CKY Algorithm, Chomsky Normal Form

Scott Farrar
CLMA, University of Washington

January 13, 2010

Today's lecture

1. Brief review
2. CKY algorithm
3. Chomsky Normal Form (CNF)
4. Homework2

Parsing strategies

- Name one reason why bottom-up parsing is inefficient?

Parsing strategies

- Name one reason why bottom-up parsing is inefficient?
  The **[search for Spock]** *was successful.*

Parsing strategies

- Name one reason why bottom-up parsing is inefficient? *The* **[search for Spock]** *was successful.*
- And for top-down?

Parsing strategies

- Name one reason why bottom-up parsing is inefficient?
  The **[search for Spock]** *was successful.*
- And for top-down?
  Which would you like?
  That one.

Parsing strategies

- Name one reason why bottom-up parsing is inefficient?
  The **[search for Spock]** *was successful.*
- And for top-down?
  Which would you like?
  That one.
- And what makes naive search so inefficient?

Parsing strategies

- Name one reason why bottom-up parsing is inefficient?
  The **[search for Spock]** *was successful.*
- And for top-down?
  Which would you like?
  That one.
- And what makes naive search so inefficient?
  There's no way to store intermediate solutions.

CKY algorithm

Cocke-Kasami-Younger (CKY) algorithm: a fast bottom-up parsing algorithm that avoids some of the inefficiency associated with purely naive search with the same bottom-up strategy.

CKY algorithm

Cocke-Kasami-Younger (CKY) algorithm: a fast bottom-up parsing algorithm that avoids some of the inefficiency associated with purely naive search with the same bottom-up strategy.

CKY algorithm

Cocke-Kasami-Younger (CKY) algorithm: a fast bottom-up parsing algorithm that avoids some of the inefficiency associated with purely naive search with the same bottom-up strategy.

- Intermediate solutions are stored.

CKY algorithm

Cocke-Kasami-Younger (CKY) algorithm: a fast bottom-up parsing algorithm that avoids some of the inefficiency associated with purely naive search with the same bottom-up strategy.

- Intermediate solutions are stored.
- Only intermediate solutions that contribute to a full parse are further pursued.

CKY algorithm

Cocke-Kasami-Younger (CKY) algorithm: a fast bottom-up parsing algorithm that avoids some of the inefficiency associated with purely naive search with the same bottom-up strategy.

- Intermediate solutions are stored.
- Only intermediate solutions that contribute to a full parse are further pursued.
- The CKY is picky about what type of grammar it accepts.

CKY algorithm

Cocke-Kasami-Younger (CKY) algorithm: a fast bottom-up parsing algorithm that avoids some of the inefficiency associated with purely naive search with the same bottom-up strategy.

- Intermediate solutions are stored.
- Only intermediate solutions that contribute to a full parse are further pursued.
- The CKY is picky about what type of grammar it accepts.
- We require that our grammar be in a special form, known as Chomsky Normal Form (CNF).

CKY algorithm

Cocke-Kasami-Younger (CKY) algorithm: a fast bottom-up parsing algorithm that avoids some of the inefficiency associated with purely naive search with the same bottom-up strategy.

- Intermediate solutions are stored.
- Only intermediate solutions that contribute to a full parse are further pursued.
- The CKY is picky about what type of grammar it accepts.
- We require that our grammar be in a special form, known as Chomsky Normal Form (CNF).
- The rationale is to fill in a chart with the solutions to the subproblems encountered in the bottom-up parsing process.

Dynamic programming

Definition

Dynamic programming: a method of reducing the runtime of algorithms by discovering solutions to subproblems along the way to the solution of the main problem; to optimally plan a multi-stage process

- good for problems with overlapping subproblems
- generally involves the caching of partial results in a table for later retrieval
- many application (outside of NLP)

What are the subproblems for the parsing task?

Well-formed substring table (WFST)

Definition

A well-formed substring table is a data structure containing partial constituency structures. It may be represented as either a chart or a graph.

Well-formed substring table (WFST)

Example

the brown dog

$NP \rightarrow DT\ Nom$, $Nom \rightarrow JJ\ NN$, $DT \rightarrow the$, etc.

the	brown	dog
DT_1		NP_5
	JJ_2	Nom_4
		NN_3

Numbers indicate order in which symbol was enterred into table.

Setting up the CKY algorithm

1. For an input of length=n, create a matrix ($n + 1$ x $n + 1$), indexed from 0 to n.
2. Each cell in the matrix $[i, j]$ is the set of all categories of constituents spanning from position i to j.
3. The algorithm forces you to fill in the table in the most efficient way.
4. Process cells left to right (across columns), bottom to top (backwards across rows).

