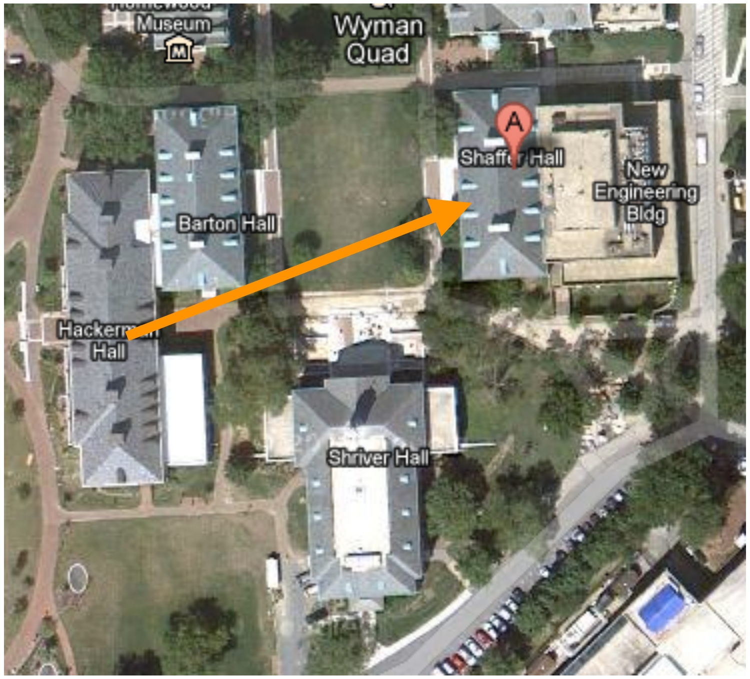
Syntax-based decoding

Tuesday, March 27, 2012

Administrative issues

- Class moving to Shaffer 202 starting this Thursday



- Assignment 2 due a week from today
  - Extra TA office hours on Monday
  - Leaderboards...
- Get your revised proposals in!

Review (1)

- We've discussed how syntactic differences between languages motivated reordering as a preprocessing step

Ich werde Ihnen den Report aushaendigen, damit Sie den eventuell uebernehmen koennen.

↓

Ich werde aushaendigen Ihnen den Report, damit Sie koennen uebernehmen den eventuell.

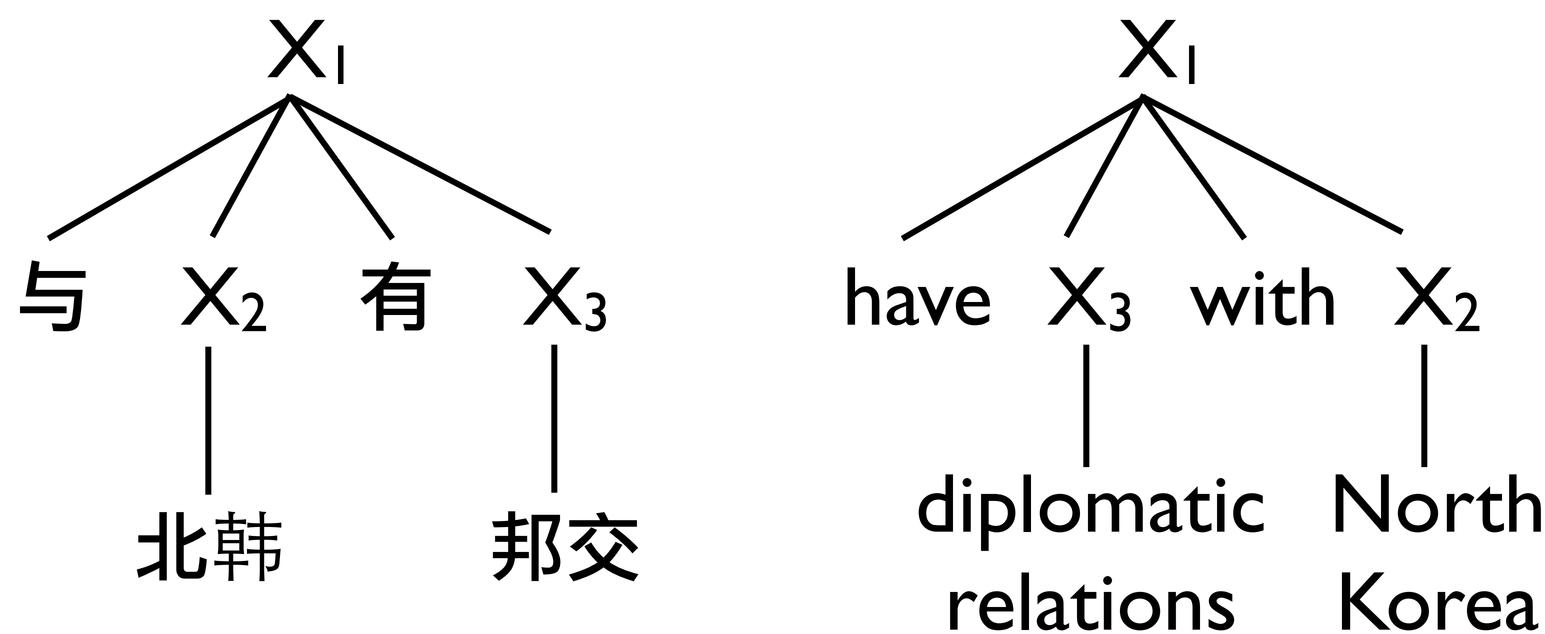
Review (2)

- We've also discussed synchronous grammar rules, which describe the generation of sentences in pairs

	Urdu	English
S →	NP① VP②	NP① VP②
VP →	PP① VP②	VP② PP①
VP →	V① AUX②	AUX② V①
PP →	NP① P②	P② NP①
NP →	*hamd ansary*	*Hamid Ansari*
NP →	*na}b sdr*	*Vice President*
V →	*namzd*	*nominated*
P →	*kylye*	*for*
AUX →	*taa*	*was*

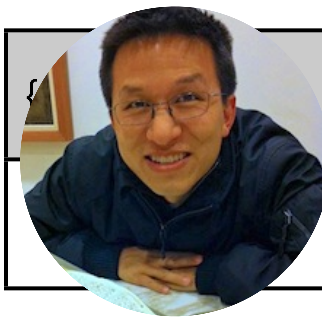
Review (3)

- ...and how we could extract those rules automatically from text



Today

- How do we actually *decode* with these grammars?
- The solution is the CKY / CYK algorithm



{	{CYK algorithm}
	~ 13,700

Google

- Outline
  - Parsing in one language
  - Parsing in two languages *with* ***inversion transduction grammar (ITG)***
  - Decoding as parsing *with* ***synchronous context-free grammars (SCFG)*** *and integrated language models*
  - Time-permitting: advanced topics

Review: monolingual parsing

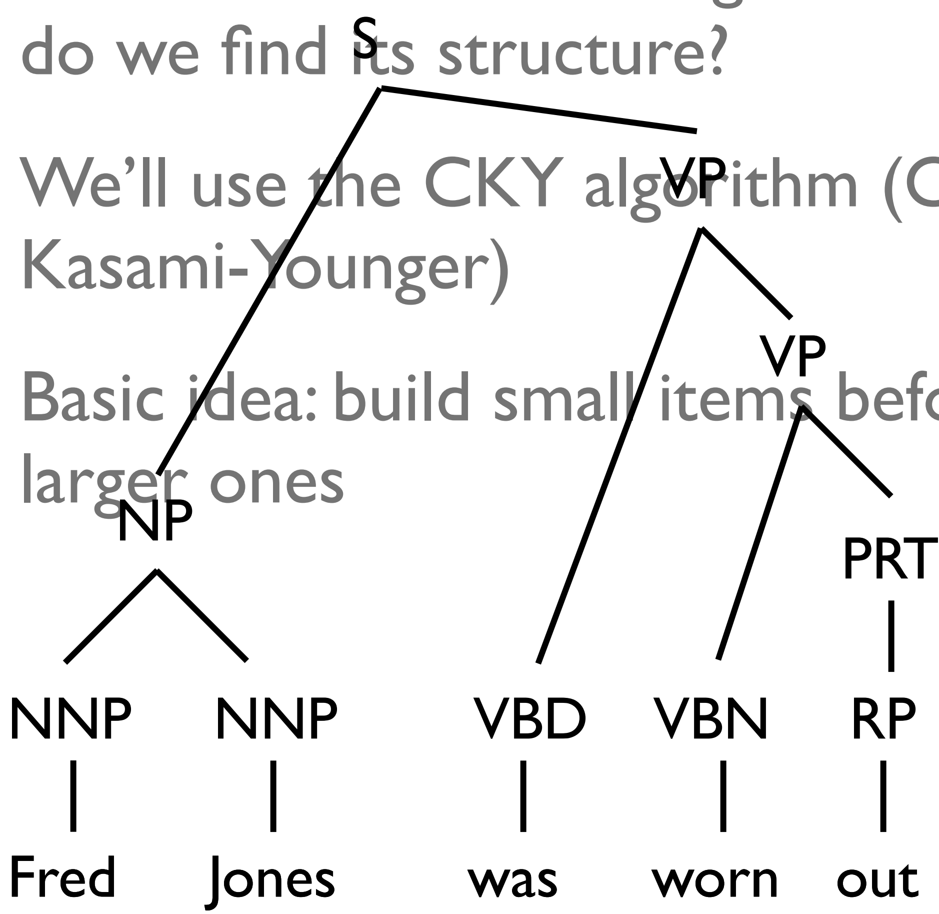
Using the CKY algorithm to find (the best) structure for a sentence given a grammar

