Announcements

• HW4 due tomorrow
Know the difference

Slow Loris

Medium speed Loris
Building a Predictive Parser
Last Time: Intro LL(1) Predictive Parser

• “predict” the parse tree top-down
• Parser structure
  – 1 token of lookahead
  – A stack tracking parse tree frontier
  – Selector/parse table
• Necessary conditions
  – Left-factored
  – Free of left-recursion
Today: Building the Parse Table

• Review Grammar transformations
  – Why they are necessary
  – How they work
• Build the selector table
  – FIRST(X): Set of terminals that can begin at a subtree rooted at X
  – FOLLOW(X): Set of terminals that can appear after X
Review LL(1) Grammar Transformations

• Necessary (but not sufficient conditions) for LL(1) Parsing:
  – Left factored
    • No rules with common prefix
    • Why? We’d need to look past the prefix to pick rule
  – Free of left recursion
    • No nonterminal loops for a production
    • Why? Need to look past list to know when to cap it
Why Left Recursion is a Problem (Blackbox View)

CFG snippet: \( XList \rightarrow XList \; x \; | \; x \)

Current parse tree: \( XList \)  
Current token: \( x \)

How should we grow the tree top-down?

Correct if there are no more \( x \)s
Correct if there are more \( x \)s

We don’t know which without more lookahead
Why Left Recursion is a Problem (Whitebox View)

CFG snippet: \( XList \rightarrow XList \ x \ | \ x \)

Current parse tree: \( XList \)

Parse table: \( XList \rightarrow XList \ x \ | \ x \)

Current token: \( x \)

(Stack overflow)
Left Recursion Elimination: Review

Replace $A \rightarrow A\alpha \mid \beta$

With $A \rightarrow \beta A'$
$A' \rightarrow \alpha A' \mid \varepsilon$

Where $\beta$ does not start with $A$ and may not be present

Preserve order (a list of $\alpha$ starting with $\beta$) but use right recursion
Left Recursion Elimination: Ex1

\[ A \rightarrow A \alpha \mid \beta \quad \rightarrow \quad A \rightarrow \beta A' \]
\[ A' \rightarrow \alpha A' \mid \epsilon \]

\[ E \rightarrow E \text{ cross id} \mid \text{id} \quad \rightarrow \quad E \rightarrow \text{id} E' \]
\[ E' \rightarrow \text{cross id} E' \mid \epsilon \]
Left Recursion Elimination: Ex2

\[
\begin{align*}
A & \rightarrow A \alpha \mid \beta \\
A' & \rightarrow \beta A' \\
A' & \rightarrow \alpha A' \mid \epsilon
\end{align*}
\]

\[
\begin{align*}
E & \rightarrow E + T \mid T \\
T & \rightarrow T * F \mid F \\
F & \rightarrow (E) \mid \text{id}
\end{align*}
\]

\[
\begin{align*}
E & \rightarrow TE' \\
E' & \rightarrow + TE' \mid \epsilon \\
T & \rightarrow FT' \\
T' & \rightarrow *FT' \mid \epsilon \\
F & \rightarrow (E) \mid \text{id}
\end{align*}
\]
Left Recursion Elimination: Ex3

\[
A \rightarrow A \alpha \mid \beta \quad \rightarrow \quad A \rightarrow \beta A'
\]

\[
A' \rightarrow \alpha A' \mid \varepsilon
\]

\[
SList \rightarrow SList D \mid \varepsilon
\]

\[
D \rightarrow Type \text{id semi}
\]

\[
Type \rightarrow bool \mid int
\]

\[
SList' \rightarrow D SList' \mid \varepsilon
\]

\[
D \rightarrow Type \text{id semi}
\]

\[
Type \rightarrow bool \mid int
\]
Left Factoring: Review

Removing common prefix from grammar

Replace

\[ A \rightarrow \alpha \beta_1 | \ldots | \alpha \beta_m | y_1 | \ldots | y_n \]

With

\[ A \rightarrow \alpha A' | y_1 | \ldots | y_n \]
\[ A' \rightarrow \beta_1 | \ldots | \beta_m \]

Where \( \beta_i \) and \( y_i \) are sequence of symbols with no common prefix
\( y_i \) May not be present, one of the \( \beta \) may be \( \epsilon \)

