Principles of Programming Languages

http://www.di.unipi.it/~andrea/Didattica/PLP-14/

Prof. Andrea Corradini
Department of Computer Science, Pisa

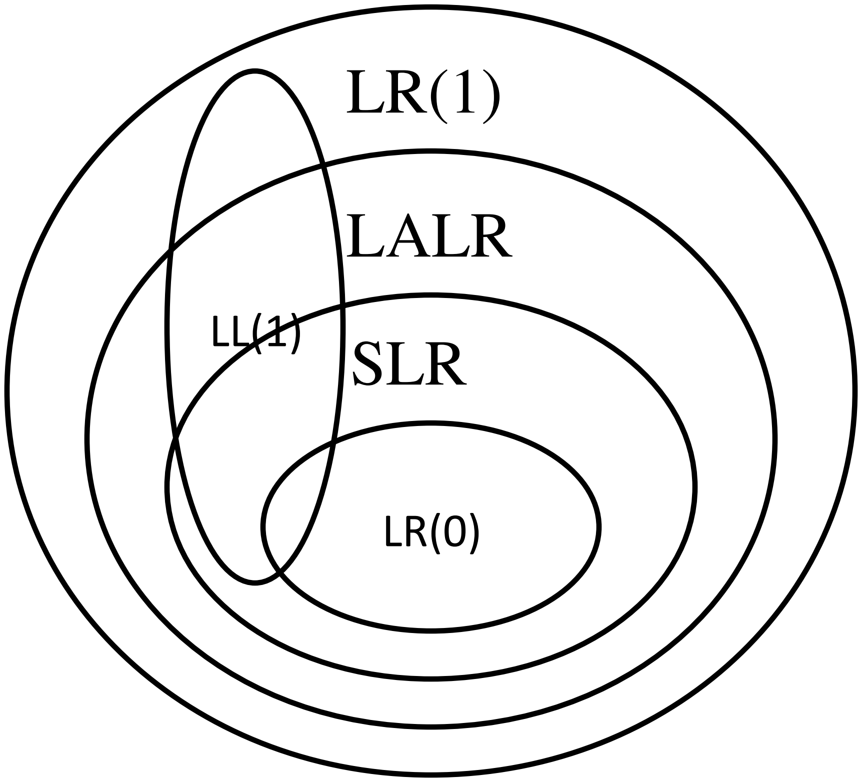
Lesson 9

- LR parsing with ambiguous grammars
- Error detection in LR parsing
- Some exercises on parsing

LL, SLR, LR, LALR Summary

- LL parse tables
  - Nonterminals × terminals → productions
  - Computed using FIRST/FOLLOW
- LR parsing tables computed using closure/goto
  - LR states × terminals → shift/reduce actions
  - LR states × nonterminals → goto state transitions
- A grammar is
  - LL(1) if its LL(1) parse table has no conflicts
  - SLR if its SLR parse table has no conflicts
  - LALR if its LALR parse table has no conflicts
  - LR(1) if its LR(1) parse table has no conflicts

LL, SLR, LR, LALR Grammars



Dealing with Ambiguous Grammars

1. $S' \rightarrow E$
2. $E \rightarrow E$ **+** E
3. $E \rightarrow$ **id**

	id	**+**	**$**	E
0	s2			1
1		s3	acc	
2		r3	r3	
3	s2			4
4		s3/r2	r2	

stack	input
$ 0	**id+id+id$**
…	…
$ 0 E 1 **+** 3 E 4	**+id$**

Shift/reduce conflict:
action[4,**+**] = shift 4
action[4,**+**] = reduce $E \rightarrow E$ **+** E

When shifting on **+**:
yields right associativity
id+(id+id)

When reducing on **+**:
yields left associativity
(id+id)+id

Using Associativity and Precedence to Resolve Conflicts

- Left-associative operators: reduce
- Right-associative operators: shift
- Operator of higher precedence on stack: reduce
- Operator of lower precedence on stack: shift

$S' \to E$
$E \to E$ **+** E
$E \to E$ ***** E
$E \to$ **id**

stack	input	
$ 0	**id*id+id$**	
…	…	
$ 0 *E* 1 ***** 3 *E* 5	**+id$**	reduce $E \to E * E$

Error Detection in LR Parsing

- Canonical LR parser uses full LR(1) parse tables and will never make a single reduction before recognizing the error when a syntax error occurs on the input
- SLR and LALR may still reduce when a syntax error occurs on the input, but will never shift the erroneous input symbol

Error Recovery in LR Parsing

- Panic mode
  - Pop until state with a goto on a nonterminal *A* is found, (where *A* represents a major programming construct), push *A*
  - Discard input symbols until one is found in the FOLLOW set of *A*
- Phrase-level recovery
  - Implement error routines for every error entry in table
- Error productions
  - Pop until state has error production, then shift on stack
  - Discard input until symbol is encountered that allows parsing to continue

Exercises on Parsing

1. Contex-free Languages strictly include Regular Language
   - Prove it by showing that $L = \{ a^n b^n \mid n > 0 \}$ is context-free but not regular
2. Consider the grammar:

   A → aB|BC

   B → bB|ε

   C→c

   - Construct the LL(1) parsing table
   - Show configurations of stack and input recognizing strings *bbc, abb*

Exercises on Parsing

- Consider the grammar augmented with a new start symbol S′ and production S′ → S:

  ① S′ → S
  ② S → A B
  ③ A → A **a**
  ④ A → ε
  ⑤ B → **b** C
  ⑥ C → **c**

a) Construct the LR(0) sets of items.

b) Construct the SLR parsing table from the LR(0) items.

c) Is the grammar LR(0)? Is it SLR?