CYK Review

- Top-down Predict what you expect to see (eg Earley algorithm)
- Bottom-up Start with the words, then incrementally build up parse trees
  - CYK (Cocke-Younger-Kasami) algorithm
  - Well-matched to probabilistic grammars
  - Dynamic programming approach
  - Resolve ambiguities by taking the most probable subtree

CYK Review

- Top-down Predict what you expect to see (eg Earley algorithm)
- Bottom-up Start with the words, then incrementally build up parse trees
  - CYK (Cocke-Younger-Kasami) algorithm
  - Well-matched to probabilistic grammars
  - Dynamic programming approach
  - Resolve ambiguities by taking the most probable subtree

CYK Review

- Top-down Predict what you expect to see (eg Earley algorithm)
- Bottom-up Start with the words, then incrementally build up parse trees
  - CYK (Cocke-Younger-Kasami) algorithm
  - Well-matched to probabilistic grammars
  - Dynamic programming approach
  - Resolve ambiguities by taking the most probable subtree

CYK Review

- Top-down Predict what you expect to see (eg Earley algorithm)
- Bottom-up Start with the words, then incrementally build up parse trees
  - CYK (Cocke-Younger-Kasami) algorithm
  - Well-matched to probabilistic grammars
  - Dynamic programming approach
  - Resolve ambiguities by taking the most probable subtree

CYK Review

- Top-down Predict what you expect to see (eg Earley algorithm)
- Bottom-up Start with the words, then incrementally build up parse trees
    - CYK (Cocke-Younger-Kasami) algorithm
    - Well-matched to probabilistic grammars
    - Dynamic programming approach
    - Resolve ambiguities by taking the most probable subtree

CYK Review

- Top-down Predict what you expect to see (eg Earley algorithm)
- Bottom-up Start with the words, then incrementally build up parse trees
  - CYK (Cocke-Younger-Kasami) algorithm
  - Well-matched to probabilistic grammars
  - Dynamic programming approach
  - Resolve ambiguities by taking the most probable subtree

Chomsky Normal Form (CNF)

- If a CFG is in Chomsky Normal Form, productions are constrained to be of two forms only:
  - **Expand to 2 non-terminals**, eg: A → B C
    with A, B, C all non-terminals
  - **Expand to 1 terminal**, eg: A → a
    where A is a non-terminal and a is a terminal
- Any CFG can be translated to a (weakly) equivalent CFG in CNF
- The CYK algorithm requires a grammar in CNF
- NLTK: use `parse.treetransforms`

Chomsky Normal Form (CNF)

- If a CFG is in Chomsky Normal Form, productions are constrained to be of two forms only:
  - **Expand to 2 non-terminals**, eg: A $\rightarrow$ B C with A, B, C all non-terminals
  - **Expand to 1 terminal**, eg: A $\rightarrow$ a where A is a non-terminal and a is a terminal
- Any CFG can be translated to a (weakly) equivalent CFG in CNF
- The CYK algorithm requires a grammar in CNF
- NLTK: use `parse.treetransforms`

Chomsky Normal Form (CNF)

- If a CFG is in Chomsky Normal Form, productions are constrained to be of two forms only:
  - **Expand to 2 non-terminals**, eg: A → B C
    with A, B, C all non-terminals
  - **Expand to 1 terminal**, eg: A → a
    where A is a non-terminal and a is a terminal
- Any CFG can be translated to a (weakly) equivalent CFG in CNF
- The CYK algorithm requires a grammar in CNF
- NLTK: use `parse.treetransforms`

Chomsky Normal Form (CNF)

- If a CFG is in Chomsky Normal Form, productions are constrained to be of two forms only:
  - **Expand to 2 non-terminals**, eg: A $\rightarrow$ B C
    with A, B, C all non-terminals
  - **Expand to 1 terminal**, eg: A $\rightarrow$ a
    where A is a non-terminal and a is a terminal
- Any CFG can be translated to a (weakly) equivalent CFG in CNF
- The CYK algorithm requires a grammar in CNF
- NLTK: use `parse.treetransforms`

Chomsky Normal Form (CNF)

- If a CFG is in Chomsky Normal Form, productions are constrained to be of two forms only:
  - **Expand to 2 non-terminals**, eg: A $\rightarrow$ B C
    with A, B, C all non-terminals
  - **Expand to 1 terminal**, eg: A $\rightarrow$ a
    where A is a non-terminal and a is a terminal
- Any CFG can be translated to a (weakly) equivalent CFG in CNF
- The CYK algorithm requires a grammar in CNF
- NLTK: use `parse.treetransforms`

