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(7)

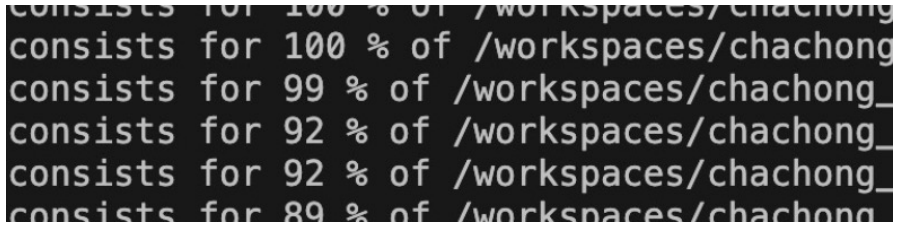
张献伟

xianweiz.github.io

DCS290, 4/9/2024

Grading[考核标准]

- **proj1查重发现多例抄袭，处理措施**
  - 本周四/4.11实验课前，
    - 找TA主动承认，共减50%（比例自行协商）
    - 不主动承认，共减100/150%（比例自行协商）
  - 抄袭被抄袭可限时补交，
    - 参照迟交予以扣分（48h内提交不再额外扣除，超出48h每天减10%）



proj1昨晚已经截止，我们会尽快完成批改！未提交的同学：如果截止前填写了问卷，自动宽限5天，请于4.2/周二23:59:59前提交（宽限期内扣除得分的5%），宽限期外每延一天再扣除得分的10%；没有填写问卷的同学，从今天开始每天扣除10%。proj2/3/4将类似执行，具体有所调整。

- 理论
  - 随机点名
    - 缺席优先
  - 随机提问
    - 后排优先
  - 随机测试
    - 不定时间
- 实验
  - 个人完成
    - **杜绝抄袭**
  - 按时提交
    - **硬性截止**
  - 侧重代码实现
    - 简略报告

Review Questions

- Q1: what are the operations in bottom-up parsing?

  Shift: move a token from buffer into stack
  Reduce: reversely apply a production

- Q2: for reduce, how to operate the stack?

  Pop RHS, push LHS.

- Q3: when to reduce?

  When there is a handle at the stack top.

- Q4: how to recognize a handle?

  Right sentential form - phase - simple phase - leftmost simple phase

- Q5: is it possible for a handle not at stack top?

  Yes. Possible not yet moved into stack.

Top-down vs. Bottom-up

Matched	Stack (unmatched)	Input	Action
	E $	int * int $	E → TE’
	T E’ $	int * int $	T → int T’
int	int T’ E’ $	int * int $	match
int	T’ E’ $	* int $	T’ → *T
int	* T E’ $	* int $	match
int *	T E’ $	int $	T → int T’
int *	int T’ E’ $	int $	match
int * int	T’ E’ $	$	T’ → ε
int * int	E’ $	$	E’ → ε
int * int	$	$	Halt-accept

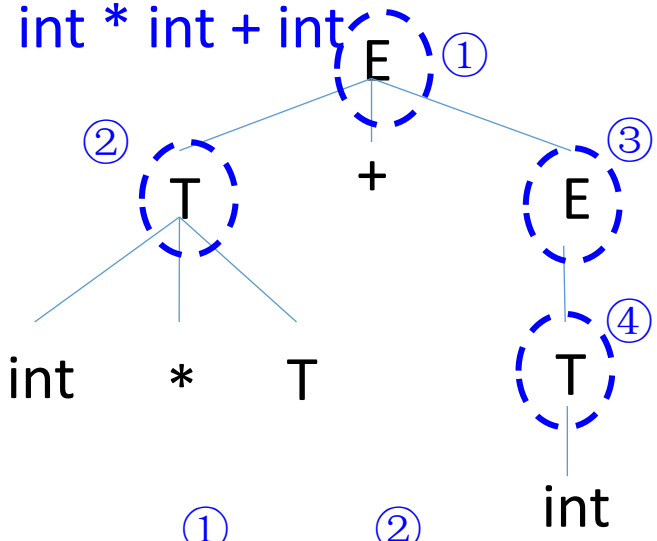
Top-down

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
T # + int	Shift
T + # int	Shift
T + int #	Reduce T → int
T + T #	Reduce E → T
T + E #	Reduce E → T+E
E #	

Bottom-up

Handle: Example

- Grammar
  - E → T+E|T
  - T → int*T | int | (E)
- String
  - int * int + int



Phrase: int * T + int, int * T, int
Simple phrase: int * T, int
Handle: int * T

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
T # + int	Shift
T + # int	Shift
T + int #	Reduce T → int
T + T #	Reduce E → T
T + E #	Reduce E → T+E
E #	

Viable Prefix[活前缀]

- In shift-reduce parsing, the stack contents are always a **viable prefix**[活前缀/可动前缀]
  - A prefix of some right-sentential form that ends no further right than the end of the handle of that right-sentential form
    - A viable prefix has a handle at its rightmost end
  - Stack content is always a viable prefix, guaranteeing the shift / reduce is on the right track[活前缀说明移进归约是正确的 → 形成句柄]
- 定义：一个可行前缀是一个最右句型的前缀，并且它没有越过该最右句型的最右句柄的右端
  - 举例：S => bBa => bbAa，这里句柄是 bA，因此可行前缀包括 bA 的所有前缀（包括 b, bb, bbA），但不能是 bbAa（因为越过了句柄）

... => T + int => int * T + int
Handle: int * T
Viable prefix: int, int *, int * T

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

Ambiguous Grammars[二义文法]

- Conflicts arise with ambiguous grammars
  - Bottom up parsing predicts action w/ lookahead (just like LL)
  - If there are multiple correct actions, parse table will have conflicts
- Example:
  - Consider the ambiguous grammar E → E * E | E + E | (E) | int

Sentential form	Actions
int * int + int	shift
…	…
E * E # + int	**reduce E → E * E**
E # + int	shift
E + # int	shift
E + int #	reduce E → int
E + E #	reduce E → E + E
E #	**First * then +**

Sentential form	Actions
int * int + int	shift
…	…
E * E # + int	**shift**
E * E + # int	shift
E * E + int #	reduce E → int
E * E + E #	reduce E → E + E
E * E #	reduce E → E * E
E #	**First + then ***

Ambiguous Grammars (cont.)

