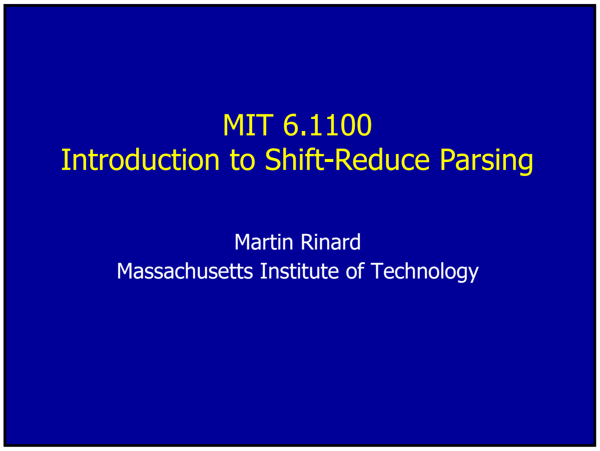
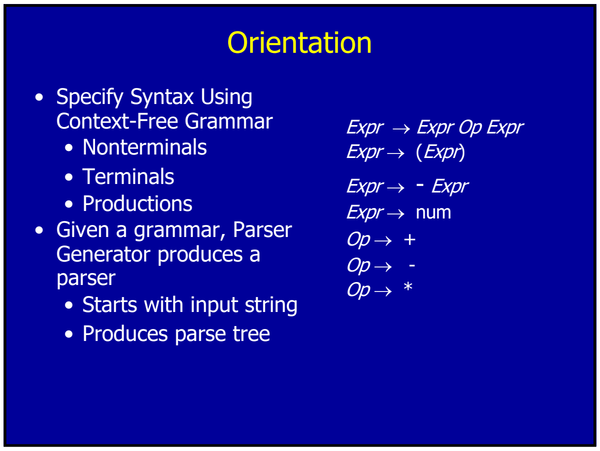


MIT 6.1100
Introduction to Shift-Reduce Parsing

Martin Rinard
Massachusetts Institute of Technology

1

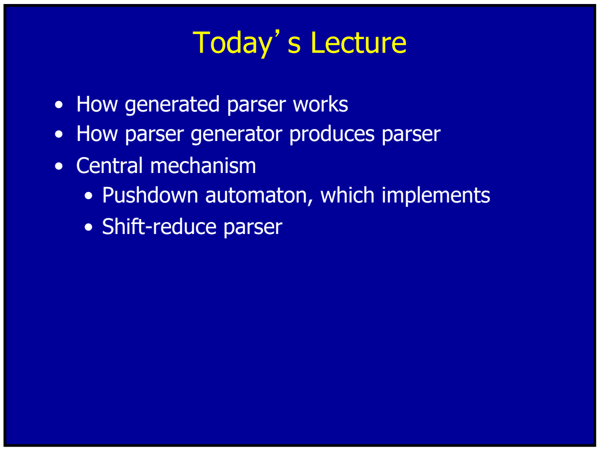


Orientation

- Specify Syntax Using Context-Free Grammar
  - Nonterminals
  - Terminals
  - Productions
- Given a grammar, Parser Generator produces a parser
  - Starts with input string
  - Produces parse tree

Expr → *Expr Op Expr*
Expr → (*Expr*)
Expr → - *Expr*
Expr → num
Op → +
Op → -
Op → *

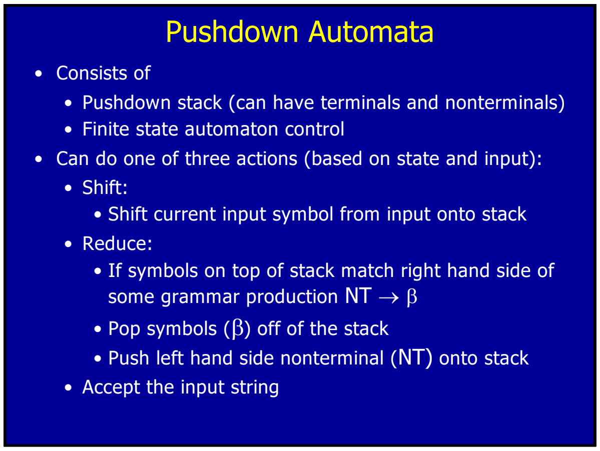
2



Today's Lecture

- How generated parser works
- How parser generator produces parser
- Central mechanism
  - Pushdown automaton, which implements
  - Shift-reduce parser

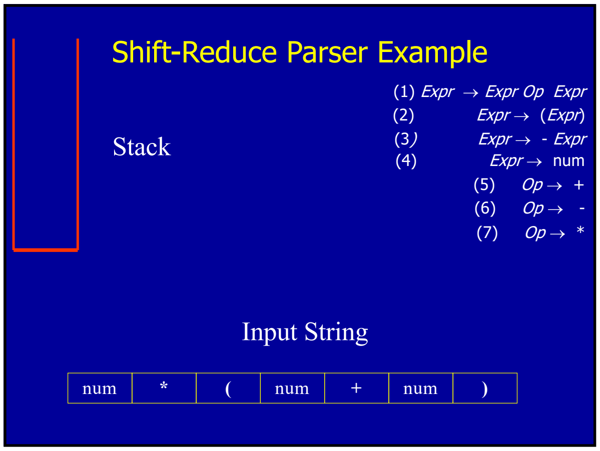
3



Pushdown Automata

- Consists of
  - Pushdown stack (can have terminals and nonterminals)
  - Finite state automaton control
- Can do one of three actions (based on state and input):
  - Shift:
    - Shift current input symbol from input onto stack
  - Reduce:
    - If symbols on top of stack match right hand side of some grammar production NT → β
    - Pop symbols (β) off of the stack
    - Push left hand side nonterminal (NT) onto stack
  - Accept the input string

4



Shift-Reduce Parser Example

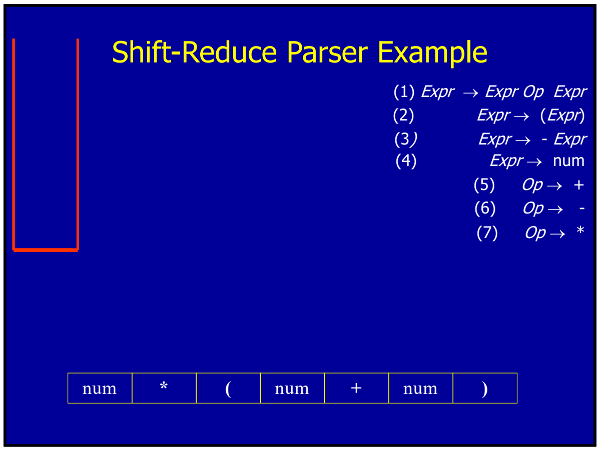
Stack

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

Input String

num	*	(	num	+	num	)

5

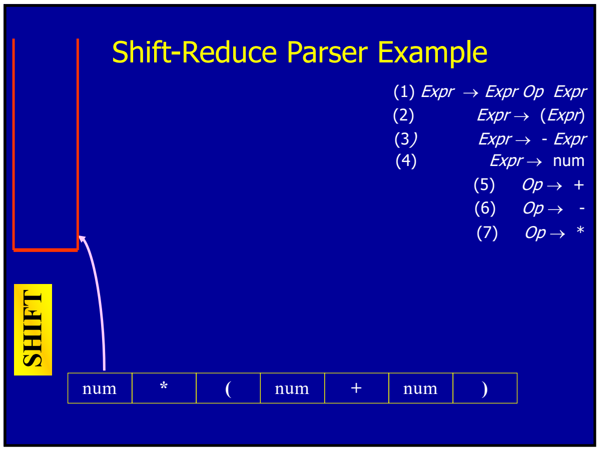


Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

num	*	(	num	+	num	)

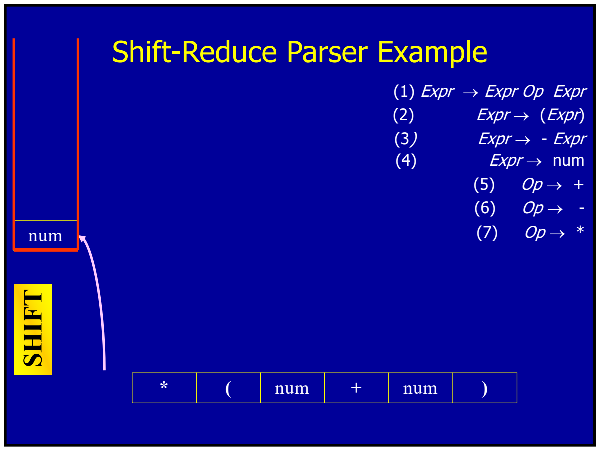
6



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

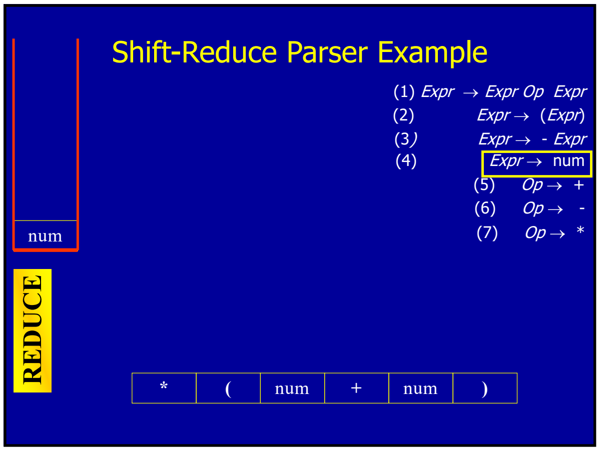
7



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

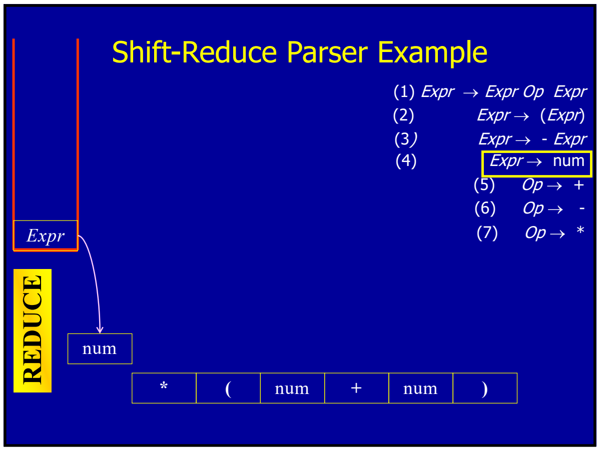
8



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

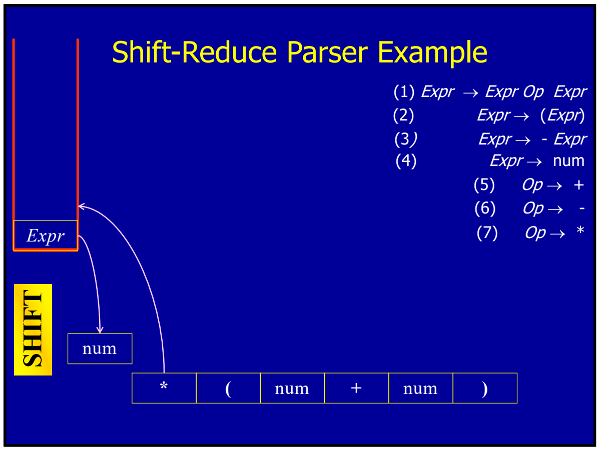
9



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

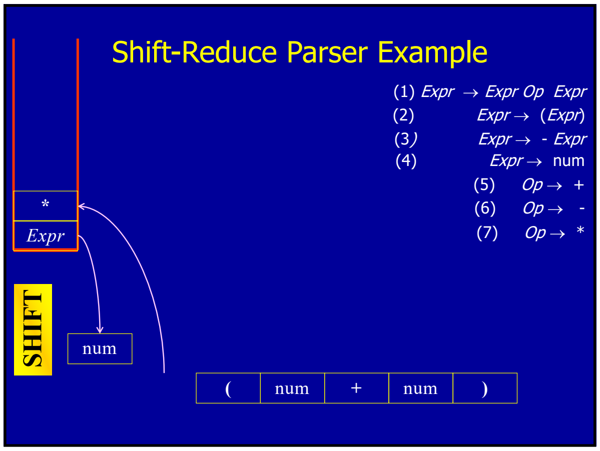
10



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

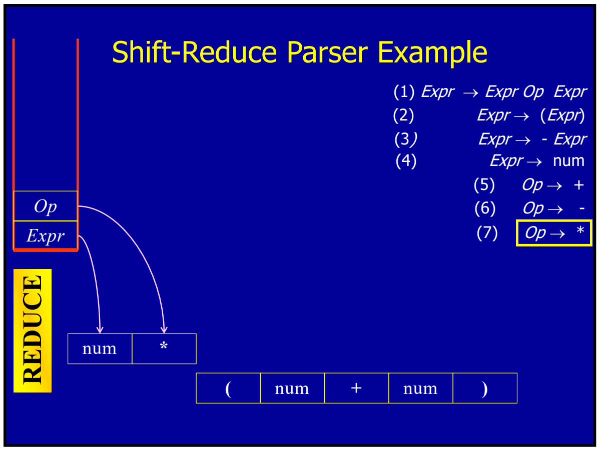
11



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

12



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

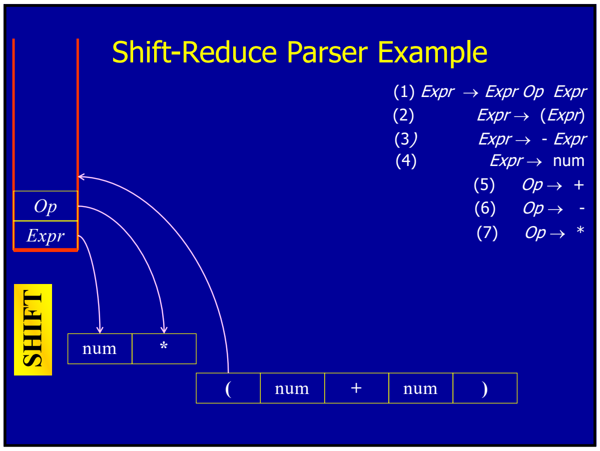
Op
Expr

REDUCE

num *

(num + num)

13



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

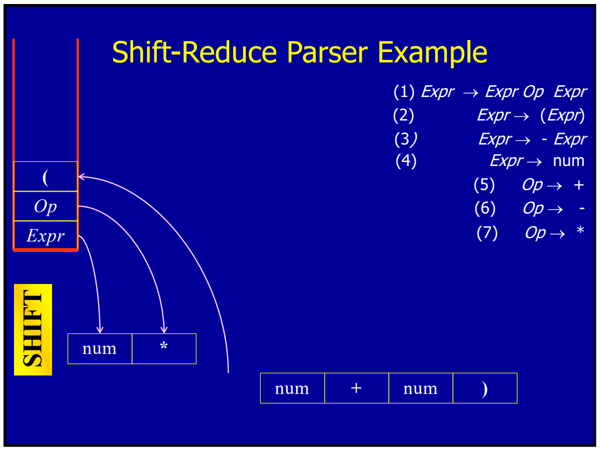
Op
Expr

SHIFT

num *

(num + num)

14



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

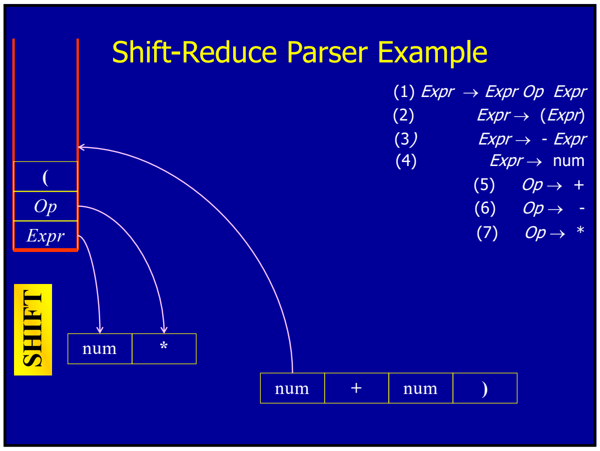
(
Op
Expr

SHIFT

num *

num + num)

15



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

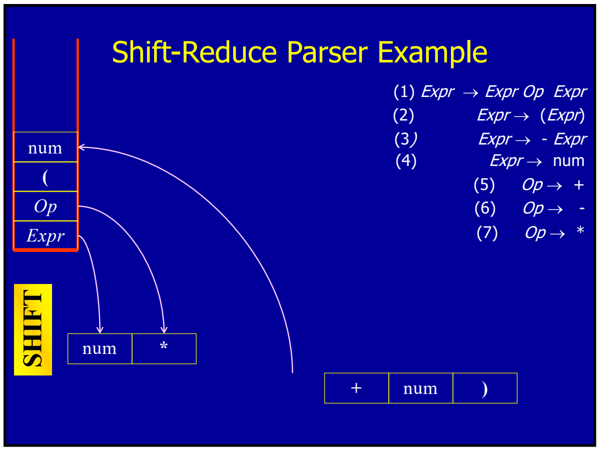
(
Op
Expr

SHIFT

num *

num + num)

16



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

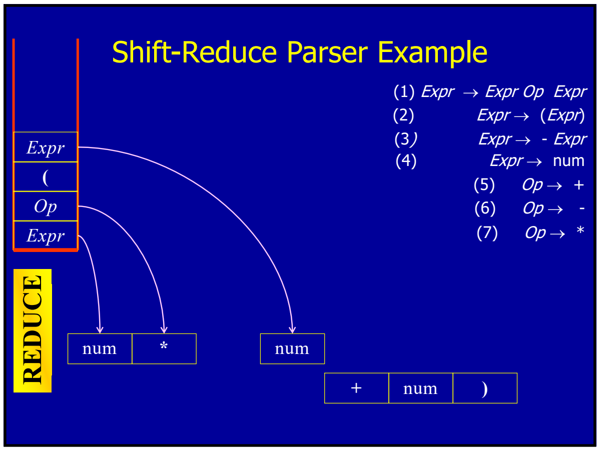
num
(
Op
Expr

SHIFT

num *

+ num)

17



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

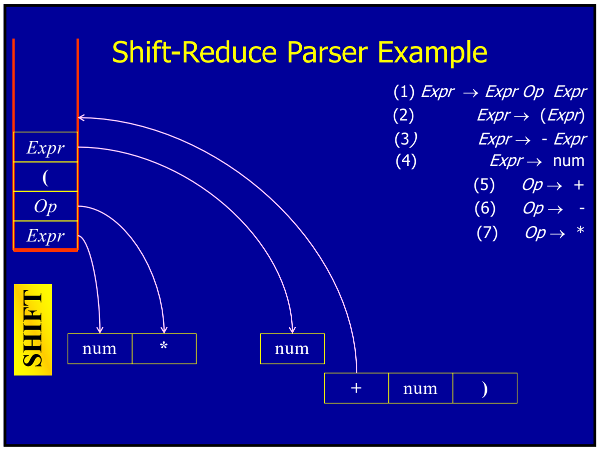
Expr
(
Op
Expr

REDUCE

num * num

+ num)

