中山大學 SUN YAT-SEN UNIVERSITY
计算机学院（软件学院）
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Compilation Principle
编 译 原 理

第11讲：语法分析(8)

张献伟

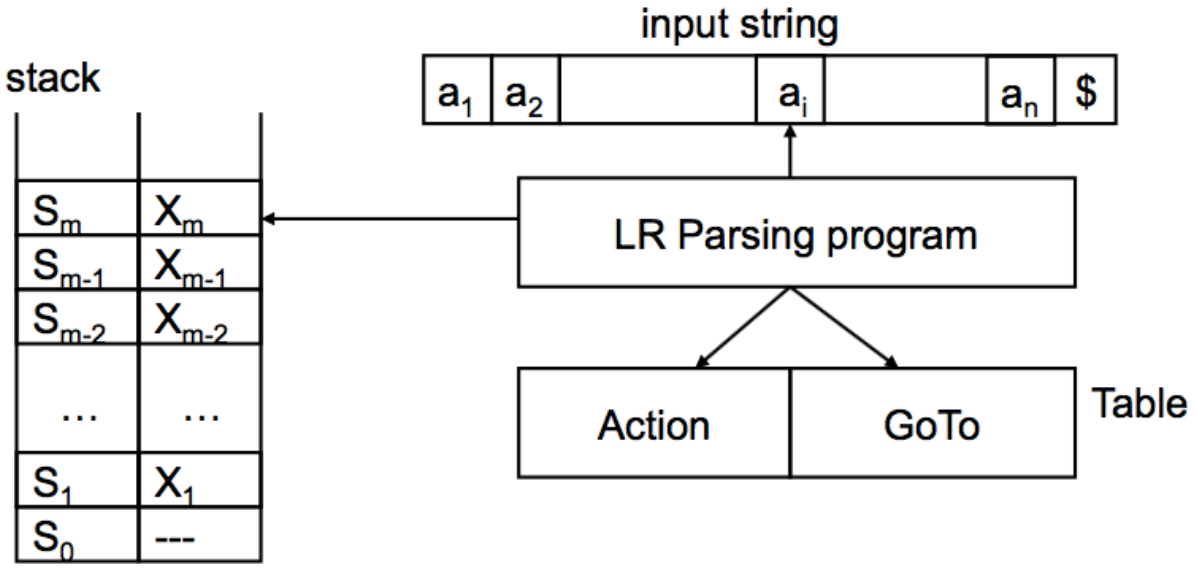
xianweiz.github.io

DCS290, 3/30/2023

Review Questions

- Q1: what are the operations in bottom-up parsing?
  - Shift: move a token from buffer into stack
  - Reduce: reversely apply a production (pop rhs, push lhs)
- Q2: when to reduce?
  - When there is a handle at the stack top.
- Q3: how to recognize a handle?
  - Right sentential form - phase - simple phase - leftmost simple phase
- Q4: what does LR(k) mean?
  - L: scan input from left to right
  - R: construct a rightmost derivation in reverse
  - k: number of symbols to lookahead
- Q5: what are components of LR parser?
  - Input buffer, stack, parse table, driver

LR Parser



- The stack holds a sequence of states, $s_0 s_1 \ldots s_m$ (s_m is the top)
  - States are to track where we are in a parse
  - Each grammar symbol X_i is associated with a state s_i
- Contents of stack + input ($X_1 X_2 \ldots X_m a_i \ldots a_n$) is a right sentential form
  - If the input string is a member of the language
- Uses $[S_m, a_i]$ to index into parsing table to determine action

Possible Actions[可能动作]

- **Shift**
  - Transfer the next input symbol onto the top of the stack
- **Reduce**
  - If there's a rule $A \rightarrow w$, and if the contents of stack are qw for some q (q may be empty), then we can reduce the stack to qA
- **Accept**
  - The special case of reduce: reducing the entire contents of stack to the start symbol with no remaining input[完全归约到开始符号]
  - Last step in a successful parse: have recognized input as a valid sentence[输入串被识别为符合语法]
- **Error**
  - Cannot reduce, and shifting would create a sequence on the stack that cannot eventually be reduced to the start symbol

Possible Actions (cont.)

- Grammar

  S → E

  E → T | E + T

  T → id | (E)

- Input: (id + id)
  - #(id + id)$ => (id#+ id)$ => (T#+ id)$ => (E#+ id)$ => (E+id#)$ => (E+T#)$ => (E#)$ => (E)#$ => T#$ => E#$ => S#$

    Accept

- Input: id+)
  - #id+)$ => id#+)$ => T#+)$ => E#+)$ => E+#)$ …

    Error

Example: Parse Table

Grammar:

(1) S → BB

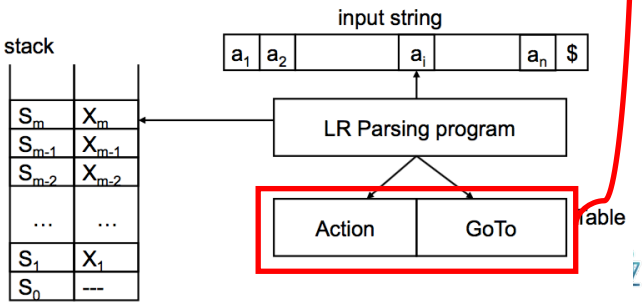
(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

- Table entry:
  - s*i*: shifts the input symbol and moves to state *i* (i.e., push state on stack)
  - r*j*: reduce by production numbered *j*
  - acc: accept
  - blank: error



Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

state → 0

b a b

symbol → $

b a b $

Tree

Stack

Buffer

Example: Parse Table (cont.)

