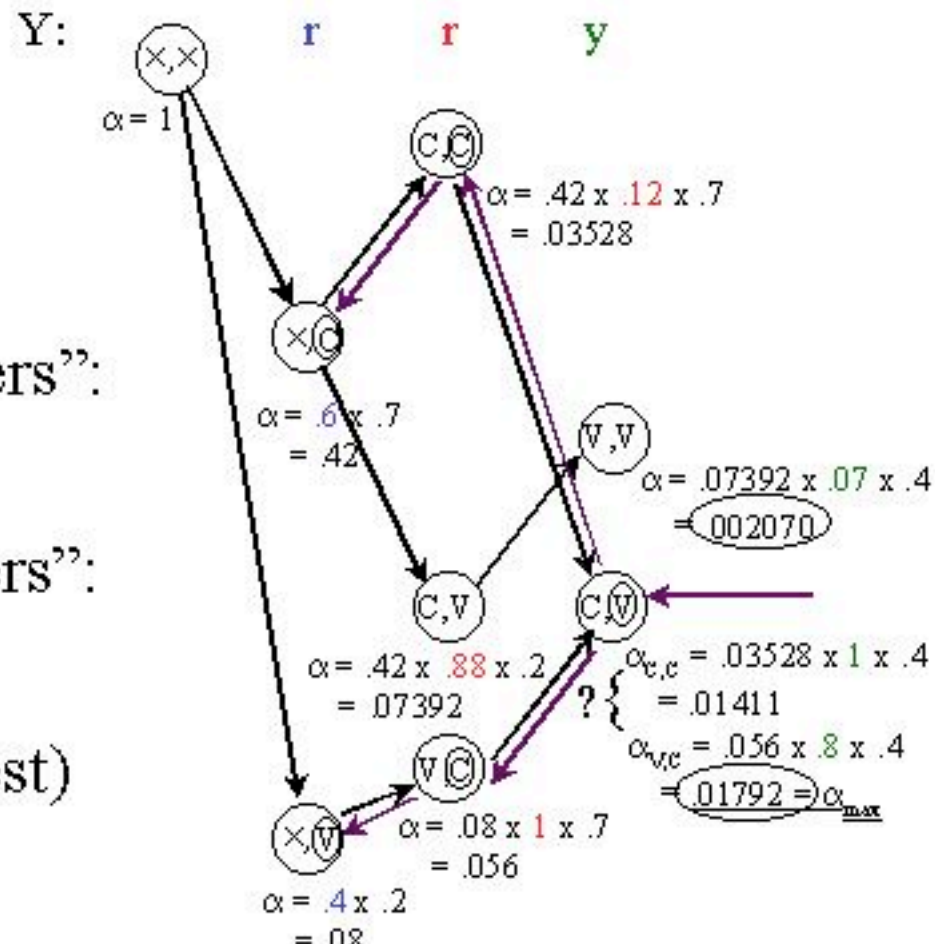
$p(t|C) = .3$   
 $p(r|C) = .7$   
 $p(o|V) = .1$   
 $p(e|V) = .3$   
 $p(y|V) = .4$   
 $p(r|V) = .2$

$\alpha$  in trellis  
 state:  
 best prob  
 from start  
 to here



# n-best State Sequences

- Keep track of n best “back pointers”:
- Ex.:  $n=2$ :  
Two “winners”:  
VCV (best)  
CCV (2<sup>nd</sup> best)

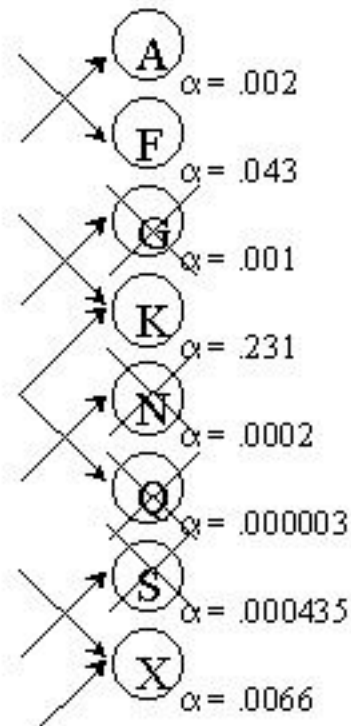


# Tracking Back the n-best paths

- Backtracking-style algorithm:
  - Start at the end, in the best of the n states ( $s_{best}$ )
  - Put the other n-1 best nodes/back pointer pairs on stack, except those leading from  $s_{best}$  to the same best-back state.
- Follow the back “beam” towards the start of the data, spitting out nodes on the way (backwards of course) using always only the best back pointer.
- At every beam split, push the diverging node/back pointer pairs onto the stack (node/beam width is sufficient!).
- When you reach the start of data, close the path, and pop the top-most node/back pointer(width) pair from the stack.
- Repeat until the stack is empty; expand the result tree if necessary.

# Pruning

- Sometimes, too many trellis states in a stage:



- criteria:
- (a)  $\alpha < \text{threshold}$
  - (b)  $\sum \pi < \text{threshold}$
  - (c) # of states  $> \text{threshold}$   
(get rid of smallest  $\alpha$ )

# Introduction to Parsing

# Context-free Grammars

- Chomsky hierarchy
  - Type 0 Grammars/Languages
    - rewrite rules  $\alpha \rightarrow \beta$ ;  $\alpha, \beta$  are any string of terminals and nonterminals
  - Context-sensitive Grammars/Languages
    - rewrite rules:  $\alpha X \beta \rightarrow \alpha \gamma \beta$ , where  $X$  is nonterminal,  $\alpha, \beta, \gamma$  any string of terminals and nonterminals ( $\gamma$  must not be empty)
  - Context-free Grammars/Languages
    - rewrite rules:  $X \rightarrow \gamma$ , where  $X$  is nonterminal,  $\gamma$  any string of terminals and nonterminals
  - Regular Grammars/Languages
    - rewrite rules:  $X \rightarrow \alpha Y$  where  $X, Y$  are nonterminals,  $\alpha$  string of terminal symbols;  $Y$  might be missing

# Parsing Regular Grammars

- Finite state automata
  - Grammar  $\leftrightarrow$  regular expression  $\leftrightarrow$  finite state automaton
- Space needed:
  - constant
- Time needed to parse:
  - linear ( $\sim$  length of input string)
- Cannot do e.g.  $a^n b^n$ , embedded recursion (context-free grammars can)