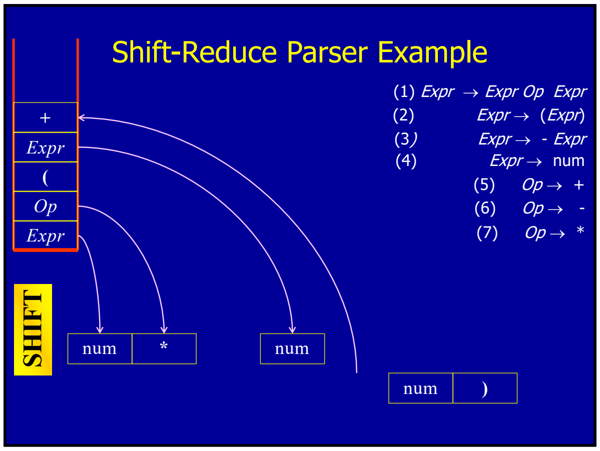
18



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

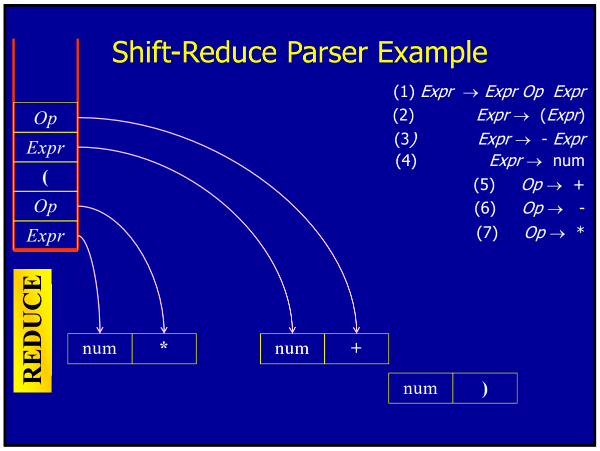
19



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

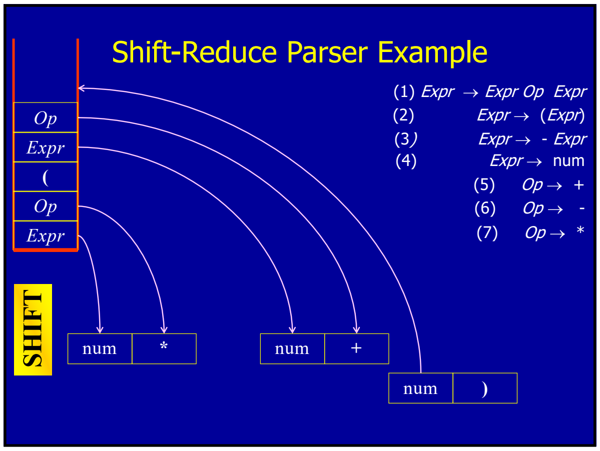
20



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

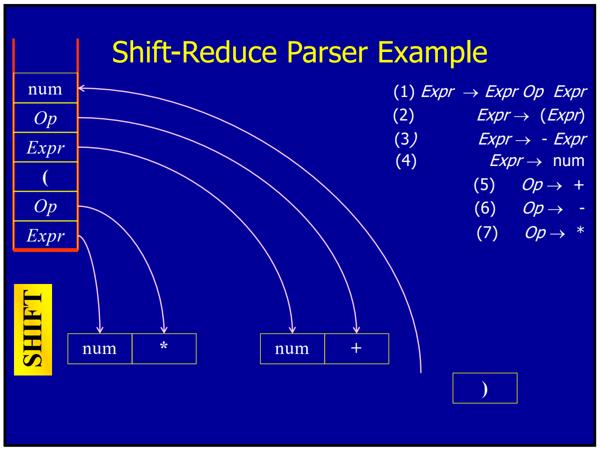
21



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

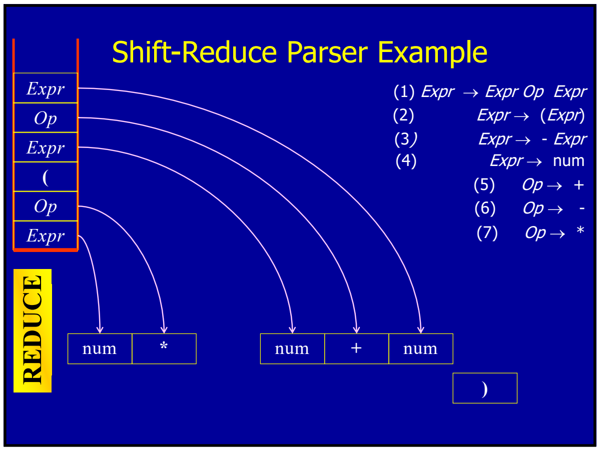
22



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

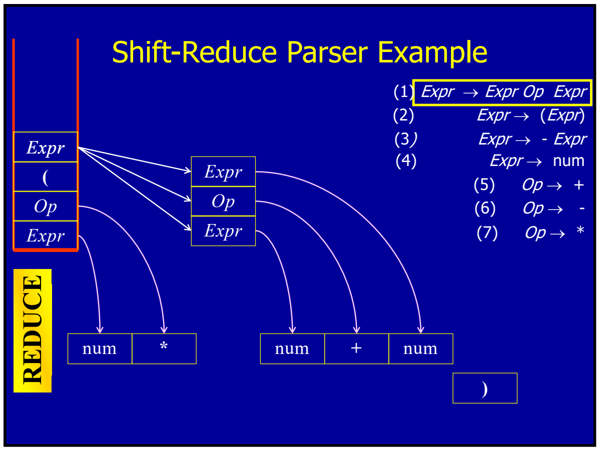
23



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

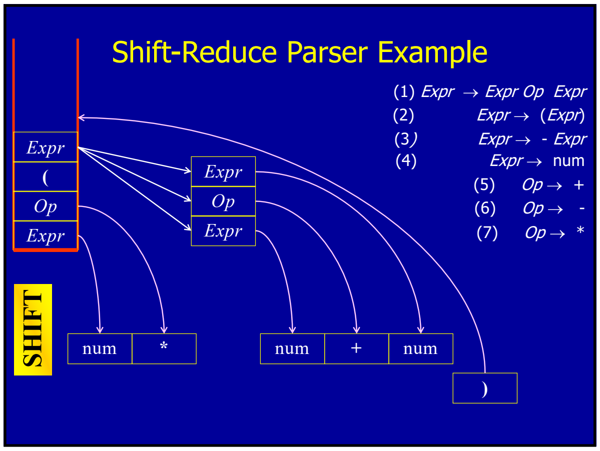
24



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

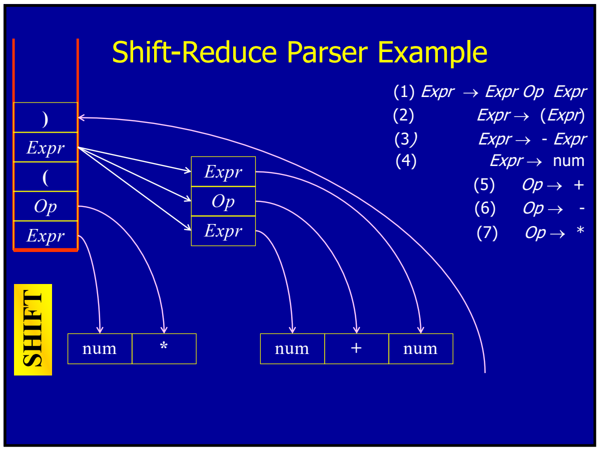
25



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

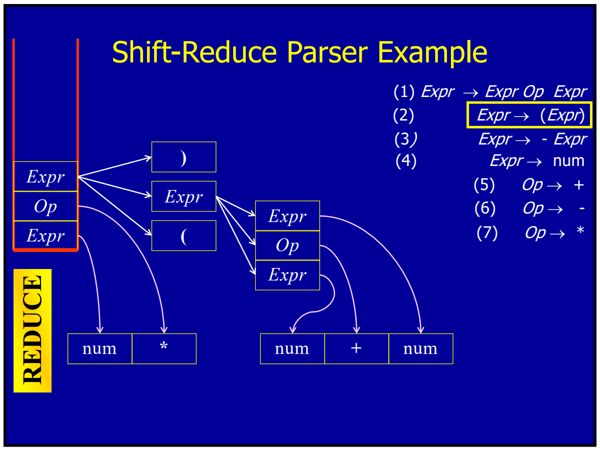
26



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

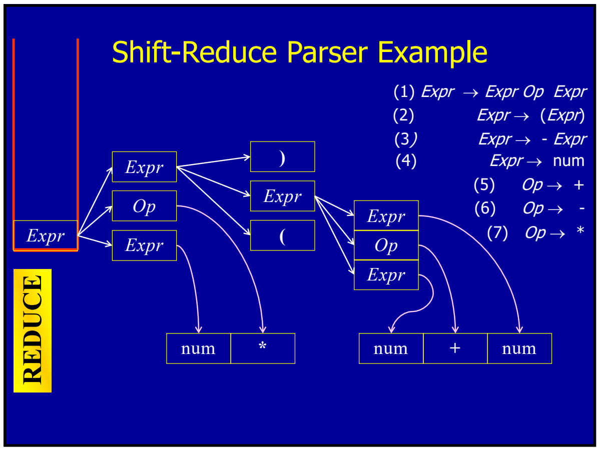
27



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

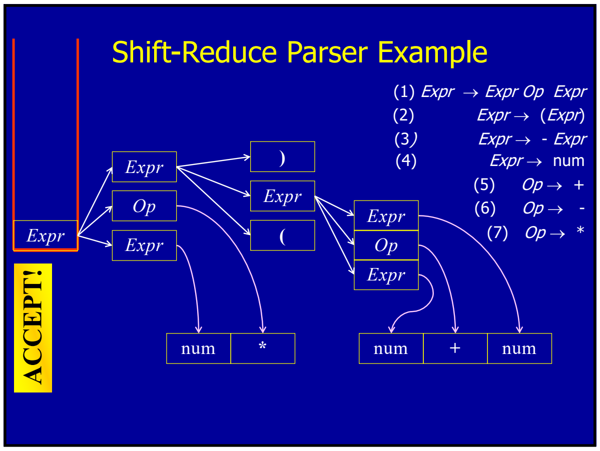
28



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

29



Shift-Reduce Parser Example

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → (*Expr*)
(3) *Expr* → - *Expr*
(4) *Expr* → num
(5) *Op* → +
(6) *Op* → -
(7) *Op* → *

30

Basic Idea

- Goal: reconstruct parse tree for input string
- Read input from left to right
- Build tree in a bottom-up fashion
- Use stack to hold pending sequences of terminals and nonterminals

31

Potential Conflicts

- Reduce/Reduce Conflict
  - Top of the stack may match RHS of multiple productions
  - Which production to use in the reduction?
- Shift/Reduce Conflict
  - Stack may match RHS of production
  - But that may not be the right match
  - May need to shift an input and later find a different reduction

32

Conflicts

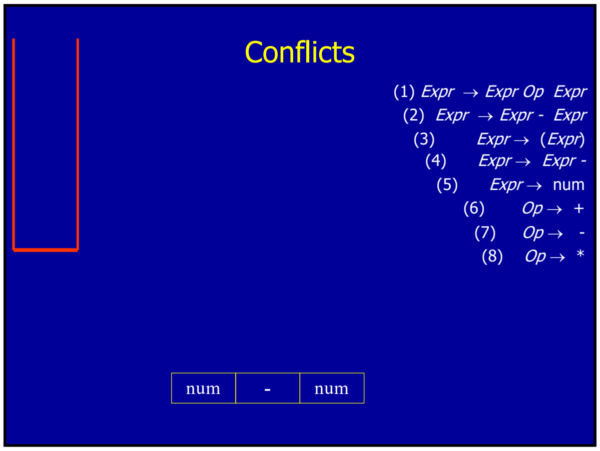
•Original Grammar

Expr → *Expr Op Expr*
Expr → (*Expr*)
Expr → - *Expr*
Expr → num
Op → +
Op → -
Op → *

•New Grammar

Expr → *Expr Op Expr*
Expr → *Expr - Expr*
Expr → (*Expr*)
Expr → *Expr -*
Expr → num
Op → +
Op → -
Op → *

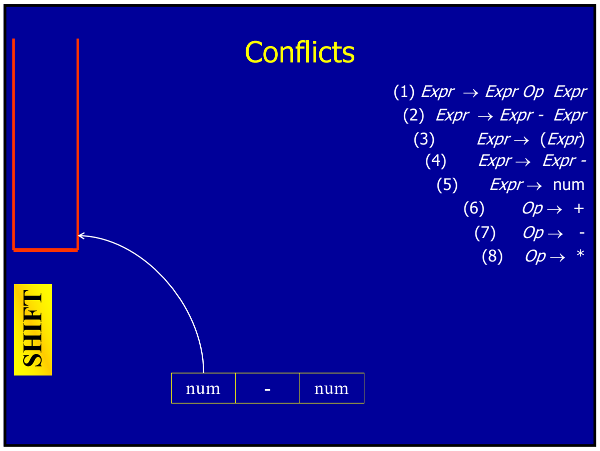
33



Conflicts

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr - Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr -*
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

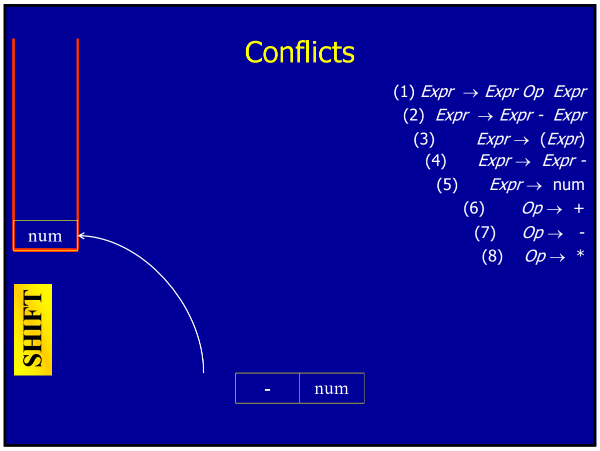
34



Conflicts

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr - Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr -*
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

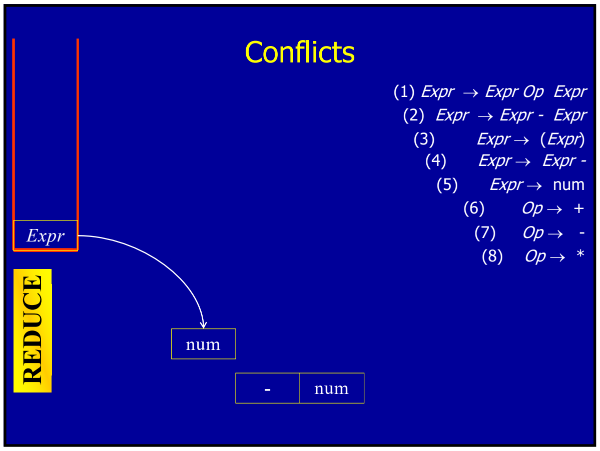
35



Conflicts

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr - Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr -*
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

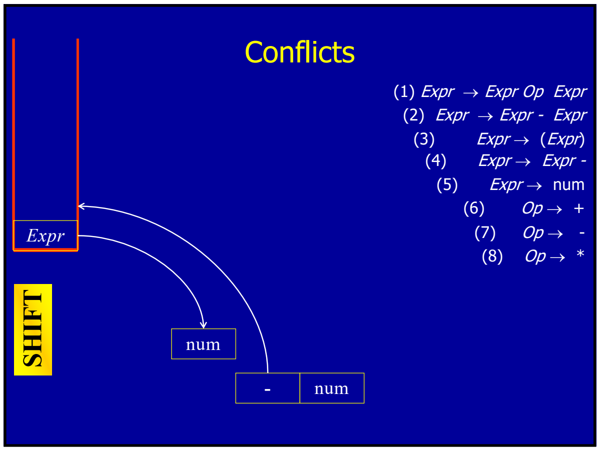
36



Conflicts

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

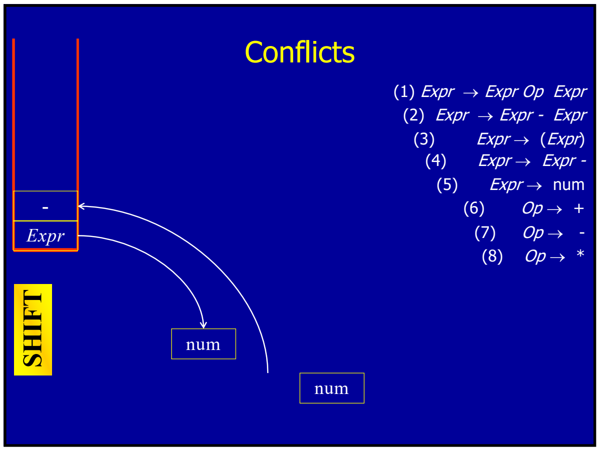
37



Conflicts

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

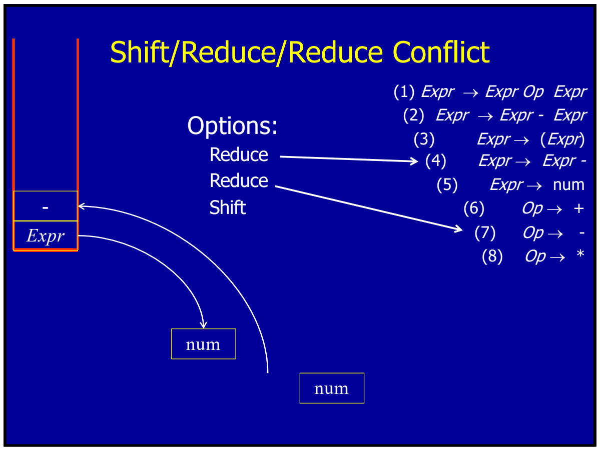
38



Conflicts

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

39

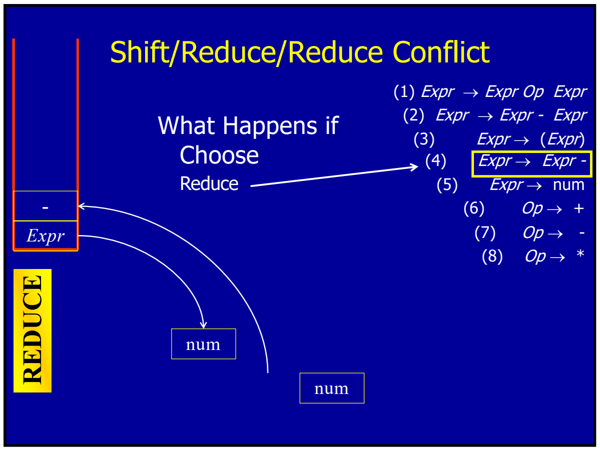


Shift/Reduce/Reduce Conflict

Options:
Reduce
Reduce
Shift

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

40

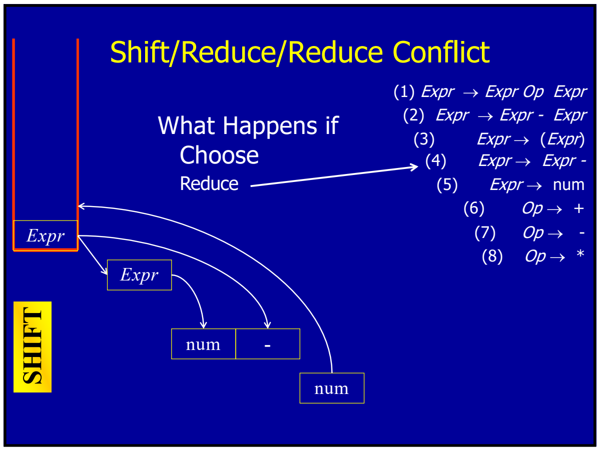


Shift/Reduce/Reduce Conflict

What Happens if Choose
Reduce

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

41

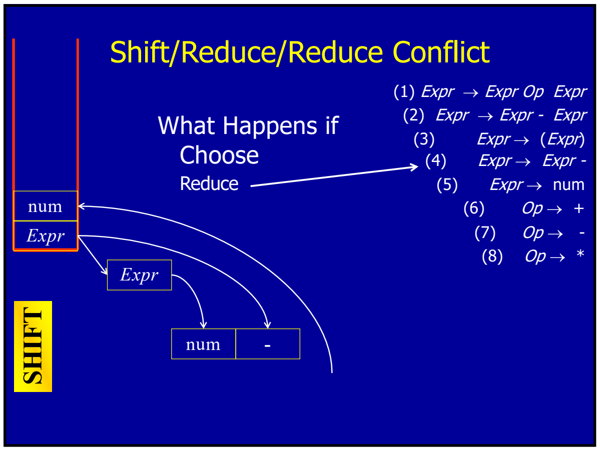


Shift/Reduce/Reduce Conflict

What Happens if Choose
Reduce

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

42



Shift/Reduce/Reduce Conflict

(1) *Expr* → *Expr Op Expr*

(2) *Expr* → *Expr* - *Expr*

(3) *Expr* → (*Expr*)

