CMSC 330: Organization of Programming Languages

Parser Examples 2

Left Recursion Elimination Algorithm
- Given grammar
  \[ A \rightarrow A_1 \mid A_2 \mid \ldots \mid A_n \mid \beta \]
- Rewrite grammar as
  \[ A \rightarrow \beta L \]
  \[ L \rightarrow \alpha_1 L \mid \alpha_2 L \mid \ldots \mid \alpha_n L \mid \epsilon \]
- Repeat as necessary

Left Factoring Algorithm
- Given grammar
  \[ A \rightarrow \alpha \mid \beta \]
- Rewrite grammar as
  \[ A \rightarrow \alpha \uparrow L \]
  \[ L \rightarrow \beta \mid \alpha \mid \ldots \mid \alpha \]
- Repeat as necessary

Example 1 – Left Factoring
Consider the grammar
\[ S \rightarrow T + S | T \]
\[ T \rightarrow U * T | U \]
\[ U \rightarrow (S) | V \]
\[ V \rightarrow 0 | 1 | \ldots | 9 \]

Common prefix requires applying left factoring
\[ S \rightarrow T + S | T \]
\[ T \rightarrow U * T | U \]
\[ U \rightarrow (S) | V \]
\[ V \rightarrow 0 | 1 | \ldots | 9 \]

Example 2 – Computing First Sets
Consider the grammar
\[ S \rightarrow A S | b \]
\[ A \rightarrow S A | a \]

Compute First Sets
\[ \text{First}(S) = \{a, b\} \]
\[ \text{First}(A) = \{a, b\} \]
\[ \text{First}(b) = \{b\} \]
\[ \text{First}(a) = \{a\} \]

First sets for RHS
\[ \text{First}(a) = \{a\} \]
\[ \text{First}(b) = \{b\} \]

Example 3 – Computing First Sets
Consider the grammar
\[ S \rightarrow A S | b \]
\[ A \rightarrow S A | a \]

Compute First Sets
\[ \text{First}(S) = \{a, b\} \]
\[ \text{First}(A) = \{a, b\} \]
\[ \text{First}(b) = \{b\} \]

First sets for RHS
\[ \text{First}(a) = \{a\} \]
\[ \text{First}(b) = \{b\} \]

Example 3 – Using First Sets
Grammar is predictive if...
\[ \text{First}(AS) \cap \text{First}(b) = \emptyset \]
\[ \text{First}(SA) \cap \text{First}(a) = \emptyset \]
And not left recursive
Verify...
\[ \text{First}(AS) \cap \text{First}(b) = \emptyset \]
\[ \text{First}(SA) \cap \text{First}(a) = \emptyset \]

Overlap! Grammar is not predictive
Example 3 – Ambiguous Grammars

Grammar

\[
S \rightarrow AS \mid b \\
A \rightarrow SA \mid a
\]

c) Can the grammar be parsed by a backtracking parser?
No, due to mutual left recursion, where \( S \rightarrow AS \) derives \( S \rightarrow SA \)
d) Is the grammar ambiguous?
Yes. 2 left-most derivations of "abab"
e) Are all ambiguous grammars non-parseable by predictive parsers?
Yes. RHS guaranteed to conflict
f) Are all non-ambiguous grammars parseable by predictive parsers?
No. Consider \( S \rightarrow aab \mid aac \)

Example 4 – Computing First Sets

Consider the grammar

\[
S \rightarrow (L) \mid a \\
L \rightarrow LS \mid S
\]

Compute First Sets

First(\( S \)) = First(\( (L) \) for now
First(\( L \)) = First(\( (L), S \) for now
First(\( a \)) = \{ a \}

First(\( (L) \)) = \{ ( \}
First(\( a \)) = \{ a \}
First(\( (L), S \)) = \{ (, a \}

Example 4 – Using First Sets

Grammar

\[
S \rightarrow (L) \mid a \\
L \rightarrow LS \mid S
\]

c) Can the grammar be parsed by a backtracking parser?
No, due to left recursion, \( L \rightarrow L,S \)
d) Rewrite grammar using the rule for eliminating left recursion
S \rightarrow (L) \mid a \\
L \rightarrow LS \mid S
M \rightarrow S M \mid ε

Example 4 – Rewriting the Grammar

Grammar

\[
S \rightarrow (L) \mid a \\
L \rightarrow LS \mid S
\]

c) Can the grammar be parsed by a backtracking parser?
No, due to left recursion, \( L \rightarrow L,S \)
d) Rewrite grammar using the rule for eliminating left recursion
S \rightarrow (L) \mid a \\
L \rightarrow LS \mid S
M \rightarrow S M \mid ε