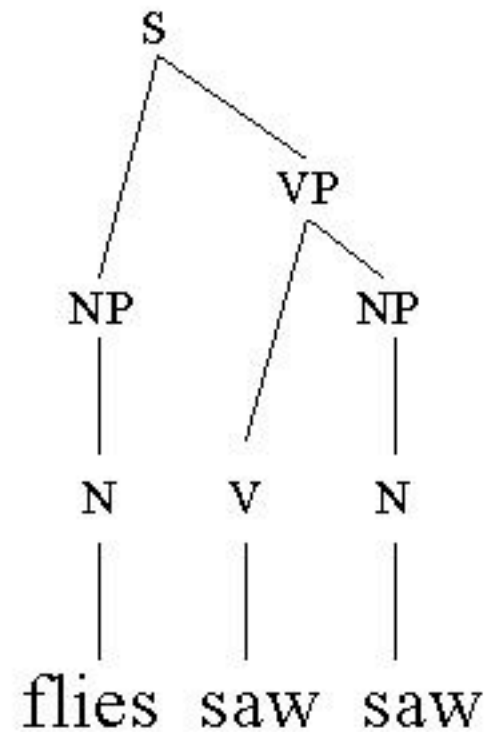
# Parsing Context Free Grammars

- Widely used for surface syntax description (or better to say, for correct word-order specification) of natural languages
- Space needed:
  - stack (sometimes stack of stacks)
    - in general: items  $\sim$  levels of actual (i.e. in data) recursions
- Time: in general,  $O(n^3)$
- Cannot do: e.g.  $a^n b^n c^n$  (Context-sensitive grammars can)



## Example Toy NL Grammar

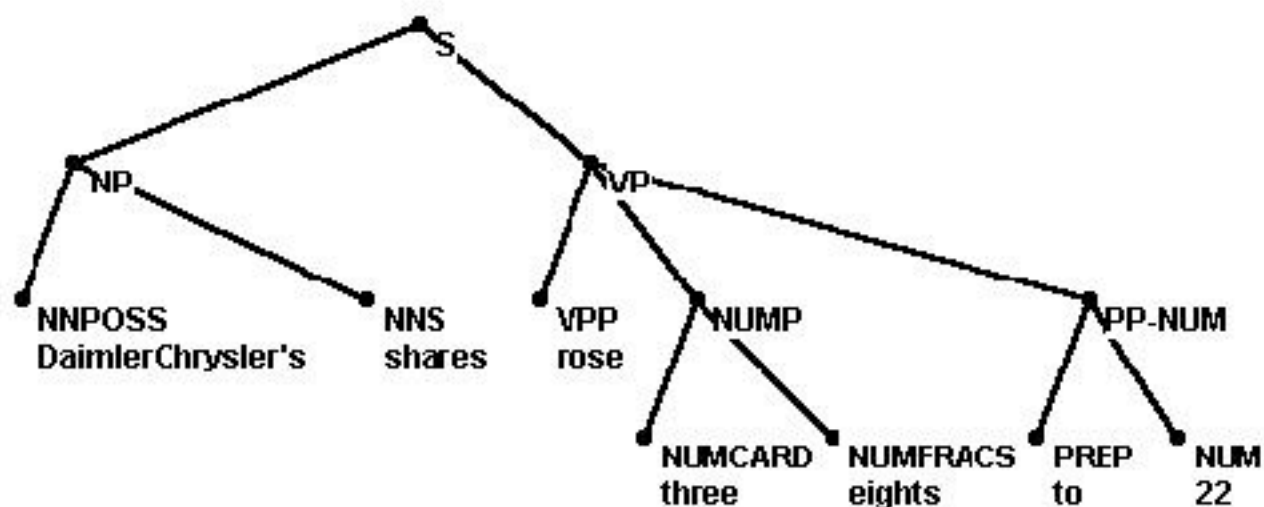
- #1  $S \rightarrow NP$
- #2  $S \rightarrow NP VP$
- #3  $VP \rightarrow V NP$
- #4  $NP \rightarrow N$
- #5  $N \rightarrow \text{flies}$
- #6  $N \rightarrow \text{saw}$
- #7  $V \rightarrow \text{flies}$
- #8  $V \rightarrow \text{saw}$



# Treebanks, Evaluation

# Phrase Structure Tree

- Example:



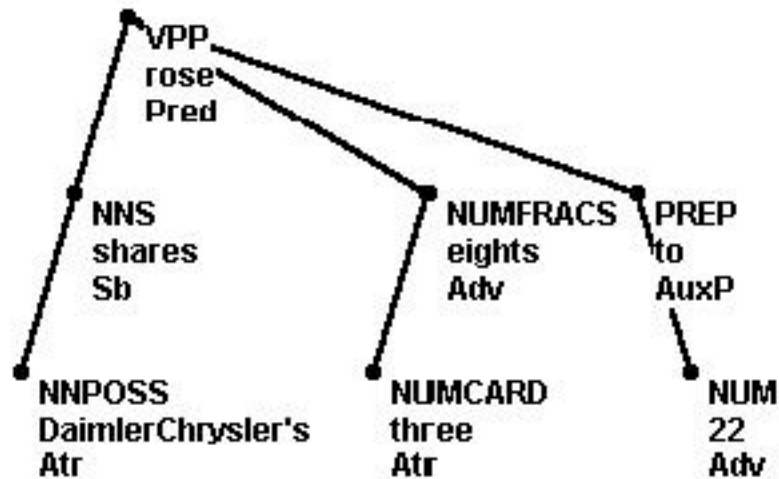
---

DaimlerChrysler's shares rose three eights to 22

$((\text{DaimlerChrysler's shares})_{\text{NP}} (\text{rose } (\text{three eights})_{\text{NUMP}} (\text{to } 22)_{\text{PP-NUM}})_{\text{VP}})_{\text{S}}$

# Dependency Tree

- Example:



DaimlerChrysler's shares rose three eights to 22

$rose_{Pred}(\text{shares}_{Sb}(\text{DaimlerChrysler's}_{Atr}), \text{eights}_{Adv}(\text{three}_{Atr}), \text{to}_{AuxP}(22_{Adv}))$

# Data Selection and Size

- Type of data
  - Task dependent (Newspaper, Journals, Novels, Technical Manuals, Dialogs, ...)
- Size
  - The more the better! (Resource-limited)
- Data structure:
  - Eventually; training + development test + eval test sets
    - more test sets needed for the long term (development, evaluation)
  - Multilevel annotation:
    - training level 1, test level 1; separate training level 2, test level 2, ...