Chomsky Normal Form (CNF)

- If a CFG is in Chomsky Normal Form, productions are constrained to be of two forms only:
  - **Expand to 2 non-terminals**, eg: A $\rightarrow$ B C
    with A, B, C all non-terminals
  - **Expand to 1 terminal**, eg: A $\rightarrow$ a
    where A is a non-terminal and a is a terminal
- Any CFG can be translated to a (weakly) equivalent CFG in CNF
- The CYK algorithm requires a grammar in CNF
- NLTK: use `parse.treetransforms`

CYK Example

Alice called Bob from Cardiff

Context-Free Grammar:

S $\rightarrow$ NP VP	NP $\rightarrow$ Alice
NP $\rightarrow$ NP PP	NP $\rightarrow$ Bob
VP $\rightarrow$ V NP	NP $\rightarrow$ Cardiff
VP $\rightarrow$ VP PP	V $\rightarrow$ called
PP $\rightarrow$ P NP	P $\rightarrow$ from

CYK Parsing

				Cardiff
			from	
		Bob		
	called			
Alice				

CYK Parse Chart

CYK Parsing

				Cardiff
			from	
		Bob		
NP	called			
Alice				

Base case

CYK Parsing

				Cardiff
			from	
	V	Bob		
NP	**called**			
Alice				

Base case

CYK Parsing

				Cardiff
		NP	from	
	V	**Bob**		
NP	called			
Alice				

Base case

CYK Parsing

			P	Cardiff
		NP	**from**	
	V	Bob		
NP	called			
Alice				

Base case

CYK Parsing

				NP
			P	**Cardiff**
		NP	from	
	V	Bob		
NP	called			
Alice				

Base case

CYK Parsing

				NP
			P	Cardiff
		NP	from	
X	V	Bob		
NP	called			
Alice				

Recursion

CYK Parsing

				NP
			P	Cardiff
	VP	**NP**	from	
X	**V**	**Bob**		
NP	**called**			
Alice				

Recursion

CYK Parsing

				NP
		X	P	Cardiff
	VP	NP	from	
X	V	Bob		
NP	called			
Alice				

Recursion

CYK Parsing

			PP	**NP**
		X	**P**	**Cardiff**
	VP	NP	**from**	
X	V	Bob		
NP	called			
Alice				

Recursion

CYK Parsing

			PP	NP
		X	P	Cardiff
S	**VP**	**NP**	from	
X	**V**	**Bob**		
NP	**called**			
Alice				

Recursion

CYK Parsing

			PP	NP
	X	X	P	Cardiff
S	VP	NP	from	
X	V	Bob		
NP	called			
Alice				

Recursion

CYK Parsing

		NP	PP	NP
	X	X	P	Cardiff
S	VP	NP	from	
X	V	Bob		
NP	called			
Alice				

Recursion

CYK Parsing

		NP	PP	NP
X	X	X	P	Cardiff
S	VP	NP	from	
X	V	Bob		
NP	called			
Alice				

Recursion

CYK Parsing

	VP$_1$	NP	**PP**	**NP**
X	X	X	**P**	**Cardiff**
S	**VP**	**NP**	**from**	
X	**V**	**Bob**		
NP	**called**			
Alice				

Recursion

CYK Parsing

	VP_2	**NP**	**PP**	**NP**
X	X	X	**P**	**Cardiff**
S	VP	**NP**	**from**	
X	**V**	**Bob**		
NP	**called**			
Alice				

Recursion

CYK Parsing

	VP_1/VP_2	**NP**	**PP**	NP
X	X	X	P	**Cardiff**
S	**VP**	NP	**from**	
X	**V**	**Bob**		
NP	**called**			
Alice				

Recursion

CYK Parsing

S	**VP**	NP	**PP**	**NP**
X	X	X	**P**	**Cardiff**
S	**VP**	**NP**	**from**	
X	**V**	**Bob**		
NP	**called**			
Alice				

