Compiler construction
in4020 – lecture 4

Koen Langendoen
Delft University of Technology
The Netherlands

Overview
• syntax analysis: tokens → AST
  - bottom-up parsing
  - push-down automaton
  - ACTION/GOTO tables
  - LR(0), SLR(1), LR(1), LALR(1)

Bottom-up (LR) parsing
• Left-to-right parse, Rightmost-derivation
• create a node when all children are present
• handle: nodes representing the right-hand side of a production

LR(0) parsing
• running example: expression grammar
  - short-hand notation

LR(0) parsing
• keep track of progress inside potential handles when consuming input tokens
• LR items: \[ N \rightarrow \alpha \bullet \beta \]
• initial set
• ε-closure: expand dots in front of non-terminals
LR(0) parsing

stack | input
--- | ---
S0    | i + i $

- shift input token (i) onto the stack
- compute new state

LR(0) parsing

stack | input
--- | ---
S0 S1 | + i $

- reduce handle on top of the stack
- compute new state

LR(0) parsing

stack | input
--- | ---
S0 T S2 | + i $

- reduce handle on top of the stack
- compute new state

LR(0) parsing

stack | input
--- | ---
S0 E S3 | + i $

- shift input token on top of the stack
- compute new state

LR(0) parsing

stack | input
--- | ---
S0 E S3 + S4 | $ 

- shift input token on top of the stack
- compute new state

LR(0) parsing

stack | input
--- | ---
S0 E S3 + S4 S1 | $ 

- reduce handle on top of the stack
- compute new state
**LR(0) parsing**

- reduce handle on top of the stack
- compute new state

**LR(0) parsing**

- shift input token on top of the stack
- compute new state

**LR(0) parsing**

- reduce handle on top of the stack
- compute new state

**LR(0) parsing**

- accept!

**Transition diagram**

**Exercise (8 min.)**

- complete the transition diagram for the LR(0) automaton
- can you think of a single input expression that causes all states to be used? If yes, give an example. If no, explain.
Answers

The LR tables

<table>
<thead>
<tr>
<th>state</th>
<th>GOTO table</th>
<th>ACTION table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>i</td>
</tr>
<tr>
<td>2</td>
<td>E → T</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>shift</td>
</tr>
<tr>
<td>5</td>
<td>E → E + T</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Z → E $</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>T → ( E )</td>
<td></td>
</tr>
</tbody>
</table>

LR(0) parsing concise notation

<table>
<thead>
<tr>
<th>stack</th>
<th>input</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>i</td>
<td>shift</td>
</tr>
<tr>
<td>S0</td>
<td>i S1</td>
<td>+ i $</td>
</tr>
<tr>
<td>S0 T S2</td>
<td>+ i $</td>
<td>reduce by E → T</td>
</tr>
<tr>
<td>S0 E S3</td>
<td>+ i $</td>
<td>shift</td>
</tr>
<tr>
<td>S0 E S3 + S4</td>
<td>i $</td>
<td>shift</td>
</tr>
<tr>
<td>S0 E S3 + S4 i S1</td>
<td>$</td>
<td>reduce by T → i</td>
</tr>
<tr>
<td>S0 E S3 + S4 T S5</td>
<td>$</td>
<td>reduce by E → E + T</td>
</tr>
<tr>
<td>S0 E S3</td>
<td>$</td>
<td>shift</td>
</tr>
<tr>
<td>S0 E S3 + S6</td>
<td>reduce by Z → E $</td>
<td></td>
</tr>
<tr>
<td>S0 Z</td>
<td>accept</td>
<td></td>
</tr>
</tbody>
</table>

The LR push-down automaton

SWITCH action_table[top of stack]:

CASE "shift":
see book;

CASE ("reduce", N → α):
POE the symbols of α FROM the stack;
SET state TO top of stack;
PUSH new state ON the stack;
CASE empty:
ERROR;

LR(0) conflicts

- shift-reduce conflict
- array indexing: T → i [ E ]
  T → i [ E ] (shift)
  T → i (reduce)
- ε-rule: RestExpr → ε
  Expr → Term • RestExpr (shift)
  RestExpr → • (reduce)

LR(0) conflicts

- reduce-reduce conflict
- assignment statement: Z → V := E $
  V → i (reduce)
  T → i (reduce)
- typical LR(0) table contains many conflicts
Handling LR(0) conflicts

- **solution**: use a one-token look-ahead two-dimensional ACTION table [state,token]
- different construction of ACTION table
  - SLR(1) – Simple LR
  - LR(1)
  - LALR(1) – Look-Ahead LR

SLR(1) parsing

- solves (some) shift-reduce conflicts
- **reduce** $N \rightarrow \alpha$ iff token $\in$ FOLLOW($N$)

