CS143: Parsing IV

David L. Dill

Stanford University

Parsing

- LL(1) Parsing and Derivations
- Bottom-up Parsing
- Shift-reduce Parsing
- LR(0) Machine
- SLR(1) Parsing

LL(1) Parsing and Derivations

	n	+	*	(	)	$
E	E→TA			E→TA		
T	T→FB			T→FB		
F	F→n			F→(E)		
A		A→+TA			A→ε	A→ε
B		B→ε	B→*FB		B→ε	B→ε

Input	stack
n+n*n$	E$
n+n*n$	TA$
n+n*n$	FBA$
n+n*n$	nBA$
+n*n$	BA$
+n*n$	A$
+n*n$	+TA$
n*n$	TA$

	n	+	*	(	)	$
E	E→TA			E→TA		
T	T→FB			T→FB		
F	F→n			F→(E)		
A		A→+TA			A→ε	A→ε
B		B→ε	B→*FB		B→ε	B→ε

Input	stack
n+n*n$	E$
n+n*n$	TA$
n+n*n$	FBA$
n+n*n$	nBA$
+n*n$	BA$
+n*n$	A$
+n*n$	+TA$
n*n$	TA$

$E \Rightarrow TA \Rightarrow FBA \Rightarrow nBA$

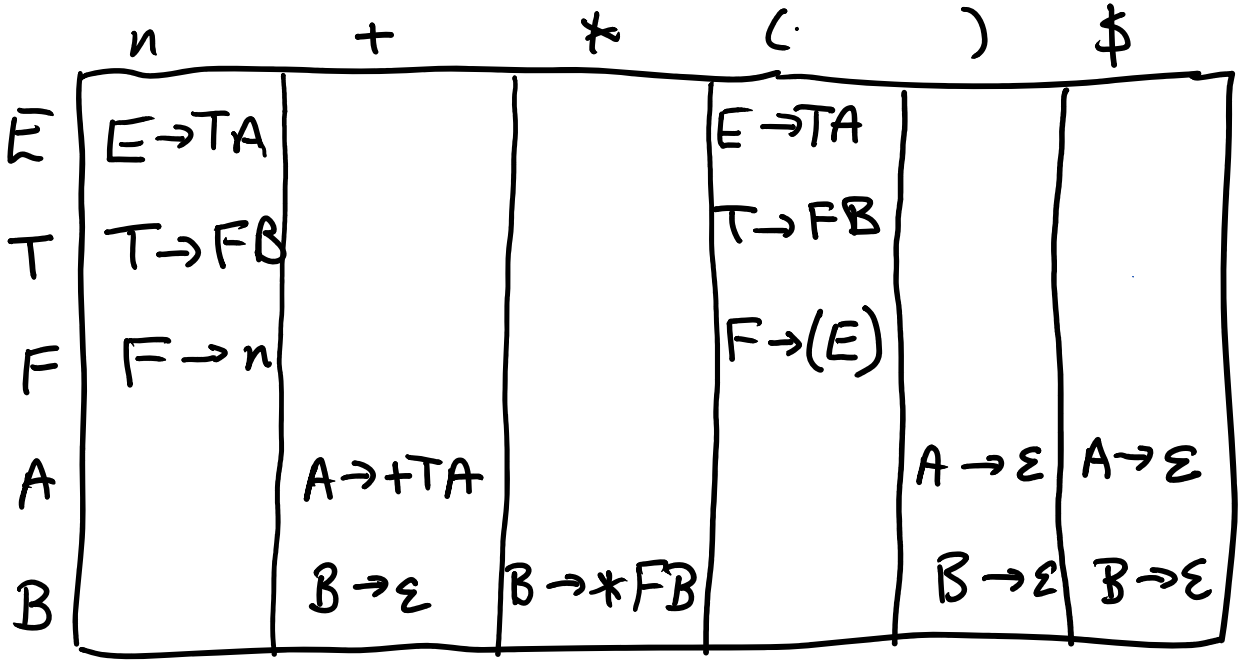
Expand actions become derivation steps.

	n	+	*	(	)	$
E	E→TA			E→TA		
T	T→FB			T→FB		
F	F→n			F→(E)		
A		A→+TA			A→ε	A→ε
B		B→ε	B→*FB		B→ε	B→ε

Input	stack
n+n*n$	E$
n+n*n$	TA$
n+n*n$	FBA$
n+n*n$	nBA$
+n*n$	BA$
+n*n$	A$
+n*n$	+TA$
n*n$	TA$

$E \Rightarrow TA \Rightarrow FBA \Rightarrow nBA \Rightarrow nA \Rightarrow n+TA$

Leftmost nonterminal is top of stack.



Input	stack
n+n*n$	E$
n+n*n$	TA$
n+n*n$	FBA$
n+n*n$	nBA$
+n*n$	BA$
+n*n$	A$
+n*n$	+TA$
n*n$	TA$

$E \Rightarrow TA \Rightarrow FBA \Rightarrow nBA \Rightarrow nA \Rightarrow n+TA$

terminals (matched)

	n	+	*	(	)	$
E	E→TA			E→TA		
T	T→FB			T→FB		
F	F→n			F→(E)		
A		A→+TA			A→ε	A→ε
B		B→ε	B→*FB		B→ε	B→ε

Input	stack
n+n*n$	E$
n+n*n$	TA$
n+n*n$	FBA$
n+n*n$	nBA$
+n*n$	BA$
+n*n$	A$
+n*n$	+TA$
n*n$	TA$

$E \Rightarrow TA \Rightarrow FBA \Rightarrow nBA \Rightarrow nA \Rightarrow n+TA$

top of stack

	n	+	*	(	)	$
E	E→TA			E→TA		
T	T→FB			T→FB		
F	F→n			F→(E)		
A		A→+TA			A→ε	A→ε
B		B→ε	B→*FB		B→ε	B→ε

Input	stack
n+n*n$	E$
n+n*n$	TA$
n+n*n$	FBA$
n+n*n$	nBA$
+n*n$	BA$
+n*n$	A$
+n*n$	+TA$
n*n$	TA$

$E \Rightarrow TA \Rightarrow FBA \Rightarrow nBA \Rightarrow nA \Rightarrow n+TA$

LL(1) → leftmost derivation

other symbols

Bottom-up Parsing

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow n$

$n + n * n$

$E \rightarrow E + T$

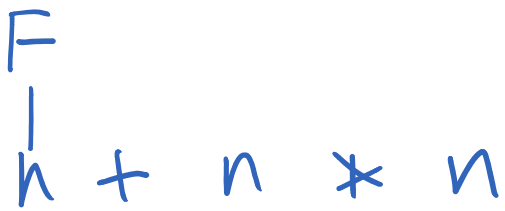
$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow n$



$E \rightarrow E + T$

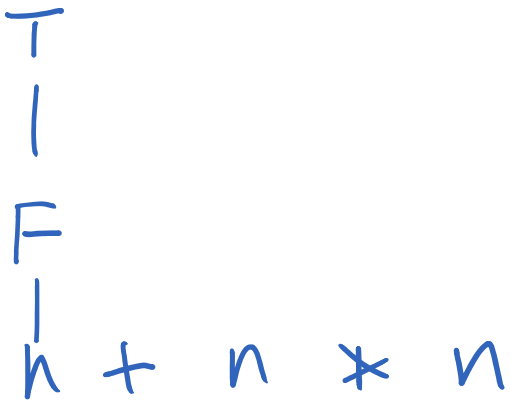
$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow n$



$E \rightarrow E + T$

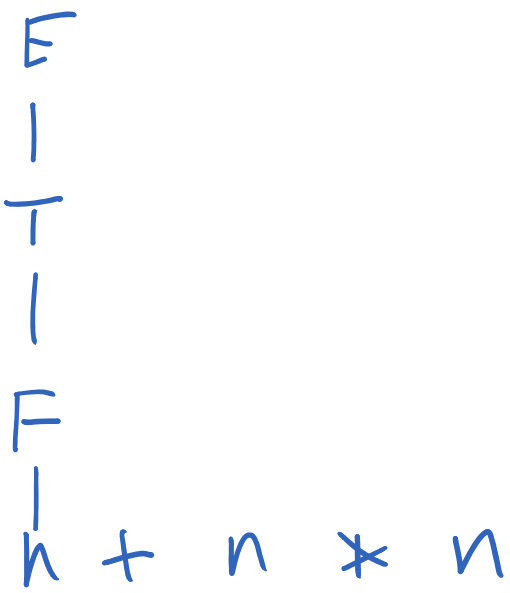
$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow n$



$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow n$

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow n$

$E \rightarrow E + T$

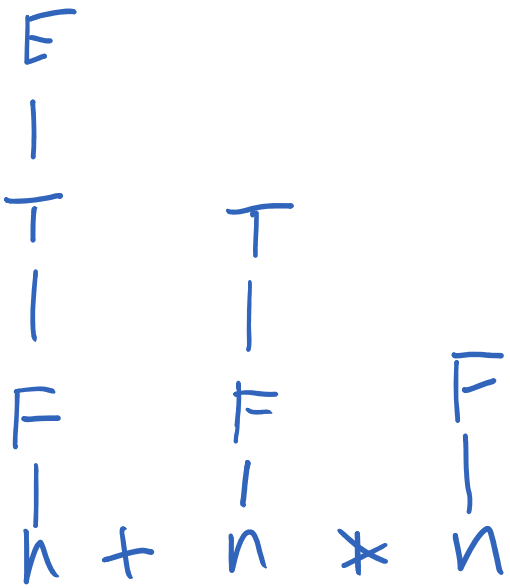
$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow n$