Well-formed substring table (WFST)

Example

the brown dog

$NP \rightarrow DT\ Nom$, $Nom \rightarrow JJ\ NN$, $DT \rightarrow the$, etc.

the	brown	dog
DT_1		NP_5
	JJ_2	Nom_4
		NN_3

Numbers indicate order in which symbol was enterred into table.

CKY: assumptions

Critical observation: any portion of the input string spanning i to j can be split at k, and structure can then be built using sub-solutions spanning i to k and sub-solutions spanning k to j.

Example

$\bullet_0$ *the* $\bullet_1$ *brown* $\bullet_2$ *dog* $\bullet_3$

- $k = 1$: possible constituents are [0,1] and [1,3]
- $k = 2$: possible constituents are [0,2] and [2,3]

Simple grammar

$S \rightarrow NP\ VBZ$	$DT \rightarrow the$
$S \rightarrow NP\ VP$	$NN \rightarrow chef$
$VP \rightarrow VP\ PP$	$NNS \rightarrow fish$
$VP \rightarrow VBZ\ NP$	$NNS \rightarrow chopsticks$
$VP \rightarrow VBZ\ PP$	$VBP \rightarrow fish$
$VP \rightarrow VBZ\ NNS$	$VBZ \rightarrow eats$
$VP \rightarrow VBZ\ VP$	$IN \rightarrow with$
$VP \rightarrow VBP\ NP$	
$VP \rightarrow VBP\ PP$	
$NP \rightarrow DT\ NN$	
$NP \rightarrow DT\ NNS$	
$PP \rightarrow IN\ NP$	

$\bullet_0$ the $\bullet_1$ chef $\bullet_2$ eats $\bullet_3$ fish $\bullet_4$ with $\bullet_5$ the $\bullet_6$ chopsticks $\bullet_7$

0	1	2	3	4	5	6	7
0							
1							
2							
3							
4							
5							
6							

Build an n+1 x n+1 matrix, where n = number of words in input

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	[0,1]						
1		[1,2]					
2			[2,3]				
3				[3,4]			
4					[4,5]		
5						[5,6]	
6							[6,7]

Illustrate the numbering of cells: [i,j]'s represent spans.

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0							
1		[1,2]					
2							
3							
4							
5							
6							

Notice how the spans (e.g, [1,2]) differ from the word indices (e.g, 'chef', 2).

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]						
1		[1,2]					
2			[2,3]				
3				[3,4]			
4					[4,5]		
5						[5,6]	
6							[6,7]

'the' is labelled DT

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]						
1		NN [1,2]					
2			[2,3]				
3				[3,4]			
4					[4,5]		
5						[5,6]	
6							[6,7]

'chef' is labelled NN

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]					
1		NN [1,2]					
2			[2,3]				
3				[3,4]			
4					[4,5]		
5						[5,6]	
6							[6,7]

Found an NP: [0,1], [1,2]

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]					
1		NN [1,2]					
2			VBZ [2,3]				
3				[3,4]			
4					[4,5]		
5						[5,6]	
6							[6,7]

'eats' is labelled VBZ

	the	chef	eats	fish	with	the	chopsticks
0	1	2	3	4	5	6	7
0	DT [0,1]	NP [0,2]	S [0,3]				
1		NN [1,2]					
2			VBZ [2,3]				
3				[3,4]			
4					[4,5]		
5						[5,6]	
6							[6,7]

Found an S: [0,2],[2,3]

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]				
1		NN [1,2]					
2			VBZ [2,3]				
3				NNS [3,4]			
4					[4,5]		
5						[5,6]	
6							[6,7]

‘fish’ is labelled NNS

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]				
1		NN [1,2]					
2			VBZ [2,3]				
3				NNS,VBP [3,4]			
4					[4,5]		
5						[5,6]	
6							[6,7]

'fish' is labelled VBP

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]				
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			
3				NNS,VBP [3,4]			
4					[4,5]		
5						[5,6]	
6							[6,7]

Found a VP: [2,3], [3,4]

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]	S [0,4]			
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			
3				NNS,VBP [3,4]			
4					[4,5]		
5						[5,6]	
6							[6,7]

Found an S: [0,2],[2,4]

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]	S [0,4]			
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			
3				NNS,VBP [3,4]			
4					IN [4,5]		
5						[5,6]	
6							[6,7]

'with' is labelled IN

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]	S [0,4]			
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			
3				NNS,VBP [3,4]			
4					IN [4,5]		
5						DT [5,6]	
6							[6,7]