Formal definitions

- **Formal languages** are (possibly infinite) sets of strings that are generated by a grammar
    - e.g., {a+} is a language of all strings with one or more *a*s
    - Its grammar could be written as
      A → Aa
      A → a
- We can view **natural languages** in this manner, too
    - e.g., the **English language** is the set of word sequences that constitute valid English sentences
    - We believe there to be a grammar that generates those sentences
    - We don't know what it is, but we have some guesses and approximations

Parsing

- Given a sentence and a grammar, how do we find its structure?
- We'll use the CKY algorithm (Cocke-Kasami-Younger)
- Basic idea: build small items before larger ones

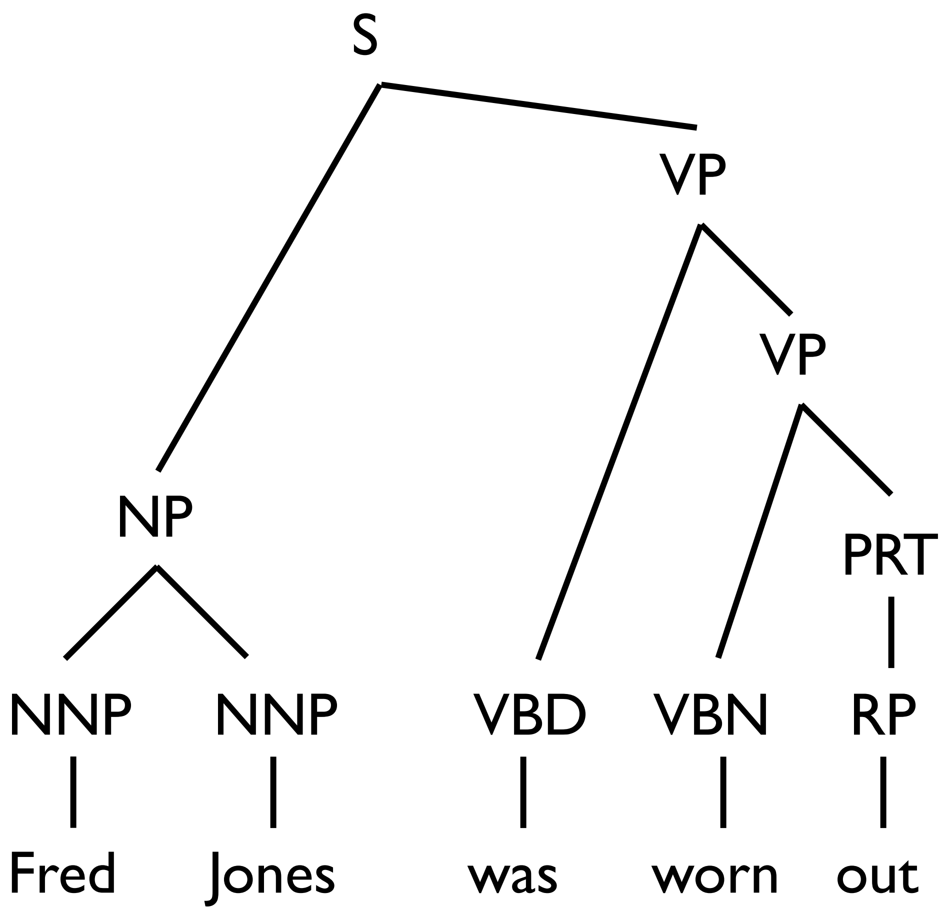


sentence

S	→	NP VP
VP	→	VBN PRT
PRT	→	RP
VP	→	VBD VP
NP	→	NNP NNP
NNP	→	Fred | Jones
VBD	→	was
VBN	→	worn
RP	→	out

grammar

Parsing with CKY



sentence

S	→	NP VP
VP	→	VBN PRT
PRT	→	RP
VP	→	VBD VP
NP	→	NNP NNP
NNP	→	Fred | Jones
VBD	→	was
VBN	→	worn
RP	→	out

grammar

Implementation details

- Dynamic programming maintains a chart of items
  - Each cell item represents the dynamic programming state
    - (NNP,1,1), (S,1,5)
  - The chart is the collection of all items

```
struct item {
  // d.p. state
  string nt;
  int i, j;
  // backpointer
  float score;
  Rule* rule;
  item* rhs1,
        rhs2;
}
```

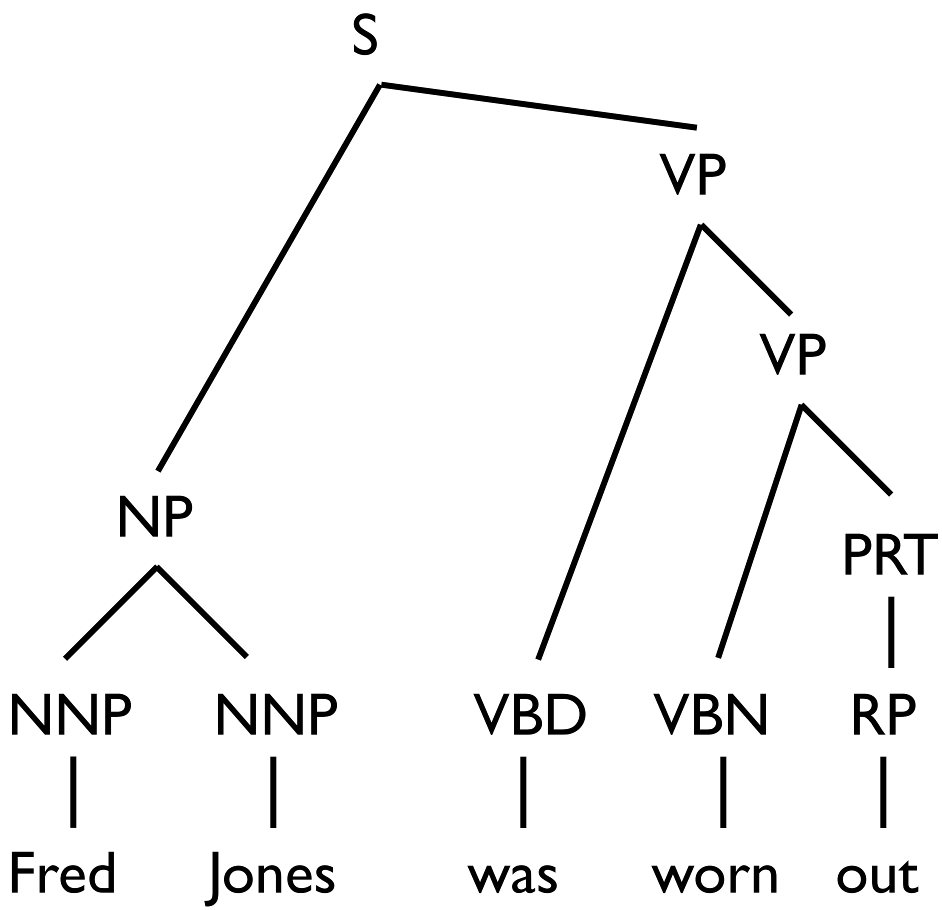
- The score resolves alternate ways of constructing an item
- We also store backpointers: the items and rule used to construct each item

a.k.a. "predecessor"

CKY algorithm

```
input: words[1..N]
for i in 1..N
  for each unary rule X → words[i]
    add (X,i,i) to the chart
for span in 1..N
  for i in 1..(N-span)
    j = i + span
    for k in i..j
      for rule X → Y Z
        if (Y,i,k) and (Z,k,j)
          add (X,i,j) to the chart
output: (S,1,N)
```

