Compilation Principle
编 译 原 理

第10讲：语法分析(7)

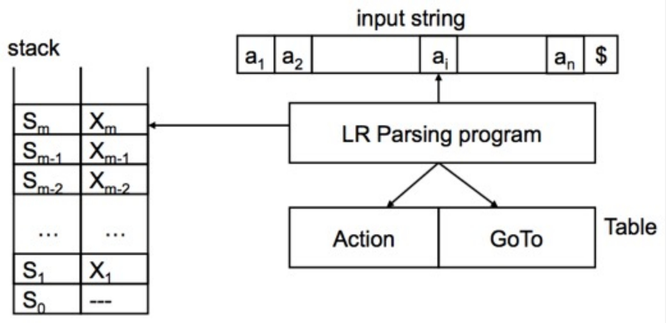
张献伟

xianweiz.github.io

DCS290, 3/24/2022

Review Questions

- What does LR(k) mean?
  L: scan the input from **l**eft to right
  R: construct a **r**ightmost derivation in reverse
  k: use **k** input symbols of lookahead



- What are the parts of a LR parser?
  Input buffer, stack, parse table, driver
- What are held in the stack of a LR parser?
  A sequence of states, and each has an associated grammar symbol
- The LR parsing table is split into two, what are they?
  Action table for terminals, Goto table for non-terminals
- What are the possible actions in Action table?
  Shift, reduce, accept, 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		

state → 0

b a b   symbol → $ b a b $

SUN YAT-SEN UNIVERSITY

Parser Actions[解析动作]

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

- 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

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

Parser Actions (cont.)

Initial

s_0	
\$	$a_1a_2…a_n\$$

General

$s_0s_1 … s_m$	
$\$X_1…X_m$	$a_ia_{i+1}…a_n\$$

- If ACTION[s_m, a_i] = rx, (i.e., the x^{th} production: $A \rightarrow X_{m-(k-1)}…X_m$), then do **reduce**[规约]
  - 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

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

↓

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

Parser Actions (cont.)

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

- If ACTION[s_m, a_i] = rx, (i.e., the x^{th} production: $A \rightarrow X_{m-(k-1)} \dots X_m$), then do **reduce**[规约]
  - 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

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

Parser Actions (cont.)

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

- 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]

LR Parsing Program[解析程序]