# Parse Representation

- Core of the Treebank Design
- Parse representation
  - Dependency vs. Parse tree
    - Task-dependent
    - (1 : n) mapping from dependency to parse tree (in general)
  - Attributes
    - What to encode: words, morphological, syntactic, ... information
    - At tree nodes vs. arcs
      - e.g. Word, Lemma, POSTag, Function, Phrase-name, Dep-type, ...
    - Different for leaves? (Yes - parse trees, No - dependency trees)
  - Reference & Bookkeeping Attributes
    - bibliograph. ref., date, time, who did what

# Low-level Representation

- Linear representation:
  - SGML/XML (Standard Generalized Markup Language)  
[www.oasis-open.org/cover/sgml-xml.html](http://www.oasis-open.org/cover/sgml-xml.html)
  - TEI, TEILite, CES: Text Encoding Initiative  
[www.uic.edu/orgs/tei](http://www.uic.edu/orgs/tei)  
[www.lpl.univ-aix.fr/projects/multext/CES/CES1.html](http://www.lpl.univ-aix.fr/projects/multext/CES/CES1.html)
  - Extension / your own
    - Ex.: Workshop'98 (Dependency representation encoding):  
[www.clsp.jhu.edu/ws98/projects/nlp/doc/data/a0022.dtd](http://www.clsp.jhu.edu/ws98/projects/nlp/doc/data/a0022.dtd)

# Organization Issues

- The Team

Approx. need for  
1 mil. word size:

- Team leader; bookkeeping/hiring person 1
- Guidelines person(s) (editing) 1
- Linguistic issues person 1
- Annotators 3-5 (x2)<sup>\*</sup>
- Technical staff/programming 1-2
- Checking person(s) 2

- <sup>\*</sup>Double-annotation if possible



# Annotation

- Text vs. Graphics
  - text: easy to implement, directly stored in low-level format
    - e.g. use Emacs macros; Word macros; special SW
  - graphics: more intuitive (at least for linguists)
    - special tools needed
      - annotation
      - bookkeeping
      - “undo”
      - batch processing capability

# Treebanking Plan

- The main points (apart from securing financing...):
  - Planning
  - Basic Guidelines Development
  - Annotation & Guidelines Refinement
  - Consistency Checking, Guidelines Finalization
  - Packaging and Distribution (Data, Guidelines, Viewer)
- Time needed:
  - in the order of 2 years per 1 mil. words
  - only about 1/3 of the total effort is annotation

# Parser Development

- Use training data for learning phase
  - segment as needed (e.g., for heldout)
  - use all for
    - manually written rules (seldom today)
    - automatically learned rules/statistics
- Occasionally, test progress on Development Test Set
  - (simulates real-world data)
- When done, test on Evaluation Test Set
- *Unbreakable Rule #1: Never look at Evaluation Test Data (not even indirectly, e.g. performance numbers)*

# Evaluation

- Evaluation of parsers (regardless of whether manual-rule-based or automatically learned)
- Repeat: Test against Evaluation Test Data
- Measures:
  - Dependency trees:
    - Dependency Accuracy, Precision, Recall
  - Parse trees:
    - Crossing brackets
    - Labeled precision, recall [F-measure]

# Dependency Parser Evaluation

- Dependency Recall:
  - $R_D = \text{Correct}(D) / |S|$ 
    - **Correct(D): number of correct dependencies**
      - correct: word attached to its true head
      - Tree root is correct if marked as root
    - **|S| - size of test data in words (since |dependencies| = |words|)**
- Dependency precision (if output not a tree, partial):
  - $P_D = \text{Correct}(D) / \text{Generated}(D)$ 
    - **Generated(D) is the number of dependencies output**
      - some words without a link to their head
      - some words with several links to (several different) heads

# Phrase Structure (Parse Tree) Evaluation

- Crossing Brackets measure
  - Example “truth” (evaluation test set):
    - ((the ((New York) - based company)) (announced (yesterday)))
  - Parser output - 0 crossing brackets:
    - ((the New York - based company) (announced yesterday))
  - Parser output - 2 crossing brackets:
    - (((the New York) - based) (company (announced (yesterday))))
- Labeled Precision/Recall:
  - Usual computation using bracket labels (phrase markers)
    - T: ((Computers)<sub>NP</sub> (are down)<sub>VP</sub>)<sub>S</sub> ↔ P: ((Computers)<sub>NP</sub> (are (down)<sub>NP</sub>)<sub>VP</sub>)<sub>S</sub>
    - Recall = 100%, Precision = 75%