(4) *Expr* → *Expr* -

(5) *Expr* → num

(6) *Op* → +

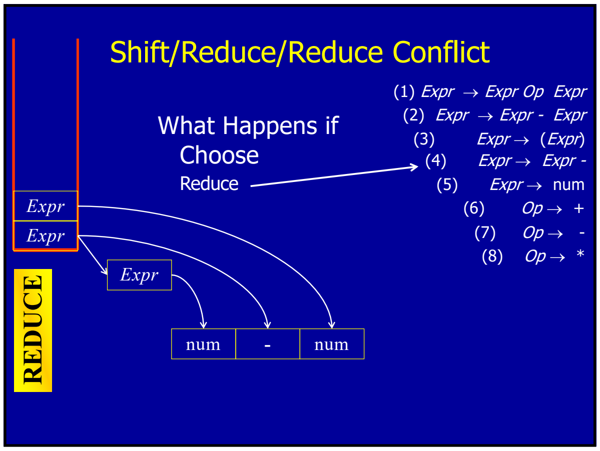
(7) *Op* → -

(8) *Op* → *

What Happens if Choose

Reduce

43



Shift/Reduce/Reduce Conflict

(1) *Expr* → *Expr Op Expr*

(2) *Expr* → *Expr* - *Expr*

(3) *Expr* → (*Expr*)

(4) *Expr* → *Expr* -

(5) *Expr* → num

(6) *Op* → +

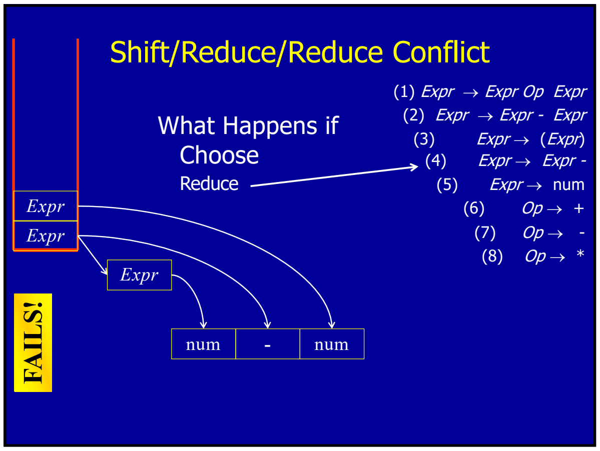
(7) *Op* → -

(8) *Op* → *

What Happens if Choose

Reduce

44



Shift/Reduce/Reduce Conflict

(1) *Expr* → *Expr Op Expr*

(2) *Expr* → *Expr* - *Expr*

(3) *Expr* → (*Expr*)

(4) *Expr* → *Expr* -

(5) *Expr* → num

(6) *Op* → +

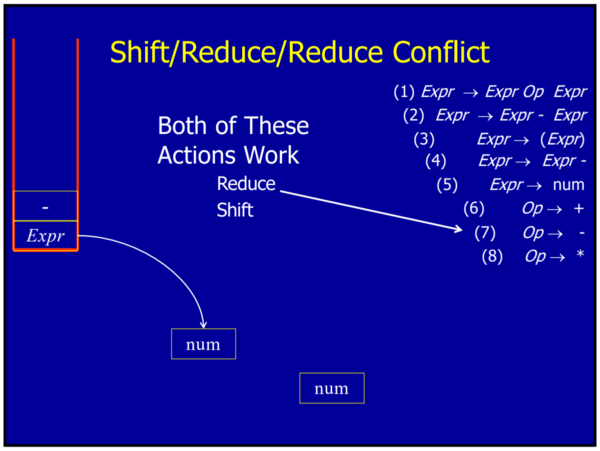
(7) *Op* → -

(8) *Op* → *

What Happens if Choose

Reduce

45



Shift/Reduce/Reduce Conflict

(1) *Expr* → *Expr Op Expr*

(2) *Expr* → *Expr* - *Expr*

(3) *Expr* → (*Expr*)

(4) *Expr* → *Expr* -

(5) *Expr* → num

(6) *Op* → +

(7) *Op* → -

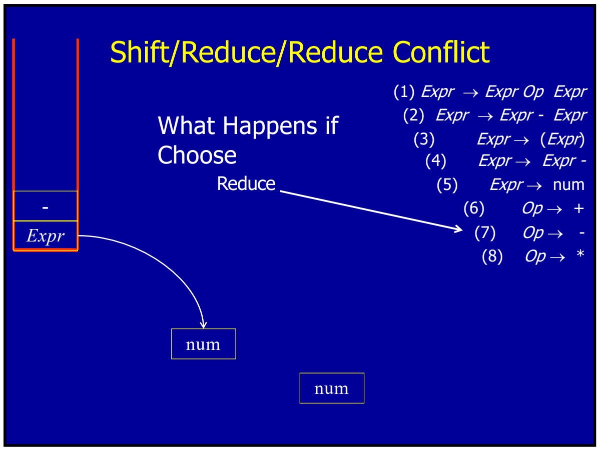
(8) *Op* → *

Both of These Actions Work

Reduce

Shift

46



Shift/Reduce/Reduce Conflict

(1) *Expr* → *Expr Op Expr*

(2) *Expr* → *Expr* - *Expr*

(3) *Expr* → (*Expr*)

(4) *Expr* → *Expr* -

(5) *Expr* → num

(6) *Op* → +

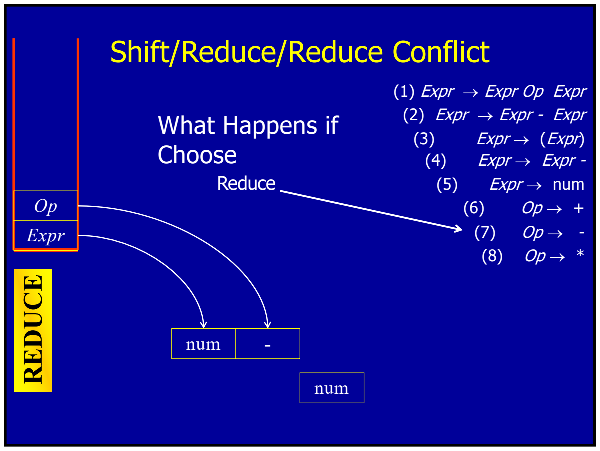
(7) *Op* → -

(8) *Op* → *

What Happens if Choose

Reduce

47



Shift/Reduce/Reduce Conflict

(1) *Expr* → *Expr Op Expr*

(2) *Expr* → *Expr* - *Expr*

(3) *Expr* → (*Expr*)

(4) *Expr* → *Expr* -

(5) *Expr* → num

(6) *Op* → +

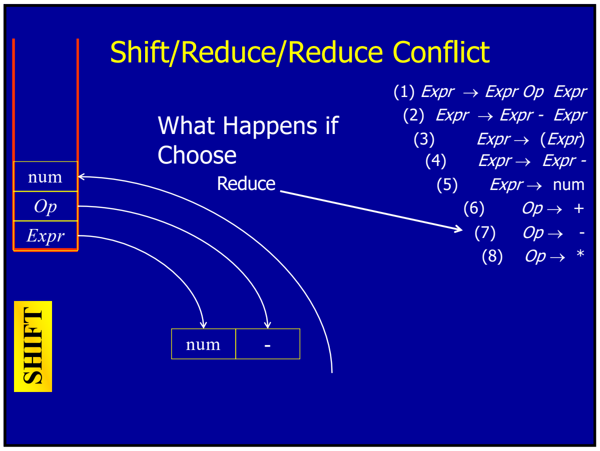
(7) *Op* → -

(8) *Op* → *

What Happens if Choose

Reduce

48



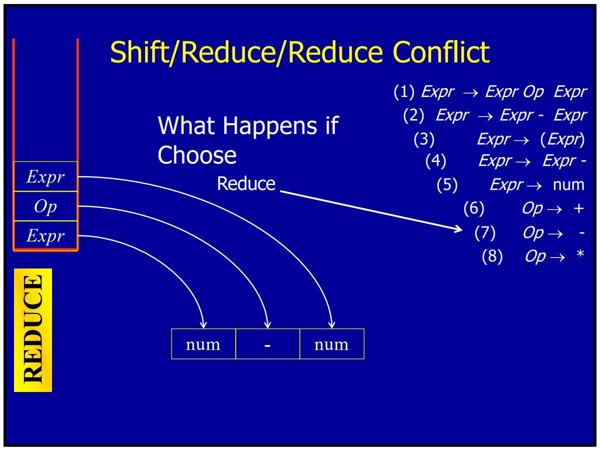
Shift/Reduce/Reduce Conflict

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

What Happens if Choose

Reduce

49



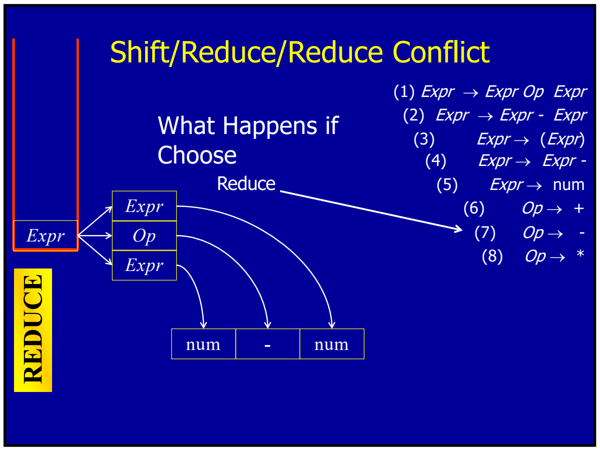
Shift/Reduce/Reduce Conflict

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

What Happens if Choose

Reduce

50



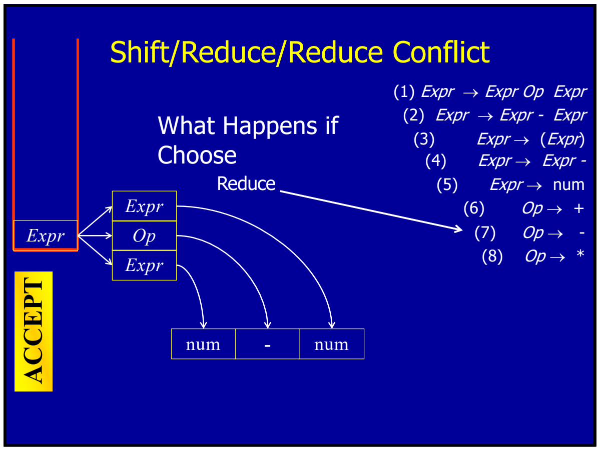
Shift/Reduce/Reduce Conflict

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

What Happens if Choose

Reduce

51



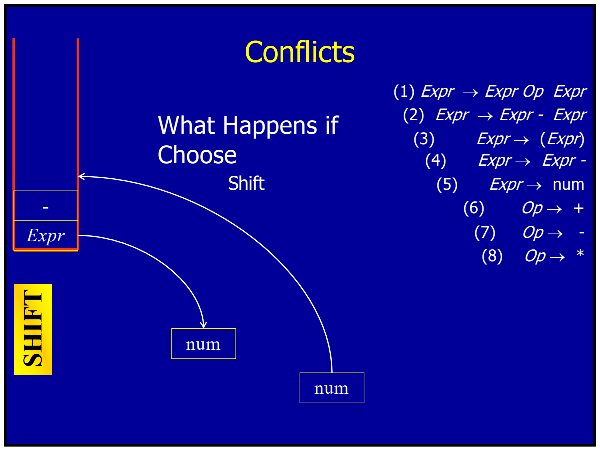
Shift/Reduce/Reduce Conflict

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

What Happens if Choose

Reduce

52



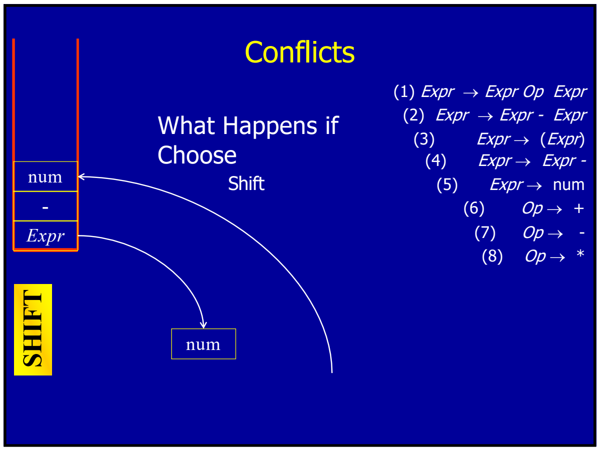
Conflicts

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

What Happens if Choose

Shift

53



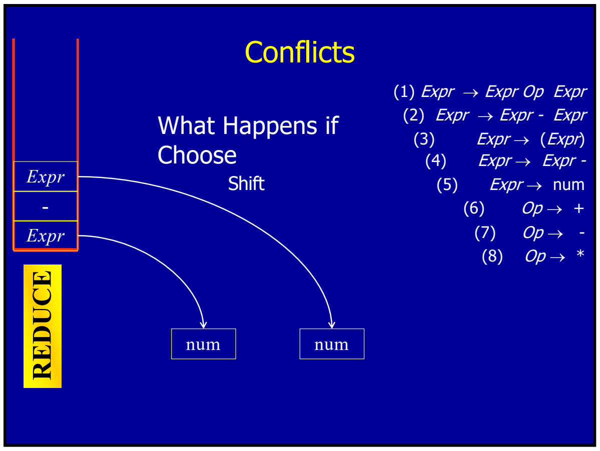
Conflicts

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

What Happens if Choose

Shift

54



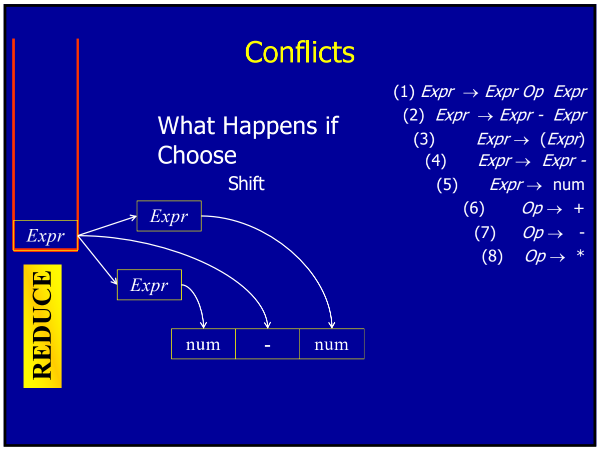
Conflicts

What Happens if Choose

Shift

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

55



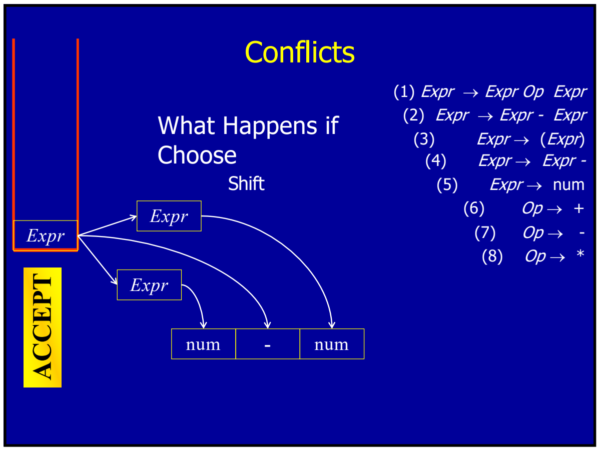
Conflicts

What Happens if Choose

Shift

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

56



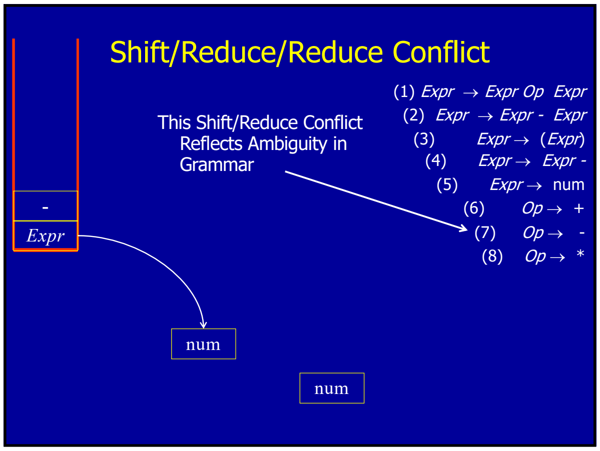
Conflicts

What Happens if Choose

Shift

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

57

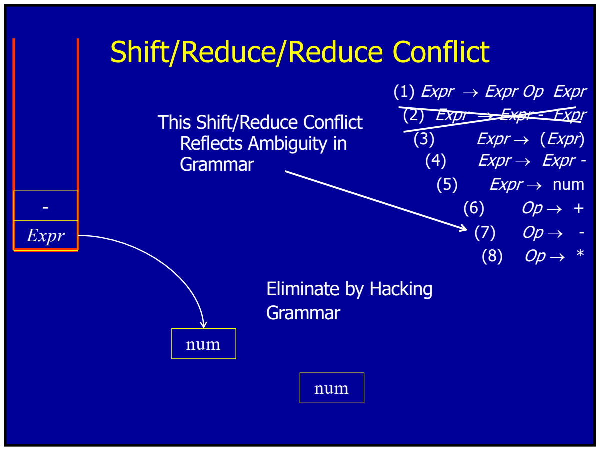


Shift/Reduce/Reduce Conflict

This Shift/Reduce Conflict Reflects Ambiguity in Grammar

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

58



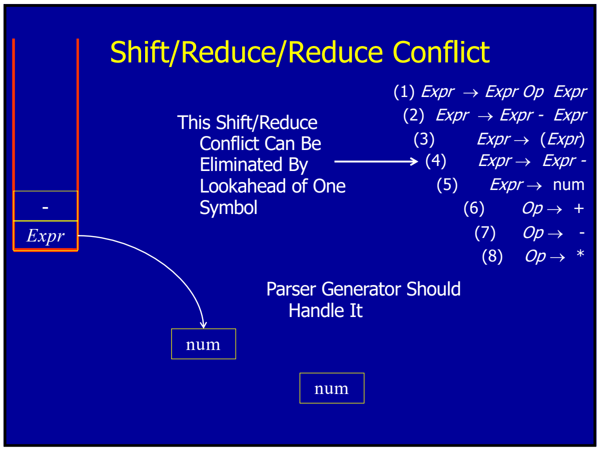
Shift/Reduce/Reduce Conflict

This Shift/Reduce Conflict Reflects Ambiguity in Grammar

(1) *Expr* → *Expr Op Expr*
~~(2) *Expr* → *Expr* - *Expr*~~
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

