Basic Parsing with Context-Free Grammars

Some slides adapted from Julia Hirschberg and Dan Jurafsky

Announcements

- To view past videos:
  - http://globe.cvn.columbia.edu:8080/oncampus.php?c=133ae14752e27fde909fdbd64c06b337
- Usually available only for 1 week. Right now, available for all previous lectures

Earley Parsing

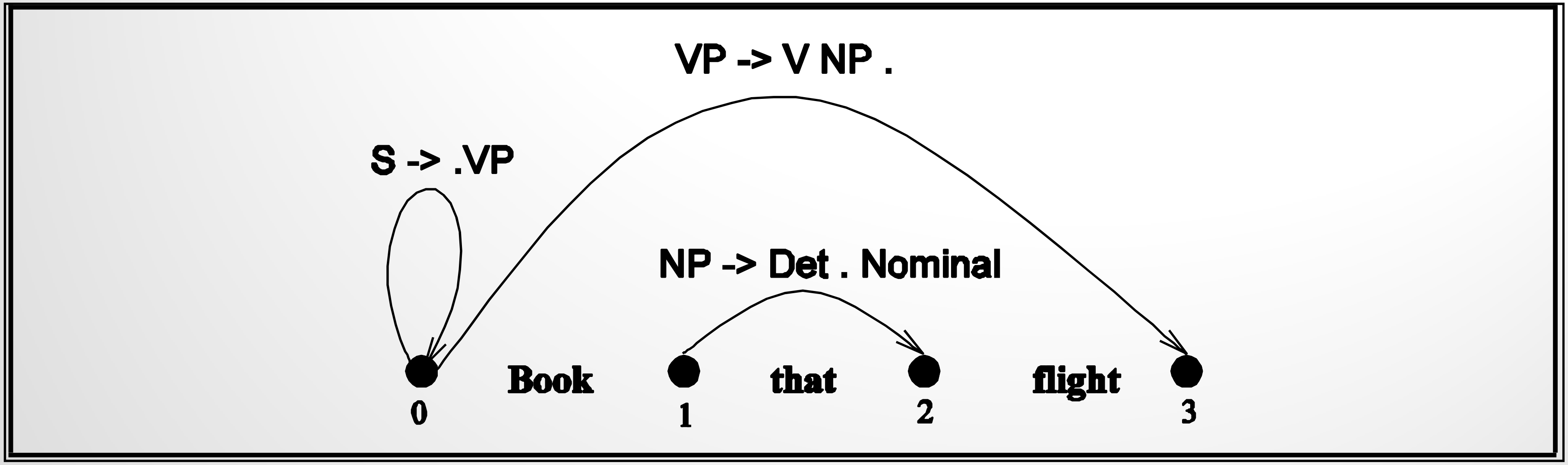
- Allows arbitrary CFGs
- Fills a table in a single sweep over the input words
  - Table is length N+1; N is number of words
  - Table entries represent
    - Completed constituents and their locations
    - In-progress constituents
    - Predicted constituents

States/Locations

- It would be nice to know where these things are in the input so…

S -> · VP [0,0]	A VP is predicted at the start of the sentence
NP -> Det · Nominal [1,2]	An NP is in progress; the Det goes from 1 to 2
VP -> V NP · [0,3]	A VP has been found starting at 0 and ending at 3

Graphically



Earley Algorithm

- March through chart left-to-right.
- At each step, apply 1 of 3 operators
  - Predictor
    - Create new states representing top-down expectations
  - Scanner
    - Match word predictions (rule with word after dot) to words
  - Completer
    - When a state is complete, see what rules were looking for that completed constituent
- Done when an S spans from 0 to n

Predictor

- Given a state
  - With a non-terminal to right of dot (not a part-of-speech category)
  - Create a new state for each expansion of the non-terminal
  - Place these new states into same chart entry as generated state, beginning and ending where generating state ends.
  - So predictor looking at
    - S -> . VP [0,0]
  - results in
    - VP -> . Verb [0,0]
    - VP -> . Verb NP [0,0]

Scanner

- Given a state
  - With a non-terminal to right of dot that is a part-of-speech category
  - If the next word in the input matches this POS
  - Create a new state with dot moved over the non-terminal
  - So scanner looking at VP -> . Verb NP [0,0]
  - If the next word, “book”, can be a verb, add new state:
    - VP -> Verb . NP [0,1]
  - Add this state to chart entry following current one
  - Note: Earley algorithm uses top-down input to disambiguate POS! Only POS predicted by some state can get added to chart!

Completer

- Applied to a state when its dot has reached right end of role.
- Parser has discovered a category over some span of input.
- Find and advance all previous states that were looking for this category
  - copy state, move dot, insert in current chart entry
- Given:
  - NP -> Det Nominal . [1,3]
  - VP -> Verb. NP [0,1]
- Add
  - VP -> Verb NP . [0,3]

How do we know we are done?

- Find an S state in the final column that spans from 0 to n and is complete.

- If that's the case you're done.
  - S -> α · [0,n]

Earley

- More specifically…

  1. Predict all the states you can upfront

  2. Read a word
     1. Extend states based on matches
     2. Add new predictions
     3. Go to 2

  3. Look at N to see if you have a winner

Example

- Book that flight
- We should find… an S from 0 to 3 that is a completed state…

CFG for Fragment of English

S → NP VP	VP → V
S → Aux NP VP	PP -> Prep NP
NP → Det Nom	N → old | dog | footsteps | young
NP →PropN	V → dog | include | prefer
Nom -> Adj Nom	Aux → does
Nom → N	Prep →from | to | on | of
Nom → N Nom	PropN → Bush | McCain | Obama
Nom → Nom PP	Det → that | this | a| the
VP → V NP	Adj -> old | green | red

Example

Chart[0]	S0	$\gamma \rightarrow \bullet S$	[0,0]	Dummy start state
	S1	$S \rightarrow \bullet NP\ VP$	[0,0]	Predictor
	S2	$S \rightarrow \bullet Aux\ NP\ VP$	[0,0]	Predictor
	S3	$S \rightarrow \bullet VP$	[0,0]	Predictor
	S4	$NP \rightarrow \bullet Pronoun$	[0,0]	Predictor
	S5	$NP \rightarrow \bullet Proper\text{-}Noun$	[0,0]	Predictor
	S6	$NP \rightarrow \bullet Det\ Nominal$	[0,0]	Predictor
	S7	$VP \rightarrow \bullet Verb$	[0,0]	Predictor
	S8	$VP \rightarrow \bullet Verb\ NP$	[0,0]	Predictor
	S9	$VP \rightarrow \bullet Verb\ NP\ PP$	[0,0]	Predictor
	S10	$VP \rightarrow \bullet Verb\ PP$	[0,0]	Predictor
	S11	$VP \rightarrow \bullet VP\ PP$	[0,0]	Predictor

Example

Chart[1]	S12	*Verb* $\rightarrow$ *book* •	[0,1]	Scanner
	S13	*VP* $\rightarrow$ *Verb* •	[0,1]	Completer
	S14	*VP* $\rightarrow$ *Verb* • *NP*	[0,1]	Completer
	S15	*VP* $\rightarrow$ *Verb* • *NP PP*	[0,0]	Predictor
	S16	*VP* $\rightarrow$ *Verb* • *PP*	[0,0]	Predictor
	S17	*S* $\rightarrow$ *VP* •	[0,1]	Completer
	S18	*VP* $\rightarrow$ *VP* • *PP*	[0,1]	Completer
	S19	*NP* $\rightarrow$ • *Pronoun*	[1,1]	Predictor
	S20	*NP* $\rightarrow$ • *Proper-Noun*	[1,1]	Predictor
	S21	*NP* $\rightarrow$ • *Det Nominal*	[1,1]	Predictor
	S22	*PP* $\rightarrow$ • *Prep NP*	[1,1]	Predictor