Example 4 – Recomputing First Sets

Grammar

\[
S \rightarrow (L) \mid a \\
L \rightarrow S M \mid M \rightarrow S M \mid ε
\]

e) Compute First Sets
First(\( S \)) = \{ (, a \}
First(\( L \)) = \{ (a \}
First(\( M \)) = \{ ε \}
First(\( a \)) = \{ a \}

f) Grammar is predictive if...
First(\( (L) \)) \cap First(\( (a \) = ∅
First(\( L,S \)) \cap First(\( S \) = ∅
First(\( S \)) \cap First(\( (L) \) = ∅
First(\( S \)) \cap First(\( a \) = ∅
First(\( M \)) \cap First(\( (a \) = ∅
First(\( M \)) \cap First(\( ε \) = ∅
Verify...
First(\( (L) \)) \cap First(\( (a \) = ∅
First(\( L,S \)) \cap First(\( S \) = ∅
First(\( S \)) \cap First(\( (L) \) = ∅
First(\( S \)) \cap First(\( a \) = ∅
First(\( M \)) \cap First(\( (a \) = ∅
First(\( M \)) \cap First(\( ε \) = ∅

No overlap, grammar is predictive
Consider the grammar

```
E -> E + T | T
T -> a | ( E )
```

**Example 5 – Computing First Sets**

**Consider the grammar**

```
E -> E + T | T
T -> a | ( E )
```

**Compute First Sets**

First(E) = { ( ) for now
First(T) = { ( ) for now
First(a) = { ( a )
First( E ) = { ( )
First( T ) = { ( a )
First( ( ) = { ( , a )}
```

**Example 5 – Recomputing First Sets**

**Consider the grammar**

```
E -> E + T | T
T -> a | ( E )
```

**f) Compute First Sets**

First(a) = { ( a )
First( E ) = { ( )
```

**Example 5 – Recursive Descent Parser**

**Grammar**

```
E -> E + T | T
T -> a | ( E )
```

**parse_L()**

```
if (lookahead == "a") {
  // L -> a
  return parse_L();
}
else {
  // L -> E
  return parse_E();
}
```

**Recursive descent parser**

```
parse_E() {
  // E -> E + T
  if (lookahead == "+") {
    // E -> E + T
    return First(E) + parse_L();
  } else {
    // E -> ( E )
    return First(E) + parse_L();
  }
}
```

```c
// L -> a
return parse_L();
```
Example 5 – Parsing Input

<table>
<thead>
<tr>
<th>Grammar</th>
<th>Parse &quot;a+a+a&quot;</th>
<th>Remaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>E → T L</td>
<td>parse_E()</td>
<td>&quot;a+a+a&quot;</td>
</tr>
<tr>
<td>L → + T L</td>
<td>parse_T()</td>
<td>&quot;a+a+a&quot;</td>
</tr>
<tr>
<td>T → a</td>
<td>match(&quot;a&quot;)</td>
<td>&quot;a+a+a&quot;</td>
</tr>
<tr>
<td></td>
<td>parse_L()</td>
<td>&quot;a+a+a&quot;</td>
</tr>
<tr>
<td></td>
<td>match(&quot;+&quot;)</td>
<td>&quot;a+a+a&quot;</td>
</tr>
<tr>
<td></td>
<td>parse_T()</td>
<td>&quot;a+a+a&quot;</td>
</tr>
<tr>
<td></td>
<td>match(&quot;a&quot;)</td>
<td>&quot;a+a+a&quot;</td>
</tr>
<tr>
<td></td>
<td>parse_L()</td>
<td>&quot;a+a+a&quot;</td>
</tr>
<tr>
<td></td>
<td>match(&quot;+&quot;)</td>
<td>&quot;a+a+a&quot;</td>
</tr>
<tr>
<td></td>
<td>parse_T()</td>
<td>&quot;a+a+a&quot;</td>
</tr>
<tr>
<td></td>
<td>match(&quot;a&quot;)</td>
<td>&quot;a+a+a&quot;</td>
</tr>
<tr>
<td></td>
<td>parse_L()</td>
<td>&quot;a+a+a&quot;</td>
</tr>
</tbody>
</table>

Lookahead in red

Example 5 – A Slightly Different Grammar

Consider the grammar

\[ E \rightarrow T + E \mid T \]

\[ T \rightarrow a \mid (E) \]

Vs. previous grammar

\[ E \rightarrow E + T \mid T \]

\[ T \rightarrow a \mid (E) \]

a) Can the grammar be parsed by a predictive parser?
   No, since First(T+E) \& First(T) \& \emptyset

b) Would grammar accept same language?
   Yes – all sums of a's

c) What is the difference between this grammar and the previous grammar rewritten to eliminate left recursion?
   This grammar is right associative
   Previous grammar is left associative