CS 536 Announcements for Thursday, March 24, 2022

Homework 3 has been released
Programming Assignment 3 – due Monday, March 28
Midterm 2 – Wednesday, March 30

Last Time
- review grammar transformations
- building a predictive parser
- FIRST and FOLLOW sets

Today
- FIRST and FOLLOW sets
- building a predictive parser
- predictive parsing and syntax-directed translation

Next Time
- review, static semantic analysis

Recap of where we are

Predictive parser builds the parse tree top-down
- 1 token lookahead
- parse(selector) table
- stack tracking current parse tree's frontier

Building the parse table – given production $lhs \rightarrow rhs$, determine what terminals would lead us to choose that production

$\text{figure out } T \text{ so that } \text{table}[lhs][T] = rhs$

FIRST($\alpha$) = \{ $T \mid (T \in \Sigma \land \alpha =>^* T\beta) \lor (T = \varepsilon \land \alpha =>^* \varepsilon)$ \}

FOLLOW($a$) = \{ $T \mid (T \in \Sigma \land s =>^* aaT\beta) \lor (T = \text{EOF} \land s =>^* aa)$ \}

Look at FIRST(ac) & FIRST(ba)
If D is in neither, look at FOLLOW(s) & ba =>^* \varepsilon
FIRST and FOLLOW sets

FIRST(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 is in FIRST(β) or β is empty

Note that
FIRST sets
- only contain alphabet terminals and ε
- defined for arbitrary RHS and nonterminals
- constructed by started at the beginning of a production

FOLLOW sets
- only contain alphabet terminals and EOF
- defined for nonterminals only
- constructed by jumping into production

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 sets to fill parse table for each production

Building the parse table

```plaintext
for each production x → α { 
  for each terminal T in FIRST(α) { 
    put α in table[x][T] 
  } 
  if ε is in FIRST(α) { 
    for each terminal T in FOLLOW(x) { 
      put α in table[x][T] 
    } 
  } 
}
```
Example

Original CFG

\[
\begin{align*}
\text{expr} & \rightarrow \text{expr} + \text{term} \\
| & \text{term} \\
\text{term} & \rightarrow \text{term} \times \text{factor} \\
| & \text{factor} \\
\text{factor} & \rightarrow \text{exponent} \times \text{factor} \\
| & \text{exponent} \\
\text{exponent} & \rightarrow \text{INTLIT} \\
| & (\text{expr})
\end{align*}
\]

Transformed CFG

\[
\begin{align*}
\text{expr} & \rightarrow \text{term expr'} \\
\text{expr'} & \rightarrow + \text{term expr'} | \varepsilon \\
\text{term} & \rightarrow \text{factor term'} \\
\text{term'} & \rightarrow * \text{factor term'} | \varepsilon \\
\text{factor} & \rightarrow \text{exponent factor'} \\
\text{factor'} & \rightarrow ^* \text{factor} | \varepsilon \\
\text{exponent} & \rightarrow \text{INTLIT} | (\text{expr})
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>\text{FIRST}</th>
<th>\text{FOLLOW}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{expr}</td>
<td>\text{INTLIT}</td>
<td>\text{EOF}</td>
</tr>
<tr>
<td>\text{expr'}</td>
<td>+, \varepsilon</td>
<td>\text{EOF}</td>
</tr>
<tr>
<td>\text{term}</td>
<td>\text{INTLIT}</td>
<td>+, \text{EOF}</td>
</tr>
<tr>
<td>\text{term'}</td>
<td>* , \varepsilon</td>
<td>+, \text{EOF}</td>
</tr>
<tr>
<td>\text{factor}</td>
<td>\text{INTLIT}</td>
<td>*, \text{EOF}</td>
</tr>
<tr>
<td>\text{factor'}</td>
<td>^, \varepsilon</td>
<td>*, \text{EOF}</td>
</tr>
<tr>
<td>\text{exponent}</td>
<td>\text{INTLIT}</td>
<td>*, \text{EOF}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>\text{FIRST}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{expr}</td>
<td>\text{term expr'}</td>
</tr>
<tr>
<td>\text{expr'}</td>
<td>+ \text{term expr'}</td>
</tr>
<tr>
<td>\text{term}</td>
<td>\text{factor term'}</td>
</tr>
<tr>
<td>\text{term'}</td>
<td>* \text{factor term'}</td>
</tr>
<tr>
<td>\text{factor}</td>
<td>\text{exponent factor'}</td>
</tr>
<tr>
<td>\text{factor'}</td>
<td>^ \text{factor}</td>
</tr>
<tr>
<td>\text{exponent}</td>
<td>\text{INTLIT}</td>
</tr>
<tr>
<td>\text{exponent}</td>
<td>(\text{expr})</td>
</tr>
</tbody>
</table>
Example