Parsing with CKY



	1	2	3	4	5	
Fred	NNP					1
Jones	NP	NNP				2
was			VBD			3
worn				VBN		4
out	S		VP	VP	RP PRT	5
	Fred	Jones	was	worn	out	

```
item
  nt = "S";
  i = 1, j = 5;
  score = -42.5;
  Rule = &rule("S → NP VP")
  rhs1 = &item(NP,1,2);
  rhs2 = &item(VP,3,5);
```

Reconstructing the best parse

- We can reconstruct the best parse by following backpointers

```
nodes.append(item(S,1,N))
while nodes.size() > 0:
  item = nodes.pop()
  print item
  nodes.append(item.rhsr)
  nodes.append(item.rhsl)
```

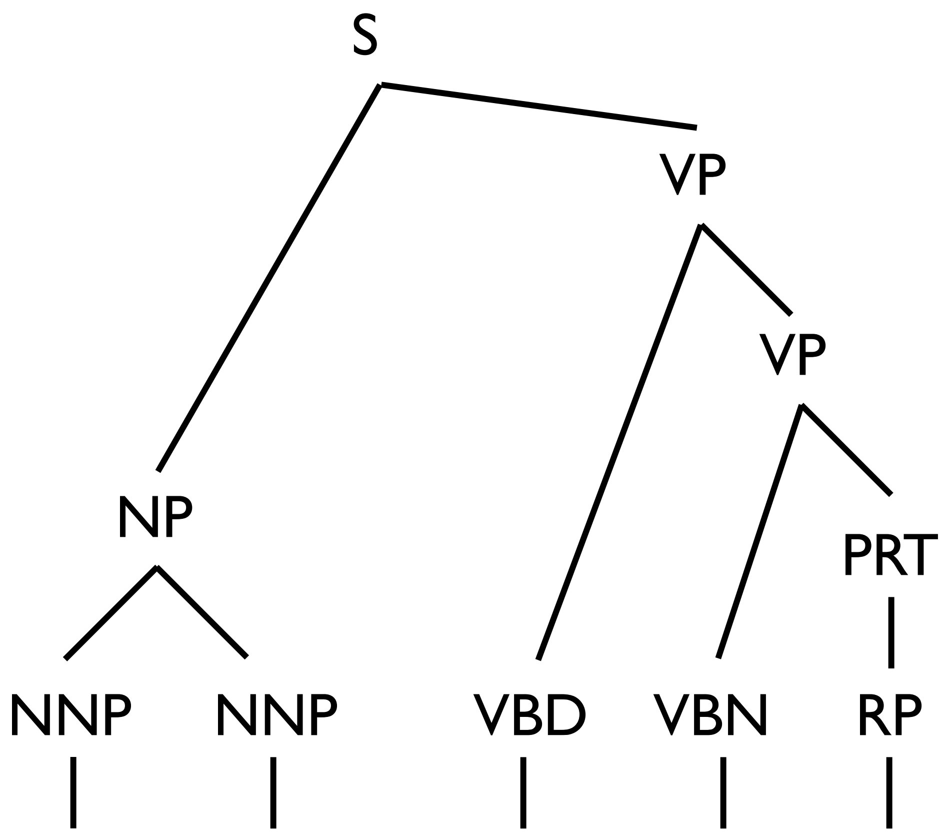
nodes

(NNP,1,1)

Fred	NNP				
Jones	NP	NNP			
was			VBD		
worn				VBN	
out	S		VP	VP	RP PRT
	Fred	Jones	was	worn	out

- S → NP VP (1,5)
  - NP → NNP NNP (1,2)
    - NNP → Fred (1,1)
    - NNP → Jones (2,2)
  - VP → VBD VP (3,5)
    - VBD → was (3,3)
    - VP → VBN PRT (4,5)
      - VBN → worn (4,4)
      - PRT → RP (5,5)
        - RP → out (5,5)

Parsing with CKY

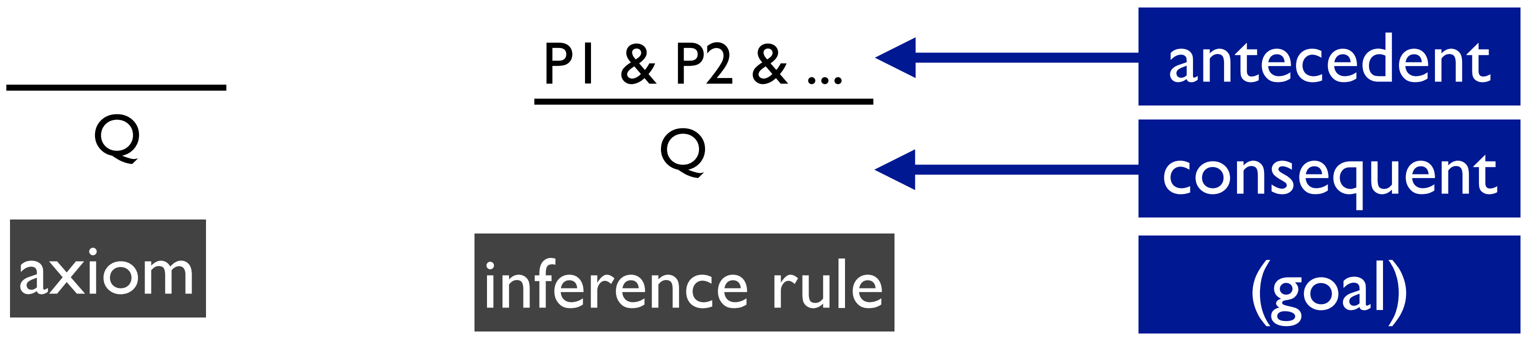


	1	2	3	4	5	
Fred	NNP					1
Jones	NP	NNP				2
was			VBD			3
worn				VBN		4
out	S		VP	VP	RP PRT	5
	Fred	Jones	was	worn	out	

Fred Jones was worn out *from caring for his often screaming and crying wife during the day but he couldn't sleep at night for she in a stupor from the drugs that didn't ease the pain would set the house ablaze with a cigarette*

Parsing as (weighted) deduction

- Deductive reasoning:
  - axioms: statements that are true or false ("it is raining")
  - inference rules: statements that are conditionally true ("If it is raining and I am outside, I'll get wet")
  - goals: statements that are licensed by combinations of axioms, inference rules, and other conclusions ("I am wet")



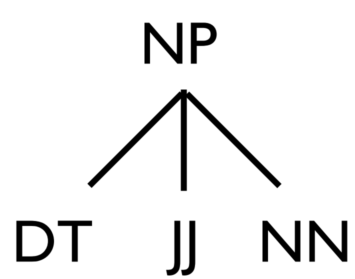
Parsing as (weighted) deduction

- input: words w[1..N]

Axioms	$\dfrac{}{X \to w[i]}$	for all (X → w[i])
Inference rules	$\dfrac{X \to w[i]}{(X, i, i)}$ $\dfrac{(B, i, j) \quad (C, j, k) \quad A \to BC}{(A, i, k)}$	in bottom-up order (smaller spans first)
Goal	(S, 1, n)	

Complexity

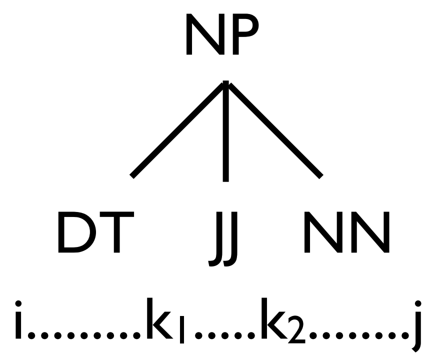
- Complexity of parsing is $O(Gn^3)$
  - G - number of (binarized) rules in the grammar
  - n - length of the sentence
- All those rules were binary; what about longer rules?
  - e.g.,



  - We have to enumerate every split point!

