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12讲：语法分析(8)

张献伟

xianweiz.github.io

DCS290, 4/11/2024

Review Questions

- Q1: what is LR(k)?

  L: scan input from left to right

  R: construct a rightmost derivation in reverse

  k: number of lookahead symbols

- Q2: what are the components of a LR parser?

  Input buffer, stack, parse table, driver.

- Q3: describe LR parse table.

  ACTION for terminals+$, GOTO for non-terminals;

  Each row is an state.

State	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	r2	r2	r2		5

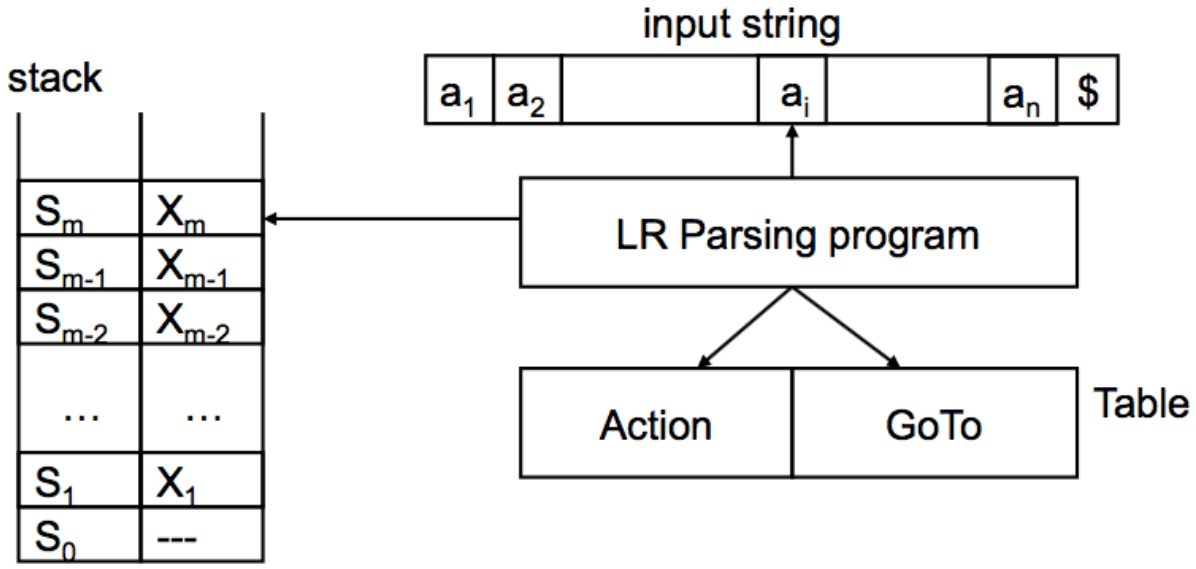
- Q4: what does ‘s3’ mean?

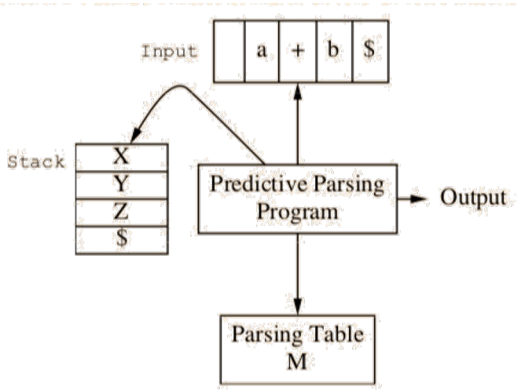
  Shift a token from buffer to stack, and associate state ‘3’.

- Q5: what does ‘r2’ mean?

  Reduce using the rule numbered ‘2’.

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

Example: Parse Table

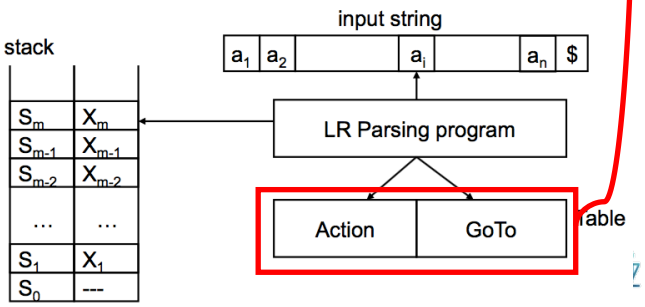
Grammar:

(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



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)

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)?
- 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 an string derivable from *XYZ* next on the input
  - A → X·YZ
    - Indicates that we have just seen on the input an string derivable from *X* and that we hope next to see an 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 → X·YZ
  - We have just recognized *X* and expect the upcoming input to contain a sequence derivable from *YZ* (say, Y → 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 → X·YZ

      Y → ·u

      Y → ·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

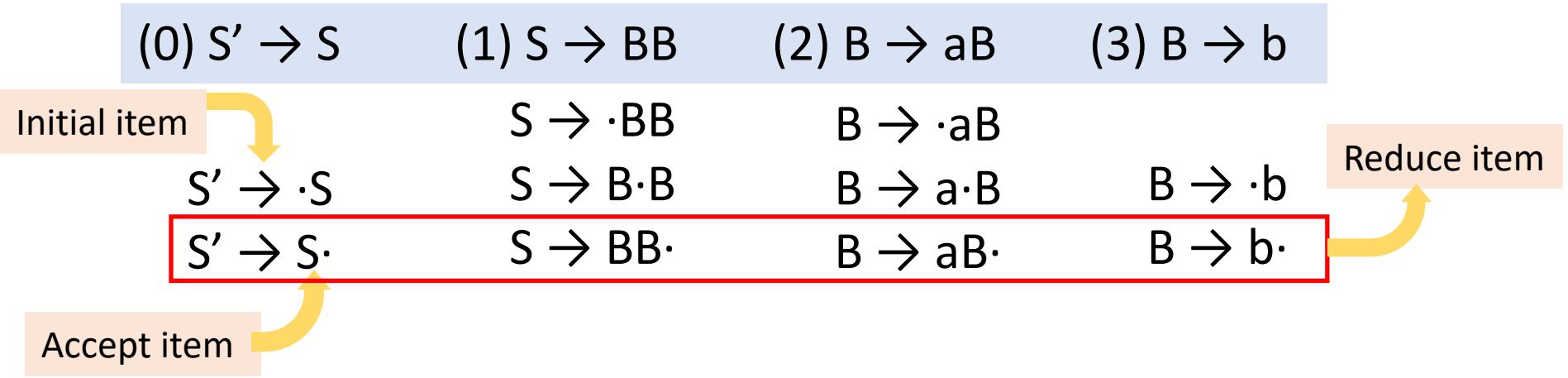
Grammar:

(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

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’ → S

(1) S → BB

(2) B → aB

(3) B → b

I_0:

S’ → ·S

S → ·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

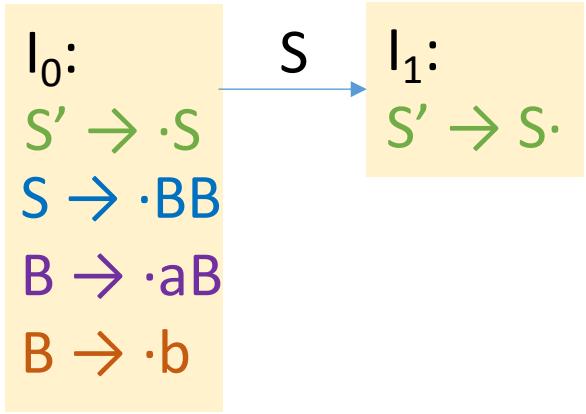
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



Example

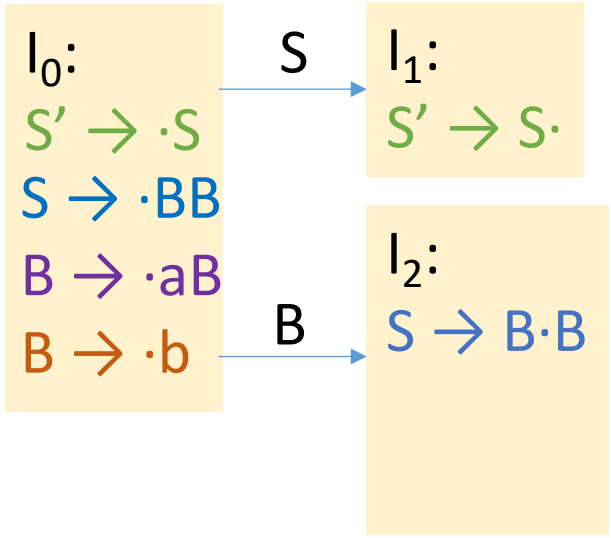
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



Example

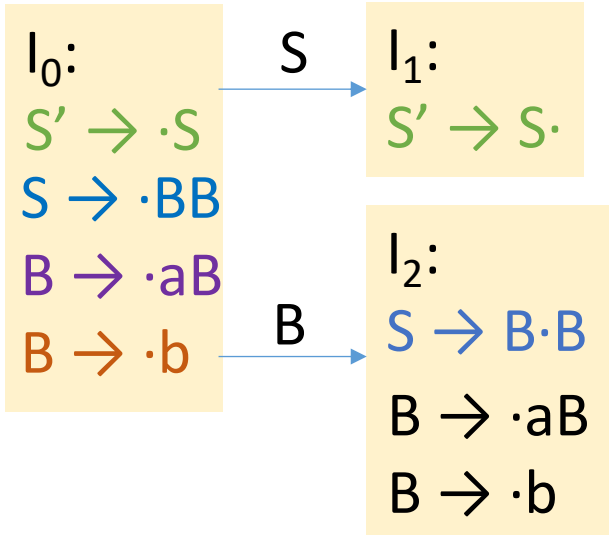
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



Example

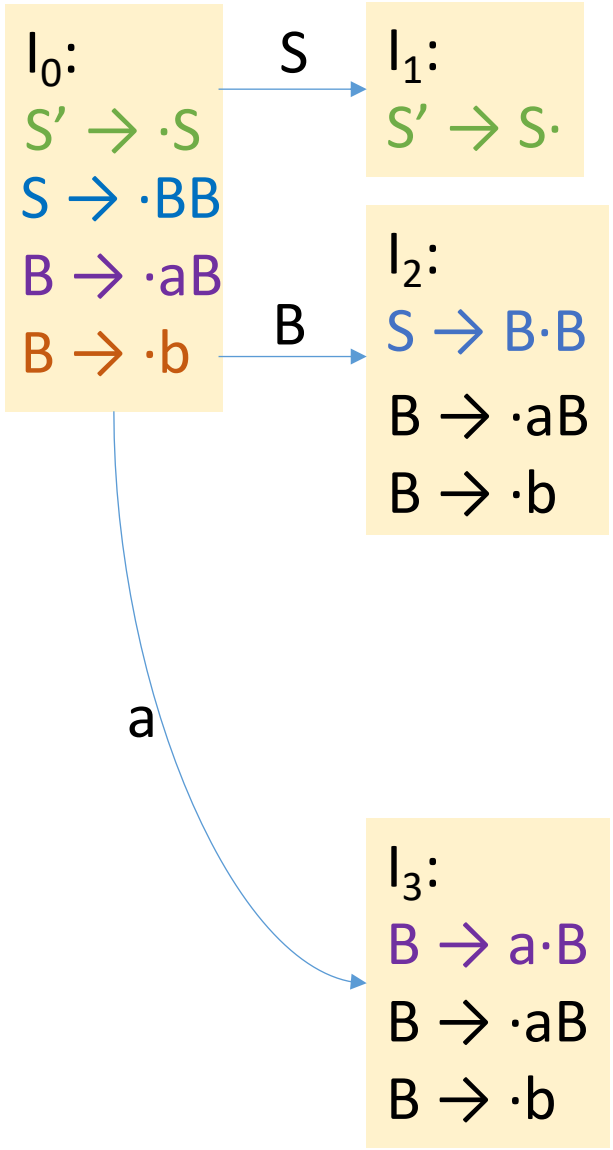
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



Example

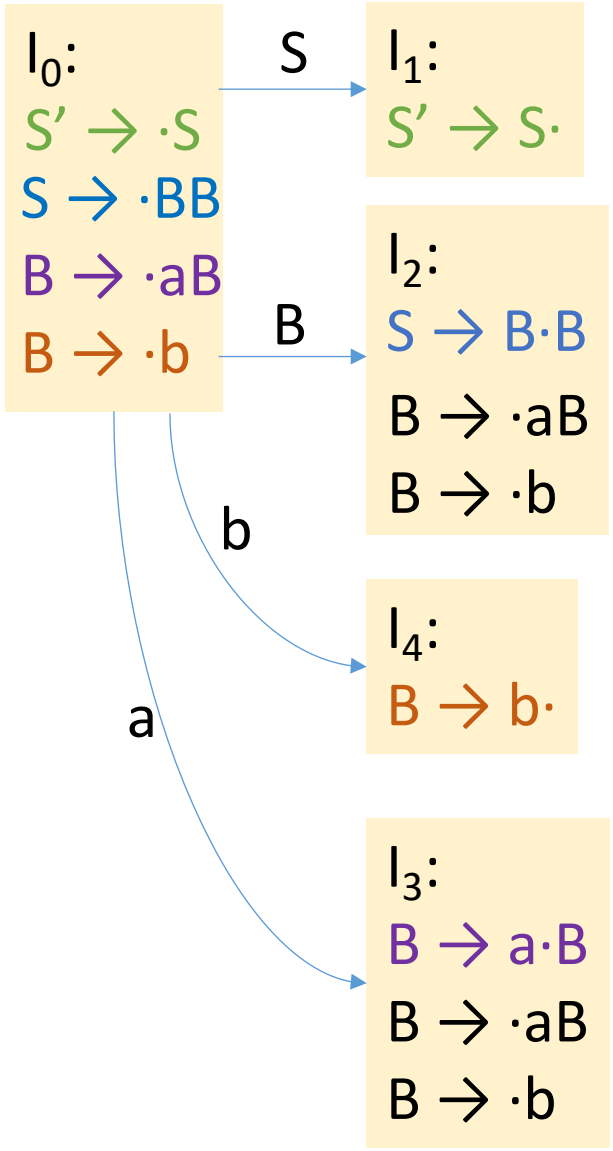
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



