Parsing CFGs

- Consider the problem of parsing with arbitrary CFGs
- For any input string, the parser has to produce a parse tree
- The simpler problem: print yes if the input string is generated by the grammar, print no otherwise
- This problem is called recognition
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 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
  \[ A \rightarrow B \ C \]
  \[ C \rightarrow \varepsilon \ D \ a \]
  \[ D \rightarrow b \quad B \rightarrow b \]
• After \( \varepsilon \)-removal:
  \[ A \rightarrow B \ D \ a \ D \ a \ a \ D \ D \ a \ D \ D \]

Removal of Chain Rules

• Second step, remove chain rules
  \[ A \rightarrow B \ C \ D \ C \]
  \[ C \rightarrow D \ a \]
  \[ D \rightarrow d \quad B \rightarrow b \]
• After removal of chain rules:
  \[ A \rightarrow B \ D \ a \ D \ D \ a \ a \ D \ D \]

10/22/07
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:
  \[ A \rightarrow B \ N_3 \quad N_1 \rightarrow a \]
  \[ N_3 \rightarrow N_1 \ N_4 \quad N_2 \rightarrow d \]
  \[ N_4 \rightarrow C \ N_2 \]
CKY algorithm

- We will consider the working of the algorithm on an example CFG and input string
- Example CFG:
  \[
  S \rightarrow A \ X \mid Y \ B \\
  X \rightarrow A \ B \mid B \ A \\
  Y \rightarrow B \ A \\
  A \rightarrow a \\
  B \rightarrow a \\
  \]
- Example input string: \textit{aaa}

CKY Algorithm

\[
\begin{array}{c|ccc|c}
0 & A, B & A \rightarrow a & B \rightarrow a & X, Y \\
1 & A, B & A \rightarrow a & B \rightarrow a & X \rightarrow A \ B \mid B \ A \\
2 & & & & X \rightarrow A \ B \mid B \ A \\
   & & & & Y \rightarrow B \ A \\
   & & & & X, Y \\
   & & & & S \\
   & & & & S \rightarrow A_{(0,1)} X_{(1,3)} \\
   & & & & S \rightarrow Y_{(0,2)} B_{(2,3)} \\
\end{array}
\]

\[
\begin{array}{cccc}
a & a & a & a \\
\end{array}
\]
Parse trees

```
S
  |
  |  Y
  |  B
B  A  a  a  a
```

```
S
  |
  |  A
  |  X
A  B  a  a  a
```

```
S
  |
  |  A
  |  X
B  A  a  a  a
```

---

CKY Algorithm

Input string `input` of size \( n \)
Create a 2D table `chart` of size \( n^2 \)

```python
for i=0 to n-1
    chart[i][i+1] = A if there is a rule A \rightarrow 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 \rightarrow 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

GLR – Generalized LR Parsing

- Works for any CFG (just like CKY algorithm)
  - Masaru Tomita [1986]
- If you have shift/reduce conflict, just clone your stack and shift in one clone, reduce in the other clone
  - proceed in lockstep
  - parser that get into error states die
  - merge parsers that lead to identical reductions (graph structured stack)
- Careful implementation can provide $O(n^3)$ bound
- However for some grammars, parser will be exponential in grammar size
Parsing - Summary

• Parsing arbitrary CFGs using the CKY algorithm: $O(n^3)$ time complexity
• Chomsky Normal Form (CNF) provides the $n^3$ time bound
• LR parsers can be extended to Generalized LR parsers to deal with arbitrary CFGs, complexity is still $O(n^3)$

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
  – $O(n^3)$ worst case time for unambiguous CFGs
  – $O(n)$ for specific unambiguous grammars
  (e.g. $S \rightarrow aSa \mid bSb \mid \varepsilon$)