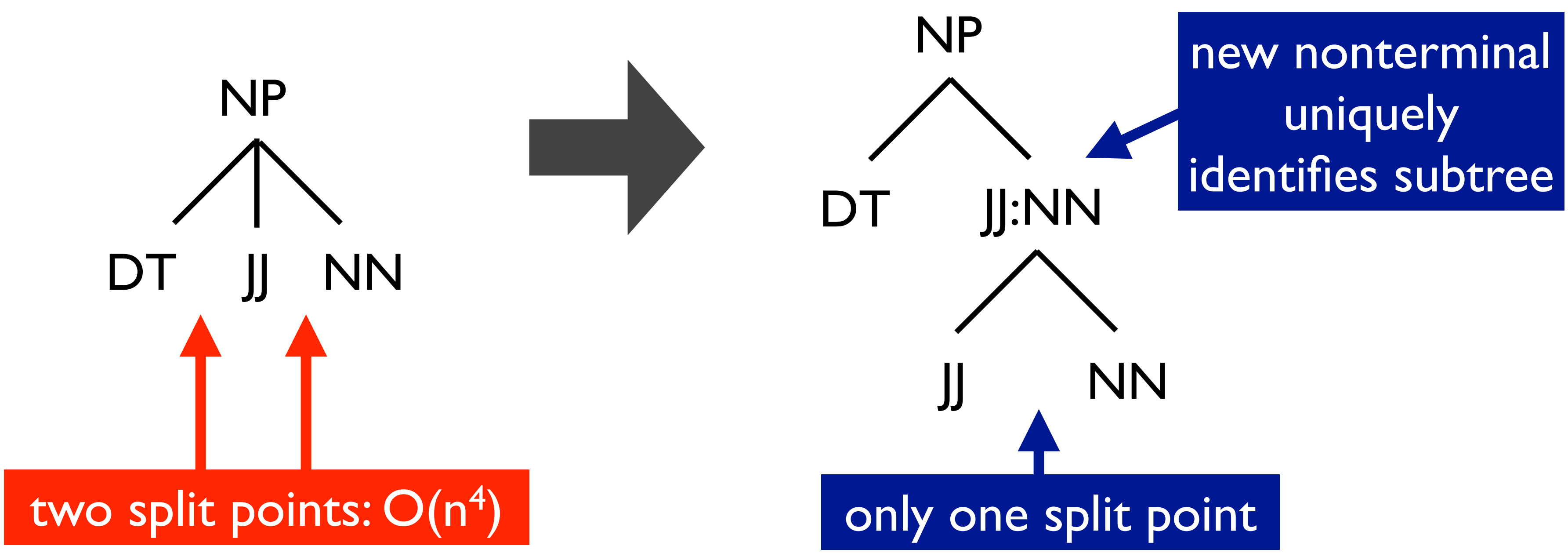
CKY algorithm

```
input: words[1..N]
for i in 1..N
  for each unary rule X → words[i]
    add (X,i,i) to the chart
for span in 1..N
  for i in 1..(N-span)
    j = i + span
    for k1 in i..j-1
      for k2 in k1..j
        for rule X → W Y Z
          if (W,i,k1) and (Y,k1,k2) and (Z,k2,j)
            add (X,i,j) to the chart
output: (S,1,N)
```



Binarization into Chomsky Normal Form

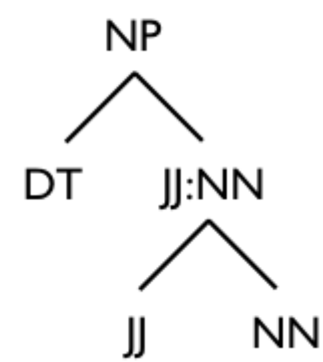
- In general, for a rule with k RHS items, complexity is $O(n^{k+1})$ (and cumbersome, since you have to explicitly add inner loops to enumerate them)
- Fortunately, we can **binarize** rules to make them all have a rank of 2



CKY algorithm

- In summary, monolingual parsing:
  - finds the best structure
  - works bottom-up, enumerating all spans, from small to large, building searching for applicable rules and building new chart items
  - works with the binarized form of a grammars (easily unbinarized afterward) for a complexity of O(Gn^3)
  - all grammars are binarizable

	1	2	3	4	5	
Fred	NNP					1
Jones	NP	NNP				2
was			VBD			3
worn				VBN		4
out	S		VP	VP	RP	5
	Fred	Jones	was	worn	out	



Synchronous parsing

Synchronous parsing

- We can extend CKY to parse two languages at once!
- Consider the following grammar:

A → fat, guapos	(lexical)
A → thin, delgados	
N → cats, gatos	
VP → eat, comen	
NP → $A^{(1)}$ $N^{(2)}$, $N^{(2)}$ $A^{(1)}$	(inverted)
S → $NP^{(1)}$ $VP^{(2)}$, $NP^{(1)}$ $VP^{(2)}$	(straight)

- and the following sentence pair:

fat cats eat / gatos guapos comen

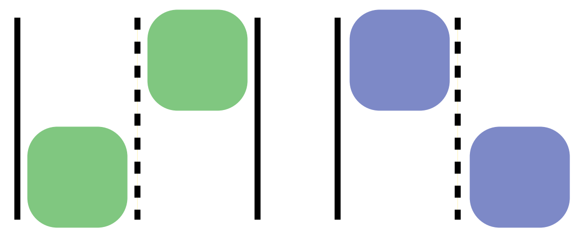
Synchronous parsing

- We now have to enumerate *pairs* of spans
  - instead of (i,j)...
  - ...we have (i,j) and (s,t)
- For each of the bilingual blocks, we attempt to match both straight and inverted rules

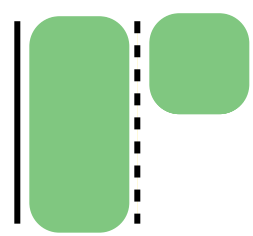
$A \rightarrow$ fat, guapos
$N \rightarrow$ cats, gatos
$VP \rightarrow$ eat, comen
$VP \rightarrow$ eat, como
$NP \rightarrow A^{(1)}\ N^{(2)},\ N^{(2)}\ A^{(1)}$
$S \rightarrow NP^{(1)}\ VP^{(2)},\ NP^{(1)}\ VP^{(2)}$

Relation to monolingual parsing

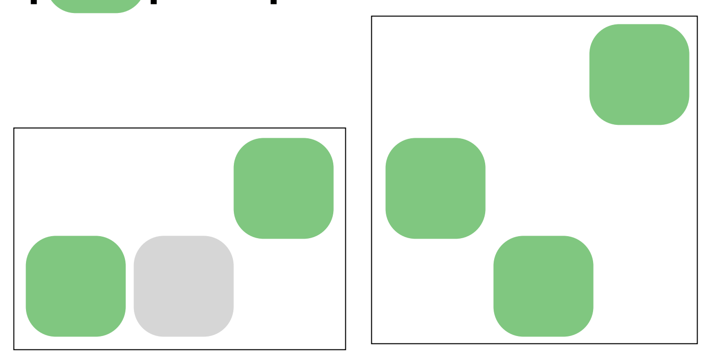
- Why do we combine like this?
  - Think about monolingual CKY: combine adjacent spans
  - These pieces are adjacent in both languages; it's only when we consider them *together* that reordering comes into play



- Why can't we do this?
  - It doesn't make sense!



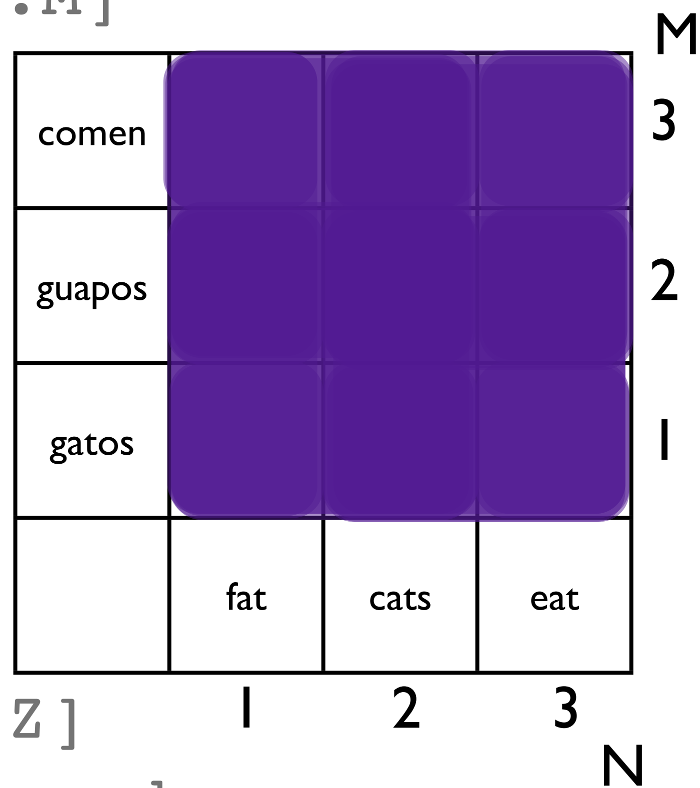
- What about these?
  - Possible, but complex



gap   rank > 2

CKY for synchronous parsing

```
input: source[1..N], target[1..M]
for span₁ in 1..N
  for i in 1..(N-span₁)
    j = i + span₁
      for k in i..j
        for span₂ in 1..M
          for s in 1..(M-span₂)
            t = s + span₂
              for u in s..t
                for rule X → [Y Z]
                  if (Y,i,k,s,u) and
                     (Z,k,j,u,v) then
                    add (X,i,j,s,t) to chart
output: (S,1,N,1,M)
```