Example

Chart[2]	S23	*Det* $\rightarrow$ *that* $\bullet$	[1,2]	Scanner
	S24	*NP* $\rightarrow$ *Det* $\bullet$ *Nominal*	[1,2]	Completer
	S25	*Nominal* $\rightarrow$ $\bullet$ *Noun*	[2,2]	Predictor
	S26	*Nominal* $\rightarrow$ $\bullet$ *Nominal Noun*	[2,2]	Predictor
	S27	*Nominal* $\rightarrow$ $\bullet$ *Nominal PP*	[2,2]	Predictor
Chart[3]	S28	*Noun* $\rightarrow$ *flight* $\bullet$	[2,3]	Scanner
	S29	*Nominal* $\rightarrow$ *Noun* $\bullet$	[2,3]	Completer
	S30	*NP* $\rightarrow$ *Det Nominal* $\bullet$	[1,3]	Completer
	S31	*Nominal* $\rightarrow$ *Nominal* $\bullet$ *Noun*	[2,3]	Completer
	S32	*Nominal* $\rightarrow$ *Nominal* $\bullet$ *PP*	[2,3]	Completer
	S33	*VP* $\rightarrow$ *Verb NP* $\bullet$	[0,3]	Completer
	S34	*VP* $\rightarrow$ *Verb NP* $\bullet$ *PP*	[0,3]	Completer
	S35	*PP* $\rightarrow$ $\bullet$ *Prep NP*	[3,3]	Predictor
	S36	*S* $\rightarrow$ *VP* $\bullet$	[0,3]	Completer

Details

- What kind of algorithms did we just describe
  - Not parsers – recognizers
    - The presence of an S state with the right attributes in the right place indicates a successful recognition.
    - But no parse tree… no parser
    - That’s how we solve (not) an exponential problem in polynomial time

Converting Earley from Recognizer to Parser

- With the addition of a few pointers we have a parser
- Augment the “Completer” to point to where we came from.

Augmenting the chart with structural information

Chart[1]

S8	*Verb*	*book*	[0,1]	Scanner	
S9	*VP*	*Verb*	[0,1]	Completer	S8
S10	*S*	*VP*	[0,1]	Completer	S9
S11	*VP*	*Verb NP*	[0,1]	Completer	S8
S12	*NP*	*Det NOMINAL*	[1,1]	Predictor	
S13	*NP*	*Proper-Noun*	[1,1]	Predictor	

Chart[2]

Det	*that*	[1,2]	Scanner
NP	*Det NOMINAL*	[1,2]	Completer
NOMINAL	*Noun*	[2,2]	Predictor
NOMINAL	*Noun NOMINAL*	[2,2]	Predictor

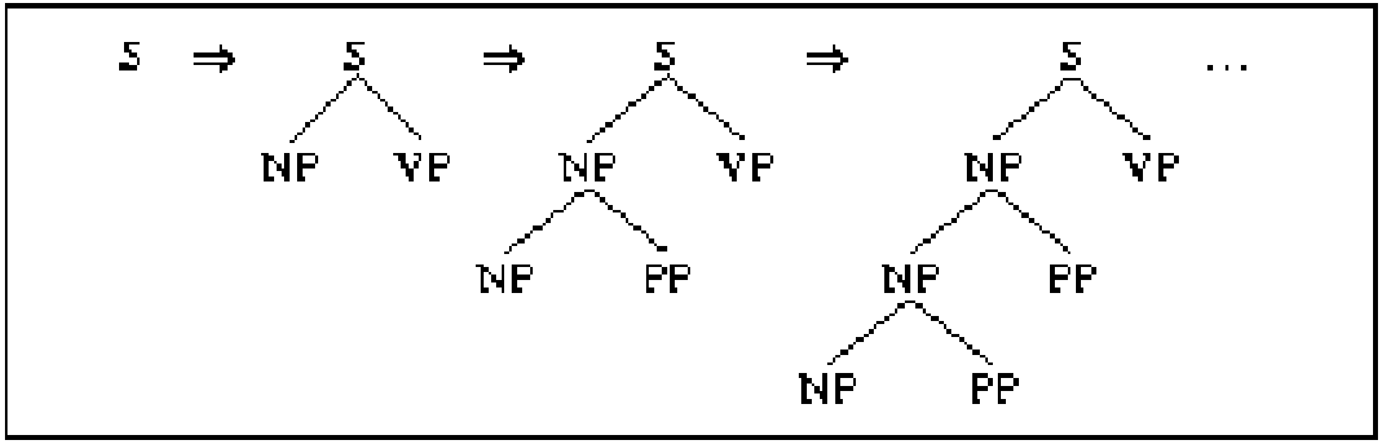
Retrieving Parse Trees from Chart

- All the possible parses for an input are in the table
- We just need to read off all the backpointers from every complete S in the last column of the table
- Find all the S -> X . [0,N+1]
- Follow the structural traces from the Completer
- Of course, this won't be polynomial time, since there could be an exponential number of trees
- We can at least represent ambiguity efficiently

Left Recursion vs. Right Recursion

- Depth-first search will never terminate if grammar is *left recursive* (e.g. NP --> NP PP)

$$(A \xrightarrow{*} \alpha A B, \alpha \xrightarrow{*} \varepsilon)$$



- Solutions:
  - Rewrite the grammar (automatically?) to a *weakly equivalent* one which is not left-recursive

    e.g. The man {on the hill with the telescope…}

    NP → NP PP (wanted: Nom plus a sequence of PPs)

    NP → Nom PP

    NP → Nom

    Nom → Det N

    …becomes…

    NP → Nom NP’

    Nom → Det N

    NP’ → PP NP’ (wanted: a sequence of PPs)