# PCFGs Introduction

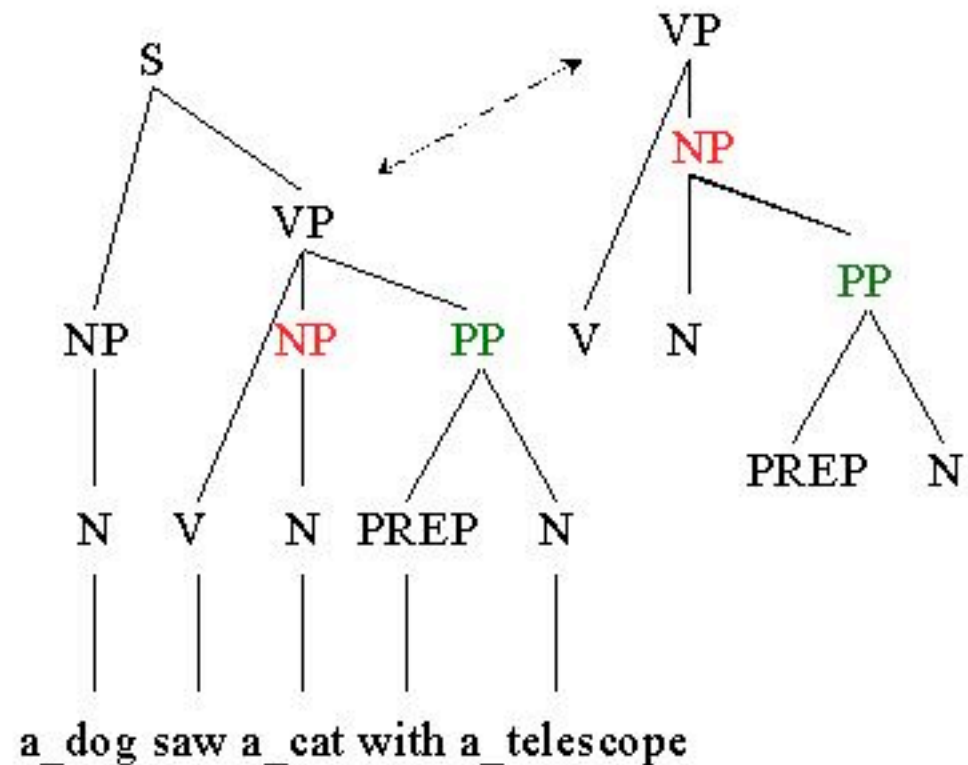
# Context-free Grammars

- Chomsky hierarchy
  - Type 0 Grammars/Languages
    - rewrite rules  $\alpha \rightarrow \beta$ ;  $\alpha, \beta$  are any string of terminals and nonterminals
  - Context-sensitive Grammars/Languages
    - rewrite rules:  $\alpha X \beta \rightarrow \alpha \gamma \beta$ , where  $X$  is nonterminal,  $\alpha, \beta, \gamma$  any string of terminals and nonterminals ( $\gamma$  must not be empty)
  - **Context-free Grammars/Languages**
    - rewrite rules:  $X \rightarrow \gamma$ , where  $X$  is nonterminal,  $\gamma$  any string of terminals and nonterminals
  - Regular Grammars/Languages
    - rewrite rules:  $X \rightarrow \alpha Y$  where  $X, Y$  are nonterminals,  $\alpha$  string of terminal symbols;  $Y$  might be missing



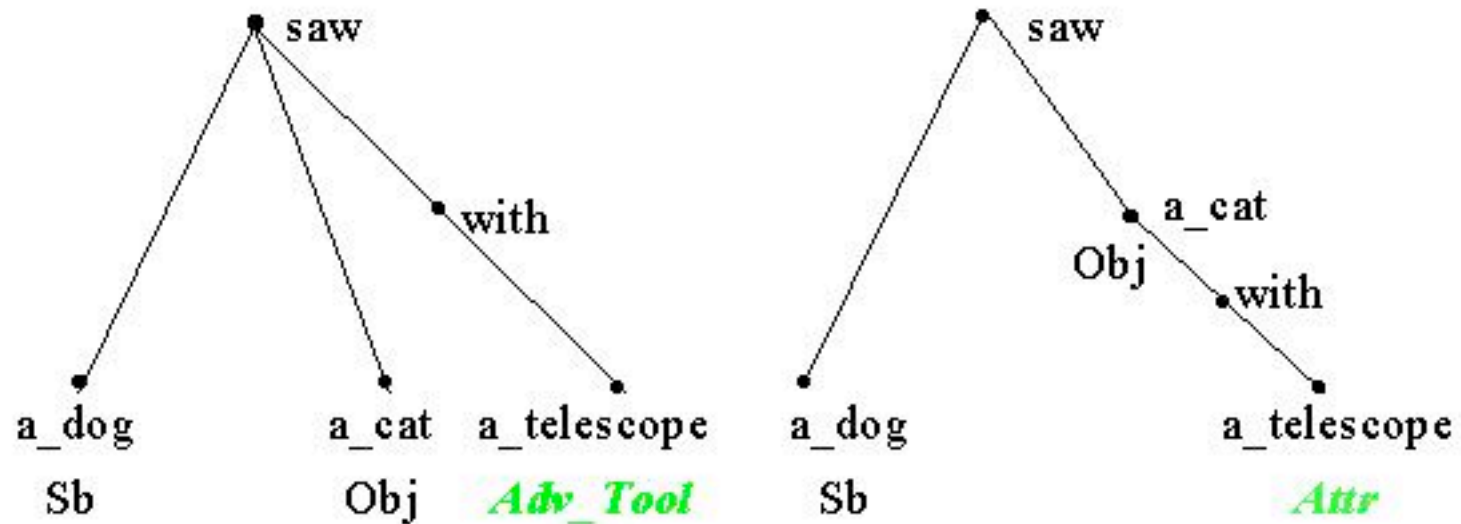
# Another NLP Example

- #1  $S \rightarrow NP VP$
- #2  $VP \rightarrow V NP PP$
- #3  $VP \rightarrow V NP$
- #4  $NP \rightarrow N$
- #5  $NP \rightarrow N PP$
- #6  $PP \rightarrow PREP N$
- #7  $N \rightarrow a\_dog$
- #8  $N \rightarrow a\_cat$
- #9  $N \rightarrow a\_telescope$
- #10  $V \rightarrow saw$
- #11  $PREP \rightarrow with$



# Dependency Style Example

- Same example, dependency representation



# Probability of a Derivation Tree

- Both phrase/parse/derivational “grammatical”
- Different meaning: which is better [in context]?
- “Internal context”: relations among phrases, words
- Probabilistic CFG:
  - relations among a mother node & daughter nodes
  - in terms of expansion [rewrite, derivation] probability
  - define probability of a derivation (i.e. parse) tree:

$$P(T) = \prod_{i=1..n} p(r(i))$$

$r(i)$  are all rules of the CFG used to generate the sentence  $W$  of which  $T$  is a parse

# Assumptions

- Independence assumptions (very strong!)
- Independence of context (neighboring subtrees)
- Independence of ancestors (upper levels)
- Place-independence (regardless where in tree it appears) ~ time invariance in HMM

## Probability of a Rule

- Rule  $r(i): A \rightarrow \alpha$ ;
- Let  $R_A$  be the set of all rules  $r(j)$ , which have nonterminal  $A$  at the left-hand side;
- Then define probability distribution on  $R_A$ :

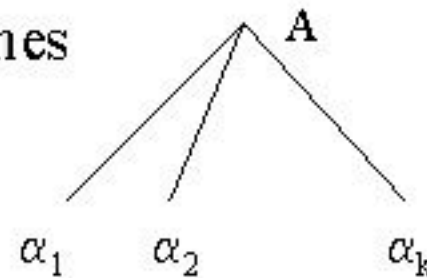
$$\sum_{r \in R_A} p(r) = 1, 0 \leq p(r) \leq 1$$

- Another point of view:

$$p(\alpha|A) = p(r), \text{ where } r = A \rightarrow \alpha, \alpha \in (N \cup T)^+$$

# Estimating Probability of a Rule

- MLE from a treebank *following a CFG grammar*
- Let's  $r = A \rightarrow \alpha_1 \alpha_2 \dots \alpha_k$  :
  - $p(r) = c(r) / c(A)$
  - Counting rules ( $c(r)$ ): how many times



appears in the treebank.

