Compiler Design

Fatemeh Deldar

Isfahan University of Technology

1403-1404

Ambiguous Grammars

- **The "Dangling-Else" Ambiguity**

$$\begin{array}{lcl} stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\ & | & \textbf{if } expr \textbf{ then } stmt \\ & | & \textbf{other} \end{array}$$

$$\begin{array}{lcl} S' & \rightarrow & S \\ S & \rightarrow & i\ S\ e\ S \mid i\ S \mid a \end{array}$$

$$\begin{array}{ll} I_0: & S' \rightarrow \cdot S \\ & S \rightarrow \cdot iSeS \\ & S \rightarrow \cdot iS \\ & S \rightarrow \cdot a \\ I_1: & S' \rightarrow S\cdot \\ I_2: & S \rightarrow i\cdot SeS \\ & S \rightarrow i\cdot S \\ & S \rightarrow \cdot iSeS \\ & S \rightarrow \cdot iS \\ & S \rightarrow \cdot a \end{array}$$

$$\begin{array}{ll} I_3: & S \rightarrow a\cdot \\ I_4: & S \rightarrow iS\cdot eS \\ & S \rightarrow iS\cdot \\ I_5: & S \rightarrow iSe\cdot S \\ & S \rightarrow \cdot iSeS \\ & S \rightarrow \cdot iS \\ & S \rightarrow \cdot a \\ I_6: & S \rightarrow iSeS\cdot \end{array}$$

- ***We should shift else, because it is "associated" with the previous then***

Ambiguous Grammars

- **The "Dangling-Else" Ambiguity**

STATE	ACTION				GOTO
	i	e	a	$\$$	S
0	s2		s3		1
1				acc	
2	s2		s3		4
3		r3		r3	
4		s5		r2	
5	s2		s3		6
6		r1		r1	

	STACK	SYMBOLS	INPUT	ACTION
(1)	0		$i\,i\,a\,e\,a\,\$$	shift
(2)	0 2	i	$i\,a\,e\,a\,\$$	shift
(3)	0 2 2	$i\,i$	$a\,e\,a\,\$$	shift
(4)	0 2 2 3	$i\,i\,a$	$e\,a\,\$$	shift
(5)	0 2 2 4	$i\,i\,S$	$e\,a\,\$$	reduce by $S \to a$
(6)	0 2 2 4 5	$i\,i\,S\,e$	$a\,\$$	shift
(7)	0 2 2 4 5 3	$i\,i\,S\,e\,a$	$\$$	reduce by $S \to a$
(8)	0 2 2 4 5 6	$i\,i\,S\,e\,S$	$\$$	reduce by $S \to iSeS$
(9)	0 2 4	$i\,S$	$\$$	reduce by $S \to iS$
(10)	0 1	S	$\$$	accept

Ambiguous Grammars

Exercise 4.8.1: The following is an ambiguous grammar for expressions with n binary, infix operators, at n different levels of precedence:

$$E \;\rightarrow\; E\;\theta_1\;E \mid E\;\theta_2\;E \mid \cdots \mid E\;\theta_n\;E \mid (\;E\;) \mid \mathbf{id}$$

a) As a function of n, what are the SLR sets of items?

b) How would you resolve the conflicts in the SLR items so that all operators are left associative, and θ_n takes precedence over θ_{n-1}, which takes precedence over θ_{n-2}, and so on?

c) Show the SLR parsing table that results from your decisions in part (b).

Error Recovery in LR Parsing

- An LR parser will detect an error when it consults the parsing **action table** and finds an error entry
  - **All empty entries in the action table are error entries**
- A **canonical LR parser (LR(1) parser)** will never make even a single reduction before announcing an error
- **The SLR and LALR parsers may make several reductions before announcing an error**
- But, all LR parsers (LR(1), LALR and SLR parsers) will never shift an erroneous input symbol onto the stack

Panic Mode Error Recovery in LR Parsing

- In LR parsing, we can implement panic-mode error recovery as follows:
  - Scan down the stack until a **state $\boldsymbol{s}$** with a goto on a particular **non-terminal $\boldsymbol{A}$** is found
  - Discard zero or more input symbols until a **symbol $\boldsymbol{a}$** is found that can legitimately follow $\boldsymbol{A}$
  - The **symbol $\boldsymbol{a}$** is simply in $\boldsymbol{FOLLOW(A)}$, but this may not work for all situations
  - The parser stacks the **non-terminal $\boldsymbol{A}$** and the state $\boldsymbol{goto[s,A]}$, and it resumes the normal parsing