- **Input**: input string ω and parse table with ACTION/GOTO
- **Output**: shift-reduce steps of ω's bottom-up 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;
         push t onto the stack;
        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;
    } else if (ACTION[s,a] = accept) break; /* parsing is done */
    else call error-recovery routine;
}
```

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)

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(1)** parsing / canonical LR / LR
  - LR parser that considers next token (lookahead of 1)
  - Compared to LR(0), more complex alg 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 in LR Parsing[状态]

- How does a shift-reduce parser know when to shift and when to reduce?[何时移进？何时规约？]
  - 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**"

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 $X \rightarrow \varepsilon$ is $X \rightarrow \cdot$
  - Items are often called "**LR(0) items**" (a.k.a., **configuration**)
- The items for $A \rightarrow XYZ$ are
  - $A \rightarrow \cdot XYZ$
    - Indicates that we hope to see a string derivable from XYZ next on the input
  - $A \rightarrow X \cdot YZ$
    - 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
  - $A \rightarrow XY \cdot Z$
  - $A \rightarrow XYZ \cdot$
    - 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, $Y \rightarrow u|w$)[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 <u>finite automaton</u> 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 <u>dot before the start symbol *S*</u> and move to an item with <u>a dot after *S*</u>
  - Represents shifting and reducing an entire sentence of the grammar[完成了整个句子的移进规约]
  - 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

(0) $S' \to S$ (1) $S \to BB$ (2) $B \to aB$ (3) $B \to b$

Example

(0) $S' \rightarrow S$ (1) $S \rightarrow BB$ (2) $B \rightarrow aB$ (3) $B \rightarrow b$

$S' \rightarrow \cdot S$

$S' \rightarrow S\cdot$

Example

(0) $S' \rightarrow S$	(1) $S \rightarrow BB$	(2) $B \rightarrow aB$	(3) $B \rightarrow b$

$S \rightarrow \cdot BB$

$S \rightarrow B \cdot B$

$S \rightarrow BB \cdot$

$S' \rightarrow \cdot S$

$S' \rightarrow S \cdot$

Example

(0) S' → S	(1) S → BB	(2) B → aB	(3) B → b
	S → ·BB	B → ·aB	
S' → ·S	S → B·B	B → a·B	
S' → S·	S → BB·	B → aB·	

SUN YAT-SEN UNIVERSITY 中山大學

Example

(0) $S' \rightarrow S$	(1) $S \rightarrow BB$	(2) $B \rightarrow aB$	(3) $B \rightarrow b$
	$S \rightarrow \cdot BB$	$B \rightarrow \cdot aB$	
$S' \rightarrow \cdot S$	$S \rightarrow B \cdot B$	$B \rightarrow a \cdot B$	$B \rightarrow \cdot b$
$S' \rightarrow S \cdot$	$S \rightarrow BB \cdot$	$B \rightarrow aB \cdot$	$B \rightarrow b \cdot$

Example

(0) S' → S	(1) S → BB	(2) B → aB	(3) B → b

Initial item

S' → ·S	S → ·BB	B → ·aB	B → ·b
S' → S·	S → B·B	B → a·B	B → b·
	S → BB·	B → aB·	

Example

(0) S' → S	(1) S → BB	(2) B → aB	(3) B → b
	S → ·BB	B → ·aB	
S' → ·S	S → B·B	B → a·B	B → ·b
S' → S·	S → BB·	B → aB·	B → b·

Initial item

Accept item

Example

(0) S’ → S	(1) S → BB	(2) B → aB	(3) B → b
	S → ·BB	B → ·aB	
S’ → ·S	S → B·B	B → a·B	B → ·b
S’ → S·	S → BB·	B → aB·	B → b·

Initial item

Reduce item

Accept item

Example

(0) $S' \to S$	(1) $S \to BB$	(2) $B \to aB$	(3) $B \to b$
	$S \to \cdot BB$	$B \to \cdot aB$	
$S' \to \cdot S$	$S \to B \cdot B$	$B \to a \cdot B$	$B \to \cdot b$
$S' \to S \cdot$	$S \to BB \cdot$	$B \to aB \cdot$	$B \to b \cdot$

Initial item

Reduce item

Accept item

- **Closure**: the action of adding equivalent items to a set
  - Example: $S' \to \cdot S$ $\quad S \to \cdot BB$ $\quad B \to \cdot aB$ $\quad B \to \cdot b$

Example

(0) S' → S	(1) S → BB	(2) B → aB	(3) B → b
	S → ·BB	B → ·aB	
S' → ·S	S → B·B	B → a·B	B → ·b
S' → S·	S → BB·	B → aB·	B → b·

Initial item

Reduce item

Accept item

- **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 <u>substring derivable from Bβ</u> (_*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

