CS 4120
Introduction to Compilers
Ross Tate
Cornell University
Lecture 7: LR parsing and parser generators

Shift-reduce parsing

<table>
<thead>
<tr>
<th>derivation</th>
<th>stack</th>
<th>input stream</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+2+(3+4)+5 ← (1+2+(3+4))+5</td>
<td>shift</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1+2+(3+4)+5 ← (1+2+(3+4))+5</td>
<td>shift</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1+2+(3+4)+5 ← (1+2+(3+4))+5</td>
<td>reduce E → n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E+2+(3+4))+5 ← (E+2+(3+4))+5</td>
<td>shift</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E+2+(3+4))+5 ← (E+2+(3+4))+5</td>
<td>reduce S → E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S+2+(3+4))+5 ← (S+2+(3+4))+5</td>
<td>shift</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S+2+(3+4))+5 ← (S+2+(3+4))+5</td>
<td>reduce E → n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S+E+2+(3+4))+5 ← (S+E+2+(3+4))+5</td>
<td>reduce S → E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S+(3+4))+5 ← (S+(3+4))+5</td>
<td>shift</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S+(3+4))+5 ← (S+(3+4))+5</td>
<td>reduce E → n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S+(3+4))+5 ← (S+(3+4))+5</td>
<td>reduce E → n</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

LR(0) states
- A state is a set of items keeping track of progress on possible upcoming reductions
- An LR(0) item is a production from the language with a separator “." somewhere in the RHS of the production
- Stuff before “.” is already on stack (beginnings of possible γ’s to be reduced)
- Stuff after “.” : what we might see next
- The prefixes α represented by state itself

LR(1) state
- LR(1) state = set of LR(1) items
- LR(1) item = LR(0) item + 1 lookahead

LR(1) closure
Consider closure of item $A \rightarrow \beta . C \delta \lambda$
Closure formed just as for LR(0) except
1. Lookahead symbols include characters following the non-terminal symbol to the right of dot: FIRST(δ)
2. If non-terminal symbol may produce last symbol of production (δ is nullable), lookahead symbols include lookahead symbols of production (λ)
LR(1) construction

- Problem with LR(1): too many states
- LALR(1) (Look-Ahead LR)
  - Merge any two LR(1) states whose items are identical except for look-ahead
  - Results in smaller parser tables—works extremely well in practice
  - Common technology for automatic parser generators

LALR grammars

LALR(1) vs. LR(1)

How are parsers written?

- Automatic parser generators: yacc, bison, CUP
- Accept LALR(1) grammar specification
  - plus: declarations of precedence, associativity
  - output: LR parser code (inc. parsing table)
- Some parser generators accept LL(1), e.g. javacc – less powerful, or LL(k), e.g. ANTLR
Resolving Ambiguity

\[ E \rightarrow E + E \mid E * E \]

When reducing op conflicts with shifting a production containing op
- choose reduce = left assoc.
- choose shift = right assoc.

When reducing op conflicts with shifting a production containing op'
- choose reduce = op \text{<} op'
- choose shift = op \text{>} op'

Can we use parsers for programs \textit{besides} compilers?