FOLLOW($T$) = \{'+', ')', $\}'
FOLLOW($E$) = \{'+', ')', $\}'
FOLLOW($Z$) = \{$\}$

SLR(1) ACTION table

<table>
<thead>
<tr>
<th>state</th>
<th>look-ahead token</th>
<th>i</th>
<th>+</th>
<th>(</th>
<th>)</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>shift</td>
<td>s1</td>
<td>s7</td>
<td>s3</td>
<td>s2</td>
<td>p</td>
</tr>
<tr>
<td>1</td>
<td>T \rightarrow i</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>r4</td>
<td>s4</td>
</tr>
<tr>
<td>2</td>
<td>E \rightarrow T</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>r2</td>
<td>s6</td>
</tr>
<tr>
<td>3</td>
<td>shift</td>
<td>s4</td>
<td>s4</td>
<td>s4</td>
<td>s4</td>
<td>s8</td>
</tr>
<tr>
<td>4</td>
<td>shift</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>r3</td>
<td>s5</td>
</tr>
<tr>
<td>5</td>
<td>E \rightarrow E + T</td>
<td>s7</td>
<td>s7</td>
<td>s7</td>
<td>s7</td>
<td>s2</td>
</tr>
<tr>
<td>6</td>
<td>Z \rightarrow E $</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>r1</td>
<td>s1</td>
</tr>
<tr>
<td>7</td>
<td>shift</td>
<td>s4</td>
<td>s4</td>
<td>s4</td>
<td>s4</td>
<td>s6</td>
</tr>
<tr>
<td>8</td>
<td>T \rightarrow ( E )</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>s9</td>
</tr>
<tr>
<td>9</td>
<td>T \rightarrow ( E )</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>r5</td>
<td>s9</td>
</tr>
</tbody>
</table>

Unfortunately ...

- SLR(1) leaves many shift-reduce conflicts unsolved
- problem: FOLLOW($N$) set is a union of all contexts in which $N$ may occur
- example
  
  \[
  S \rightarrow A \mid x \ b \\
  A \rightarrow a \ A \ b \mid x
  \]
Exercise (6 min.)

- derive the SLR(1) ACTION/GOTO table (with shift-reduce conflict) for the following grammar:

\[
S \rightarrow A \mid x b \\
A \rightarrow a A b \\
A \rightarrow x \\
S \rightarrow x b \\
A \rightarrow x
\]

Answers

LR(1) parsing

- maintain follow set per item
  LR(1) item: \( N \rightarrow \alpha \bullet \beta \{ \sigma \} \)
- \( \varepsilon \) - closure for LR(1) item sets:
  
  if set \( S \) contains an item \( P \rightarrow \alpha \bullet N \beta \{ \sigma \} \) then
  
  foreach production rule \( N \rightarrow \gamma \)
  
  S must contain the item \( N \rightarrow \star \gamma \{ \tau \} \)
  
  where \( \tau = \text{FIRST}(\beta \{\sigma\}) \)

LALR(1) parsing

- LR tables are big
- combine “equal” sets by merging look-ahead sets
### LALR(1) automaton

```
S0  S → A* ($)  
S1  A → x* ($)  
S2  A → x* b* ($)  
S3  S → A + ($)  
S4  A → x A b* ($)  
S5  A → x A b* ($)  
S6  A → x A b* ($)  
S7  A → x A b* ($)  
```

### LALR(1) ACTION/GOTO table

<table>
<thead>
<tr>
<th>state</th>
<th>stack symbol / look-ahead token</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a b x $ A</td>
</tr>
<tr>
<td>1</td>
<td>s4 s2 r4</td>
</tr>
<tr>
<td>2</td>
<td>r2</td>
</tr>
<tr>
<td>3</td>
<td>r1</td>
</tr>
<tr>
<td>4</td>
<td>s4 s5 s6</td>
</tr>
<tr>
<td>5</td>
<td>r4</td>
</tr>
<tr>
<td>6</td>
<td>s7</td>
</tr>
<tr>
<td>7</td>
<td>r3</td>
</tr>
</tbody>
</table>

### Making grammars LR(1) – or not?

- grammars are often ambiguous
  
  \[ E \rightarrow E \cdot E \mid E \cdot E \mid i \]

- handle shift-reduce conflicts
  - (default) longest match: shift
  - precedence directives

- syntax analysis: tokens → AST

- bottom-up parsing
  - push-down automaton
  - ACTION/GOTO tables
    - LR(0) NO look-ahead
    - SLR(1) one-token look-ahead, FOLLOW sets to solve shift-reduce conflicts
    - LR(1) SLR(1), but FOLLOW set per item
    - LALR(1) LR(1), but "equal" states are merged

### Summary

- study sections:
  - 1.10 closure algorithm
  - 2.2.5.8 error handling in LR(1) parsers
- print handout for next week [blackboard]
- find a partner for the “practicum”
- register your group
- send e-mail to koen@pds.twi.tudelft.nl