Natural Language Processing

Anoop Sarkar

http://www.cs.sfu.ca/~anoop

Ambiguity

- An input is ambiguous with respect to a CFG if it can be derived with two different parse trees
- A parser needs a mechanical definition of ambiguity as it parses the input string
- Is a parser choice really ambiguous, i.e. does it lead to ambiguous parse trees? or not?
- We can formally define ambiguity in terms of the derivations possible in a CFG

Arithmetic Expressions

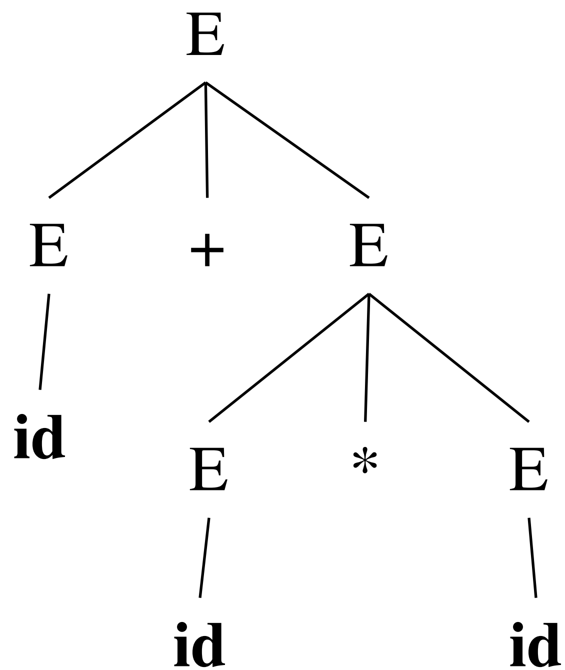
- $E \rightarrow E + E$
- $E \rightarrow E * E$
- $E \rightarrow (E)$
- $E \rightarrow - E$
- $E \rightarrow \mathbf{id}$

Leftmost derivations for
id + id * id

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow - E$
$E \rightarrow \mathbf{id}$

- $E \Rightarrow E + E$

$\Rightarrow \mathbf{id} + E$
$\Rightarrow \mathbf{id} + E * E$
$\Rightarrow \mathbf{id} + \mathbf{id} * E$
$\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id}$

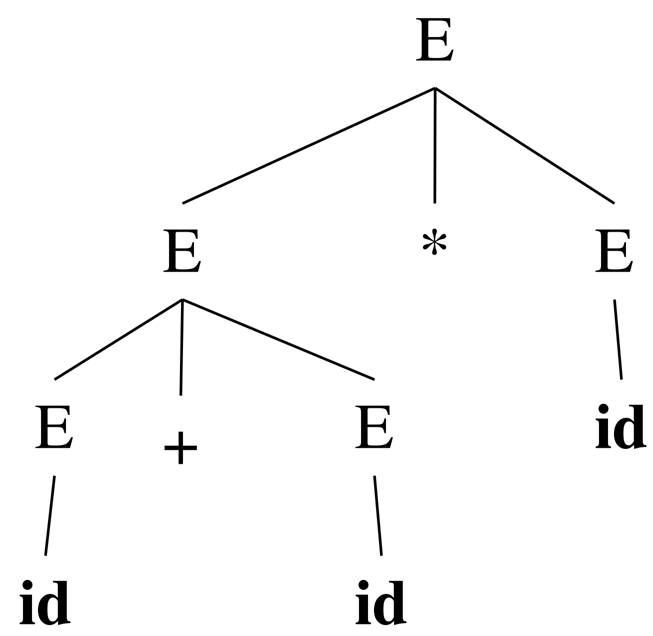


Leftmost derivations for **id + id * id**

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow - E$
$E \rightarrow \textbf{id}$