    NP’ → e

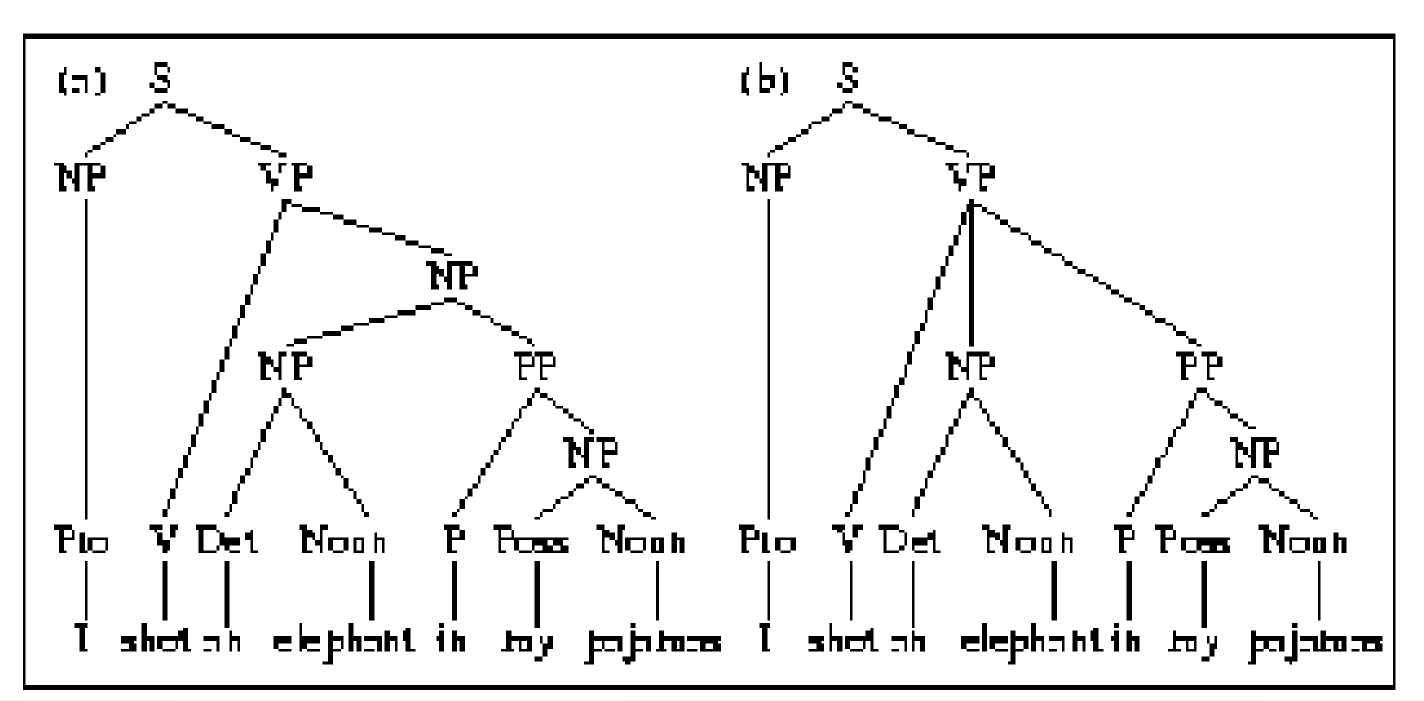
    - *Not so obvious what these rules mean…*

- Harder to detect and eliminate *non-immediate left recursion*
    - NP --> Nom PP
    - Nom --> NP
- Fix depth of search explicitly
- Rule ordering: non-recursive rules first
  - NP --> Det Nom
  - NP --> NP PP

Another Problem: Structural ambiguity

- Multiple legal structures
  - Attachment (e.g. I saw a man on a hill with a telescope)
  - Coordination (e.g. younger cats and dogs)
  - NP bracketing (e.g. Spanish language teachers)

NP vs. VP Attachment



- Solution?
  - Return all possible parses and disambiguate using “other methods”

Summing Up

- Parsing is a search problem which may be implemented with many control strategies
  - Top-Down or Bottom-Up approaches each have problems
    - Combining the two solves some but not all issues
  - Left recursion
  - Syntactic ambiguity
- Rest of today (and next time): Making use of statistical information about syntactic constituents
  - Read Ch 14

Probabilistic Parsing

How to do parse disambiguation

- Probabilistic methods
- Augment the grammar with probabilities
- Then modify the parser to keep only most probable parses
- And at the end, return the most probable parse

Probabilistic CFGs

- The probabilistic model
  - Assigning probabilities to parse trees
- Getting the probabilities for the model
- Parsing with probabilities
  - Slight modification to dynamic programming approach
  - Task is to find the max probability tree for an input

Probability Model

- Attach probabilities to grammar rules
- The expansions for a given non-terminal sum to 1

  VP -> Verb .55

  VP -> Verb NP .40

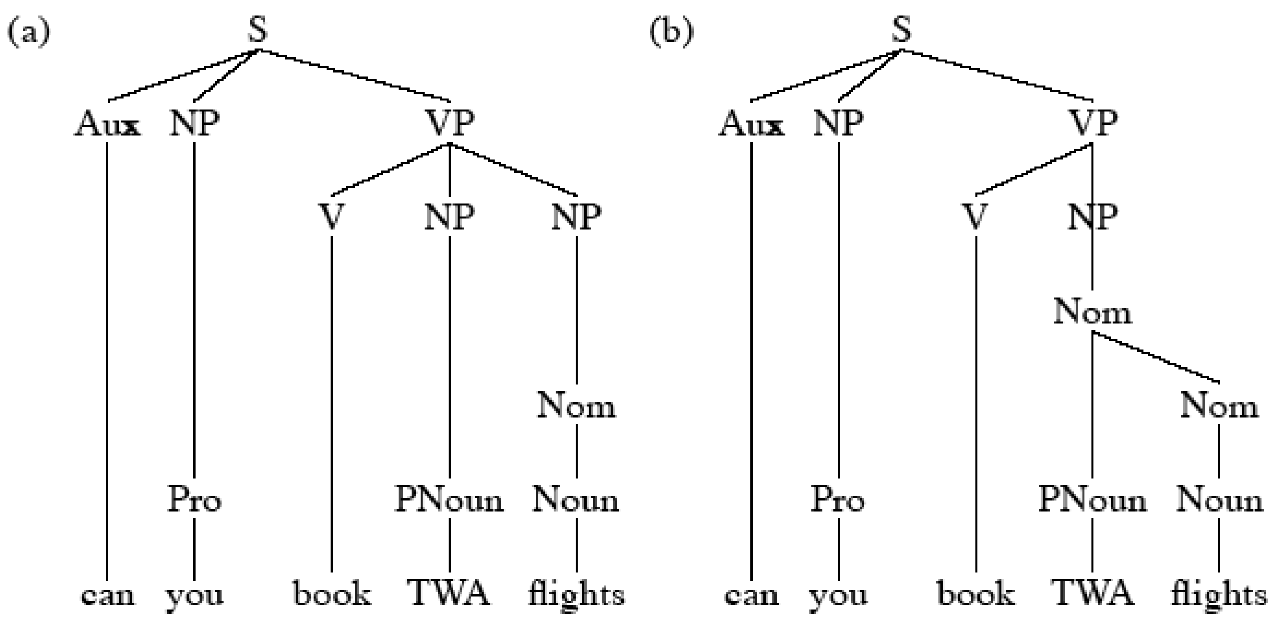
  VP -> Verb NP NP .05

  - Read this as P(Specific rule | LHS)

PCFG

$S \rightarrow NP\ VP$	[.80]	$Det \rightarrow that\ [.05] \mid the\ [.80] \mid a$	[.15]
$S \rightarrow Aux\ NP\ VP$	[.15]	$Noun \rightarrow book$	[.10]
$S \rightarrow VP$	[.05]	$Noun \rightarrow flights$	[.50]
$NP \rightarrow Det\ Nom$	[.20]	$Noun \rightarrow meal$	[.40]
$NP \rightarrow Proper\text{-}Noun$	[.35]	$Verb \rightarrow book$	[.30]
$NP \rightarrow Nom$	[.05]	$Verb \rightarrow include$	[.30]
$NP \rightarrow Pronoun$	[.40]	$Verb \rightarrow want$	[.40]
$Nom \rightarrow Noun$	[.75]	$Aux \rightarrow can$	[.40]
$Nom \rightarrow Noun\ Nom$	[.20]	$Aux \rightarrow does$	[.30]
$Nom \rightarrow Proper\text{-}Noun\ Nom$	[.05]	$Aux \rightarrow do$	[.30]
$VP \rightarrow Verb$	[.55]	$Proper\text{-}Noun \rightarrow TWA$	[.40]
$VP \rightarrow Verb\ NP$	[.40]	$Proper\text{-}Noun \rightarrow Denver$	[.40]
$VP \rightarrow Verb\ NP\ NP$	[.05]	$Pronoun \rightarrow you\ [.40] \mid I\ [.60]$	