Example

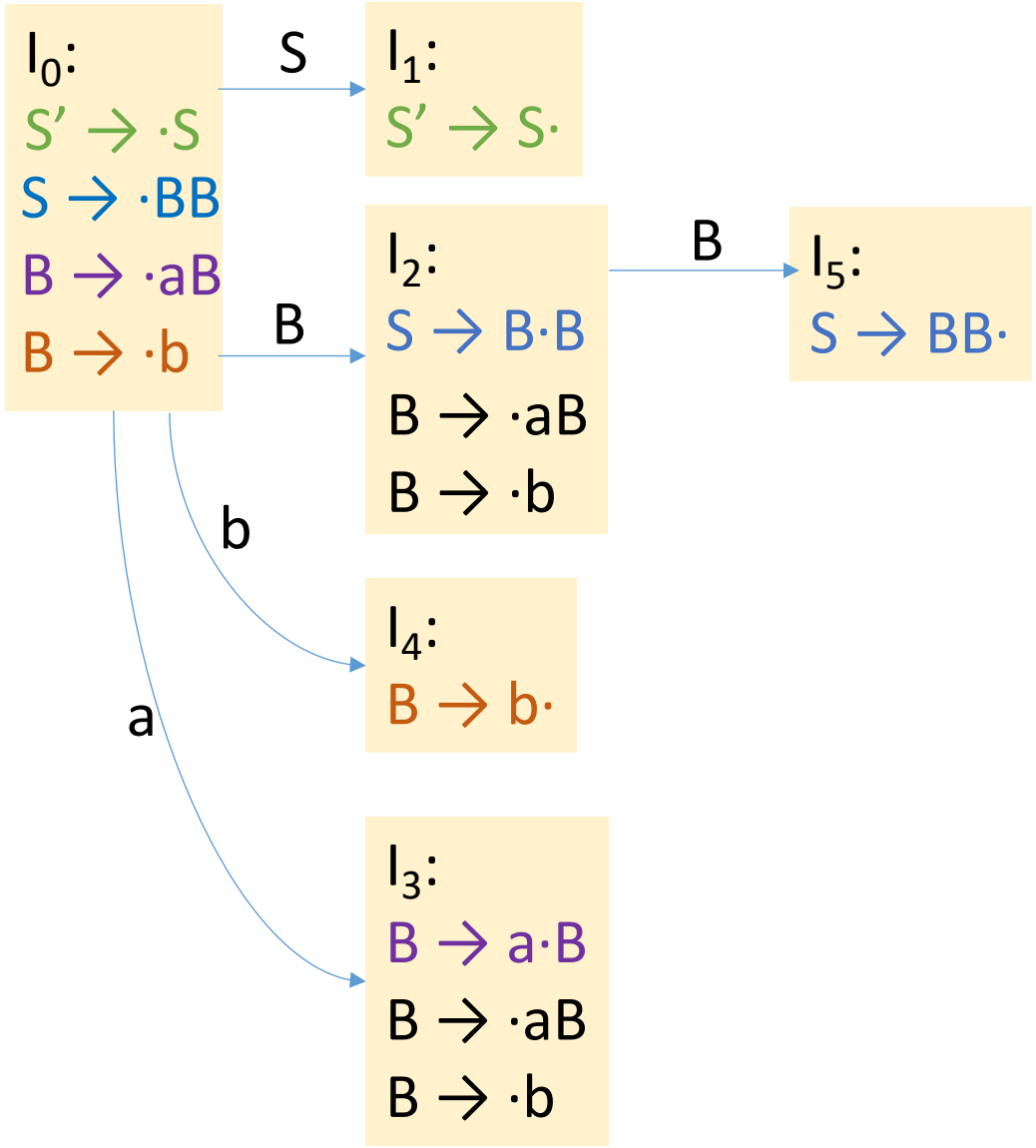
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



Example

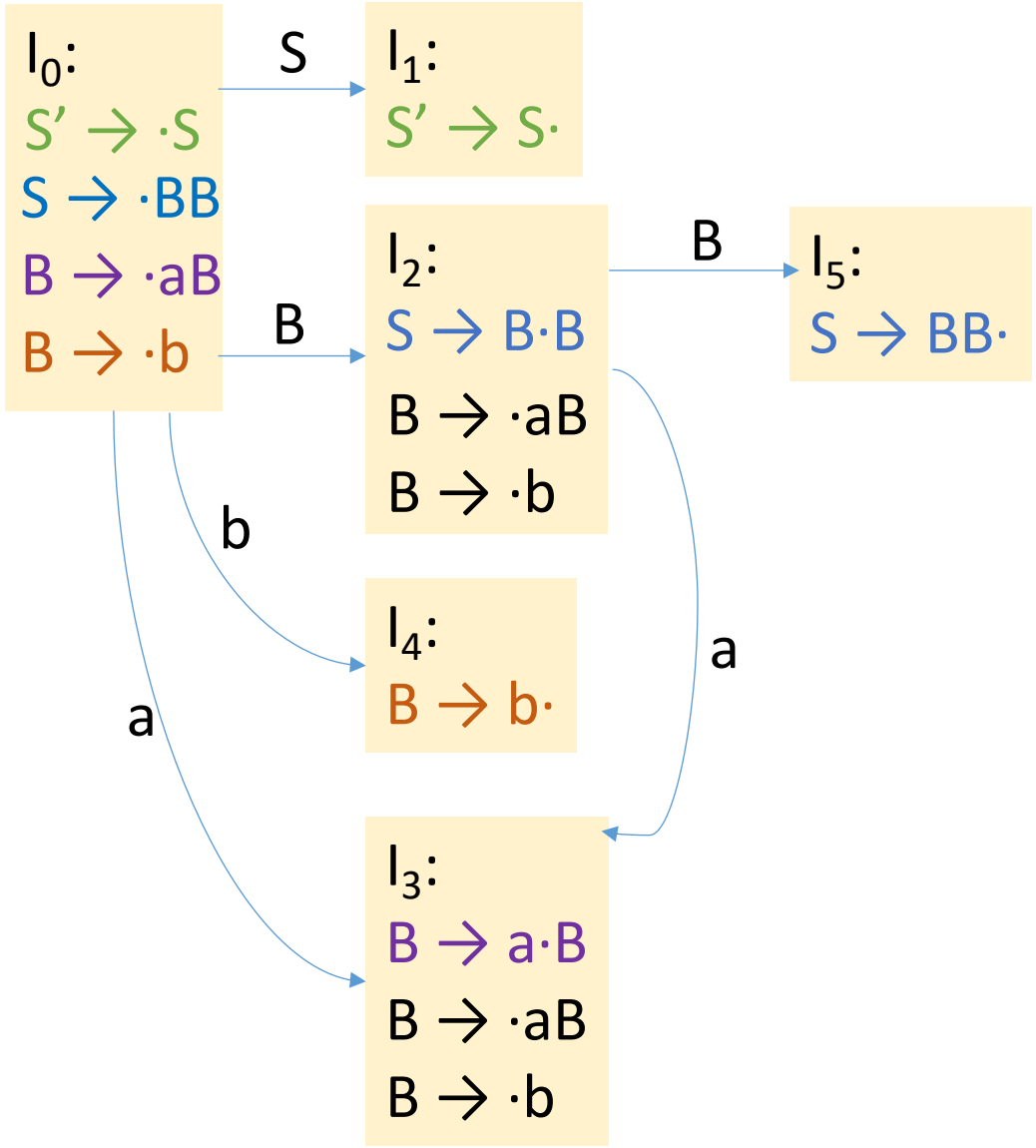
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



Example

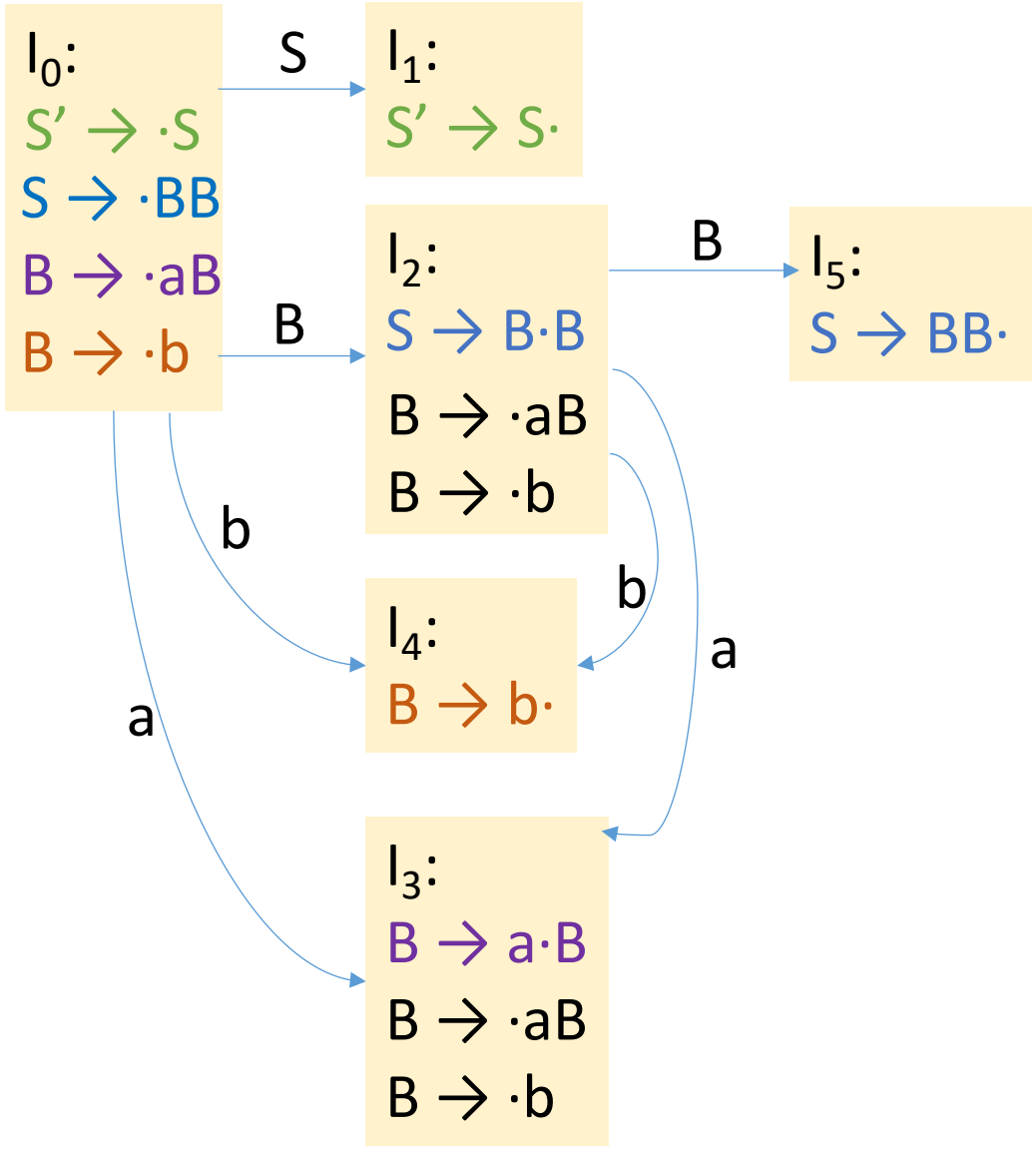
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