Example

Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b

Example

Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b

I_0:

S’ → ·S

Example

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

I_0:

$S' \rightarrow \cdot S$

$S \rightarrow \cdot BB$

Example

Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b

I_0:

S' → ·S

S → ·BB

B → ·aB

B → ·b

Example

Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b

I_0:

S' → ·S

S → ·BB

B → ·aB

B → ·b

$I_0 \xrightarrow{S} I_1$

I_1:

S' → S·

Example

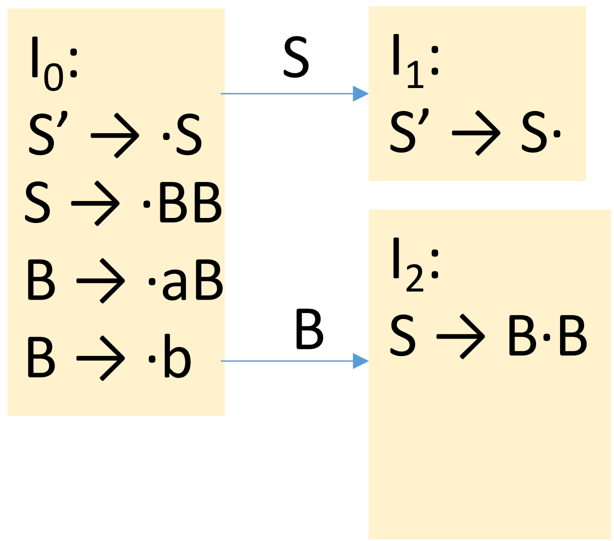
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



Example

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

I_0:

$S' \rightarrow \cdot S$

$S \rightarrow \cdot BB$

$B \rightarrow \cdot aB$

$B \rightarrow \cdot b$

$I_0 \xrightarrow{S} I_1$

I_1:

$S' \rightarrow S\cdot$

$I_0 \xrightarrow{B} I_2$

I_2:

$S \rightarrow B\cdot B$

$B \rightarrow \cdot aB$

$B \rightarrow \cdot b$

Example

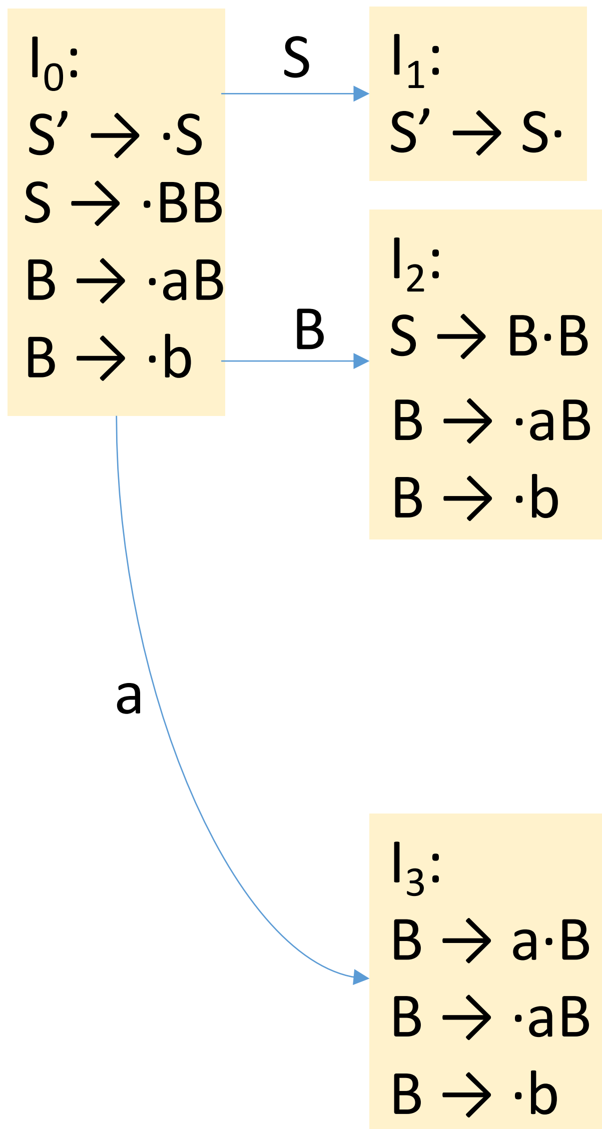
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



Example

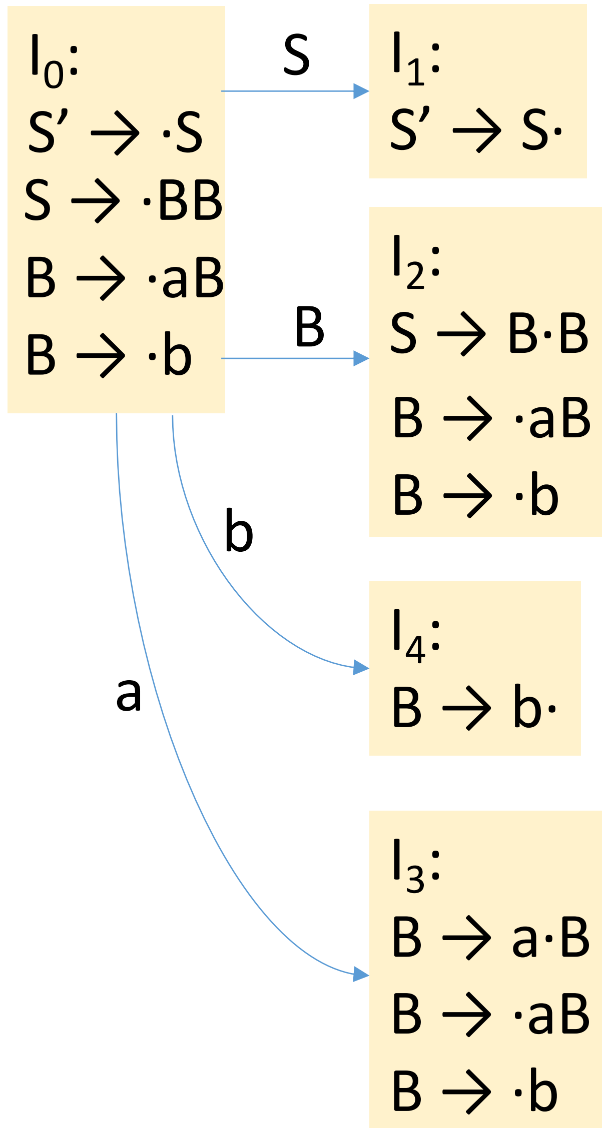
Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$



Example

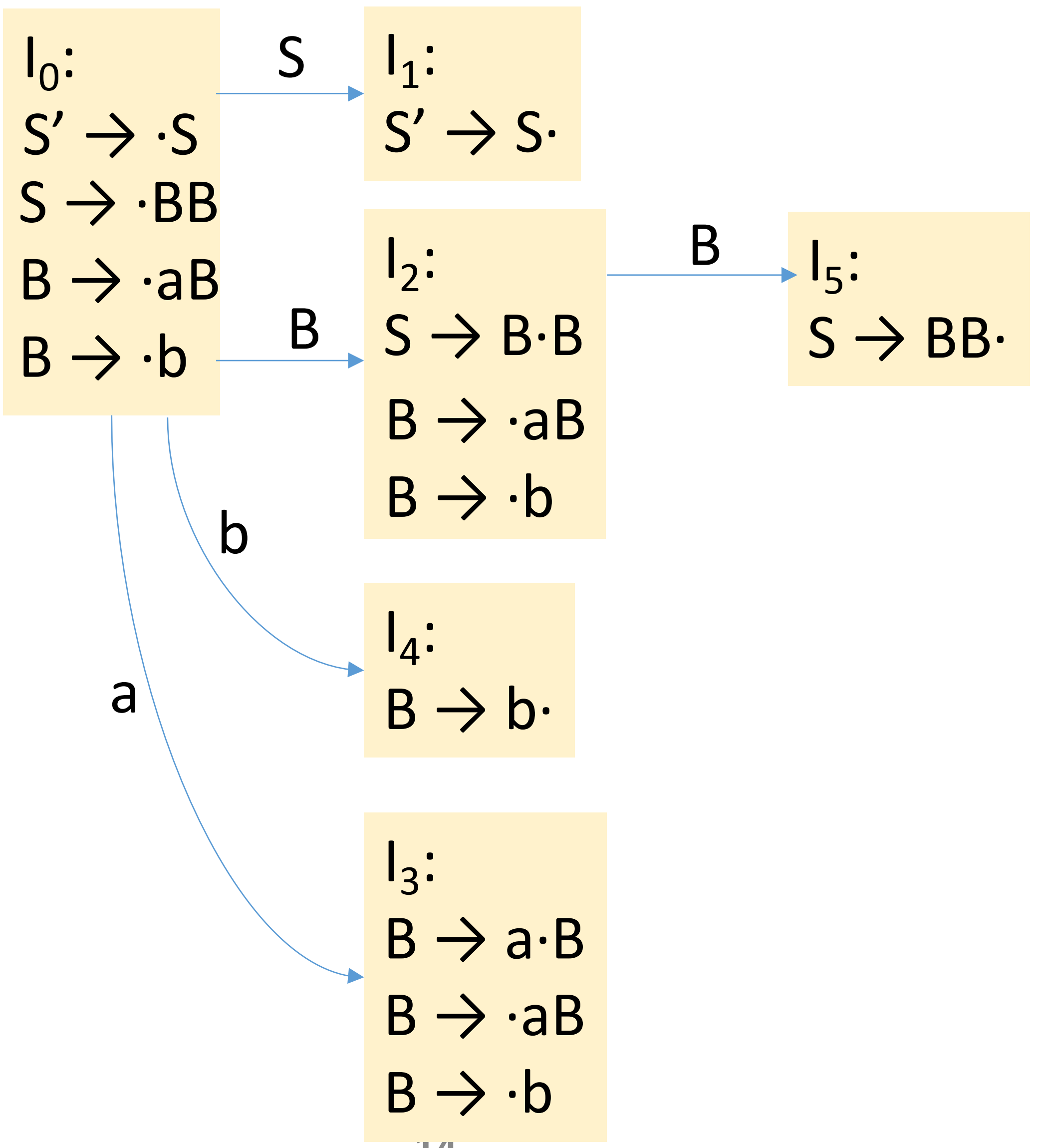
Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$



Example

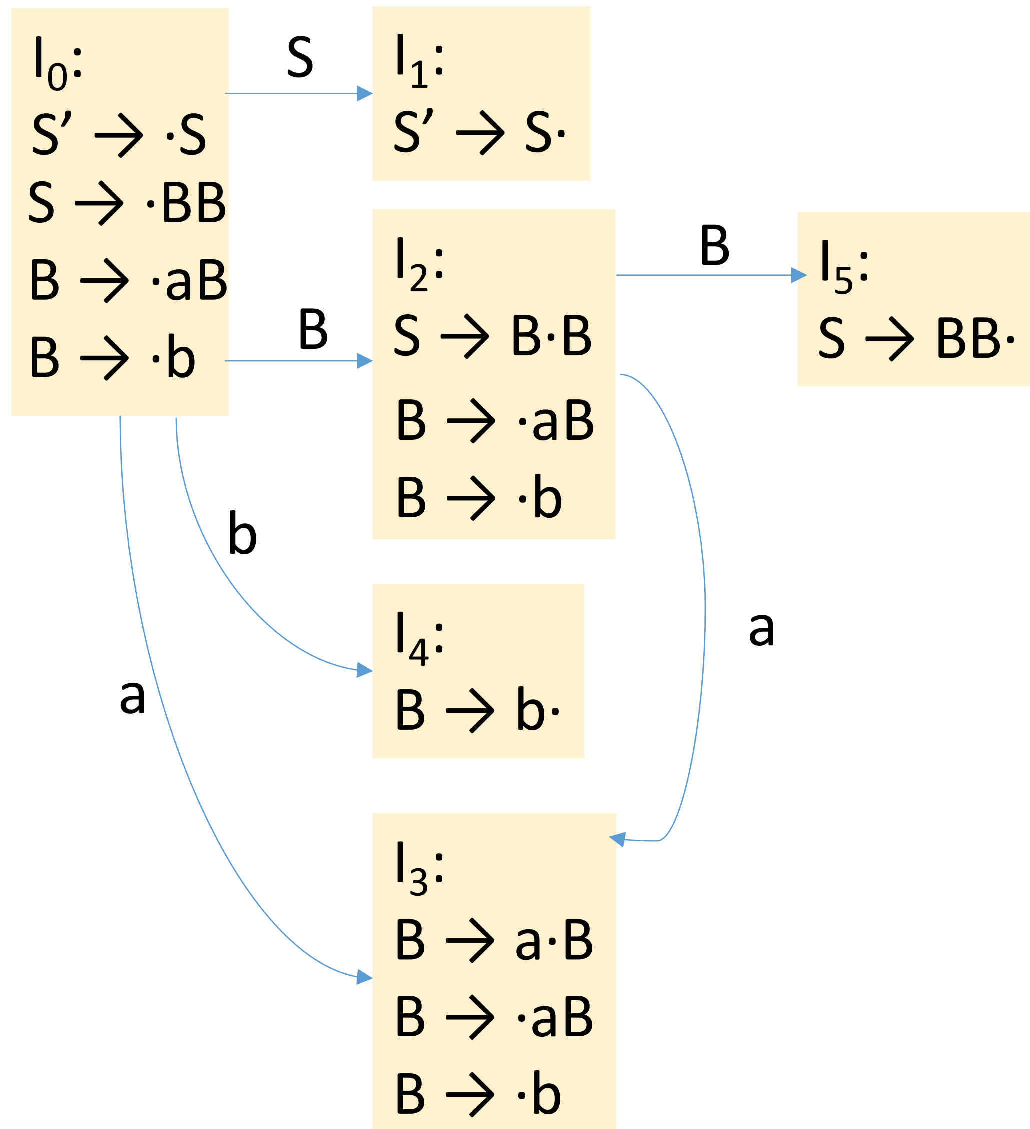
Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$



Example

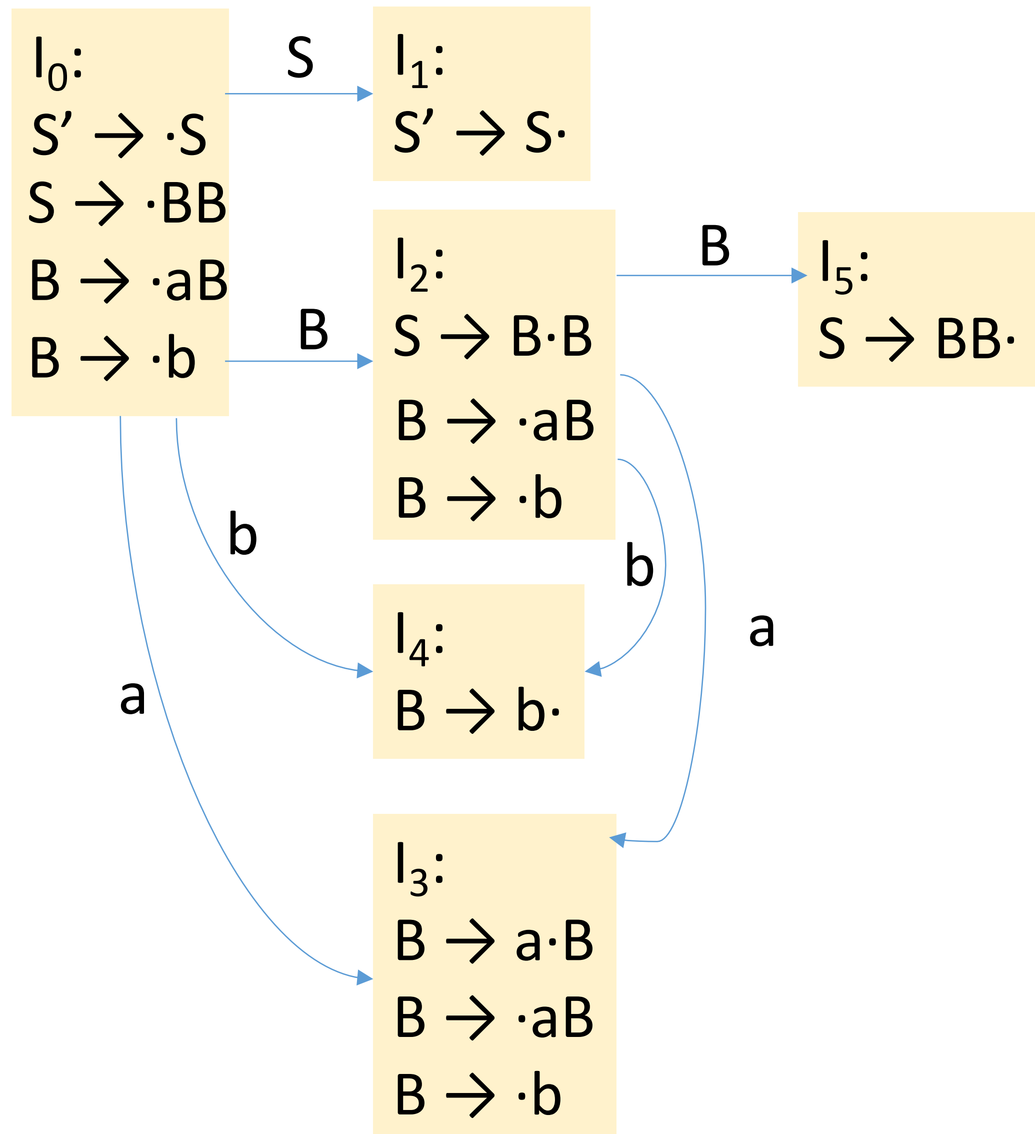
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



Example

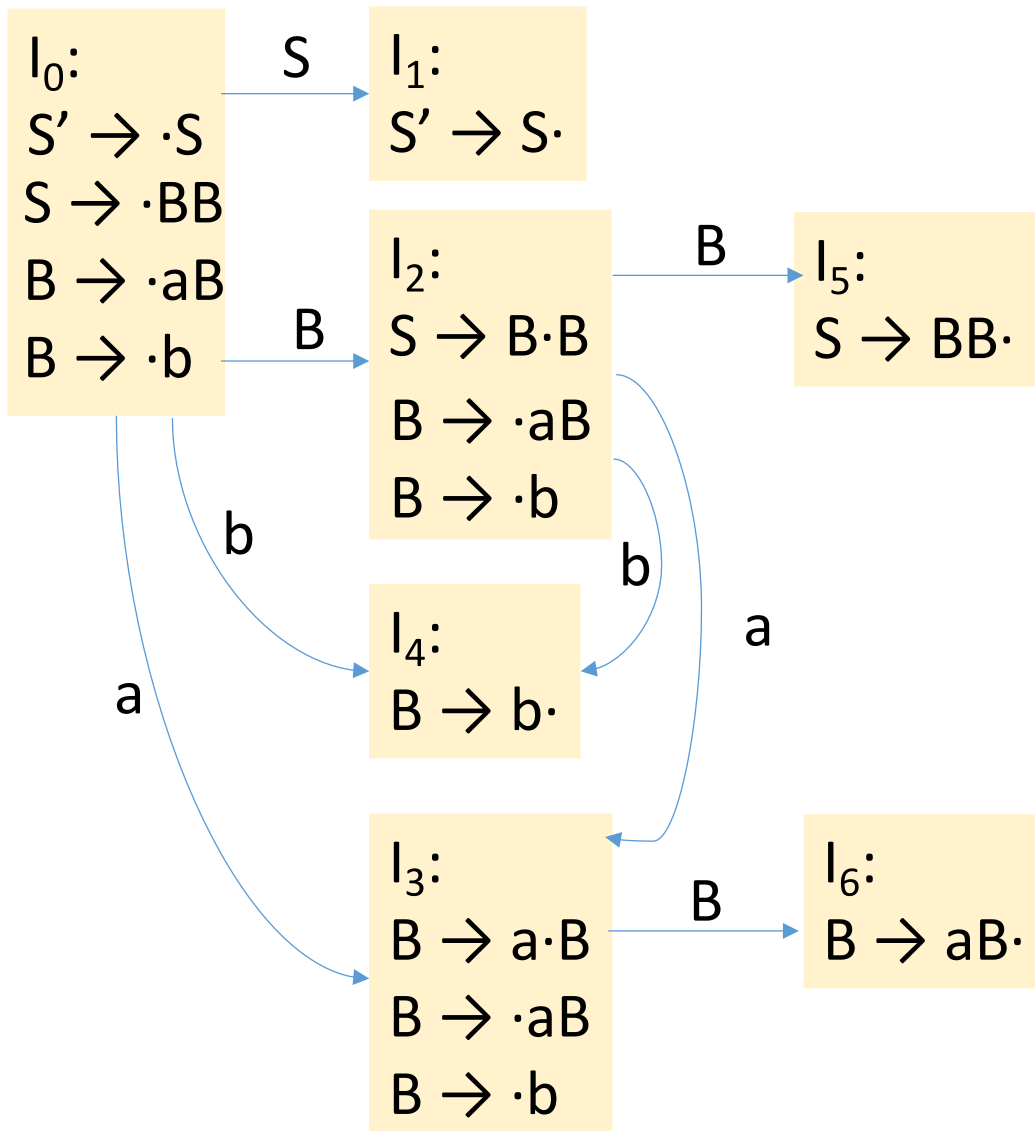
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



Example

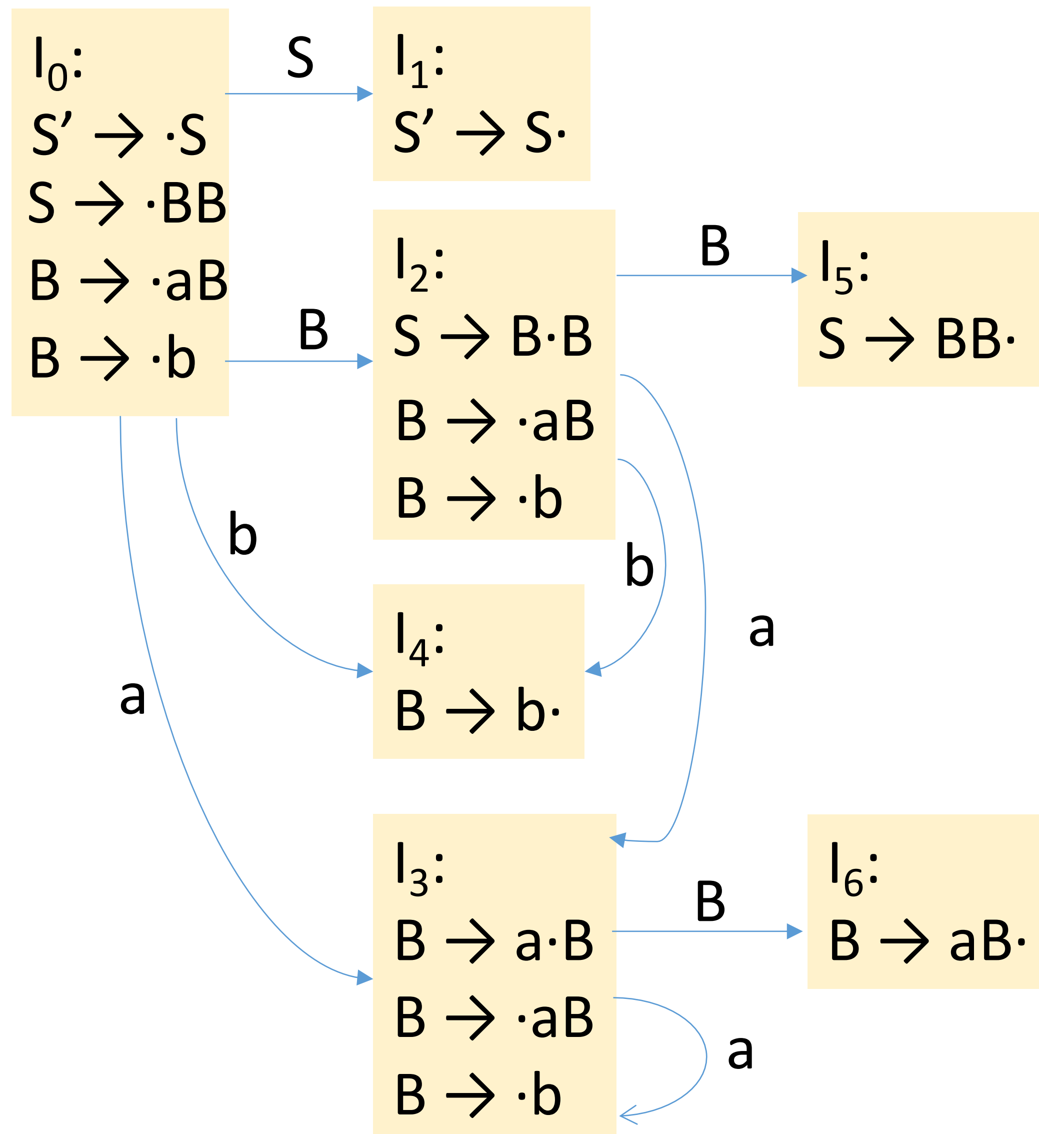
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



Example

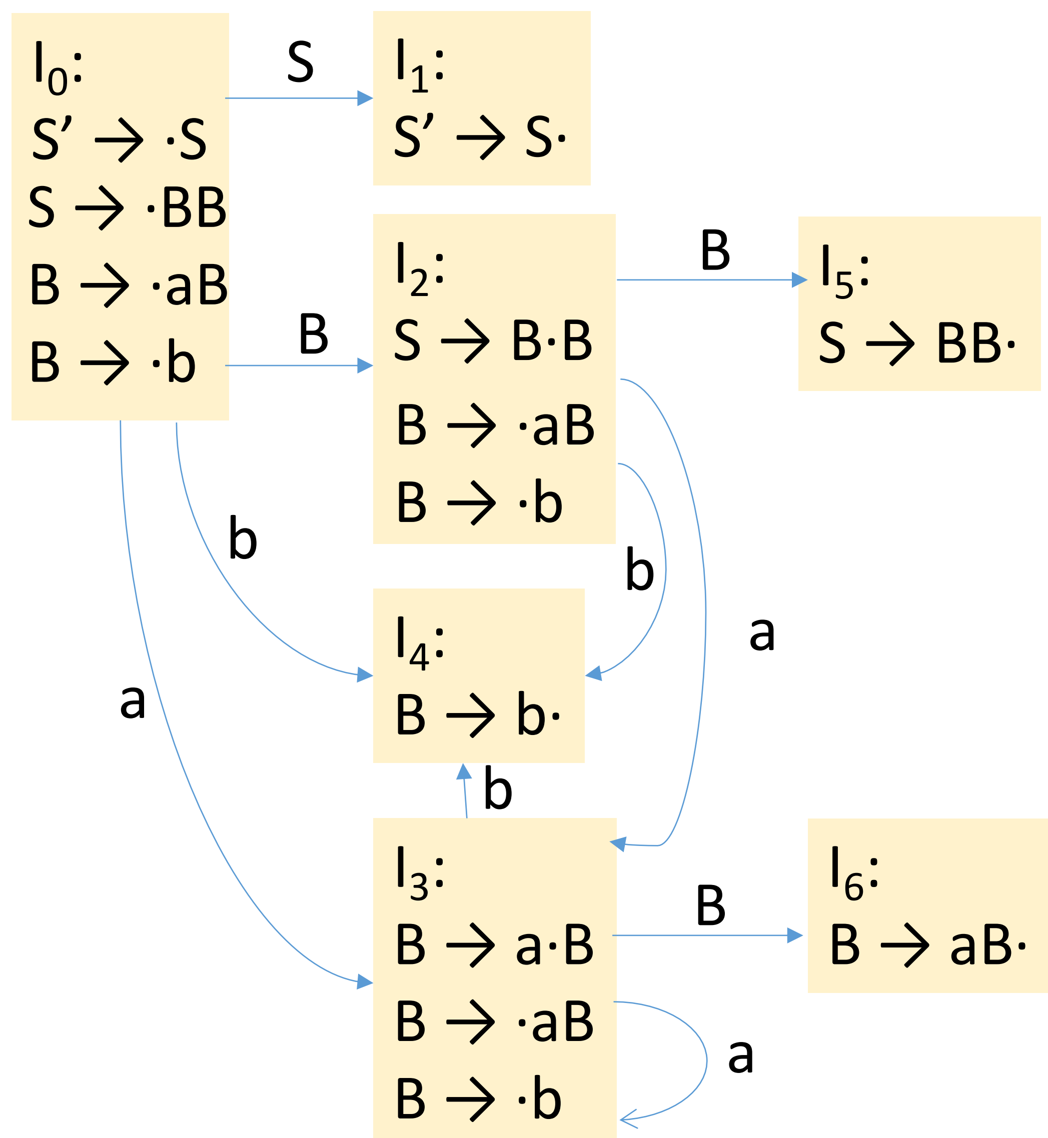
Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$



Example

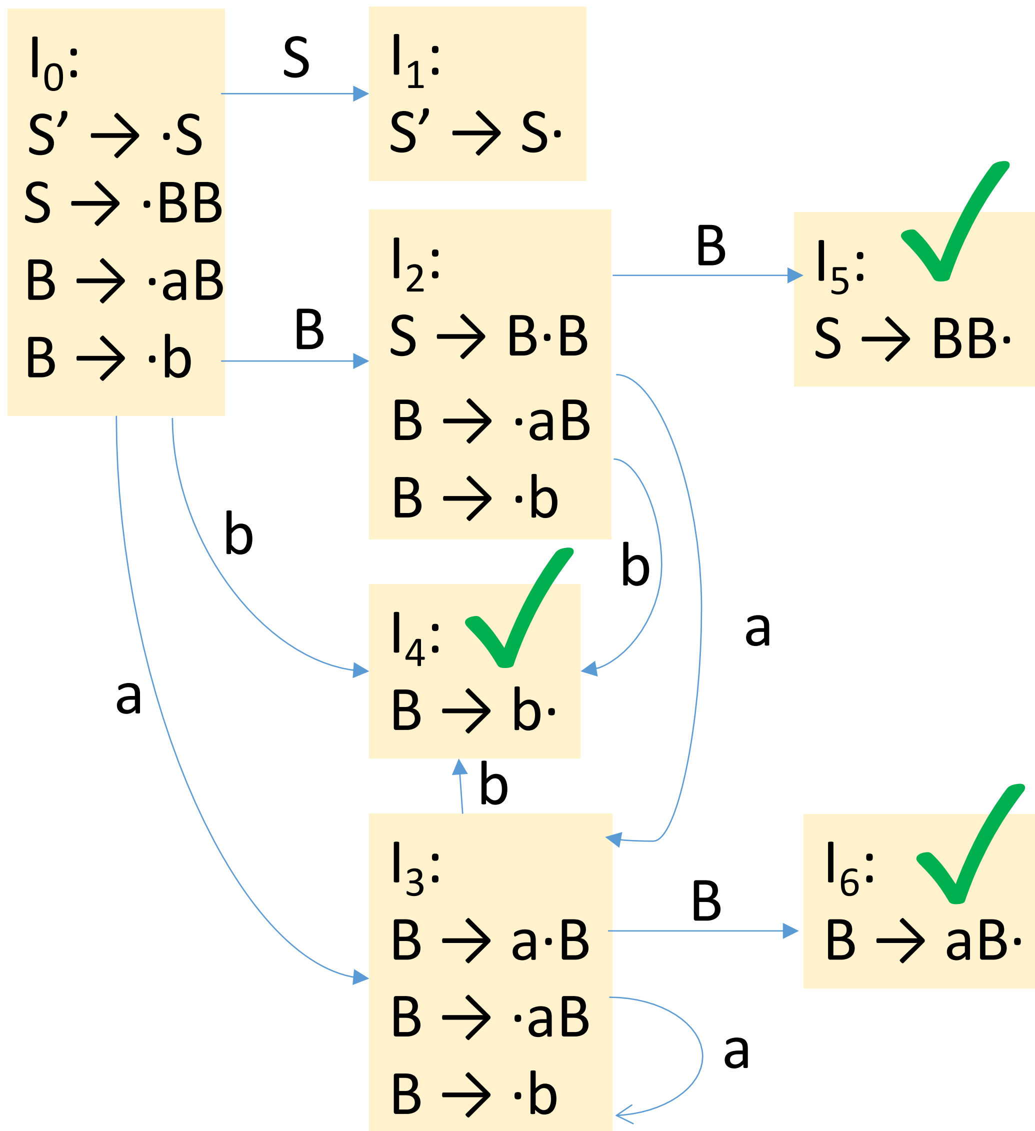
Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$



Example (cont.)

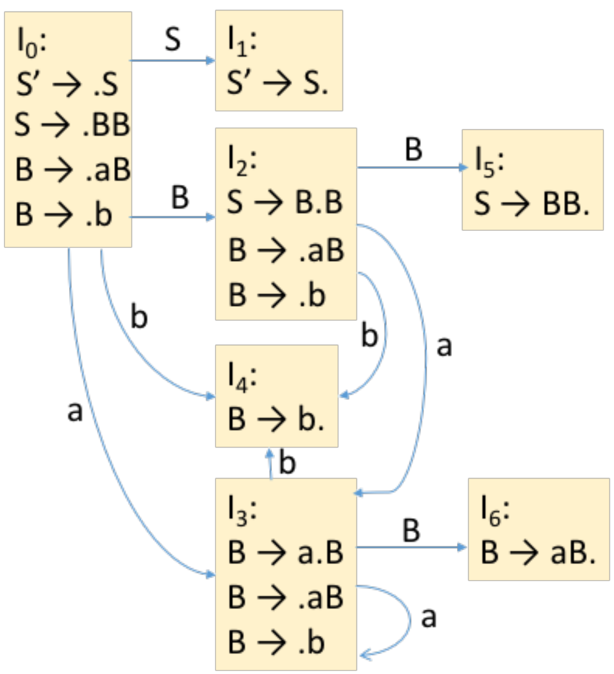
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



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 *j*” refers to the state corresponding to the set of items I_j

Example (cont.)

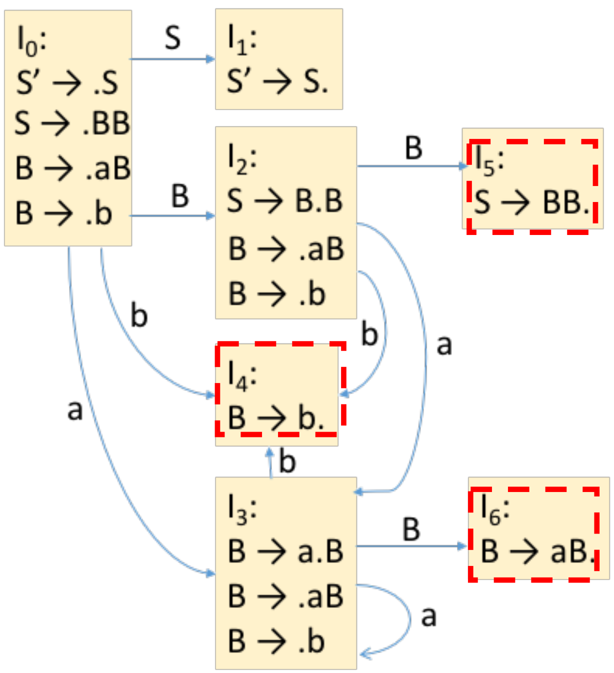
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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 *j*" refers to the state corresponding to the set of items I_j

Example (cont.)

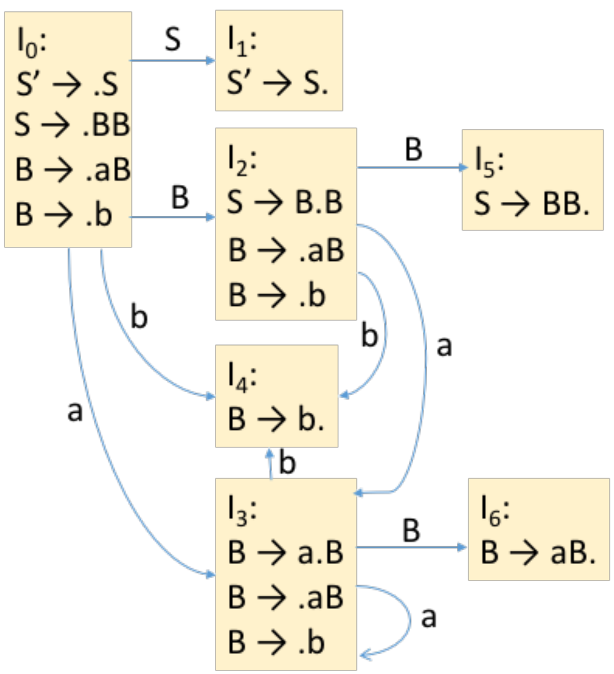
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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 *j*" refers to the state corresponding to the set of items I_j

Example (cont.)

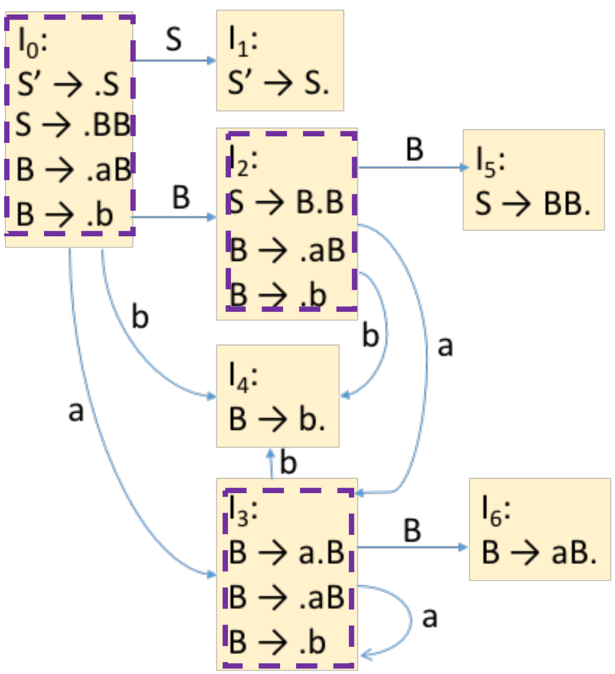
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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 *j*" refers to the state corresponding to the set of items I_j

Example (cont.)

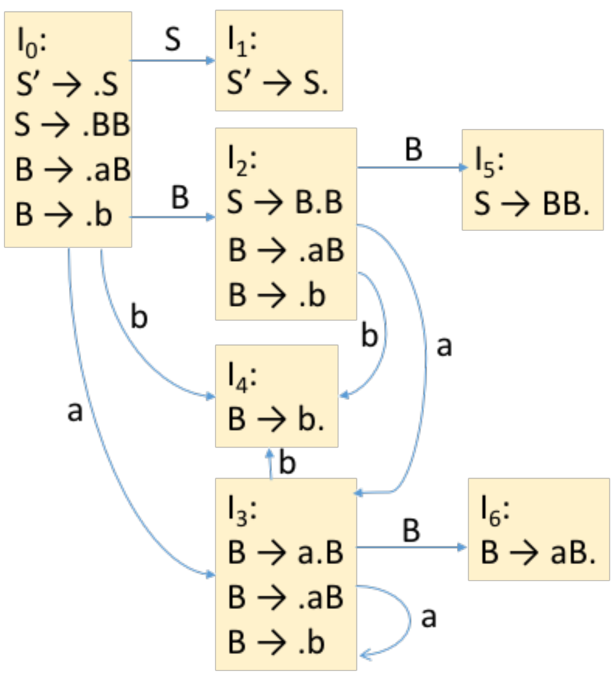
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



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 *j*” refers to the state corresponding to the set of items I_j

Example (cont.)

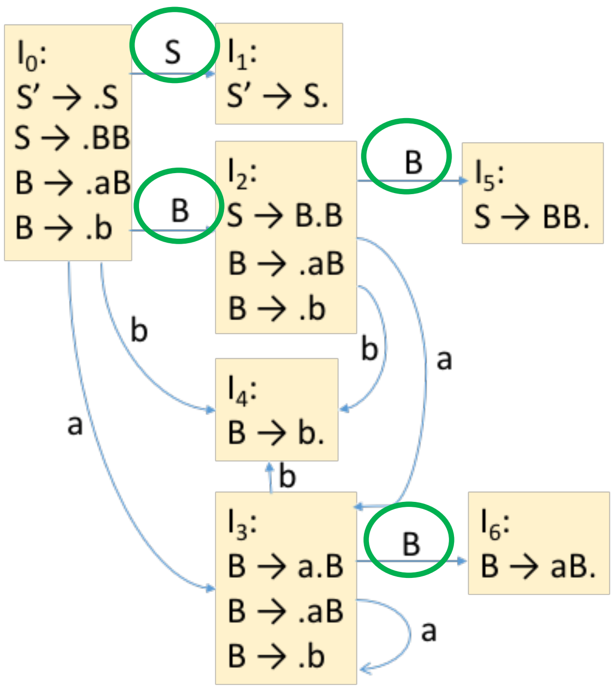
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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 *j*" refers to the state corresponding to the set of items I_j

Example (cont.)

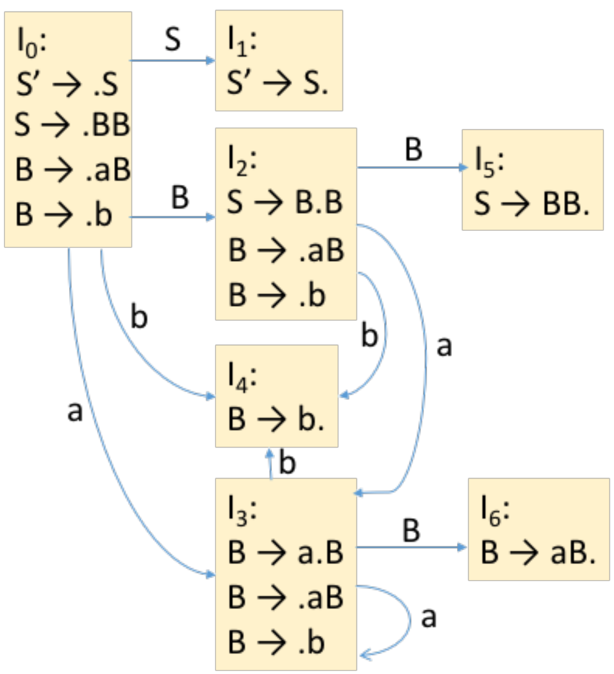
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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 *j*" refers to the state corresponding to the set of items I_j

CLOSURE()[闭包]

- **Closure of item sets**: if *I* is a set of items for a grammar *G*, then *closure(I)* is the set of items constructed from *I* by the two rules:
  - Initially, add every item in *I* to *CLOSURE(I)*
  - If $A \rightarrow \alpha \cdot B\beta$ is in *CLOSURE(I)* and $B \rightarrow \gamma$ is a production, then add item $B \rightarrow \cdot\gamma$ to *CLOSURE(I)*, if it is not already there