PCFG



Rules			P	Rules			P
S	→	Aux NP VP	.15	S	→	Aux NP VP	.15
NP	→	Pro	.40	NP	→	Pro	.40
VP	→	V NP NP	.05	VP	→	V NP	.40
NP	→	Nom	.05	NP	→	Nom	.05
NP	→	PNoun	.35	Nom	→	PNoun Nom	.05
Nom	→	Noun	.75	Nom	→	Noun	.75
Aux	→	Can	.40	Aux	→	Can	.40
NP	→	Pro	.40	NP	→	Pro	.40
Pro	→	you	.40	Pro	→	you	.40
Verb	→	book	.30	Verb	→	book	.30
PNoun	→	TWA	.40	Pnoun	→	TWA	.40
Noun	→	flights	.50	Noun	→	flights	.50

Probability Model (1)

- A derivation (tree) consists of the set of grammar rules that are in the tree

- The probability of a tree is just the product of the probabilities of the rules in the derivation.

Probability model

$$P(T,S) = \prod_{n \in T} p(r_n)$$

P(T,S) = P(T)P(S|T) = P(T); since P(S|T)=1

$$\begin{aligned} P(T_l) &= .15 * .40 * .05 * .05 * .35 * .75 * .40 * .40 * .40 \\ &\quad * .30 * .40 * .50 \\ &= 1.5 \times 10^{-6} \end{aligned}$$

$$\begin{aligned} P(T_r) &= .15 * .40 * .40 * .05 * .05 * .75 * .40 * .40 * .40 \\ &\quad * .30 * .40 * .50 \\ &= 1.7 \times 10^{-6} \end{aligned}$$

Probability Model (1.1)

- The probability of a word sequence P(S) is the probability of its tree in the unambiguous case.
- It's the sum of the probabilities of the trees in the ambiguous case.

Getting the Probabilities

- From an annotated database (a treebank)
  - So for example, to get the probability for a particular VP rule just count all the times the rule is used and divide by the number of VPs overall.

TreeBanks

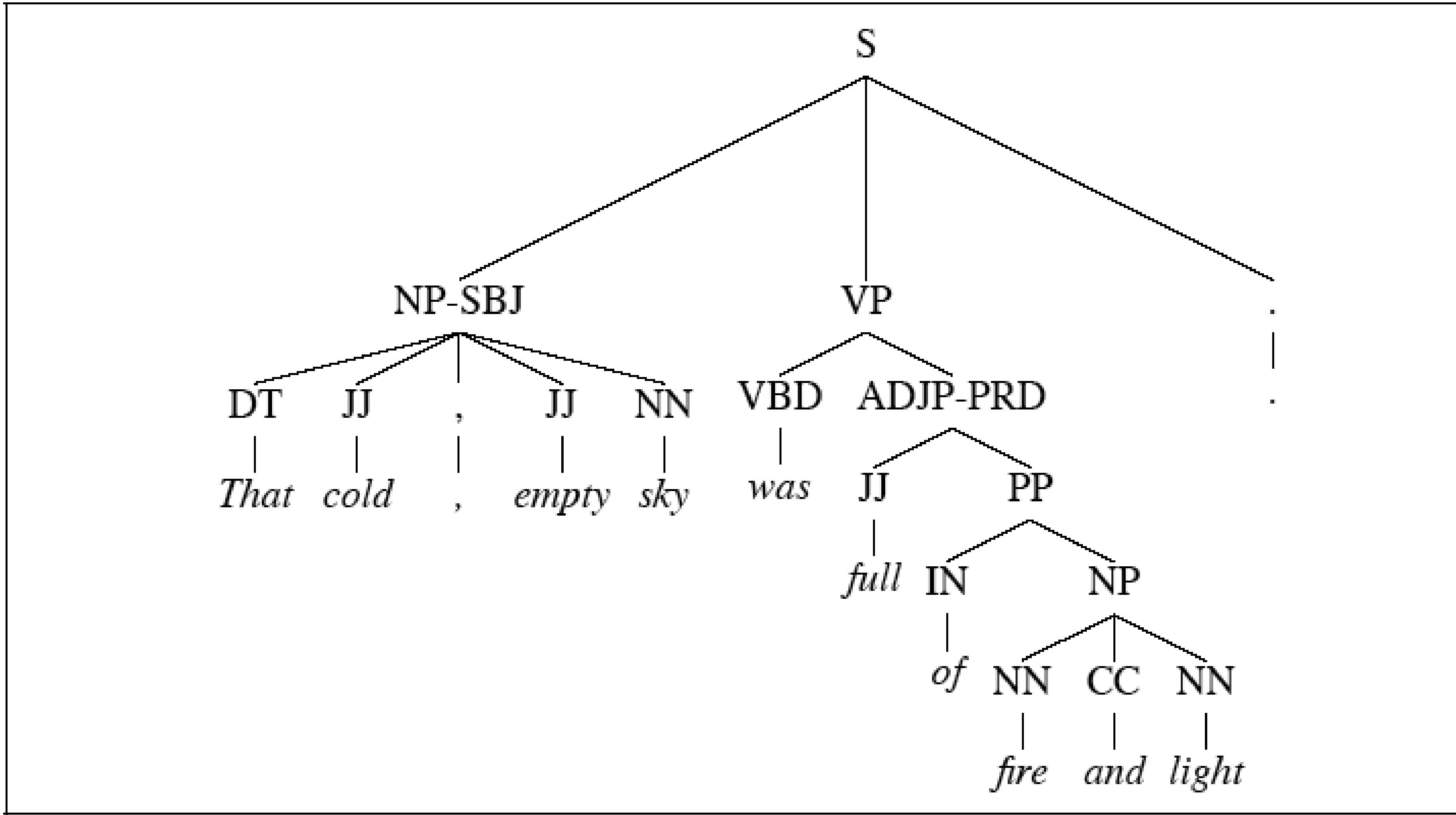
```
((S
   (NP-SBJ (DT That)
     (JJ cold) (, ,)
     (JJ empty) (NN sky) )
   (VP (VBD was)
     (ADJP-PRD (JJ full)
       (PP (IN of)
         (NP (NN fire)
           (CC and)
           (NN light) ))))
   (. .) ))
```

(a)

```
((S
   (NP-SBJ The/DT flight/NN )
   (VP should/MD
     (VP arrive/VB
       (PP-TMP at/IN
         (NP eleven/CD a.m/RB ))
       (NP-TMP tomorrow/NN )))))
```

(b)

Treebanks


S
NP-SBJ
VP
.
DT
JJ
,
JJ
NN
VBD
ADJP-PRD
.
That
cold
,
empty
sky
was
JJ
PP
full
IN
NP
of
NN
CC
NN
fire
and
light

Treebanks

```
( (S ('' '')
    (S-TPC-2
      (NP-SBJ-1 (PRP We) )
      (VP (MD would)
        (VP (VB have)
          (S
            (NP-SBJ (-NONE- *-1) )
            (VP (TO to)
              (VP (VB wait)
                (SBAR-TMP (IN until)
                  (S
                    (NP-SBJ (PRP we) )
                    (VP (VBP have)
                      (VP (VBN collected)
                        (PP-CLR (IN on)
                          (NP (DT those) (NNS assets) )))))))))))))
    (, ,) ('' '')
    (NP-SBJ (PRP he) )
    (VP (VBD said)
      (S (-NONE- *T*-2) ))
    (. .) ))
```