Example

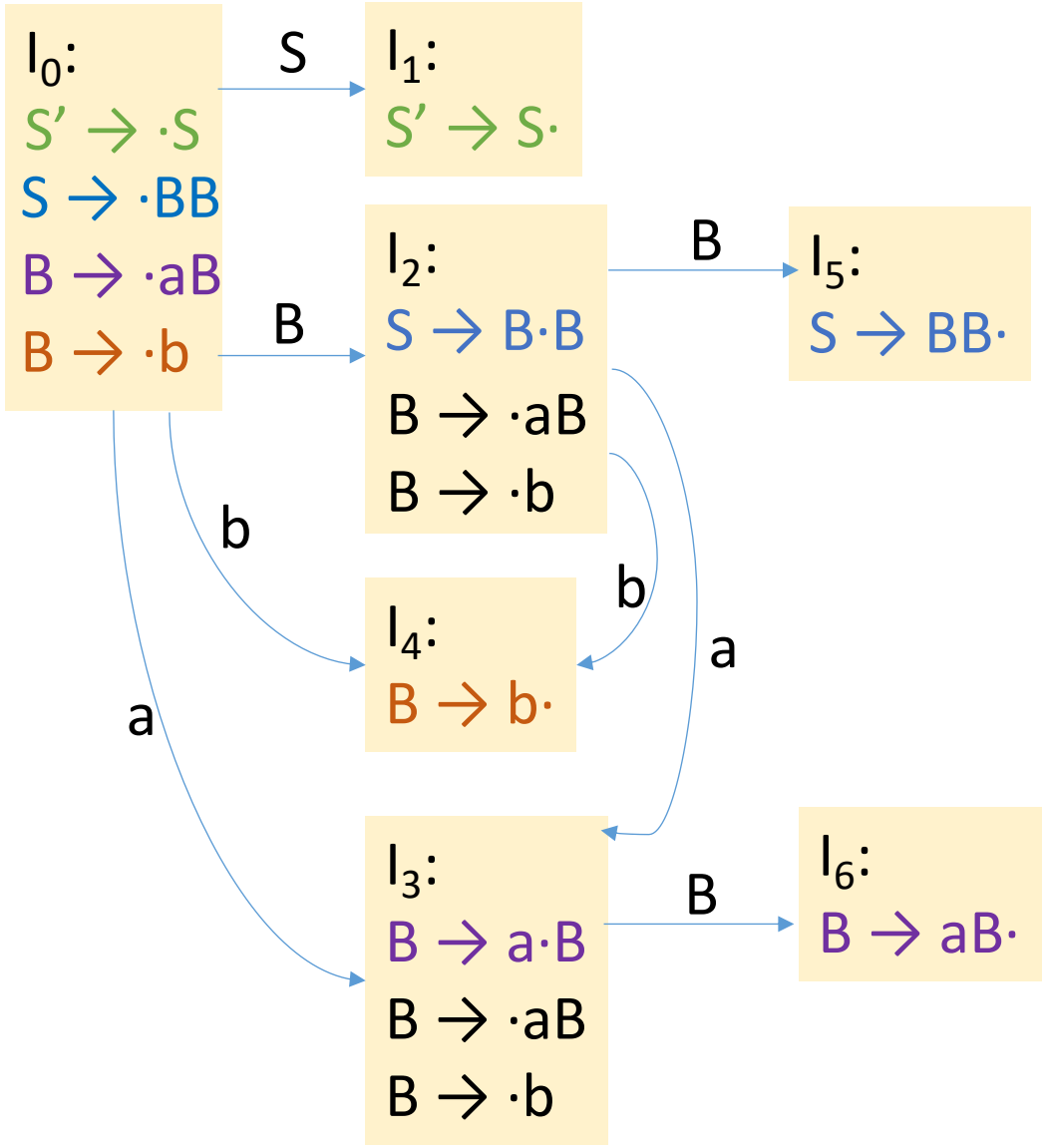
Grammar:

(0) S’ → S

(1) S → BB

(2) B → aB

(3) B → b



Example

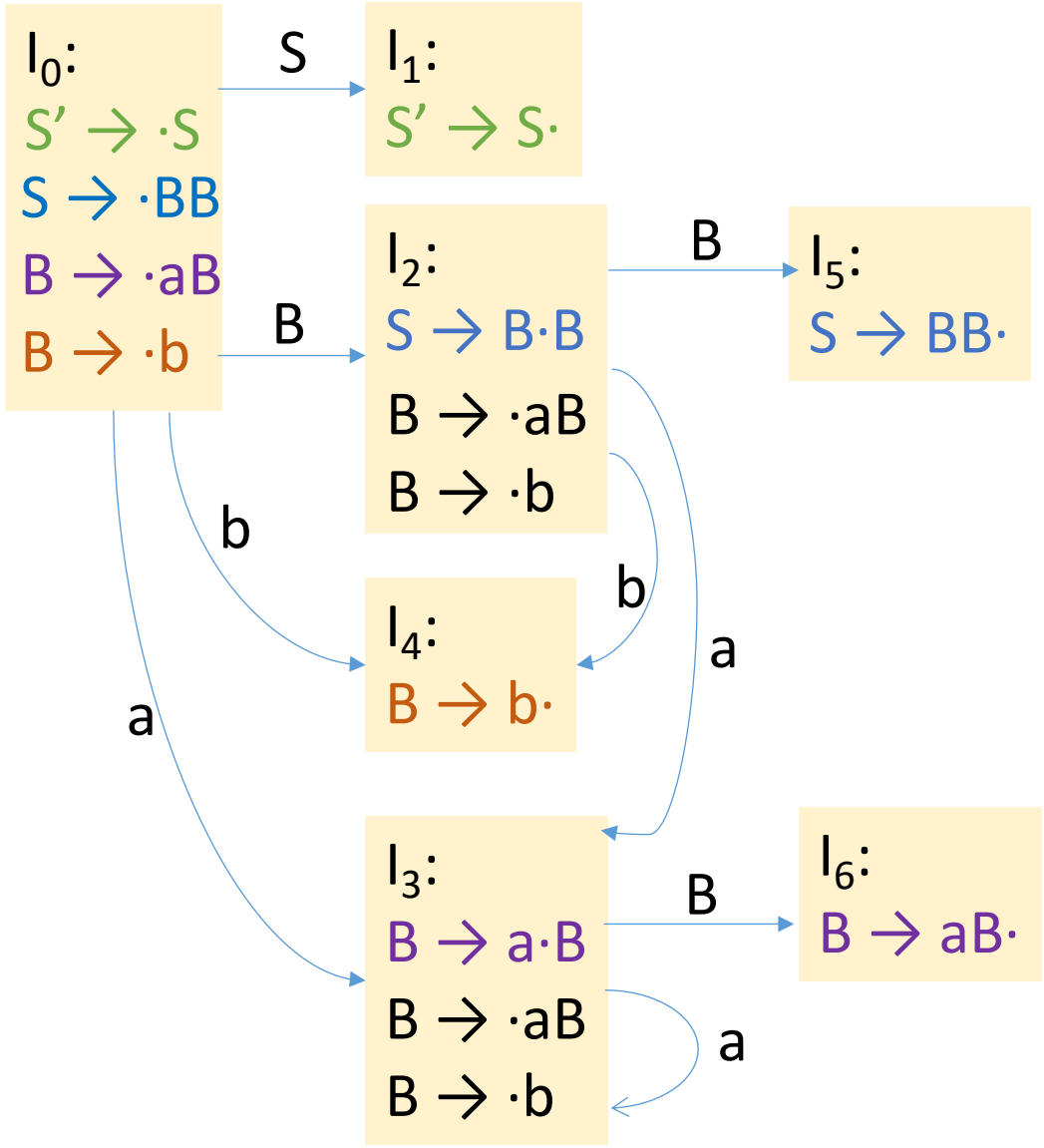
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



Example

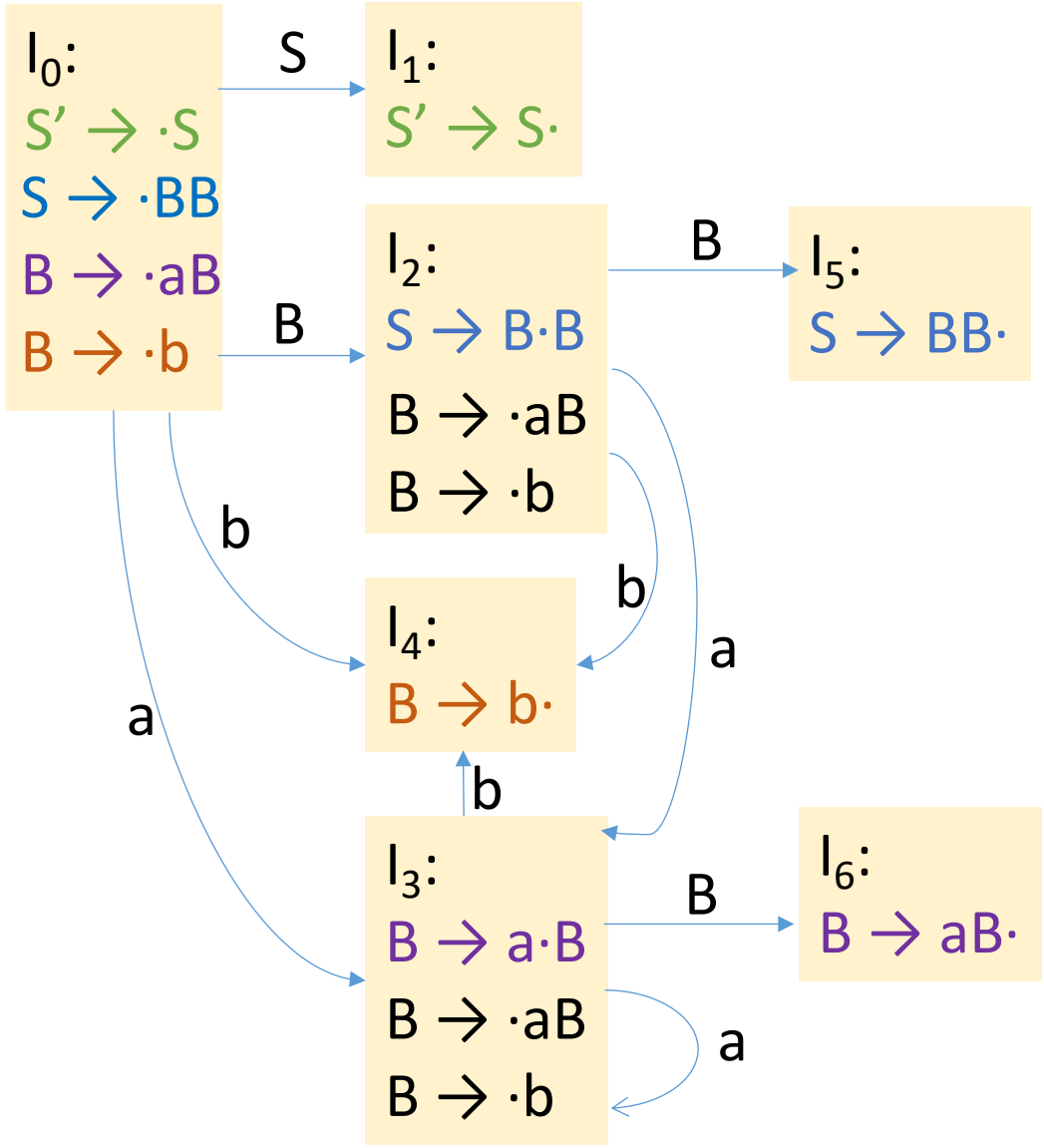
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



Example

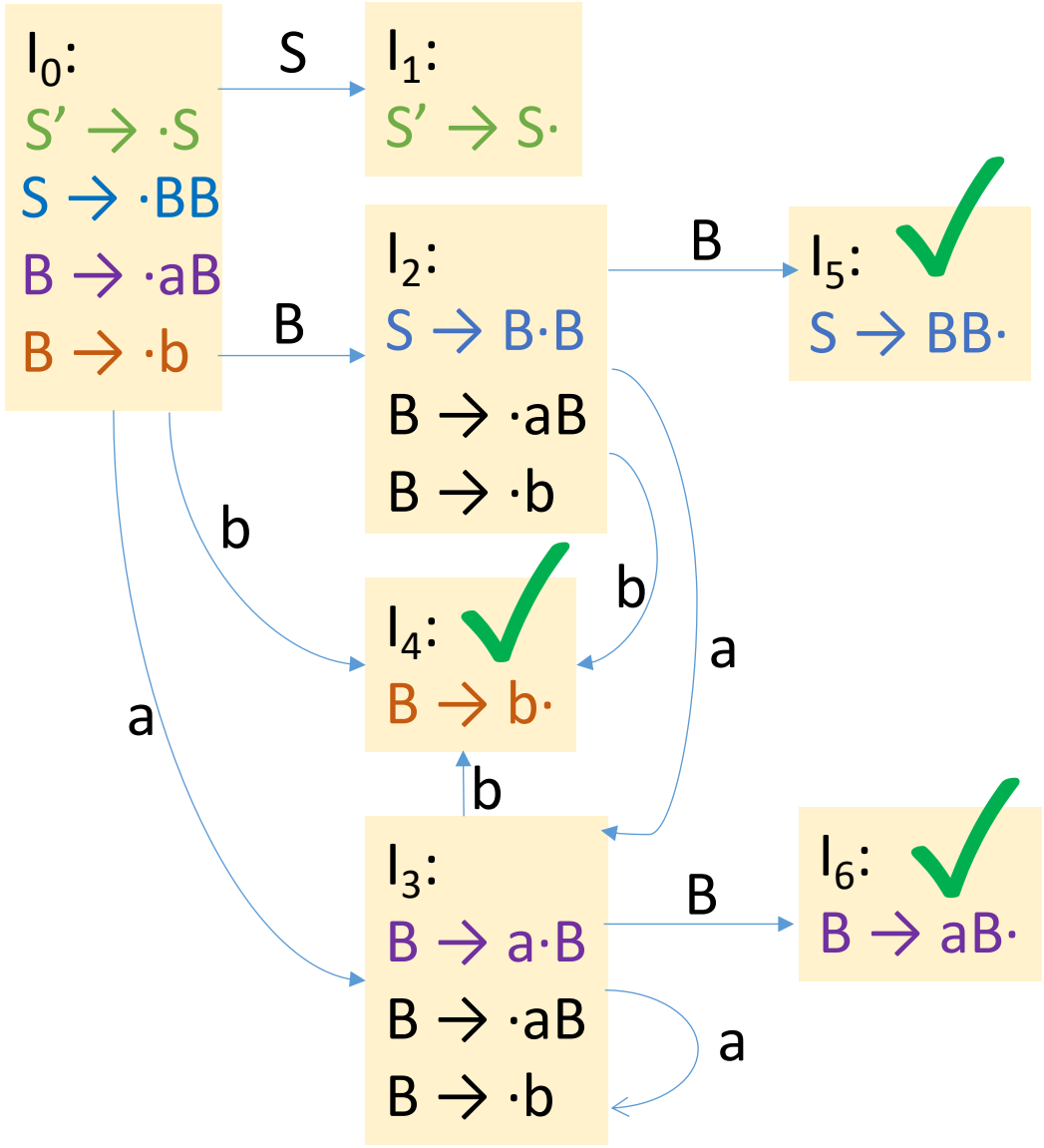
Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b



Example (cont.)