- In the red step shown, can either shift + or reduce by E → E * E
  - Both okay since precedence of + and * not specified in grammar
  - Same problem with associativity of + and *
- As usual, remove conflicts due to ambiguity ...
  - 1. rewrite grammar/parser to encode precedence and associativity[指定优先级和结合性]
    - Rewriting grammar results in more convoluted grammars
    - Parser tools have other means to encode precedence and association
  - 2. get rid of remaining ambiguity (e.g. if-then-else)
    - No choice but to modify grammar
- Is ambiguity the only source of conflicts?
  - Limitations in lookahead-based prediction can cause conflicts
  - But these cases are very rare

Sentential form	Actions
int * int + int	shift
…	…
E * E # + int	reduce E → E * E
E # + int	shift
E + # int	shift
E + int #	reduce E → int
E + E #	reduce E → E + E
E #	

Sentential form	Actions
int * int + int	shift
…	…
E * E # + int	shift
E * E + # int	shift
E * E + int #	reduce E → int
E * E + E #	reduce E → E + E
E * E #	reduce E → E * E
E #	

Properties of Bottom-up Parsing[属性]

- Handles **always** appear at the **top of the stack**[句柄总在栈顶]
  - Never in middle of stack
  - Justifies use of stack in shift – reduce parsing
- Results in an easily generalized **shift – reduce** strategy
  - If there is no handle at the top of the stack, shift[无句柄，移入]
  - If there is a handle, reduce to the non-terminal[有句柄，归约]
  - Easy to automate the synthesis of the parser using a table
- Can have conflicts[冲突可能发生]
  - If it is legal to either shift or reduce then there is a shift-reduce conflict[移入-归约冲突]
  - If there are two legal reductions, then there is a reduce-reduce conflict[归约-归约冲突]
  - Most often occur because of ambiguous grammars
    - In rare cases, because of non-ambiguous grammars not amenable to parser

Types of Bottom-Up Parsers[类型]

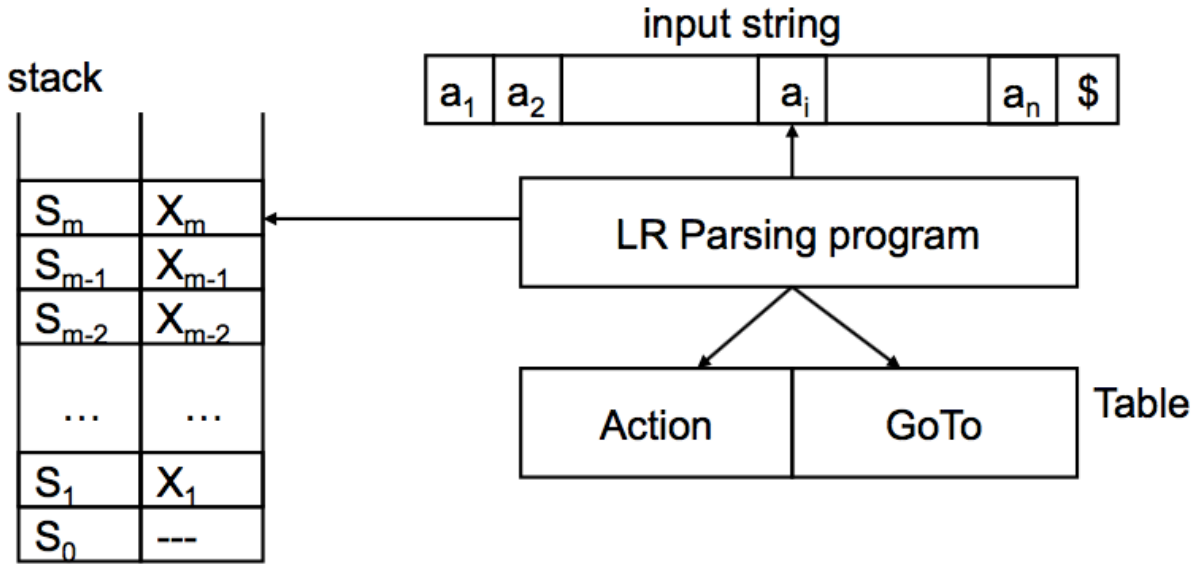
- Types of bottom up parsers
  - Simple precedence parsers[简单优先解析器]
  - Operator precedence parsers[运算符优先解析器]
  - Recursive ascent parsers[递归提升解析器]
  - LR family parsers[LR类解析器]
  - …

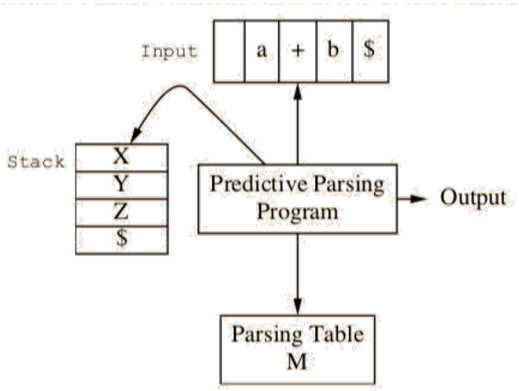
- In this course, we will only discuss **LR family parsers**
  - Efficient, table-driven shift-reduce parsers
  - Most automated tools for bottom-up parsing generate LR family
  - Categories: LR(0), LR(1), SLR, LALR, …

