Discussion #4

Ambiguity & Precedence Grammars
Topics

- Ambiguity
- Precedence Grammars
- Grammar Reductions
Problems with Grammars

• We are heading towards grammars that are reasonable, get the job done, and for which we can find efficient parsing algorithms.

• Not all grammars are usable!
  – Ambiguous
  – “Silly”
  – Have unproductive non-terminals
  – Have unreachable rules
Ambiguous Grammars

• A grammar for which there are two different parse trees for the same terminal string is said to be ambiguous.

• We can show that a grammar is ambiguous by giving two parse trees for the same terminal string.
An Ambiguous Grammar

\[ \Phi = \{ E \rightarrow D \mid (E) \mid E + E \mid E - E \mid E \ast E \mid E / E , D \rightarrow 0 \mid 1 \mid \ldots \mid 9 \} \]
Ambiguity & Multiple Meanings

- Precedence: $(1+2)*3 \neq 1+(2*3)$
- Associativity: $(3-2)-1 \neq 3-(2-1)$
- Different if-then-else nestings
- ...
Fixing Precedence Ambiguity

• Observe: Operators lower in the parse tree are executed first in an expression tree.
• Observe: Operators executed first have higher precedence.
• Fix: We make our grammar correctly represent precedence levels by introducing a new non-terminal symbol for each precedence level.
Example: Fixing Precedence Ambiguity

\[ \Phi = \{ E \rightarrow D \mid (E) \mid E + E \mid E - E \mid E * E \mid E / E, \]
\[ D \rightarrow 0 \mid 1 \mid \ldots \mid 9 \} \]

\[
\begin{align*}
E &\rightarrow T \mid E + T \mid E - T \\
T &\rightarrow F \mid T * F \mid T / F \\
F &\rightarrow D \mid (E) \\
D &\rightarrow 0 \mid 1 \mid \ldots \mid 9
\end{align*}
\]
Fixing Associative Ambiguity

- Left recursion yields left associativity.
- Right recursion yields right associativity.
Left Associativity ~ Left Recursion

\[ E \rightarrow D \mid D - E \]

3 – 2 – 1

E

D

E

2

D

1

\[ E \rightarrow D \mid E - D \]

3 – 2 – 1

E

E - D

E - D

1

D

2

3
Right Associativity $\sim$ Right Recursion

$E \rightarrow D \mid D \uparrow E$

$2^{3^2} = 2 \uparrow 3 \uparrow 2$

$E \rightarrow D \mid E \uparrow D$

$2 \uparrow 3 \uparrow 2$

Discussion #4
Adding the Power Operator

\[ E \rightarrow T \mid E + T \mid E - T \]
\[ T \rightarrow F \mid T \ast F \mid T / F \]
\[ F \rightarrow D \mid (E) \]

\[ E \rightarrow T \mid E + T \mid E - T \]
\[ T \rightarrow P \mid T \ast P \mid T / P \]
\[ P \rightarrow F \mid F \uparrow P \]
\[ F \rightarrow D \mid (E) \]
Grammar Reductions: “Silly” Rules

Some grammars have “silly” constructions

$$\Phi = \{ A \rightarrow A \mid a \}$$

$$A \rightarrow A \rightarrow a$$

$$A \rightarrow A \rightarrow A \rightarrow A \rightarrow a$$

Always ambiguous; rewrite as $$\Phi = \{ A \rightarrow a \}$$.
Grammar Reductions: Unreachable Rules

Some grammars have unreachable rules.

\[ \Phi = \{ S \rightarrow aABb, \]
\[ A \rightarrow a \mid aA, \]
\[ B \rightarrow b \mid bBD, \]
\[ C \rightarrow cD, \quad \text{can’t be reached} \]
\[ D \rightarrow e \} \]
Least Fixed Point Algorithm
Finds Unreachable Rules

1. Initialize the set of reachable non-terminals $R$ with the start symbol.
2. For each round, if $R$ includes the lhs of a production rule, add the non-terminals in the rhs to $R$.
3. Loop on #2 until there are no changes to $R$.
4. Rules whose lhs’s are non-terminals in $V_N$ minus the non-terminals in $R$ constitute the set of unreachable rules.

$$\Phi = \{ \begin{array}{l}
S \rightarrow aABb, \\
A \rightarrow a | aA, \\
B \rightarrow b | bBD, \\
C \rightarrow cD, \\
D \rightarrow e \end{array} \}$$

Initialize: $R = \{S\}$
Round 1: $R = \{S, A, B\}$
Round 2: $R = \{S, A, B, D\}$
Round 3: $R = \{S, A, B, D\}$
Done: no change: $V_N - \{S, A, B, D\} = \{C\}$
Grammar Reductions: Unproductive Non-terminals

Some grammars have non-terminals that can never become all terminals (unproductive, inactive).

\[ \Phi = \{ \ S \rightarrow aABb , \]
\[ A \rightarrow bC , \]
\[ B \rightarrow d \mid dB , \]
\[ C \rightarrow eC \} \]

C never produces all terminals.

\[ C \rightarrow eC \rightarrow eeC \rightarrow eeeC \rightarrow \ldots \ e^nC \]

A also because it always produces C

\[ A \rightarrow bC \rightarrow beC \rightarrow beeC \rightarrow \ldots \ be^nC \]

S also because it always produce A

\[ S \rightarrow aABb \rightarrow aAbb \rightarrow abCbb \rightarrow \ldots \]
Least Fixed Point Algorithm
Finds Unproductive Non-terminals

1. Start with the set of terminals $T$.
2. For each round, if $T$ covers a rhs of a production rule, add the lhs to $T$.
3. Loop on #2 until there are no changes to $T$.
4. The alphabet of terminals and non-terminals, $V$, minus $T$ is the set of unproductive non-terminals.

$$
\Phi = \{ \text{S} \rightarrow \text{aABb} , \text{A} \rightarrow \text{bC} , \text{B} \rightarrow \text{d} | \text{dB} , \text{C} \rightarrow \text{eC} \} \\
\begin{array}{ll}
\text{Initialize: } T = \{a, b, d, e\} \\
\text{Round 1: } T = \{a, b, d, e, B\} \\
\text{Round 2: } T = \{a, b, d, e, B\} \\
\text{Done: no change: } \{a, b, d, e, A, B, C, S\} - T = \{A, C, S\}
\end{array}
$$