'the' is labelled DT

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]	S [0,4]			
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			
3				NNS,VBP [3,4]			
4					IN [4,5]		
5						DT [5,6]	
6							NNS [6,7]

'chopsticks' is labelled NNS

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]	S [0,4]			
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			
3				NNS,VBP [3,4]			
4					IN [4,5]		
5						DT [5,6]	NP [5,7]
6							NNS [6,7]

Found an NP: [5,6], [6,7]

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]	S [0,4]			
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			
3				NNS,VBP [3,4]			
4					IN [4,5]		PP [4,7]
5						DT [5,6]	NP [5,7]
6							NNS [6,7]

Found a PP: [4,5],[5,7]

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]	S [0,4]			
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			
3				NNS,VBP [3,4]			VP [3,7]
4					IN [4,5]		PP [4,7]
5						DT [5,6]	NP [5,7]
6							NNS [6,7]

Found a VP: [3,4], [4,7]

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]	S [0,4]			
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			VP [2,7]
3				NNS,VBP [3,4]			VP [3,7]
4					IN [4,5]		PP [4,7]
5						DT [5,6]	NP [5,7]
6							NNS [6,7]

Found a VP: [2,3],[3,7]

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]	S [0,4]			
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			VP_1, VP_2 [2,7]
3				NNS,VBP [3,4]			VP [3,7]
4					IN [4,5]		PP [4,7]
5						DT [5,6]	NP [5,7]
6							NNS [6,7]

Found another VP: [2,4],[4,7]

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]	S [0,4]			S [0,7]
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			VP_1, VP_2 [2,7]
3				NNS,VBP [3,4]			VP [3,7]
4					IN [4,5]		PP [4,7]
5						DT [5,6]	NP [5,7]
6							NNS [6,7]

Found an S node: [0,2] [2,7]

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]	S [0,4]			S_1, S_2 [0,7]
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			VP_1, VP_2 [2,7]
3				NNS,VBP [3,4]			VP [3,7]
4					IN [4,5]		PP [4,7]
5						DT [5,6]	NP [5,7]
6							NNS [6,7]

Found a second S node: also [0,2] [2,7]

0	the 1	chef 2	eats 3	fish 4	with 5	the 6	chopsticks 7
0	DT [0,1]	NP [0,2]	S [0,3]	S [0,4]			S_1, S_2 [0,7]
1		NN [1,2]					
2			VBZ [2,3]	VP [2,4]			VP_1, VP_2 [2,7]
3				NNS,VBP [3,4]			VP [3,7]
4					IN [4,5]		PP [4,7]
5						DT [5,6]	NP [5,7]
6							NNS [6,7]

Found a second S node: also [0,2] [2,7]

Recognition algorithm returns **True** when a root node is found in [0,n]

The CKY Algorithm (recognition)

function CKY-Parse (*words*, *grammar*) **returns** *table*

for j $\leftarrow$ 1 **to** length(*words*) **do**: `(loop over columns)`

table[j-1,j] $\leftarrow \{A | A \rightarrow$ *words*[j] $\in$ *grammar*$\}$ `(add POS)`

for i $\leftarrow$ j-2 **downto** 0 **do**: `(loop over rows, backwards)`

for k $\leftarrow$ i+1 to j-1 **do**: `(loop over contents of cell)`

table[i,j] $\leftarrow$ *table*[i,j] $\cup$

$\{A | A \rightarrow B \;\; C \in$ *grammar*,

B $\in$ *table*[i,k]

C $\in$ *table*[k,j] $\}$

CKY recognition vs. parsing

- Returning the full parse requires storing more in a cell than just a node label.
- We also require back-pointers to constituents of that node.
- We could also store whole trees, but less space efficient.
- For parsing, we must add an extra step to the algorithm:

CKY recognition vs. parsing

- Returning the full parse requires storing more in a cell than just a node label.
- We also require back-pointers to constituents of that node.
- We could also store whole trees, but less space efficient.
- For parsing, we must add an extra step to the algorithm: follow pointers and return the parse

The CKY Algorithm (parsing)

function CKY-Parse (*words*, *grammar*) **returns** *parses*

for j ← 1 **to** length(*words*) **do**: `(loop over columns)`

table[j-1,j] ← **for all** $\{A | A \rightarrow words[j] \in grammar\}$ `(add all POS)`

for i ← j-2 **downto** 0 **do**: `(loop over rows, backwards)`

for k ← i+1 to j-1 **do**: `(loop over contents of cell)`

for all $\{A | A \rightarrow B\ C\}$: `(all productions)`

back[i,j,A] ← { k,B,C } `(add back pointer)`

return buildtree(*back*[1, length(*words*,S]), *table*[1,LENGTH(*words*),S] `(follow back pointer)`