Grammar:

(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

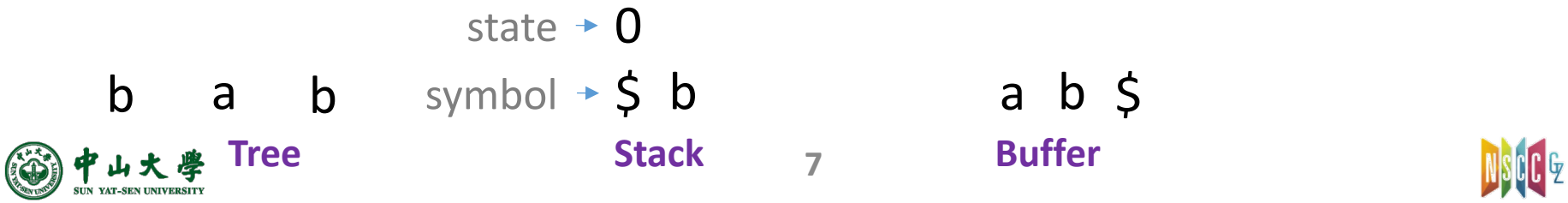
Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



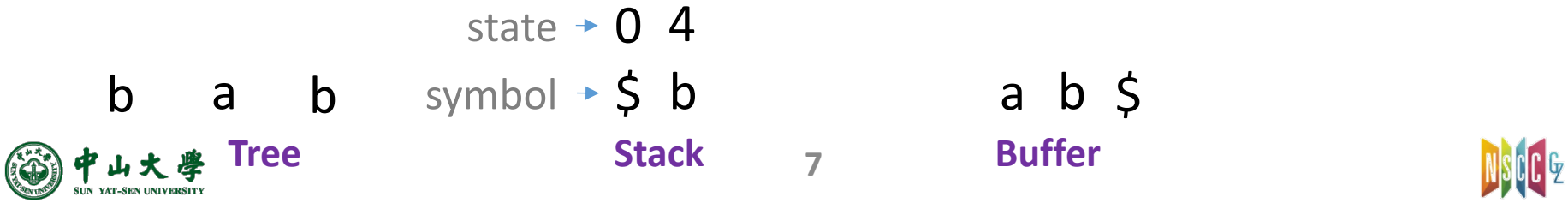
Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Example: Parse Table (cont.)

Grammar:

(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

state → 0 4

b a b symbol → $ b a b $

Tree Stack Buffer

Example: Parse Table (cont.)

Grammar:

(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

Example: Parse Table (cont.)

Grammar:

(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

b a b

Tree

state → 0 4

symbol → $ b

Stack

a b $

Buffer

Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

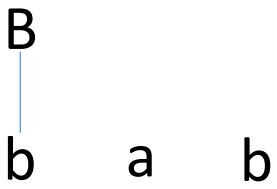
Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack

Buffer

Example: Parse Table (cont.)

Grammar:

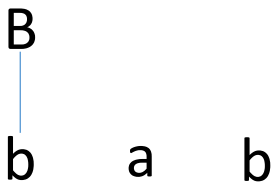
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack

Buffer

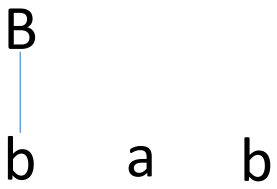
Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack

Buffer

Example: Parse Table (cont.)

Grammar:

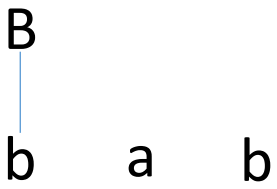
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack

Buffer

Example: Parse Table (cont.)

Grammar:

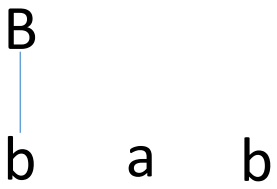
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack

Buffer

Example: Parse Table (cont.)

Grammar:

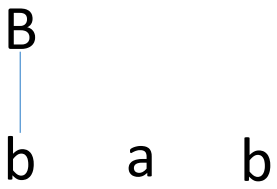
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack

Buffer

Example: Parse Table (cont.)

Grammar:

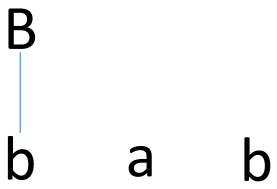
(1) S → BB

(2) B → aB

(3) B → b

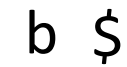
String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack



Buffer

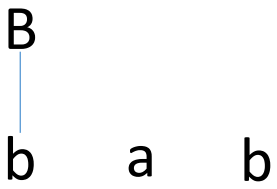
Example: Parse Table (cont.)

Grammar:

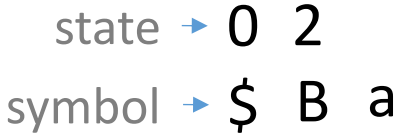
(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

Example: Parse Table (cont.)

Grammar:

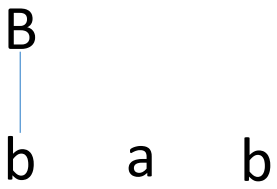
(1) S → BB

(2) B → aB

(3) B → b

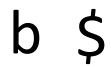
String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack



Buffer

Example: Parse Table (cont.)

Grammar:

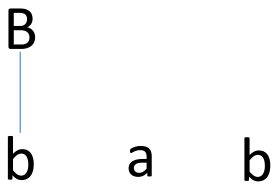
(1) S → BB

(2) B → aB

(3) B → b

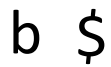
String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack



Buffer

Example: Parse Table (cont.)