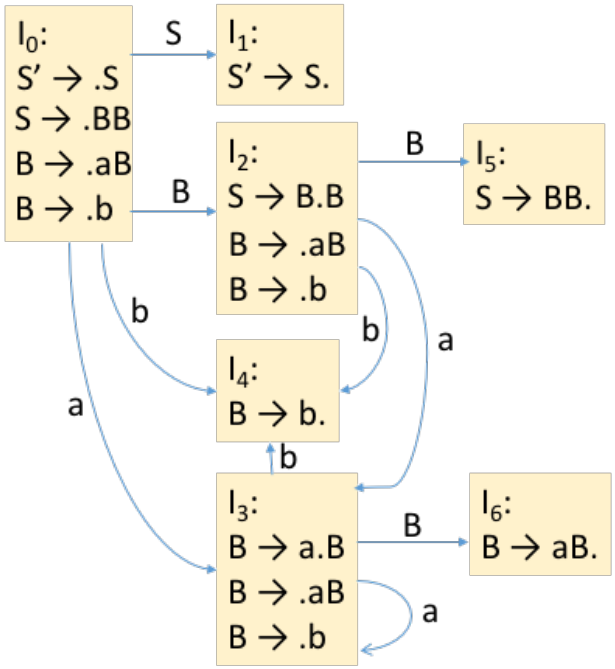
Grammar:

(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.)

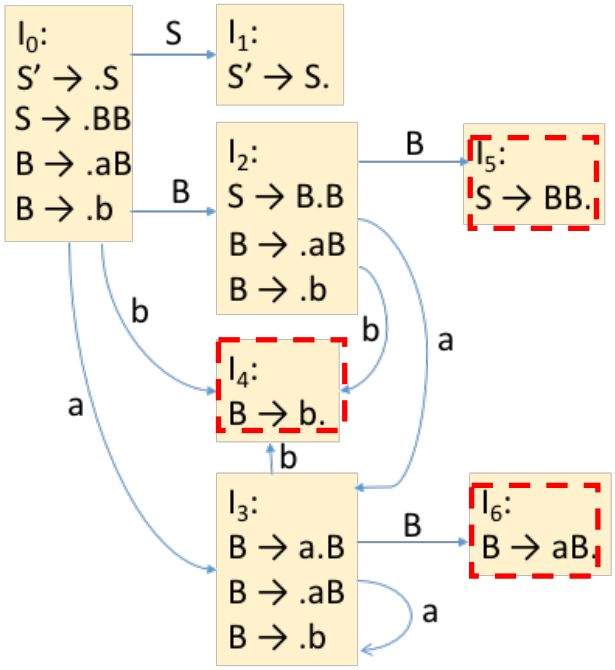
Grammar:

(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.)

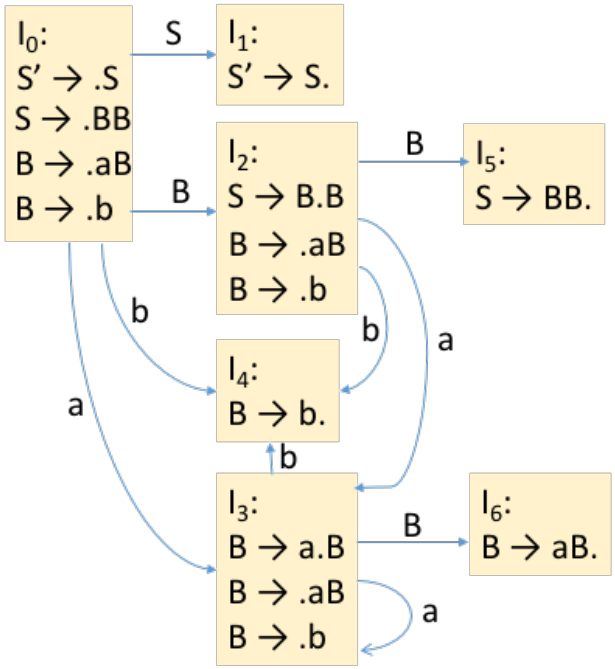
Grammar:

(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.)

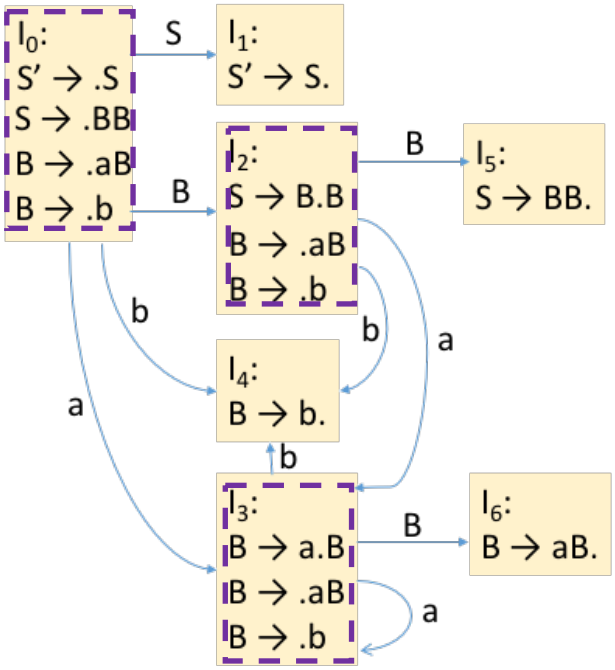
Grammar:

(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.)

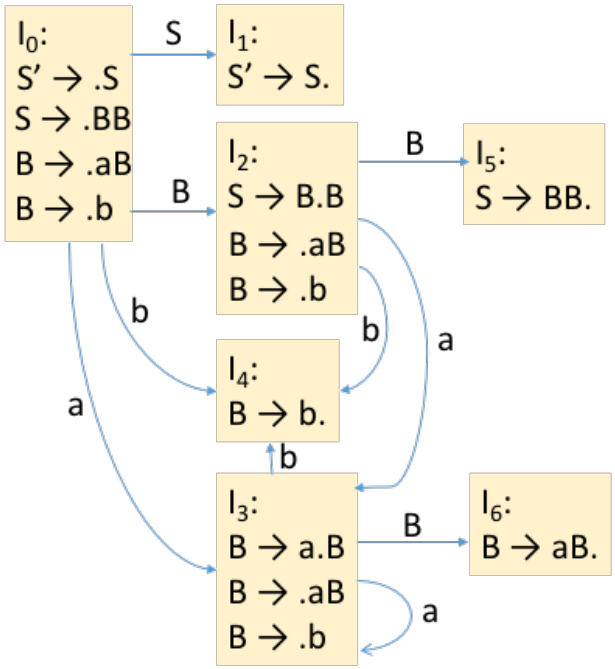
Grammar:

(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.)

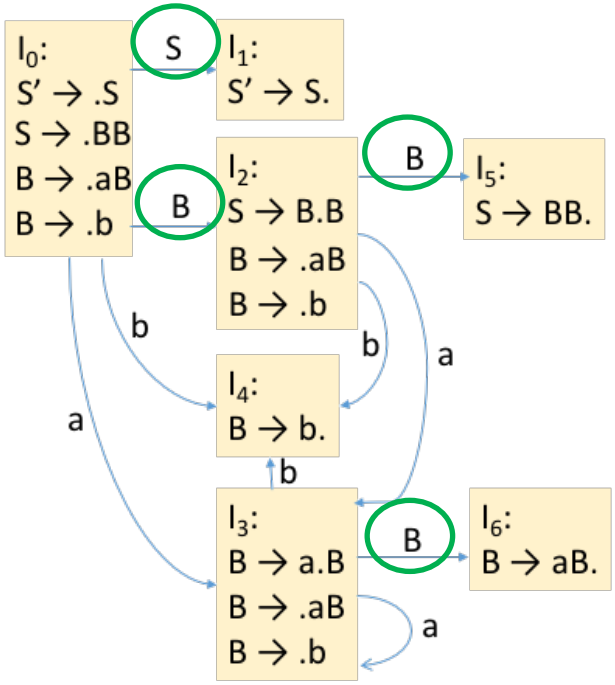
Grammar:

(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 → α·Bβ is in *CLOSURE(I)* and B → γ is a production, then add item B → ·γ to *CLOSURE(I)*, if it is not already there[期待B]
    - Apply this rule until no more new items can be added to *CLOSURE(I)*

Grammar:

(0) S' → S

(1) S → BB

(2) B → aB

(3) B → b

S' → ·S

S' → ·S

S → ·BB

B → ·aB

B → ·b

goto()[跳转]

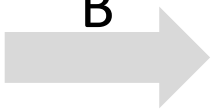
- goto(*I*, *X*): returns state (i.e., set of items) that can be reached by advancing X[即：归约到X]
  - Where *I* is a set of items and *X* is a grammar symbol
  - The closure of the set of all items [A → αX·β] such that [A → α·Xβ] is in *I*[即：识别了/归约到X后的item再闭包]
  - Used to define the transitions in the LR(0) automaton[定义了状态间的转换]
    - 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



I_2:

S → B·B
B → ·aB
B → ·b

Construct LR(0) States

- [增广文法]Create augmented grammar G' for G
  - Given G: S → α | β, create G': S' → S S → α | β
  - Creates a single rule S' → S that when reduced, signals acceptance
- [初始状态]Create 1st state by performing a closure on init item S'→ ·S
  - Closure(I): creates state from an initial set of items *I*
  - Closure({S'→ ·S}) = {S'→ ·S, S → ·α, S → ·β}
- [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 is single symbol, the following new state will be created:

    Goto({S'→ ·S, S → ·α, S → ·β}, α) =

    Closure({S → α·}) = {S → α·}
- [重复操作]Repeatedly perform gotos until 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**[LR(0)自动机]
    - 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[移入：到新状态]
  - 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., $E \rightarrow \alpha$)
    - $E \rightarrow \alpha$: 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({S' → ·S})

  = {S' → ·S, S → ·BB, B → ·aB, B → ·b}

- Goto(S_0, B) = closure({S → B·B})

  S_2 = {S →B·B, B → ·aB, B → ·b}

- Goto(S_0, a) = closure({B → a·B})

  S_3 = {B →a·B, B → ·aB, B → ·b}

- Goto(S_0, b) = closure({B → b·})

  S_4 = {B → b·}

… … …

The Example

Grammar:

(0) S' → S

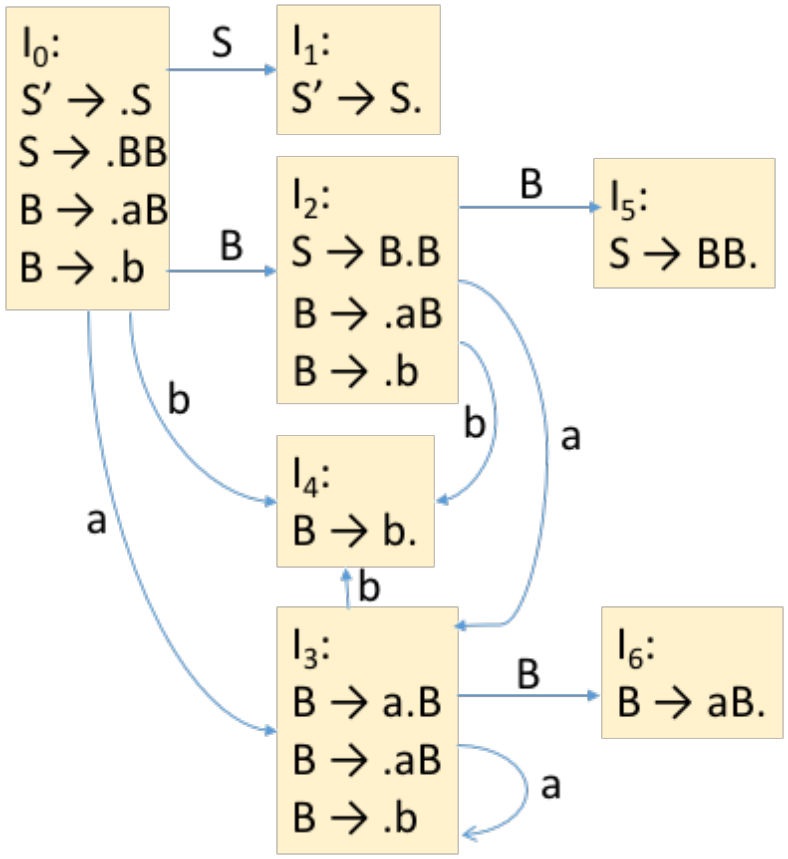
(1) S → BB

(2) B → aB

(3) B → b

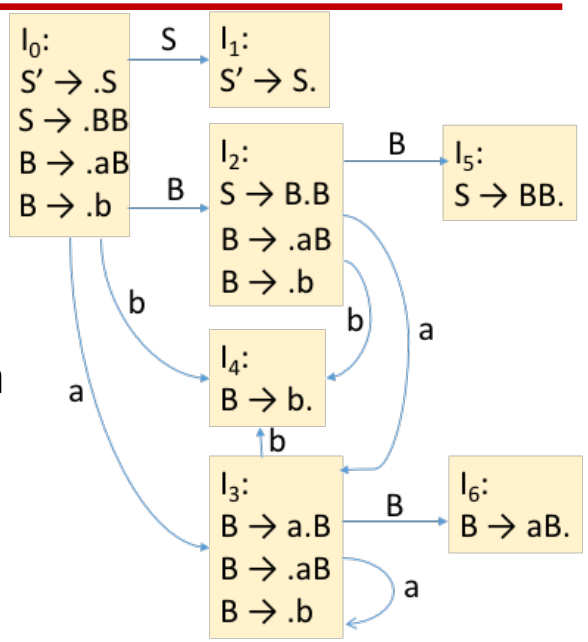
- S_0 = Closure({S' → ·S})
  = {S' → ·S, S → ·BB, B → ·aB, B → ·b}
- Goto(S_0, B) = closure({S → B·B})
  S_2 = {S →B·B, B → ·aB, B → ·b}
- Goto(S_0, a) = closure({B → a·B})
  S_3 = {B →a·B, B → ·aB, B → ·b}
- Goto(S_0, b) = closure({B → b·})
  S_4 = {B → b·}