Treebank Grammars

<table>
<tr><td>S</td><td>→</td><td>NP VP .</td><td>PRP</td><td>→</td><td>we | he</td></tr>
<tr><td></td><td></td><td>NP VP</td><td>DT</td><td>→</td><td>the | that | those</td></tr>
<tr><td></td><td></td><td>" S " , NP VP .</td><td>JJ</td><td>→</td><td>cold | empty | full</td></tr>
<tr><td></td><td></td><td>-NONE-</td><td>NN</td><td>→</td><td>sky | fire | light | flight</td></tr>
<tr><td></td><td></td><td>DT NN</td><td>NNS</td><td>→</td><td>assets</td></tr>
<tr><td></td><td></td><td>DT NN NNS</td><td>CC</td><td>→</td><td>and</td></tr>
<tr><td></td><td></td><td>NN CC NN</td><td>IN</td><td>→</td><td>of | at | until | on</td></tr>
<tr><td></td><td></td><td>CD RB</td><td>CD</td><td>→</td><td>eleven</td></tr>
<tr><td>NP</td><td>→</td><td>DT JJ , JJ NN</td><td>RB</td><td>→</td><td>a.m</td></tr>
<tr><td></td><td></td><td>PRP</td><td>VB</td><td>→</td><td>arrive | have | wait</td></tr>
<tr><td></td><td></td><td>-NONE-</td><td>VBD</td><td>→</td><td>said</td></tr>
<tr><td>VP</td><td>→</td><td>MD VP</td><td>VBP</td><td>→</td><td>have</td></tr>
<tr><td></td><td></td><td>VBD ADJP</td><td>VBN</td><td>→</td><td>collected</td></tr>
<tr><td></td><td></td><td>VBD S</td><td>MD</td><td>→</td><td>should | would</td></tr>
<tr><td></td><td></td><td>VB PP</td><td>TO</td><td>→</td><td>to</td></tr>
<tr><td></td><td></td><td>VB S</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>VB SBAR</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>VBP VP</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>VBN VP</td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td>TO VP</td><td></td><td></td><td></td></tr>
<tr><td>SBAR</td><td>→</td><td>IN S</td><td></td><td></td><td></td></tr>
<tr><td>ADJP</td><td>→</td><td>JJ PP</td><td></td><td></td><td></td></tr>
<tr><td>PP</td><td>→</td><td>IN NP</td><td></td><td></td><td></td></tr>
</table>

Lots of flat rules

NP → DT JJ NN
NP → DT JJ NNS
NP → DT JJ NN NN
NP → DT JJ JJ NN
NP → DT JJ CD NNS
NP → RB DT JJ NN NN
NP → RB DT JJ JJ NNS
NP → DT JJ JJ NNP NNS
NP → DT NNP NNP NNP NNP JJ NN
NP → DT JJ NNP CC JJ JJ NN NNS
NP → RB DT JJS NN NN SBAR
NP → DT VBG JJ NNP NNP CC NNP
NP → DT JJ NNS , NNS CC NN NNS NN
NP → DT JJ JJ VBG NN NNP NNP FW NNP
NP → NP JJ , JJ ‘‘ SBAR ’’ NNS

Example sentences from those rules

- Total: over 17,000 different grammar rules in the 1-million word Treebank corpus

(9.19) [$_{\text{DT}}$ The] [$_{\text{JJ}}$ state-owned] [$_{\text{JJ}}$ industrial] [$_{\text{VBG}}$ holding] [$_{\text{NN}}$ company] [$_{\text{NNP}}$ Instituto] [$_{\text{NNP}}$ Nacional] [$_{\text{FW}}$ de] [$_{\text{NNP}}$ Industria]

(9.20) [$_{\text{NP}}$ Shearson's] [$_{\text{JJ}}$ easy-to-film], [$_{\text{JJ}}$ black-and-white] "[$_{\text{SBAR}}$ Where We Stand]" [$_{\text{NNS}}$ commercials]

Probabilistic Grammar Assumptions

- We're assuming that there is a grammar to be used to parse with.
- We're assuming the existence of a large robust dictionary with parts of speech
- We're assuming the ability to parse (i.e. a parser)
- Given all that… we can parse probabilistically

Typical Approach

- Bottom–up (CKY) dynamic programming approach
- Assign probabilities to constituents as they are completed and placed in the table
- Use the max probability for each constituent going up

What's that last bullet mean?

- Say we're talking about a final part of a parse
  - $S \rightarrow {}_0NP_iVP_j$

The probability of the S is…
P(S->NP VP)*P(NP)*P(VP)

The green stuff is already known. We're doing bottom-up parsing

Max

- I said the P(NP) is known.
- What if there are multiple NPs for the span of text in question (0 to i)?
- Take the max (where?)

Problems with PCFGs

- The probability model we're using is just based on the rules in the derivation…
  - Doesn't use the words in any real way
  - Doesn't take into account where in the derivation a rule is used

Solution

- Add lexical dependencies to the scheme…
  - Infiltrate the predilections of particular words into the probabilities in the derivation
  - I.e. Condition the rule probabilities on the actual words