Grammar:

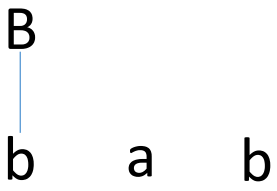
(1) S → BB

(2) B → aB

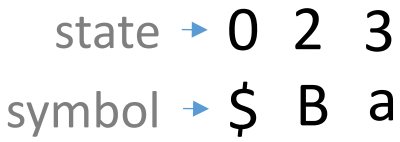
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

Example: Parse Table (cont.)

Grammar:

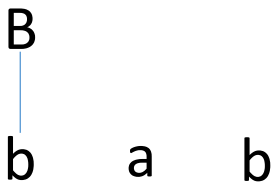
(1) S → BB

(2) B → aB

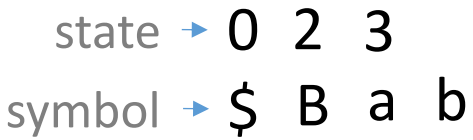
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

Example: Parse Table (cont.)

Grammar:

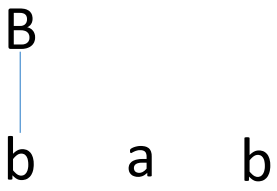
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack

Buffer

Example: Parse Table (cont.)

Grammar:

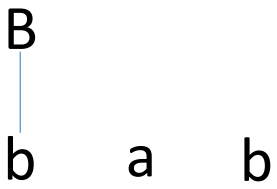
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack

Buffer

Example: Parse Table (cont.)

Grammar:

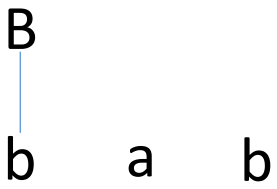
(1) S → BB

(2) B → aB

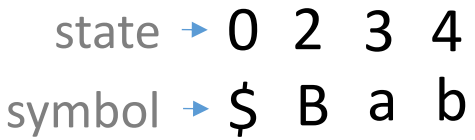
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

Example: Parse Table (cont.)

Grammar:

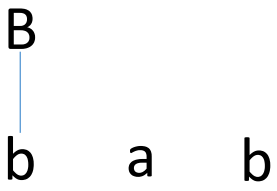
(1) S → BB

(2) B → aB

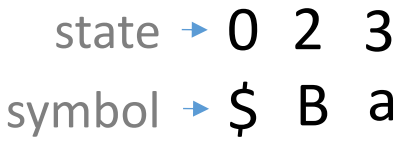
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

Example: Parse Table (cont.)

Grammar:

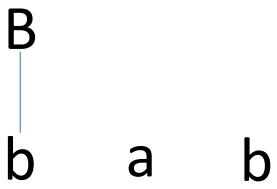
(1) S → BB

(2) B → aB

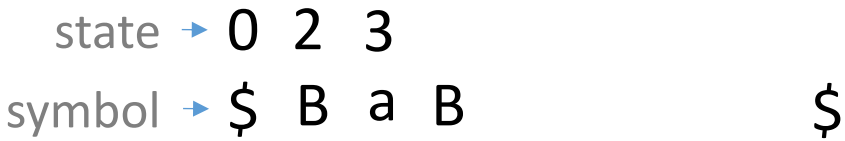
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

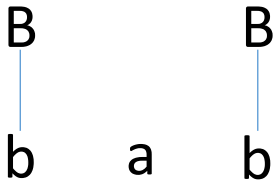
Example: Parse Table (cont.)

Grammar:

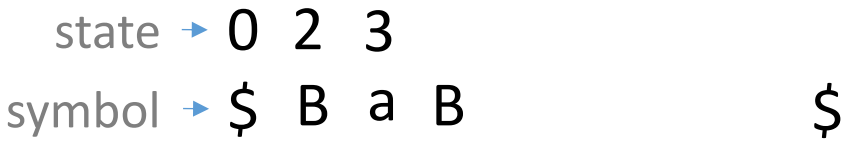
(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

$

Buffer

Example: Parse Table (cont.)

Grammar:

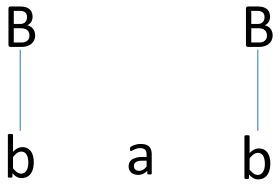
(1) S → BB

(2) B → aB

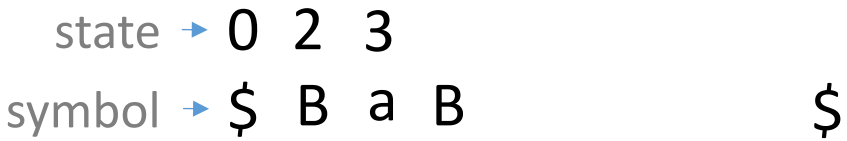
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

$

Buffer

Example: Parse Table (cont.)

Grammar:

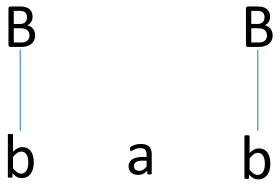
(1) S → BB

(2) B → aB

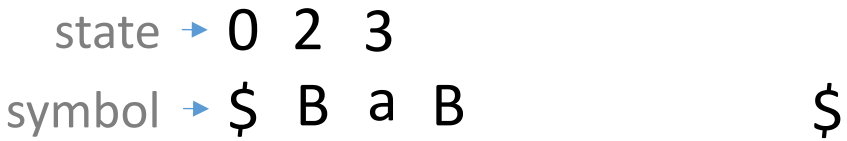
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

Example: Parse Table (cont.)