- $E \Rightarrow E * E$

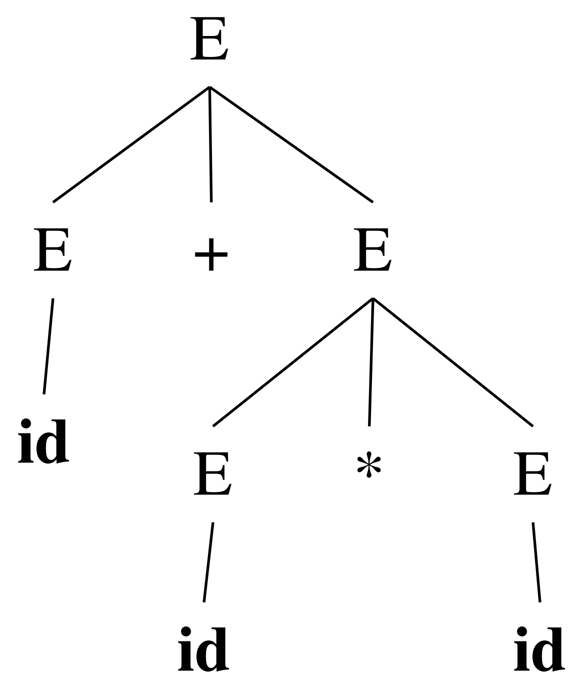
$\Rightarrow E + E * E$
$\Rightarrow \textbf{id} + E * E$
$\Rightarrow \textbf{id} + \textbf{id} * E$
$\Rightarrow \textbf{id} + \textbf{id} * \textbf{id}$



Rightmost derivation for **id + id * id**

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow (E)$
$E \rightarrow - E$
$E \rightarrow \mathbf{id}$

$$
\begin{aligned}
E &\Rightarrow E + E \\
&\Rightarrow E + E * E \\
&\Rightarrow E + E * \mathbf{id} \\
&\Rightarrow E + \mathbf{id} * \mathbf{id} \\
&\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id}
\end{aligned}
$$



Rightmost derivation for **id + id * id**

$E \rightarrow E + E$

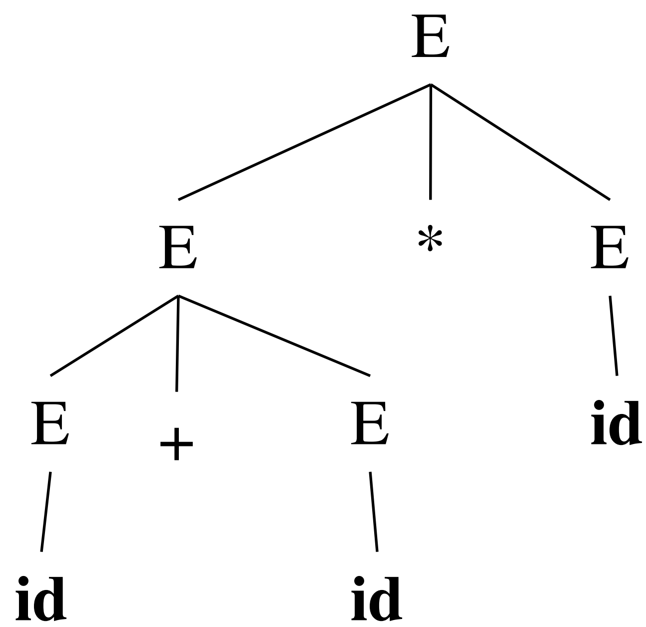
$E \rightarrow E * E$

$E \rightarrow (E)$

$E \rightarrow - E$

$E \rightarrow \mathbf{id}$

$$
\begin{aligned}
E &\Rightarrow E * E \\
&\Rightarrow E * \mathbf{id} \\
&\Rightarrow E + E * \mathbf{id} \\
&\Rightarrow E + \mathbf{id} * \mathbf{id} \\
&\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id}
\end{aligned}
$$



Ambiguity

- We can now define *ambiguity* for a context-free parser
- If a parser has a choice of two different leftmost derivations,
- or if a parser has a choice of two different rightmost derivations,
- for a particular input then that input is ambiguous

