Formal Languages and Compilers
Lecture VI—Part 4: Syntactic Analysis

Alessandro Artale

Free University of Bozen-Bolzano
Faculty of Computer Science – POS Building, Room: 2.03
artale@inf.unibz.it
http://www.inf.unibz.it/∼artale/

Formal Languages and Compilers — BSc course

2020/21 – Second Semester

Summary of Lecture VI—Part 4

- Inadequacy of SLR Parsing
- Dealing with Ambiguous Grammars - How to interact with YACC

When SLR is not Adequate

Example. Consider the following Grammar and the Canonical Collection:

Augmented Grammar

$$
\begin{array}{lcl}
S' & \to & S \\
S & \to & L = R \\
S & \to & R \\
L & \to & *R \\
L & \to & \mathrm{id} \\
R & \to & L
\end{array}
$$

$$
\begin{array}{lcll}
I_0: & S' & \to & .S \\
 & S & \to & .L = R \\
 & S & \to & .R \\
 & L & \to & .*R \\
 & L & \to & .\,\mathrm{id} \\
 & R & \to & .L \\
I_1: & S' & \to & S. \\
I_2: & S & \to & L. = R \\
 & R & \to & L. \\
I_3: & S & \to & R. \\
I_4: & L & \to & *.R \\
 & R & \to & .L \\
 & L & \to & .*R \\
 & L & \to & .\,\mathrm{id}
\end{array}
$$

$$
\begin{array}{lcll}
I_5: & L & \to & \mathrm{id}. \\
I_6: & S & \to & L =. R \\
 & R & \to & .L \\
 & L & \to & .*R \\
 & L & \to & .\,\mathrm{id} \\
I_7: & L & \to & *R. \\
I_8: & R & \to & L. \\
I_9: & S & \to & L = R.
\end{array}
$$

When SLR is not Adequate (Cont.)

Consider now the set of items in I_2:

1. If we consider the first item, then, $action[2, =] = shift\ 6$.
2. If we consider the second item, since "="$\in$ Follow(R), then, $action[2, =] = reduce\ R \rightarrow L$.

- Shift/Reduce Conflict. There is both a shift and a reduce action thus the entry for $action[2, =]$ is multiply defined!
- The Grammar is not ambiguos, thus:
  The SLR parsing technique is not powerful enough!

Towards LR(1) Parsing: A Solution to SLR failure

- Add more information in the state to rule out invalid reductions.
- Each state of an LR(1) Parser indicates what symbol can follow a handle for which there is a possible reduction.
- **Definition.** An LR(1) Item is then a pair $[A \to \alpha.\beta, a]$, where $a \in \mathbf{V}_T$ is called the Lookahead of the Item. Furthermore, $a \in \textsc{Follow}(A)$.

LALR Parsers

- LALR stands for LookAhead-LR Parser.
- It is often used in practice since the parsing tables are considerably smaller than the canonical LR tables.
- YACC is an LALR Parser;
- **Main Idea:** Merge the states (Item sets) whose items differ only in the lookahead.
  - We say that such states have the same **core**.
- **(SLR and LALR) Vs. LR(1).** The comparison is in term of size. For a language like Pascal we go from **hundreds** of states to **thousand** of states.

Summary of Lecture IV—Part 4

- Inadequacy of SLR Parsing
- Dealing with Ambiguous Grammars - How to interact with YACC

Ambiguous Grammars and LR Conflicts

- Ambiguous Grammars provide a shorter and more natural specification.
- This is reflected by a parser requiring fewer number of states.
- Ambiguous Grammars are not LR neither LL.
- A parsing table will have **multiply-defined** entries called **conflicts**.

  It will contain REDUCE/REDUCE and/or SHIFT/REDUCE conflicts.
- To deal with ambiguous grammars we use disambiguating rules that eliminate all the conflicts allowing for only one Parse-Tree.