Grammar:

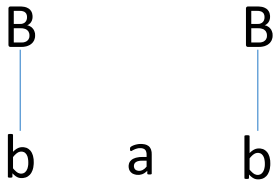
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack

Buffer

Example: Parse Table (cont.)

Grammar:

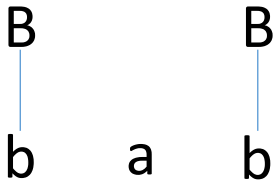
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack

Buffer

Example: Parse Table (cont.)

Grammar:

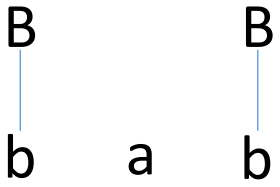
(1) S → BB

(2) B → aB

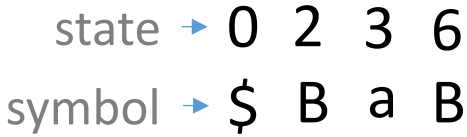
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

Example: Parse Table (cont.)

Grammar:

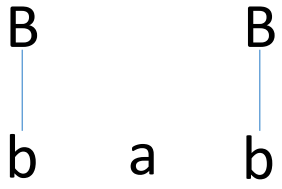
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

Stack

Buffer

Example: Parse Table (cont.)

Grammar:

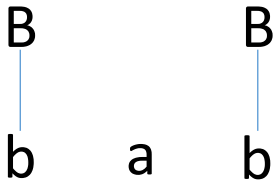
(1) S → BB

(2) B → aB

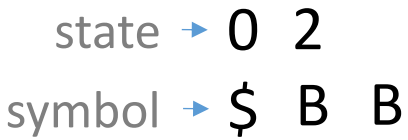
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

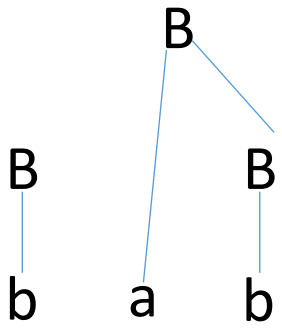
Example: Parse Table (cont.)

Grammar:

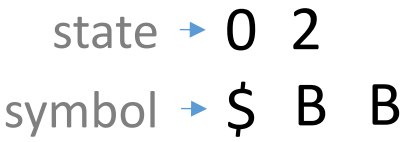
(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

Example: Parse Table (cont.)

Grammar:

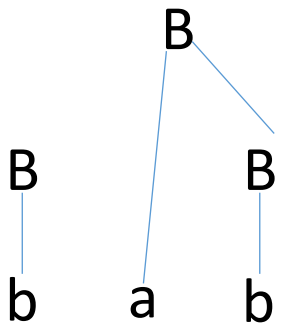
(1) S → BB

(2) B → aB

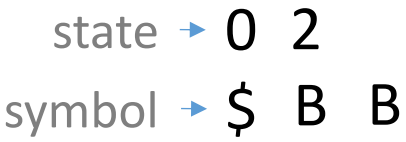
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

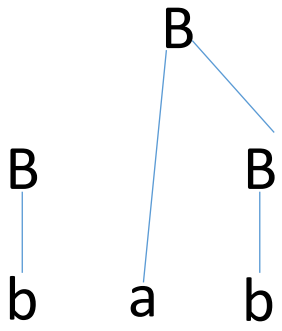
Example: Parse Table (cont.)

Grammar:

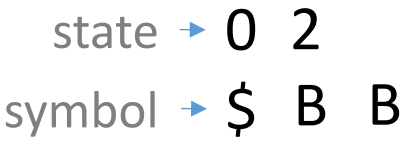
(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

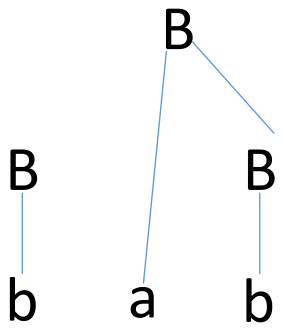
Example: Parse Table (cont.)

Grammar:

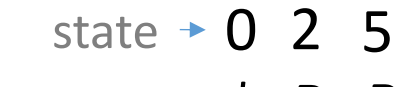
(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

Example: Parse Table (cont.)

Grammar:

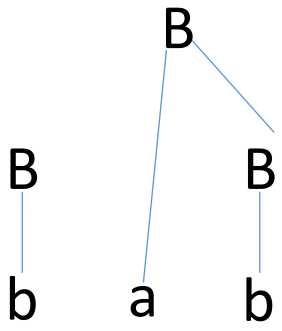
(1) S → BB

(2) B → aB

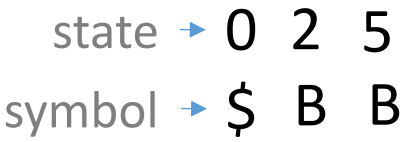
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

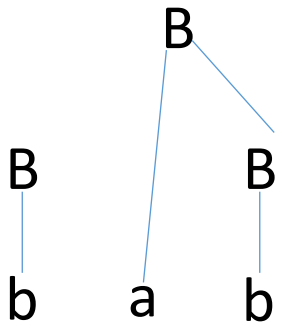
Example: Parse Table (cont.)

Grammar:

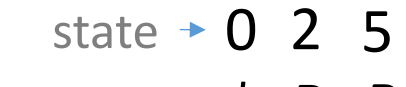
(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree



Stack

Buffer

Example: Parse Table (cont.)