$E \rightarrow E + T$

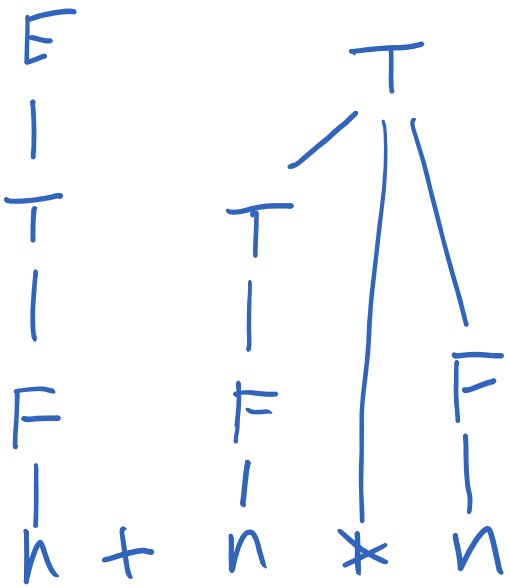
$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow n$



$E \rightarrow E + T$

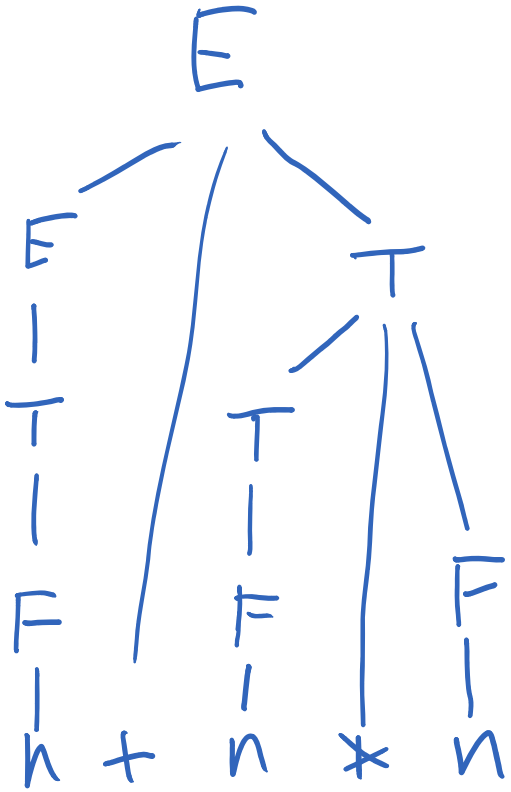
$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow n$



Shift-Reduce Parsing

$S \rightarrow (S) \mid a$

Stack (top)	input	action
$	((a))$	shift
$(	(a))$	shift
$((	a))$	shift
$((a	))$	reduce S → a
$((S	))$	shift
$((S)	)$	reduce S → (S)
$(S	)$	shift
$(S)	$	reduce S → (S)
$S	$	accept

$E \to E + T$

$E \to T$

$T \to T * F$

$T \to F$

$F \to (E)$

$F \to n$

Stack (top)	input	action
$	n+n*n$	shift
$ n	+n*n$	red F→n
$ F	+n*n$	red T→F
$ T	+n*n$	red E→T
$ E	+n*n$	shift
$ E +	n*n$	shift
$ E + n	*n$	red F→n
$ E + F	*n$	red T→F
$ E + T	*n$	shift
$ E + T *	n$	shift
$ E + T * n	$	red F→n
$ E + T * F	$	red T→T*F
$ E + T	$	red E→E+T
$ E	$	accept

Stack (top)

$
$ n
$ F
$ T
$ E
$ E +
$ E + n
$ E + F
$ E + T
$ E + T *
$ E + T * n
$ E + T * F
$ E + T
$ E

Derivation

$E \Rightarrow E + T \Rightarrow E + T * F \Rightarrow E + T * n$

$\Rightarrow E + F * n \Rightarrow E + n * n$

$\Rightarrow T + n * n \Rightarrow F + n * n$

$\Rightarrow n + n * n$

Stack (top)

$
$ n
$ F
$ T
$ E
$ E +
$ E + n
$ E + F
$ E + T
$ E + T *
$ E + T * n
$ E + T * F
$ E + T
$ E

Derivation

$E \Rightarrow E + T \Rightarrow E + T * F \Rightarrow E + T * n$

$\Rightarrow E + F * n \Rightarrow E + n * n$

$\Rightarrow T + n * n \Rightarrow F + n * n$

$\Rightarrow n + n * n$

Series of reductions is a reverse right-most derivation.

Announcements

- WA1 due today (after automatic extension).
- PA2
  - LALR(1) parsing – haven't gotten there yet.
  - **Need to convert EBNF to ordinary CFG.**
  - **No need to eliminate left recursion, left factor the grammar**.
  - **Disambiguate using precedence declaration, not by changing grammar.**

LR(0) Machine

Problem: Recognize possible reduction on stack ("handle")

Solution: Use a DFA to find it.

LR(k) parsing

- L: Left-to-right
- R: right-most derivation
- k: k symbols of lookahead

LR(0) Parsing

Too weak to be useful.

Basis for practical algorithms.

Def: A LR(0) item is a production and a position within it.

$A \rightarrow \alpha \bullet \beta$

$\bullet$ marks the position

If $|\alpha\beta| = n$, positions between $0 \ldots n$

Items track things that could be going on in the parse.

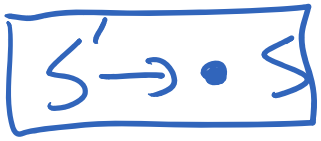
"I could be parsing an A, and might have seen α."

LR(0) Items as an NFA.

$S' \to S$

$S \to (S)$

$S \to a$



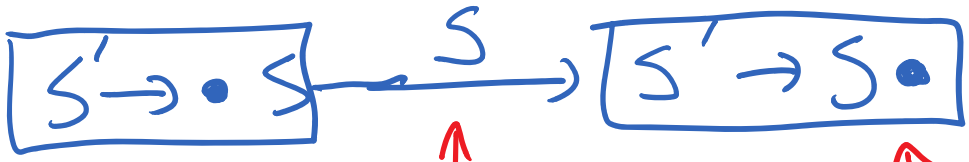
At the beginning, we're hoping to parse an S

LR(0) Items as an NFA.

$S' \to S$

$S \to (S)$

$S \to a$

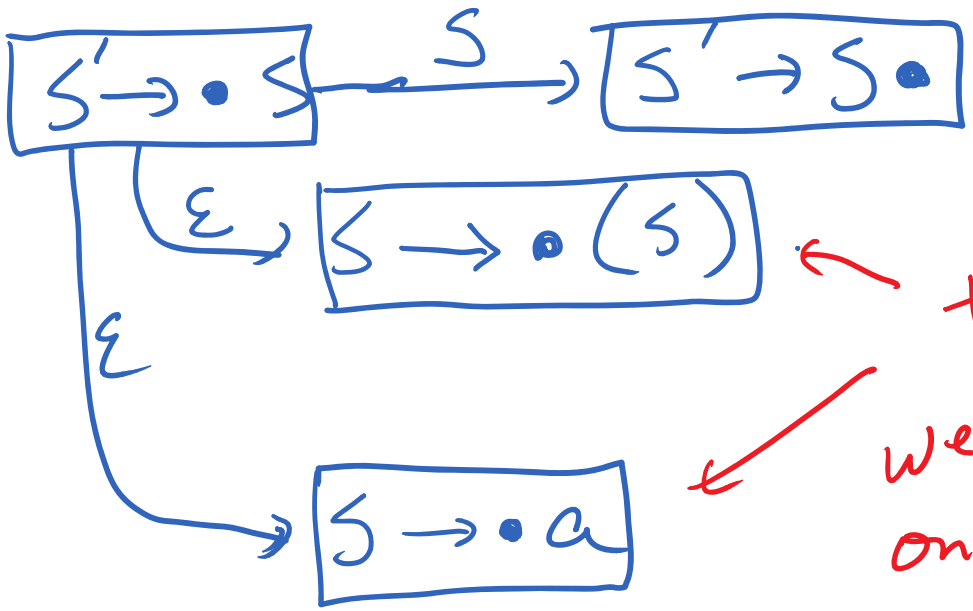


If we see an S

we go to this item.

LR(0) Items as an NFA.