Eliminate by Hacking Grammar

59



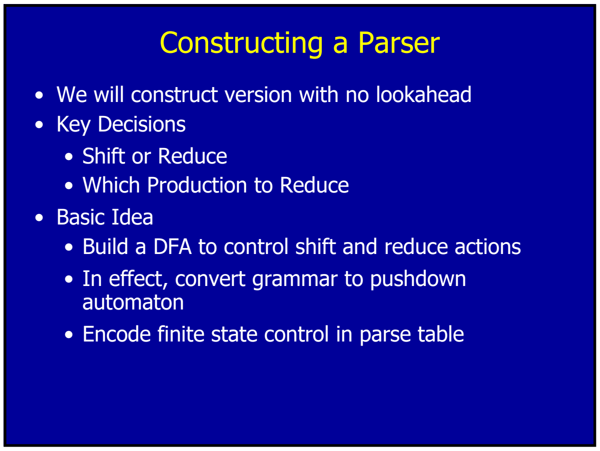
Shift/Reduce/Reduce Conflict

This Shift/Reduce Conflict Can Be Eliminated By Lookahead of One Symbol

(1) *Expr* → *Expr Op Expr*
(2) *Expr* → *Expr* - *Expr*
(3) *Expr* → (*Expr*)
(4) *Expr* → *Expr* -
(5) *Expr* → num
(6) *Op* → +
(7) *Op* → -
(8) *Op* → *

Parser Generator Should Handle It

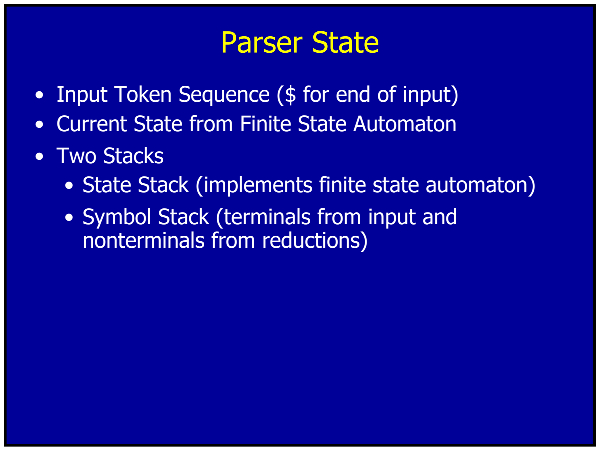
60



Constructing a Parser

- We will construct version with no lookahead
- Key Decisions
  - Shift or Reduce
  - Which Production to Reduce
- Basic Idea
  - Build a DFA to control shift and reduce actions
  - In effect, convert grammar to pushdown automaton
  - Encode finite state control in parse table

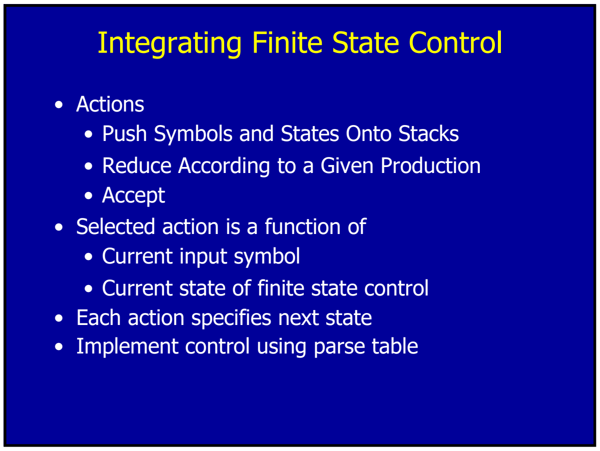
61



Parser State

- Input Token Sequence ($ for end of input)
- Current State from Finite State Automaton
- Two Stacks
  - State Stack (implements finite state automaton)
  - Symbol Stack (terminals from input and nonterminals from reductions)

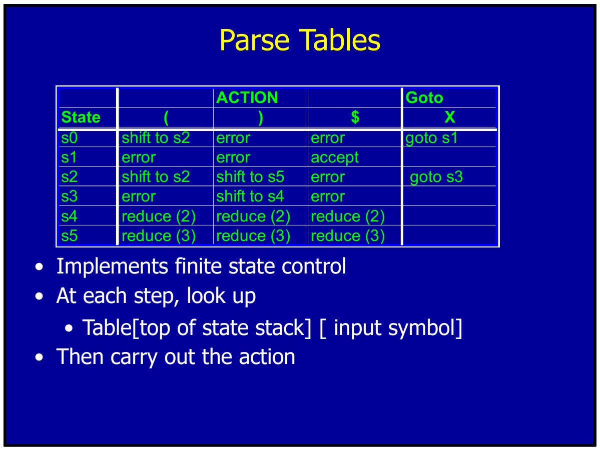
62



Integrating Finite State Control

- Actions
  - Push Symbols and States Onto Stacks
  - Reduce According to a Given Production
  - Accept
- Selected action is a function of
  - Current input symbol
  - Current state of finite state control
- Each action specifies next state
- Implement control using parse table

63

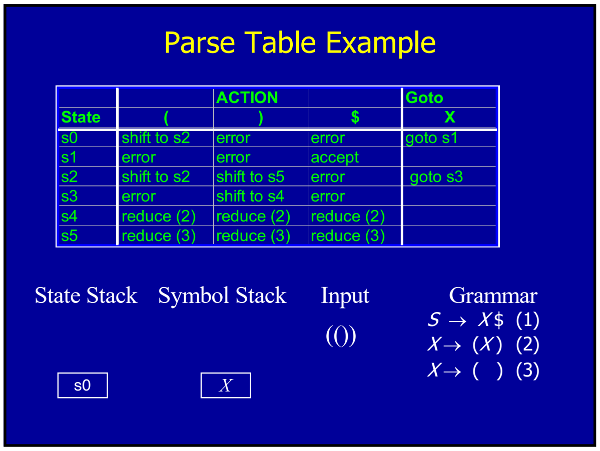


Parse Tables

		ACTION		Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Implements finite state control
- At each step, look up
  - Table[top of state stack] [input symbol]
- Then carry out the action

64



Parse Table Example

		ACTION		Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack

s0

Symbol Stack

X

Input

(())

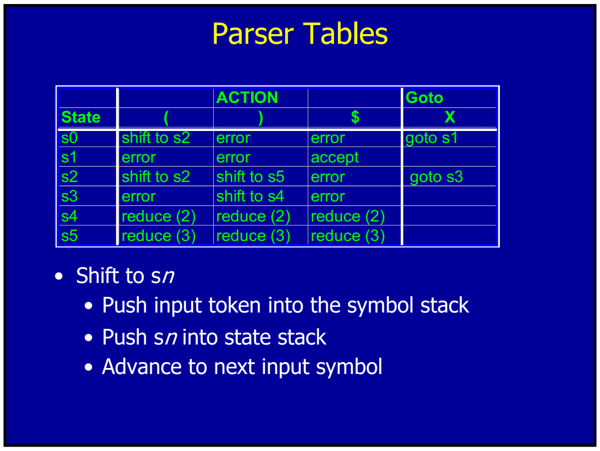
Grammar

$S \to X\$$ (1)

$X \to (X)$ (2)

$X \to (\ )$ (3)

65

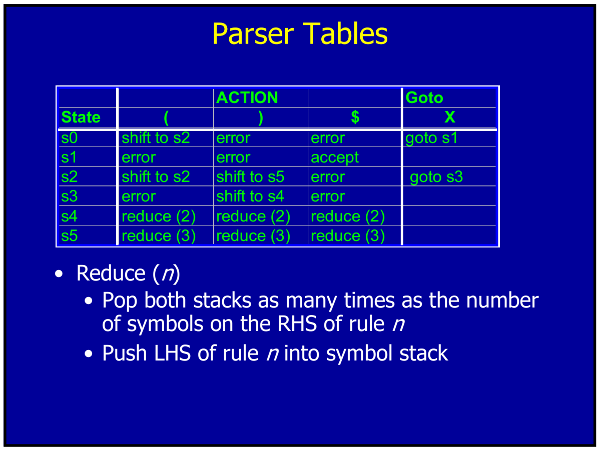


Parser Tables

		ACTION		Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Shift to s*n*
  - Push input token into the symbol stack
  - Push s*n* into state stack
  - Advance to next input symbol

66

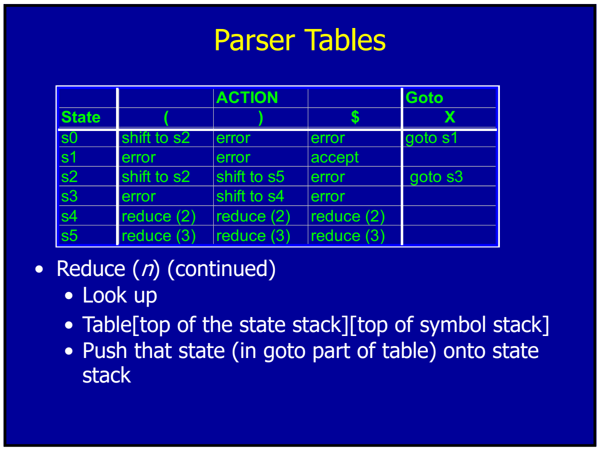


Parser Tables

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Reduce (*n*)
  - Pop both stacks as many times as the number of symbols on the RHS of rule *n*
  - Push LHS of rule *n* into symbol stack

67

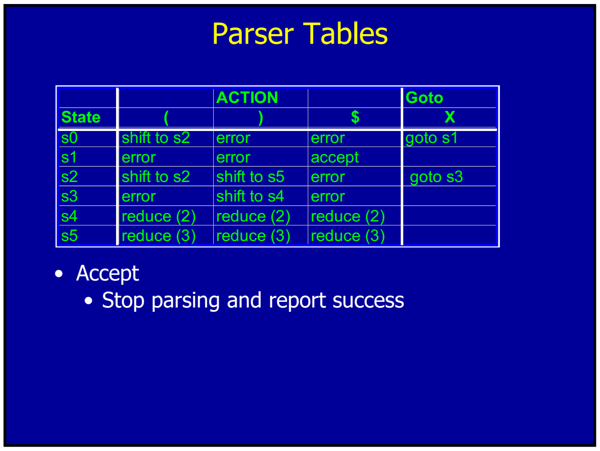


Parser Tables

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Reduce (*n*) (continued)
  - Look up
  - Table[top of the state stack][top of symbol stack]
  - Push that state (in goto part of table) onto state stack

68

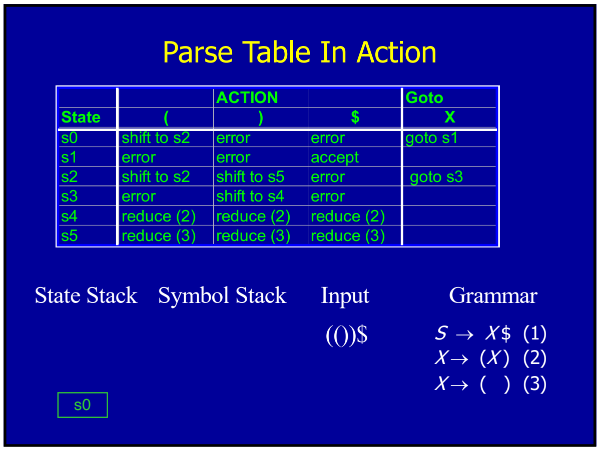


Parser Tables

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Accept
  - Stop parsing and report success

69



Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

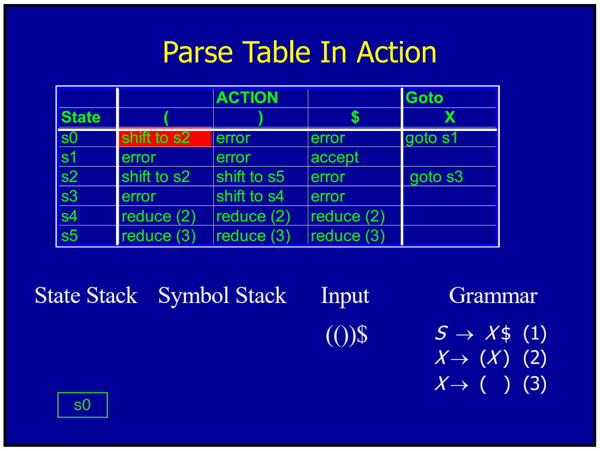
State Stack　Symbol Stack　Input　Grammar

(())$

$S \rightarrow X\$$ (1)
$X \rightarrow (X)$ (2)
$X \rightarrow (\ )$ (3)

s0

70



Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

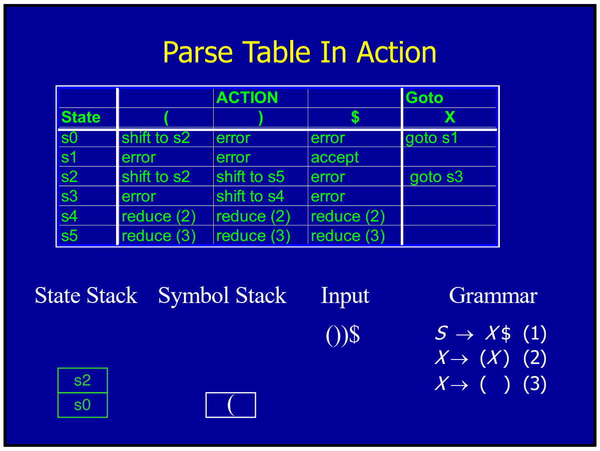
State Stack　Symbol Stack　Input　Grammar

(())$

$S \rightarrow X\$$ (1)
$X \rightarrow (X)$ (2)
$X \rightarrow (\ )$ (3)

s0

71



Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

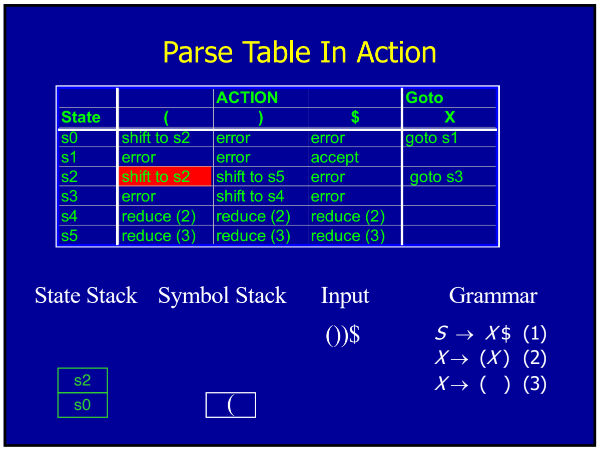
State Stack　Symbol Stack　Input　Grammar

())$

$S \rightarrow X\$$ (1)
$X \rightarrow (X)$ (2)
$X \rightarrow (\ )$ (3)

s2
s0　(

72



Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s2, s0

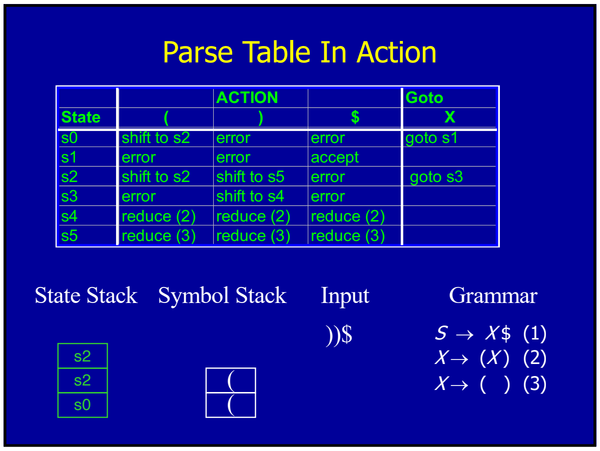
Symbol Stack: (

Input: ())$

Grammar

$S \to X\$$ (1)
$X \to (X)$ (2)
$X \to (\ )$ (3)

73



Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s2, s2, s0

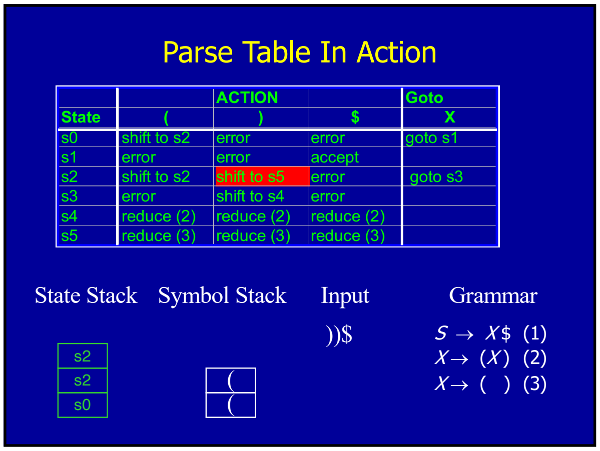
Symbol Stack: (, (

Input:))$

Grammar

$S \to X\$$ (1)
$X \to (X)$ (2)
$X \to (\ )$ (3)