Synchronous parsing

- Complexity: $O(GN^3M^3) \approx O(GN^6)$
- Why?
  - We have to enumerate all valid combinations of six variables
  - This can be seen in the six nested loops of the algorithm

A → fat, guapos
N → cats, gatos
VP → eat, comen
VP → eat, como
NP → $A^{(1)}\ N^{(2)},\ N^{(2)}\ A^{(1)}$
S → $NP^{(1)}\ VP^{(2)},\ NP^{(1)}\ VP^{(2)}$

Visualization of $O(GN^6)$ complexity

```
  input: source[1..N], target[1..M]
1 for span1 in 1..N
2   for i in 1..(N-span1)
      j = i + span1
3       for k in i..j
4         for span2 in 1..M
5           for s in 1..(M-span2)
              t = s + span2
6               for u in s..t
times all rules...  for rule X → [Y Z]
                      if (Y,i,k,s,u) and
                         (Z,k,j,u,v) then
                           add (X,i,j,s,t) to chart
  output: (S,1,N,1,M)
```

Synchronous binarization

- In the above, we considered two nonterminals (per side)
- What if we want more (Zhang et al., 2006)?

$$
\begin{aligned}
\mathrm{S} &\rightarrow \mathrm{NP}^{(1)}\ \mathrm{VP}^{(2)}\ \mathrm{PP}^{(3)},\ \mathrm{NP}^{(1)}\ \mathrm{PP}^{(3)}\ \mathrm{VP}^{(2)} \\
\mathrm{NP} &\rightarrow \text{Powell}, \textit{Baoweier} \\
\mathrm{VP} &\rightarrow \text{held a meeting}, \textit{juxing le huitan} \\
\mathrm{PP} &\rightarrow \text{with Sharon}, \textit{yu Shalong}
\end{aligned}
$$

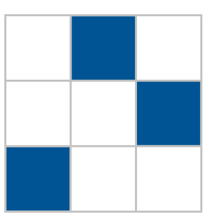
- Three nonterminals? No problem:

$$
\begin{aligned}
\mathrm{S} &\rightarrow V_{\text{NP-PP}}\ \mathrm{VP} \\
V_{\text{NP-PP}} &\rightarrow \mathrm{NP}\ \mathrm{PP}
\end{aligned}
\quad \text{or} \quad
\begin{aligned}
\mathrm{S} &\rightarrow \mathrm{NP}\ V_{\text{PP-VP}} \\
V_{\text{PP-VP}} &\rightarrow \mathrm{PP}\ \mathrm{VP}
\end{aligned}
$$

- More?

Permutations

- The nonterminals in the right-hand side of a rule define a permutation between the languages
  - wlog, we assume the source language nonterminals are in order
  - intermingled terminal symbols do not affect binarization ability
- Example: $S \rightarrow \quad NP^{(1)}\ VP^{(2)}\ PP^{(3)},\ NP^{(1)}\ PP^{(3)}\ VP^{(2)}$
  - permutation: 1 3 2



Synchronous binarization

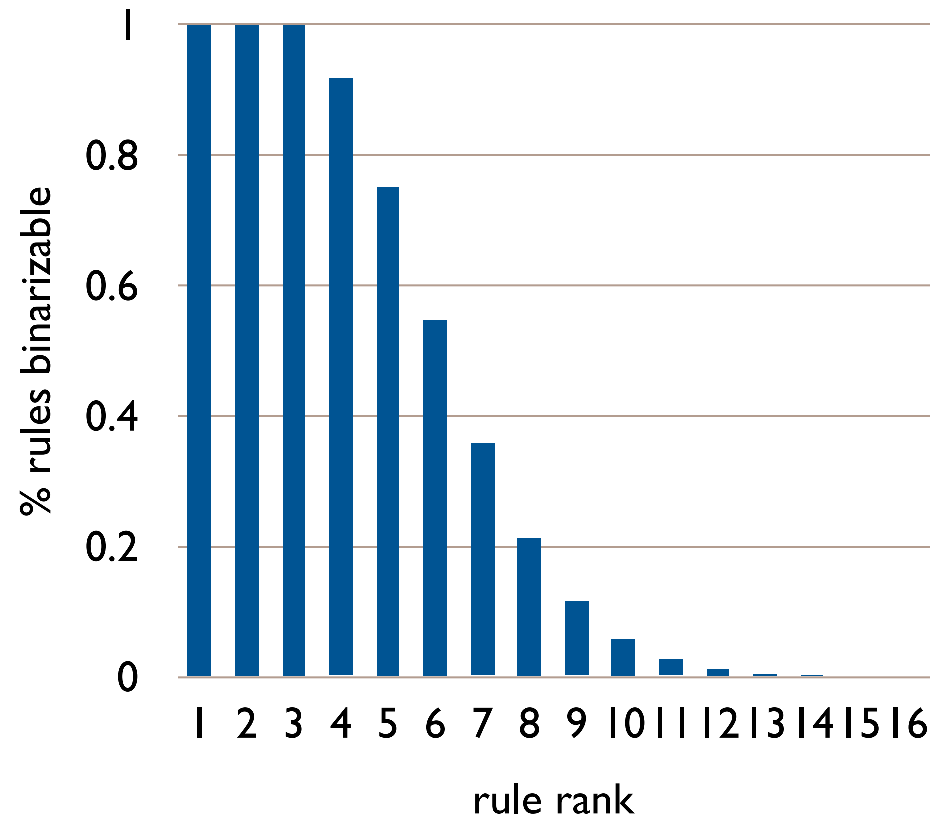
- Bad news: synchronous grammars can't be binarized in the general case (Shapiro & Stephens, 1991; Wu, 1997) *
- Famous examples: the (2,4,1,3) and (3,1,4,2) permutations

- What makes these unbinarizable?
  - Crucial: parsing works by combining *adjacent* elements
  - No pair of alignments here is adjacent in *both* languages simultaneously

(*) Technically, you can binarize any synchronous grammar, but you may increase the ***fan-out****, which mitigates the potential gains.*

Synchronous binarization

- As the rank of a rule grows, the percentage of binarizable rules approaches 0



- In summary:
  - We can't binarize all rules
  - The first unbinarizable rule has rank 4

Silver lining

- Empirically, we don't observe that many non-binarizable rules (Zhang et al., 2006):

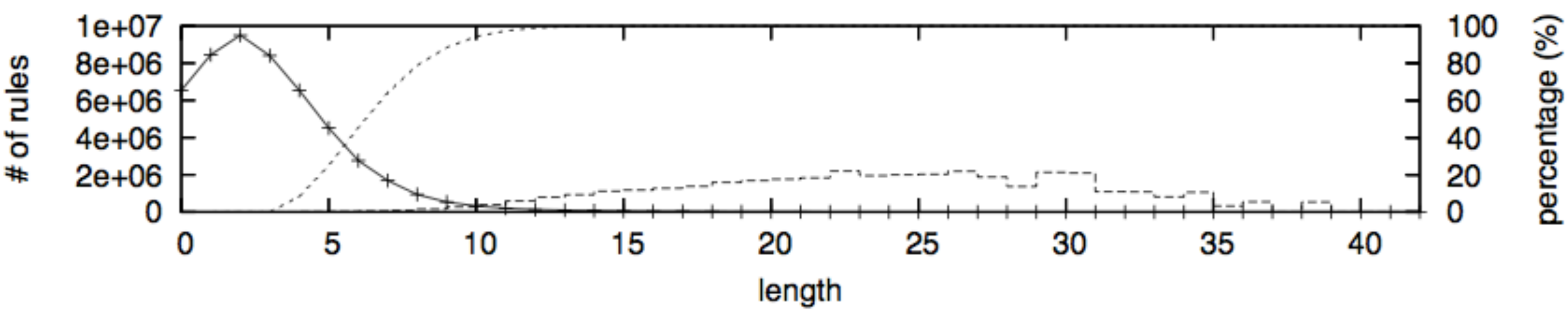


Figure 6: The solid-line curve represents the distribution of all rules against permutation lengths. The dashed-line stairs indicate the percentage of non-binarizable rules in our initial rule set while the dotted-line denotes that percentage among all permutations.