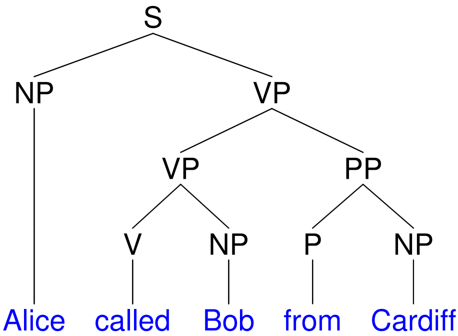
First Parse

CYK Parsing

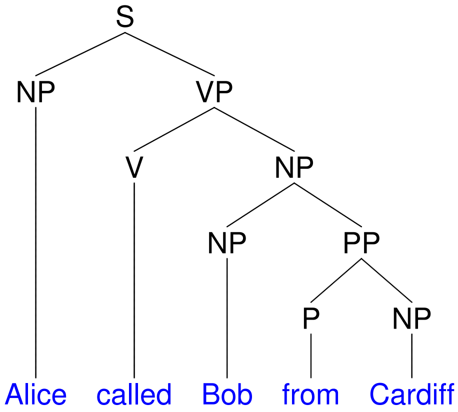
S	**VP**	**NP**	**PP**	**NP**
X	X	X	**P**	**Cardiff**
S	VP	**NP**	**from**	
X	**V**	**Bob**		
NP	**called**			
Alice				

Second Parse

First tree



Second tree



Probabilistic context-free grammars (PCFGs)

- A probabilistic context-free grammar augments each rule in a CFG with a conditional probability *p*
  $A \rightarrow \alpha \quad (p)$
- This probability is the probability that given non-terminal A it will be expanded to the sequence α; written as $P(A \rightarrow \alpha | A)$ or $P(A \rightarrow \alpha)$
- Probability of a parse tree is the product of the rule probabilities used to construct the parse

Probabilistic context-free grammars (PCFGs)

- A probabilistic context-free grammar augments each rule in a CFG with a conditional probability *p*
  A $\rightarrow \alpha$ (*p*)
- This probability is the probability that given non-terminal A it will be expanded to the sequence α; written as
  P(A $\rightarrow \alpha|$A) or *P*(A $\rightarrow \alpha$)
- Probability of a parse tree is the product of the rule probabilities used to construct the parse

Probabilistic context-free grammars (PCFGs)

- A probabilistic context-free grammar augments each rule in a CFG with a conditional probability *p*
  $\text{A} \rightarrow \alpha \quad (p)$
- This probability is the probability that given non-terminal A it will be expanded to the sequence α; written as
  $P(\text{A} \rightarrow \alpha|\text{A})$ or $P(\text{A} \rightarrow \alpha)$
- Probability of a parse tree is the product of the rule probabilities used to construct the parse

Probabilistic parsing

- Consider the rule:

  $$\text{S} \rightarrow \text{NP VP}$$

  Then the probability of S is the product of the rule probability and the probability of each subtree:

  $$P(S) = P(\text{S} \rightarrow \text{NP VP}) \cdot P(\text{NP}) \cdot P(\text{VP})$$

- We are doing bottom-up parsing... so we already know the subtree probabilities $P(\text{NP})$ and $P(\text{VP})$

Probabilistic parsing

- Consider the rule:

  $$S \rightarrow \text{NP VP}$$

  Then the probability of S is the product of the rule probability and the probability of each subtree:

  $$P(S) = P(\text{S} \rightarrow \text{NP VP}) \cdot P(\text{NP}) \cdot P(\text{VP})$$

- We are doing bottom-up parsing... so we already know the subtree probabilities $P(\text{NP})$ and $P(\text{VP})$

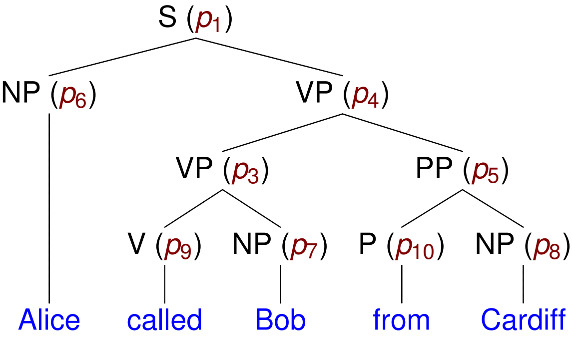
Probabilistic CYK Example

Alice called Bob from Cardiff

Context-Free Grammar:

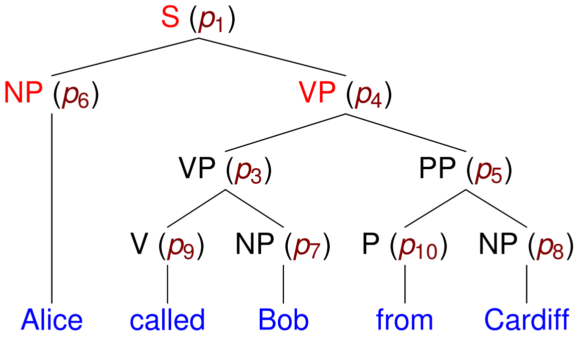
S → NP VP (p_1)	NP → Alice (p_6)
NP → NP PP (p_2)	NP → Bob (p_7)
VP → V NP (p_3)	NP → Cardiff (p_8)
VP → VP PP (p_4)	V → called (p_9)
PP → P NP (p_5)	P → from (p_{10})

First tree (T1)



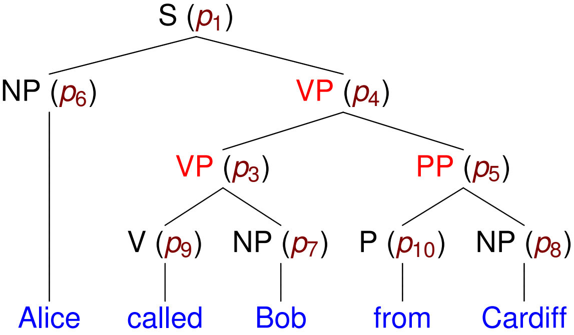
$$P(T1, S) = p_1(p_6)(p_4(p_3(p_9p_7))(p_5(p_{10}p_8)))$$

First tree (T1)



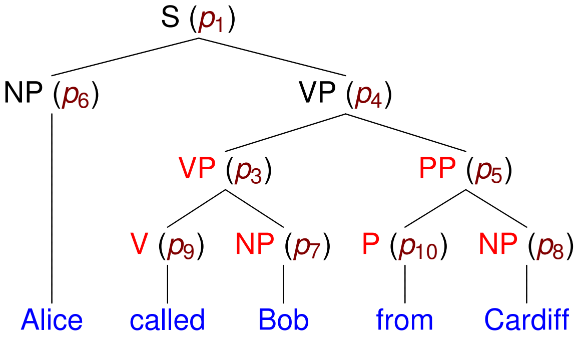
$$P(T1, S) = p_1(p_6)(p_4(p_3(p_9 p_7))(p_5(p_{10} p_8)))$$

First tree (T1)



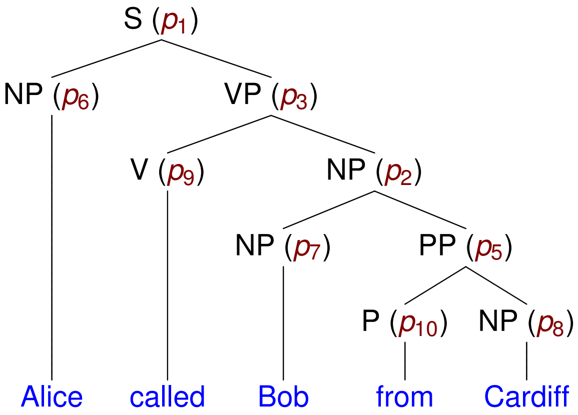
$$P(T1, S) = p_1(p_6)(p_4(p_3(p_9 p_7))(p_5(p_{10} p_8)))$$

First tree (T1)



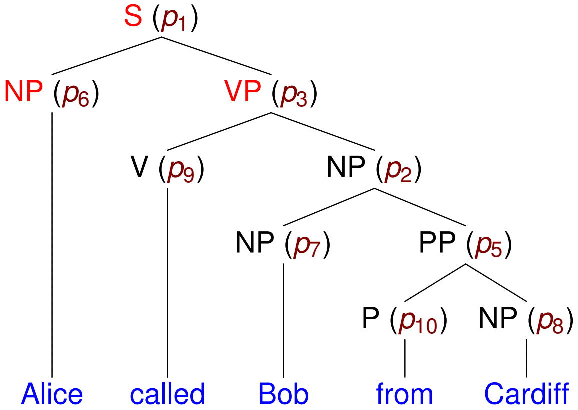
$$P(T1, S) = p_1(p_6)(p_4(p_3(p_9p_7))(p_5(p_{10}p_8)))$$