    - Apply this rule until no more new items can be added to *CLOSURE(I)*

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

CLOSURE()[闭包]

- **Closure of item sets**: if *I* is a set of items for a grammar *G*, then *closure(I)* is the set of items constructed from *I* by the two rules:
  - Initially, add every item in *I* to *CLOSURE(I)*
  - If $A \rightarrow \alpha \cdot B\beta$ is in *CLOSURE(I)* and $B \rightarrow \gamma$ is a production, then add item $B \rightarrow \cdot\gamma$ to *CLOSURE(I)*, if it is not already there
    - Apply this rule until no more new items can be added to *CLOSURE(I)*

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

$S' \rightarrow \cdot S$

CLOSURE()[闭包]

- **Closure of item sets**: if *I* is a set of items for a grammar *G*, then *closure(I)* is the set of items constructed from *I* by the two rules:
  - Initially, add every item in *I* to *CLOSURE(I)*
  - If $A \rightarrow \alpha{\cdot}B\beta$ is in *CLOSURE(I)* and $B \rightarrow \gamma$ is a production, then add item $B \rightarrow {\cdot}\gamma$ to *CLOSURE(I)*, if it is not already there
    - Apply this rule until no more new items can be added to *CLOSURE(I)*

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

$S' \rightarrow {\cdot}S$

CLOSURE()[闭包]

- **Closure of item sets**: if *I* is a set of items for a grammar *G*, then *closure(I)* is the set of items constructed from *I* by the two rules:
  - Initially, add every item in *I* to *CLOSURE(I)*
  - If $A \rightarrow \alpha \cdot B\beta$ is in *CLOSURE(I)* and $B \rightarrow \gamma$ is a production, then add item $B \rightarrow \cdot\gamma$ to *CLOSURE(I)*, if it is not already there
    - Apply this rule until no more new items can be added to *CLOSURE(I)*

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

$S' \rightarrow \cdot S$ ⇒ $S' \rightarrow \cdot S$

CLOSURE()[闭包]

- **Closure of item sets**: if *I* is a set of items for a grammar *G*, then *closure(I)* is the set of items constructed from *I* by the two rules:
  - Initially, add every item in *I* to *CLOSURE(I)*
  - If $A \to \alpha{\cdot}B\beta$ is in *CLOSURE(I)* and $B \to \gamma$ is a production, then add item $B \to {\cdot}\gamma$ to *CLOSURE(I)*, if it is not already there
    - Apply this rule until no more new items can be added to *CLOSURE(I)*

Grammar:

(0) $S' \to S$

(1) $S \to BB$

(2) $B \to aB$

(3) $B \to b$

$S' \to {\cdot}S$ ⇒ $S' \to {\cdot}S$, $S \to {\cdot}BB$

CLOSURE()[闭包]

- **Closure of item sets**: if *I* is a set of items for a grammar *G*, then *closure(I)* is the set of items constructed from *I* by the two rules:
  - Initially, add every item in *I* to *CLOSURE(I)*
  - If $A \rightarrow \alpha \cdot B\beta$ is in *CLOSURE(I)* and $B \rightarrow \gamma$ is a production, then add item $B \rightarrow \cdot\gamma$ to *CLOSURE(I)*, if it is not already there
    - Apply this rule until no more new items can be added to *CLOSURE(I)*

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

$S' \rightarrow \cdot S$ ⇒

$S' \rightarrow \cdot S$

$S \rightarrow \cdot BB$

$B \rightarrow \cdot aB$

$B \rightarrow \cdot b$

GOTO()[跳转]

- GOTO(I, X): returns state (i.e., set of items) that can be reached by advancing X
  - Where I is a set of items and X is a grammar symbol
  - The closure of the set of all items $[A \rightarrow \alpha X \cdot \beta]$ such that $[A \rightarrow \alpha \cdot X \beta]$ is in I
  - Used to define the transitions in the LR(0) automaton
    - The states of the automaton correspond to sets of items, and GOTO(I, X) specifies the transition from the state for I under input X

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

I_0:

$S' \rightarrow \cdot S$

$S \rightarrow \cdot BB$

$B \rightarrow \cdot aB$

$B \rightarrow \cdot b$

GOTO()[跳转]

- GOTO(*I*, *X*): returns state (i.e., set of items) that can be reached by advancing X