- Counting nonterminals  $c(A)$ :  
just count'em (in the treebank)

# Probability of a Derivation Tree

- Probabilistic CFG:
  - relations among a mother node & daughter nodes
  - in terms of expansion [rewrite, derivation] probability
  - define probability of a derivation (i.e. parse) tree:

$$P(T) = \prod_{i=1..n} p(r(i))$$

$r(i)$  are all rules of the CFG used to generate the sentence  $W$  of which  $T$  is a parse

- Probability of a string  $W = (w_1, w_2, \dots, w_n)$  ?
- Non-trivial, because there may be many trees  $T_j$  as a result of a parsing  $W$ .

# Probability of a String

- Input string:  $W$
- Parses:  $\{T_j\}_{j=1..n} = \text{Parse}(W)$ .

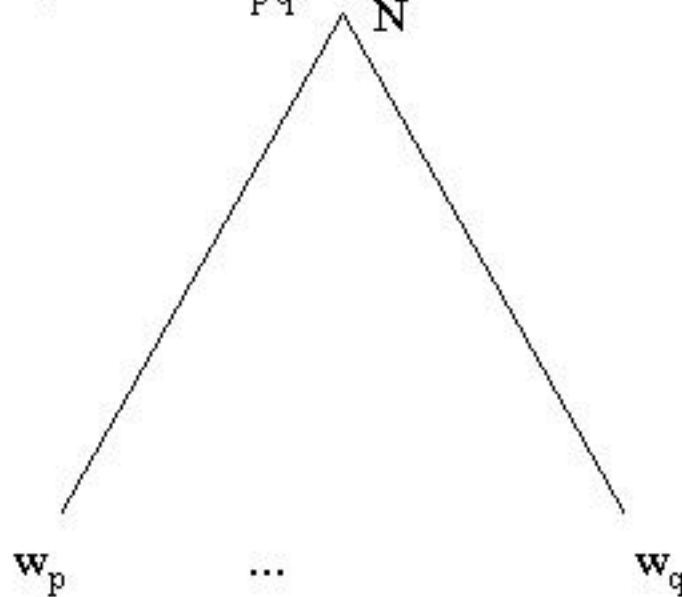
$$P(W) = \sum_{j=1..n} P(T_j) \quad !$$

- Impossible to use the naive method.



# Inside Probability

- $\beta_N(p,q) = P(N \Rightarrow *w_{pq})$



## Formula for Inside Probability

- $\beta_N(p, q) =$

$$\sum_{A, B} \sum_{d=p..q-1} P(N \rightarrow A, B) \beta_A(p, d) \beta_B(d+1, q)$$

assuming the grammar  $G$  has rules of the form

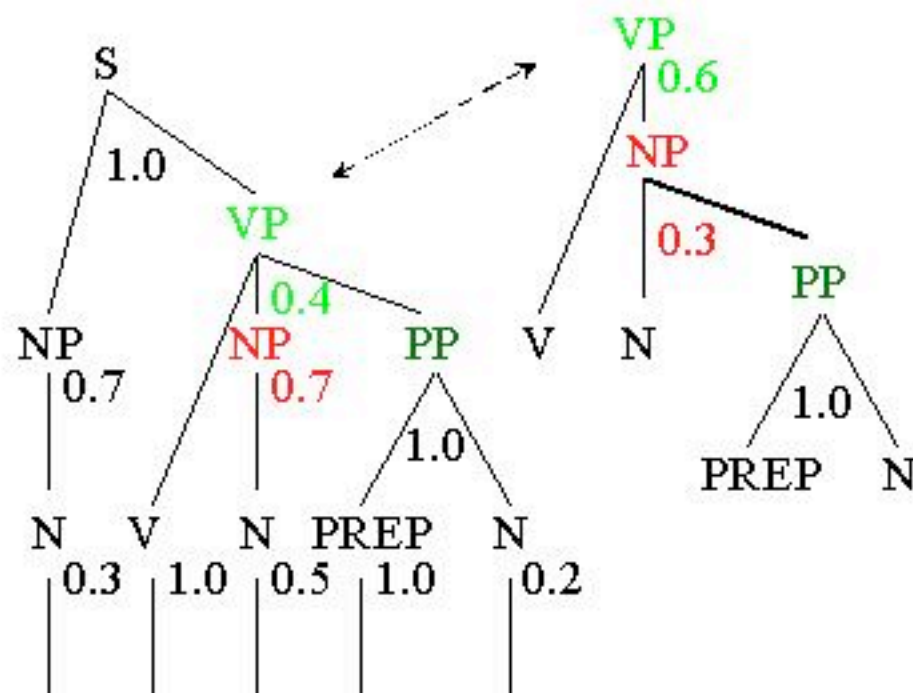
$N \rightarrow \omega$  (terminal string only)

$N \rightarrow A B$  (two nonterminals)

only (Chomsky Normal Form).

# Example PCFG

- #1 S → NP VP 1.0
- #2 VP → V NP PP 0.4
- #3 VP → V NP 0.6
- #4 NP → N 0.7
- #5 NP → N PP 0.3
- #6 PP → PREP N 1.0
- #7 N → a\_dog 0.3
- #8 N → a\_cat 0.5
- #9 N → a\_telescope 0.2
- #10 V → saw 1.0
- #11 PREP → with 1.0



$P(\text{a\_dog saw a\_cat with a\_telescope}) =$

$$1 \times .7 \times .4 \times .3 \times .7 \times 1 \times .5 \times 1 \times 1 \times .2 + \dots \times .6 \dots \times .3 \dots = .00588 + .00378 = .00966$$

# Computing String Probability

- a\_dog saw a\_cat with a\_telescope

1      2      3      4      5

from\to	1	2	3	4	5
1	NP .21 N .3		S .441		S .00966
2		V 1	VP .21		VP .046
3			NP .35 N .5		NP .03
4				PREP 1	PP .2
5					N .2

- Create table  $n \times n$  ( $n = \text{length of string}$ ). Cells might have more “lines”.
- Initialize on diagonal, using  $N \rightarrow \alpha$  rules.
- Recursively compute along the diagonal towards the upper right corner.