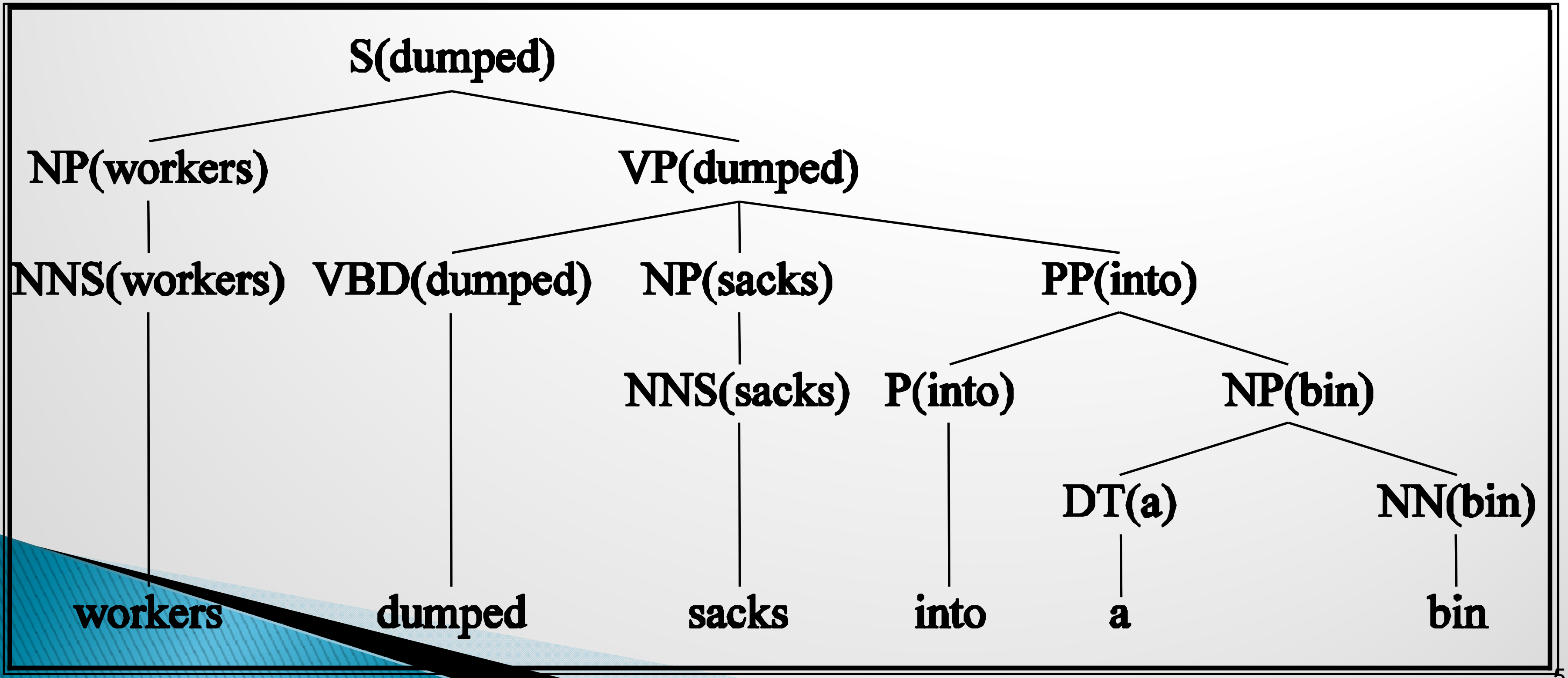
Heads

- To do that we're going to make use of the notion of the head of a phrase
  - The head of an NP is its noun
  - The head of a VP is its verb
  - The head of a PP is its preposition

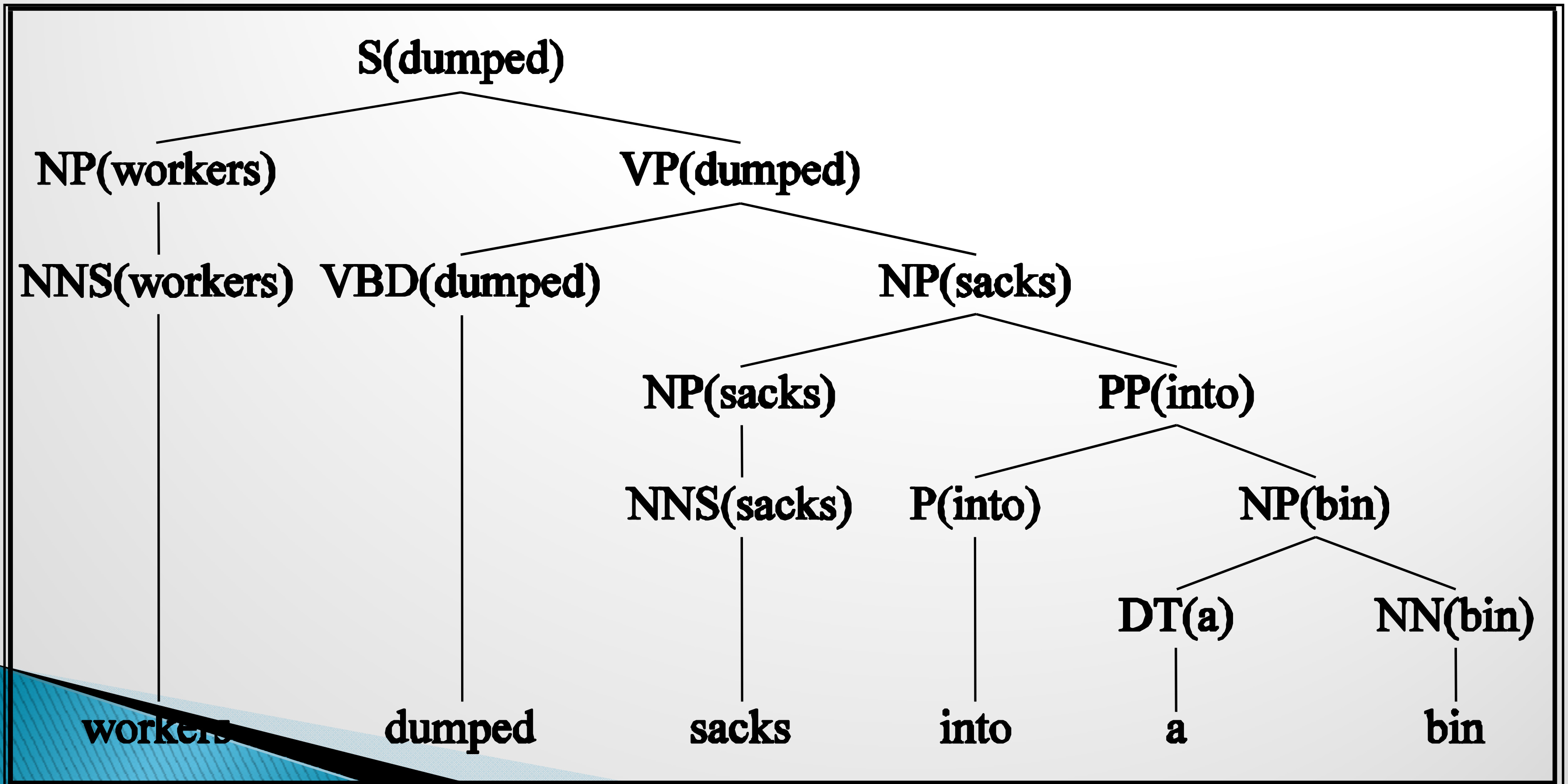
(It's really more complicated than that but this will do.)

Example (right)

Attribute grammar



Example (wrong)



How?

- We used to have
  - VP -> V NP PP P(rule|VP)
    - That's the count of this rule divided by the number of VPs in a treebank
- Now we have
  - VP(dumped)-> V(dumped) NP(sacks)PP(in)
  - P(r|VP ^ dumped is the verb ^ sacks is the head of the NP ^ in is the head of the PP)
  - Not likely to have significant counts in any treebank

Declare Independence

- When stuck, exploit independence and collect the statistics you can…
- We'll focus on capturing two things
  - Verb subcategorization
    - Particular verbs have affinities for particular VPs
  - Objects affinities for their predicates (mostly their mothers and grandmothers)
    - Some objects fit better with some predicates than others