Grammar:

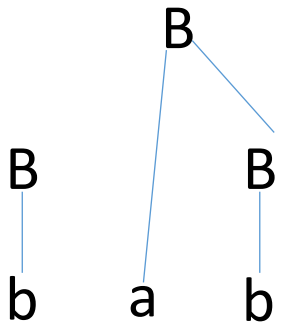
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

state → 0

symbol → $

Stack

$

Buffer

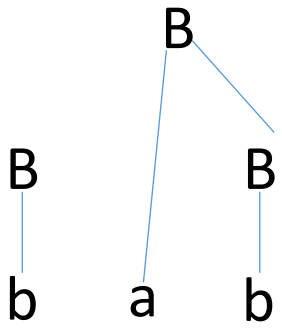
Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

state → 0
symbol → $ S

Stack

$

Buffer

Example: Parse Table (cont.)

Grammar:

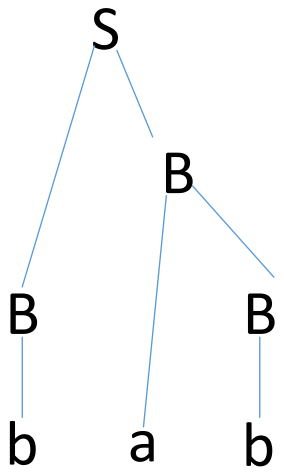
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

state → 0

symbol → $ S

Stack

$

Buffer

Example: Parse Table (cont.)

Grammar:

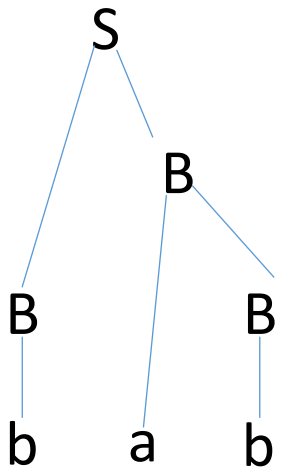
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



state → 0

symbol → $ S

$

Example: Parse Table (cont.)

Grammar:

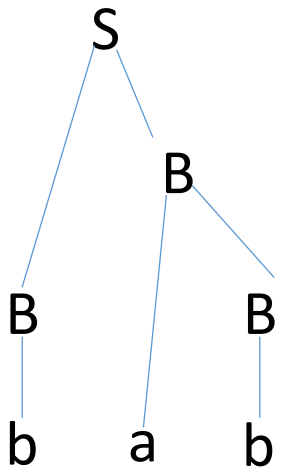
(1) S → BB

(2) B → aB

(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



Tree

state → 0

symbol → $ S

Stack

$

Buffer

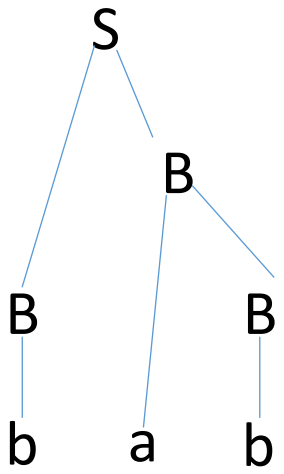
Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



state → 0 1
symbol → $ S

$

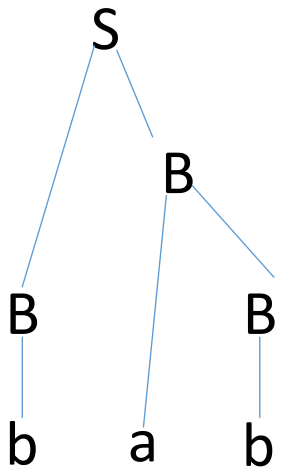
Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

String: bab

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		



state → 0 1
symbol → $ S

$

Parser Actions[解析动作]

Initial	s_0	
	$	$a_1a_2...a_n$$
General	$s_0s_1 \dots s_m$	
	$X_1...X_m$	$a_ia_{i+1}...a_n$$

- If ACTION[s_m, a_i] = sx, then do **shift**[移进]
  - Pushes a_i on stack
    - a_i is removed from input
  - Enters state x
    - i.e., pushes state x on stack
    - 自带下一状态

	$s_0s_1 \dots s_mx$	
	$X_1...X_ma_i$	$a_{i+1}...a_n$$

Parser Actions (cont.)

Initial	s_0 $ $a_1a_2...a_n$$
General	$s_0s_1 \dots s_m$ $$X_1...X_m$ $a_ia_{i+1}...a_n$$

- If ACTION[s_m, a_i] = r**x**, (i.e., the x^{th} production: $A \rightarrow X_{m-(k-1)}...X_m$), then do **reduce**[归约]
  - Pops *k* symbols from stack
  - Pushes *A* on stack
  - No change on input
  - GOTO[S_{m-k}, A] = y, then
    - 需寻找下一状态

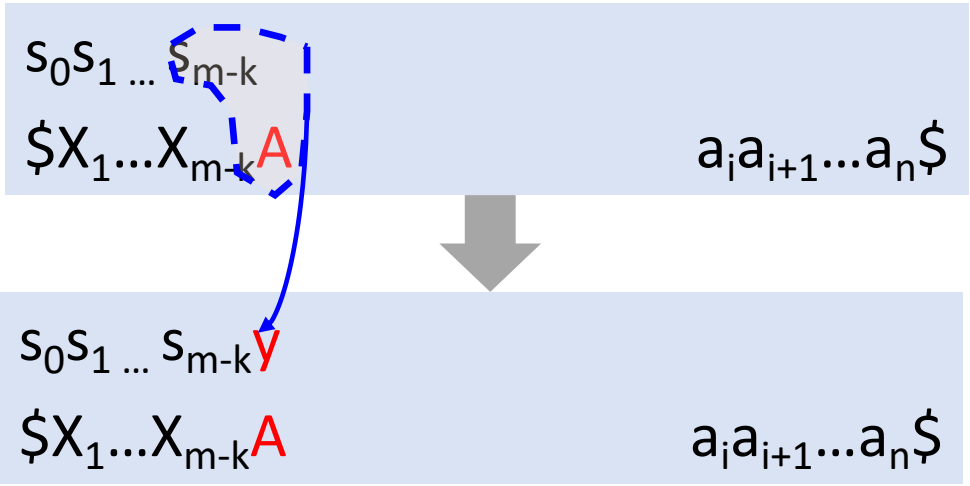
$s_0s_1 \dots s_{m-k}$
 $$X_1...X_{m-k}$A $a_ia_{i+1}...a_n$$