74



Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s2, s2, s0

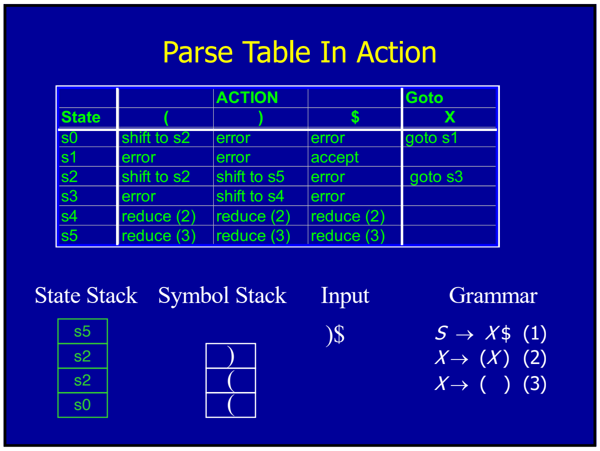
Symbol Stack: (, (

Input:))$

Grammar

$S \to X\$$ (1)
$X \to (X)$ (2)
$X \to (\ )$ (3)

75



Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s5, s2, s2, s0

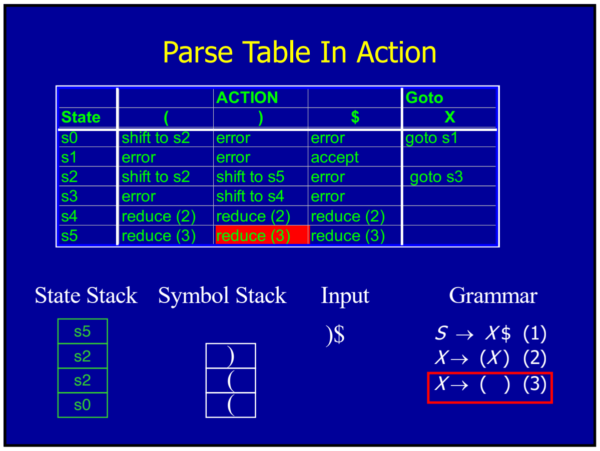
Symbol Stack:), (, (

Input:)$

Grammar

$S \to X\$$ (1)
$X \to (X)$ (2)
$X \to (\ )$ (3)

76



Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s5, s2, s2, s0

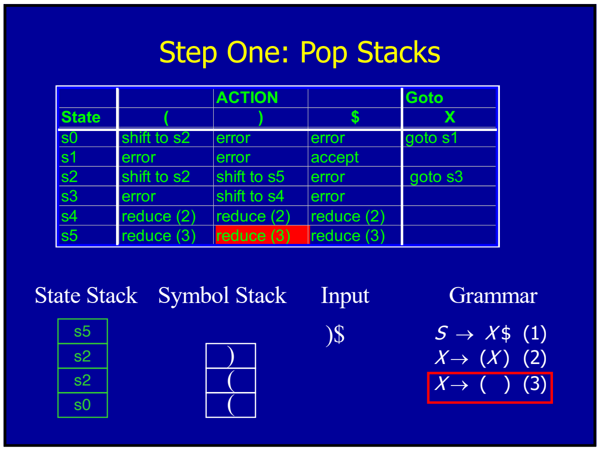
Symbol Stack:), (, (

Input:)$

Grammar

$S \to X\$$ (1)
$X \to (X)$ (2)
$\boxed{X \to (\ )\ (3)}$

77



Step One: Pop Stacks

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s5, s2, s2, s0

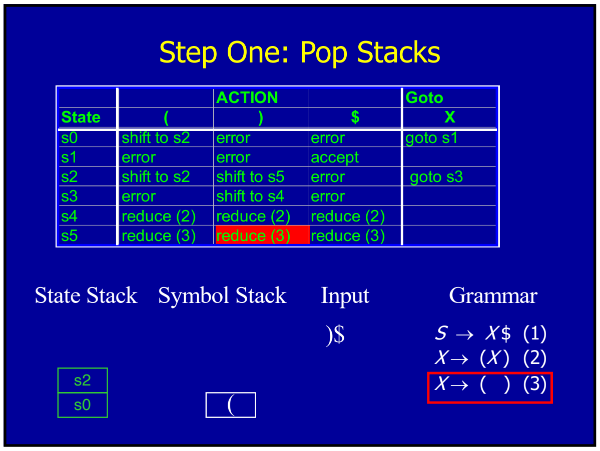
Symbol Stack:), (, (

Input:)$

Grammar

$S \to X\$$ (1)
$X \to (X)$ (2)
$\boxed{X \to (\ )\ (3)}$

78



Step One: Pop Stacks

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s2, s0

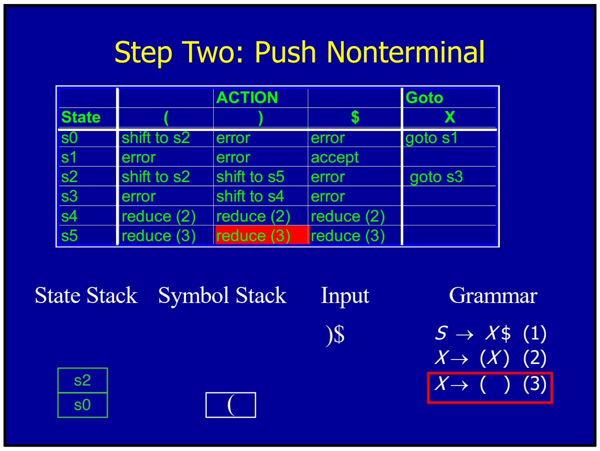
Symbol Stack: (

Input:)$

Grammar:

$S \rightarrow X\$$ (1)
$X \rightarrow (X)$ (2)
$X \rightarrow (\ )$ (3)

79



Step Two: Push Nonterminal

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s2, s0

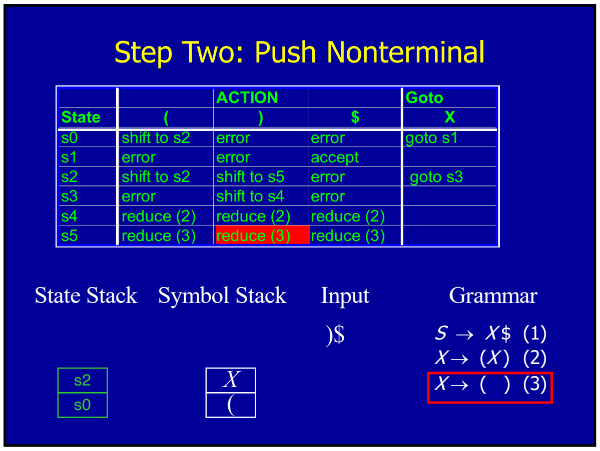
Symbol Stack: (

Input:)$

Grammar:

$S \rightarrow X\$$ (1)
$X \rightarrow (X)$ (2)
$X \rightarrow (\ )$ (3)

80



Step Two: Push Nonterminal

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s2, s0

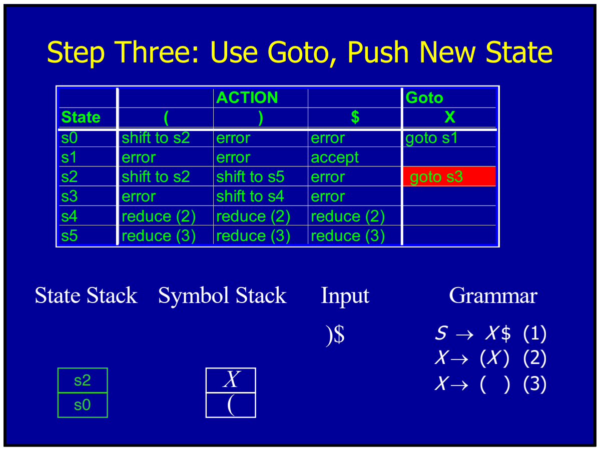
Symbol Stack: X, (

Input:)$

Grammar:

$S \rightarrow X\$$ (1)
$X \rightarrow (X)$ (2)
$X \rightarrow (\ )$ (3)

81



Step Three: Use Goto, Push New State

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s2, s0

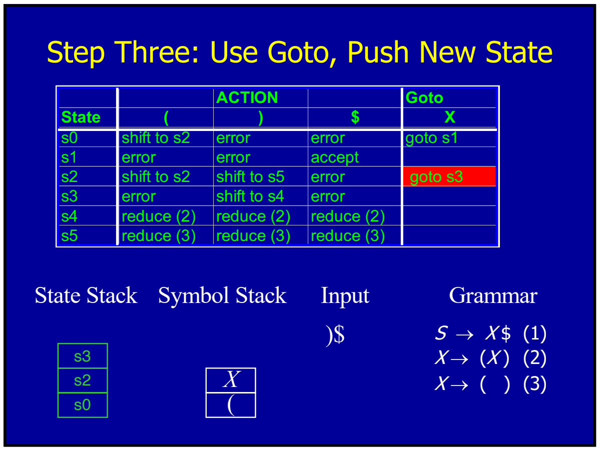
Symbol Stack: X, (

Input:)$

Grammar:

$S \rightarrow X\$$ (1)
$X \rightarrow (X)$ (2)
$X \rightarrow (\ )$ (3)

82



Step Three: Use Goto, Push New State

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s3, s2, s0

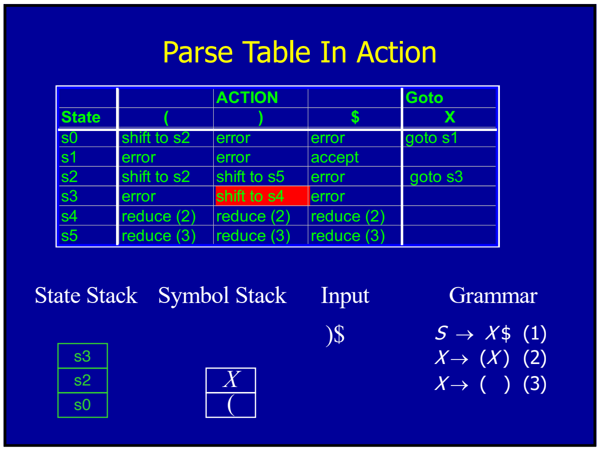
Symbol Stack: X, (

Input:)$

Grammar:

$S \rightarrow X\$$ (1)
$X \rightarrow (X)$ (2)
$X \rightarrow (\ )$ (3)

83



Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack: s3, s2, s0

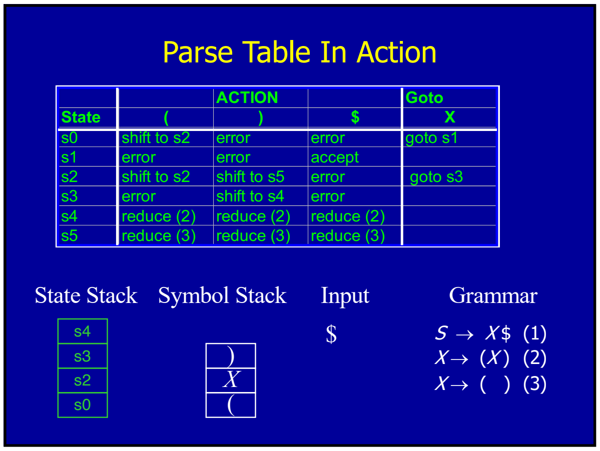
Symbol Stack: X, (

Input:)$

Grammar:

$S \rightarrow X\$$ (1)
$X \rightarrow (X)$ (2)
$X \rightarrow (\ )$ (3)

84

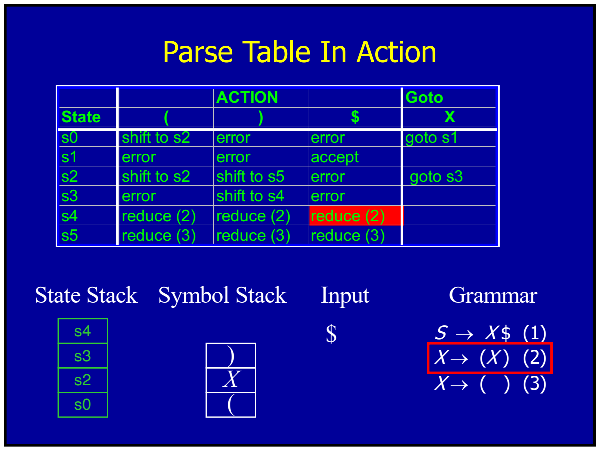


Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
s4		$	$S \to X\$$ (1)
s3	)		$X \to (X)$ (2)
s2	X		$X \to (\ )$ (3)
s0	(		

85

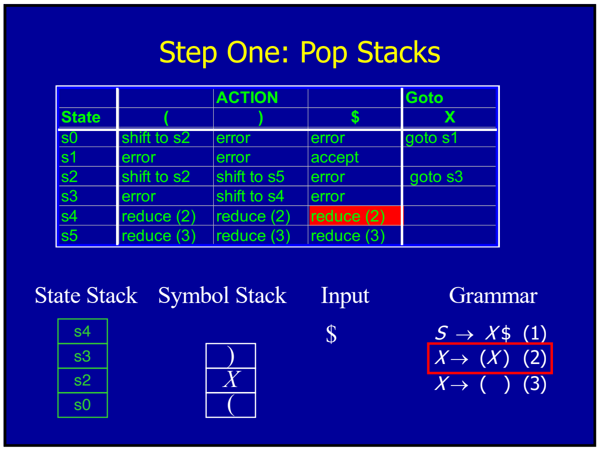


Parse Table In Action

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
s4		$	$S \to X\$$ (1)
s3	)		$\boxed{X \to (X)\ (2)}$
s2	X		$X \to (\ )$ (3)
s0	(		

86

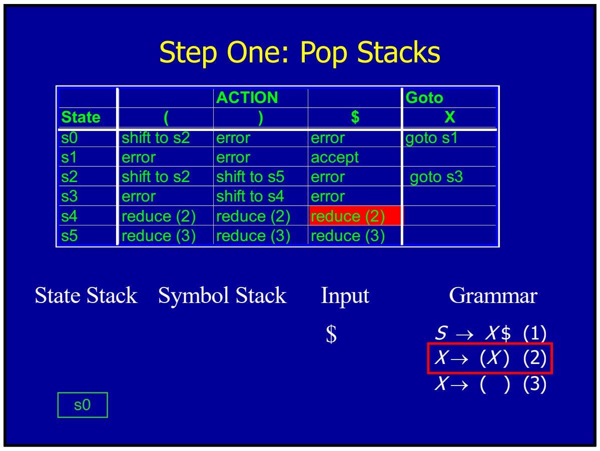


Step One: Pop Stacks

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
s4		$	$S \to X\$$ (1)
s3	)		$\boxed{X \to (X)\ (2)}$
s2	X		$X \to (\ )$ (3)
s0	(		

87

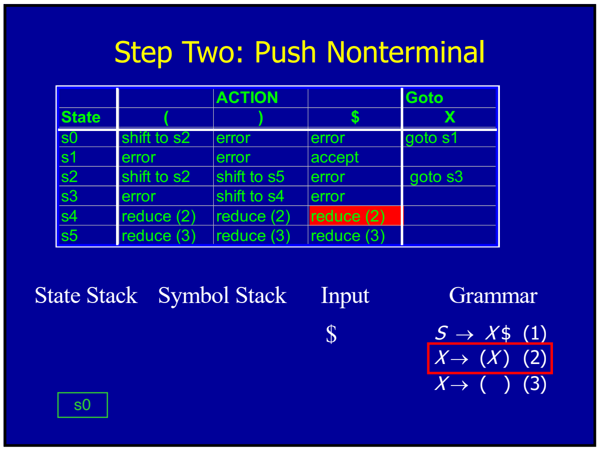


Step One: Pop Stacks

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
		$	$S \to X\$$ (1)
			$\boxed{X \to (X)\ (2)}$
			$X \to (\ )$ (3)
s0			

88

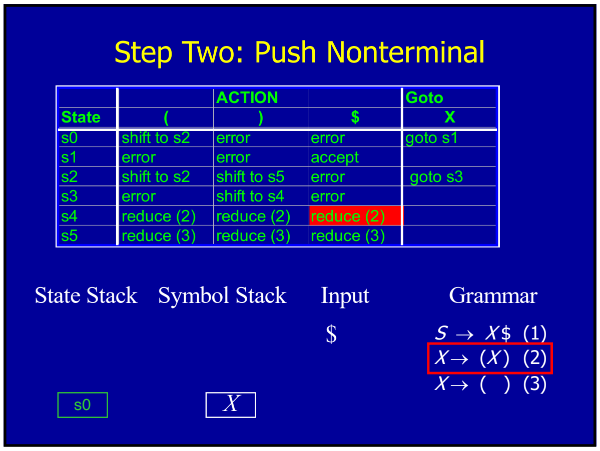


Step Two: Push Nonterminal

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
		$	$S \to X\$$ (1)
			$\boxed{X \to (X)\ (2)}$
			$X \to (\ )$ (3)
s0			

89

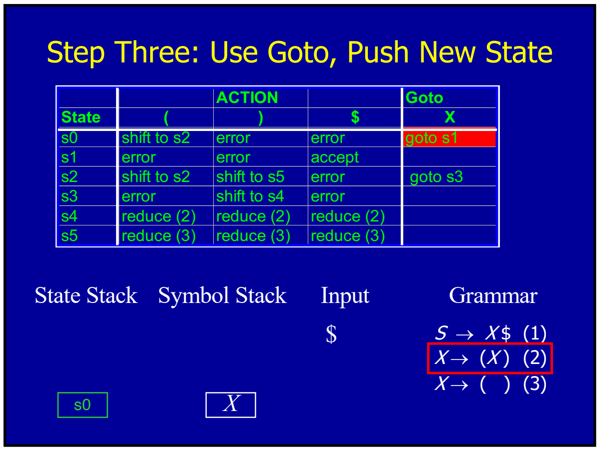


Step Two: Push Nonterminal

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack	Symbol Stack	Input	Grammar
		$	$S \to X\$$ (1)
			$\boxed{X \to (X)\ (2)}$
			$X \to (\ )$ (3)
s0	X		

90



Step Three: Use Goto, Push New State

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack Symbol Stack Input Grammar

$

$S \rightarrow X\$$ (1)

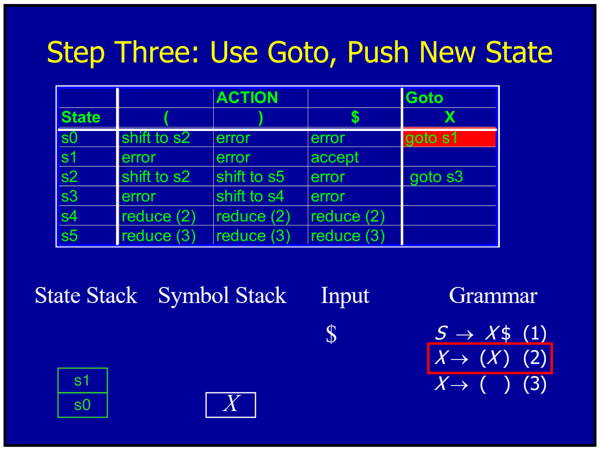
$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

s0

X

91



Step Three: Use Goto, Push New State

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack Symbol Stack Input Grammar

$

$S \rightarrow X\$$ (1)

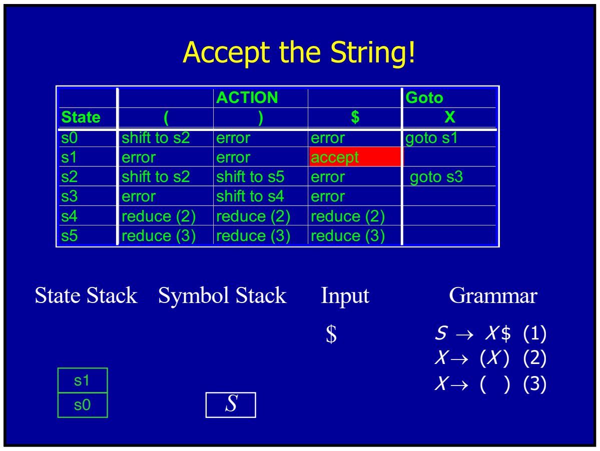
$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

s1
s0

X

92



Accept the String!