Subcategorization

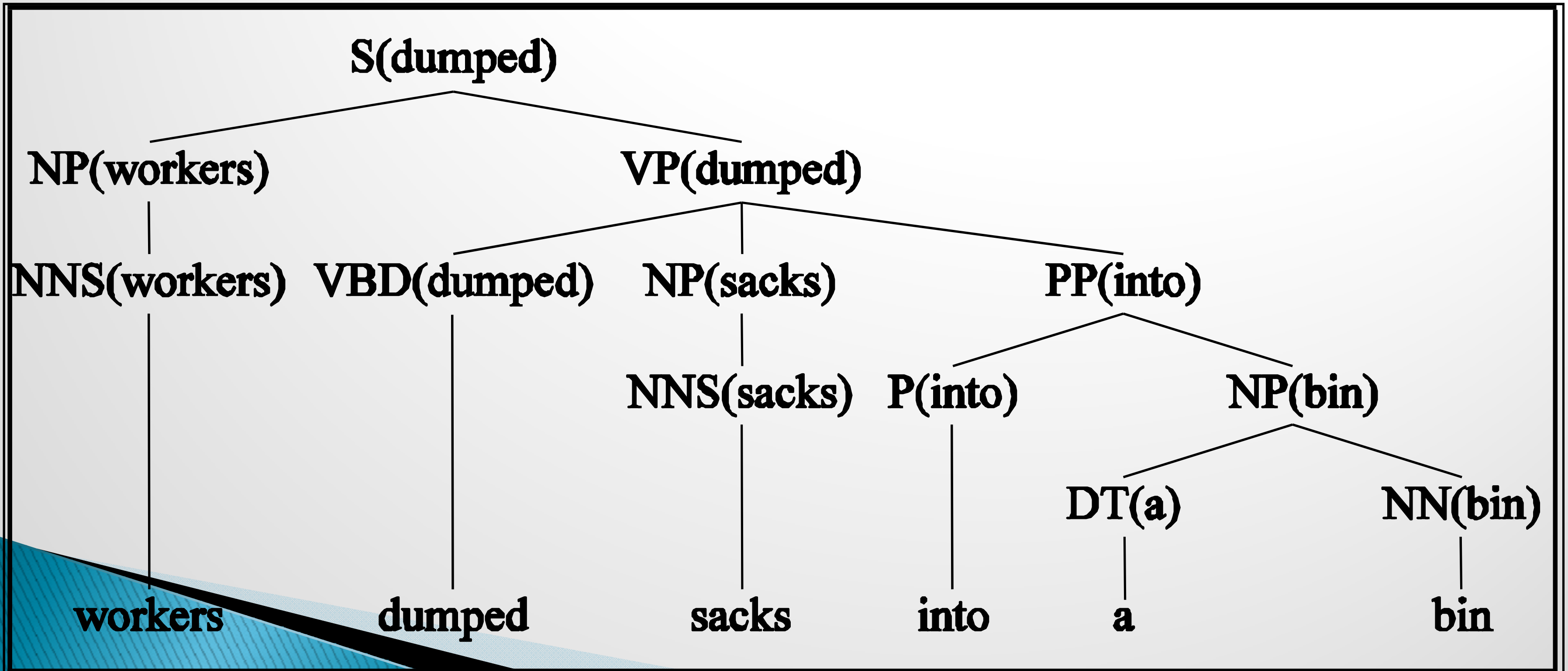
- Condition particular VP rules on their head… so
  r: VP -> V NP PP P(r|VP)
  Becomes
  P(r | VP ^ dumped)

  What's the count?
  How many times was this rule used with (head) dump, divided by the number of VPs that dump appears (as head) in total

  Think of left and right modifiers to the head

Example (right)

Attribute grammar



Probability model

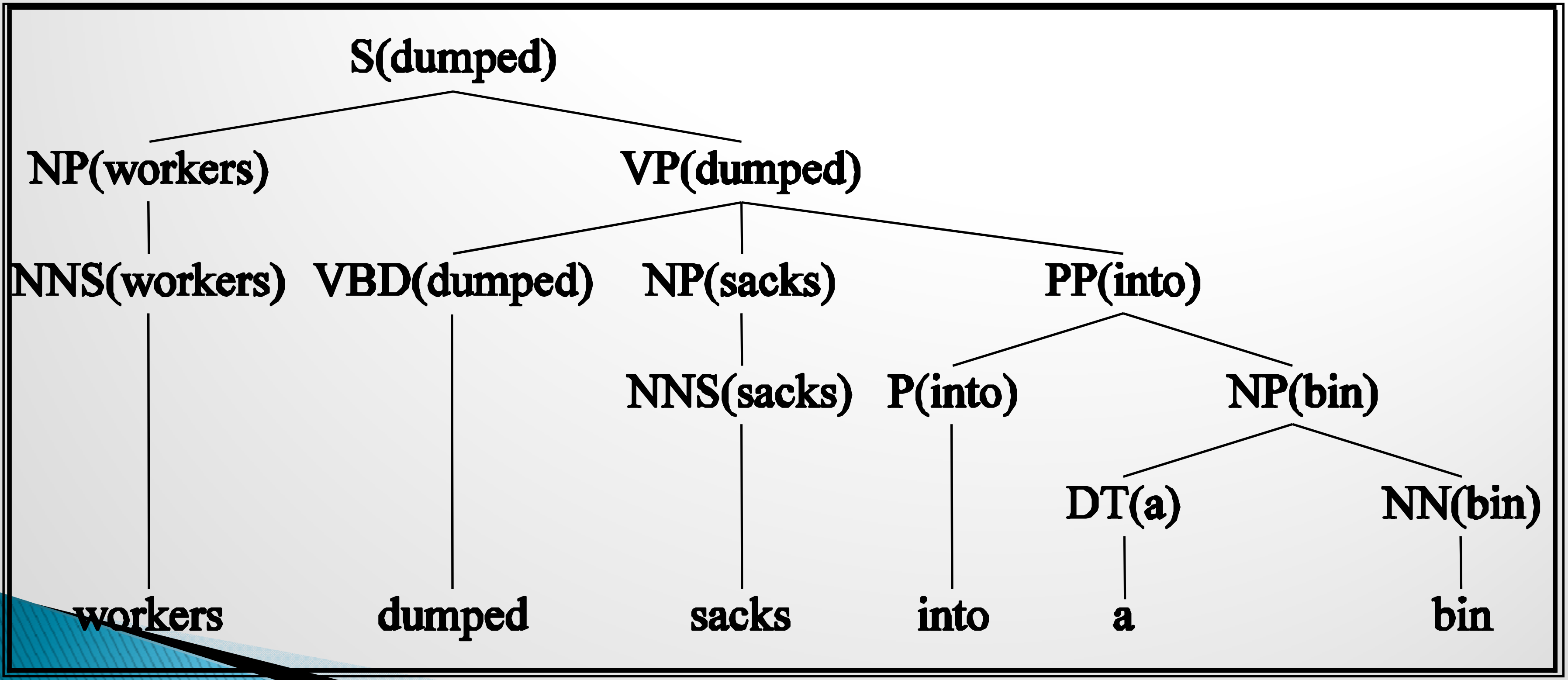
$$P(T,S) = \prod_{n \in T} p(r_n)$$

- P(T,S) = S-> NP VP (.5)*
- VP(dumped) -> V NP PP (.5) (T1)
- VP(ate) -> V NP PP (.03)
- VP(dumped) -> V NP (.2) (T2)