- ...and we can safely throw out the ones we do find
  - 99.7% of rules extracted were binarizable
  - many not were due to alignment errors

Decoding as parsing

Synchronous decoding

- Enough parsing; what we care about is decoding
- Parsing is relevant, though, because we can view decoding as a task where we are doing synchronous parsing but we don't happen to know the target side text
- This works by parsing with a source-side projection of the synchronous grammar rules
  - At the end, we can follow backpointers to discover the most probable target side

Updated data structure

- Just like regular parsing, we combine items in pairs to produce new items over larger spans:

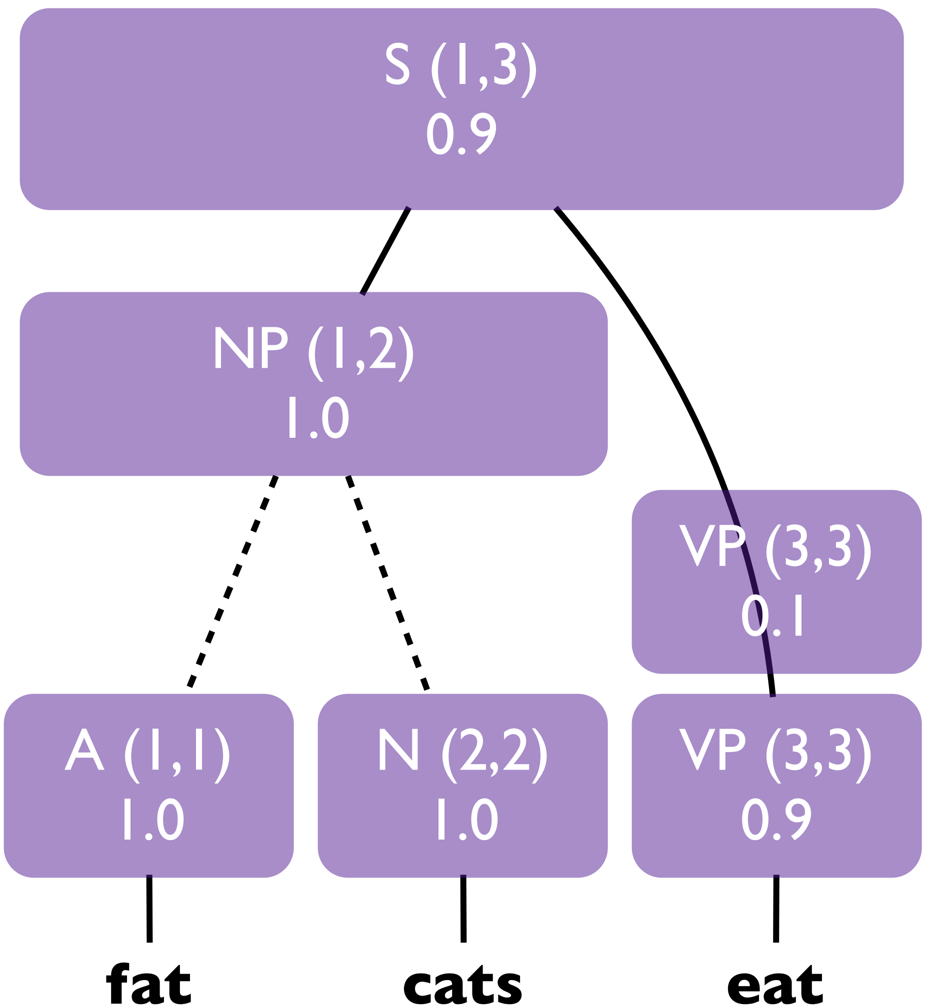
$$\frac{(A,1,1) \quad (N,2,2)}{(NP,1,2)}$$

- However, we also have to maintain our guess of the target side

$A \rightarrow$ fat, guapos
$N \rightarrow$ cats, gatos
$VP \rightarrow$ eat, comen
$VP \rightarrow$ eat, como
$NP \rightarrow A^{(1)}\ N^{(2)},\ N^{(2)}\ A^{(1)}$
$S \rightarrow NP^{(1)}\ VP^{(2)},\ NP^{(1)}\ VP^{(2)}$

Decoding

- Again, a bottom-up process



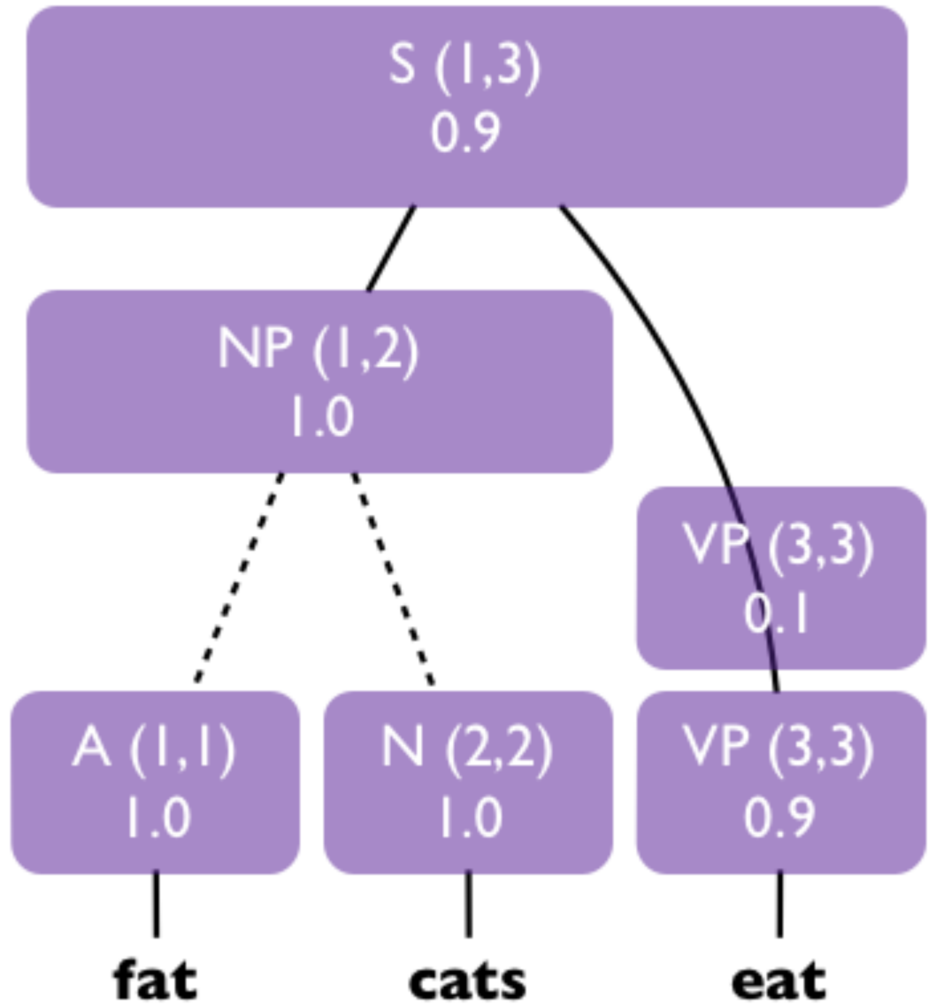
A → fat, guapos	1.0
N → cats, gatos	1.0
VP → eat, comen	0.1
VP → eat, como	0.9
NP → $A^{(1)}$ $N^{(2)}$, $N^{(2)}$ $A^{(1)}$	1.0
S → $NP^{(1)}$ $VP^{(2)}$, $NP^{(1)}$ $VP^{(2)}$	1.0

Legend

- straight rule application (solid line)
- inverted rule application (dashed line)

Getting the translation

- Follow the backpointers
  - (S,1,3)
    - (NP,1,2)
      - (N,2,2) → gatos
      - (A,1,1) → guapos
    - (VP,3,3) → como
- translation:
  gatos guapos como
  * *cats fat 1ps-eat*

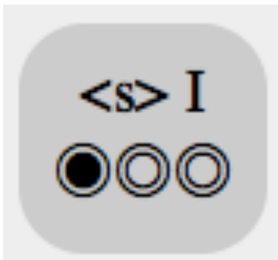


What happened?