Issues with CKY

Efficiency

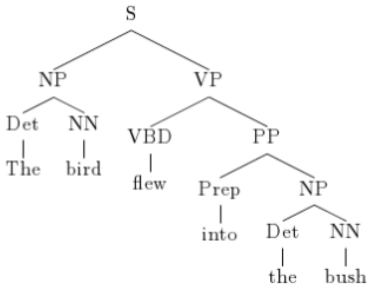
- The CKY can be performed in cubic time: $O(n^3)$, where n=number of words in sentence.
- The complexity of the inner most loop is bounded by the square of the number of non-terminals.
- The more rules, the less efficient; but this increases at a constant rate $L = r^2$ where r is the number of non-terminals.

Issues with CKY

Grammar requirements

- The basic algoritm requires a binary grammar, in fact a grammar in Chomsky Normal Form.
- Basic algorithm can be extended to account for arbitrary CFGs.
- However, transforming a grammar into a CNF grammar is easier and more efficient than parsing with an arbitrary grammar.
- Later, we'll look at the Earley Algorithm for parsing arbitrary CFGs.

Binary tree



Chomsky Normal Form grammar

Definition

CNF grammar: a context-free grammar where the RHS of each production rule is restricted to be either two non-terminals or one terminal, and no empty productions are allowed.

There can be:

- no mixed rules ($NP \rightarrow the\ NN$)
- no unit productions ($NP \rightarrow NNP$), except for $NN \rightarrow dog$
- no right hand sides of more than two non-terminals ($VP \rightarrow VBZ \quad NP \quad PP$).

Grammar equivalence

Any CFG can be converted to a weakly equivalent grammar in CNF.

Grammar equivalence

Any CFG can be converted to a weakly equivalent grammar in CNF.

Definition

Weak equivalence: Two grammars are weakly equivalent if they generate the same set of strings (sentences). Transforming a grammar to CNF results in a new grammar that is weakly equivalent.

Grammar equivalence

Any CFG can be converted to a weakly equivalent grammar in CNF.

Definition

Weak equivalence: Two grammars are weakly equivalent if they generate the same set of strings (sentences). Transforming a grammar to CNF results in a new grammar that is weakly equivalent.

Definition

Strong equivalence: Two grammars are strongly equivalent if they generate the same set of strings AND the same structures over those strings. If only the variable names are diff. then the grammar are said to be *isomorphic*.

Symbol naming conventions

- Use new symbols (binarization): $X1$, $X2$, ..., $Y3$
  $S \rightarrow NP\ \ VP\ \ PUNC$ becomes:
  $S \rightarrow NP\ \ X1$, $X1 \rightarrow VP\ \ PUNC$
- Delete a symbol (unary collapsing):
  $SBAR \rightarrow S$, $S \rightarrow NP\ \ VP$ becomes
  $SBAR \rightarrow NP\ \ VP$

CNF conversion algorithm

1. Removing unit-productions (unary collapsing):
   - while there is a unit-production $A \longrightarrow B$,
   - Remove $A \longrightarrow B$.
   - **foreach** $B \longrightarrow u$, add $A \longrightarrow u$.

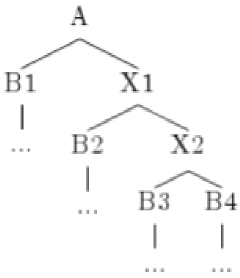
CNF conversion algorithm

1. Removing unit-productions (unary collapsing):
    - while there is a unit-production $A \longrightarrow B$,
    - Remove $A \longrightarrow B$.
    - **foreach** $B \longrightarrow u$, add $A \longrightarrow u$.
2. Remove terminals from mixed rules
    - **foreach** production $A \longrightarrow B_1\ B_2 ... B_k$, containing a terminal x
    - Add new non-terminal/production $X1 \longrightarrow x$ (unless it has already been added)
    - Replace every $B_i = x$ with $X1$

CNF conversion algorithm

1. Removing unit-productions (unary collapsing):
   - while there is a unit-production $A \rightarrow B$,
   - Remove $A \rightarrow B$.
   - **foreach** $B \rightarrow u$, add $A \rightarrow u$.
2. Remove terminals from mixed rules
   - **foreach** production $A \rightarrow B_1\ B_2 \ldots B_k$, containing a terminal x
   - Add new non-terminal/production $X1 \rightarrow x$ (unless it has already been added)
   - Replace every $B_i = x$ with $X1$
3. Remove rules with more than two nonterminals on the RHS (binarization)
   - **foreach** rule p of form $A \rightarrow B_1\ B_2 \ldots B_k$
   - replace p with $A \rightarrow B_1\ X1$, $X1 \rightarrow B_2\ X2$, $X2 \rightarrow B_3\ X3$, ..., $X(k-2) \rightarrow B_{k-1}\ B_k$ (Xi's are new variables.)

