Lecture 9 Outline

• Third Phase: Semantic Analysis and Parsing
• Finish Parsing

Reading Assignment:
Sections: 7.6, 4.8, 4.9, 5.1

Semantic Actions during Parsing

• We can now illustrate how semantic actions are implemented for LR parsing
• Keep attributes on the semantic stack - parallel syntax stack
• On shift a, push attribute for a on semantic stack
• On reduce X → α
  - pop attributes for α
  - compute attribute for X
  - and push it on the semantic stack

Performing Semantic Actions, Example

• Recall the example
  E → T * E1  { E.val = T.val * E1.val }
  | T       { E.val = T.val }
  T → int * T1  { T.val = int.val * T1.val }
  | int     { T.val = int.val }

• Consider the parsing of the string 3 * 5 + 8
Performing Semantic Actions, Example

| int * int + int | shift |
| int3 * int + int | shift |
| int3 * | int + int | shift |
| int3 * int5 | + int | reduce T → int |
| T15 | + int | shift |
| T15 * | int | shift |
| T15 * int8 | + int | reduce T → int |
| T15 + T8 | reduce E → T |
| T15 + E8 | reduce E → T + E |
| E23 | accept |

Using Parser Generators

- Most common parser generators are LALR(1)
- A parser generator constructs a LALR(1) table
- And reports an error when a table entry is multiply defined:
  - A shift and a reduce. Called shift/reduce conflict
  - Multiple reduces. Called reduce/reduce conflict
- An ambiguous grammar will generate conflicts
- Must resolve conflicts

Shift/Reduce Conflicts

- Typically due to ambiguities in the grammar
- Classic example: the dangling else
  S → if E then S | if E then S else S | OTHER
- Will have DFA state containing
  [S → if E then S, else]  [S → if E then S, else S, x]
- if else follows then we can shift or reduce
- Default (yacc, bison, CUP, etc.) is to shift
  - Default behavior is as needed in this case
More Shift/Reduce Conflicts

Consider the ambiguous grammar:

\[ E \rightarrow E + E \mid E \times E \mid \text{int} \]

We will have the states containing:

\[
\begin{align*}
[E \rightarrow E \times E, +] & \quad [E \rightarrow E \times E, +] \\
[E \rightarrow E + E, +] & \quad [E \rightarrow E + E, +]
\end{align*}
\]

Again we have a shift/reduce on input +
- We need to reduce (\(\times\) is higher than +)
- Recall solution: declare the precedence of \(\times\) and +

In Bison declare precedence and associativity:

```bison
%left +
%left *
```

Precedence of a rule = that of its last terminal
- See bison manual for ways to override this default

Resolve shift/reduce conflict with a shift if:
- no precedence declared for either rule or terminal
- input terminal has higher precedence than the rule
- the precedences are the same and right associative

Using Precedence to Solve S/R Conflicts

Back to example:

\[
\begin{align*}
[E \rightarrow E \times E, +] & \quad [E \rightarrow E \times E, +] \\
[E \rightarrow E + E, +] & \quad [E \rightarrow E + E, +]
\end{align*}
\]

Will choose reduce because precedence of rule \(E \rightarrow E \times E\)
is higher than of terminal +
Using Precedence to Solve S/R Conflicts

- Same grammar as before
  \[ E \rightarrow E + E | E * E | \text{int} \]
- We will also have the states
  \[ \{ E \rightarrow E \cdot E, + \} \]
  \[ \{ E \rightarrow E + E, \cdot \} \]
  \[ \{ E \rightarrow E \cdot E, + \} \Rightarrow \{ E \rightarrow E \cdot E, + \} \]
- Now we also have a shift/reduce on input +
  - We choose reduce because \( E \rightarrow E + E \) and + have the same precedence and + is left-associative

Using Precedence to Solve S/R Conflicts

- Back to our dangling else example
  \[ S \rightarrow \text{if } E \text{ then } S \cdot \text{ else} \]
  \[ S \rightarrow \text{if } E \text{ then } S \cdot \text{ else } S \cdot \text{ x} \]
- Can eliminate conflict by declaring else with higher precedence than then
- But this starts to look like "hacking the tables"
- Best to avoid overuse of precedence declarations or you'll end with unexpected parse trees

Reduce/Reduce Conflicts

- Usually due to ambiguity in the grammar
- Example: a sequence of identifiers
  \[ S \rightarrow \epsilon | \text{id} | \text{id } S \]
- There are two parse trees for the string id
  \[ S \rightarrow \text{id} \]
  \[ S \rightarrow \text{id } S \rightarrow \text{id} \]
- How does this confuse the parser?
More on Reduce/Reduce Conflicts

- Consider the states
  \[ S \to id ., \] \[ S \to ., S \] \[ S \to ., S \to id \] \[ S \to . id, \] \[ S \to . id S, S \to id \] \[ S \to ., S \to id \] \[ S \to . id, \] \[ S \to . id S, S \to id \]
- Reduce/reduce conflict on input $ S' \to S \to id S' \to S \to id S \to id$
- Better rewrite the grammar: $ S \to \epsilon | id S$

Compaction of LR parsing tables

A typical language grammar with 50-100 terminals and 100 productions may have an LALR parser with several hundred states & thousands of action entries

- often row of table are the same so share the rows
- rows can be made shorter - use lists (input, action)
- slows access to the table

Error recovery in LR parsing

Error detected when parser consults parsing table - empty entry
Canonical LR will not make a single reduction before announcing an error
SLR & LALR parser may make several reductions before announcing an error but not shift an erroneous symbol on the stack
Simple error recovery:
- continue to scan down the stack until a state $S$ with a
goto on a particular non-terminal $A$ is found
- zero or more input symbols are discarded until a
symbol 'a' is found that can follow $A$
goto $[s,A]$ put on stack & parsing continues
Choice of A - non-terminal representing major program piece
e.g. if $A = stmt$ then 'a' may be 'end' or ';

Notes on Parsing

- Parsing
  - A solid foundation: context-free grammars
  - A simple parser: LL(1)
  - A more powerful parser: LR(1)
  - An efficiency hack: LALR(1)
  - LALR(1) parser generators