Precedence and Associativity Rules

- The following ambiguous Grammar for arithmetic expressions does not specify the associativity and the precedence of $+, *$:

$$E \to E + E \mid E * E \mid (E) \mid \text{id}$$

- The unambiguous Grammar:

$$\begin{array}{lcl} E & \to & E+T \mid T \\ T & \to & T*F \mid F \\ F & \to & (E) \mid \text{id} \end{array}$$

  Enforces precedence of $*$ over $+$, and left-associativity of $+$ and $*$.
- **Note.** The parser for the unambiguous Grammar is less efficient:
  - The time spent for reducing the *unit productions* $E \to T$ and $T \to F$ has the sole function to enforce precedence.

Precedence and Associativity Rules (Cont.)

- Let us consider the following LR(0) item set for the ambiguous augmented Grammar (see the Book, Sect. 4.8, for the complete set of states):

$$
\begin{array}{lrcl}
I_7: & E & \rightarrow & E+E. \\
 & E & \rightarrow & E.+E \\
 & E & \rightarrow & E.*E
\end{array}
\qquad
\begin{array}{lrcl}
I_8: & E & \rightarrow & E*E. \\
 & E & \rightarrow & E.+E \\
 & E & \rightarrow & E.*E
\end{array}
$$

  $\textsc{Follow}(E) = \{\$, +, *\}$

- State I_7 generates a conflict between "**reduce** with $E \rightarrow E + E$", and "**shift** with input $+$ and $*$".
- State I_8 generates a conflict between "**reduce** with $E \rightarrow E * E$", and "**shift** with input $+$ and $*$".
- *Both Conflicts can be solved using the Precedence and Associativity for the operations $+$ and $*$.*

Precedence and Associativity Rules (Cont.)

- Consider the input `id + id * id` and the following configuration:

Stack	Input
$0E1+4E7$	$*$ `id` \$

- Assuming that $*$ takes precedence over $+$, the parser should **shift** $*$ onto the stack instead of **reducing** with $E \rightarrow E + E$.

Precedence and Associativity Rules (Cont.)

- Consider the input `id + id + id` and the following configuration:

Stack	Input
$0E1+4E7$	`+ id $`

- Assuming that $+$ *is left-associative*, the parser should **reduce** with $E \rightarrow E + E$ instead of **shift** $+$ onto the stack.

Precedence and Associativity Rules (Cont.)

- **Summary.** Precedence and Associativity uniquely determine the actions of the parser eliminating the ambiguity:
  1. *Left-Associativity* of $+$: $action(7, +) =$ **reduce** with $E \rightarrow E + E$
  2. *Left-Associativity* of $*$: $action(8, *) =$ **reduce** with $E \rightarrow E * E$
  3. *Precedence* of $*$ over $+$: $action(7, *) =$ **shift** $*$
  4. *Precedence* of $*$ over $+$: $action(8, +) =$ **reduce** with $E \rightarrow E * E$

Interacting with YACC

- YACC has built-in disambiguation rules, so it can parse grammars even in the presence of conflicts!!!
  1. SHIFT/REDUCE conflicts will be solved by giving preference to SHIFT.
  2. REDUCE/REDUCE conflicts will be solved by giving preference to the first grammar rule listed in the YACC file.
- YACC also provides way to overrule the above default rules and defines both precedence and associativity rules for the operators (see YACC Tutorial in the LAB).

Dangling-Else Ambiguity

- Consider the following Grammar for IF-THEN-ELSE statements:
  $S' \rightarrow S$
  $S \rightarrow iSeS \mid iS \mid a$
  Where, i stands for "if *expr* then", e stands for else, and a stands for "all other productions".
- Let us consider the following LR(0) item set for this ambiguous Grammar (see the Book for the complete set of states):
  $I_4 : \; S \rightarrow iS.eS$
  $\quad\quad S \rightarrow iS.$
  $\textsc{Follow}(S) = \{\$, \text{else}\}$
- Assuming that the stack contains: "if *expr* then *stmt*" (iS) with state 4 on top, and the next input is else there is a SHIFT/REDUCE conflict.
- **Solution.** else *must match the last* if: *action*(4, else) = "**shift** else". Since YACC solves shift/reduce conflicts in favor of shift the dangling-else is handled correctly.

Summary of Lecture VI—Part 4

- Inadequacy of SLR Parsing
- Dealing with Ambiguous Grammars - How to interact with YACC