# Statistical Parsing

# Language Model vs. Parsing Model

- Language model:

- interested in string probability:

$P(W)$  = probability definition using a formula such as

$$= \prod_{i=1..n} p(w_i | w_{i-2}, w_{i-1}) \quad \text{trigram language model}$$

$$= \sum_{s \in S} p(W, s) = \sum_{s \in S} \prod_{r \in s} r \quad \text{PCFG; } r \sim \text{rule used in parse tree}$$

- Parsing model

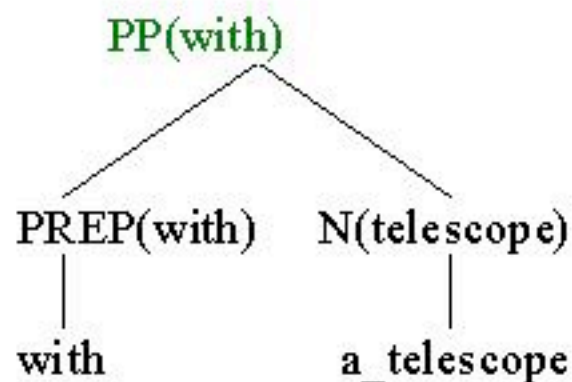
- conditional probability of tree given string:

$$P(s|W) = P(W, s) / P(W) = P(s) / P(W) \quad !! P(W, s) = P(s) !!$$

- for argmax, just use  $P(s)$  ( $P(W)$  is constant)

## Once again, Lexicalization

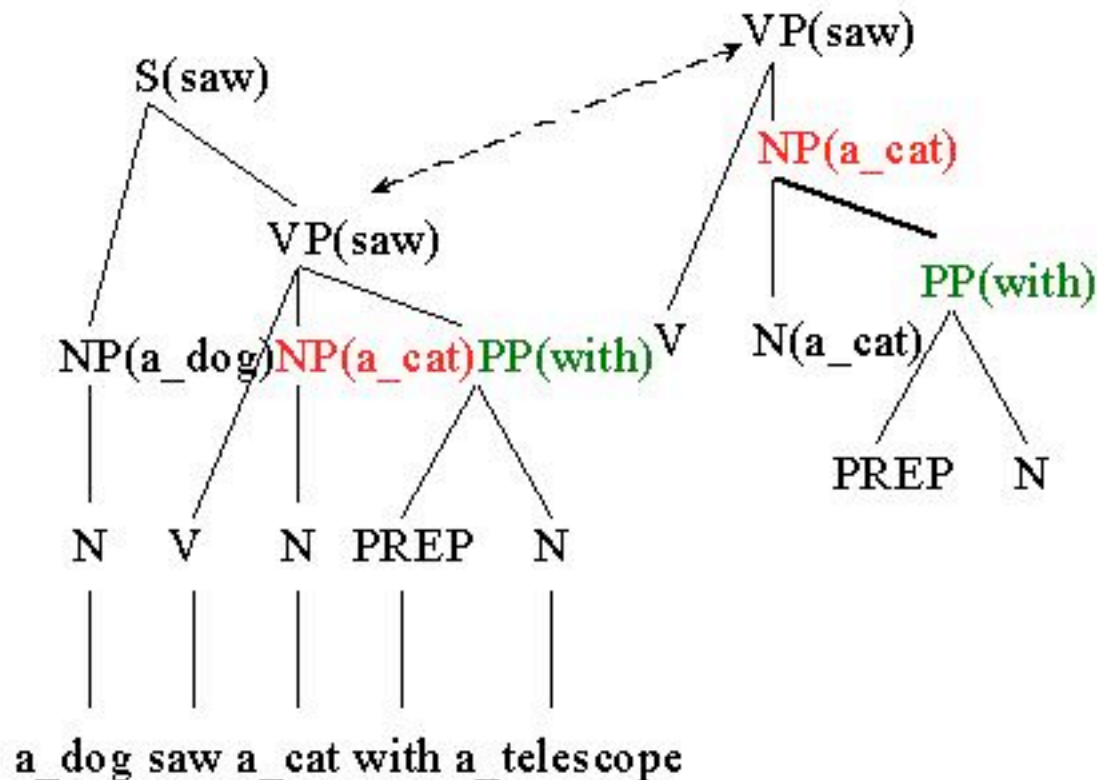
- Lexicalized parse tree (~ dependency tree+phrase labels)
- Ex. subtree:



- Pre-terminals (above leaves): assign the word below
- Recursive step (step up one level): (a) select node, (b) copy word up.

# Lexicalized Tree Example

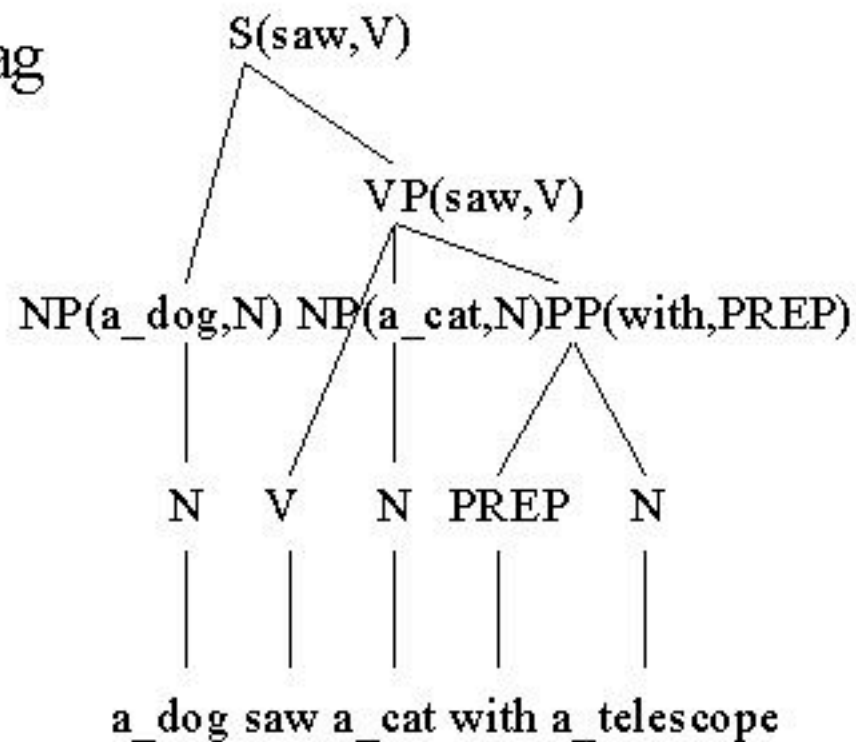
- #1 S → NP VP
- #2 VP → V NP PP
- #3 VP → V NP
- #4 NP → N
- #5 NP → N PP
- #6 PP → PREP N
- #7 N → a\_dog
- #8 N → a\_cat
- #9 N → a\_telescope
- #10 V → saw
- #11 PREP → with





