Bottom-up Parsing

Xiaokang Qiu
Purdue University

LR Parsers

- Parser which does a Left-to-right, Right-most derivation
  - Rather than parse top-down, like LL parsers do, parse bottom-up, starting from leaves
- Basic idea: put tokens on a stack until an entire production is found

LR Parsers

- Basic idea:
  - **shift** tokens onto the stack. At any step, keep the set of productions that could generate the read-in tokens
  - **reduce** the RHS of recognized productions to the corresponding non-terminal on the LHS of the production. Replace the RHS tokens on the stack with the LHS non-terminal.

Data structures

- At each state, given the next token,
  - A *goto table* defines the successor state
  - An *action table* defines whether to
    - *shift* – put the next state and token on the stack
    - *reduce* – an RHS is found; process the production
    - *terminate* – parsing is complete

Parsing using an LR(0) parser

- Maintain a *parse stack* that tells you what state you're in
  - Start in state 0
- In each state, look up in action table whether to:
  - *shift*: consume a token off the input; look for next state in goto table; push next state onto stack
  - *reduce*: match a production; pop off as many symbols from state stack as seen in production; look up where to go according to non-terminal we just matched; push next state onto stack
  - *accept*: terminate parse

Simple example

1. P → S
2. S → x ; S
3. S → e

		Symbol					
		x	;	e	P	S	Action
State	0	1		3		5	Shift
	1		2				Shift
	2	1		3		4	Shift
	3						Reduce 3
	4						Reduce 2
	5						Accept

Example

- Parse “x ; x ; e”

Step	Parse Stack	Reading Input	Parser Action
1	0	| x ; x ; e	Shift 1
2	0 1	x | ; x ; e	Shift 2
3	0 1 2	x ; | x ; e	Shift 1
4	0 1 2 1	x ; x | ; e	Shift 2
5	0 1 2 1 2	x ; x ; | e	Shift 3
6	0 1 2 1 2 3	x ; x ; e |	Reduce 3 (goto 4)
7	0 1 2 1 2 4	x ; x ; S |	Reduce 2 (goto 4)
8	0 1 2 4	x ; S |	Reduce 2 (goto 4)
9	0 5	S |	Accept