Second tree (T2)



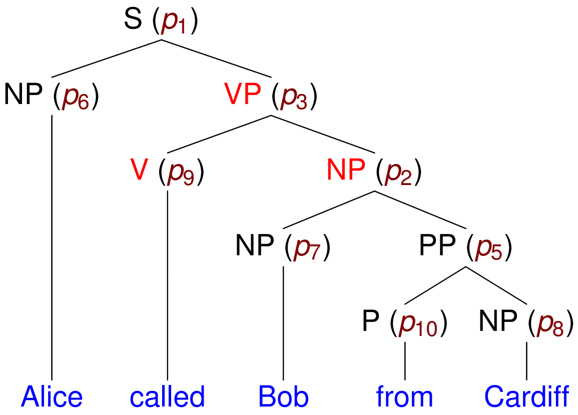
$$P(T2, s) = p_1(p_6)(p_3(p_9)(p_2(p_7)(p_5(p_{10}p_8))))$$

Second tree (T2)



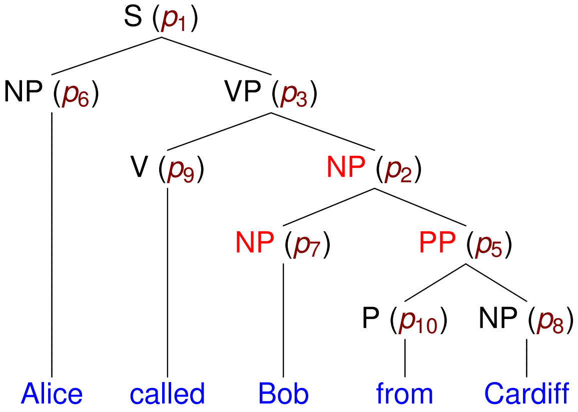
$$P(T2, s) = p_1(p_6)(p_3(p_9)(p_2(p_7)(p_5(p_{10}p_8)))))$$

Second tree (T2)



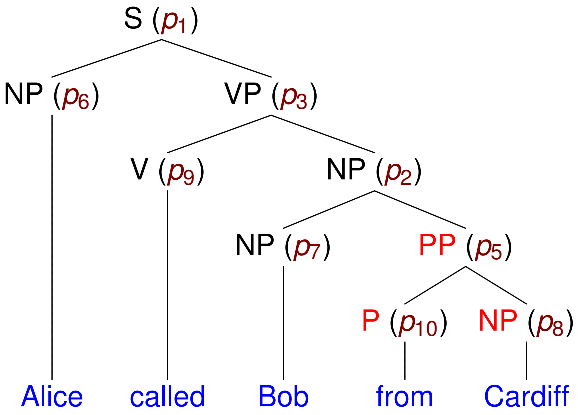
$$P(T2, s) = p_1(p_6)(p_3(p_9)(p_2(p_7)(p_5(p_{10}p_8))))$$

Second tree (T2)



$$P(T2, s) = p_1(p_6)(p_3(p_9)(p_2(p_7)(p_5(p_{10}p_8))))$$

Second tree (T2)



$$P(T2, s) = p_1(p_6)(p_3(p_9)(p_2(p_7)(p_5(p_{10}p_8))))$$

Choosing the tree

Choose tree 1 if $P(T1, S)/P(T2, S) > 1$.

$$\frac{P(T1, S)}{P(T2, S)} = \frac{p_1(p_6)(p_4(p_3(p_9 p_7))(p_5(p_{10} p_9)))}{p_1(p_6)(p_3(p_9)(p_2(p_7)(p_5(p_{10} p_8))))} = \frac{p_4}{p_2}$$

Probabilistic CYK

If $p_4 p_3 p_7 p_9 p_5 p_8 p_{10} < p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$:

S $p_1 p_6 p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$	VP_1 $p_4 p_3 p_7 p_9 p_5 p_8 p_{10}$	NP $p_2 p_7 p_5 p_8 p_{10}$	PP $p_5 p_8 p_{10}$	NP p_8
X	X	X	P p_{10}	Cardiff
S $p_1 p_6 p_3 p_7 p_9$	VP $p_3 p_7 p_9$	NP p_7	from	
X	V p_9	Bob		
NP p_6	called			
Alice				

Probabilistic CYK

If $p_4 p_3 p_7 p_9 p_5 p_8 p_{10} < p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$:

S $p_1 p_6 p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$	VP_1 $p_4 p_3 p_7 p_9 p_5 p_8 p_{10}$	NP $p_2 p_7 p_5 p_8 p_{10}$	PP $p_5 p_8 p_{10}$	NP p_8
X	X	X	P p_{10}	Cardiff
S $p_1 p_6 p_3 p_7 p_9$	VP $p_3 p_7 p_9$	NP p_7	from	
X	V p_9	Bob		
NP p_6	called			
Alice				

Probabilistic CYK

If $p_4 p_3 p_7 p_9 p_5 p_8 p_{10} < p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$:

S $p_1 p_6 p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$	VP_1 $p_4 p_3 p_7 p_9 p_5 p_8 p_{10}$	NP $p_2 p_7 p_5 p_8 p_{10}$	PP $p_5 p_8 p_{10}$	NP p_8
X	X	X	P p_{10}	Cardiff
S $p_1 p_6 p_3 p_7 p_9$	VP $p_3 p_7 p_9$	NP p_7	from	
X	V p_9	Bob		
NP p_6	called			
Alice				

Probabilistic CYK

If $p_4 p_3 p_7 p_9 p_5 p_8 p_{10} < p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$:

S $p_1 p_6 p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$	VP_1 $p_4 p_3 p_7 p_9 p_5 p_8 p_{10}$	NP $p_2 p_7 p_5 p_8 p_{10}$	PP $p_5 p_8 p_{10}$	NP p_8
X	X	X	P p_{10}	Cardiff
S $p_1 p_6 p_3 p_7 p_9$	VP $p_3 p_7 p_9$	NP p_7	from	
X	V p_9	Bob		
NP p_6	called			
Alice				

Probabilistic CYK

If $p_4 p_3 p_7 p_9 p_5 p_8 p_{10} < p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$:

S $p_1 p_6 p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$	VP_2 $p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$	NP $p_2 p_7 p_5 p_8 p_{10}$	PP $p_5 p_8 p_{10}$	NP p_8
X	X	X	P p_{10}	Cardiff
S $p_1 p_6 p_3 p_7 p_9$	VP $p_3 p_7 p_9$	NP p_7	from	
X	V p_9	Bob		
NP p_6	called			
Alice				

Probabilistic CYK

If $p_4 p_3 p_7 p_9 p_5 p_8 p_{10} > p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$:

S $p_1 p_6 p_4 p_3 p_7 p_9 p_5 p_8 p_{10}$	VP_1 $p_4 p_3 p_7 p_9 p_5 p_8 p_{10}$	NP $p_2 p_7 p_5 p_8 p_{10}$	PP $p_5 p_8 p_{10}$	NP p_8
X	X	X	P p_{10}	Cardiff
S $p_1 p_6 p_3 p_7 p_9$	VP $p_3 p_7 p_9$	NP p_7	from	
X	V p_9	Bob		
NP p_6	called			
Alice				

Probabilistic CYK

If $p_4 p_3 p_7 p_9 p_5 p_8 p_{10} < p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$:

S $p_1 p_6 p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$	VP_2 $p_3 p_9 p_2 p_7 p_5 p_8 p_{10}$	NP $p_2 p_7 p_5 p_8 p_{10}$	PP $p_5 p_8 p_{10}$	NP p_8
X	X	X	P p_{10}	Cardiff
S $p_1 p_6 p_3 p_7 p_9$	VP $p_3 p_7 p_9$	NP p_7	from	
X	V p_9	Bob		
NP p_6	called			
Alice				

Estimating PCFG Probabilities

- Treebank—corpus of parsed sentences
- Given a treebank compute the probability of each non-terminal expansion ($A \rightarrow \alpha$) based on the counts $c(A \rightarrow \alpha)$:

$$P(A \rightarrow \alpha | A) = \frac{c(A \rightarrow \alpha)}{\sum_Y c(A \rightarrow Y)} = \frac{c(A \rightarrow \alpha)}{c(A)}$$

Estimating PCFG Probabilities

- Treebank—corpus of parsed sentences
- Given a treebank compute the probability of each non-terminal expansion ($A \to \alpha$) based on the counts $c(A \to \alpha)$:

$$P(A \to \alpha | A) = \frac{c(A \to \alpha)}{\sum_Y c(A \to Y)} = \frac{c(A \to \alpha)}{c(A)}$$