  - Where *I* is a set of items and *X* is a grammar symbol
  - The closure of the set of all items [A → αX·β] such that [A → α·Xβ] is in *I*
  - Used to define the transitions in the LR(0) automaton
    - The states of the automaton correspond to sets of items, and GOTO(I, X) specifies the transition from the state for *I* under input *X*

Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b

I_0:

S' → ·S

S → ·BB

B → ·aB

B → ·b

B

GOTO()[跳转]

- GOTO(I, X): returns state (i.e., set of items) that can be reached by advancing X
  - Where I is a set of items and X is a grammar symbol
  - The closure of the set of all items [$A \rightarrow \alpha X \cdot \beta$] such that [$A \rightarrow \alpha \cdot X \beta$] is in I
  - Used to define the transitions in the LR(0) automaton
    - The states of the automaton correspond to sets of items, and GOTO(I, X) specifies the transition from the state for I under input X

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

I_0:

$S' \rightarrow \cdot S$

$S \rightarrow \cdot BB$

$B \rightarrow \cdot aB$

$B \rightarrow \cdot b$

B

GOTO()[跳转]

- GOTO(*I*, *X*): returns state (i.e., set of items) that can be reached by advancing X
  - Where *I* is a set of items and *X* is a grammar symbol
  - The closure of the set of all items [A → αX·β] such that [A → α·Xβ] is in *I*
  - Used to define the transitions in the LR(0) automaton
    - The states of the automaton correspond to sets of items, and GOTO(I, X) specifies the transition from the state for *I* under input *X*

Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b

I_0:

S’ → ·S

S → ·BB

B → ·aB

B → ·b

B ⟶

I_2:

S → B·B

GOTO()[跳转]

- GOTO(I, X): returns state (i.e., set of items) that can be reached by advancing X
  - Where I is a set of items and X is a grammar symbol
  - The closure of the set of all items $[A \rightarrow \alpha X \cdot \beta]$ such that $[A \rightarrow \alpha \cdot X \beta]$ is in I
  - Used to define the transitions in the LR(0) automaton
    - The states of the automaton correspond to sets of items, and GOTO(I, X) specifies the transition from the state for I under input X

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

I_0:

$S' \rightarrow \cdot S$

$S \rightarrow \cdot BB$

$B \rightarrow \cdot aB$

$B \rightarrow \cdot b$

$\xrightarrow{B}$

I_2:

$S \rightarrow B \cdot B$

$B \rightarrow \cdot aB$

$B \rightarrow \cdot b$

Construct LR(0) States

- [增广文法]Create augmented grammar G' for G
  - Given G: $S \rightarrow \alpha \mid \beta$, create G': $S' \rightarrow S \quad S \rightarrow \alpha \mid \beta$
  - Creates a single rule $S' \rightarrow S$ that when reduced, signals acceptance
- [初始状态]Create 1st state by performing a closure on initial item $S' \rightarrow \cdot S$
  - Closure(I): creates state from an initial set of items I
  - $\text{Closure}(\{S' \rightarrow \cdot S\}) = \{S' \rightarrow \cdot S, S \rightarrow \cdot\alpha, S \rightarrow \cdot\beta\}$
- [添加状态]Create additional states by performing a goto on each symbol
  - Goto(I, X): creates state that can be reached from I by advancing X
  - If α was single symbol, the following new state would be created:
    $\text{Goto}(\{S' \rightarrow \cdot S, S \rightarrow \cdot\alpha, S \rightarrow \cdot\beta\}, \alpha) =$
    $\text{Closure}(\{S \rightarrow \alpha\cdot\}) = \{S \rightarrow \alpha\cdot\}$
- [重复操作]Repeatedly perform gotos until there are no more states to add

Construct DFA

- Compute canonical LR(0) collection[规范LR(0)项集族, C], i.e., set of all states in DFA
  - One collection of sets of LR(0) items provides the basis for constructing a DFA that is used to make parsing decisions
  - Such an automaton is called an **LR(0) automaton**
    - Each state of the LR(0) automaton represents a set of items in the C
- All new states are added through GOTO(I, X)
  - State transitions are done on symbol X