LR(k) Parser

- **LR(k)**: member of LR family of parsers
  - L: scan input from left to right
  - R: construct a rightmost derivation in reverse
  - k: number of input symbols of lookahead to make decisions
    - k = 0 or 1 are of particular interests, is assumed to be 1 when omitted
- Comparison with LL(k) parser[对比]
  - Efficient as LL(k)
    - Linear in time and space to length of input (same as LL(k))
  - Convenient as LL(k)
    - Can generate automatically from grammar – YACC, Bison
  - More complex than LL(k)
    - Harder to debug parser when grammar causes conflicting predictions
  - More powerful than LL(k)
    - Handles more grammars: no left recursion removal, left factoring unneeded
    - Handles more (and most practical) languages: LL(1) ⊂ LR(1)

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

Parse Table[分析表]

- LR parsers use two tables: **action table** and **goto table**
  - The two tables are usually combined
  - Action table specifies entries for terminals
  - Goto table specifies entries for non-terminals

- Action table[动作表]
  - Action[*s*, *a*] tells the parser what to do when the state on top of the stack is *s* and terminal *a* is the next input token
  - Possible actions: **shift**, **reduce**, **accept**, **error**
- Goto table[跳转表]
  - Goto[*s*, *X*] indicates the new state to place on top of the stack after a reduction of the non-terminal *X* while state *s* is on top of the stack

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

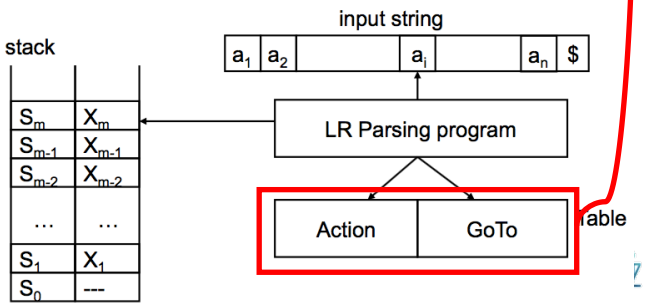
(2) B → aB

(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		

state → 0

b a b

symbol → $

b a b $

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:

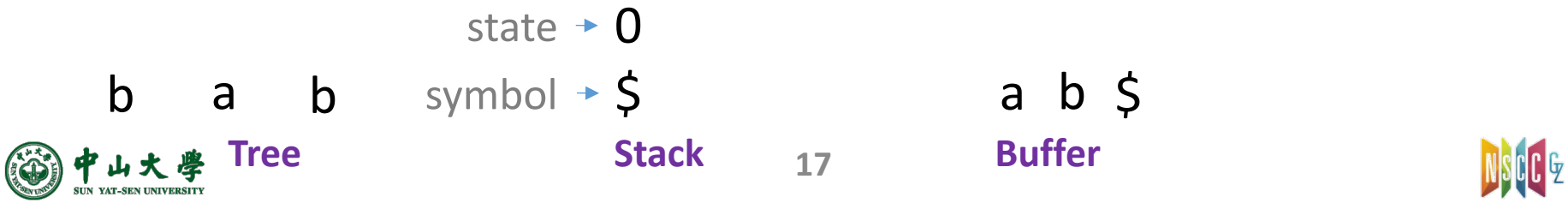
(1) S → BB

(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		



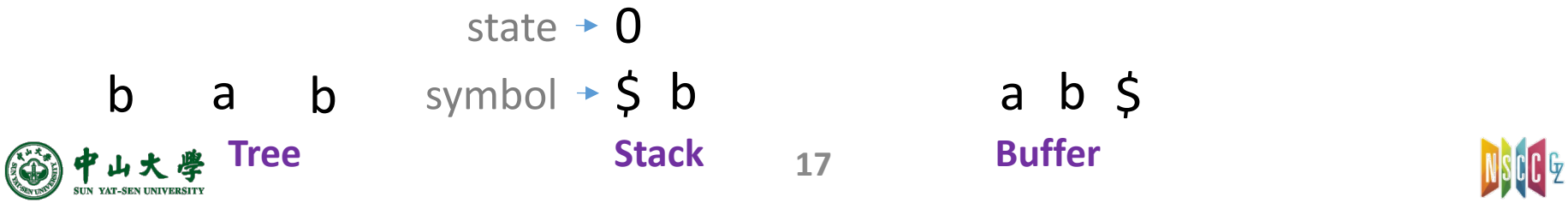
Example: Parse Table (cont.)

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		



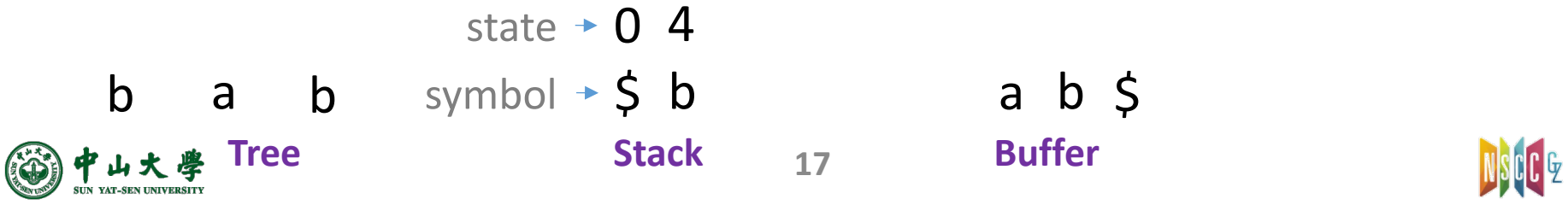
Example: Parse Table (cont.)

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		



Example: Parse Table (cont.)