- We forgot the language model
- We're inventing the target side (which is what decoding does), so we need to incorporate it
- How?
  - Stack-based decoding: we maintained the last word
  - Integration was easy because hypotheses always extended to the right
  - Here, hypotheses are *merged* either straight or inverted

Language model integration

phrase-based



+

tengo
→am

=

synchronous grammars

A (1,1)
1.0

+

N (2,2)
1.0

$$NP \rightarrow A^{(1)}\ N^{(2)},\ N^{(2)}\ A^{(1)}$$

=

N (1,2)
1.0
gatos guapos

A (1,1)
1.0

+

N (2,2)
1.0

$$NP \rightarrow A^{(1)}\ N^{(2)},\ A^{(1)}\ N^{(2)}$$

=

N (1,2)
1.0
guapos gatos

Language model integration

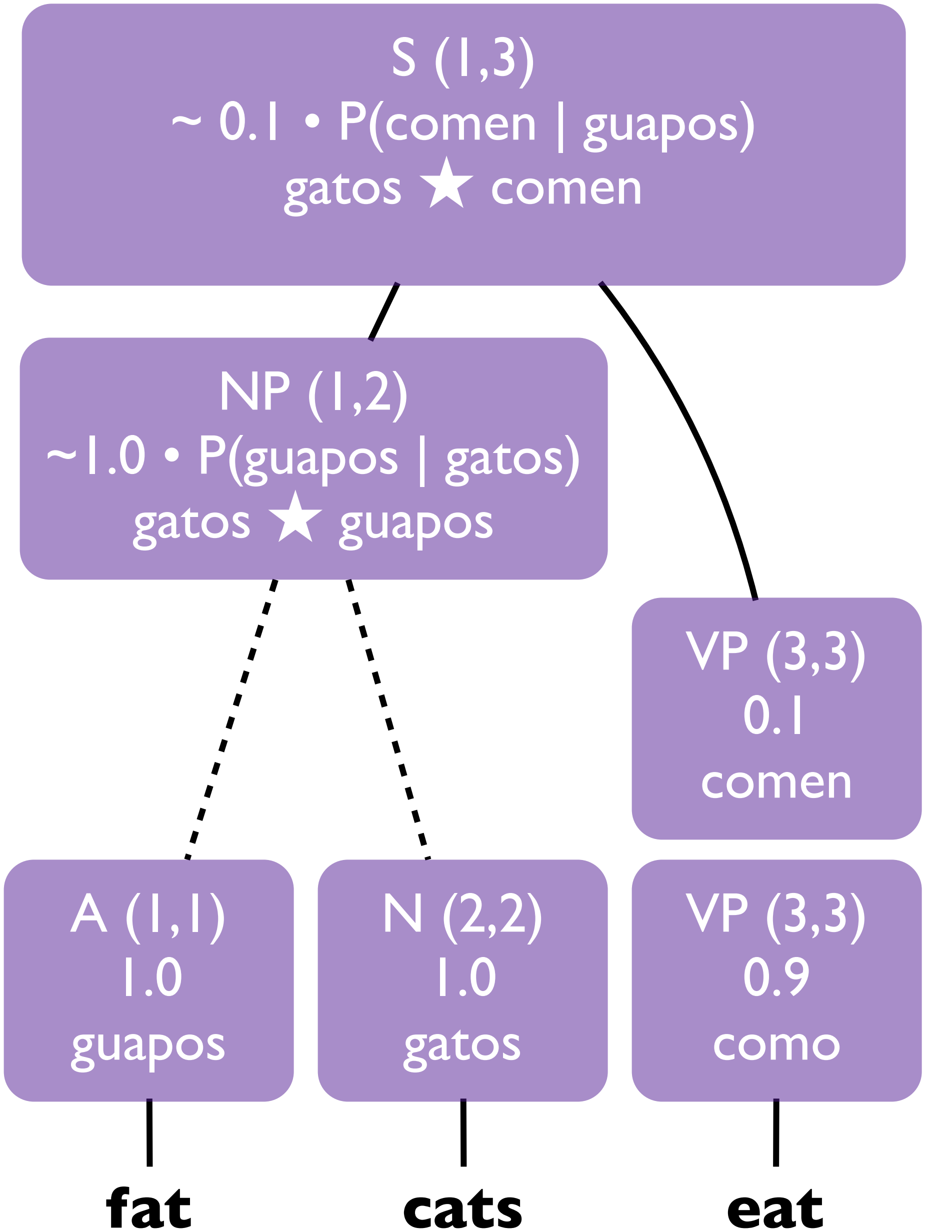
- We still maintain a chart of items, but now the items have to contain the target side words
- Just like regular parsing, we combine items in pairs to produce new items over larger spans
- When items are merged, we can use these words to compute a language model probability
- Formally, we are intersecting a *context-free grammar* (the translation model) with a *regular grammar* (Bar-Hillel et al., 1964; Wu, 1996)

Updated data structure

- With dynamic programming, we only need a word on either side
  - (for bigram LMs; for the general case, see Chiang (2007, §5.3.2))
  - Following Chiang, we represent the elided middle portion with a ★
  - The complete string can be reconstructed by following the backpointers

```
struct item {
  // d.p. state
  string nt;
  int i, j;
  string left_words;
  string right_words;
  // backpointer
  float score;
  Rule* rule;
  item* rhs1,
        rhs2;
}
```

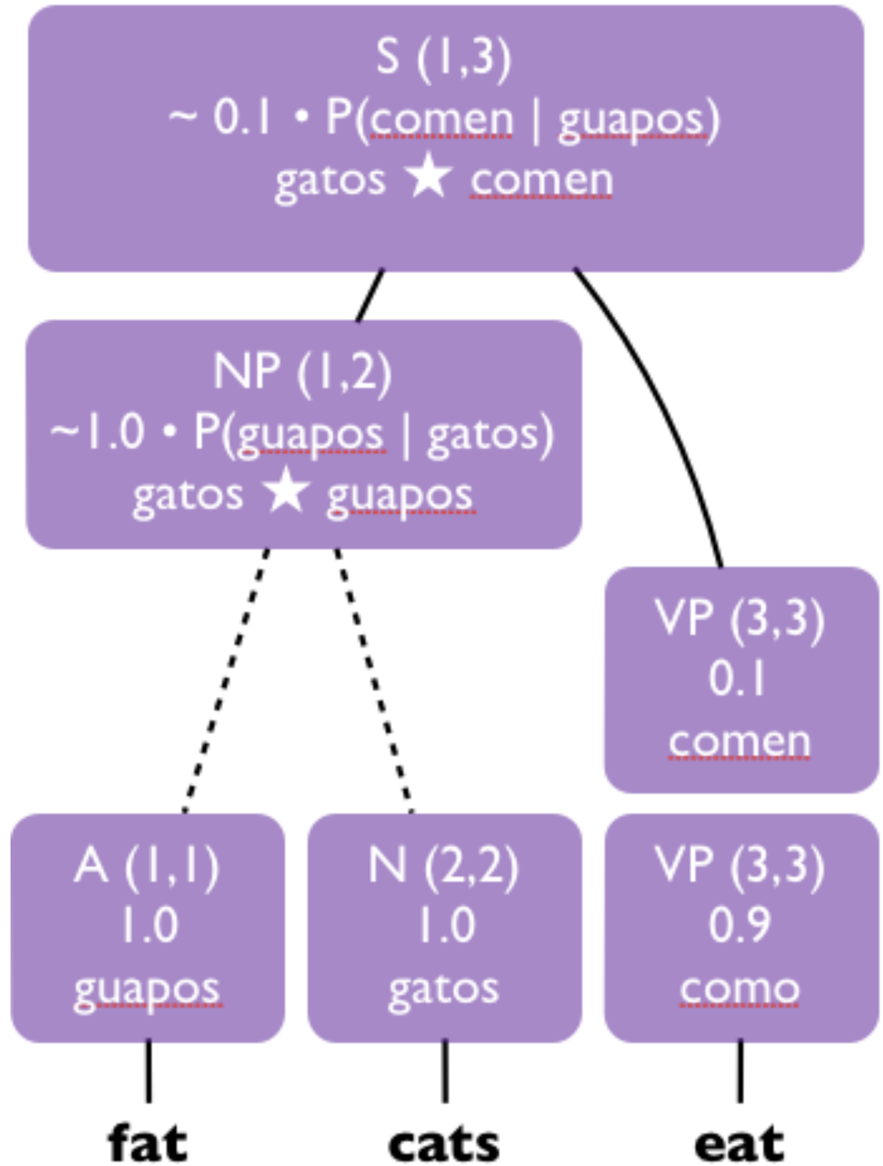
Decoding with an integrated LM



A → fat, guapos	1.0
N → cats, gatos	1.0
VP → eat, comen	0.1
VP → eat, como	0.9
NP → $A^{(1)}$ $N^{(2)}$, $N^{(2)}$ $A^{(1)}$	1.0
S → $NP^{(1)}$ $VP^{(2)}$, $NP^{(1)}$ $VP^{(2)}$	1.0

Getting the translation

- Follow the backpointers
  - (S,1,3,gatos★comen)
    - (NP,1,2,gatos★guapos)
      - (N,2,2,gatos) → gatos
      - (A,1,1,guapos) → guapos
    - (VP,3,3,comen) → comen
- translation:
  gatos guapos comen
  cats fat 3pp-eat