CKY Recognition Algorithm

- The Cocke-Kasami-Younger algorithm
- As we shall see it runs in time that is polynomial in the size of the input
- It takes space polynomial in the size of the input
- **Remarkable fact:** it can find all possible parse trees (exponentially many) in polynomial time

Chomsky Normal Form

- Before we can see how CKY works, we need to convert the input CFG into Chomsky Normal Form
- CNF is one of many grammar transformations that *preserve* the language
- CNF means that the input CFG G is converted to a new CFG G’ in which all rules are of the form:

  $A \rightarrow B\ C$

  $A \rightarrow a$

Epsilon Removal

- First step, remove epsilon rules

  $A \rightarrow B\ C$

  $C \rightarrow \varepsilon \mid C\ D \mid a$

  $D \rightarrow b \quad B \rightarrow b$

- After ε-removal:

  $A \rightarrow B \mid B\ C\ D \mid B\ a \mid BC$

  $C \rightarrow D \mid C\ D\ D \mid a\ D \mid C\ D \mid a$

  $D \rightarrow b \quad B \rightarrow b$

Removal of Chain Rules

- Second step, remove chain rules

  $A \rightarrow B\ C \mid C\ D\ C$

  $C \rightarrow D \mid a$

  $D \rightarrow d \qquad B \rightarrow b$

- After removal of chain rules:

  $A \rightarrow B\ a \mid B\ D \mid a\ D\ a \mid a\ D\ D \mid D\ D\ a \mid D\ D\ D$

  $D \rightarrow d \qquad B \rightarrow b$

Eliminate terminals from RHS

- Third step, remove terminals from the rhs of rules

  $A \rightarrow B\ a\ C\ d$

- After removal of terminals from the rhs:

  $A \rightarrow B\ N_1\ C\ N_2$

  $N_1 \rightarrow a$

  $N_2 \rightarrow d$

Binarize RHS with Nonterminals

- Fourth step, convert the rhs of each rule to have two non-terminals

  $A \rightarrow B\ N_1\ C\ N_2$

  $N_1 \rightarrow a$

  $N_2 \rightarrow d$

- After converting to binary form:

$A \rightarrow B\ N_3$	$N_1 \rightarrow a$
$N_3 \rightarrow N_1\ N_4$	$N_2 \rightarrow d$
$N_4 \rightarrow C\ N_2$	

CKY algorithm

- We will consider the working of the algorithm on an example CFG and input string
- Example CFG:

  $S \rightarrow A\ X \mid Y\ B$

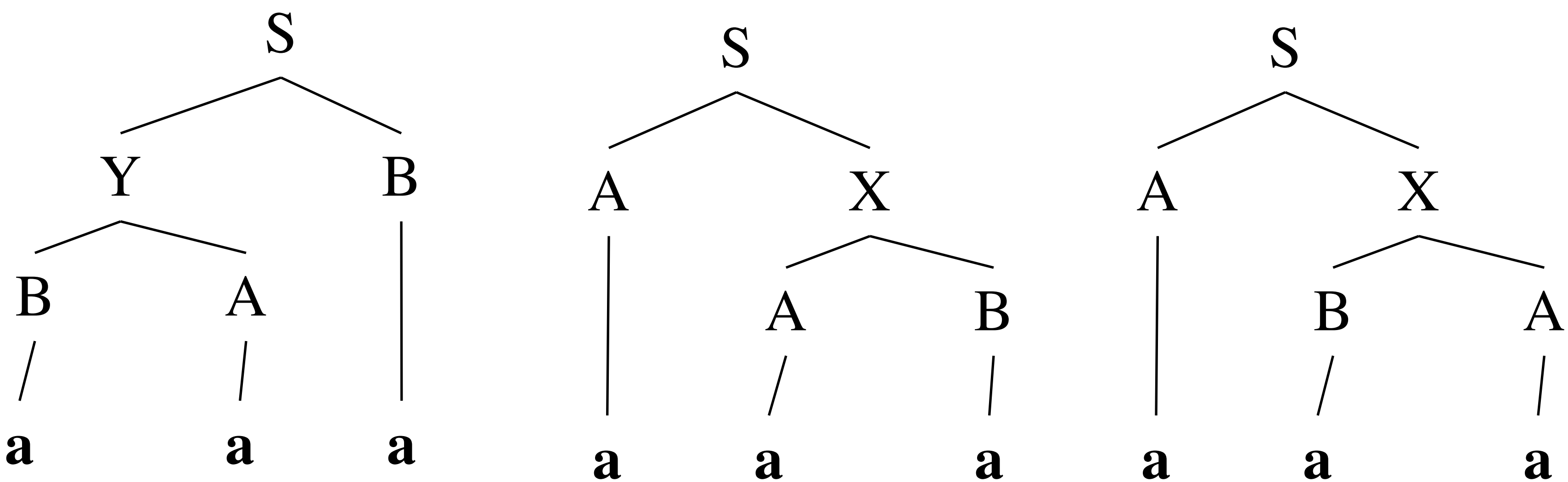
  $X \rightarrow A\ B \mid B\ A \qquad Y \rightarrow B\ A$