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

State Stack Symbol Stack Input Grammar

$

$S \rightarrow X\$$ (1)

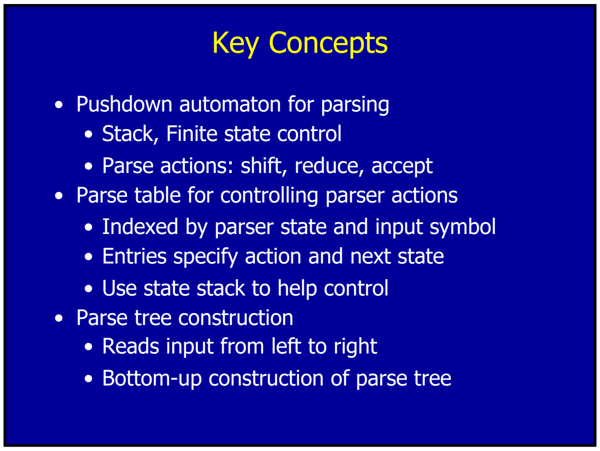
$X \rightarrow (X)$ (2)

$X \rightarrow (\ )$ (3)

s1
s0

S

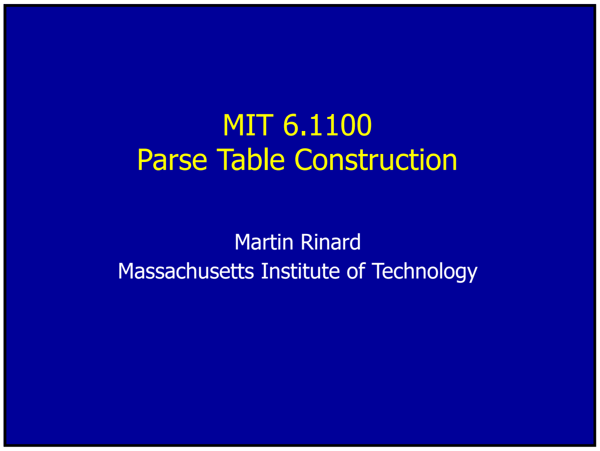
93



Key Concepts

- Pushdown automaton for parsing
  - Stack, Finite state control
  - Parse actions: shift, reduce, accept
- Parse table for controlling parser actions
  - Indexed by parser state and input symbol
  - Entries specify action and next state
  - Use state stack to help control
- Parse tree construction
  - Reads input from left to right
  - Bottom-up construction of parse tree

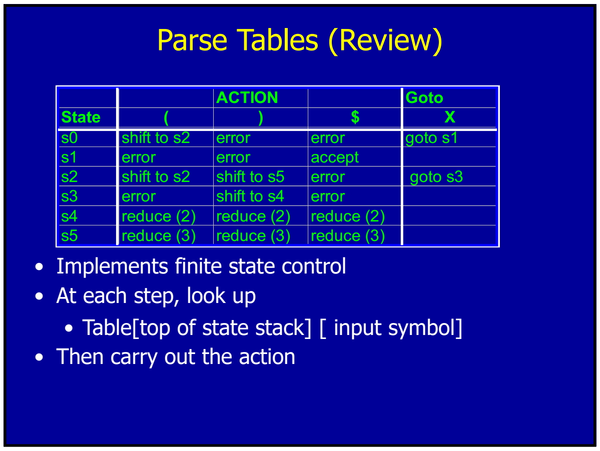
94



MIT 6.1100
Parse Table Construction

Martin Rinard
Massachusetts Institute of Technology

95

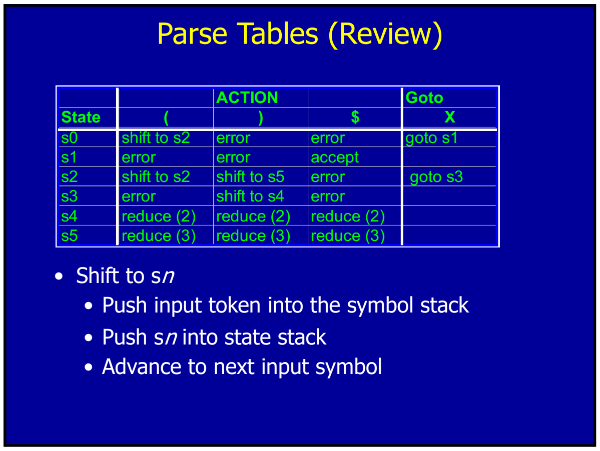


Parse Tables (Review)

	ACTION			Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Implements finite state control
- At each step, look up
  - Table[top of state stack] [input symbol]
- Then carry out the action

96

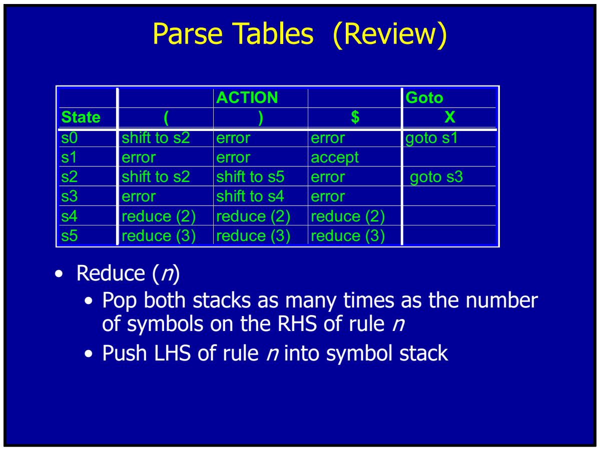


Parse Tables (Review)

		ACTION		Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Shift to s*n*
  - Push input token into the symbol stack
  - Push s*n* into state stack
  - Advance to next input symbol

97

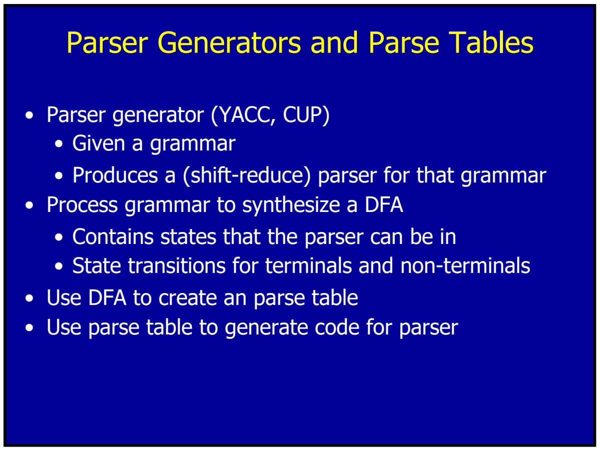


Parse Tables (Review)

		ACTION		Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

- Reduce (*n*)
  - Pop both stacks as many times as the number of symbols on the RHS of rule *n*
  - Push LHS of rule *n* into symbol stack

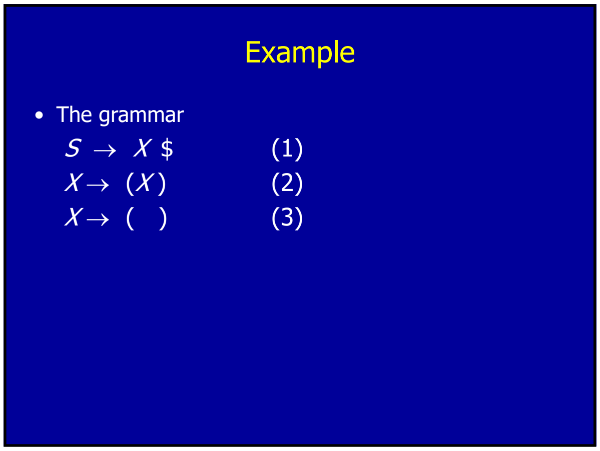
98



Parser Generators and Parse Tables

- Parser generator (YACC, CUP)
  - Given a grammar
  - Produces a (shift-reduce) parser for that grammar
- Process grammar to synthesize a DFA
  - Contains states that the parser can be in
  - State transitions for terminals and non-terminals
- Use DFA to create an parse table
- Use parse table to generate code for parser

99

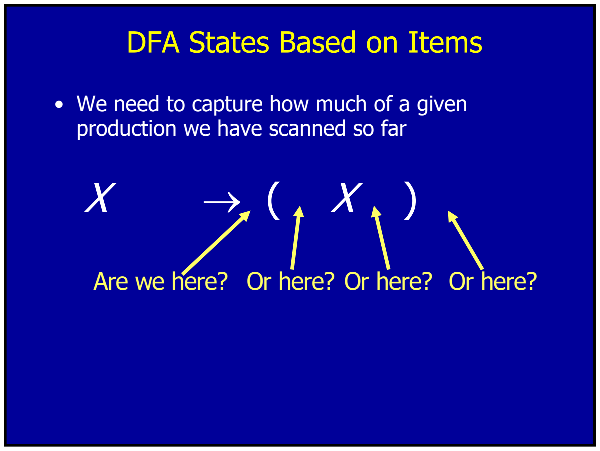


Example

- The grammar

$S \to X\ \$$ (1)
$X \to (X)$ (2)
$X \to (\ )$ (3)

100



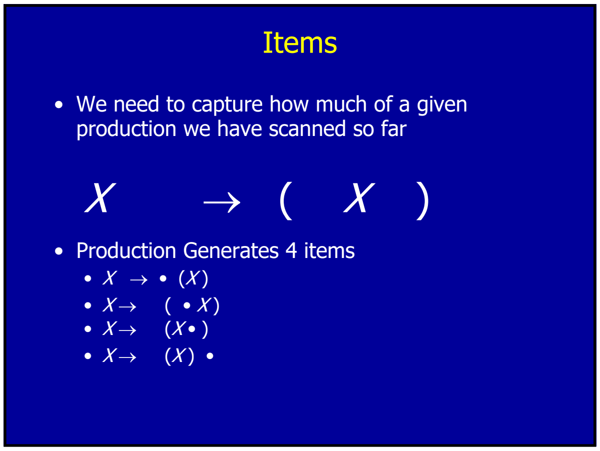
DFA States Based on Items

- We need to capture how much of a given production we have scanned so far

$X \to (X)$

Are we here? Or here? Or here? Or here?

101



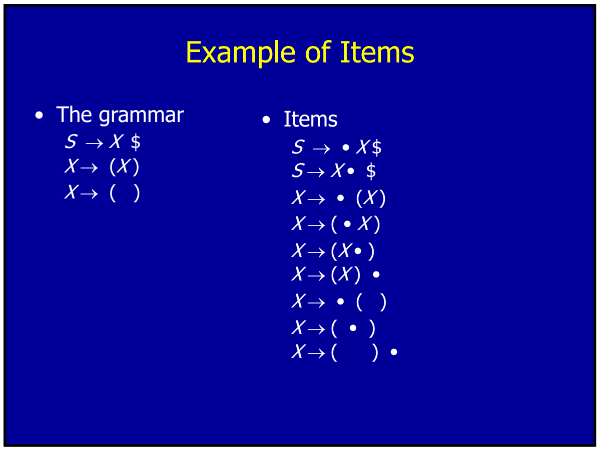
Items

- We need to capture how much of a given production we have scanned so far

$X \to (X)$

- Production Generates 4 items
  - $X \to \bullet (X)$
  - $X \to (\bullet X)$
  - $X \to (X \bullet)$
  - $X \to (X) \bullet$

102



Example of Items

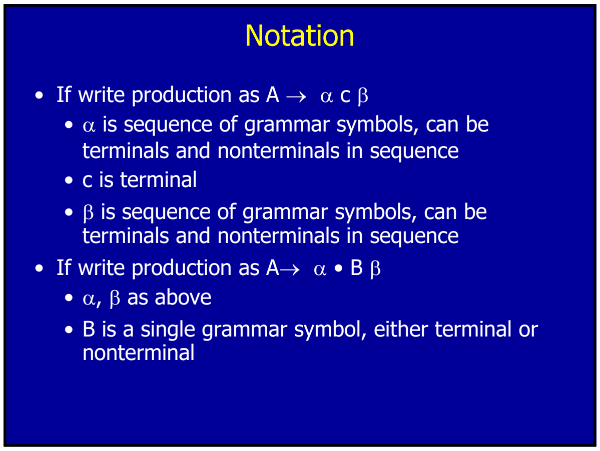
- The grammar

  $S \to X\,\$$
  $X \to (X)$
  $X \to (\;)$

- Items

  $S \to \bullet X\$$
  $S \to X \bullet \$$
  $X \to \bullet\, (X)$
  $X \to (\bullet X)$
  $X \to (X \bullet)$
  $X \to (X) \bullet$
  $X \to \bullet\, (\;)$
  $X \to (\bullet)$
  $X \to (\;) \bullet$

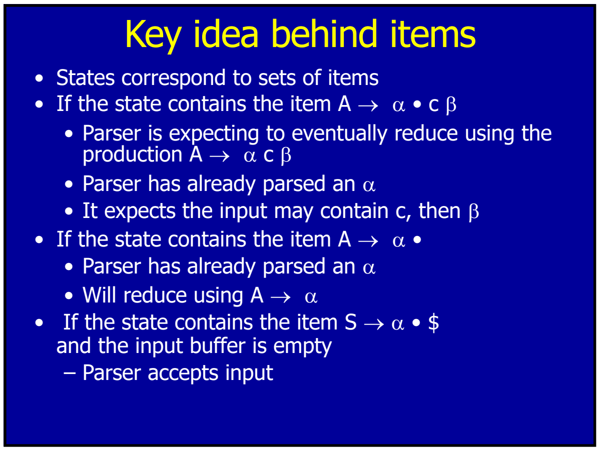
103



Notation

- If write production as A → α c β
  - α is sequence of grammar symbols, can be terminals and nonterminals in sequence
  - c is terminal
  - β is sequence of grammar symbols, can be terminals and nonterminals in sequence
- If write production as A→ α • B β
  - α, β as above
  - B is a single grammar symbol, either terminal or nonterminal

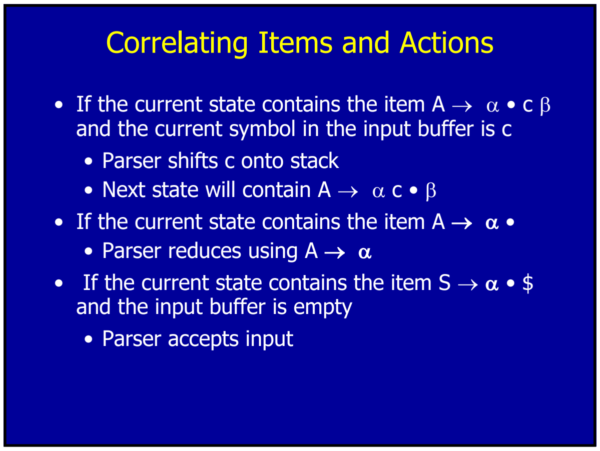
104



Key idea behind items

- States correspond to sets of items
- If the state contains the item A → α • c β
  - Parser is expecting to eventually reduce using the production A → α c β
  - Parser has already parsed an α
  - It expects the input may contain c, then β
- If the state contains the item A → α •
  - Parser has already parsed an α
  - Will reduce using A → α
- If the state contains the item S → α • $ and the input buffer is empty
  - Parser accepts input

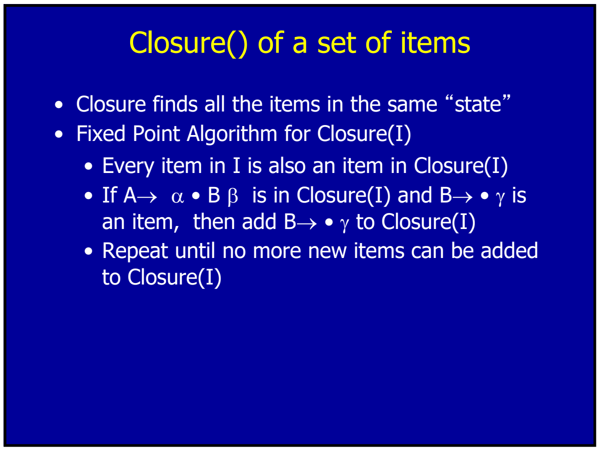
105



Correlating Items and Actions

- If the current state contains the item A → α • c β and the current symbol in the input buffer is c
  - Parser shifts c onto stack
  - Next state will contain A → α c • β
- If the current state contains the item A → **α** •
  - Parser reduces using A → **α**
- If the current state contains the item S → **α** • $ and the input buffer is empty
  - Parser accepts input

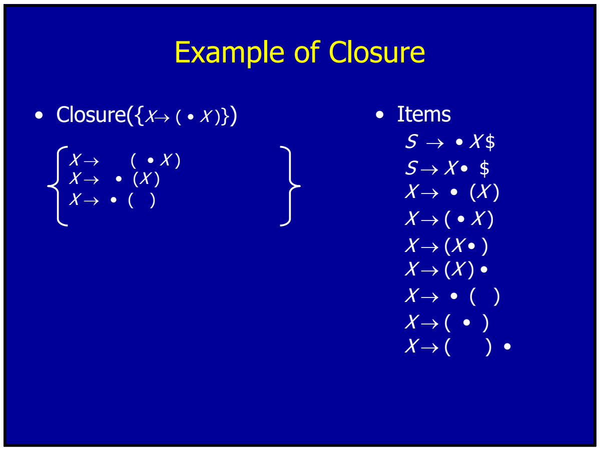
106



Closure() of a set of items

- Closure finds all the items in the same “state”
- Fixed Point Algorithm for Closure(I)
  - Every item in I is also an item in Closure(I)
  - If A→ α • B β is in Closure(I) and B→ • γ is an item, then add B→ • γ to Closure(I)
  - Repeat until no more new items can be added to Closure(I)

107



Example of Closure

- Closure($\{X \to (\bullet X)\}$)

$$\left\{\begin{array}{l} X \to (\bullet X) \\ X \to \bullet\, (X) \\ X \to \bullet\, (\;) \end{array}\right\}$$