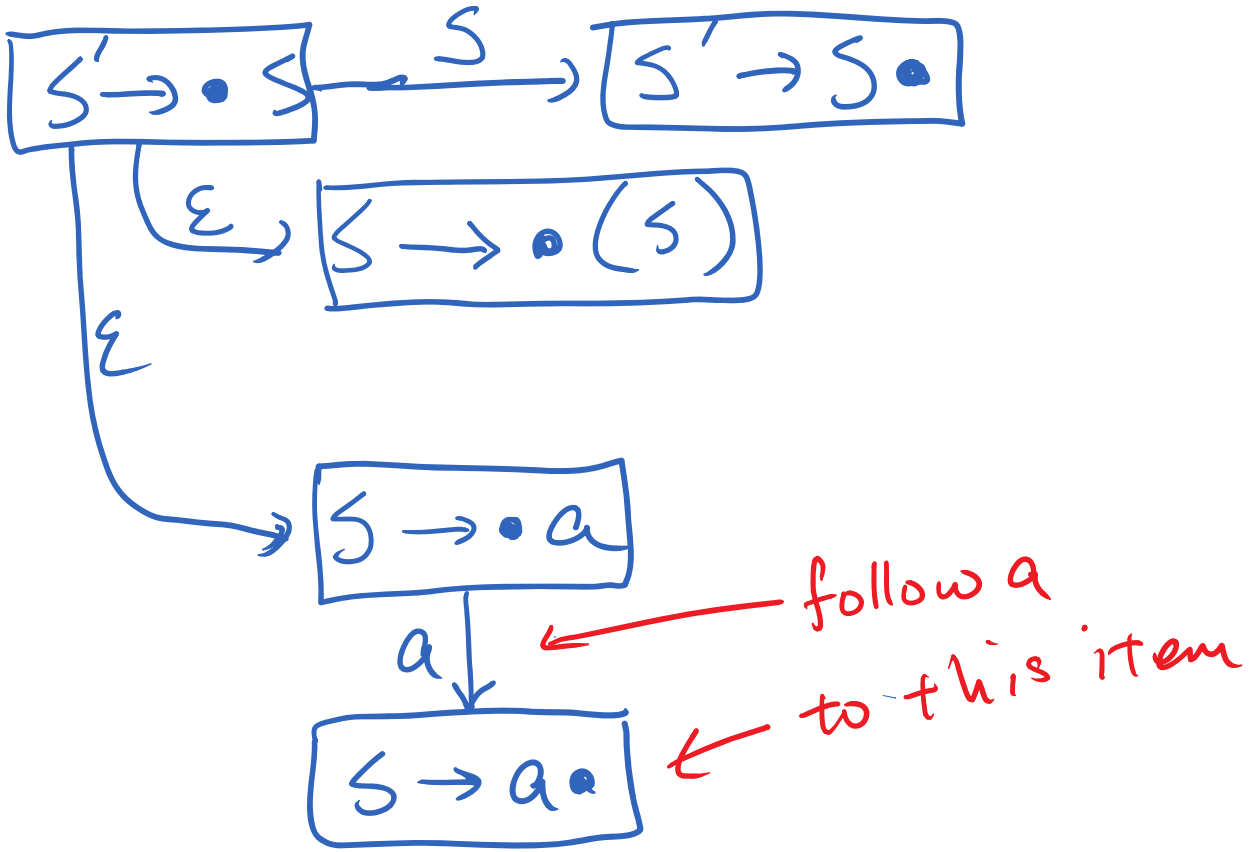
$S' \to S$
$S \to (S)$
$S \to a$



to see an S, we have to see one of these strings that reduces to it.

LR(0) Items as an NFA.

$S' \to S$
$S \to (S)$
$S \to a$

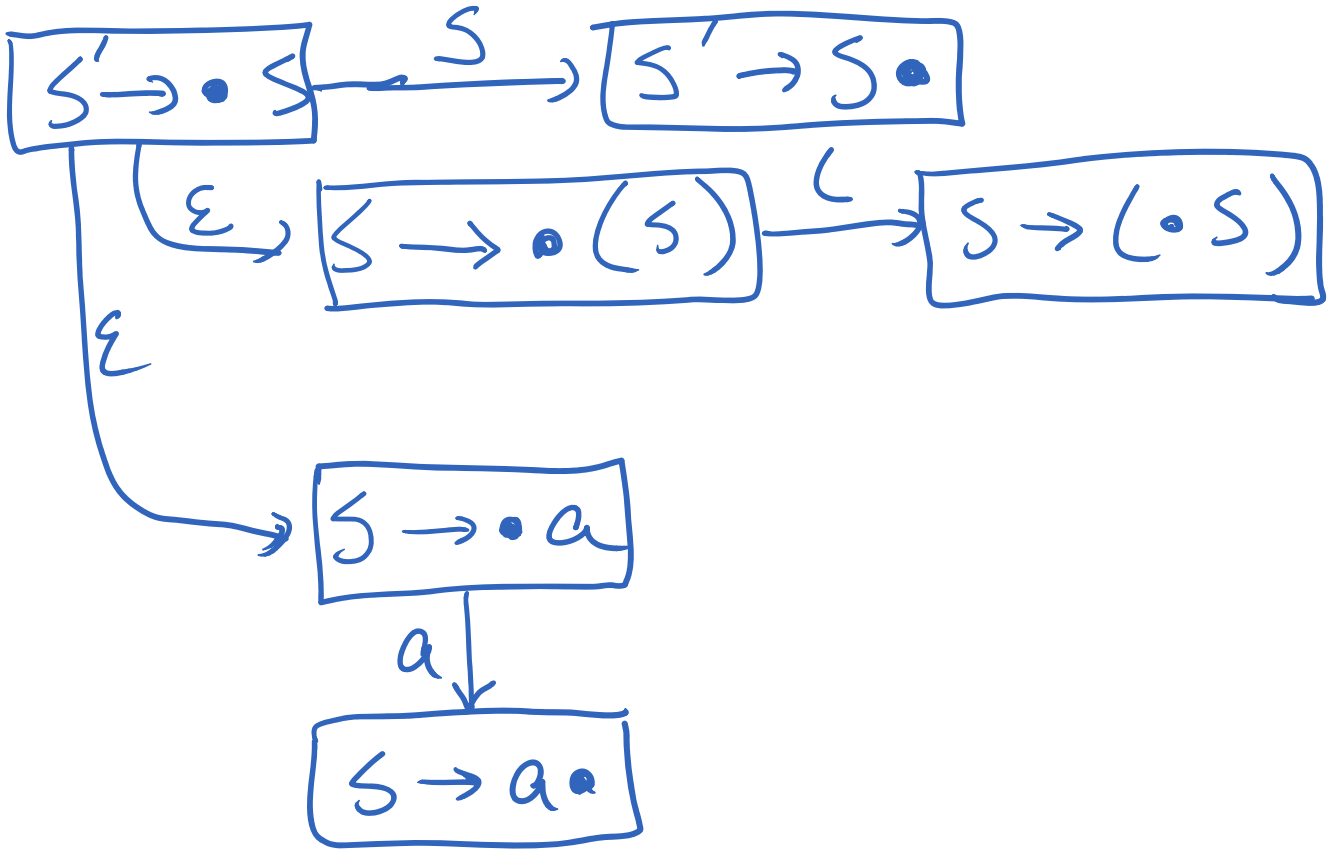


LR(0) Items as an NFA.

$S' \to S$

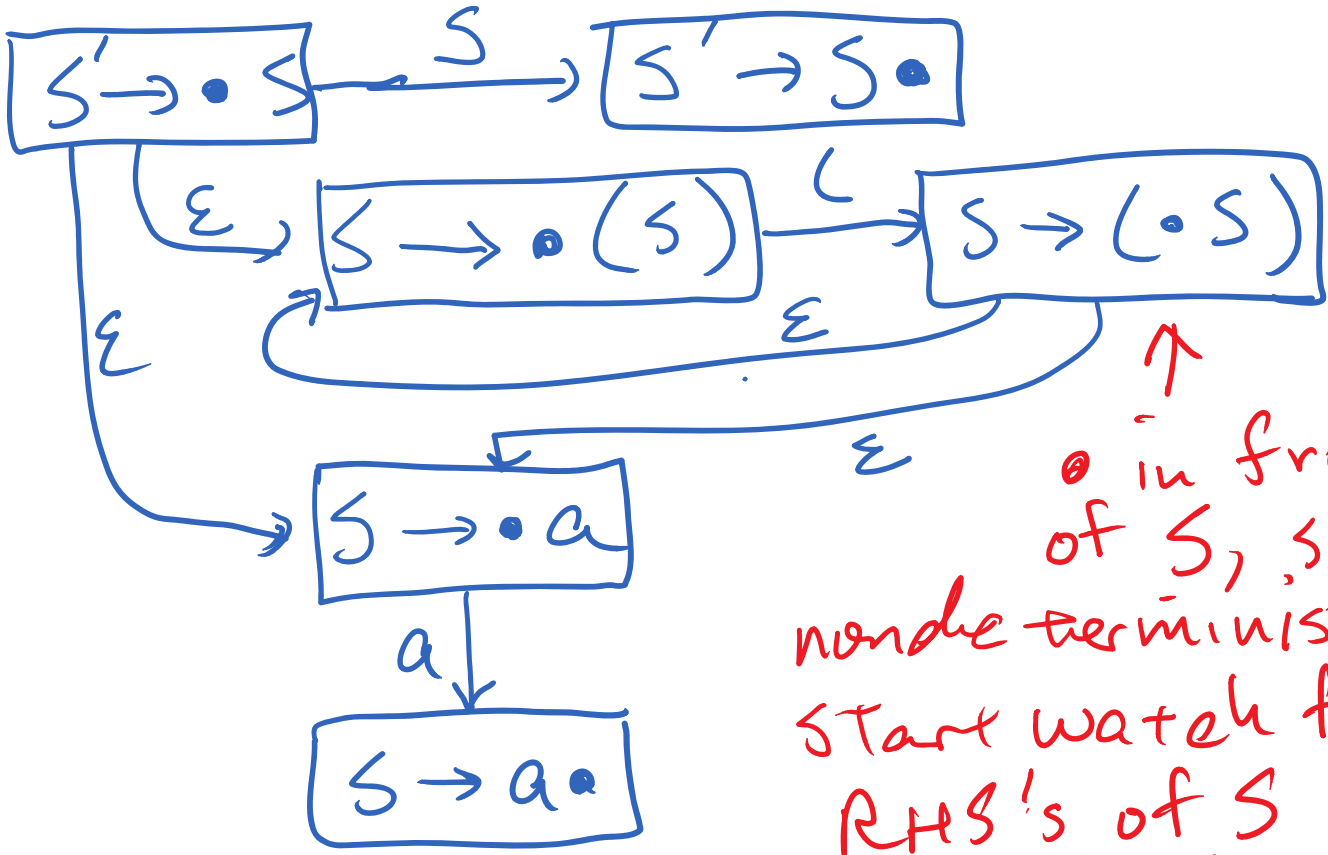
$S \to (S)$

$S \to a$



LR(0) Items as an NFA.