```
void items(G') {                       // G': the augmented grammar
   C = { CLOSURE({[S' → ·S]}) }; // C: the canonical collection of sets of LR(0) items
   repeat
      for ( each state I in C )
            for ( each grammar symbol X )
               if ( GOTO(I, X) is not empty and not in C)
                  add GOTO(I, X) to C;
    until no new states are added to C
}
```

LR(0) Automaton[自动机]

- The LR(0) automaton: each time we perform a shift we are following a transition to a new state
  - States: the sets of items in C
    - Start state: CLOSURE({[S' → ·S]})
    - State j refers to the state corresponding to the set of items I_j
  - Transitions are given by the GOTO function
- How can the automaton help with shift-reduce decisions?
  - Suppose that the string γ of grammar symbols takes the LR(0) automaton from the start state 0 to some state j
  - Then, shift on next input symbol a if state j has a transition on a
  - Otherwise, we choose to reduce
    - The items in state j tell us which production to use (e.g., $E \to \alpha$)
    - $E \to \alpha$: pop states for α, bringing state x to the top and look for a transition on E to state y (i.e., state x has a transition on E to state y), which is then pushed to stack

The Example

Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b

- S_0 = Closure({ })

… … …

The Example

Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b

- S_0 = Closure({S' → .S})

… … …

The Example

Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b

- S_0 = Closure({S' → .S})

  = {S' →.S, S →.BB, B → .aB, B → .b}

… … …

The Example

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

- S_0 = Closure({$S' \rightarrow .S$})

  = {$S' \rightarrow .S$, $S \rightarrow .BB$, $B \rightarrow .aB$, $B \rightarrow .b$}

- Goto(S_0, B) = closure({ })

… … …

The Example

Grammar:

(0) $S' \to S$

(1) $S \to BB$

(2) $B \to aB$

(3) $B \to b$

- S_0 = Closure({$S' \to .S$})

  = {$S' \to .S$, $S \to .BB$, $B \to .aB$, $B \to .b$}

- Goto(S_0, B) = closure({$S \to B.B$})

… … …

The Example

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

- S_0 = Closure($\{S' \rightarrow .S\}$)

  = $\{S' \rightarrow .S,\ S \rightarrow .BB,\ B \rightarrow .aB,\ B \rightarrow .b\}$

- Goto(S_0, B) = closure($\{S \rightarrow B.B\}$)

  = $\{S \rightarrow B.B,\ B \rightarrow .aB,\ B \rightarrow .b\}$

… … …

The Example

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

- S_0 = Closure({$S' \rightarrow .S$})

  = {$S' \rightarrow .S$, $S \rightarrow .BB$, $B \rightarrow .aB$, $B \rightarrow .b$}
- Goto(S_0, B) = closure({$S \rightarrow B.B$})

  = {$S \rightarrow B.B$, $B \rightarrow .aB$, $B \rightarrow .b$}
- Goto(S_0, a) = closure({ })

… … …

The Example

Grammar:

(0) $S' \to S$

(1) $S \to BB$

(2) $B \to aB$

(3) $B \to b$

- S_0 = Closure({$S' \to .S$})

  = {$S' \to .S$, $S \to .BB$, $B \to .aB$, $B \to .b$}
- Goto(S_0, B) = closure({$S \to B.B$})

  = {$S \to B.B$, $B \to .aB$, $B \to .b$}
- Goto(S_0, a) = closure({$B \to a.B$})

… … …

The Example

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

- S_0 = Closure({$S' \rightarrow .S$})