Squash all “problem” rules starting with \( \alpha \) together into one rule \( \alpha A' \)
Now \( A' \) represents the suffix of the “problem” rules
Left Factoring: Example 1

\[ A \rightarrow \alpha \beta_1 \mid \ldots \mid \alpha \beta_m \mid y_1 \mid \ldots \mid y_n \\rightarrow A \rightarrow \alpha A' \mid y_1 \mid \ldots \mid y_n \]

\[ A' \rightarrow \beta_1 \mid \ldots \mid \beta_m \]

\[ X \rightarrow \langle a \rangle \mid \langle b \rangle \mid \langle c \rangle \mid d \]

\[ X \rightarrow \langle X' \rangle \mid d \]

\[ X' \rightarrow a \mid b \mid c \]

\[ \beta_1 \beta_2 \beta_3 \]

\[ \alpha \beta_1 \alpha \beta_2 \alpha \beta_3 \gamma_1 \]
Left Factoring: Example 2

\[ A \rightarrow \alpha \beta_1 | \ldots | \alpha \beta_m | y_1 | \ldots | y_n \quad \longrightarrow \quad A \rightarrow \alpha A' | y_1 | \ldots | y_n \]

\[ A' \rightarrow \beta_1 | \ldots | \beta_m \]

\[ Stmt \rightarrow \text{id assign } E | \text{id ( } E\text{List } | \text{return} \]

\[ E \rightarrow \text{intlit } | \text{id} \]

\[ E\text{List } \rightarrow E | E \text{ comma } E\text{List} \]

\[ Stmt \rightarrow \text{id } Stmt' | \text{return} \]

\[ Stmt' \rightarrow \text{assign } E | ( \ E\text{List } ) \]

\[ E \rightarrow \text{intlit } | \text{id} \]

\[ E\text{List } \rightarrow E | E \text{ comma } E\text{List} \]
Left Factoring: Example 3

\[ A \rightarrow \alpha \beta_1 | \ldots | \alpha \beta_m | y_1 | \ldots | y_n \]

\[ A \rightarrow \alpha A' | y_1 | \ldots | y_n \]
\[ A' \rightarrow \beta_1 | \ldots | \beta_m \]

\[ S \rightarrow \text{if } E \text{ then } S | \text{if } E \text{ then } S \text{ else } S | \text{semi} \]

\[ E \rightarrow \text{boollit} \]

\[ S \rightarrow \text{if } E \text{ then } S \ S' | \text{semi} \]
\[ S' \rightarrow \text{else } S | \varepsilon \]
\[ E \rightarrow \text{boollit} \]
Left Factoring: Not Always Immediate

\[ A \rightarrow \alpha \beta_1 | \ldots | \alpha \beta_m | y_1 | \ldots | y_n \quad \rightarrow \quad A \rightarrow \alpha A' | y_1 | \ldots | y_n\]
\[ A' \rightarrow \beta_1 | \ldots | \beta_m \]

This snippet yearns for left-factoring

\[ S \rightarrow A | C | \text{return} \]
\[ A \rightarrow \text{id assign } E \]
\[ C \rightarrow \text{id ( EList )} \]

but we cannot! At least without inlining

\[ S \rightarrow \text{id assign } E | \text{id ( EList )} | \text{return} \]
Let’s be more constructive

- So far, we’ve only talked about what precludes us from building a predictive parser
- It’s time to actually build the parse table
Building the Parse Table

• What do we actually need to ensure arbitrary production $A \rightarrow \alpha$ is the correct one to apply?  
  
  \textit{(assume $\alpha$ is an arbitrary symbol string)}

1. What terminals could possibly start $\alpha$ (we call this the FIRST set)

2. What terminal could possibly come after $A$ (we call this the FOLLOW set)
FIRST Sets

• FIRST(α) is the set of terminals that begin the strings derivable from α, and also, if α can derive ε, then ε is in FIRST(α).

• Formally, FIRST(α) =
Why is FIRST Important?

• Assume the top-of-stack symbol is $A$ and current token is $a$
  – Production 1: $A \rightarrow \alpha$
  – Production 2: $A \rightarrow \beta$

• FIRST let us disambiguate:
  – If $a$ is in $\text{FIRST}(\alpha)$, it tells us that Production 1 is a viable choice
  – If $a$ is in $\text{FIRST}(\beta)$, it tells us that Production 2 is a viable choice
  – If $a$ is in only $\text{FIRST}(\alpha) \text{xor} \text{FIRST}(\beta)$, we can predict the rule we need.
FIRST Construction: Single Symbol

• We begin by doing FIRST sets for a single, arbitrary symbol X
  – If X is a terminal: FIRST(X) = { X }
  – If X is ε: FIRST(ε) = { ε }
  – If X is a nonterminal, for each X → Y₁ Y₂ ... Yₖ
    • Put FIRST(Y₁) - {ε} into FIRST(X)
    • If ε is in FIRST(Y₁), put FIRST(Y₂) - {ε} into FIRST(X)
    • If ε is also in FIRST(Y₂), put FIRST(Y₃) - {ε} into FIRST(X)
    • ...
    • If ε is in FIRST of all Yᵢ symbols, put ε into FIRST(X)
FIRST(X) Example