$s_0s_1 \dots s_{m-k}$**y**
 $$X_1...X_{m-k}$A $a_ia_{i+1}...a_n$$

Parser Actions (cont.)

Initial	s_0 $ $a_1a_2...a_n$
General	$s_0s_1 \dots s_m$ $ $a_ia_{i+1}...a_n$

- If ACTION[s_m, a_i] = r**x**, (i.e., the x^{th} production: A → $X_{m-(k-1)}...X_m$), then do **reduce**[归约]
  - Pops *k* symbols from stack
  - Pushes *A* on stack
  - No change on input
  - GOTO[S_{m-k}, A] = y, then
    - 需寻找下一状态

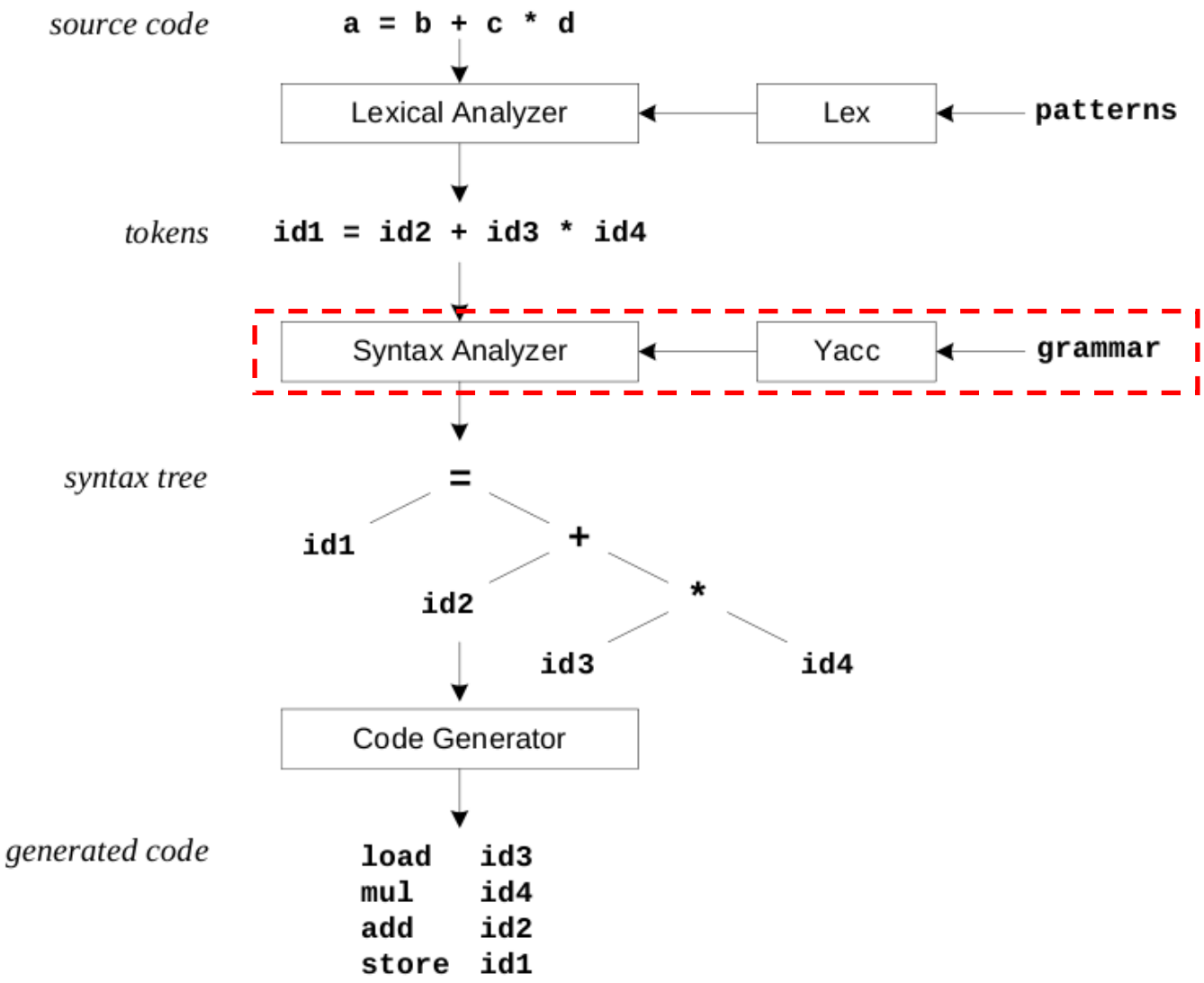


Parser Actions (cont.)