… … …



Build Parse Table from DFA

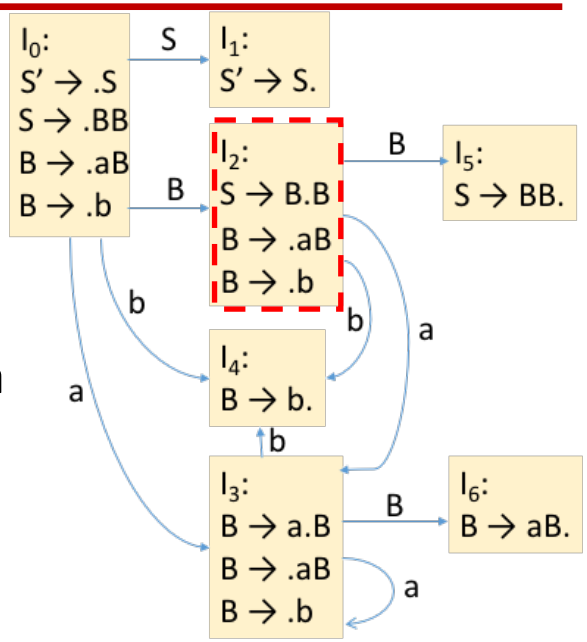
- ACTION: [*state*, *terminal symbol*]
- GOTO: [*state*, *non-terminal symbol*]
- ACTION[动作]
  - If [A→α·aβ] is in S_i and goto(S_i, a) = S_j, where "a" is a terminal, then ACTION[S_i, a] = shift j (*sj*)
  - If [A→α·] is in S_i and A→α is rule numbered j, then ACTION[S_i, a] = reduce j (*rj*)
  - If [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		

Build Parse Table from DFA

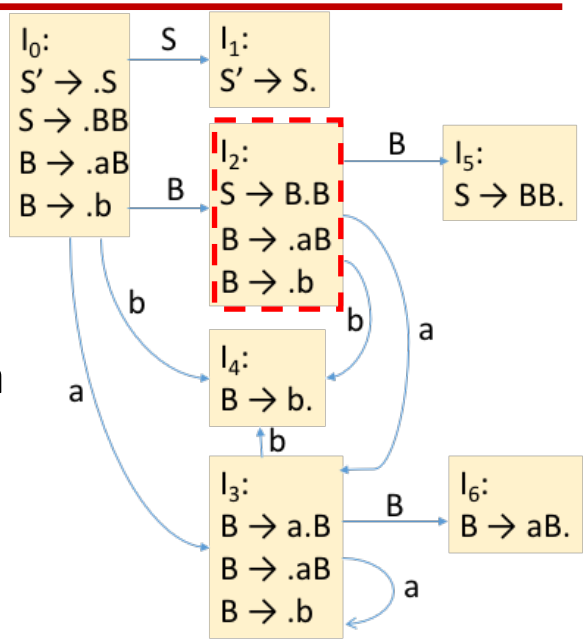
- ACTION: [*state*, *terminal symbol*]
- GOTO: [*state*, *non-terminal symbol*]
- ACTION[动作]
  - If [A→α·aβ] is in S_i and goto(S_i, a) = S_j, where "a" is a terminal, then ACTION[S_i, a] = shift j (*sj*)
  - If [A→α·] is in S_i and A→α is rule numbered j, then ACTION[S_i, a] = reduce j (*rj*)
  - If [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		

Build Parse Table from DFA

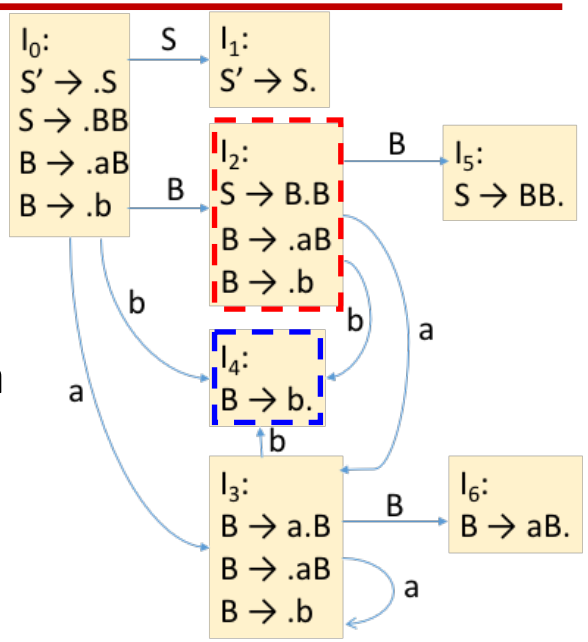
- ACTION: [*state*, *terminal symbol*]
- GOTO: [*state*, *non-terminal symbol*]
- ACTION[动作]
  - If [A→α·aβ] is in S_i and goto(S_i, a) = S_j, where "a" is a terminal, then ACTION[S_i, a] = shift j (*sj*)
  - If [A→α·] is in S_i and A→α is rule numbered j, then ACTION[S_i, a] = reduce j (*rj*)
  - If [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		

Build Parse Table from DFA

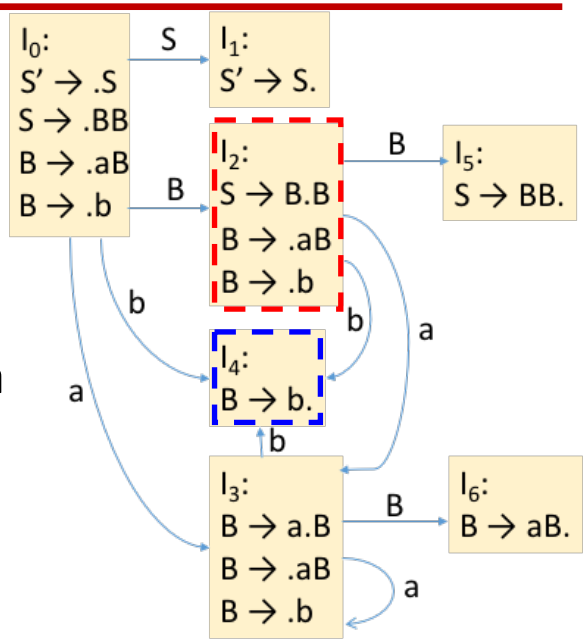
- ACTION: [*state*, *terminal symbol*]
- GOTO: [*state*, *non-terminal symbol*]
- ACTION[动作]
  - If [A→α·aβ] is in S_i and goto(S_i, a) = S_j, where "a" is a terminal, then ACTION[S_i, a] = shift j (*sj*)
  - If [A→α·] is in S_i and A→α is rule numbered j, then ACTION[S_i, a] = reduce j (*rj*)
  - If [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		

Build Parse Table from DFA

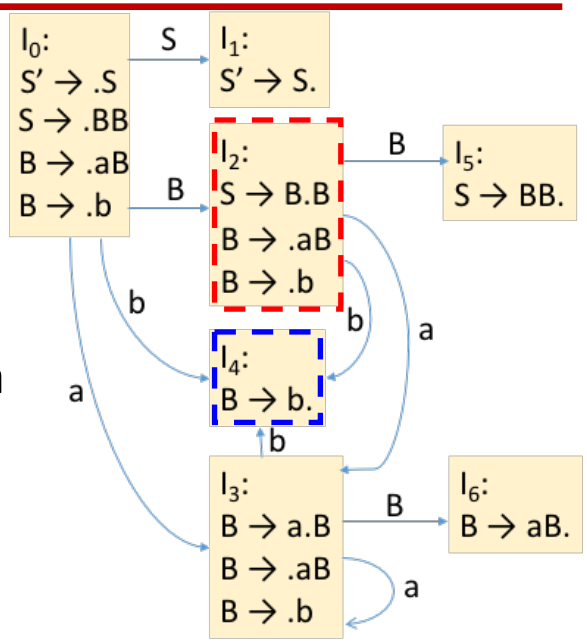
- ACTION: [*state*, *terminal symbol*]
- GOTO: [*state*, *non-terminal symbol*]
- ACTION[动作]
  - If [A→α·aβ] is in S_i and goto(S_i, a) = S_j, where "a" is a terminal, then ACTION[S_i, a] = shift j (*sj*)
  - If [A→α·] is in S_i and A→α is rule numbered j, then ACTION[S_i, a] = reduce j (*rj*)
  - If [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		

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 (*sj*)
  - If [A→α·] is in S_i and A→α is rule numbered j, then ACTION[S_i, a] = reduce j (*rj*)
  - 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		

The Example

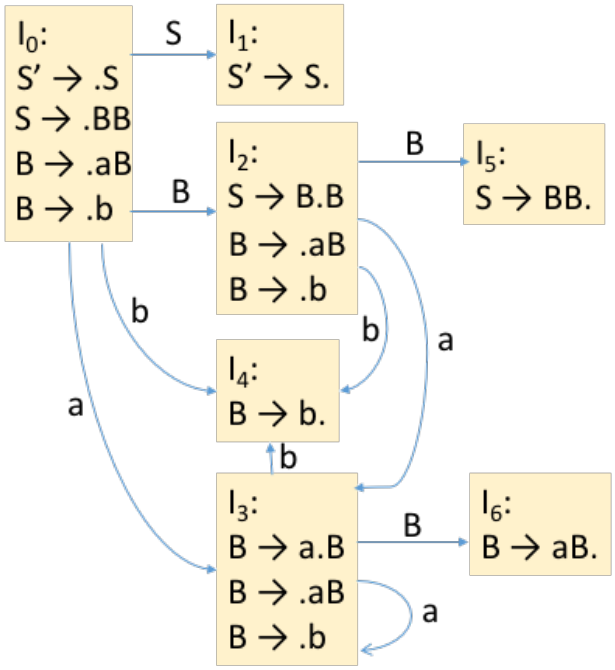
Grammar:

(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

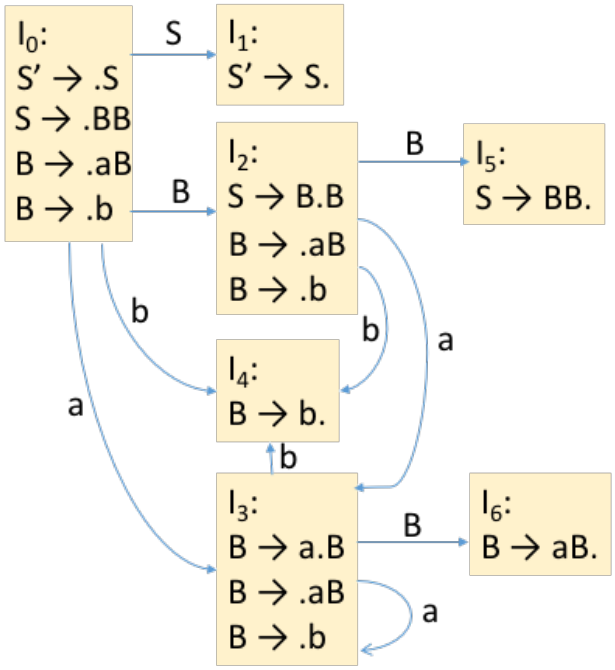
Grammar:

(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

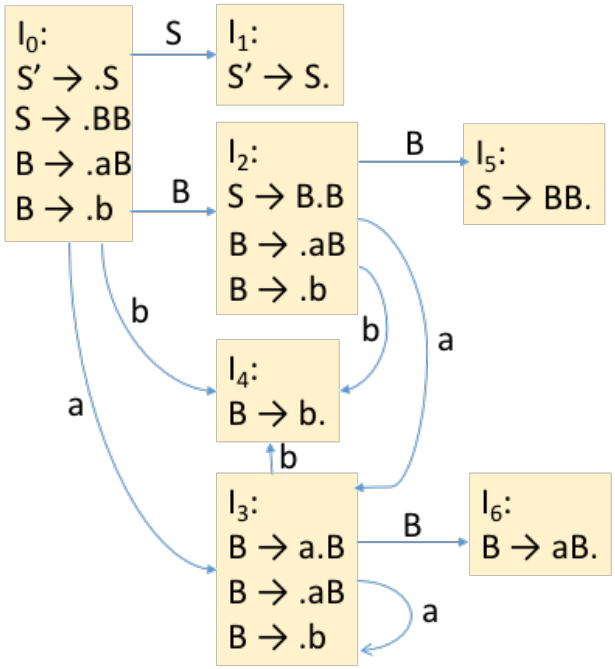
Grammar:

(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

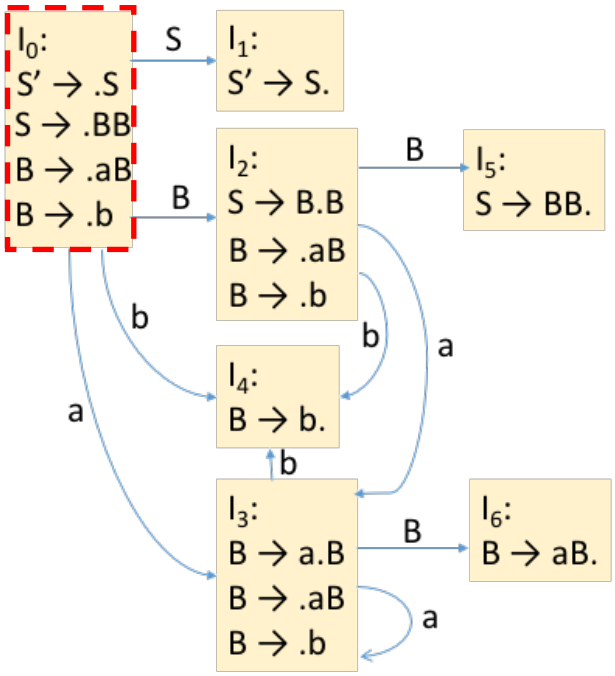
Grammar:

(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

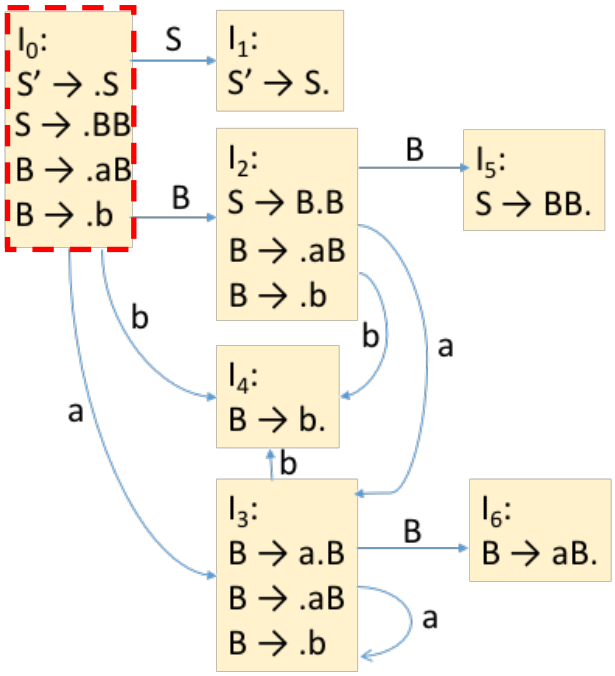
Grammar:

(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$

0

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

The Example

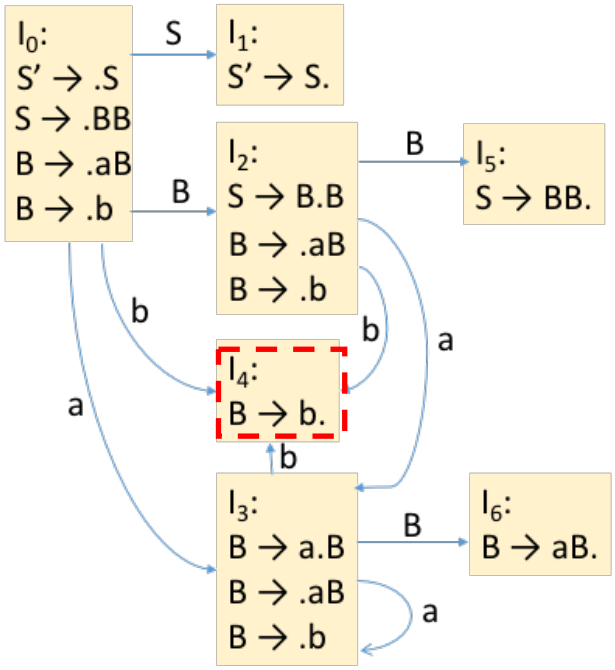
Grammar:

(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$

0

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

The Example

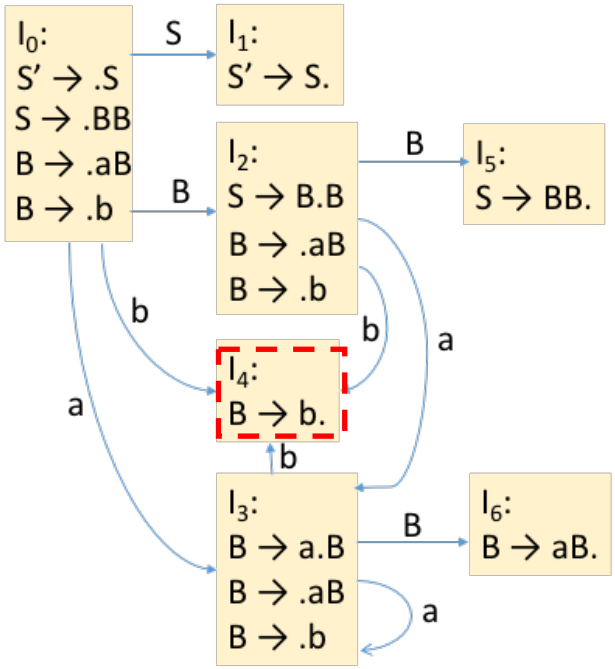
Grammar:

(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$

0 0 4

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

The Example

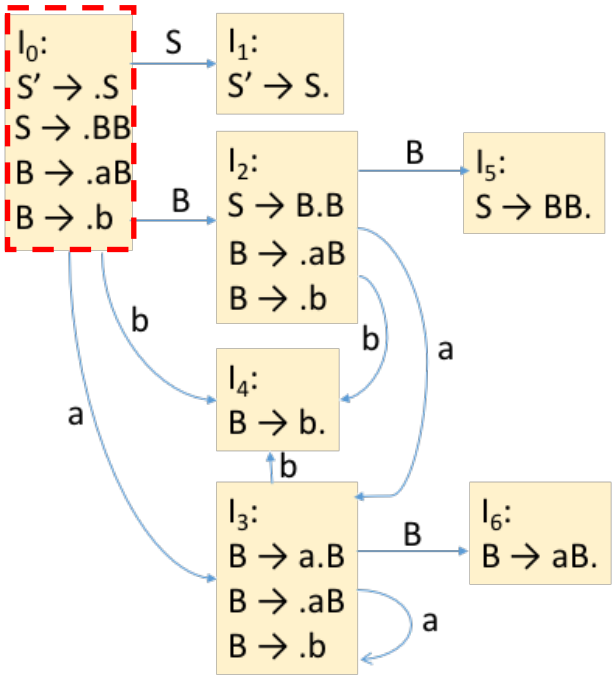
Grammar:

(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$

0 0 4

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

The Example

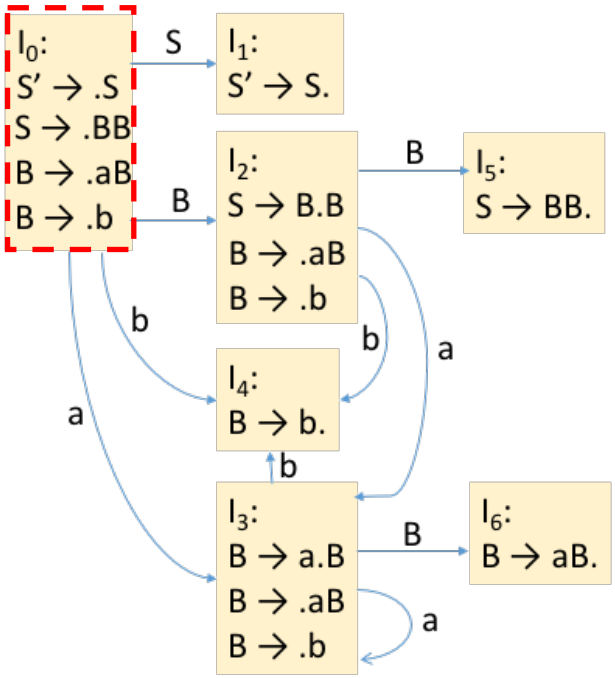
Grammar:

(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$

0 0 4 0

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

The Example

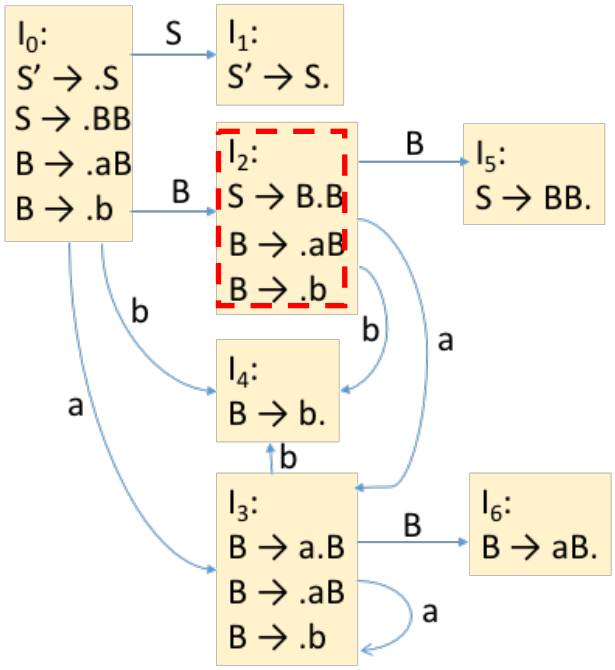
Grammar:

(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$

0 0 4 0

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

The Example

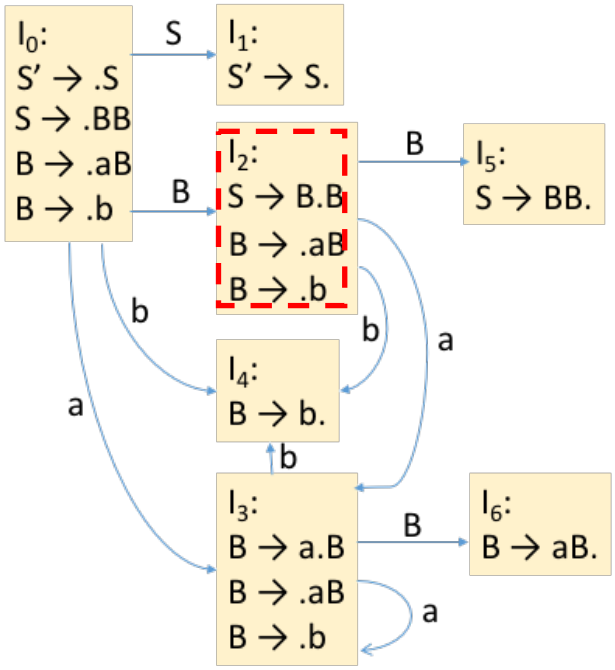
Grammar:

(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$

0 0 4 0 2

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

The Example

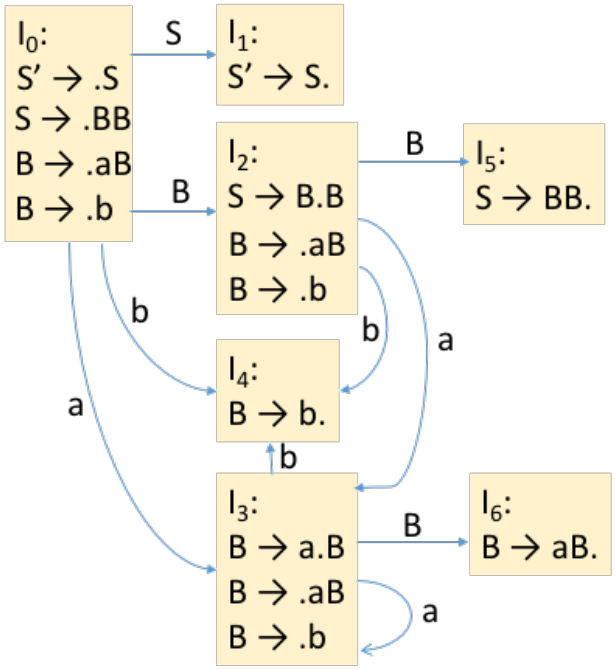
Grammar:

(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$

0 0 4 0 2

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

The Example

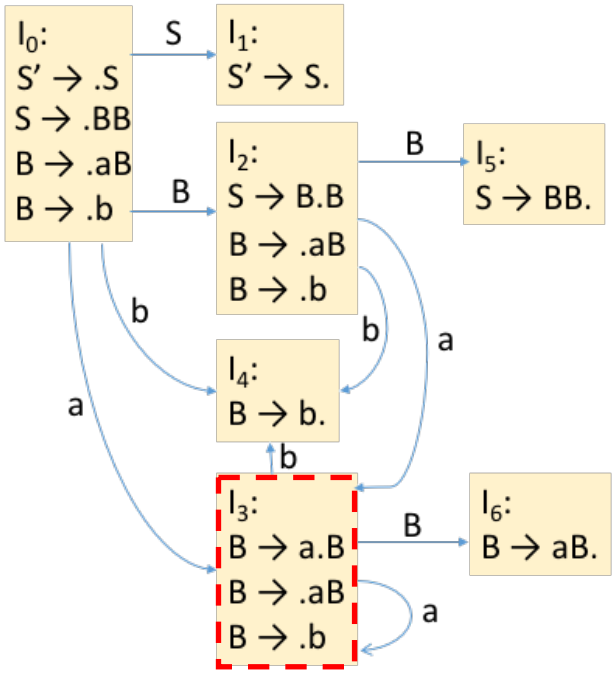
Grammar:

(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$

0 0 4 0 2

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

The Example

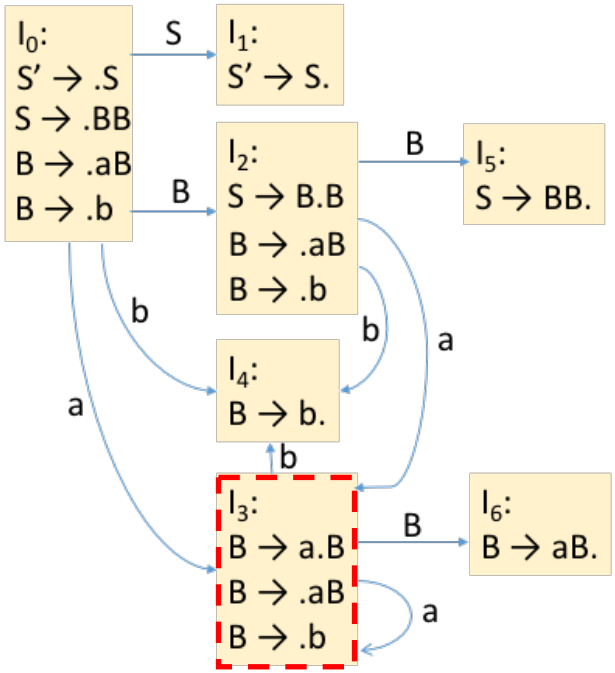
Grammar:

(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$

0 0 4 0 2 0 2 3

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

The Example

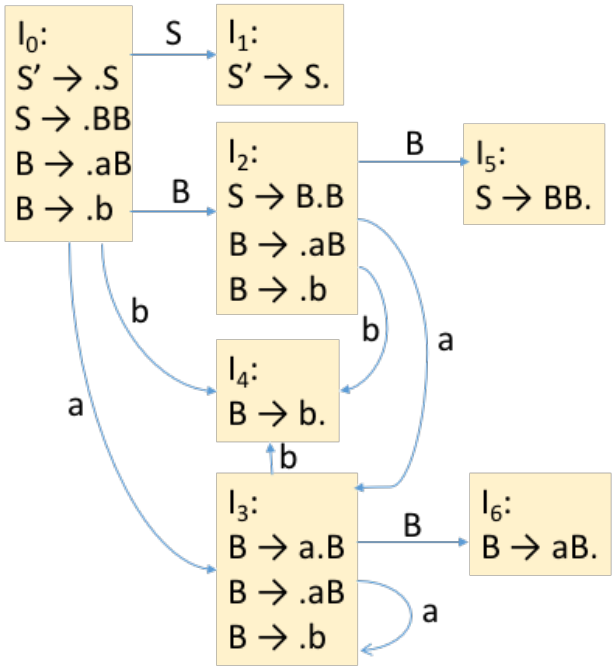
Grammar:

(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$

0 0 4 0 2 0 2 3

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

The Example

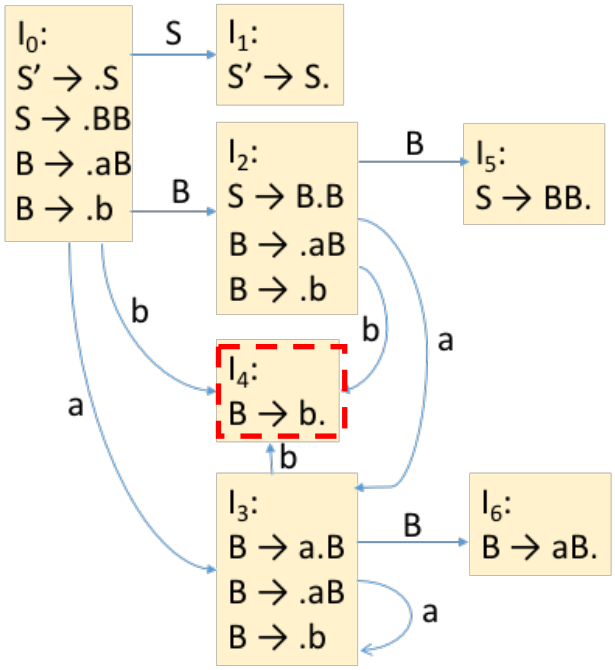
Grammar:

(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$

0 0 4 0 2 0 2 3

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

The Example

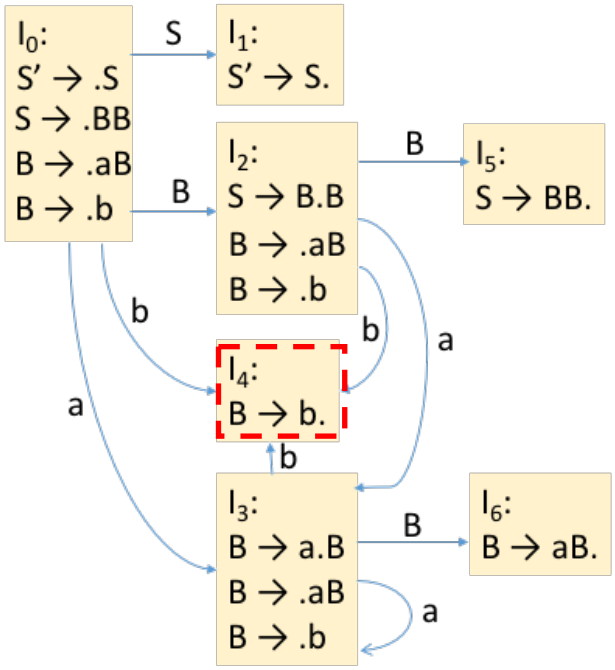
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4

The Example

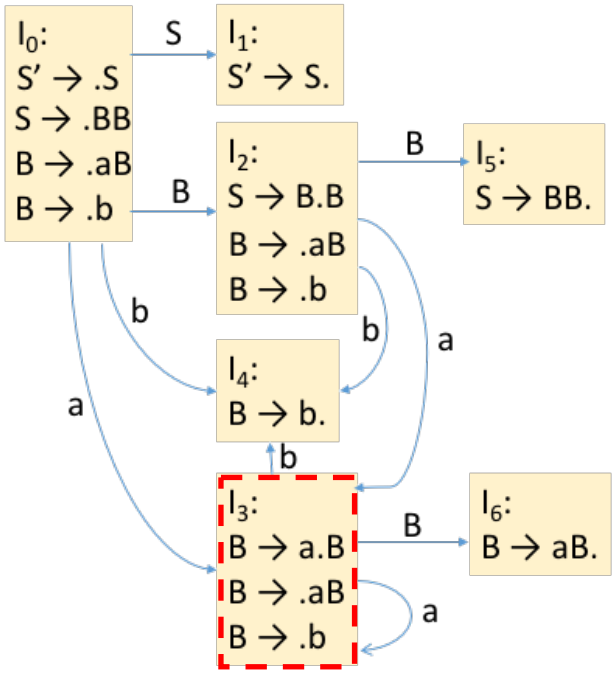
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4

The Example

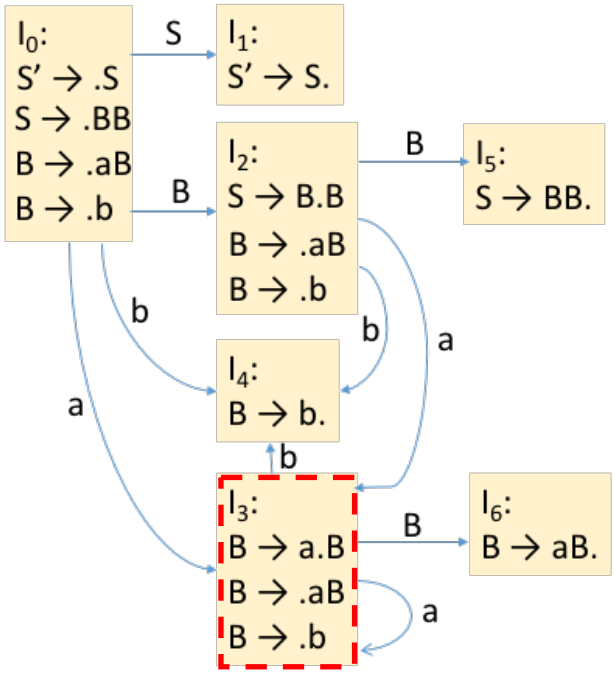
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4 0 2 3

The Example

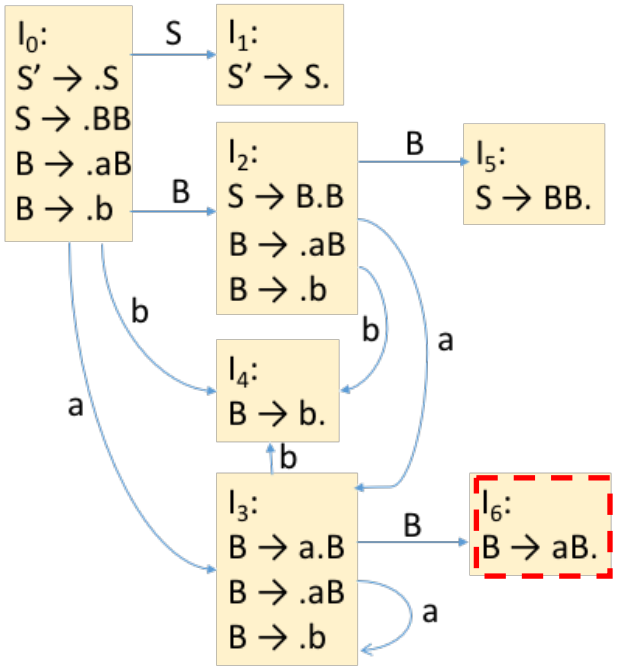
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4 0 2 3

The Example

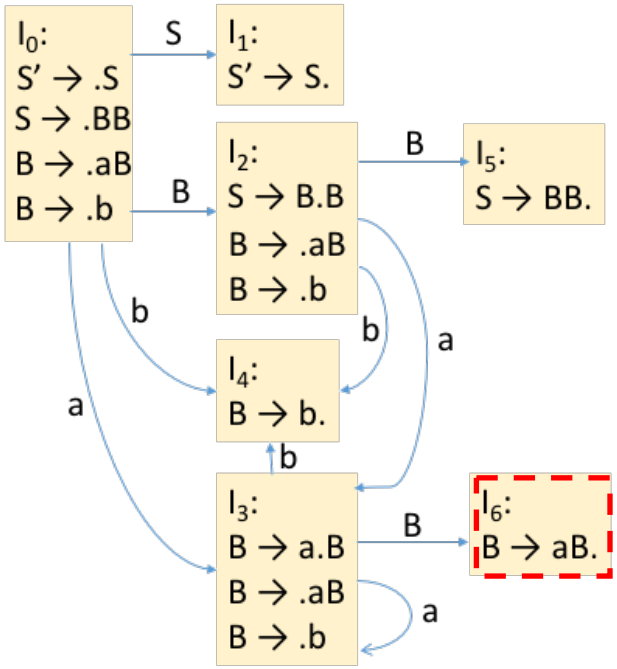
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4 0 2 3 6

The Example

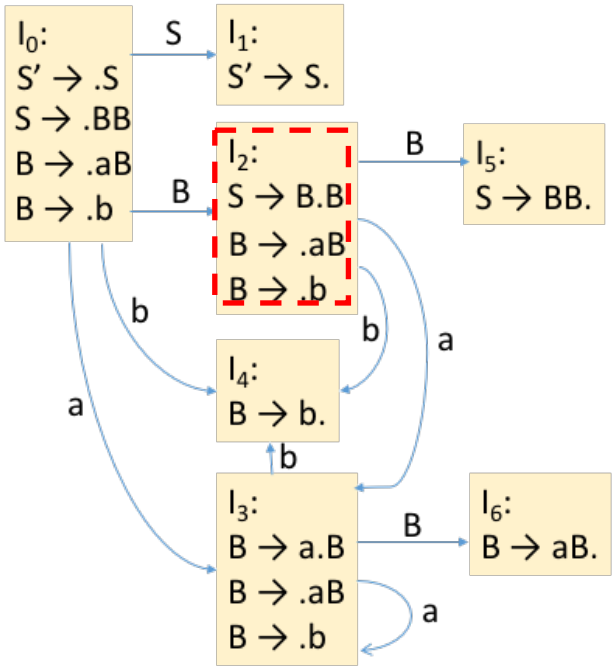
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4 0 2 3 6

The Example

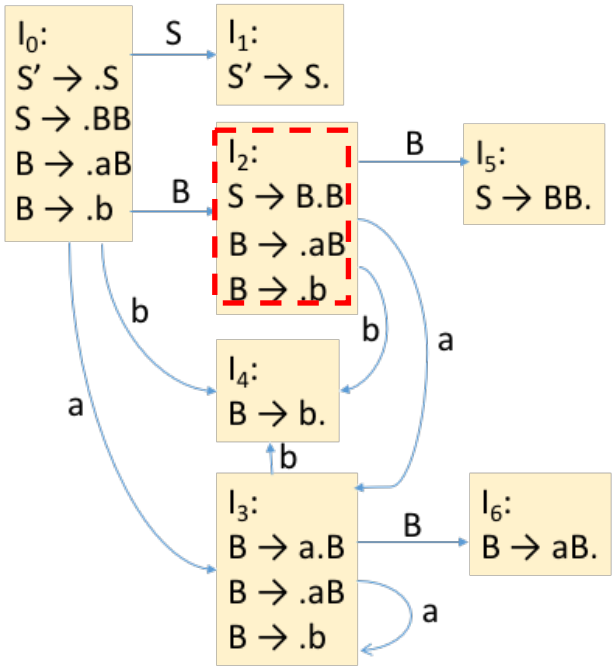
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4 0 2 3 6 0 2