Building FIRST(X) for nonterm X
for each \( X \rightarrow Y_1 \ Y_2 \ldots \ Y_k \)
- Add \( \text{FIRST}(Y_1) - \{\varepsilon\} \)
- If \( \varepsilon \) is in \( \text{FIRST}(Y_1 \text{ to } i-1) \): add \( \text{FIRST}(Y_i) - \{\varepsilon\} \)
- If \( \varepsilon \) is in all RHS symbols, add \( \varepsilon \)

\[
\begin{align*}
\text{Exp} & \rightarrow \text{Term Exp'} \\
\text{Exp'} & \rightarrow \text{minus Term Exp'} | \varepsilon \\
\text{Term} & \rightarrow \text{Factor Term'} \\
\text{Term'} & \rightarrow \text{divide Factor Term'} | \varepsilon \\
\text{Factor} & \rightarrow \text{intlit} | \text{lp} \text{Exp} \text{rp}
\end{align*}
\]

FIRST(\text{Factor}) = \{ \text{intlit, lp} \}
FIRST(\text{Term'}) = \{ \text{divide}, \varepsilon \}
FIRST(\text{Term}) = \{ \text{intlit, lp} \}
FIRST(\text{Exp'}) = \{ \text{minus}, \varepsilon \}
FIRST(\text{Exp}) = \{ \text{intlit, lp} \}
FIRST(\(\alpha\))

- **We now extend FIRST to strings of symbols** \(\alpha\)
  - We want to define FIRST for all RHS
- **Looks very similar to the procedure for single symbols**
- **Let** \(\alpha = Y_1 \ Y_2 \ldots \ Y_k\)
  - Put FIRST(\(Y_1\)) - \(\{\varepsilon\}\) in FIRST(\(\alpha\))
  - If \(\varepsilon\) is in FIRST(\(Y_1\)): add FIRST(\(Y_2\)) – \(\{\varepsilon\}\) to FIRST(\(\alpha\))
  - If \(\varepsilon\) is in FIRST(\(Y_2\)): add FIRST(\(Y_3\)) – \(\{\varepsilon\}\) to FIRST(\(\alpha\))
  - ... 
  - If \(\varepsilon\) is in FIRST of all \(Y_i\) symbols, put \(\varepsilon\) into FIRST(\(\alpha\))
Building FIRST($\alpha$) from FIRST($X$)

Building FIRST($X$) for nonterm $X$
for each $X \rightarrow Y_1 \ Y_2 \ldots Y_k$
  • Add FIRST($Y_1$) - {$\varepsilon$}
  • If $\varepsilon$ is in FIRST($Y_1$ to $i-1$): add FIRST($Y_i$) - {$\varepsilon$}
  • If $\varepsilon$ is in all RHS symbols, add $\varepsilon$

Building FIRST($\alpha$)
Let $\alpha = Y_1 \ Y_2 \ldots Y_k$
  • Add FIRST($Y_1$) - {$\varepsilon$}
  • If $\varepsilon$ is in FIRST($Y_1$ to $i-1$): add FIRST($Y_i$) – {$\varepsilon$}
  • If $\varepsilon$ is in all RHS symbols, add $\varepsilon$
FIRST(\(\alpha\)) Example

Building FIRST(\(\alpha\))
Let \(\alpha = Y_1 Y_2 ... Y_k\)
  - Add FIRST(\(Y_1\)) - \{\(\varepsilon\)\}
  - If \(\varepsilon\) is in FIRST(\(Y_{1 \text{ to } i-1}\)): add FIRST(\(Y_i\)) – \{\(\varepsilon\)\}
  - If \(\varepsilon\) is in all RHS symbols, add \(\varepsilon\)

\[
\begin{align*}
E & \to TX \\
X & \to +TX | \varepsilon \\
T & \to FY \\
Y & \to *FY | \varepsilon \\
F & \to (E) | id
\end{align*}
\]

\[
\begin{align*}
\text{FIRST}(E) & = \{(, \text{id}\} \\
\text{FIRST}(T) & = \{(, \text{id}\} \\
\text{FIRST}(F) & = \{(, \text{id}\} \\
\text{FIRST}(X) & = \{+, \varepsilon\} \\
\text{FIRST}(Y) & = \{*, \varepsilon\} \\
\text{FIRST}(\text{id}) & = \{\text{id}\} \\
\text{FIRST}(TX) & = \{(, \text{id}\} \\
\text{FIRST}(+TX) & = \{+\} \\
\text{FIRST}(FY) & = \{(, \text{id}\} \\
\text{FIRST}(*FY) & = \{*\} \\
\text{FIRST}( (E) ) & = \{(\} \\
\text{FIRST}(\text{id}) & = \{\text{id}\}
\end{align*}
\]
FIRST Sets aren’t enough for Parse Tables