Initial	s_0	
	$	$a_1a_2...a_n$$
General	$s_0s_1 \dots s_m$	
	$$X_1...X_m$	$a_ia_{i+1}...a_n$$

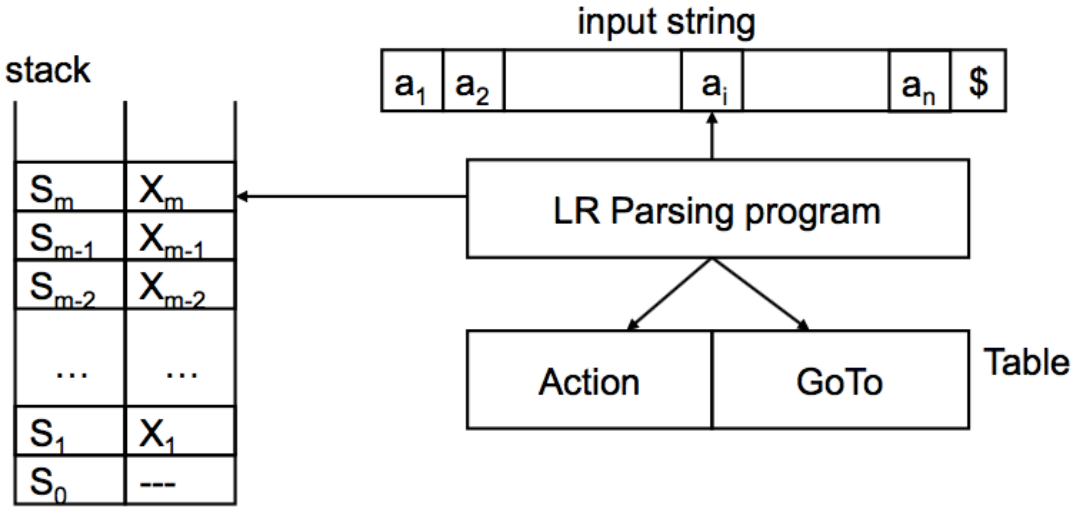
- If ACTION[s_m, a_i] = acc, then parsing is **complete**[接收]
- If ACTION[s_m, a_i] = <empty>, then report **error** and stop[报错]

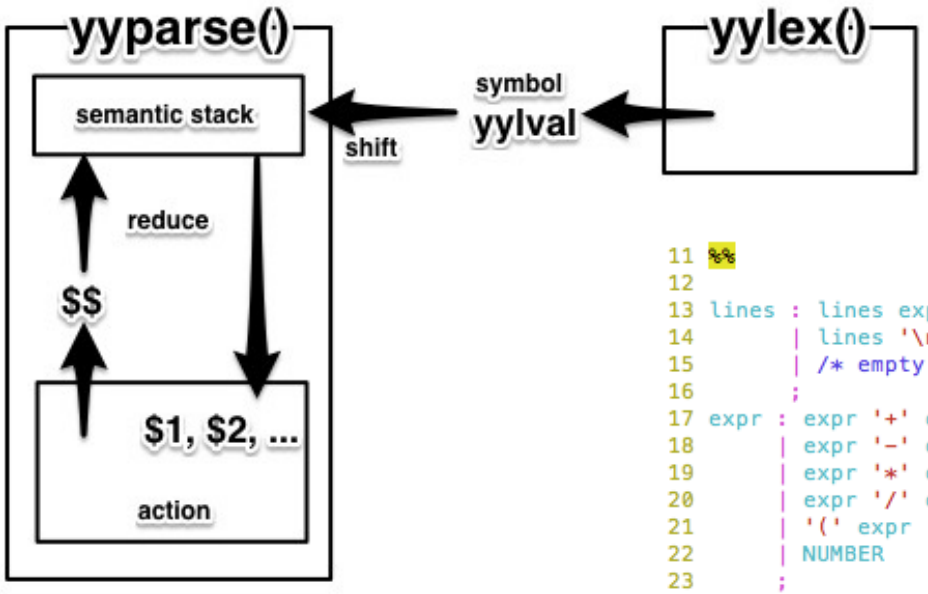
Parser in Practice



https://funningboy.blogspot.com/2010/08/lex-yacc-case-study-ply.html

Parser in Practice (cont.)






```
11 %%
12
13 lines : lines expr '\n' { printf("= %g\n", $2); }
14       | lines '\n'
15       | /* empty */
16       ;
17 expr : expr '+' expr { $$ = $1 + $3; }
18       | expr '-' expr { $$ = $1 - $3; }
19       | expr '*' expr { $$ = $1 * $3; }
20       | expr '/' expr { $$ = $1 / $3; }
21       | '(' expr ')' { $$ = $2; }
22       | NUMBER
23       ;
```


http://ruby-hacking-guide.github.io/yacc.html

LR Parsing Program[解析程序]

- **Input**: input string ω and parse table with ACTION/GOTO
- **Output**: shift-reduce steps of ω's parsing, or error
- **Initial**: s_0 on the stack, ω$ in the input buffer

```
let a be the first symbol of ω$
while (1) { /* repeat forever */
    let s be the state on top of the stack;
    if (ACTION[s,a] = shift t) {
        push a onto the stack; // a is token
        push t onto the stack;  // t is state
        advance to next symbol in ω;
    } else if (ACTION[s,a] = reduce A -> β) {
        pop |β| symbols off the stack;
        let state t now be on top of the stack;
        push GOTO[t, A] onto the stack;
        output the production A-> β; // i.e., build the tree
    } else if (ACTION[s, a] = accept) break; /* parsing is done */
    else call error-recovery routine; // illegal
}
```

Construct Parse Table[构建解析表]