  = {$S' \rightarrow .S$, $S \rightarrow .BB$, $B \rightarrow .aB$, $B \rightarrow .b$}
- Goto(S_0, B) = closure({$S \rightarrow B.B$})

  = {$S \rightarrow B.B$, $B \rightarrow .aB$, $B \rightarrow .b$}
- Goto(S_0, a) = closure({$B \rightarrow a.B$})

  = {$B \rightarrow a.B$, $B \rightarrow .aB$, $B \rightarrow .b$}

… … …

The Example

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

- S_0 = Closure({$S' \rightarrow .S$})

  = {$S' \rightarrow .S$, $S \rightarrow .BB$, $B \rightarrow .aB$, $B \rightarrow .b$}
- Goto(S_0, B) = closure({$S \rightarrow B.B$})

  = {$S \rightarrow B.B$, $B \rightarrow .aB$, $B \rightarrow .b$}
- Goto(S_0, a) = closure({$B \rightarrow a.B$})

  = {$B \rightarrow a.B$, $B \rightarrow .aB$, $B \rightarrow .b$}
- Goto(S_0, b) = closure({ })

… … …

SUN YAT-SEN UNIVERSITY

The Example

Grammar:

(0) $S' \to S$

(1) $S \to BB$

(2) $B \to aB$

(3) $B \to b$

- S_0 = Closure({$S' \to .S$})

  = {$S' \to .S$, $S \to .BB$, $B \to .aB$, $B \to .b$}

- Goto(S_0, B) = closure({$S \to B.B$})

  = {$S \to B.B$, $B \to .aB$, $B \to .b$}

- Goto(S_0, a) = closure({$B \to a.B$})

  = {$B \to a.B$, $B \to .aB$, $B \to .b$}

- Goto(S_0, b) = closure({$B \to b.$})

… … …

SUN YAT-SEN UNIVERSITY

The Example

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

- S_0 = Closure({$S' \rightarrow .S$})

  = {$S' \rightarrow .S$, $S \rightarrow .BB$, $B \rightarrow .aB$, $B \rightarrow .b$}

- Goto(S_0, B) = closure({$S \rightarrow B.B$})

  = {$S \rightarrow B.B$, $B \rightarrow .aB$, $B \rightarrow .b$}

- Goto(S_0, a) = closure({$B \rightarrow a.B$})

  = {$B \rightarrow a.B$, $B \rightarrow .aB$, $B \rightarrow .b$}

- Goto(S_0, b) = closure({$B \rightarrow b.$})

  = {$B \rightarrow b.$}

… … …

The Example

Grammar:

(0) $S' \to S$

(1) $S \to BB$

(2) $B \to aB$

(3) $B \to b$

- S_0 = Closure({$S' \to .S$})

  = {$S' \to .S$, $S \to .BB$, $B \to .aB$, $B \to .b$}

- Goto(S_0, B) = closure({$S \to B.B$})

  S_2 = {$S \to B.B$, $B \to .aB$, $B \to .b$}

- Goto(S_0, a) = closure({$B \to a.B$})

  S_3 = {$B \to a.B$, $B \to .aB$, $B \to .b$}

- Goto(S_0, b) = closure({$B \to b.$})

  S_4 = {$B \to b.$}

… … …

The Example

Grammar:

(0) $S' \rightarrow S$

(1) $S \rightarrow BB$

(2) $B \rightarrow aB$

(3) $B \rightarrow b$

- S_0 = Closure({$S' \rightarrow .S$})

  = {$S' \rightarrow .S$, $S \rightarrow .BB$, $B \rightarrow .aB$, $B \rightarrow .b$}

- Goto(S_0, B) = closure({$S \rightarrow B.B$})

  S_2 = {$S \rightarrow B.B$, $B \rightarrow .aB$, $B \rightarrow .b$}

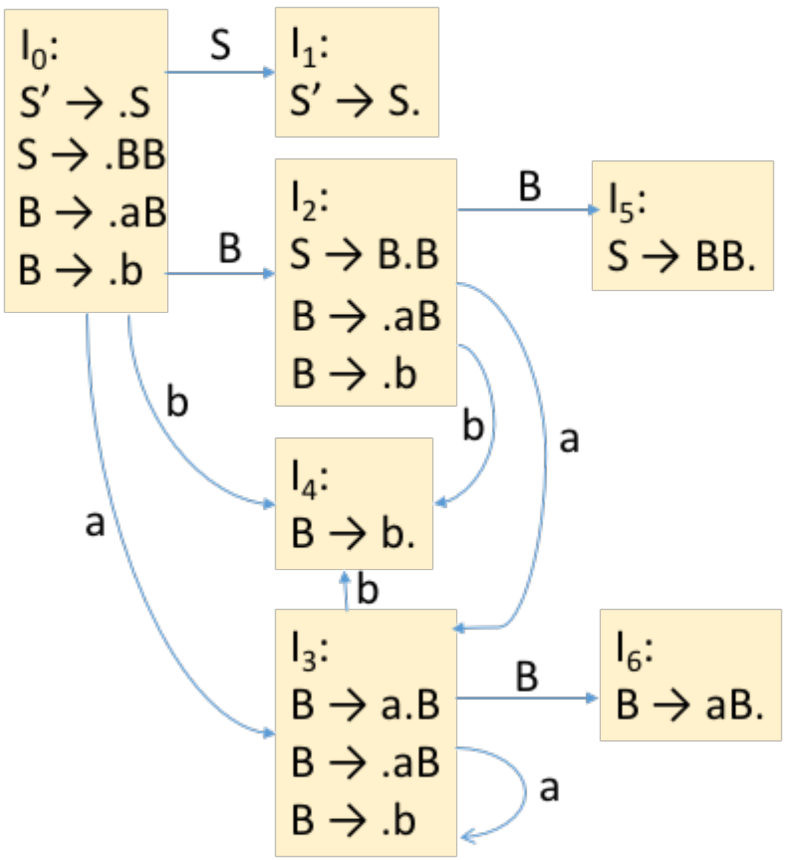
- Goto(S_0, a) = closure({$B \rightarrow a.B$})

  S_3 = {$B \rightarrow a.B$, $B \rightarrow .aB$, $B \rightarrow .b$}

- Goto(S_0, b) = closure({$B \rightarrow b.$})

  S_4 = {$B \rightarrow b.$}

… … …



Build Parse Table from DFA

- ACTION [*state*, *terminal symbol*]
- GOTO [*state*, *non-terminal symbol*]
- ACTION[动作]
  - If [A→α·aβ] is in S_i and goto(S_i, a) = S_j, where "a" is a terminal then ACTION[S_i, a] = shift j (s*j*)
  - If [A→α·] is in S_i and A→α is rule numbered j then ACTION[S_i, a] = reduce j (r*j*)