• If a rule can derive $\varepsilon$, we need to know what comes next
  – Obviously, some productions won’t work
FOLLOW Sets

• For nonterminal A, FOLLOW(A) is the set of terminals that can appear immediately to the right of A

• Formally, FOLLOW(A) =
FOLLOW Sets: Pictorially

• For nonterminal $A$, FOLLOW($A$) is the set of terminals that can appear immediately to the right of $A$
**FOLLOW Sets: Construction**

- To build FOLLOW(A)
  - If A is the start nonterminal, add *eof*  
    Where α, β may be empty
  - For rules $X \rightarrow \alpha A \beta$
    - Add FIRST(β) – {ε}
    - If ε is in FIRST(β) or β is empty, add FOLLOW(X)

- Continue building FOLLOW sets until saturation
FOLLOW Sets Example

FOLLOW(A) for $X \rightarrow \alpha A \beta$
If A is the start, add `eof`
Add FIRST(β) – {ε}
Add FOLLOW(X) if ε in FIRST(β) or β is empty

S → B c | D B
B → a b | c S
D → d | ε

<table>
<thead>
<tr>
<th>Nonterminal</th>
<th>FIRST</th>
<th>FOLLOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>{ a, c, d }</td>
<td>{ eof }</td>
</tr>
<tr>
<td>B</td>
<td>{ a, c }</td>
<td>{ c, eof }</td>
</tr>
<tr>
<td>D</td>
<td>{ d, e }</td>
<td>{ a, c }</td>
</tr>
<tr>
<td>B c</td>
<td>{ a, c }</td>
<td>{ eof, c }</td>
</tr>
<tr>
<td>D B</td>
<td>{ d, a, c }</td>
<td>{ c, eof }</td>
</tr>
<tr>
<td>a b</td>
<td>{ a }</td>
<td>{ a, c }</td>
</tr>
<tr>
<td>c S</td>
<td>{ c }</td>
<td>{ eof, c }</td>
</tr>
</tbody>
</table>

31
Building the Parse Table

for each production $X \rightarrow \alpha$ {
    for each terminal $t$ in FIRST($\alpha$) {
        put $\alpha$ in Table[$X$][$t$]
    }
    if $\varepsilon$ is in FIRST($\alpha$) {
        for each terminal $t$ in FOLLOW($X$) {
            put $\alpha$ in Table[$X$][$t$]
        }
    }
}

Table collision $\iff$ Grammar is not LL(1)
Putting it all together

• Build FIRST sets for each nonterminal
• Build FIRST sets for each production’s RHS
• Build FOLLOW sets for each nonterminal
• Use FIRST and FOLLOW to fill parse table for each production
Tips n’ Tricks

• FIRST sets
  – Only contain alphabet terminals and $\varepsilon$
  – Defined for arbitrary RHS and nonterminals
  – Constructed by starting at the beginning of a production

• FOLLOW sets
  – Only contain alphabet terminals and `eof`
  – Defined for nonterminals only
  – Constructed by jumping into production
FOLLOW(α) for α = Y₁ Y₂ ... Yₖ
Add FIRST(Y₁) - {ε}
If ε is in FIRST(Y₁ to i-1): add FIRST(Yᵢ) – {ε}
If ε is in all RHS symbols, add ε

FOLLOW(A) for X → α A β
If A is the start, add eof
Add FIRST(β) – {ε}
Add FOLLOW(X) if ε in FIRST(β) or β empty

<table>
<thead>
<tr>
<th>Table[X][t]</th>
<th>CFG</th>
</tr>
</thead>
<tbody>
<tr>
<td>for each production X → α</td>
<td>S → B c</td>
</tr>
<tr>
<td>for each terminal t in FIRST(α)</td>
<td>B → a b</td>
</tr>
<tr>
<td>put α in Table[X][t]</td>
<td>D → d</td>
</tr>
<tr>
<td>if ε is in FIRST(α)</td>
<td></td>
</tr>
<tr>
<td>for each terminal t in FOLLOW(X)</td>
<td></td>
</tr>
<tr>
<td>put α in Table[X][t]</td>
<td></td>
</tr>
</tbody>
</table>

| FIRST (S) = { a, c, d } | FOLLOW (S) = { eof, c } |
| FIRST (B) = { a, c } | FOLLOW (B) = { c, eof } |
| FIRST (D) = { d, ε } | FOLLOW (D) = { a, c } |
| FIRST (B c) = { a, c } | | |
| FIRST (D B) = { d, a, c } | | |
| FIRST (a b) = { a } | | |
| FIRST (c S) = { c } | | |

Not LL(1)