# Using POS Tags

- Head ~ word,tag



# Conditioning

- Original PCFG:  $P(\alpha B \gamma D \varepsilon \dots | A)$ 
  - No “lexical” units (words)
- Introducing words:

$$P(\alpha B(\text{head}_B) \gamma D(\text{head}_D) \varepsilon \dots | A(\text{head}_A))$$

where  $\text{head}_A$  is one of the heads on the left

E.g. rule  $VP(\text{saw}) \rightarrow V(\text{saw}) NP(\text{a\_cat})$ :

$$P(V(\text{saw}) NP(\text{a\_cat}) | VP(\text{saw}))$$

# Independence Assumptions

- Too many rules
- Decompose:  
$$P(\alpha B(\text{head}_B) \gamma D(\text{head}_D) \varepsilon \dots | A(\text{head}_A)) =$$
- In general (total independence):  
$$P(\alpha | A(\text{head}_A)) \times P(B(\text{head}_B) | A(\text{head}_A)) \times$$
  
$$\dots \times P(\varepsilon | A(\text{head}_A))$$
- Too much independent: need a compromise.

# The Decomposition

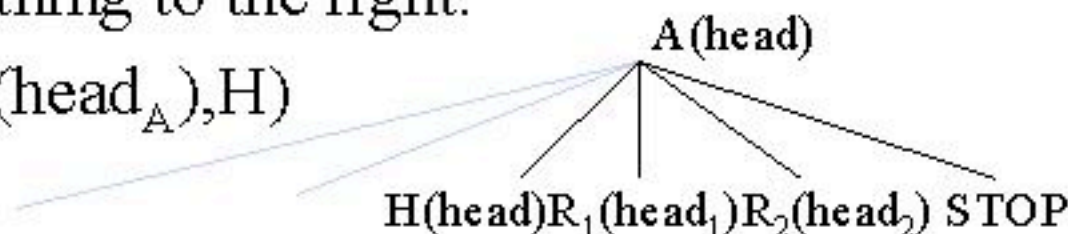
- Order does not matter, let's use intuition (“linguistics”):
- Select the head daughter category:

$$P_H(H(\text{head}_A) | A(\text{head}_A))$$



- Select everything to the right:

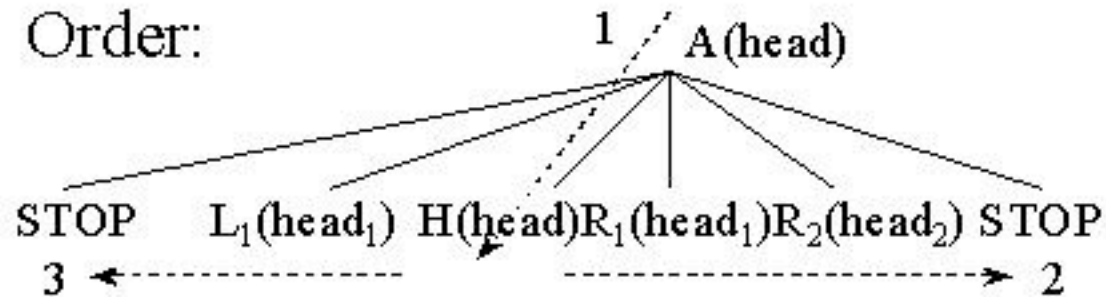
$$P_R(R_i(r_i) | A(\text{head}_A), H)$$



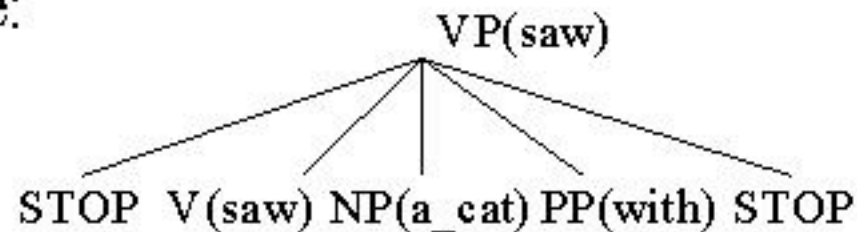
- Also, choose when to finish:  $R_{m+1}(r_{m+1}) = STOP$
- Similarly, for the left direction:  $P_L(L_i(l_i) | A(\text{head}_A), H)$

# Example Decomposition

- Order:



- Example:

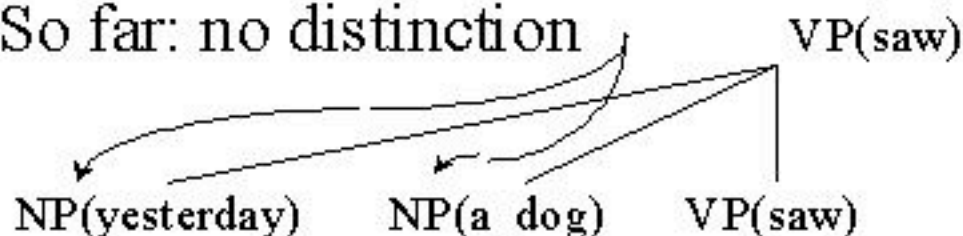


## More Conditioning: Distance

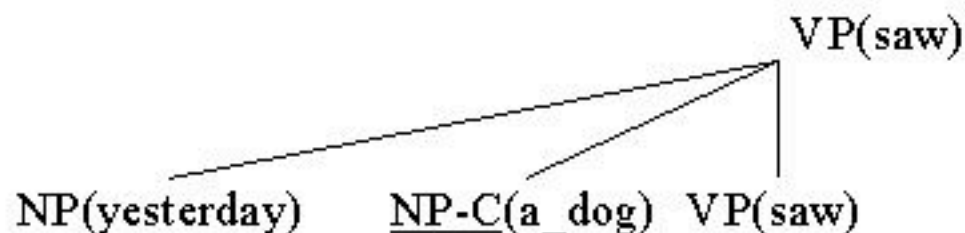
- Motivation:
  - close words tend to be dependents (or phrases) more likely
  - ex.: walking on a sidewalk on a sunny day without looking on...
- Number of words too detailed, though:
  - use more sophisticated (yet more robust) distance measure  $d_{r/l}$ :
    - distinguish 0 and non-zero distance (2)
    - distinguish if verb is in-between the head and the constituent in question (2)
    - distinguish if there are commas in-between: 0, 1, 2, >2 commas (4).
    - ...total: 16 possibilities added to the condition:  $P_R(R_i(r_j) | A(\text{head}_A), H, d_r)$
    - same to the left:  $P_L(L_i(l_j) | A(\text{head}_A), H, d_l)$

## More Conditioning: Complement/Adjunct

- So far: no distinction



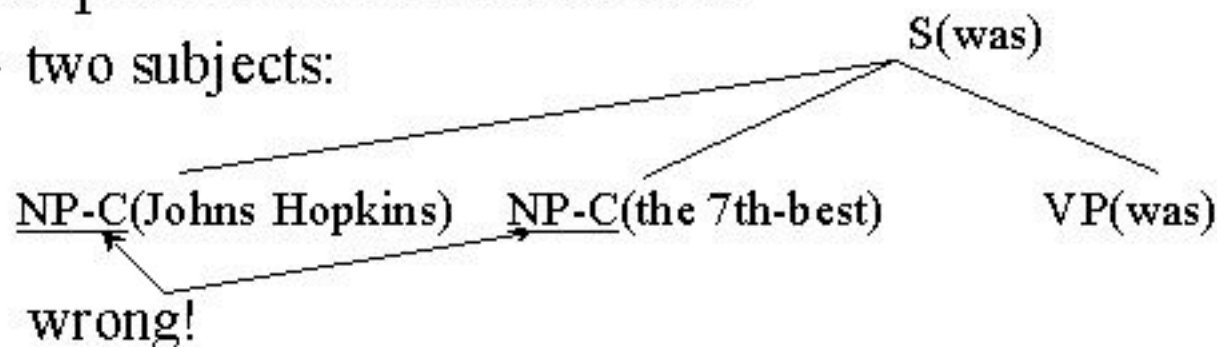
- ...but: time NP  $\neq$  subject NP
- also, Subject NP cannot repeat... useful during parsing



[Must be added in training data]

## More Conditioning: Subcategorization

- The problem still not solved:
  - two subjects:



- Need: relation among complements.
  - [linguistic observation: adjuncts can repeat freely.]
- Introduce:
  - Left & Right Subcategorization Frames (multisets)



## Inserting Subcategorization

- Use head probability as before:

$$P_H(H(\text{head}_A) | A(\text{head}_A))$$

- Then, add left & right subcat frame:

$$P_{lc}(LC | A(\text{head}_A), H), P_{rc}(RC | A(\text{head}_A), H)$$

– LC, RC: list (multiset) of phrase labels (not words)

- Add them to context condition:

$$\text{(left) } P_L(L_i(l_i) | A(\text{head}_A), H, d_i, LC) \quad \text{[right: similar]}$$

- LC/RC: “dynamic”: remove labels when generated
  - $P(\text{STOP} | \dots, LC) = 0$  if LC non-empty

# Smoothing

- Adding conditions... ~ adding parameters
- Sparse data problem as usual (head ~ <word,tag>!)
- Smooth (step-wise):
  - $P_{\text{smooth-H}}(\text{H}(\text{head}_A)|\text{A}(\text{head}_A)) = \lambda_1 P_{\text{H}}(\text{H}(\text{head}_A)|\text{A}(\text{head}_A)) + (1-\lambda_1) P_{\text{smooth-H}}(\text{H}(\text{head}_A)|\text{A}(\text{tag}_A))$
  - $P_{\text{smooth-H}}(\text{H}(\text{head}_A)|\text{A}(\text{tag}_A)) = \lambda_2 P_{\text{H}}(\text{H}(\text{head}_A)|\text{A}(\text{tag}_A)) + (1-\lambda_2) P_{\text{H}}(\text{H}(\text{head}_A)|\text{A})$
- Similarly, for  $P_{\text{R}}$  and  $P_{\text{L}}$ .

# The Parsing Algorithm for a Lexicalized PCFG

- Bottom-up Chart parsing
  - Elements of a chart: a pair
    - $\langle (\text{from-position}, \text{to-position}, \text{label}, \text{head}, \text{distance}), \text{probability} \rangle$ 
      - span  $\uparrow$  score  $\uparrow$
  - Total probability = multiplying elementary probabilities  
 $\Rightarrow$  enables dynamic programming:
    - discard chart element with the same span but lower score.
- “Score” computation:
  - joining chart elements: [for 2]:  $\langle e_1, p_1 \rangle, \langle e_2, p_2 \rangle, \dots, \langle e_n, p_n \rangle$ :

$$P(\mathbf{e}_{\text{new}}) = p_1 \times p_2 \times \dots \times p_n \times P_H(\dots) \times \prod P_R(\dots) \times \prod P_L(\dots);$$

# Results

- English, WSJ, Penn Treebank, 40k sentences

	< 40Words	< 100 Words
– Labeled Recall:	88.1%	87.5%
– Labeled Precision:	88.6%	88.1%
– Crossing Brackets (avg):	0.91	1.07
– Sentences With 0 CBs:	66.4%	63.9%
- Czech, Prague Dependency Treebank, 13k sentences:
  - Dependency Accuracy overall: 80.0%
  - (~ unlabelled precision/recall)