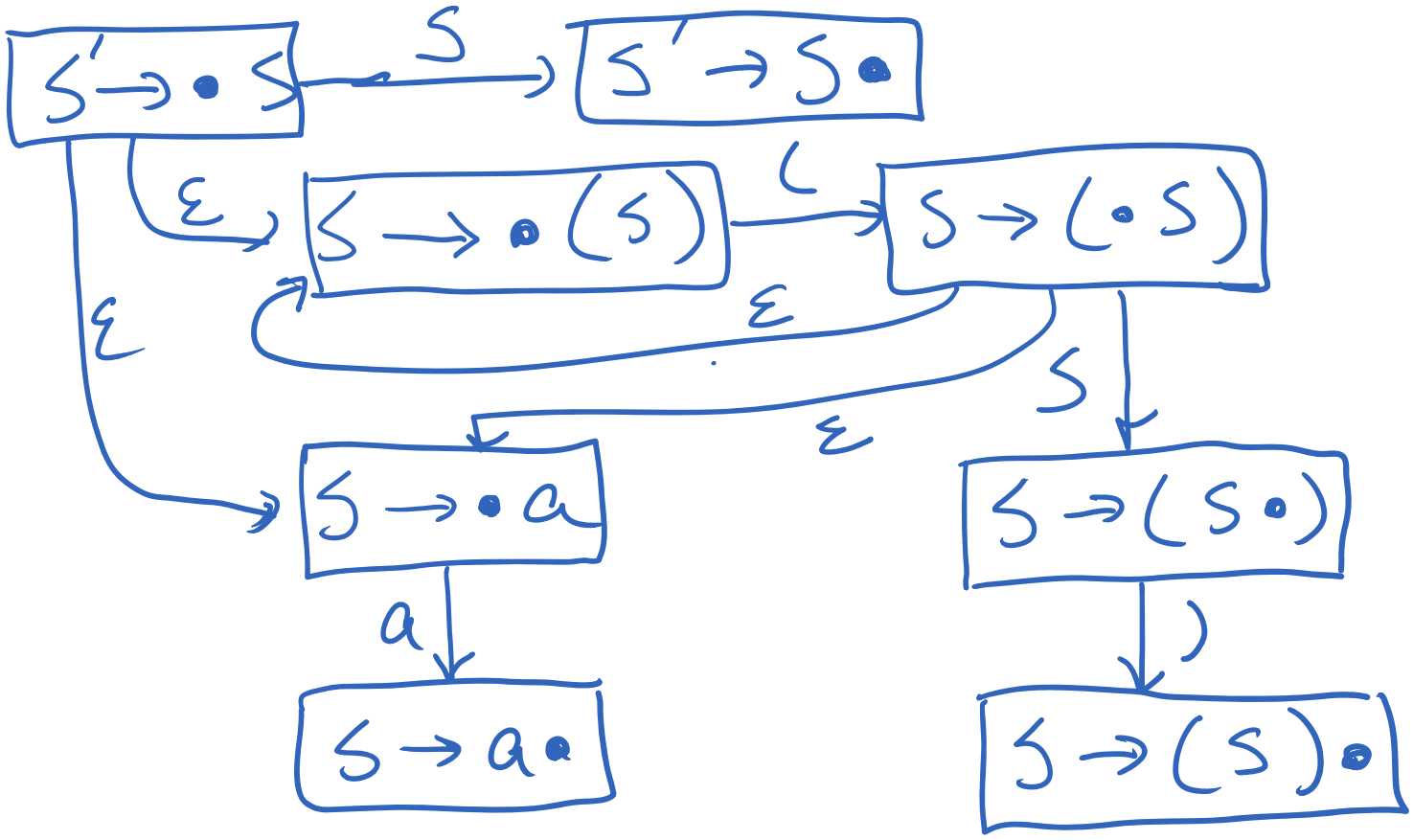
$S' \to S$
$S \to (S)$
$S \to a$



• in front of S, so nondeterministically start watch for RHS's of S productions.

LR(0) Items as an NFA.

$S' \to S$
$S \to (S)$
$S \to a$



LR(0) Machine

Could use powerset construction to convert to DFA

States of DFA = sets of items

Or use direct construction

Direct Construction of LR(0) Machine

1. Add new start symbol S', new production $S' \to S$.
2. Initial item set starts with $\{ S' \to \bullet S \}$ (more items will be added).

CLOSURE(itemset)

- Whenever $A \to \alpha \bullet B \beta$ is in itemset add to itemset all items of the form:

$$B \to \bullet \gamma$$

- Repeat until no more items need to be added.

Example

$S' \to S$

$S \to (S)$

$S \to a$

$\{ S' \to \bullet S \}$

Add $S \to \bullet (S)$

$S \to \bullet a$

Example

$S' \to S$

$S \to (S)$

$S \to a$

$\{ S' \to \bullet S \}$

Add $S \to \bullet (S)$

$S \to \bullet a$

Complete item set

$\{ S' \to \bullet S$

$S \to \bullet (S)$

$S \to \bullet a \}$

GOTO(itemset, X):

Whenever $A \to \alpha \bullet X \beta$ in itemset

- Start a new itemset
- add $A \to \alpha X \bullet \beta$ for all items of the form $A \to \alpha \bullet X \beta$ in the original itemset.
- Do CLOSURE on new itemset
- If result is same as existing itemset use that instead.
- Add transition on X from old itemset to new itemset.