Binarization



A sample CFG

$S \rightarrow NP \quad VP \quad PUNC$

A sample CFG

$S \rightarrow NP \quad VP \quad PUNC$ (non-binary)

A sample CFG

$$S \rightarrow NP \quad VP \quad PUNC \qquad \text{(non-binary)}$$
$$S \rightarrow S \quad and \quad S$$

A sample CFG

$$
\begin{aligned}
S \rightarrow NP \quad VP \quad PUNC & \qquad \text{(non-binary)} \\
S \rightarrow S \quad and \quad S & \qquad \text{(mixed)}
\end{aligned}
$$

A sample CFG

$$\begin{aligned} S &\rightarrow NP \quad VP \quad PUNC && \text{(non-binary)} \\ S &\rightarrow S \quad and \quad S && \text{(mixed)} \\ NP &\rightarrow DT \quad NP \end{aligned}$$

A sample CFG

$$S \rightarrow NP \quad VP \quad PUNC \qquad \text{(non-binary)}$$
$$S \rightarrow S \quad and \quad S \qquad \text{(mixed)}$$
$$NP \rightarrow DT \quad NP \qquad \text{(OK)}$$

A sample CFG

$$
\begin{aligned}
S &\rightarrow NP \;\; VP \;\; PUNC && \text{(non-binary)} \\
S &\rightarrow S \;\; and \;\; S && \text{(mixed)} \\
NP &\rightarrow DT \;\; NP && \text{(OK)} \\
NP &\rightarrow NN &&
\end{aligned}
$$

A sample CFG

$$
\begin{aligned}
S \rightarrow NP \;\; VP \;\; PUNC \quad & \text{(non-binary)} \\
S \rightarrow S \;\; and \;\; S \quad & \text{(mixed)} \\
NP \rightarrow DT \;\; NP \quad & \text{(OK)} \\
NP \rightarrow NN \quad & \text{(unit production)}
\end{aligned}
$$

A sample CFG

$S \rightarrow NP \;\; VP \;\; PUNC$	(non-binary)
$S \rightarrow S \;\; and \;\; S$	(mixed)
$NP \rightarrow DT \;\; NP$	(OK)
$NP \rightarrow NN$	(unit production)
$NN \rightarrow dog$	

A sample CFG

$S \rightarrow NP \quad VP \quad PUNC$	(non-binary)
$S \rightarrow S \quad and \quad S$	(mixed)
$NP \rightarrow DT \quad NP$	(OK)
$NP \rightarrow NN$	(unit production)
$NN \rightarrow dog$	(OK)

A sample CFG

$$
\begin{aligned}
S &\rightarrow NP \;\; VP \;\; PUNC && \text{(non-binary)} \\
S &\rightarrow S \;\; and \;\; S && \text{(mixed)} \\
NP &\rightarrow DT \;\; NP && \text{(OK)} \\
NP &\rightarrow NN && \text{(unit production)} \\
NN &\rightarrow dog && \text{(OK)} \\
NN &\rightarrow cat &&
\end{aligned}
$$

A sample CFG

$$
\begin{aligned}
S &\rightarrow NP \;\; VP \;\; PUNC && \text{(non-binary)} \\
S &\rightarrow S \;\; and \;\; S && \text{(mixed)} \\
NP &\rightarrow DT \;\; NP && \text{(OK)} \\
NP &\rightarrow NN && \text{(unit production)} \\
NN &\rightarrow dog && \text{(OK)} \\
NN &\rightarrow cat && \text{(OK)}
\end{aligned}
$$

A sample CFG

$S \rightarrow NP \;\; VP \;\; PUNC$	(non-binary)
$S \rightarrow S \;\; and \;\; S$	(mixed)
$NP \rightarrow DT \;\; NP$	(OK)
$NP \rightarrow NN$	(unit production)
$NN \rightarrow dog$	(OK)
$NN \rightarrow cat$	(OK)
$VP \rightarrow VBZ \;\; NP$	

A sample CFG

$S \rightarrow NP\ \ VP\ \ PUNC$	(non-binary)
$S \rightarrow S\ \ and\ \ S$	(mixed)
$NP \rightarrow DT\ \ NP$	(OK)
$NP \rightarrow NN$	(unit production)
$NN \rightarrow dog$	(OK)
$NN \rightarrow cat$	(OK)
$VP \rightarrow VBZ\ \ NP$	(OK)