Problems with PCFGs

Structural Rule probabilities are independent of location in the parse tree. For example pronouns are much more likely to be subjects than objects

Lexical PCFGs only capture lexical information in the expansion of pre-terminals.
But, lexical dependencies can often be used to choose the correct parse, eg:
Carol eats chips with ketchup
Carol eats chips with a fork

Problems with PCFGs

Structural Rule probabilities are independent of location in the parse tree. For example pronouns are much more likely to be subjects than objects

Lexical PCFGs only capture lexical information in the expansion of pre-terminals.
But, lexical dependencies can often be used to choose the correct parse, eg:
Carol eats chips with ketchup
Carol eats chips with a fork

Lexical dependence

Data from Penn Treebank:

Rule	*come*	*take*	*think*	*want*
VP $\rightarrow$ V	0.095	0.026	0.046	0.057
VP $\rightarrow$ V NP	0.011	0.321	0.002	0.139
VP $\rightarrow$ V PP	0.345	0.031	0.071	0.003
VP $\rightarrow$ V S	0.022	0.013	0.048	0.708

The rule used to expand VP is strongly dependent on the verb

Lexicalised PCFGs

- Annotated each non-terminal with its *lexical head*
- Each rule has a head child on the left hand side; the headword for a node is then the headword of its head child
- Easy for simple examples (eg the noun is the head of an NP, the verb is the head of a VP); harder in practice
- Various heuristics for finding the head robustly (mainly developed on Penn Treebank)
- A “simple” lexicalised CFG is a basic CFG with a lot more rules (ie each rule is copied for each headword) — but this is impractical!
- Make some independence assumptions and condition the probability of each rule on the head (or nearby heads)

Lexicalised PCFGs

- Annotated each non-terminal with its *lexical head*
- Each rule has a head child on the left hand side; the headword for a node is then the headword of its head child
- Easy for simple examples (eg the noun is the head of an NP, the verb is the head of a VP); harder in practice
- Various heuristics for finding the head robustly (mainly developed on Penn Treebank)
- A “simple” lexicalised CFG is a basic CFG with a lot more rules (ie each rule is copied for each headword) — but this is impractical!
- Make some independence assumptions and condition the probability of each rule on the head (or nearby heads)

Lexicalised PCFGs

- Annotated each non-terminal with its *lexical head*
- Each rule has a head child on the left hand side; the headword for a node is then the headword of its head child
- Easy for simple examples (eg the noun is the head of an NP, the verb is the head of a VP); harder in practice
- Various heuristics for finding the head robustly (mainly developed on Penn Treebank)
- A “simple” lexicalised CFG is a basic CFG with a lot more rules (ie each rule is copied for each headword) — but this is impractical!
- Make some independence assumptions and condition the probability of each rule on the head (or nearby heads)

Lexicalised PCFGs

- Annotated each non-terminal with its *lexical head*
- Each rule has a head child on the left hand side; the headword for a node is then the headword of its head child
- Easy for simple examples (eg the noun is the head of an NP, the verb is the head of a VP); harder in practice
- Various heuristics for finding the head robustly (mainly developed on Penn Treebank)
- A “simple” lexicalised CFG is a basic CFG with a lot more rules (ie each rule is copied for each headword) — but this is impractical!
- Make some independence assumptions and condition the probability of each rule on the head (or nearby heads)

Lexicalised PCFGs

- Annotated each non-terminal with its *lexical head*
- Each rule has a head child on the left hand side; the headword for a node is then the headword of its head child
- Easy for simple examples (eg the noun is the head of an NP, the verb is the head of a VP); harder in practice
- Various heuristics for finding the head robustly (mainly developed on Penn Treebank)
- A “simple” lexicalised CFG is a basic CFG with a lot more rules (ie each rule is copied for each headword) — but this is impractical!
- Make some independence assumptions and condition the probability of each rule on the head (or nearby heads)

Lexicalised PCFGs

- Annotated each non-terminal with its *lexical head*
- Each rule has a head child on the left hand side; the headword for a node is then the headword of its head child
- Easy for simple examples (eg the noun is the head of an NP, the verb is the head of a VP); harder in practice
- Various heuristics for finding the head robustly (mainly developed on Penn Treebank)
- A “simple” lexicalised CFG is a basic CFG with a lot more rules (ie each rule is copied for each headword) — but this is impractical!
- Make some independence assumptions and condition the probability of each rule on the head (or nearby heads)