CFG

\[
\begin{align*}
  s & \rightarrow \textcolor{red}{a\ C} \mid \textcolor{green}{ba} \\
  a & \rightarrow \textcolor{blue}{AB} \mid \textcolor{purple}{Cs} \\
  b & \rightarrow \textcolor{orange}{D} \mid \varepsilon
\end{align*}
\]

FIRST and FOLLOW sets

<table>
<thead>
<tr>
<th></th>
<th>FIRST sets</th>
<th>FOLLOW sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s)</td>
<td>(A\ C D)</td>
<td>(\text{EOF}\ C)</td>
</tr>
<tr>
<td>(a)</td>
<td>(A\ C)</td>
<td>(C\ \text{EOF})</td>
</tr>
<tr>
<td>(b)</td>
<td>(D\ \varepsilon)</td>
<td>(A\ C)</td>
</tr>
</tbody>
</table>

| \(s \rightarrow a\ C\) | \(A\ C\) |
| \(s \rightarrow b\ a\) | \(D\ \text{AC}\) |
| \(a \rightarrow A\ B\) | \(A\) |
| \(a \rightarrow C\ s\) | \(C\) |
| \(b \rightarrow D\) | \(D\) |
| \(b \rightarrow \varepsilon\) | \(E\) |

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[\(x][T\)]

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
& A & B & C & D & \text{EOF} \\
\hline
s & aC, ba & & aC, ba & ba & \\
\hline
a & AB & & & & \\
\hline
b & \varepsilon & & \varepsilon & D & \\
\hline
\end{array}
\]

\(\text{not LL(1)}\)
Parsing and syntax-directed translation

Instead of building parse tree, give parser a second, \textit{semantic} stack

- holds nonterminals' translations

SDT \textit{rules} are converted to \textit{actions}

- pop translations of RHS nonterminals
- push computed translation of LHS nonterm on

Augment the parsing algorithm

- number the actions
- when RHS of production is pushed onto symbol stack, include the actions
- when action is the top of symbol stack, pop & perform the action

Placing the \textit{action numbers} in the productions

- action numbers go
  - \textit{after} their corresponding \textit{non-terminal}
  - \textit{before} their corresponding \textit{terminal}

Note: translations are popped from the semantic stack right-to-left
Example

CFG:

expr → expr + term #1
  | term

term → term * factor #2
  | factor

factor → #3 INTLIT
  | ( expr )

Translation goal: evaluate the expression, e.g., 5 + 2*3 produces 11

#1 : termTrans = pop()
exprTrans = pop()
push(exprTrans + termTrans)

#2 : factorTrans = pop()
termTrans = pop()
push(termTrans * factorTrans)

#3 : push(INTLIT.val)

Converting to LL(1)

CFG:

expr → expr + term #1
  | term

term → term * factor #2
  | factor

factor → #3 INTLIT
  | ( expr )

Then compute FIRST & FOLLOW sets
- treating action #& like E

and build parse table
Example SDT on transformed grammar

CFG:

expr → term expr'
expr' → + term #1 expr' | ε
term → factor term'
term' → * factor #2 term' | ε
factor → #3 INTLIT | ( expr )

SDT actions:

#1 : termTrans = pop()
   exprTrans = pop()
   push(exprTrans + termTrans)
#2 : factorTrans = pop()
   termTrans = pop()
   push(termTrans * factorTrans)
#3 : push(INTLIT.val)

Parse table

<table>
<thead>
<tr>
<th></th>
<th>+</th>
<th>*</th>
<th>(</th>
<th>)</th>
<th>INTLIT</th>
<th>EOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>expr'</td>
<td>term</td>
<td>expr'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>expr'</td>
<td>+</td>
<td>term</td>
<td>#1</td>
<td>expr'</td>
<td>ε</td>
<td>ε</td>
</tr>
<tr>
<td>term</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>factor</td>
<td>term'</td>
</tr>
<tr>
<td>term'</td>
<td>ε</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>factor</td>
</tr>
<tr>
<td>factor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(</td>
<td>expr</td>
</tr>
</tbody>
</table>

Input: 5 + 3 * 2 EOF
What about ASTs?

Push and pop AST nodes on the semantic stack
Keep references to nodes that we pop

CFG:

- `expr → term expr'`
- `expr' → + term #1 expr' | ε`
- `term → factor term'`
- `term' → * factor #2 term' | ε`
- `factor → #3 INTLIT | ( expr )`

SDT actions:

- `#1 : termTrans = pop() exprTrans = pop() push(new PlusNode(exprTrans, termTrans))`
- `#2 : factorTrans = pop() termTrans = pop() push(new IntLitNode(INTLITival))`
- `#3 : push(new IntLitNode(INTLITival))`

Input: 1 + 2 + 3 EOF

Associativity fixed!