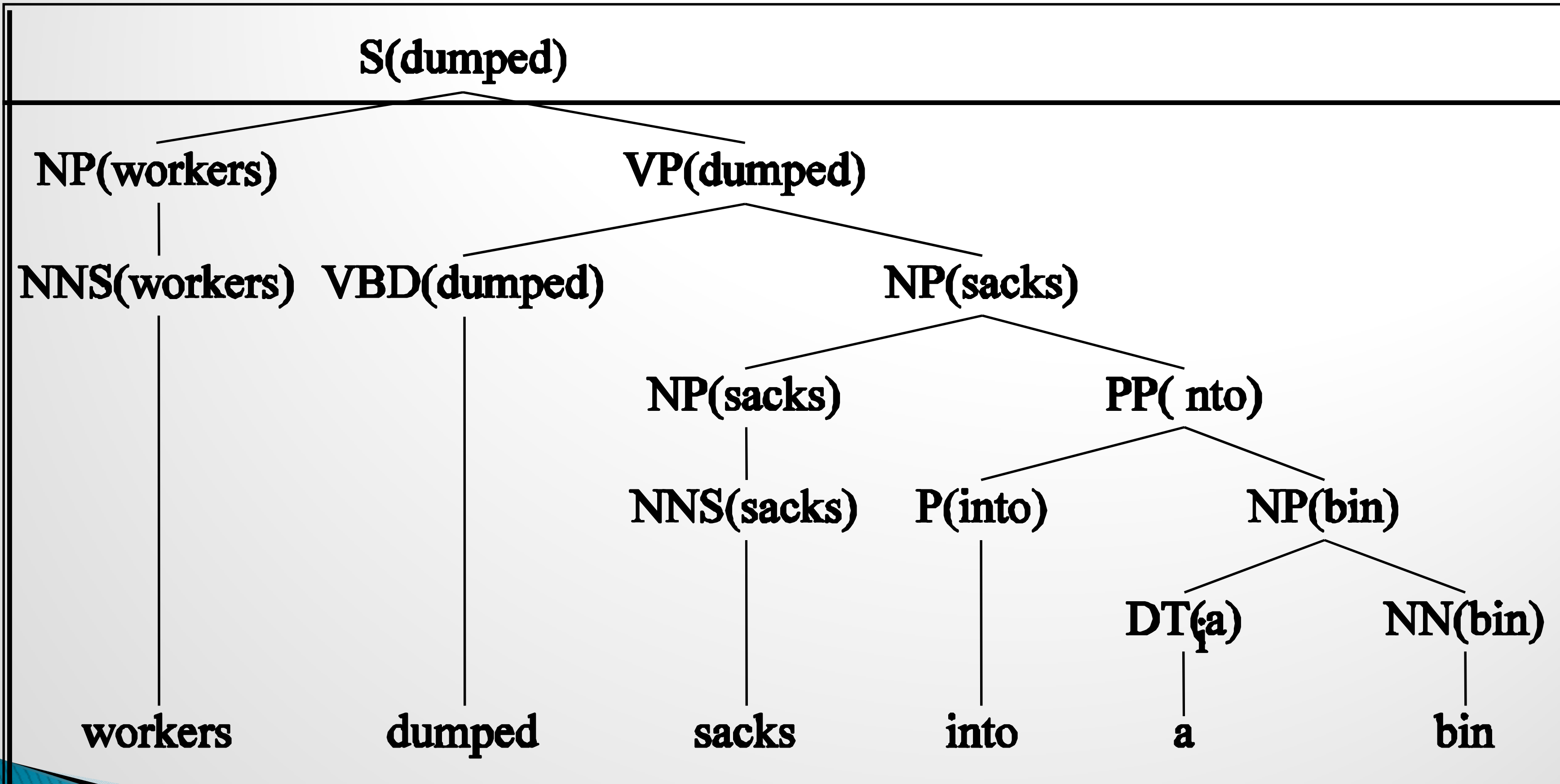
Preferences

- Subcategorization captures the affinity between VP heads (verbs) and the VP rules they go with.
- What about the affinity between VP heads and the heads of the other daughters of the VP
- Back to our examples…

Example (right)



Example (wrong)



Preferences

- The issue here is the attachment of the PP. So the affinities we care about are the ones between dumped and into vs. sacks and into.
- So count the places where dumped is the head of a constituent that has a PP daughter with into as its head and normalize
- Vs. the situation where sacks is a constituent with into as the head of a PP daughter.

Probability model

$$P(T,S) = \prod_{n \in T} p(r_n)$$

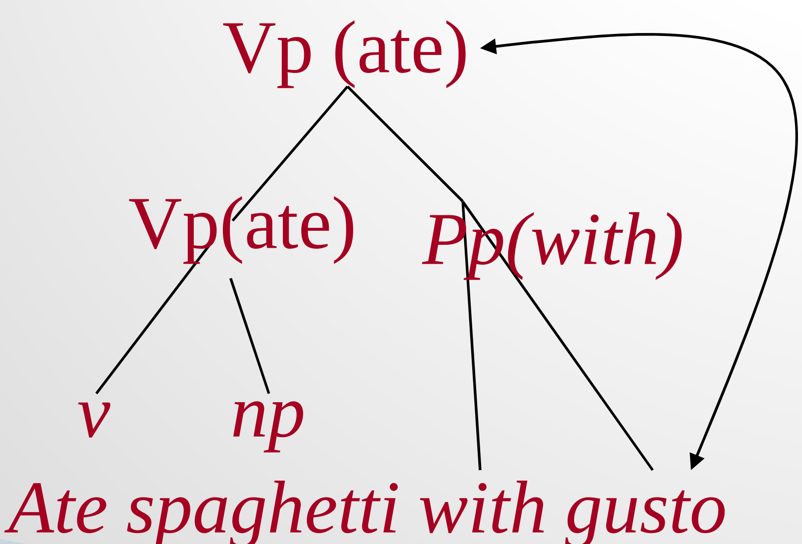
- P(T,S) = S-> NP VP (.5)*
- VP(dumped) -> V NP PP(into) (.7) (T1)
- NOM(sacks) -> NOM PP(into) (.01) (T2)

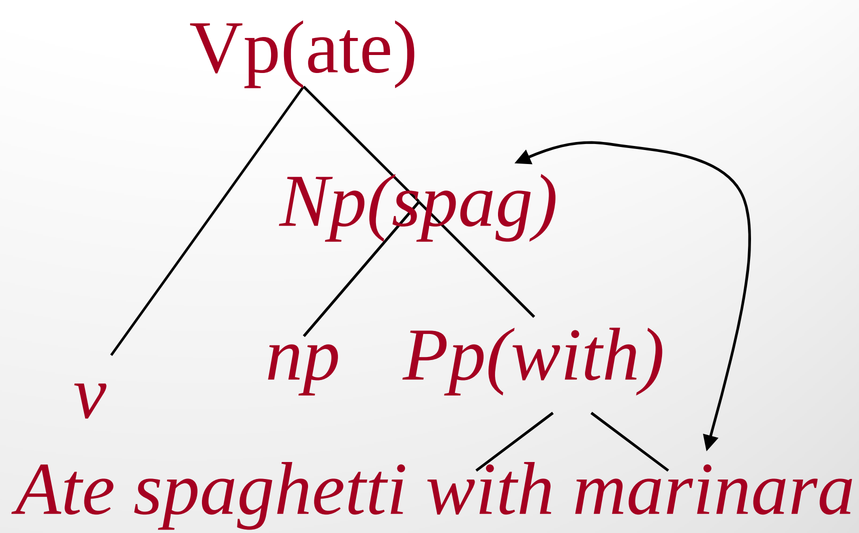
Preferences (2)

- Consider the VPs
  - Ate spaghetti with gusto
  - Ate spaghetti with marinara
- The affinity of gusto for eat is much larger than its affinity for spaghetti
- On the other hand, the affinity of marinara for spaghetti is much higher than its affinity for ate

Preferences (2)

- Note the relationship here is more distant and doesn't involve a headword since gusto and marinara aren't the heads of the PPs.





Summary

- Context-Free Grammars
- Parsing
  - Top Down, Bottom Up Metaphors
  - Dynamic Programming Parsers: CKY. Earley
- Disambiguation:
  - PCFG
  - Probabilistic Augmentations to Parsers
  - Tradeoffs: accuracy vs. data sparcity
  - Treebanks