$S' \to S$

$S \to (S)$

$S \to a$

$S' \to \bullet S$

$S' \to S$

$S \to (S)$

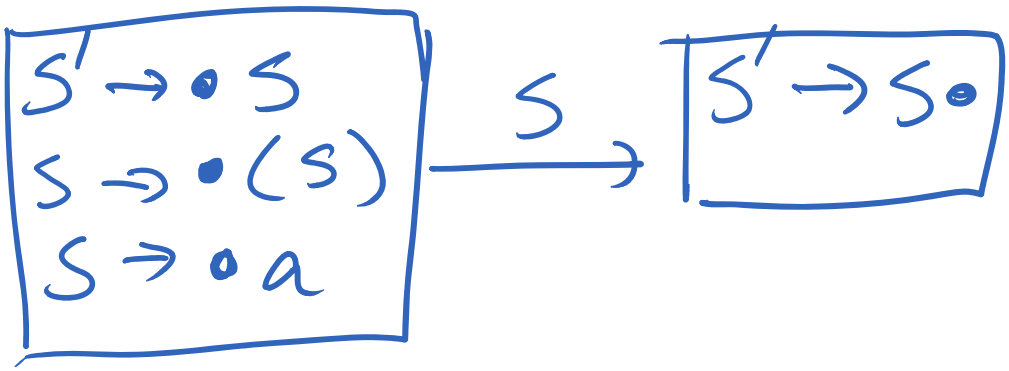
$S \to a$

$S' \to \bullet S$

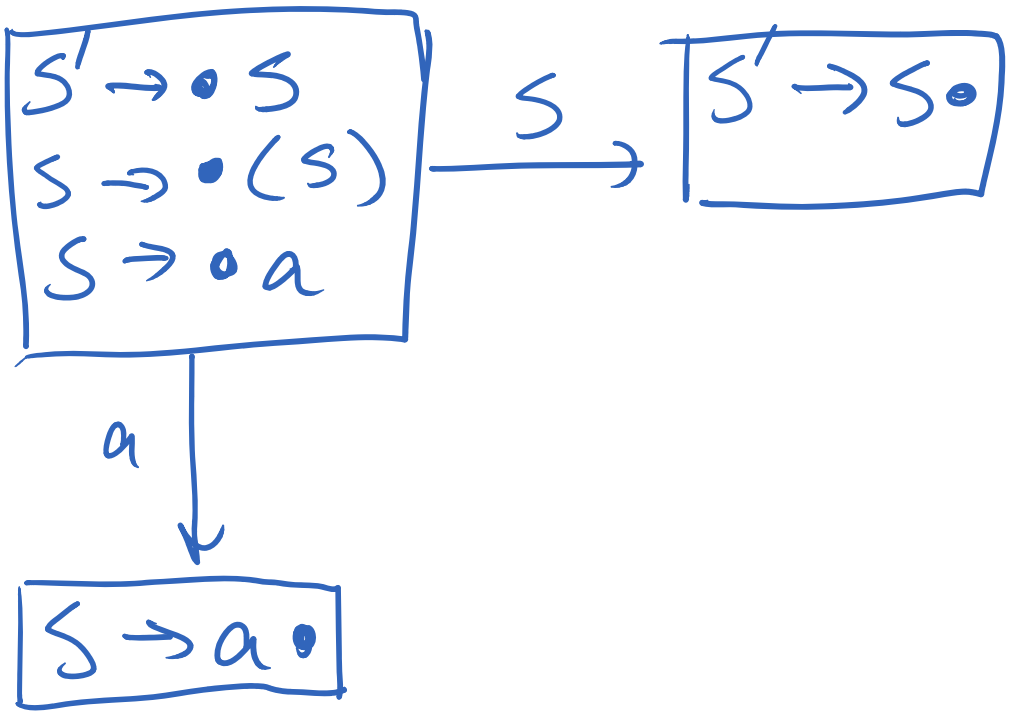
$S \to \bullet (S)$

$S \to \bullet a$

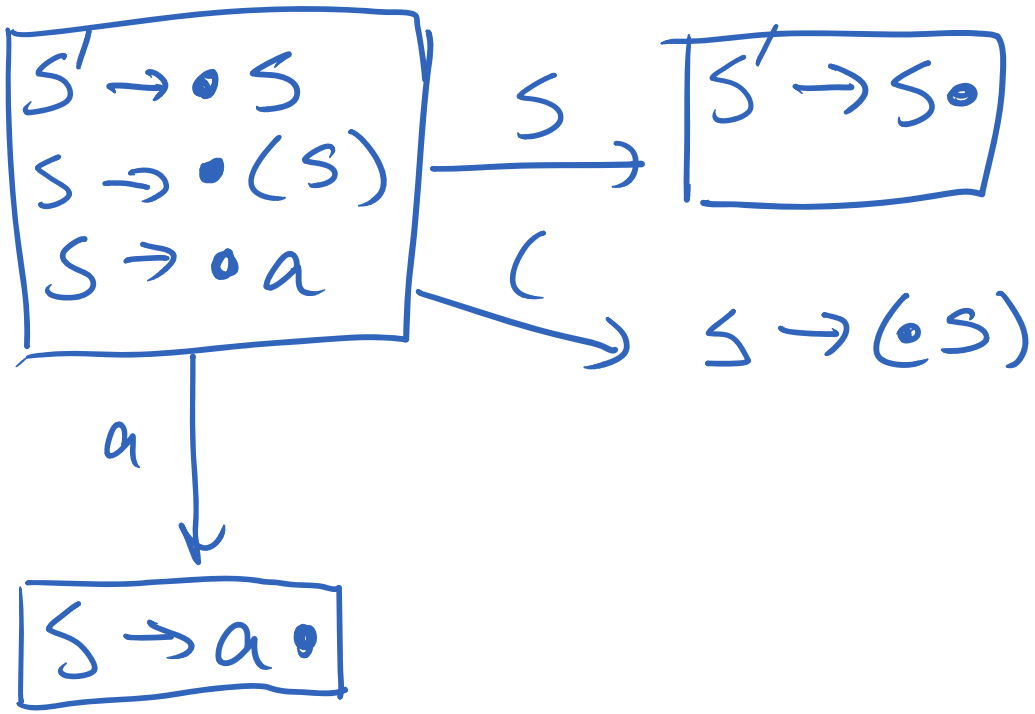
$S' \to S$
$S \to (S)$
$S \to a$