Grammar:

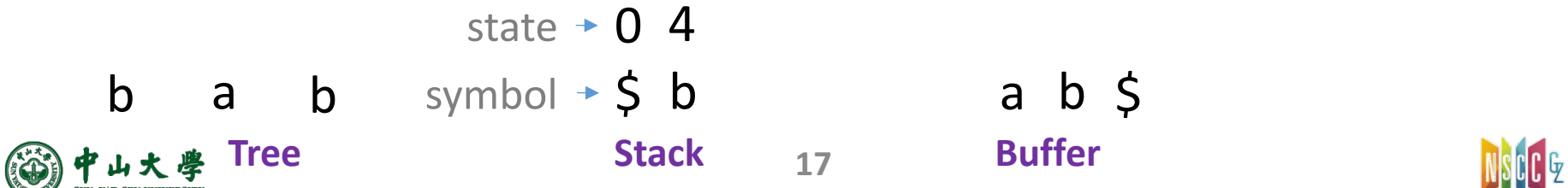
(1) S → BB

(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:

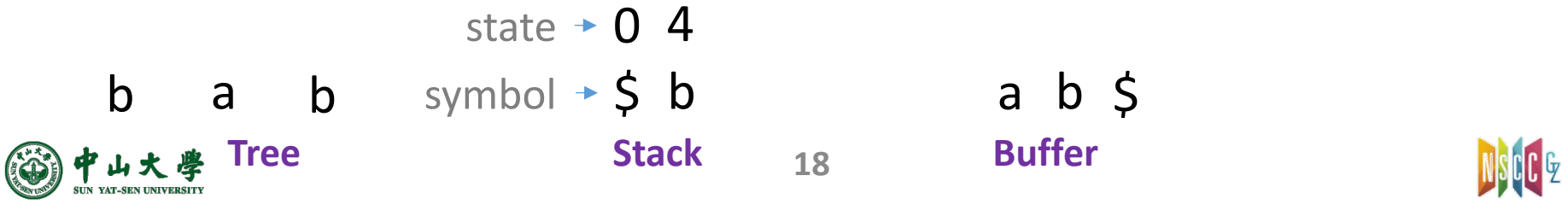
(1) S → BB

(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		

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		

state → 0 4

b a b symbol → $ 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		

b a b

Tree

state → 0

symbol → $

Stack

a b $

Buffer

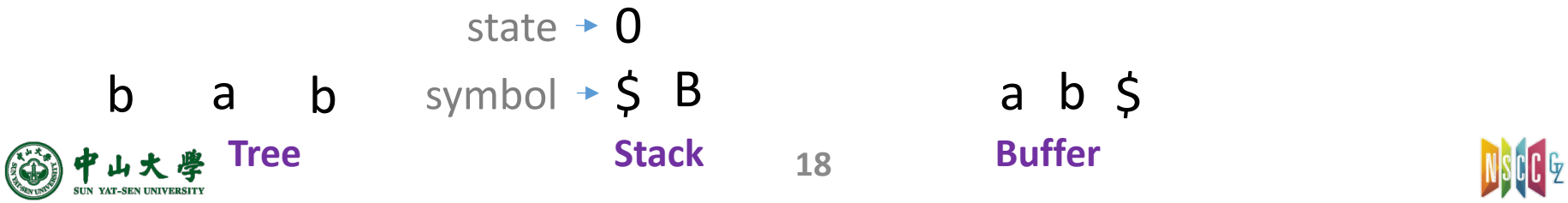
Example: Parse Table (cont.)

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:

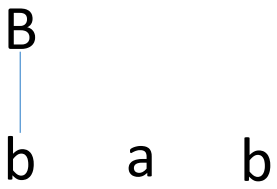
(1) S → BB

(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		



Tree

Stack

Buffer

Example: Parse Table (cont.)

Grammar:

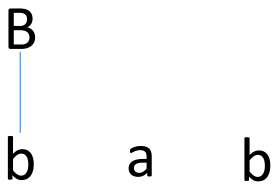
(1) S → BB

(2) B → aB

(3) B → b

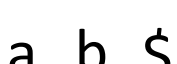
String: bab

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

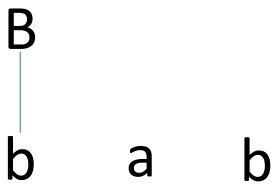
Example: Parse Table (cont.)

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

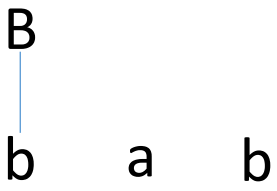
Example: Parse Table (cont.)

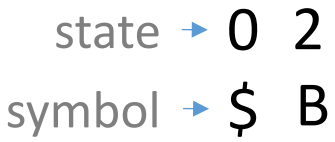
Grammar:

(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		





Example: Parse Table (cont.)

Grammar:

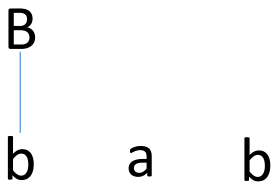
(1) S → BB

(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:

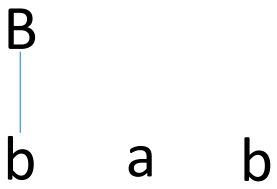
(1) S → BB

(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

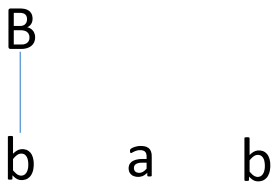
Example: Parse Table (cont.)

Grammar:

(1) S → BB
(2) B → aB
(3) B → b

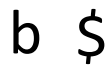
String: bab

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:

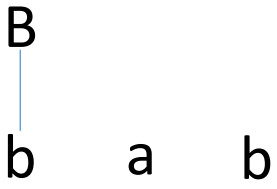
(1) S → BB

(2) B → aB

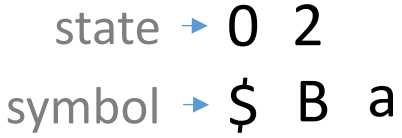
(3) B → b

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:

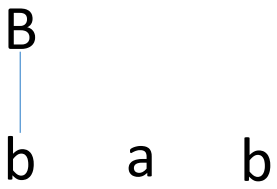
(1) S → BB

(2) B → aB

(3) B → b

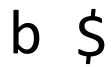
String: bab

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:

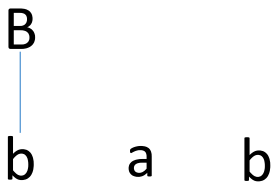
(1) S → BB

(2) B → aB

(3) B → b

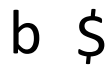
String: bab

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:

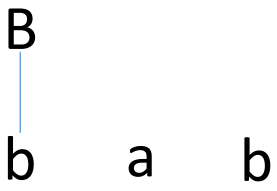
(1) S → BB

(2) B → aB

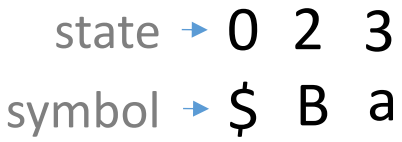
(3) B → b

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:

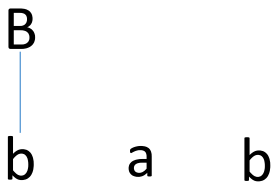
(1) S → BB

(2) B → aB

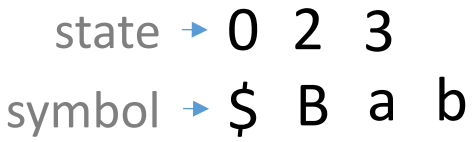
(3) B → b

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:

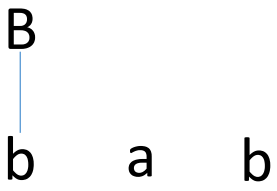
(1) S → BB

(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:

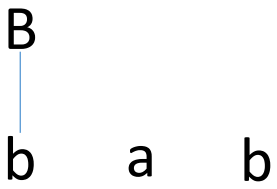
(1) S → BB

(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:

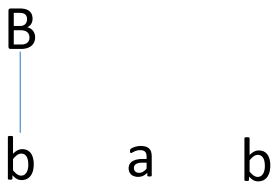
(1) S → BB

(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:

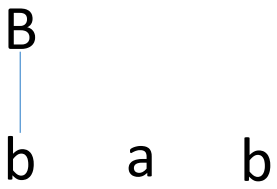
(1) S → BB

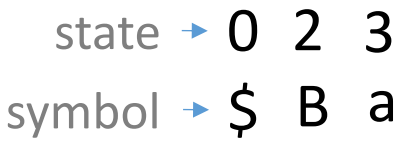
(2) B → aB

(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:

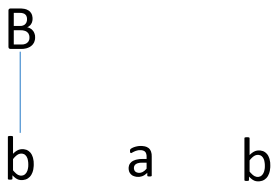
(1) S → BB

(2) B → aB

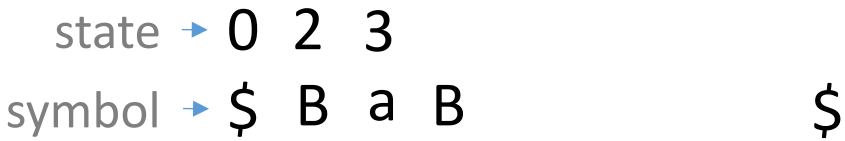
(3) B → b

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:

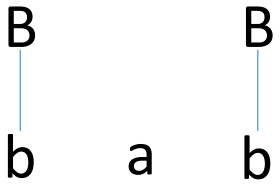
(1) S → BB

(2) B → aB

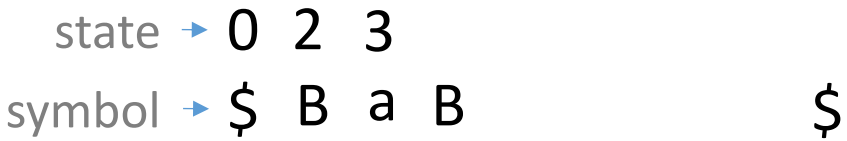
(3) B → b

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:

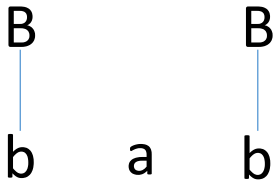
(1) S → BB

(2) B → aB

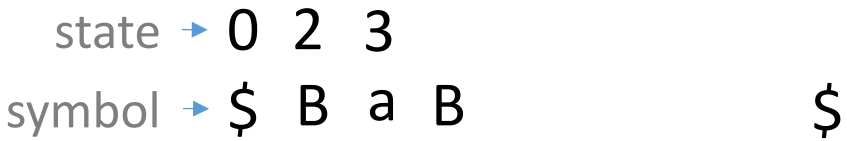
(3) B → b

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:

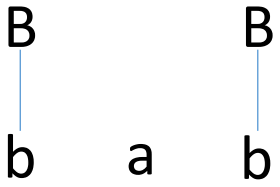
(1) S → BB

(2) B → aB

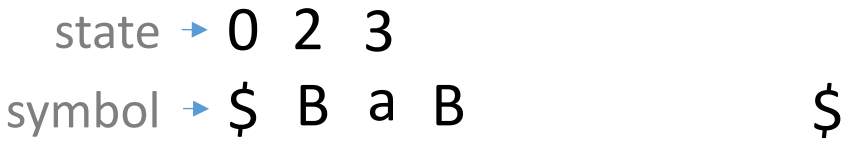
(3) B → b

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:

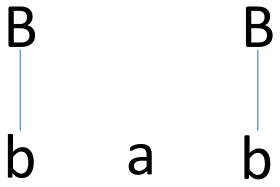
(1) S → BB

(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:

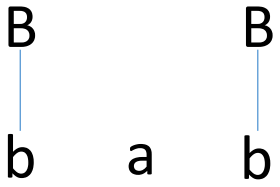
(1) S → BB

(2) B → aB

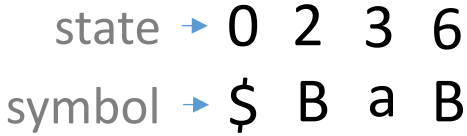
(3) B → b

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:

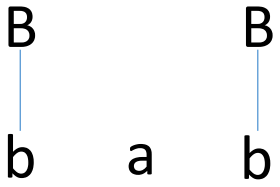
(1) S → BB

(2) B → aB

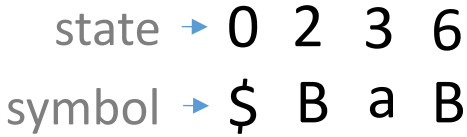
(3) B → b

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:

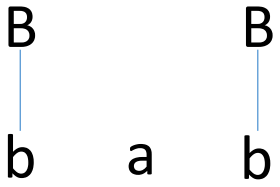
(1) S → BB

(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:

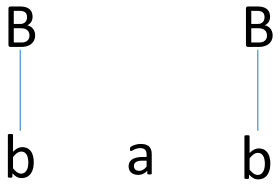
(1) S → BB

(2) B → aB

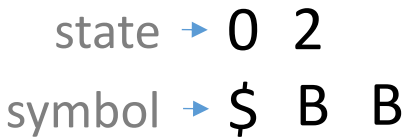
(3) B → b

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

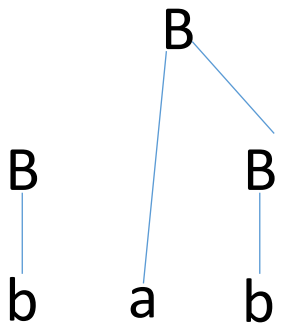
Example: Parse Table (cont.)

Grammar:

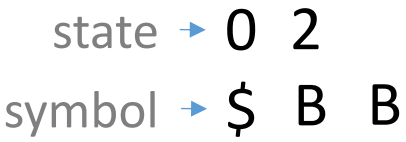
(1) S → BB
(2) B → aB
(3) B → b

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:

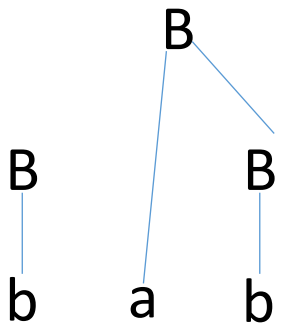
(1) S → BB

(2) B → aB

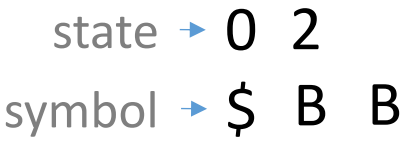
(3) B → b

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:

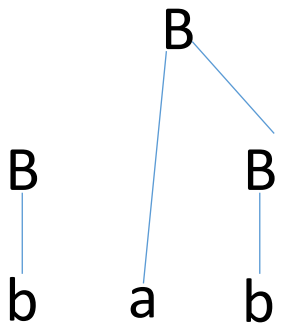
(1) S → BB

(2) B → aB

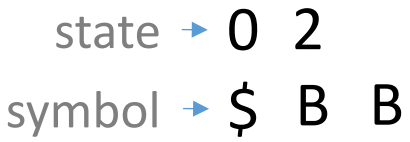
(3) B → b

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

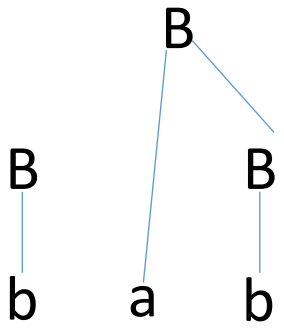
Example: Parse Table (cont.)

Grammar:

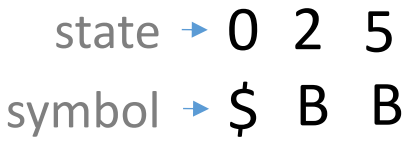
(1) S → BB
(2) B → aB
(3) B → b

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:

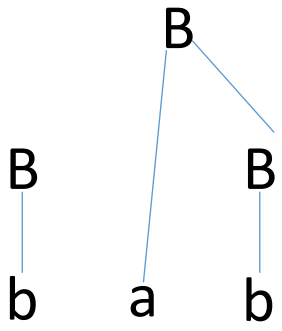
(1) S → BB

(2) B → aB

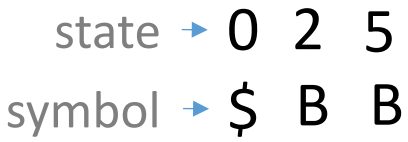
(3) B → b

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

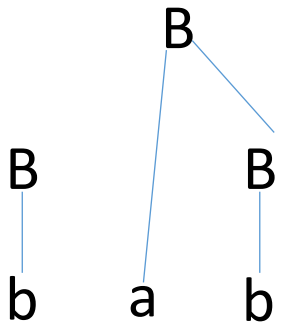
Example: Parse Table (cont.)

Grammar:

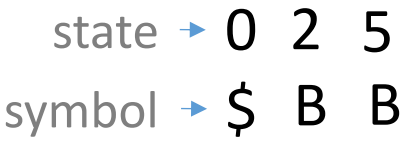
(1) S → BB
(2) B → aB
(3) B → b

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

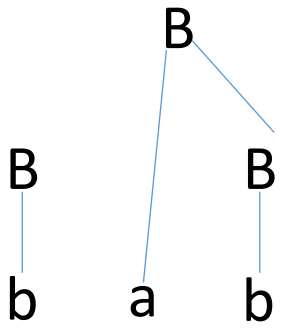
Example: Parse Table (cont.)

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

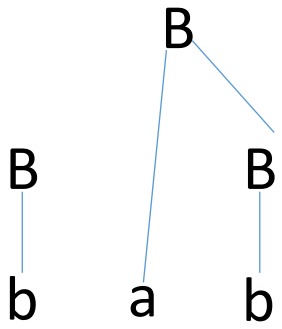
Example: Parse Table (cont.)

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 → $ S

Stack

$

Buffer

Example: Parse Table (cont.)