- Items

  $S \to \bullet X\$$
  $S \to X \bullet \$$
  $X \to \bullet\, (X)$
  $X \to (\bullet X)$
  $X \to (X \bullet)$
  $X \to (X) \bullet$
  $X \to \bullet\, (\;)$
  $X \to (\bullet)$
  $X \to (\;) \bullet$

108

Another Example

- Closure($\{S \rightarrow \bullet X\$\}$)

$$\left\{\begin{array}{l} S \rightarrow \ \bullet X\$ \\ X \rightarrow \ \bullet (X) \\ X \rightarrow \bullet (\ ) \end{array}\right\}$$

- Items

$S \rightarrow \bullet X\$$
$S \rightarrow X \bullet \ \$$
$X \rightarrow \bullet \ (X)$
$X \rightarrow (\bullet X)$
$X \rightarrow (X \bullet)$
$X \rightarrow (X) \bullet$
$X \rightarrow \bullet \ (\ \ )$
$X \rightarrow (\ \bullet \ )$
$X \rightarrow (\ \ \ ) \ \bullet$

109

Goto() of a set of items

- Goto finds the new state after consuming a grammar symbol while at the current state
- Algorithm for Goto(I, X)
  where I is a set of items
  and X is a grammar symbol

Goto(I, X) = Closure({ A→ α X • β | A→ α • X β in I })

- goto is the new set obtained by “moving the dot” over X

110

Example of Goto

- Goto ($\{X \rightarrow (\bullet X)\}, X$)

$$\left\{ X \rightarrow (X \bullet \ ) \right\}$$

- Items

$S \rightarrow \bullet X\$$
$S \rightarrow X \bullet \ \$$
$X \rightarrow \bullet \ (X)$
$X \rightarrow (\bullet X)$
$X \rightarrow (X \bullet)$
$X \rightarrow (X) \ \bullet$
$X \rightarrow \bullet \ (\ \ )$
$X \rightarrow (\ \bullet \ )$
$X \rightarrow (\ \ \ ) \ \bullet$

111

Another Example of Goto

- Goto ($\{X \rightarrow \bullet (X)\}, ($)

$$\left\{\begin{array}{l} X \rightarrow \ \ (\ \bullet X) \\ X \rightarrow \ \bullet \ (X) \\ X \rightarrow \bullet \ (\ \ ) \end{array}\right\}$$

- Items

$S \rightarrow \bullet X\$$
$S \rightarrow X \bullet \ \$$
$X \rightarrow \bullet \ (X)$
$X \rightarrow (\bullet X)$
$X \rightarrow (X \bullet)$
$X \rightarrow (X) \ \bullet$
$X \rightarrow \bullet \ (\ \ )$
$X \rightarrow (\ \bullet \ )$
$X \rightarrow (\ \ \ ) \ \bullet$

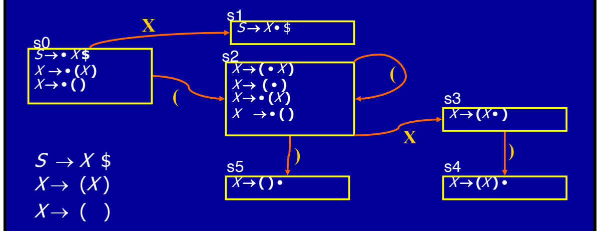
112

Building the DFA states

- Start with the item $S \rightarrow \bullet \beta \$$
- Create the first state to be Closure($\{ S \rightarrow \bullet \beta \$\}$)
- Pick a state I
  - for each item A→ α • X β in I
    - If there exists an edge X from state I to state J, then add Goto(I,X) to J
    - Otherwise make a new state J, add edge X from state I to state J, and add Goto(I,X) to J
- Repeat until no more additions possible

113

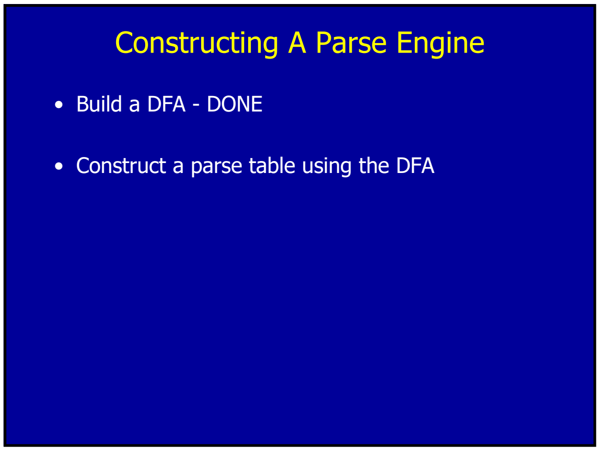
DFA Example



s0
$S \rightarrow \bullet X\$$
$X \rightarrow \bullet (X)$
$X \rightarrow \bullet (\,)$
X
s1
$S \rightarrow X \bullet \$$
(
s2
$X \rightarrow (\bullet X)$
$X \rightarrow (\bullet)$
$X \rightarrow \bullet (X)$
$X \ \rightarrow \bullet (\,)$
(
X
s3
$X \rightarrow (X \bullet)$
)
)
s5
$X \rightarrow (\,) \bullet$
s4
$X \rightarrow (X) \bullet$

$S \rightarrow X \ \$$
$X \rightarrow \ (X)$
$X \rightarrow \ (\ \ )$

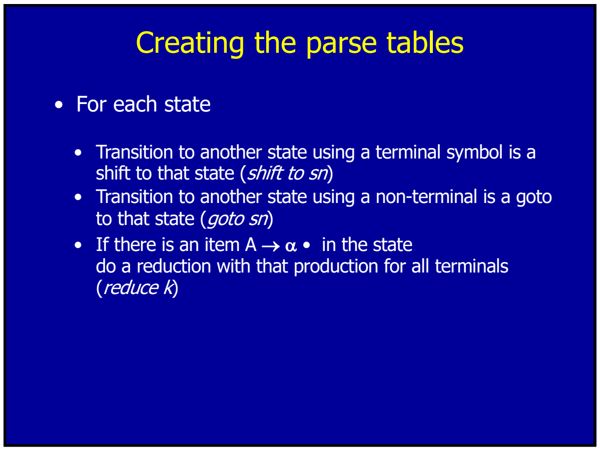
114



Constructing A Parse Engine

- Build a DFA - DONE
- Construct a parse table using the DFA

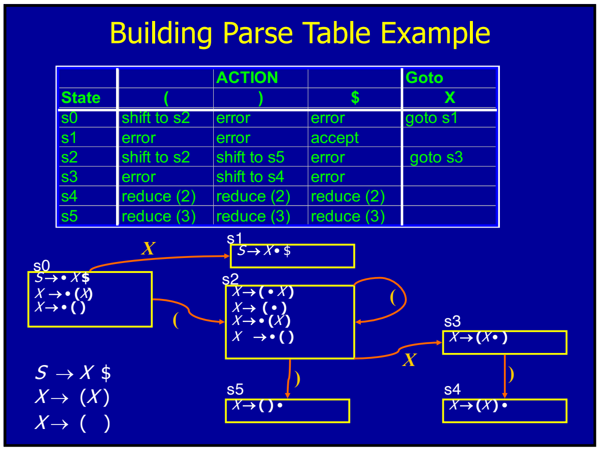
115



Creating the parse tables

- For each state
  - Transition to another state using a terminal symbol is a shift to that state (*shift to sn*)
  - Transition to another state using a non-terminal is a goto to that state (*goto sn*)
  - If there is an item A → α • in the state do a reduction with that production for all terminals (*reduce k*)

116

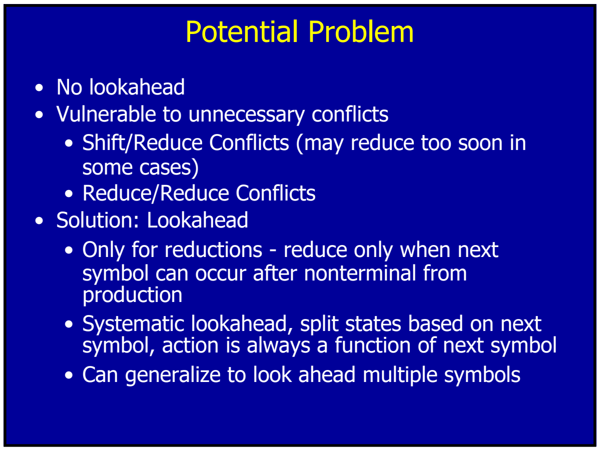


Building Parse Table Example

		ACTION		Goto
State	(	)	$	X
s0	shift to s2	error	error	goto s1
s1	error	error	accept	
s2	shift to s2	shift to s5	error	goto s3
s3	error	shift to s4	error	
s4	reduce (2)	reduce (2)	reduce (2)	
s5	reduce (3)	reduce (3)	reduce (3)	

$S \to X\ \$$
$X \to (X)$
$X \to (\ \ )$

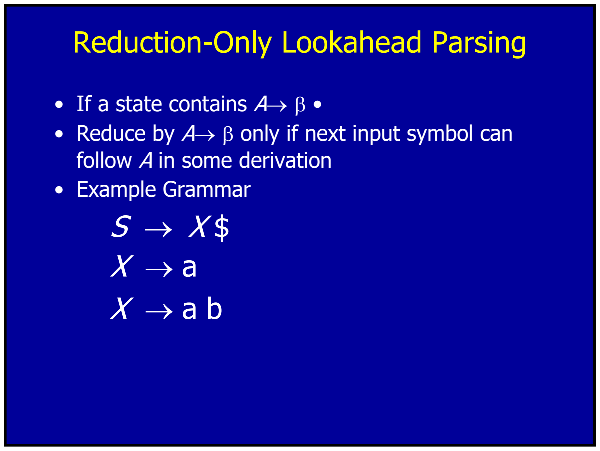
117



Potential Problem

- No lookahead
- Vulnerable to unnecessary conflicts
  - Shift/Reduce Conflicts (may reduce too soon in some cases)
  - Reduce/Reduce Conflicts
- Solution: Lookahead
  - Only for reductions - reduce only when next symbol can occur after nonterminal from production
  - Systematic lookahead, split states based on next symbol, action is always a function of next symbol
  - Can generalize to look ahead multiple symbols

118

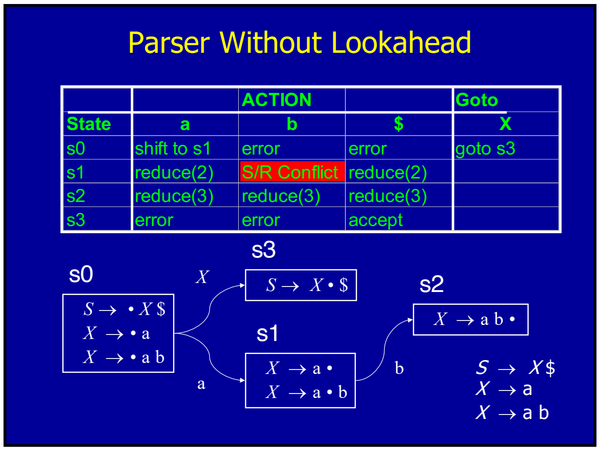


Reduction-Only Lookahead Parsing

- If a state contains *A*→ β •
- Reduce by *A*→ β only if next input symbol can follow *A* in some derivation
- Example Grammar

$S \to X\$$
$X \to \mathrm{a}$
$X \to \mathrm{a\ b}$

119

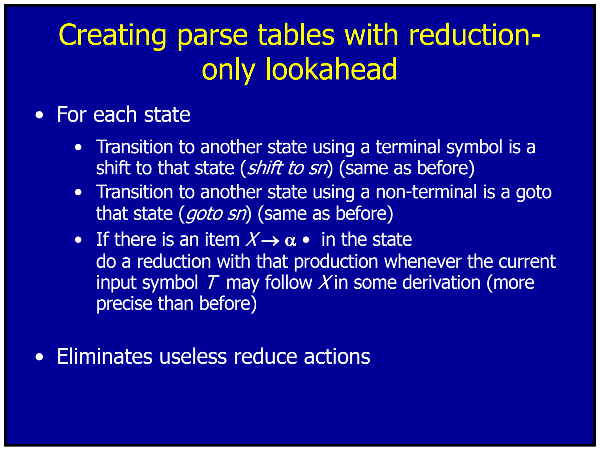


Parser Without Lookahead

		ACTION		Goto
State	a	b	$	X
s0	shift to s1	error	error	goto s3
s1	reduce(2)	S/R Conflict	reduce(2)	
s2	reduce(3)	reduce(3)	reduce(3)	
s3	error	error	accept	

$S \to X\$$
$X \to \mathrm{a}$
$X \to \mathrm{a\ b}$

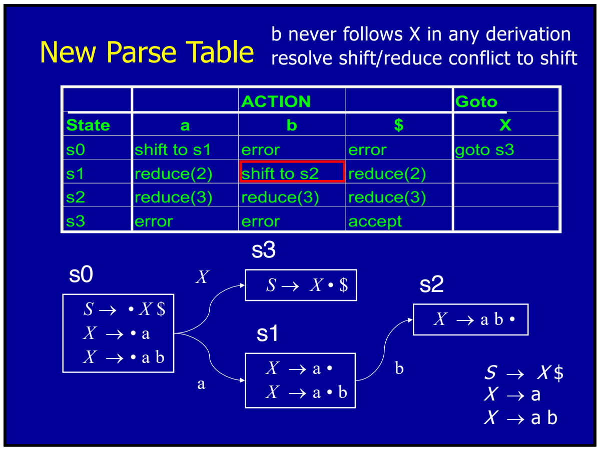
120



Creating parse tables with reduction-only lookahead

- For each state
  - Transition to another state using a terminal symbol is a shift to that state (*shift to sn*) (same as before)
  - Transition to another state using a non-terminal is a goto that state (*goto sn*) (same as before)
  - If there is an item $X \rightarrow \alpha \bullet$ in the state do a reduction with that production whenever the current input symbol T may follow X in some derivation (more precise than before)
- Eliminates useless reduce actions

121



New Parse Table

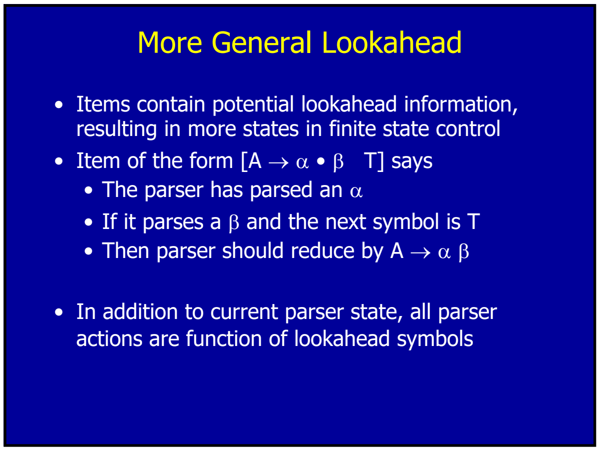
b never follows X in any derivation
resolve shift/reduce conflict to shift

		ACTION		Goto
State	**a**	**b**	**$**	**X**
s0	shift to s1	error	error	goto s3
s1	reduce(2)	shift to s2	reduce(2)	
s2	reduce(3)	reduce(3)	reduce(3)	
s3	error	error	accept	

s0
$S \rightarrow \bullet X \$$
$X \rightarrow \bullet a$
$X \rightarrow \bullet a b$
X
a
s3
$S \rightarrow X \bullet \$$
s1
$X \rightarrow a \bullet$
$X \rightarrow a \bullet b$
b
s2
$X \rightarrow a b \bullet$

$S \rightarrow X\$$
$X \rightarrow a$
$X \rightarrow a b$

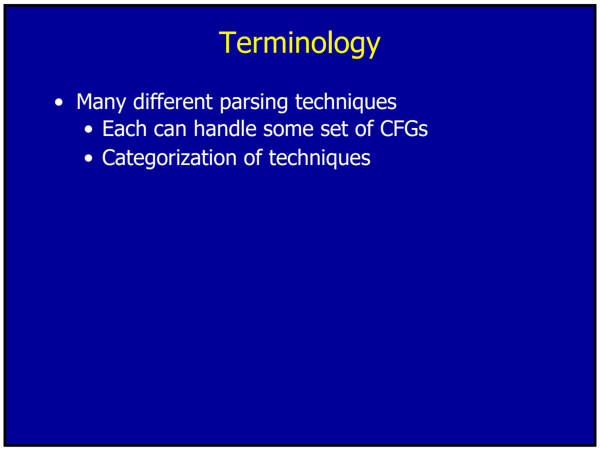
122



More General Lookahead

- Items contain potential lookahead information, resulting in more states in finite state control
- Item of the form $[A \rightarrow \alpha \bullet \beta \quad T]$ says
  - The parser has parsed an α
  - If it parses a β and the next symbol is T
  - Then parser should reduce by $A \rightarrow \alpha \beta$
- In addition to current parser state, all parser actions are function of lookahead symbols

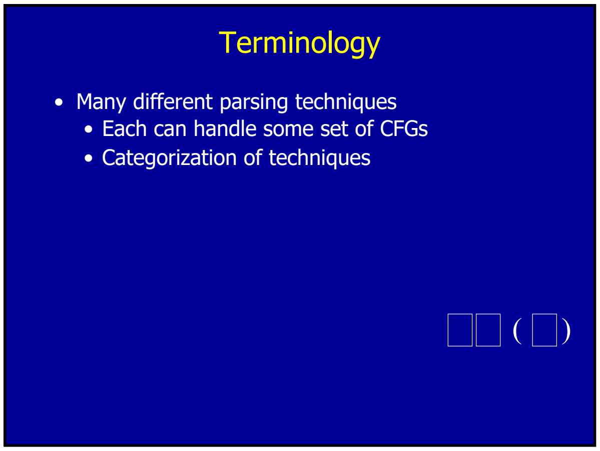
123



Terminology

- Many different parsing techniques
  - Each can handle some set of CFGs
  - Categorization of techniques

124

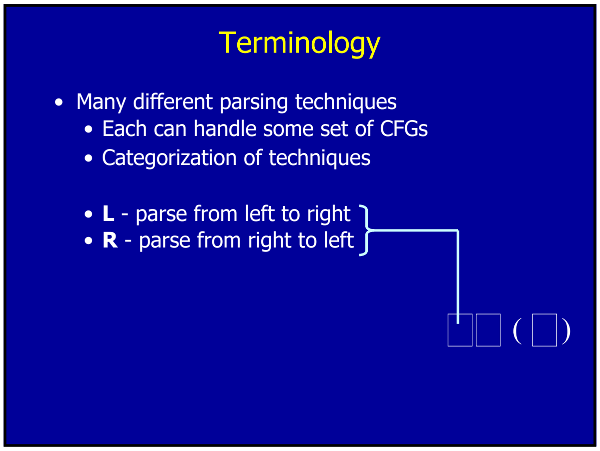


Terminology

- Many different parsing techniques
  - Each can handle some set of CFGs
  - Categorization of techniques

□□ (□)