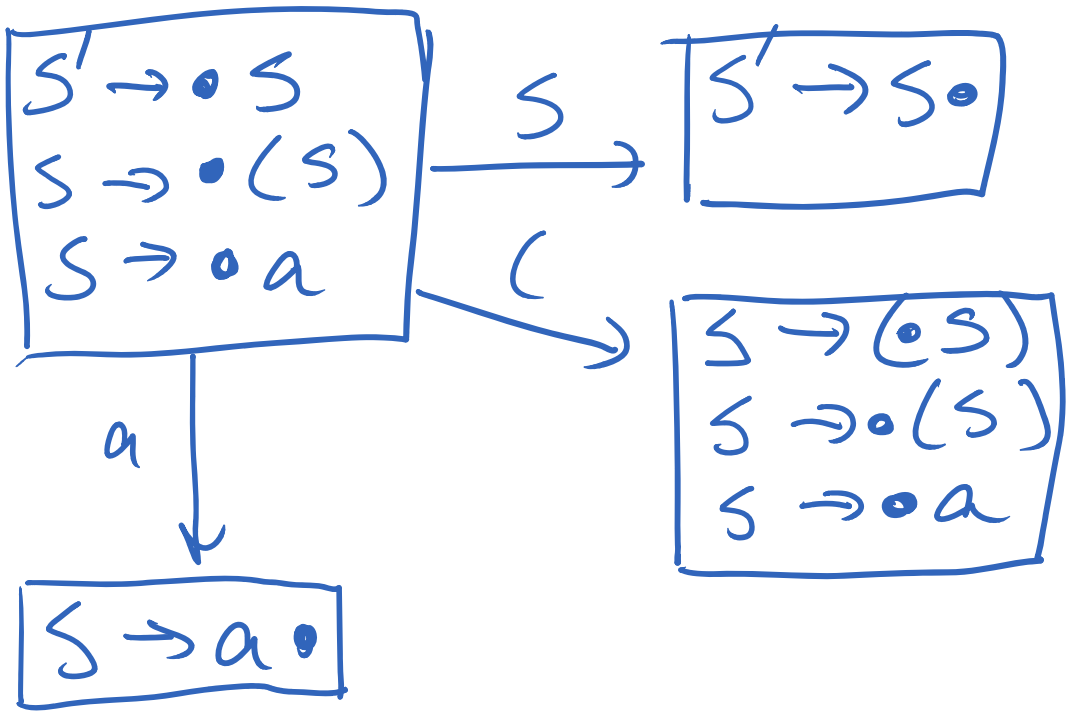
$S' \to S$
$S \to (S)$
$S \to a$



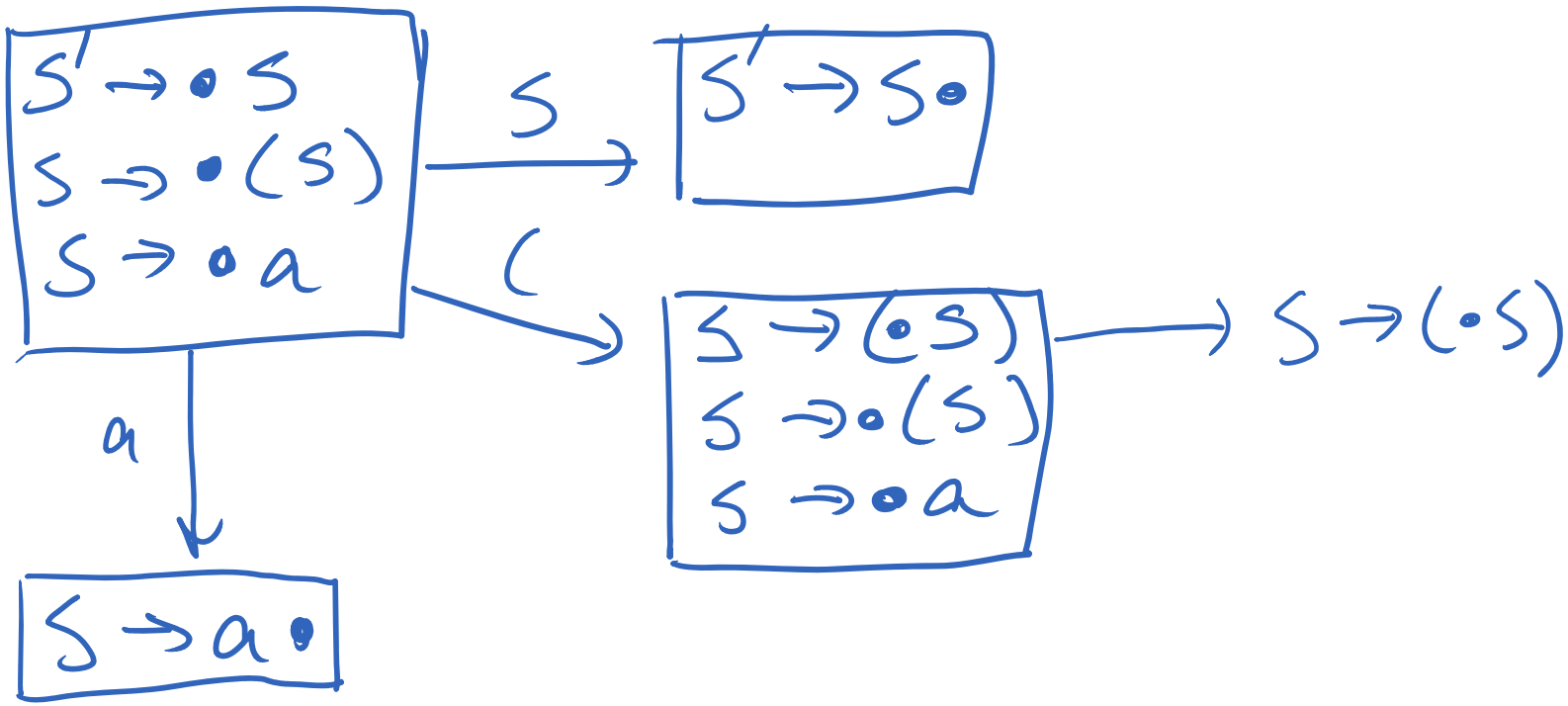
$S' \to S$
$S \to (S)$
$S \to a$



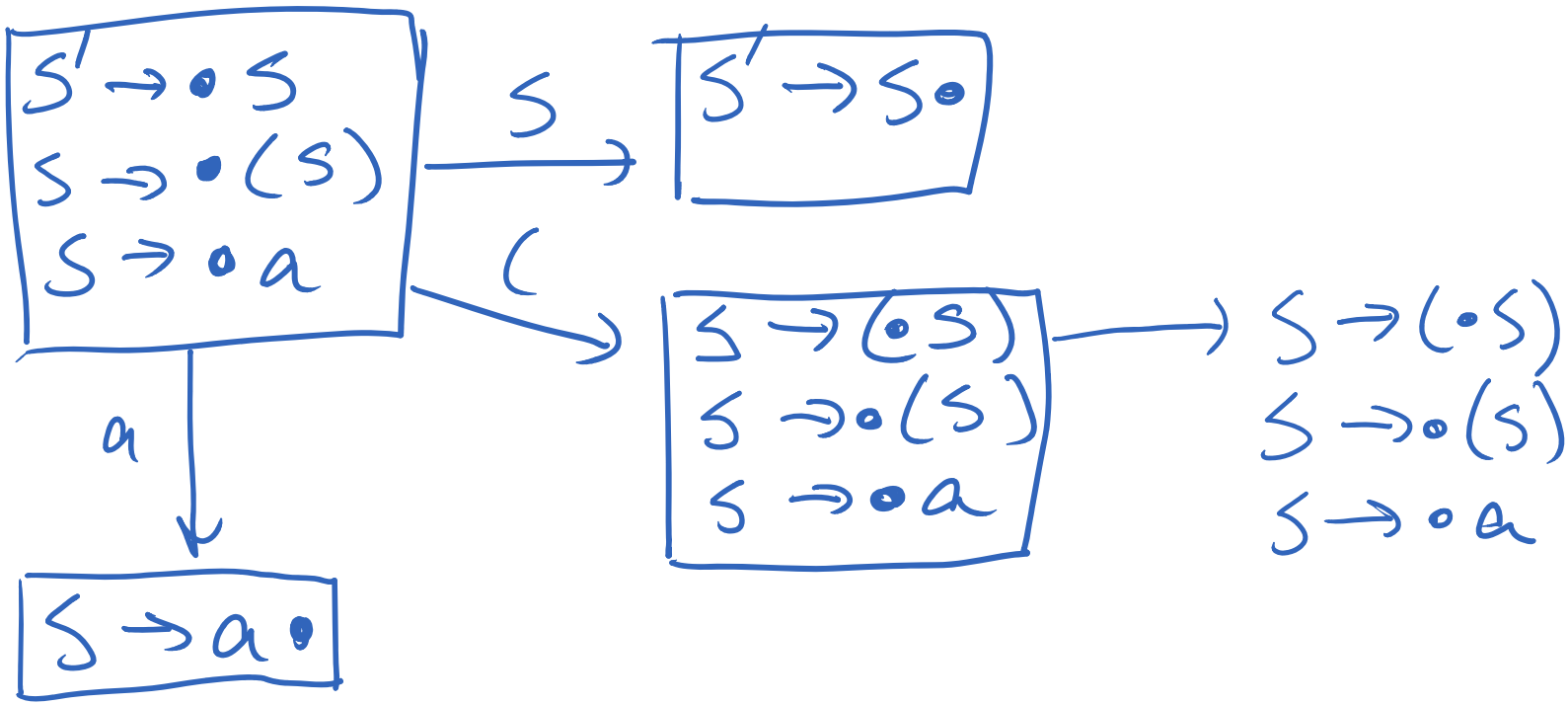
$S' \to S$
$S \to (S)$
$S \to a$



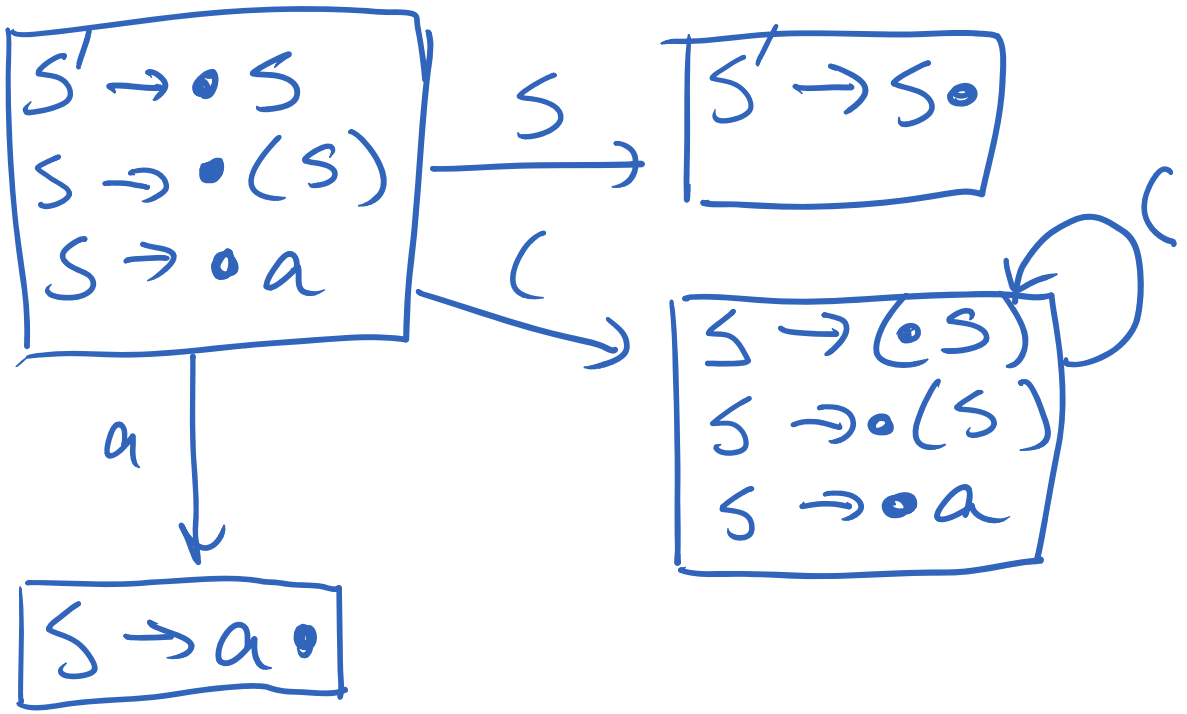
$S' \to S$
$S \to (S)$
$S \to a$