A sample CFG

$S \rightarrow NP \;\; VP \;\; PUNC$	(non-binary)
$S \rightarrow S \;\; and \;\; S$	(mixed)
$NP \rightarrow DT \;\; NP$	(OK)
$NP \rightarrow NN$	(unit production)
$NN \rightarrow dog$	(OK)
$NN \rightarrow cat$	(OK)
$VP \rightarrow VBZ \;\; NP$	(OK)
$VP \rightarrow VBZ$	

A sample CFG

$S \rightarrow NP\ \ VP\ \ PUNC$	(non-binary)
$S \rightarrow S\ \ and\ \ S$	(mixed)
$NP \rightarrow DT\ \ NP$	(OK)
$NP \rightarrow NN$	(unit production)
$NN \rightarrow dog$	(OK)
$NN \rightarrow cat$	(OK)
$VP \rightarrow VBZ\ \ NP$	(OK)
$VP \rightarrow VBZ$	(unit production)

A sample CFG

$S \rightarrow NP\ \ VP\ \ PUNC$	(non-binary)
$S \rightarrow S\ \ and\ \ S$	(mixed)
$NP \rightarrow DT\ \ NP$	(OK)
$NP \rightarrow NN$	(unit production)
$NN \rightarrow dog$	(OK)
$NN \rightarrow cat$	(OK)
$VP \rightarrow VBZ\ \ NP$	(OK)
$VP \rightarrow VBZ$	(unit production)
$VBZ \rightarrow sleeps$	

A sample CFG

$S \rightarrow NP \;\; VP \;\; PUNC$	(non-binary)
$S \rightarrow S \;\; and \;\; S$	(mixed)
$NP \rightarrow DT \;\; NP$	(OK)
$NP \rightarrow NN$	(unit production)
$NN \rightarrow dog$	(OK)
$NN \rightarrow cat$	(OK)
$VP \rightarrow VBZ \;\; NP$	(OK)
$VP \rightarrow VBZ$	(unit production)
$VBZ \rightarrow sleeps$	(OK)

A sample CFG

$S \rightarrow NP \ \ VP \ \ PUNC$	(non-binary)
$S \rightarrow S \ \ and \ \ S$	(mixed)
$NP \rightarrow DT \ \ NP$	(OK)
$NP \rightarrow NN$	(unit production)
$NN \rightarrow dog$	(OK)
$NN \rightarrow cat$	(OK)
$VP \rightarrow VBZ \ \ NP$	(OK)
$VP \rightarrow VBZ$	(unit production)
$VBZ \rightarrow sleeps$	(OK)
$VBZ \rightarrow eats$	

A sample CFG

$S \rightarrow NP \;\; VP \;\; PUNC$	(non-binary)
$S \rightarrow S \;\; and \;\; S$	(mixed)
$NP \rightarrow DT \;\; NP$	(OK)
$NP \rightarrow NN$	(unit production)
$NN \rightarrow dog$	(OK)
$NN \rightarrow cat$	(OK)
$VP \rightarrow VBZ \;\; NP$	(OK)
$VP \rightarrow VBZ$	(unit production)
$VBZ \rightarrow sleeps$	(OK)
$VBZ \rightarrow eats$	(OK)

A sample CFG

$S \rightarrow NP \quad VP \quad PUNC$	(non-binary)
$S \rightarrow S \quad and \quad S$	(mixed)
$NP \rightarrow DT \quad NP$	(OK)
$NP \rightarrow NN$	(unit production)
$NN \rightarrow dog$	(OK)
$NN \rightarrow cat$	(OK)
$VP \rightarrow VBZ \quad NP$	(OK)
$VP \rightarrow VBZ$	(unit production)
$VBZ \rightarrow sleeps$	(OK)
$VBZ \rightarrow eats$	(OK)
$DT \rightarrow the$	

A sample CFG

$$
\begin{array}{rl}
S \longrightarrow NP \;\; VP \;\; PUNC & \text{(non-binary)} \\
S \longrightarrow S \;\; and \;\; S & \text{(mixed)} \\
NP \longrightarrow DT \;\; NP & \text{(OK)} \\
NP \longrightarrow NN & \text{(unit production)} \\
NN \longrightarrow dog & \text{(OK)} \\
NN \longrightarrow cat & \text{(OK)} \\
VP \longrightarrow VBZ \;\; NP & \text{(OK)} \\
VP \longrightarrow VBZ & \text{(unit production)} \\
VBZ \longrightarrow sleeps & \text{(OK)} \\
VBZ \longrightarrow eats & \text{(OK)} \\
DT \longrightarrow the & \text{(OK)}
\end{array}
$$