Grammar:

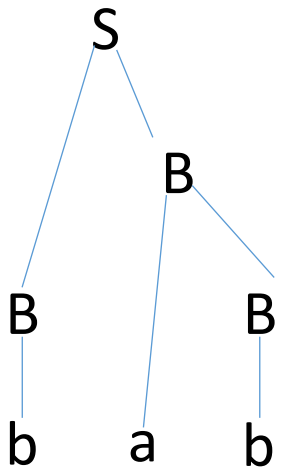
(1) S → BB

(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:

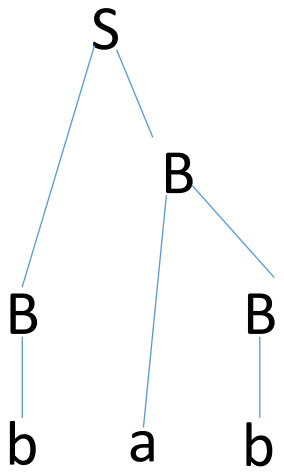
(1) S → BB

(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		



state → 0

symbol → $ S

$

Tree Stack Buffer

Example: Parse Table (cont.)

Grammar:

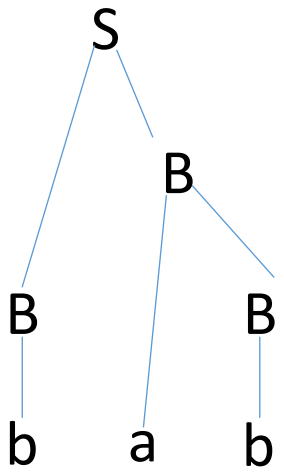
(1) S → BB

(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		



state → 0

symbol → $ S

$

Example: Parse Table (cont.)

Grammar:

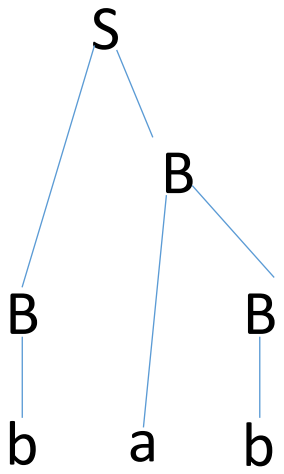
(1) S → BB

(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 1

symbol → $ S

Stack

$

Buffer

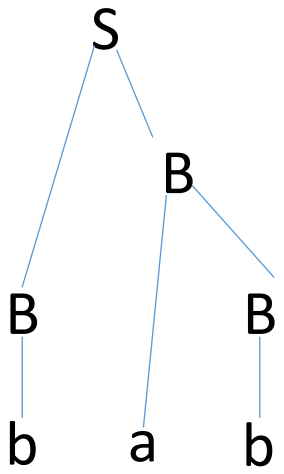
Example: Parse Table (cont.)

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

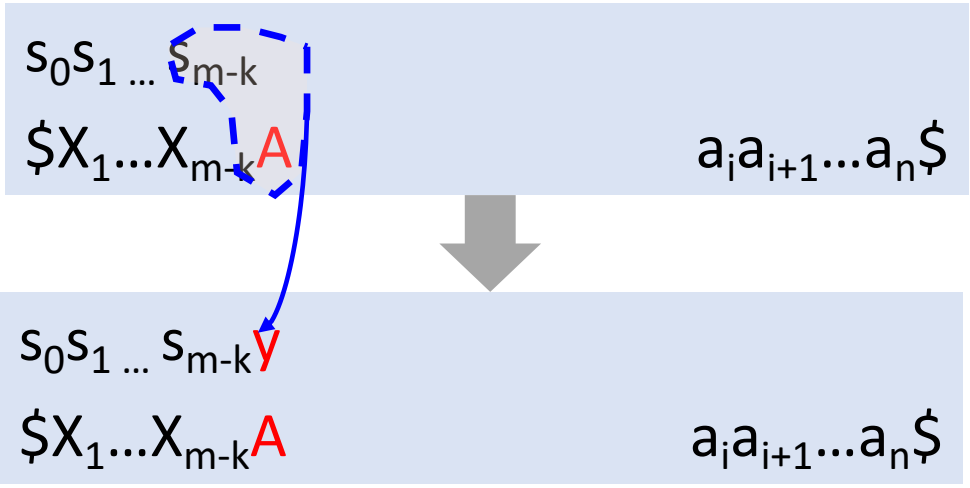
symbol → $ S

$

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 → $X_{m-(k-1)}...X_m$), then do **reduce**[归约]
  - Pop *k* symbols from stack
  - Push *A* on stack
  - No change on input
  - GOTO[S_{m-k}, A] = y, then
    - 需寻找下一状态