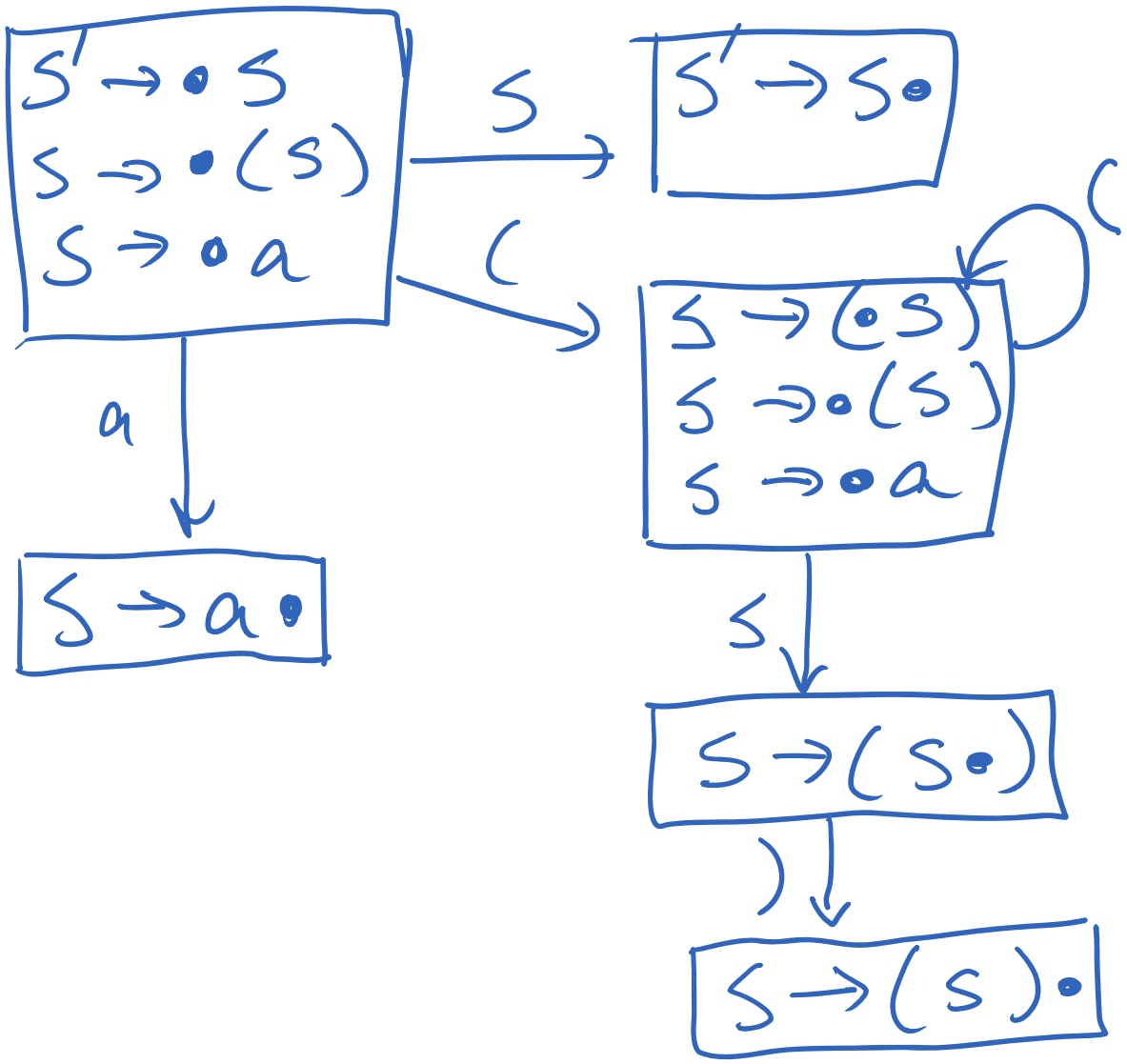
$S' \to S$
$S \to (S)$
$S \to a$



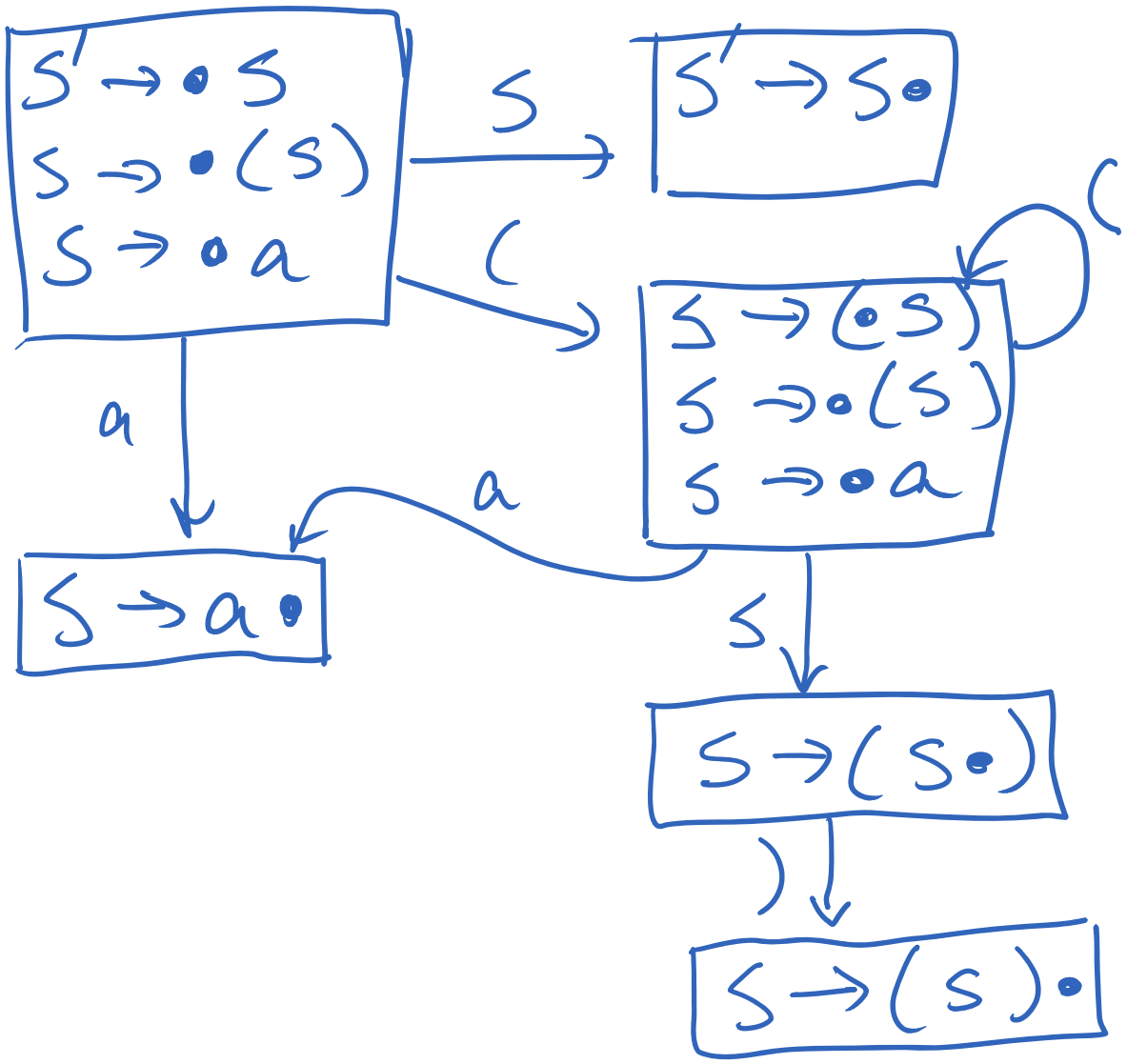
$S' \to S$
$S \to (S)$
$S \to a$



$S' \to S$
$S \to (S)$
$S \to a$



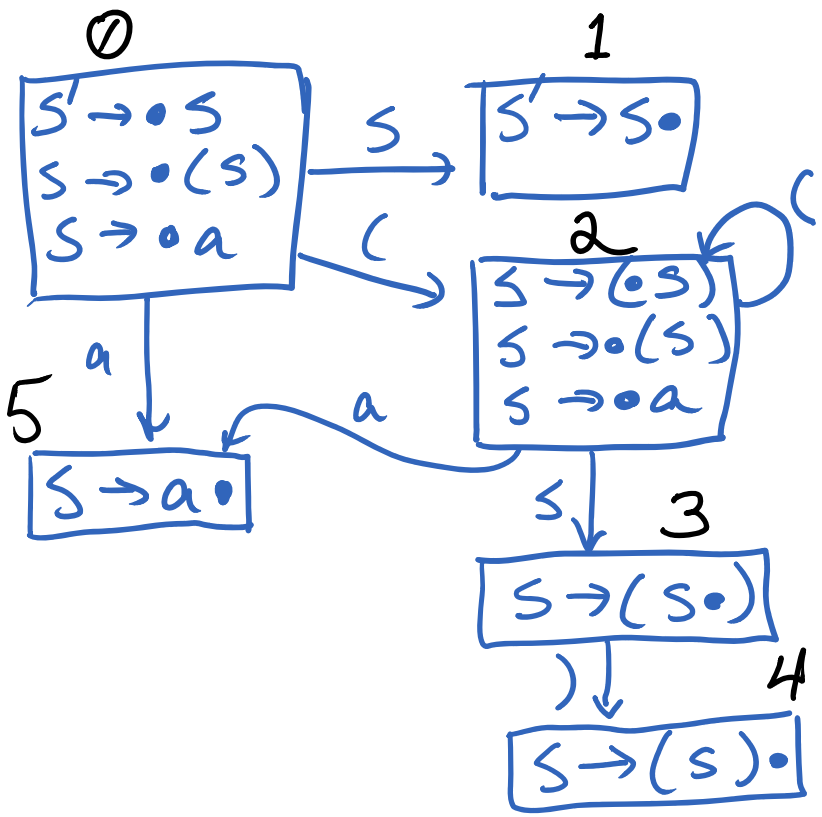
S' → S
S → (S)
S → a



LR(0) parsing

"State" = itemset

States go on the stack.