Pruning

- We have also not dealt much with ambiguity and competition amongst hypotheses
- In general, there are too many hypotheses to consider, so we keep only the top k of them (per input span (i,j))
- When considering a span (i,j) and a split point k, we have a large number of ways to combine items
  - there can be any number of applicable rules
  - there can be up to k items located at span (i,k)
  - there can be up to k items located at span (k,j)

Applying a unary rule

- The naive way is to consider the full cross product

		$[X, 6, 8; \text{the scheme}]$ 1	$[X, 6, 8; \text{the plan}]$ 4	$[X, 6, 8; \text{the project}]$ 7
$X \rightarrow \langle \text{cong } X_{\boxed{1}}, \text{from } X_{\boxed{1}} \rangle$	1	2.1	5.1	8.2
$X \rightarrow \langle \text{cong } X_{\boxed{1}}, \text{from the } X_{\boxed{1}} \rangle$	2	5.5	8.5	11.5
$X \rightarrow \langle \text{cong } X_{\boxed{1}}, \text{since } X_{\boxed{1}} \rangle$	6	7.7	10.6	13.1
$X \rightarrow \langle \text{cong } X_{\boxed{1}}, \text{through } X_{\boxed{1}} \rangle$	10	11.1	14.3	17.3

$\Rightarrow$

$[X, 5, 8; \text{from the} \star \text{the scheme}] : 2.1$

$[X, 5, 8; \text{from the} \star \text{the plan}] : 5.1$

$[X, 5, 8; \text{from the} \star \text{the scheme}] : 5.5$

$[X, 5, 8; \text{since the} \star \text{the scheme}] : 7.7$

.

Cube pruning

- When considering a span (i,j) of a length-N sentence:
  - *unary rules:* there are rk items to compute (r the number of rules, k the number of child items)
  - *binary rules:* there are Nrk^2 items to compute (since there are O(N) split points)
- However, we're only going to be keeping the top k of them!
  - this problem gets worse as k gets larger
- We'd like to avoid computing all of these new items, which we accomplish with cube pruning

Cube pruning

- We start with sorted lists of rules and the items they applied to
- Observation:
  - the best item comes from the best rule and the best cell
  - the next-best item uses either the 2nd best rule *or* the 2nd-best cell

	rule	rhsl	rhsr
1	1	1	1
2	2	7	3
3	4	9	4
...			

best item

	rule	rhsl	rhsr
1	1	1	1
2	2	7	3
3	4	9	4
...			

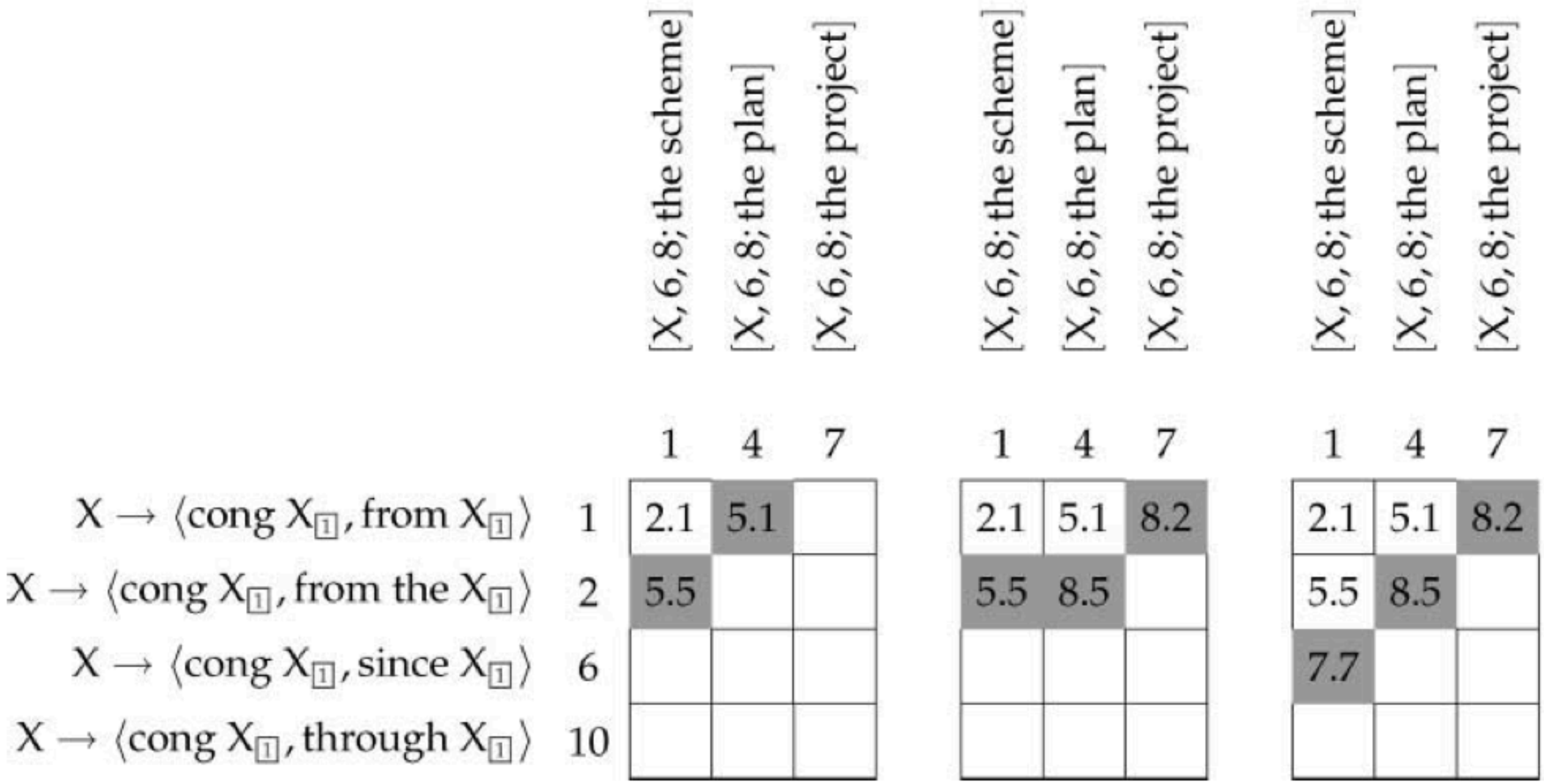
2nd-best

	rule	rhsl	rhsr
1	1	1	1
2	2	7	3
3	4	9	4
...			

3rd-best

Applying a unary rule

- The Huang & Chiang (2005) way:



Cube pruning

- We haven't discussed the language model, which complicates this procedure by making it *nonmonotonic*
- But that's the basic idea

Summary

- Today, we have reviewed
  - Monolingual parsing
  - Synchronous (bilingual) parsing
  - Decoding as parsing with an intersected bigram language model
- We have also briefly touched on efficiency considerations with cube pruning

Advanced topics

Advanced topics: implicit binarization

- We'd decoded in an ITG settings, where the rules all look like this:

$$X \rightarrow \text{boy, chico} \qquad \text{(lexical)}$$

$$X \rightarrow X^{(1)}\, X^{(2)},\; X^{(2)}\, X^{(1)} \qquad \text{(inverted)}$$

$$X \rightarrow X^{(1)}\, X^{(2)},\; X^{(1)}\, X^{(2)} \qquad \text{(straight)}$$

  - This is the closest thing to Chomsky Normal Form for synchronous grammars

- How do we decode with intermingled terminals and nonterminals?

$$X \rightarrow \text{the } X^{(1)} \text{ was } X^{(2)},\; \text{el } X^{(1)} \text{ era } X^{(2)}$$

Advanced topics: implicit binarization

- One answer: binarize (terminals can always be binarized):

$\mathbf{X} \rightarrow$ the $\mathbf{X}$ was $\mathbf{X}$, el $\mathbf{X}$ era $\mathbf{X}$

$\mathbf{X} \rightarrow$ the $\mathbf{X}_{174}$, el $\mathbf{X}_{174}$

$\mathbf{X}_{174} \rightarrow \mathbf{X}\ \mathbf{X}_{295}, \mathbf{X}\ \mathbf{X}_{295}$

$\mathbf{X}_{295} \rightarrow$ was $\mathbf{X}$, era $\mathbf{X}$

- However, this is inefficient:
    - it leads to a huge blowup in the number of nonterminals
    - it introduces a split point that has to be searched over (avoidable in this case, but not always)

Advanced topics: implicit binarization

- Instead, we'd like to do implicit, Earley-style binarization

Advanced topics: spurious ambiguity

- *Spurious ambiguity* - multiple structures leading to the same interpretation
- Especially problematic in ITG with its weak grammar
- This can be addressed in various ways
  - Grammar canonical forms