- Construct parsing table: identify the possible states and arrange the transitions among them[状态及转换]
- **LR(0)** parsing
  - Simplest LR parsing, only considers stack to decide shift/reduce
  - Weakest, not used much in practice because of its limitations
- **SLR(1)** parsing / SLR
  - Simple LR, lookahead from FIRST/FOLLOW rules derived from LR(0)
  - Keeps table as small as LR(0)
- **LR(1)** parsing / canonical LR / LR
  - LR parser that considers next token (lookahead of 1)
  - Compared to LR(0), more complex algorithm and much bigger table
- **LALR(1)** parsing / lookahead LR / LALR
  - Lookahead LR(1): fancier lookahead analysis using the same LR(0) automaton as SLR(1)

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

State in LR Parsing[状态]

- How does a shift-reduce parser know when to shift and when to reduce?[何时移进？何时归约？]
  - For the example, how does parser know that *int* on the top of the stack is not a handle, so the action is shift but not to reduce (T ← int)?
- An LR parser makes shift-reduce decisions by maintaining **states** to keep track of where we are in a parse[状态追踪]
  - States represent sets of "**items**"['项目'集合]

Grammar

E → T+E | T

T → int*T | int | (E)

String

int * int + int

Step	Operation
#int * int + int	Shift
int# * int + int	Shift
int * #int + int	Shift
int * **int** # + int	Reduce T → int
int * T # + int	Reduce T → int*T

Item[项目]

- An **item** is a production with a "·" somewhere on the RHS
  - Dot indicates extent of RHS already seen in the parsing process
    - Everything to the left of the dot has been shifted onto the parsing stack
  - The only item for X → ε is X → ·
  - Items are often called "**LR(0) items**" (a.k.a., **configuration**)
- The items for A → XYZ are
  - A → ·XYZ
    - Indicates that we hope to see a string derivable from XYZ next on the input
  - A → X·YZ
    - Indicates that we have just seen on the input a string derivable from X and that we hope next to see a string derivable from YZ
  - A → XY·Z
  - A → XYZ·
    - Indicates that we have seen the body XYZ and that it may be time to reduce XYZ to A

Item (cont.)

- Example:
  - Suppose we are currently in this position

    $A \rightarrow X \cdot YZ$
  - We have just recognized X and expect the upcoming input to contain a sequence derivable from YZ (say, $Y \rightarrow u \mid w$)[已经识别了X，期待YZ推导的串]
    - Y is further derivable from either u or w

      $A \rightarrow X \cdot YZ$

      $Y \rightarrow \cdot u$

      $Y \rightarrow \cdot w$
  - The above three items can be placed into a set, called as **configuration set**[配置集] of the LR parser
- Parsing tables have one **state** corresponding to each set
  - The states can be modeled as a finite automaton where we move from one state to another via transitions marked with a symbol of the CFG

Augmented Grammar[增广文法]

- We want to start with an item with a dot before the start symbol *S* and move to an item with a dot after *S*
  - Represents shifting and reducing an entire sentence of the grammar[完成了整个句子的移进归约]
  - Thus, we need *S* to appear on the right side of a production
  - Only one 'acc' in the table
- Modify the grammar by adding the production[修改文法]

  S' → ·S

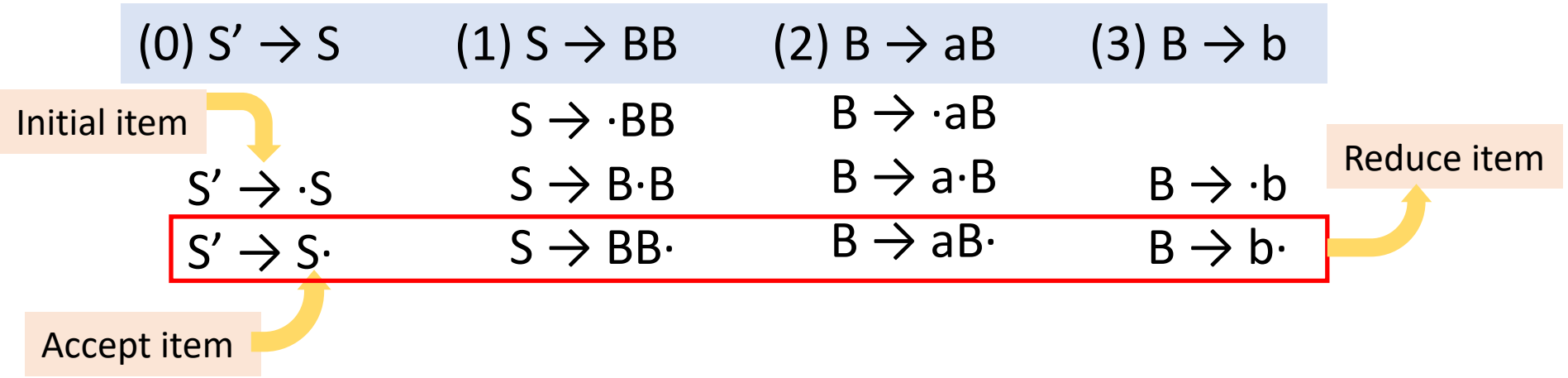
Grammar:

(1) E → E + T
(2) E → T
(3) T → T * F
… …

Augmented grammar:

(0) E' → E
(1) E → E + T
(2) E → T
(3) T → T * F
… …

Example



- **Closure**: the action of adding equivalent items to a set
  - Example: S' → ·S S → ·BB B → ·aB B → ·b
- Intuitively, A → α·Bβ means that we might next see a substring derivable from Bβ (_*sub*) as input. The _*sub* will have a prefix derivable from B by applying one of the B-productions[期待意义等价]
  - Thus, we add items for all the B-productions, i.e., if B → γ is a production, we add B → ·γ in the closure