  $A \rightarrow a \qquad B \rightarrow a$
- Example input string: *aaa*

CKY Algorithm

	0	1	2	3
0		A, B $A \rightarrow a$ $B \rightarrow a$	X, Y $X \rightarrow A\ B \mid B\ A$ $Y \rightarrow B\ A$	S $S \rightarrow A_{(0,1)}\ X_{(1,3)}$ $S \rightarrow Y_{(0,2)}\ B_{(2,3)}$
1			A, B $A \rightarrow a$ $B \rightarrow a$	X, Y $X \rightarrow A\ B \mid B\ A$ $Y \rightarrow B\ A$
2				A, B $A \rightarrow a$ $B \rightarrow a$
		a	a	a

Parse trees



CKY Algorithm

Input string **input** of size *n*

Create a 2D table **chart** of size n^2

for i=0 **to** n-1

    chart[i][i+1] = A **if** there is a rule A → a and **input**[i]=a

for j=2 **to** N

    for i=j-2 **downto** 0

        for k=i+1 **to** j-1

            chart[i][j] = A **if** there is a rule A → B C **and**
              chart[i][k] = B **and chart**[k][j] = C

return *yes* **if chart**[0][n] has the start symbol

else return *no*

CKY algorithm summary

- Parsing arbitrary CFGs
- For the CKY algorithm, the time complexity is $O(|G|^2\ n^3)$
- The space requirement is $O(n^2)$
- The CKY algorithm handles arbitrary ambiguous CFGs
- All ambiguous choices are stored in the chart
- For compilers we consider parsing algorithms for CFGs that do not handle ambiguous grammars

Parsing - Summary

- Parsing arbitrary CFGs: $O(n^3)$ time complexity
- Top-down vs. bottom-up
  - Recursive-descent parsing
  - Shift-reduce parsing
- Earley parsing
- Ambiguous grammars result in parser output with multiple parse trees for a single input string

Parsing - Additional Results

- $O(n^2)$ time complexity for linear grammars
  - All rules are of the form $S \rightarrow aSb$ or $S \rightarrow a$
  - Reason for $O(n^2)$ bound is the linear grammar normal form: $A \rightarrow aB$, $A \rightarrow Ba$, $A \rightarrow B$, $A \rightarrow a$
- Left corner parsers
  - extension of top-down parsing to arbitrary CFGs
- Earley's parsing algorithm
  - $O(n^3)$ worst case time for arbitrary CFGs just like CKY
  - $O(n^2)$ worst case time for unambiguous CFGs
  - $O(n)$ for specific unambiguous grammars (e.g. $S \rightarrow aSa \mid bSb \mid \varepsilon$)

CKY algorithm for PCFGs

- We will consider the working of the algorithm on an example PCFG and input string
- Example PCFG:

  $S \to A\ X$ (0.3) | $Y\ B$ (0.7)

  $X \to A\ B$ (0.1) | $B\ A$ (0.9) $Y \to B\ A$ (1.0)