  - If [S' →S·] is in S_i then ACTION[S_i, $] = accept
  - If no conflicts among 'shift' and 'reduce' (the first two 'if's) then this parser is able to parse the given grammar
- GOTO[跳转]
  - if goto(S_i, A) = S_j then GOTO[S_i, A] = *j*
- All entries not filled are rejects

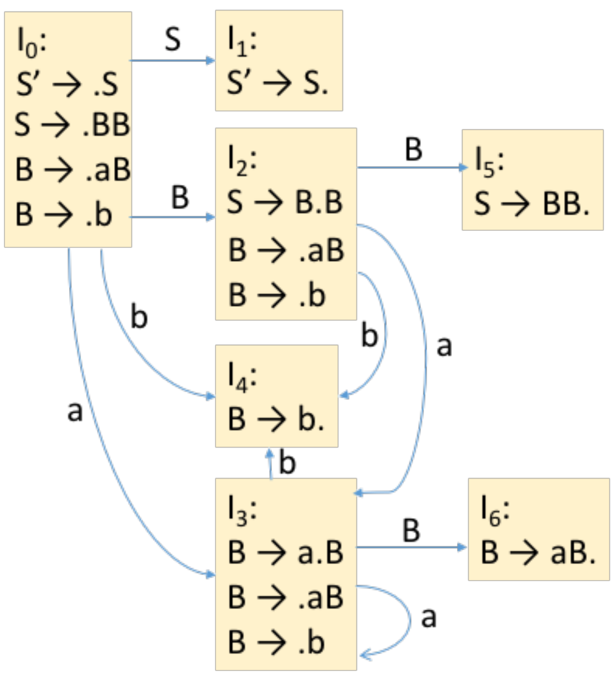
State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

中山大學 SUN YAT-SEN UNIVERSITY

The Example

Grammar:

(0) S' → S
(1) S → BB
(2) B → aB
(3) B → b



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		

The Example

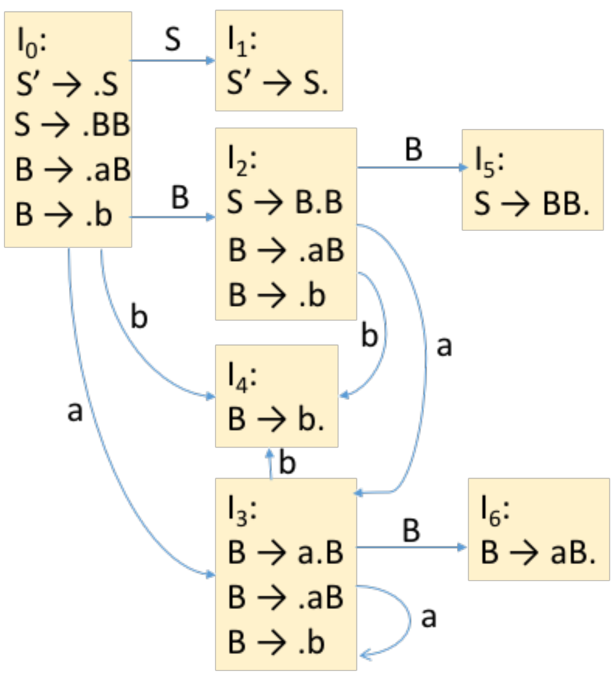
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

The Example

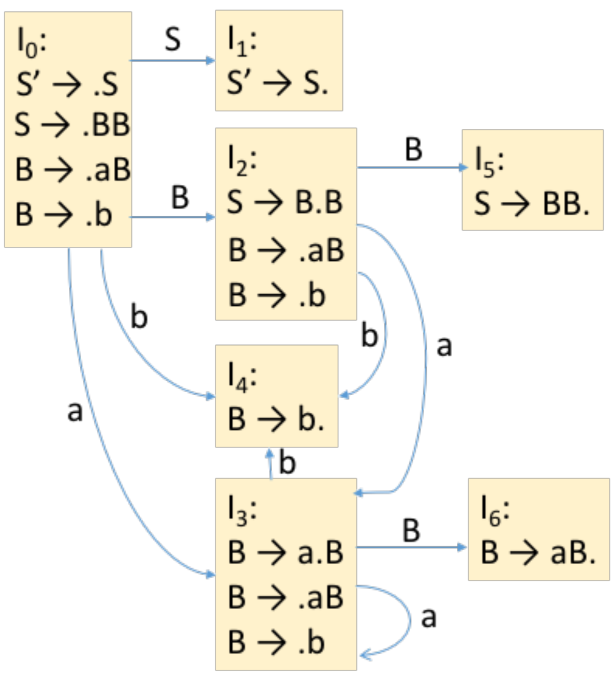
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$
=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

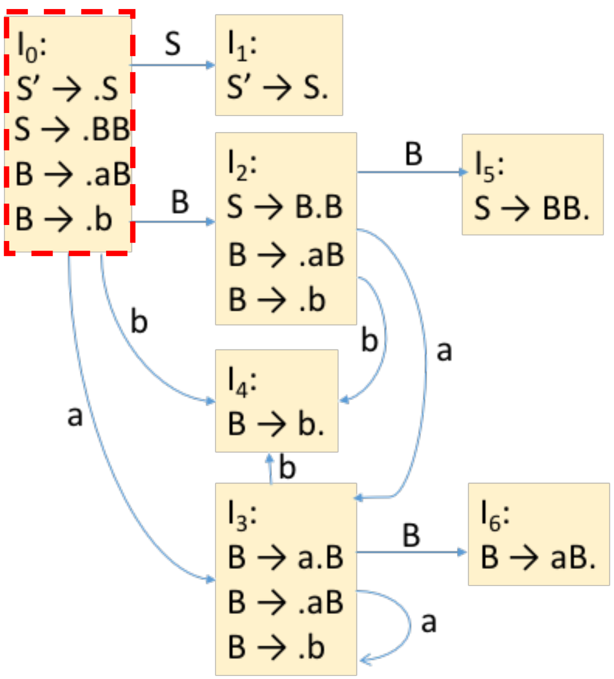
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$

=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

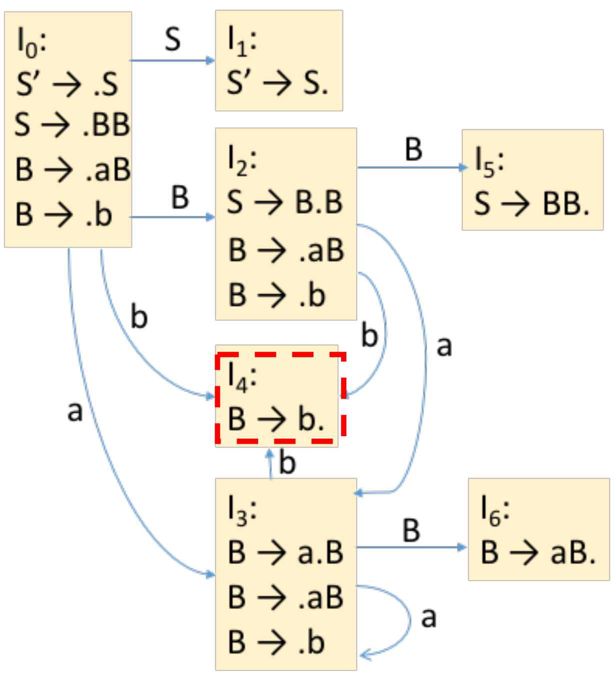
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$

=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

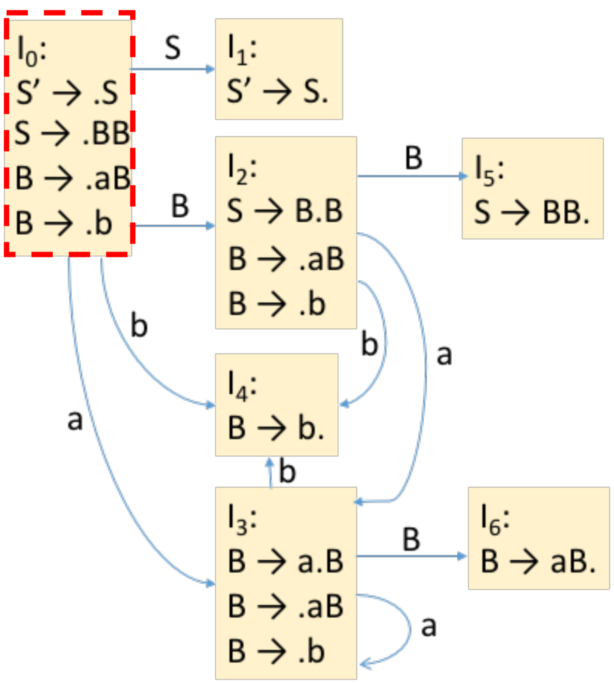
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$

=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

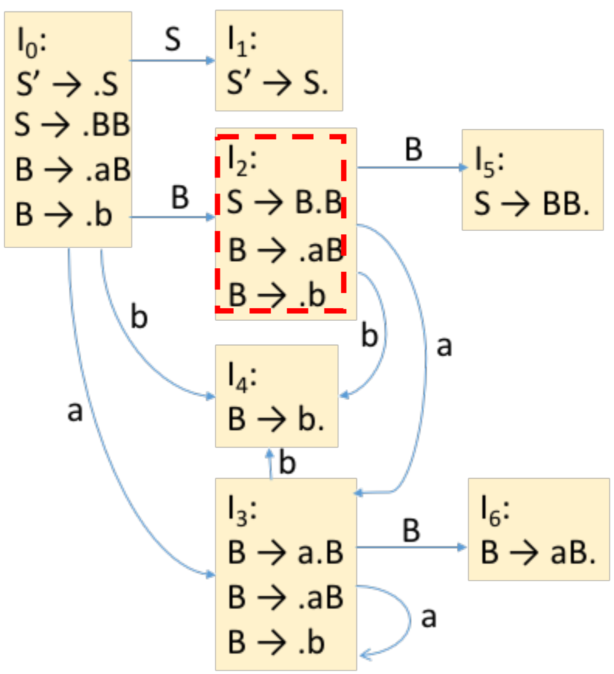
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$
=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

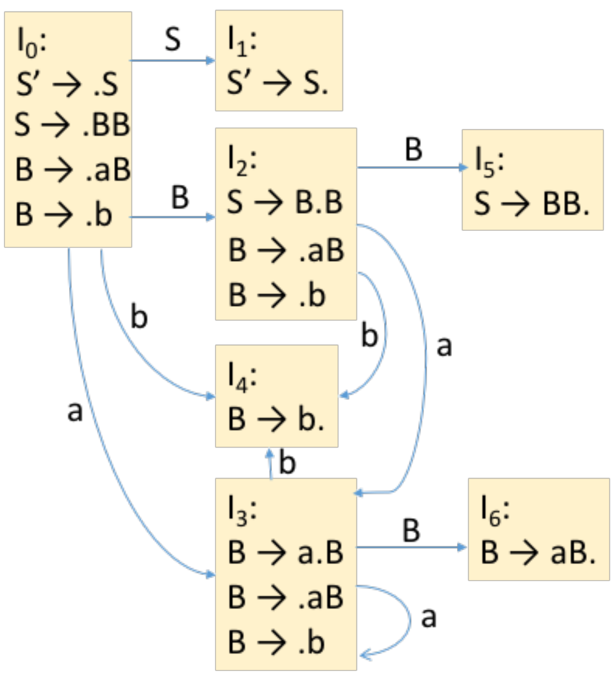
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$
=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

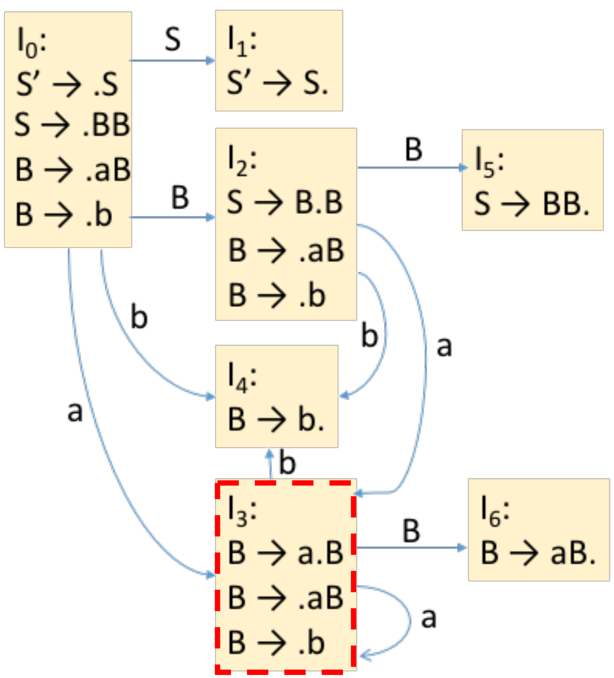
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$ => Bab#$ => BaB#$ => BB#$ => S#$

The Example

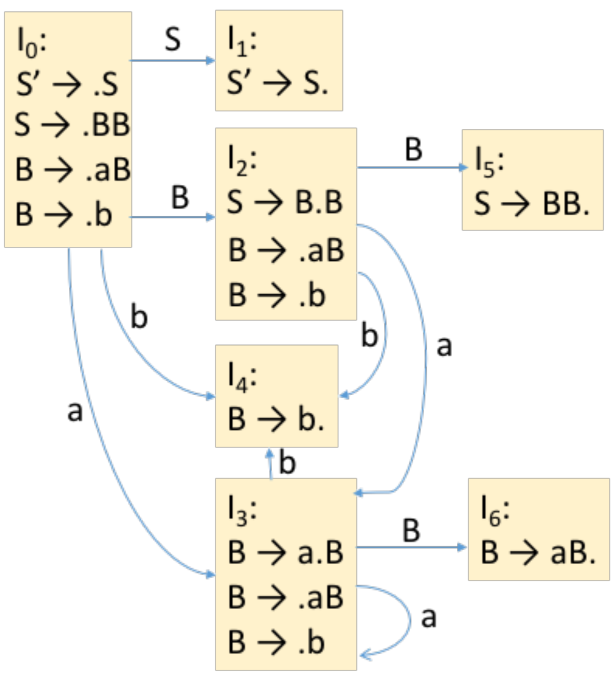
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$

=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

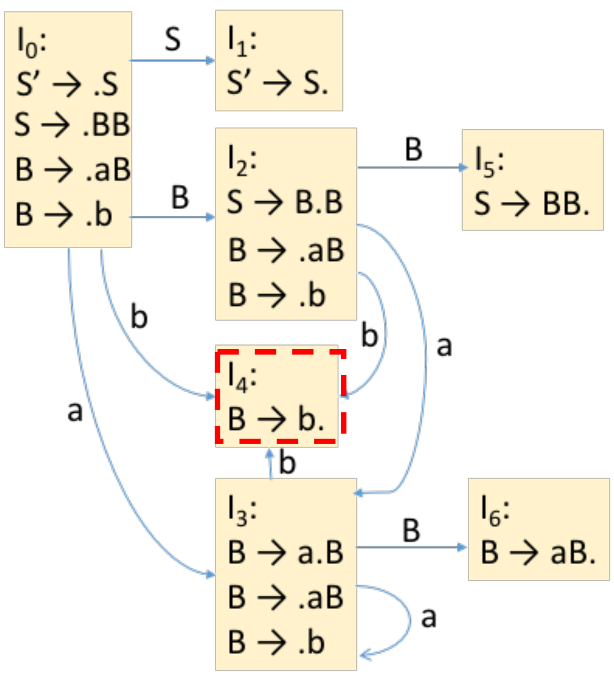
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$ => Bab#$ => BaB#$ => BB#$ => S#$

The Example

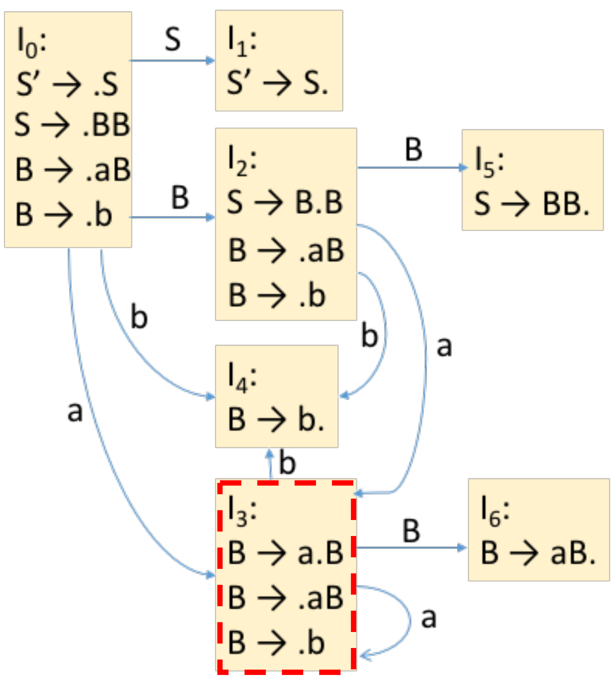
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$
=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

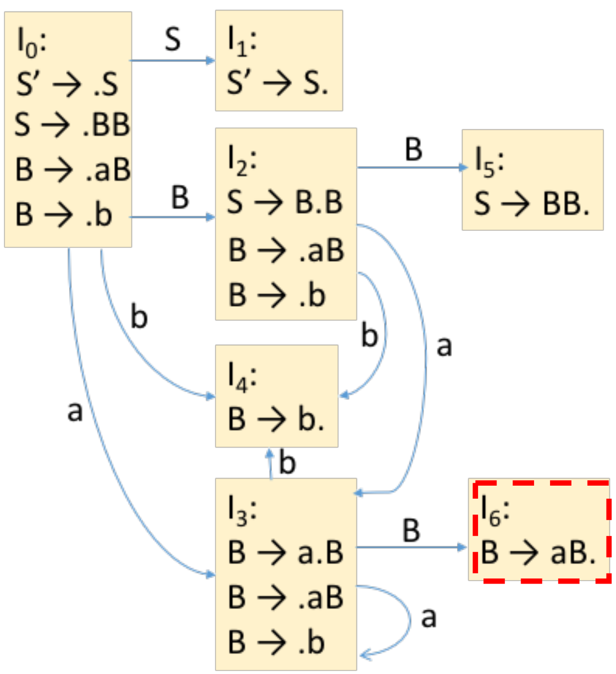
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$
=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

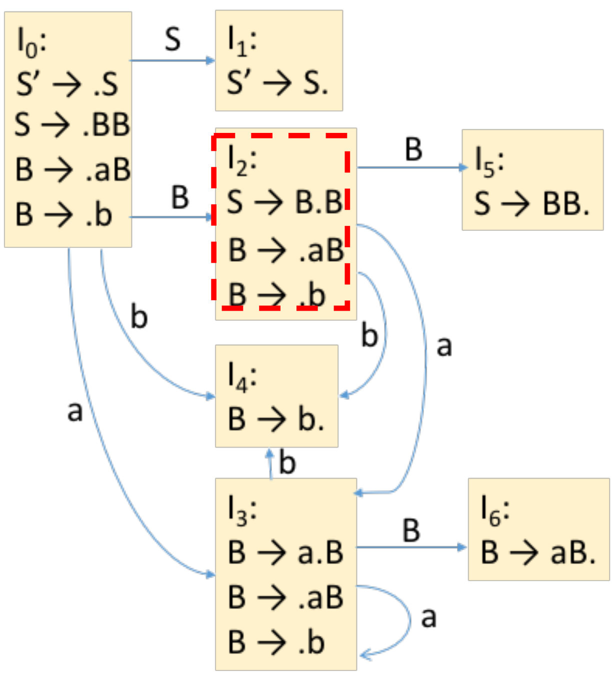
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$

=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

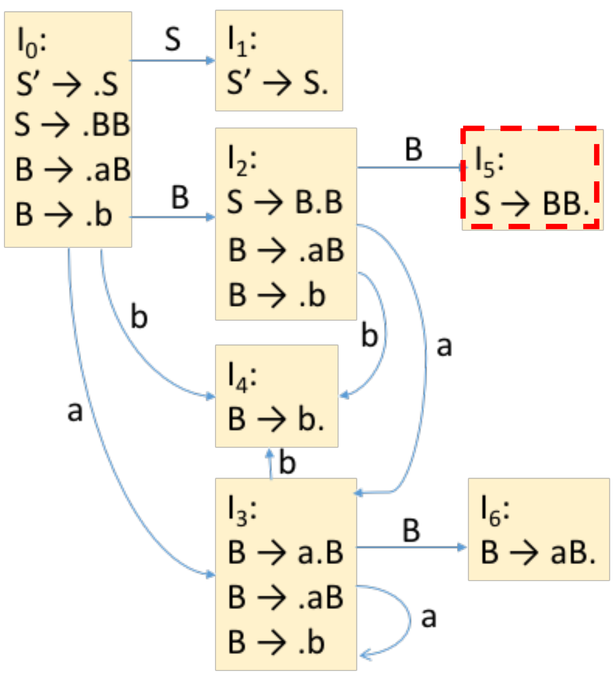
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$
=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

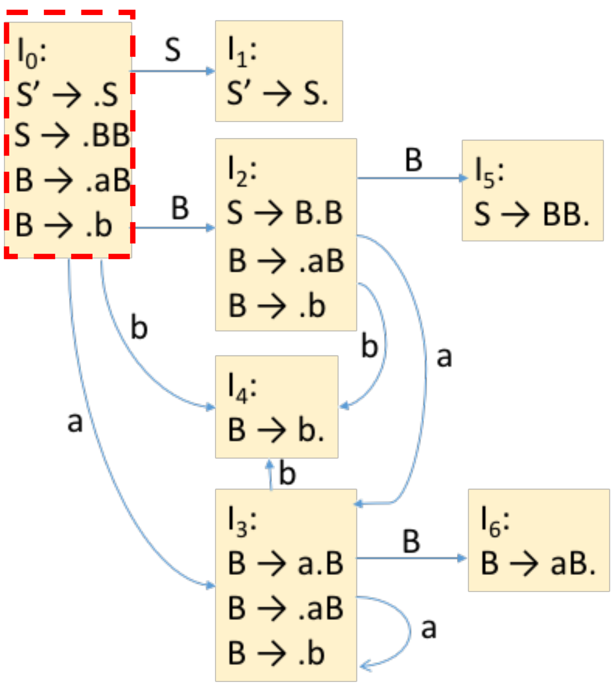
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$
=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

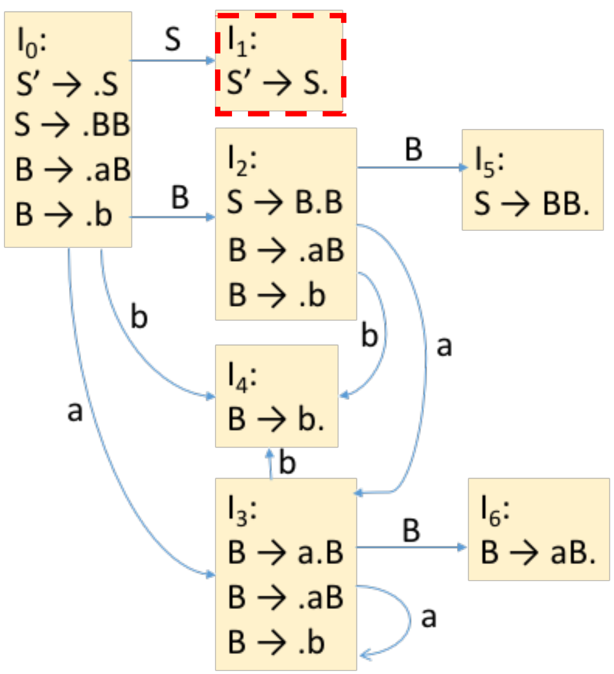
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$

=> Bab#$ => BaB#$ => BB#$ => S#$

The Example

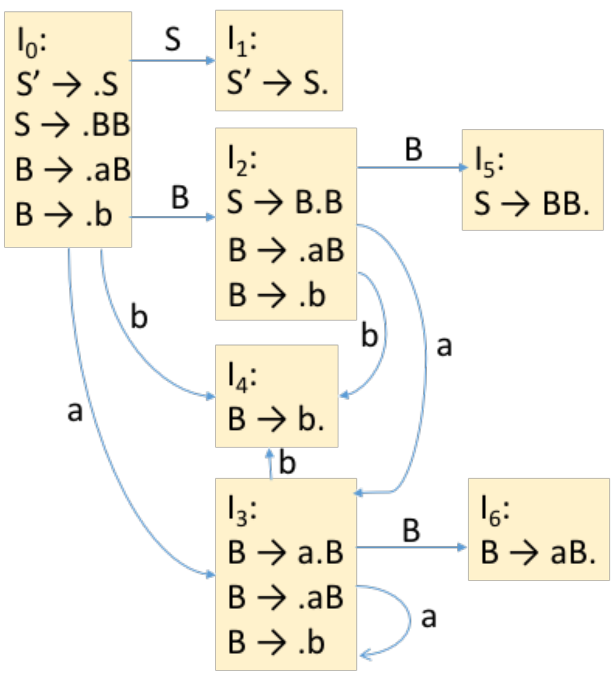
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



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		

String: bab

#bab$ => b#ab$ => B#ab$ => Ba#b$
=> Bab#$ => BaB#$ => BB#$ => S#$