Conversion of CFG to CNF: Step 1

Non-CNF grammar	CNF grammar	Action

Conversion of CFG to CNF: Step 1

Non-CNF grammar	CNF grammar	Action
$NP \rightarrow NN$		

Conversion of CFG to CNF: Step 1

Non-CNF grammar	CNF grammar	Action
$NP \rightarrow NN$		
$NN \rightarrow dog$		

Conversion of CFG to CNF: Step 1

Non-CNF grammar	CNF grammar	Action
$NP \rightarrow NN$		
$NN \rightarrow dog$		
$NN \rightarrow cat$		

Conversion of CFG to CNF: Step 1

Non-CNF grammar	CNF grammar	Action
$NP \rightarrow NN$		
$NN \rightarrow dog$		
$NN \rightarrow cat$		
	$NP \rightarrow dog$	(collapse rule)

Conversion of CFG to CNF: Step 1

Non-CNF grammar	CNF grammar	Action
$NP \rightarrow NN$		
$NN \rightarrow dog$		
$NN \rightarrow cat$		
	$NP \rightarrow dog$	(collapse rule)
	$NP \rightarrow cat$	(collapse rule)

Conversion of CFG to CNF: Step 1

Non-CNF grammar	CNF grammar	Action
$NP \rightarrow NN$		
$NN \rightarrow dog$		
$NN \rightarrow cat$		
	$NP \rightarrow dog$	(collapse rule)
	$NP \rightarrow cat$	(collapse rule)
$VP \rightarrow VBZ$		

Conversion of CFG to CNF: Step 1

Non-CNF grammar	CNF grammar	Action
$NP \rightarrow NN$		
$NN \rightarrow dog$		
$NN \rightarrow cat$		
	$NP \rightarrow dog$	(collapse rule)
	$NP \rightarrow cat$	(collapse rule)
$VP \rightarrow VBZ$		
$VBZ \rightarrow sleeps$		

Conversion of CFG to CNF: Step 1

Non-CNF grammar	CNF grammar	Action
$NP \longrightarrow NN$		
$NN \longrightarrow dog$		
$NN \longrightarrow cat$		
	$NP \longrightarrow dog$	(collapse rule)
	$NP \longrightarrow cat$	(collapse rule)
$VP \longrightarrow VBZ$		
$VBZ \longrightarrow sleeps$		
$VBZ \longrightarrow eats$		

Conversion of CFG to CNF: Step 1

Non-CNF grammar	CNF grammar	Action
$NP \longrightarrow NN$		
$NN \longrightarrow dog$		
$NN \longrightarrow cat$		
	$NP \longrightarrow dog$	(collapse rule)
	$NP \longrightarrow cat$	(collapse rule)
$VP \longrightarrow VBZ$		
$VBZ \longrightarrow sleeps$		
$VBZ \longrightarrow eats$		
	$VP \longrightarrow sleeps$	(collapse rule)
	$VP \longrightarrow eats$	(collapse rule)

Conversion of CFG to CNF: Step 2

Non-CNF grammar	CNF grammar	Action

Conversion of CFG to CNF: Step 2

Non-CNF grammar	CNF grammar	Action
$S \rightarrow S \;\; and \;\; S$		

Conversion of CFG to CNF: Step 2

Non-CNF grammar	CNF grammar	Action
$S \rightarrow S$ *and* S		

Conversion of CFG to CNF: Step 2

Non-CNF grammar	CNF grammar	Action
$S \rightarrow S \;\; and \;\; S$		
	$S \rightarrow S \;\; X1$	(new symbol)

Conversion of CFG to CNF: Step 2

Non-CNF grammar	CNF grammar	Action
$S \rightarrow S \;\; and \;\; S$		
	$S \rightarrow S \;\; X1$	(new symbol)
	$X1 \rightarrow X2 \;\; S$	(new symbol)

Conversion of CFG to CNF: Step 2

Non-CNF grammar	CNF grammar	Action
$S \rightarrow S \;\; and \;\; S$		
	$S \rightarrow S \;\; X1$	(new symbol)
	$X1 \rightarrow X2 \;\; S$	(new symbol)
	$X2 \rightarrow and$	