  $A \to a$ (1.0) $B \to a$ (1.0)
- Example input string: *aaa*

CKY Algorithm

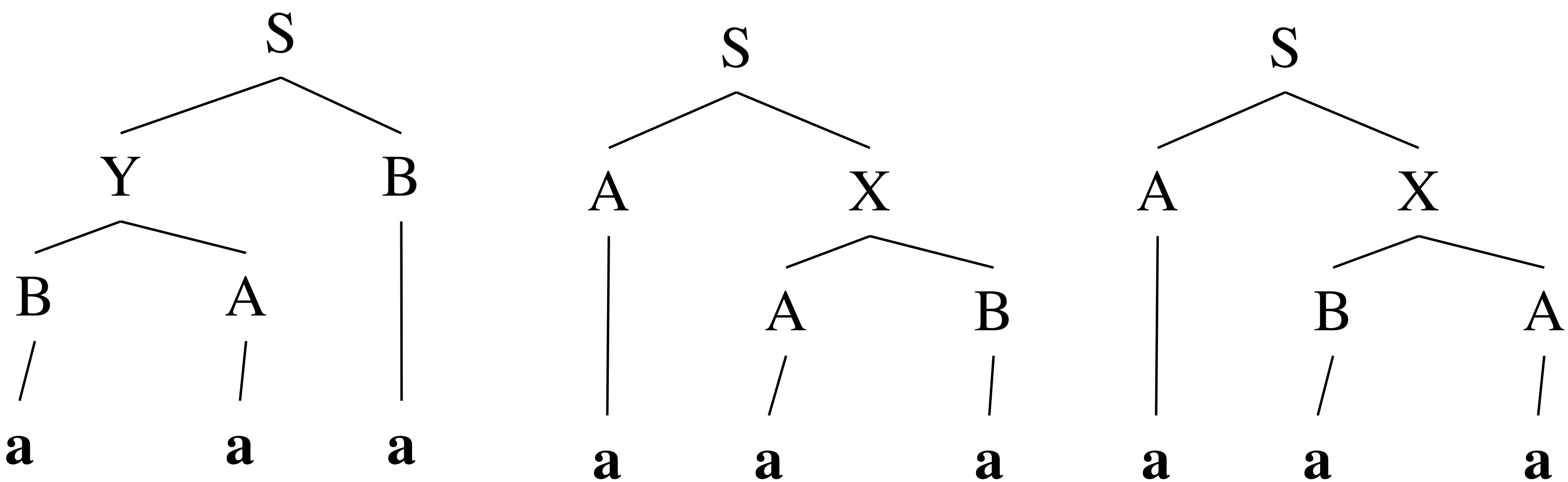
Max(0.1, 0.9)

$0.3 * 0.9 = 0.27$
Max(0.27, 0.7)

	0	1	2	3
0		A 1.0, B 1.0 $A \rightarrow a_{\ 1.0}$ $B \rightarrow a_{\ 1.0}$	X 0.9, Y 1.0 $X \rightarrow A\,B_{\ 0.1} \mid B\,A_{\ 0.9}$ $Y \rightarrow B\,A_{\ 1.0}$	S 0.7 $S \rightarrow A_{(0,1)}\, X_{(1,3)\ 0.3}$ $S \rightarrow Y_{(0,2)}\, B_{(2,3)\ 0.7}$
1			A 1.0, B 1.0 $A \rightarrow a_{\ 1.0}$ $B \rightarrow a_{\ 1.0}$	X 0.9, Y 1.0 $X \rightarrow A\,B_{\ 0.1} \mid B\,A_{\ 0.9}$ $Y \rightarrow B\,A_{\ 1.0}$
2				A 1.0, B 1.0 $A \rightarrow a_{\ 1.0}$ $B \rightarrow a_{\ 1.0}$
		a	a	a

Parse trees

PCFG is consistent:
$0.7 + 0.27 + 0.03 = 1.0$



0.7 0.27 0.03