The Example

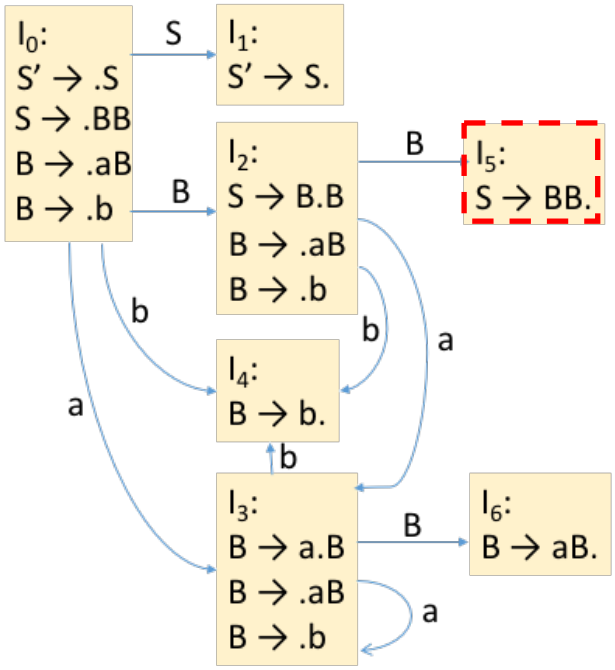
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4 0 2 3 6 0 2

The Example

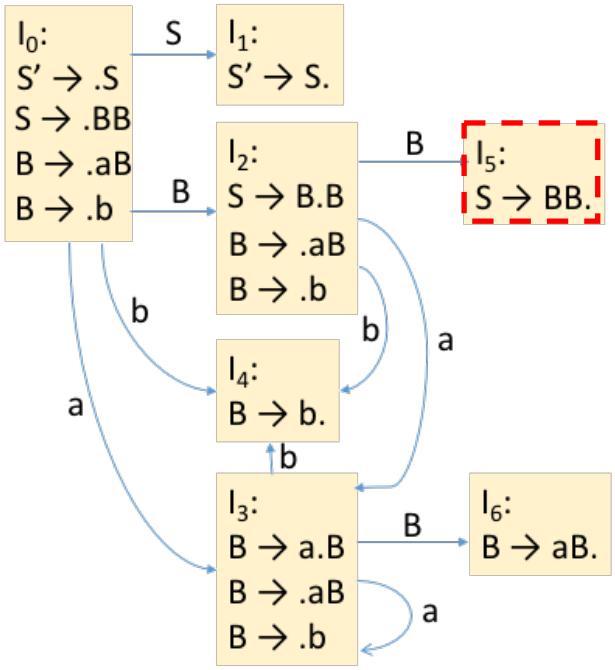
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4 0 2 3 6 0 2 5

The Example

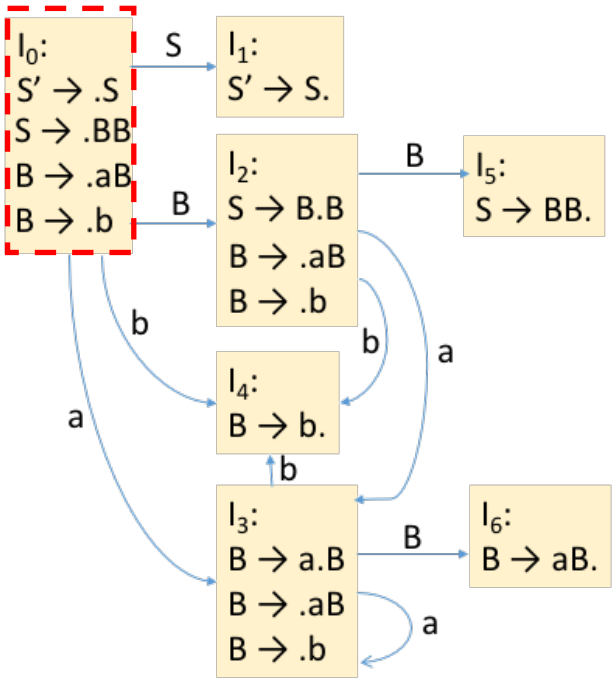
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4 0 2 3 6 0 2 5

The Example

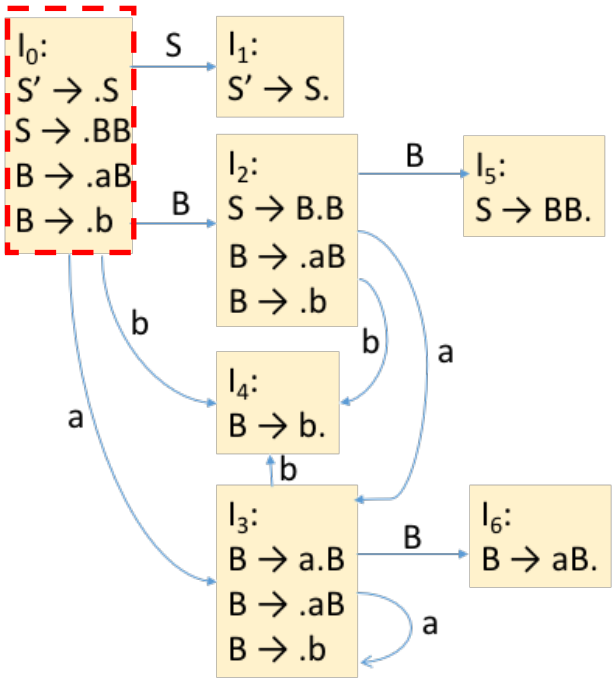
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4 0 2 3 6 0 2 5 0

The Example

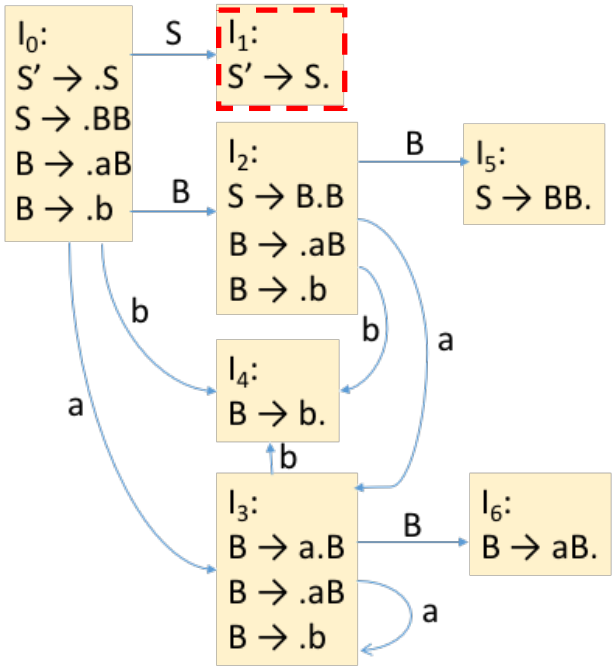
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4 0 2 3 6 0 2 5 0

The Example

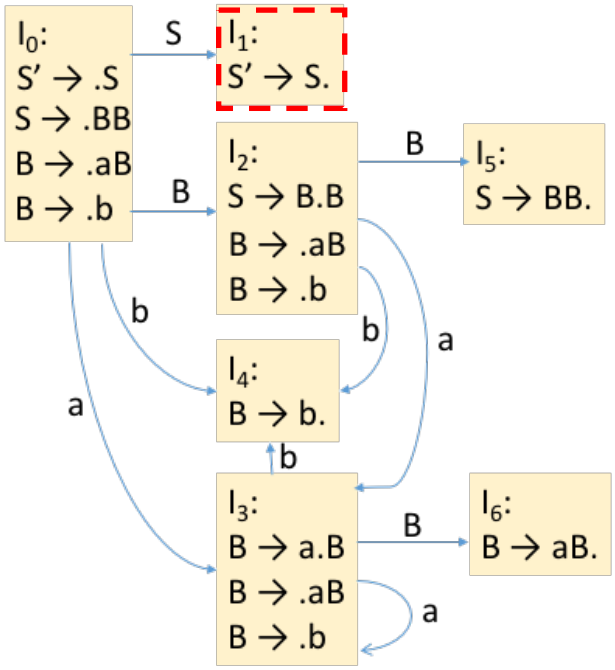
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4 0 2 3 6 0 2 5 0 1

The Example

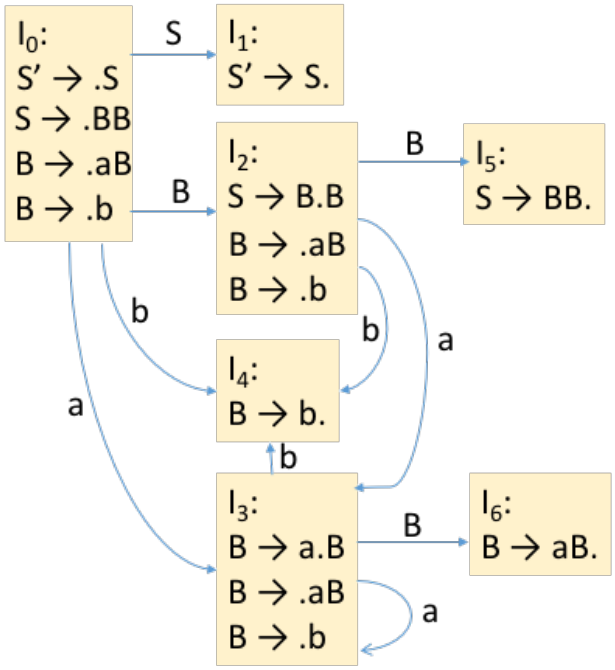
Grammar:

(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$

0 0 4 0 2 0 2 3

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

0 2 3 4 0 2 3 6 0 2 5 0 1

The Example (cont.)

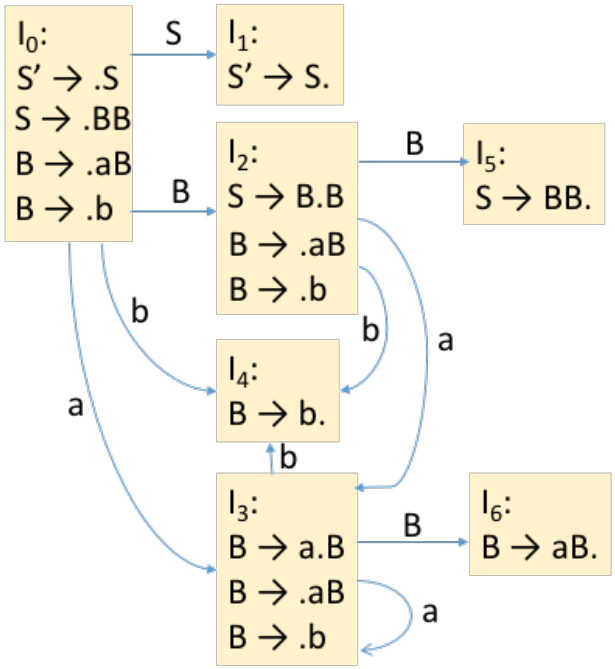
Grammar:

(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		

String: bab

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

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

The Example (cont.)

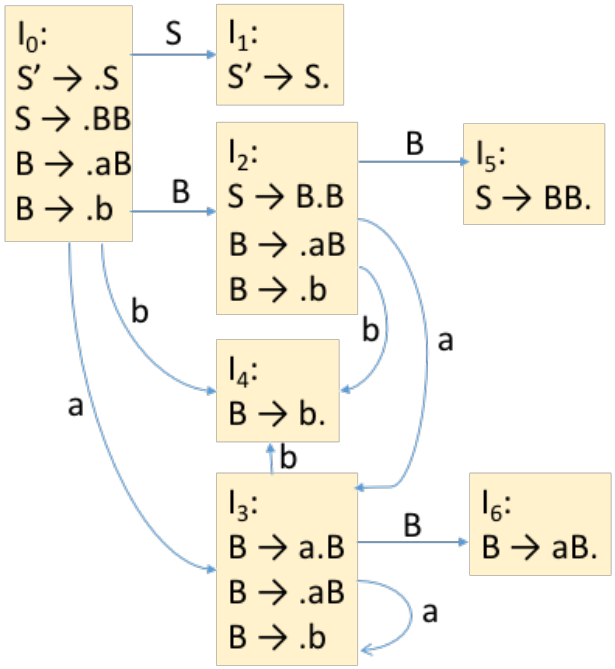
Grammar:

(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		

☆ 是LR(0)，没有任何lookahead ☆

- state直接决定了是shift/reduce，并不需要看输入符号
- 若reduce，输入符号及整个input buffer没有任何变化
- 若shift，输入符号从input buffer移入stack

LR(0) Parsing

- Construct LR(0) automaton from the Grammar[由文法构建自动机]
- Idea: assume
  - Input buffer contains α[但buffer不止有α]
  - Next input is *t*[α后是t]
  - DFA on input α terminates in state s
    - α处理完毕后处于状态s
- Next: **reduce** by X → β if[归约]
  - s contains item X → β·
- Or, **shift** if[移进]
  - s contains item X → β·*t*ω
  - Equivalent to saying s has a transition labeled *t*