Conversion of CFG to CNF: Step 3

Non-CNF grammar	CNF grammar	Action

Conversion of CFG to CNF: Step 3

Non-CNF grammar	CNF grammar	Action
$S \rightarrow NP \;\; VP \;\; PUNC$		

Conversion of CFG to CNF: Step 3

Non-CNF grammar	CNF grammar	Action
$S \rightarrow NP \;\; VP \;\; PUNC$		
	$S \rightarrow NP \;\; X3$	(new symbol)

Conversion of CFG to CNF: Step 3

Non-CNF grammar	CNF grammar	Action
$S \rightarrow NP \;\; VP \;\; PUNC$		
	$S \rightarrow NP \;\; X3$	(new symbol)
	$X3 \rightarrow VP \;\; PUNC$	

Conversion of CFG to CNF: Step 3

Non-CNF grammar	CNF grammar	Action
$S \rightarrow NP\ \ VP\ \ PUNC$	$S \rightarrow NP\ \ X3$ $X3 \rightarrow VP\ \ PUNC$	(new symbol)
$NP \rightarrow DT\ \ NP$		

Conversion of CFG to CNF: Step 3

Non-CNF grammar	CNF grammar	Action
$S \rightarrow NP \;\; VP \;\; PUNC$		
	$S \rightarrow NP \;\; X3$	(new symbol)
	$X3 \rightarrow VP \;\; PUNC$	
$NP \rightarrow DT \;\; NP$	$NP \rightarrow DT \;\; NP$	(carry over)

Conversion of CFG to CNF: Step 3

Non-CNF grammar	CNF grammar	Action
$S \rightarrow NP \;\; VP \;\; PUNC$		
	$S \rightarrow NP \;\; X3$	(new symbol)
	$X3 \rightarrow VP \;\; PUNC$	
$NP \rightarrow DT \;\; NP$	$NP \rightarrow DT \;\; NP$	(carry over)
$VP \rightarrow VBZ \;\; NP$		

Conversion of CFG to CNF: Step 3

Non-CNF grammar	CNF grammar	Action
$S \rightarrow NP\ \ VP\ \ PUNC$		
	$S \rightarrow NP\ \ X3$	(new symbol)
	$X3 \rightarrow VP\ \ PUNC$	
$NP \rightarrow DT\ \ NP$	$NP \rightarrow DT\ \ NP$	(carry over)
$VP \rightarrow VBZ\ \ NP$	$VP \rightarrow VBZ\ \ NP$	(carry over)

Conversion of CFG to CNF: Step 3

Non-CNF grammar	CNF grammar	Action
$S \rightarrow NP\ \ VP\ \ PUNC$		
	$S \rightarrow NP\ \ X3$	(new symbol)
	$X3 \rightarrow VP\ \ PUNC$	
$NP \rightarrow DT\ \ NP$	$NP \rightarrow DT\ \ NP$	(carry over)
$VP \rightarrow VBZ\ \ NP$	$VP \rightarrow VBZ\ \ NP$	(carry over)
$DT \rightarrow the$		

Conversion of CFG to CNF: Step 3

Non-CNF grammar	CNF grammar	Action
$S \rightarrow NP \;\; VP \;\; PUNC$		
	$S \rightarrow NP \;\; X3$	(new symbol)
	$X3 \rightarrow VP \;\; PUNC$	
$NP \rightarrow DT \;\; NP$	$NP \rightarrow DT \;\; NP$	(carry over)
$VP \rightarrow VBZ \;\; NP$	$VP \rightarrow VBZ \;\; NP$	(carry over)
$DT \rightarrow the$	$DT \rightarrow the$	(carry over)

CFG in CNF

$NP \rightarrow dog$
$NP \rightarrow cat$
$VP \rightarrow sleeps$
$VP \rightarrow eats$
$S \rightarrow S \;\; X1$
$X1 \rightarrow X2 \;\; S$
$X2 \rightarrow and$

$S \rightarrow NP \;\; X3$
$X3 \rightarrow VP \;\; PUNC$
$NP \rightarrow DT \;\; NP$
$VP \rightarrow VBZ \;\; NP$
$DT \rightarrow the$

Homework 2 discussion

Homework: CKY and toCNF

Symbol naming conventions

Refer to NLTK `treetransforms` module

- Create new symbols from old (binarization):
  $S \rightarrow NP \;\; VP \;\; PUNC$ becomes:
  $S \rightarrow NP \;\; \mathrm{S}|\langle VP\text{-PUNC}\rangle$, $\mathrm{S}|\langle\text{VP-PUNC}\rangle \rightarrow VP \;\; PUNC$
- Create new symbols from old (unary collapsing):
  $SBAR \rightarrow S$, $S \rightarrow NP \;\; VP$ becomes
  $SBAR{+}S \rightarrow NP \;\; VP$