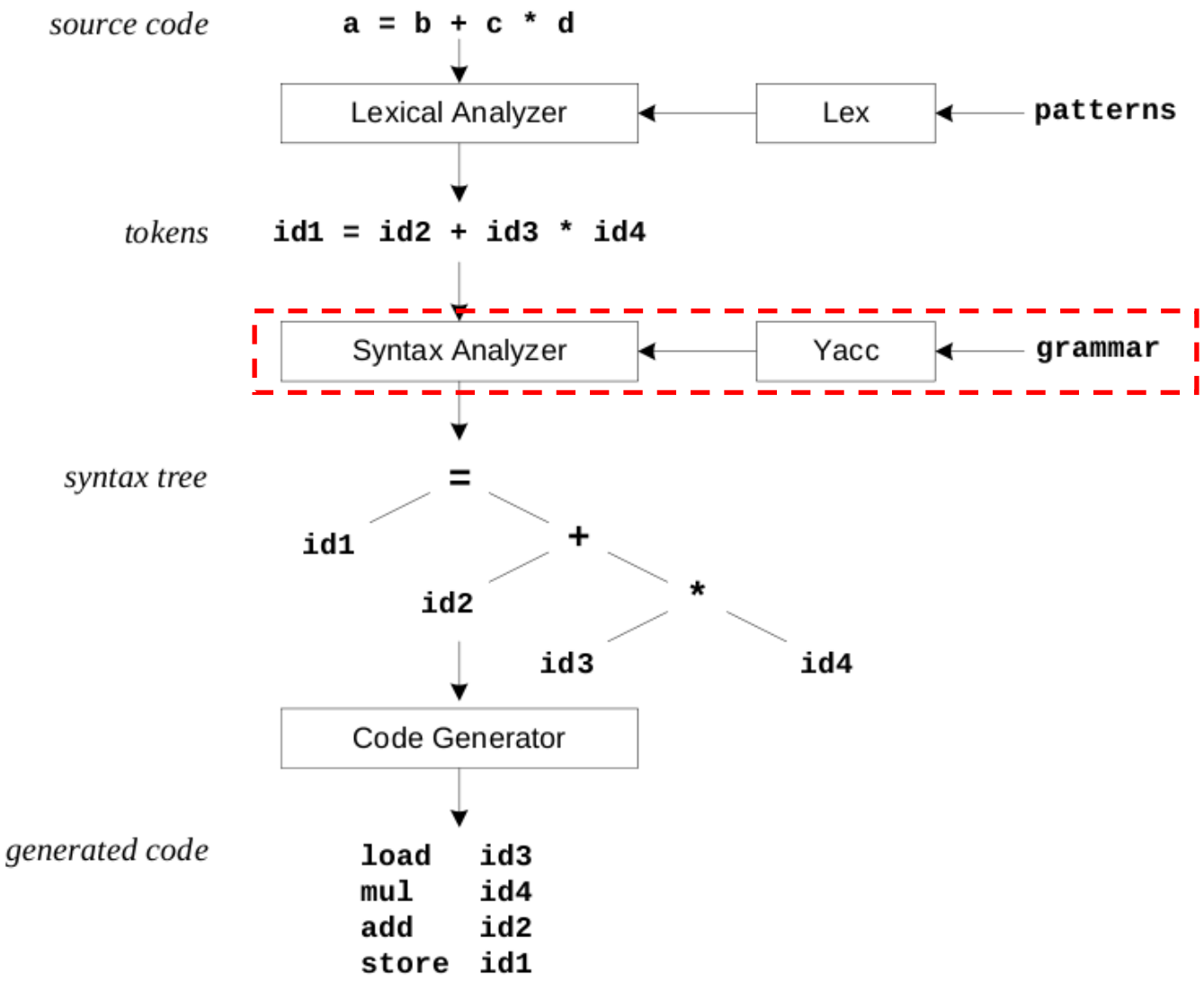


Parser Actions (cont.)

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

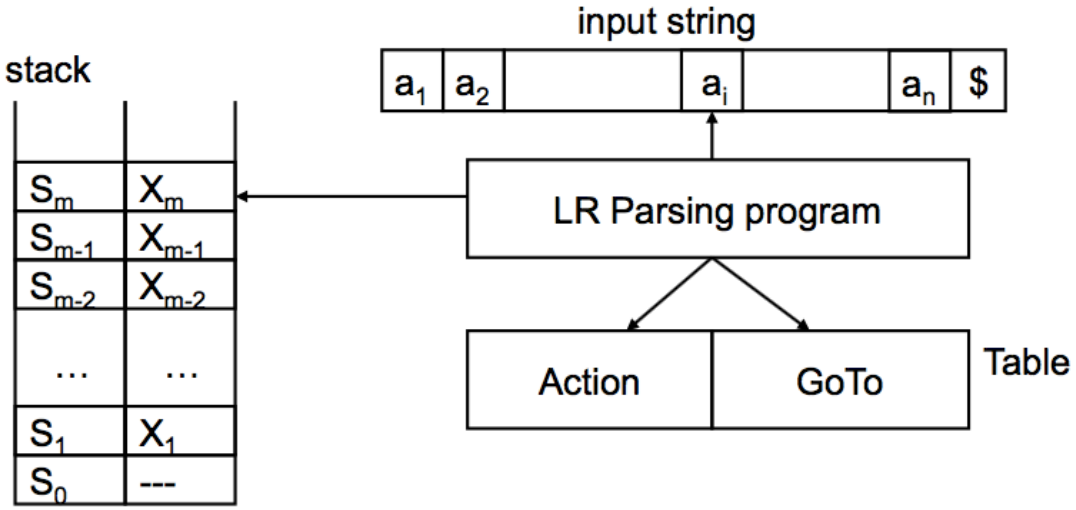
- If ACTION[s_m, a_i] = acc, then parsing is **complete**[接收]
- If ACTION[s_m, a_i] = <empty>, then report **error** and stop[报错]

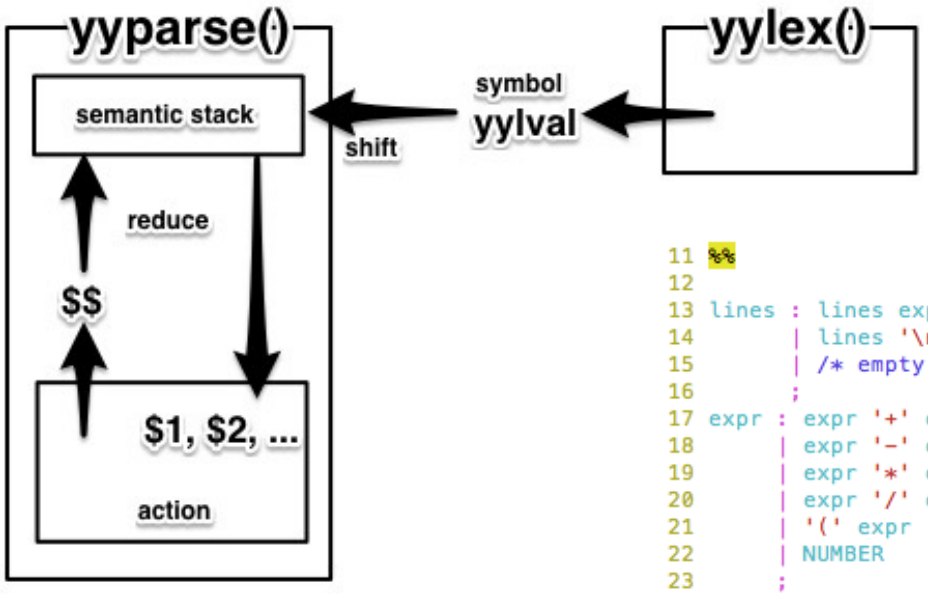
Parser in Practice



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]

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		

- **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)