125

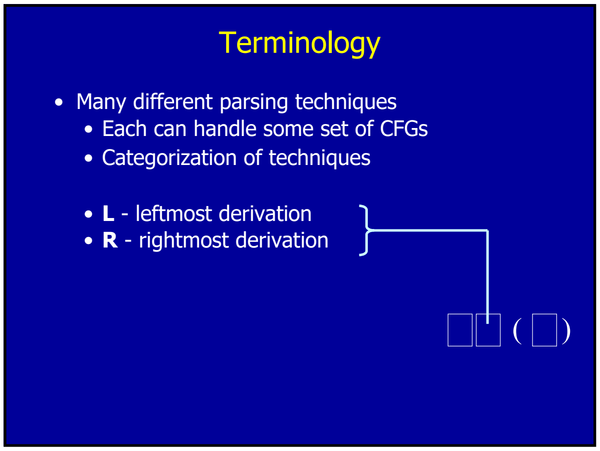


Terminology

- Many different parsing techniques
  - Each can handle some set of CFGs
  - Categorization of techniques
  - **L** - parse from left to right
  - **R** - parse from right to left

□□ (□)

126

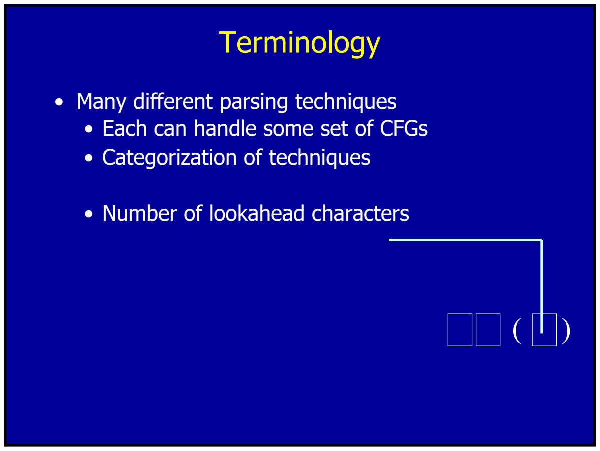


Terminology

- Many different parsing techniques
  - Each can handle some set of CFGs
  - Categorization of techniques
  - **L** - leftmost derivation
  - **R** - rightmost derivation

□□ (□)

127

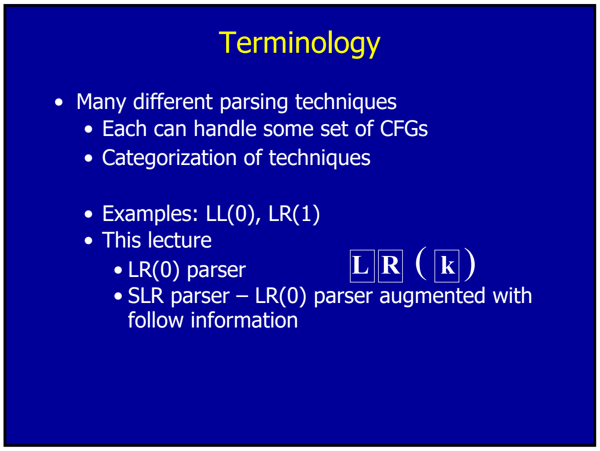


Terminology

- Many different parsing techniques
  - Each can handle some set of CFGs
  - Categorization of techniques
  - Number of lookahead characters

□□ (□)

128

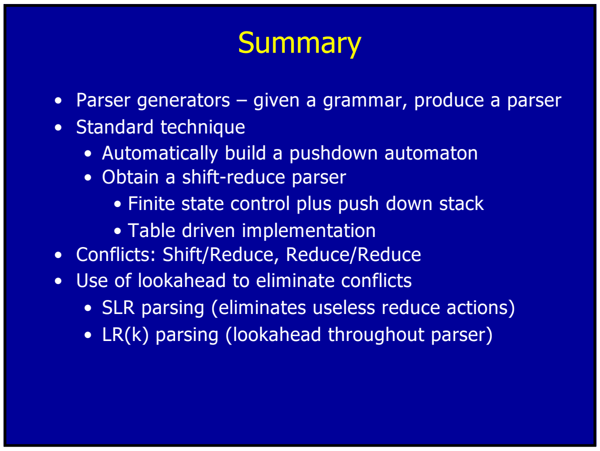


Terminology

- Many different parsing techniques
  - Each can handle some set of CFGs
  - Categorization of techniques
  - Examples: LL(0), LR(1)
  - This lecture
    - LR(0) parser
    - SLR parser – LR(0) parser augmented with follow information

$\boxed{\mathbf{L}}\boxed{\mathbf{R}}\,(\boxed{\mathbf{k}})$

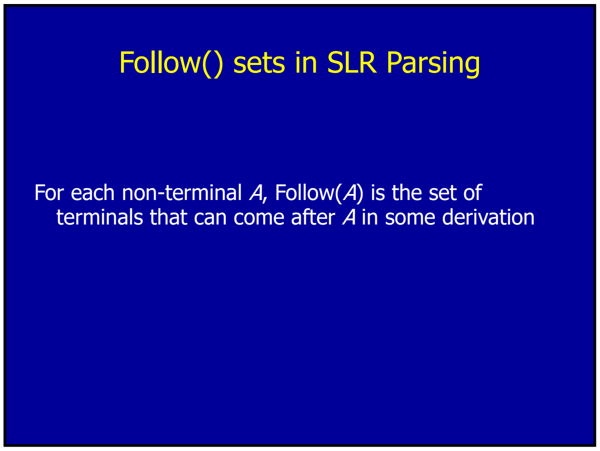
129



Summary

- Parser generators – given a grammar, produce a parser
- Standard technique
  - Automatically build a pushdown automaton
  - Obtain a shift-reduce parser
    - Finite state control plus push down stack
    - Table driven implementation
- Conflicts: Shift/Reduce, Reduce/Reduce
- Use of lookahead to eliminate conflicts
  - SLR parsing (eliminates useless reduce actions)
  - LR(k) parsing (lookahead throughout parser)

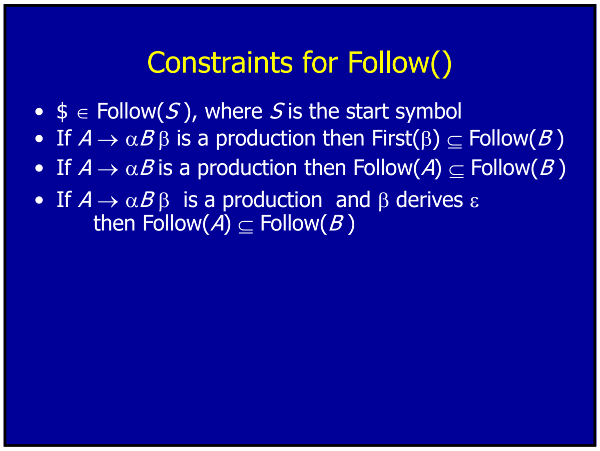
130



Follow() sets in SLR Parsing

For each non-terminal *A*, Follow(*A*) is the set of terminals that can come after *A* in some derivation

131



Constraints for Follow()

- $ $\in$ Follow(S), where S is the start symbol
- If $A \to \alpha B \beta$ is a production then First(β) $\subseteq$ Follow(B)
- If $A \to \alpha B$ is a production then Follow(A) $\subseteq$ Follow(B)
- If $A \to \alpha B \beta$ is a production and β derives ε then Follow(A) $\subseteq$ Follow(B)

132

Algorithm for Follow

```
for all nonterminals NT
   Follow(NT) = {}
Follow(S) = { $ }
while Follow sets keep changing
   for all productions A → αBβ
        Follow(B) = Follow(B) ∪ First(β)
        if (β derives ε) Follow(B) = Follow(B)∪Follow(A)
   for all productions A → αB
        Follow(B) = Follow(B)∪Follow(A)
```

133

Augmenting Example with Follow

- Example Grammar for Follow

$S \rightarrow X\$$

$X \rightarrow \mathrm{a}$

$X \rightarrow \mathrm{a\ b}$

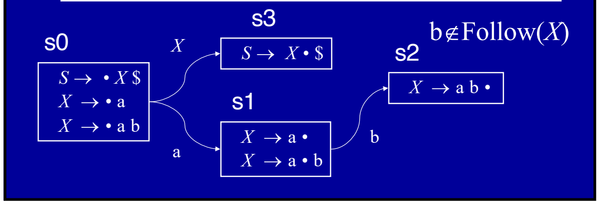
Follow(S) = { $ }

Follow(X) = { $ }

134

SLR Eliminates Shift/Reduce Conflict

		ACTION		Goto
State	a	b	$	X
s0	shift to s1	error	error	goto s3
s1	reduce(2)	shift to s2	reduce(2)	
s2	reduce(3)	reduce(3)	reduce(3)	
s3	error	error	accept	



135

Basic Idea Behind LR(1)

- Split states in LR(0) DFA based on lookahead
- Reduce based on item and lookahead

136

LR(1) Items

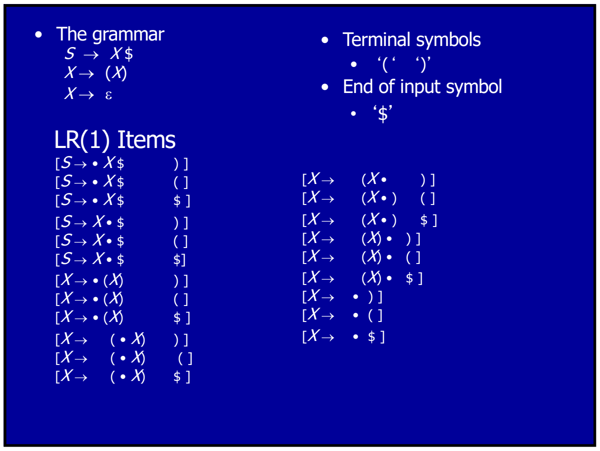
- Items will keep info on
  - production
  - right-hand-side position (the dot)
  - look ahead symbol
- LR(1) item is of the form [A → α • β   T]
  - A → α β is a production
  - The dot in A → α • β denotes the position
  - T is a terminal or the end marker ($)

137

Meaning of LR(1) Items

- Item [A → α • β   T] means
  - The parser has parsed an α
  - If it parses a β and the next symbol is T
  - Then parser should reduce by A → α β

138



- The grammar
  $S \rightarrow X\$$
  $X \rightarrow (X)$
  $X \rightarrow \varepsilon$

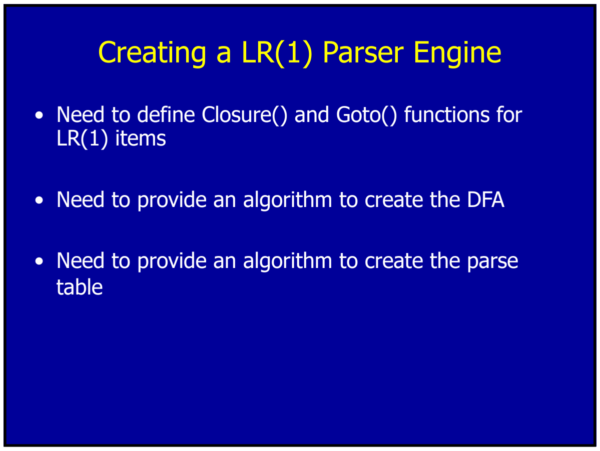
- Terminal symbols
  - '(' ')'
- End of input symbol
  - '\$'

LR(1) Items

$[S \rightarrow \bullet X\$ \quad)]$
$[S \rightarrow \bullet X\$ \quad (]$
$[S \rightarrow \bullet X\$ \quad \$]$
$[S \rightarrow X \bullet \$ \quad)]$
$[S \rightarrow X \bullet \$ \quad (]$
$[S \rightarrow X \bullet \$ \quad \$]$
$[X \rightarrow \bullet (X) \quad)]$
$[X \rightarrow \bullet (X) \quad (]$
$[X \rightarrow \bullet (X) \quad \$]$
$[X \rightarrow (\bullet X) \quad)]$
$[X \rightarrow (\bullet X) \quad (]$
$[X \rightarrow (\bullet X) \quad \$]$

$[X \rightarrow (X \bullet \quad)]$
$[X \rightarrow (X \bullet) \quad (]$
$[X \rightarrow (X \bullet) \quad \$]$
$[X \rightarrow (X) \bullet \quad)]$
$[X \rightarrow (X) \bullet \quad (]$
$[X \rightarrow (X) \bullet \quad \$]$
$[X \rightarrow \bullet \quad)]$
$[X \rightarrow \bullet \quad (]$
$[X \rightarrow \bullet \quad \$]$

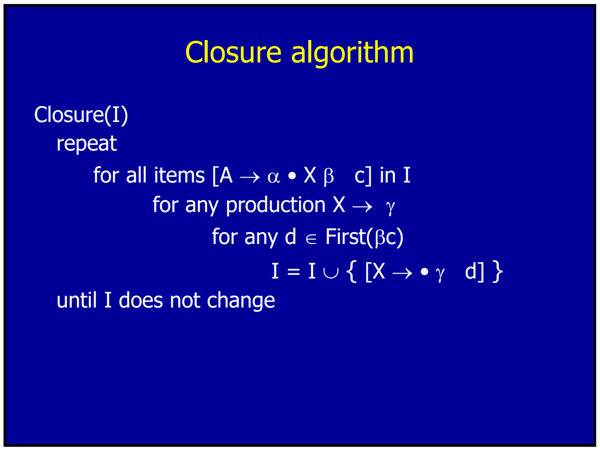
139



Creating a LR(1) Parser Engine

- Need to define Closure() and Goto() functions for LR(1) items
- Need to provide an algorithm to create the DFA
- Need to provide an algorithm to create the parse table

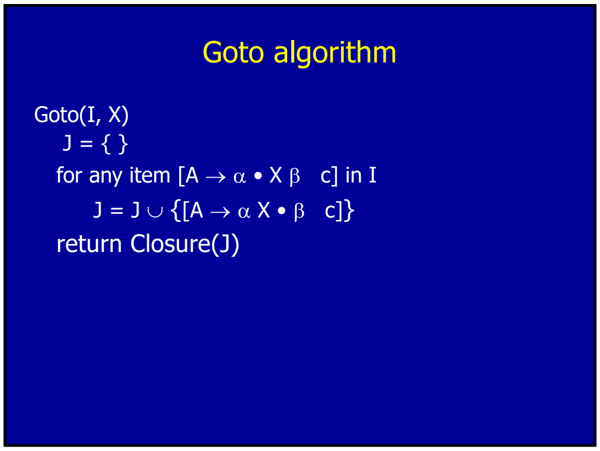
140



Closure algorithm

```
Closure(I)
  repeat
     for all items $[A \rightarrow \alpha \bullet X \beta \quad c]$ in I
          for any production $X \rightarrow \gamma$
               for any $d \in First(\beta c)$
                    $I = I \cup \{ [X \rightarrow \bullet \gamma \quad d] \}$
  until I does not change
```

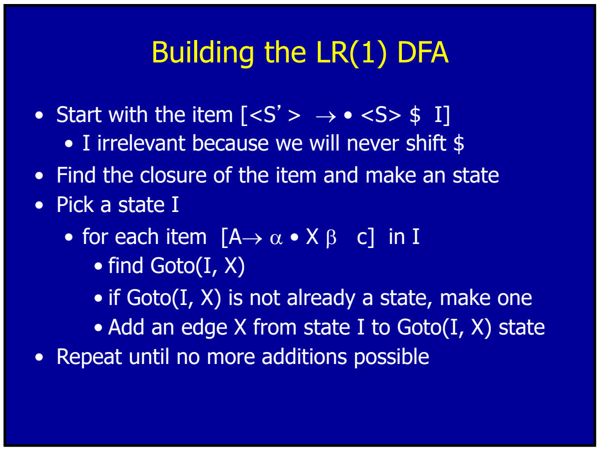
141



Goto algorithm

```
Goto(I, X)
  J = { }
  for any item $[A \rightarrow \alpha \bullet X \beta \quad c]$ in I
     $J = J \cup \{[A \rightarrow \alpha X \bullet \beta \quad c]\}$
  return Closure(J)
```

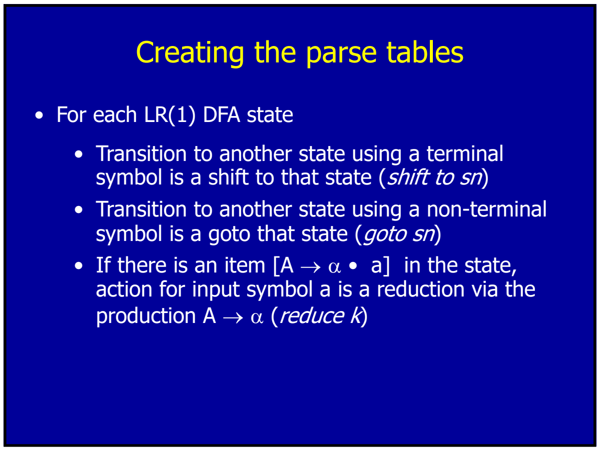
142



Building the LR(1) DFA

- Start with the item [<S' > → • <S> \$ I]
  - I irrelevant because we will never shift \$
- Find the closure of the item and make an state
- Pick a state I
  - for each item $[A \rightarrow \alpha \bullet X \beta \quad c]$ in I
    - find Goto(I, X)
    - if Goto(I, X) is not already a state, make one
    - Add an edge X from state I to Goto(I, X) state
- Repeat until no more additions possible

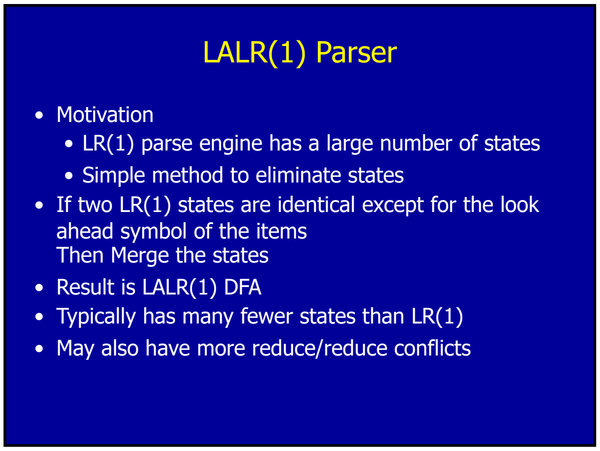
143



Creating the parse tables

- For each LR(1) DFA state
  - Transition to another state using a terminal symbol is a shift to that state (*shift to sn*)
  - Transition to another state using a non-terminal symbol is a goto that state (*goto sn*)
  - If there is an item $[A \rightarrow \alpha \bullet \quad a]$ in the state, action for input symbol a is a reduction via the production $A \rightarrow \alpha$ (*reduce k*)

144



LALR(1) Parser

- Motivation
  - LR(1) parse engine has a large number of states
  - Simple method to eliminate states
- If two LR(1) states are identical except for the look ahead symbol of the items
  Then Merge the states
- Result is LALR(1) DFA
- Typically has many fewer states than LR(1)
- May also have more reduce/reduce conflicts

145