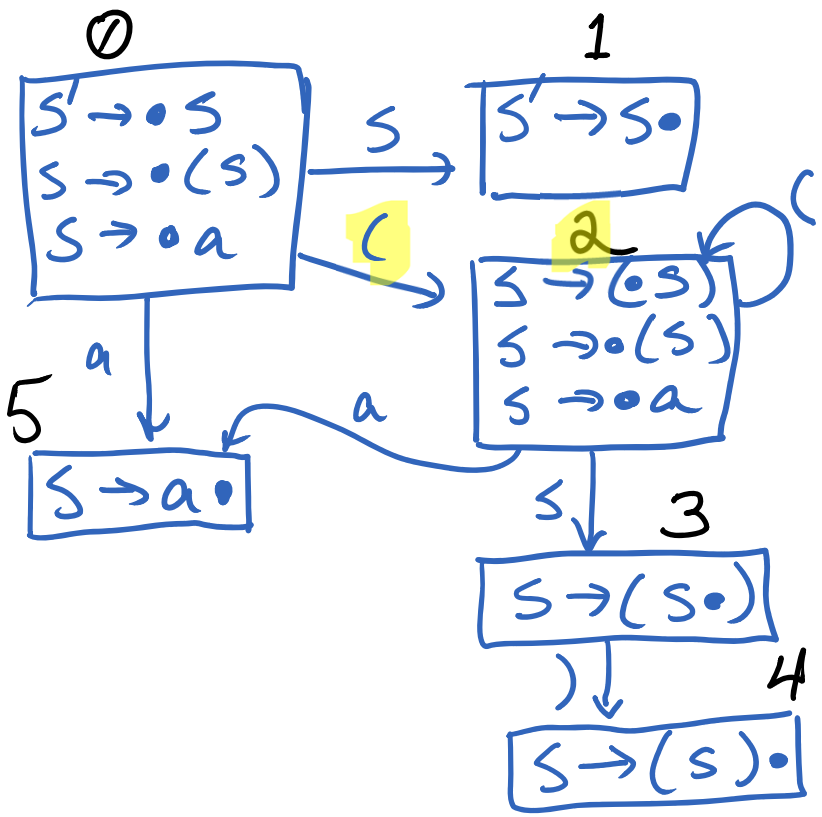
Stack	Input
0	((a))$
$	

LR(0) parsing

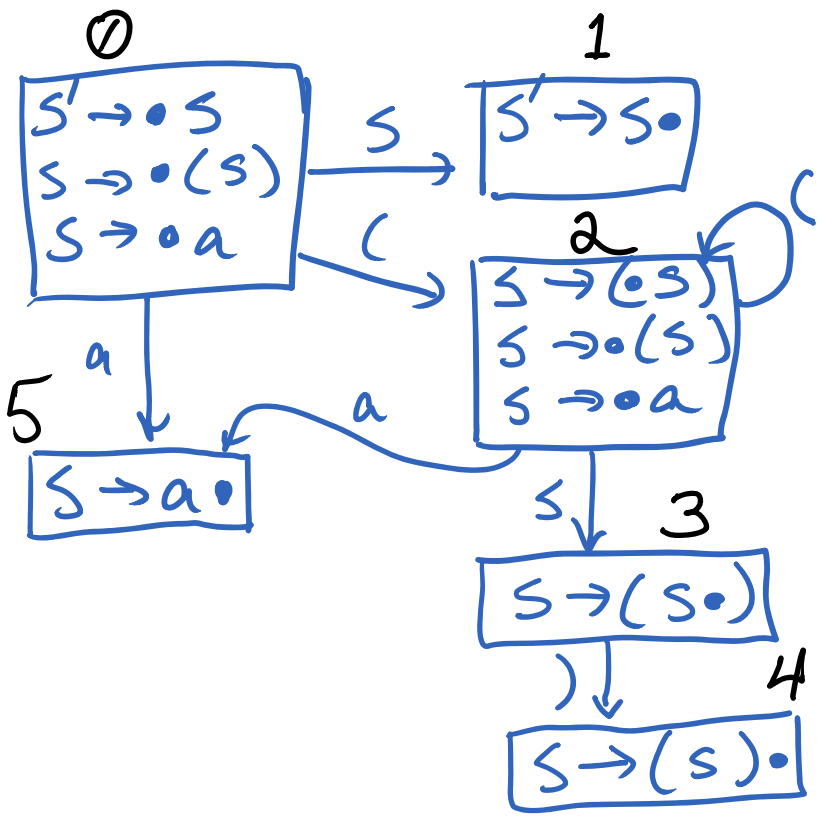
Shift item $A \rightarrow \alpha \bullet a \beta$

↑ dot in front of terminal

Shift when "a" is next input and
$A \rightarrow \alpha \bullet a \beta$ is in current itemset



Stack	Input
0	((a))$
$	
0 2	(a))$
$ (	



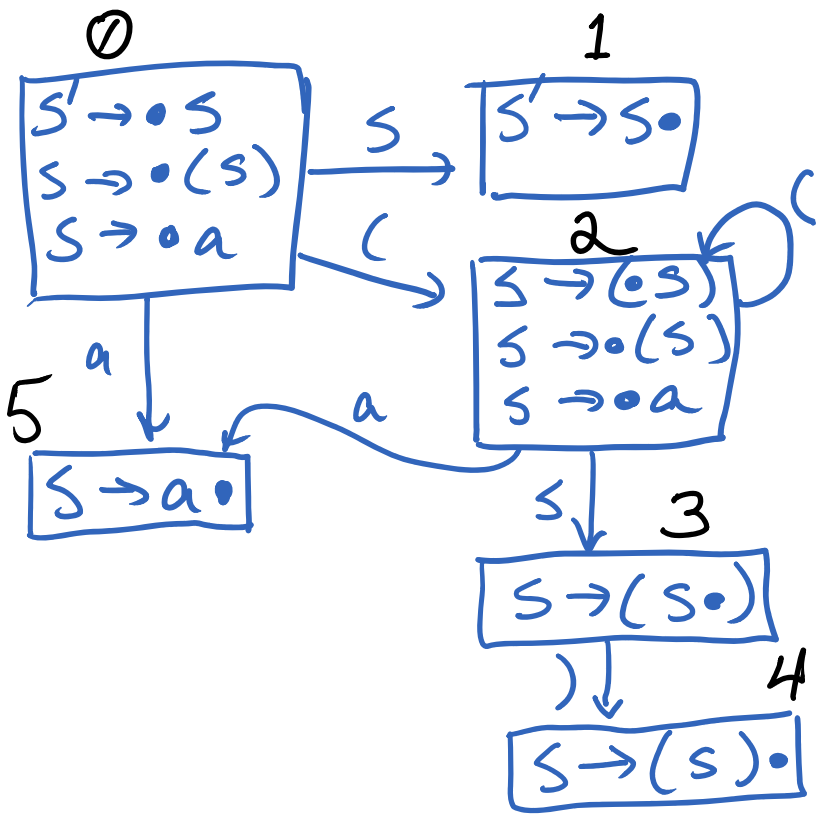
Stack	Input
0	((a))$
$	
0 2	(a))$
$ (	
0 2 2	a))$
$ ((	
0 2 2 5	))$
$ ((a	

Reduce item: $A \rightarrow \alpha \bullet$

↑ dot at end.

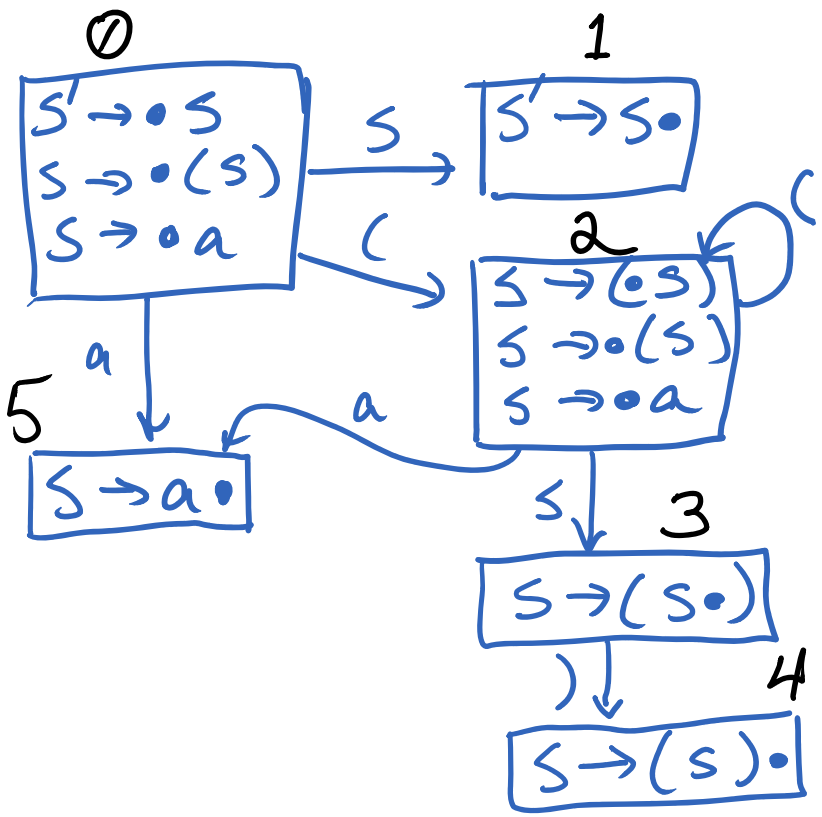
Reduce:

- Pop a state for each symbol in α
- From new top state, follow 'A' transition and push that state.



Stack	Input
0	((a))$
$	
0 2	(a))$
$ (	
0 2 2	a))$
$ ((	
0 2 2 5	))$
$ ((a	
0 2 2	))$
$ ((	

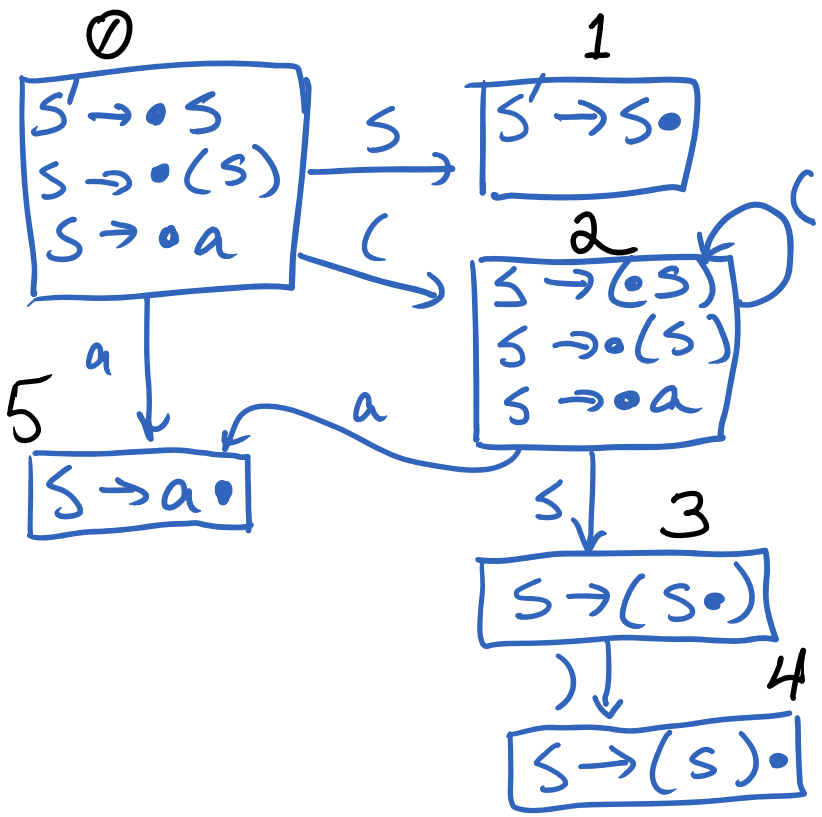
pop 1 state



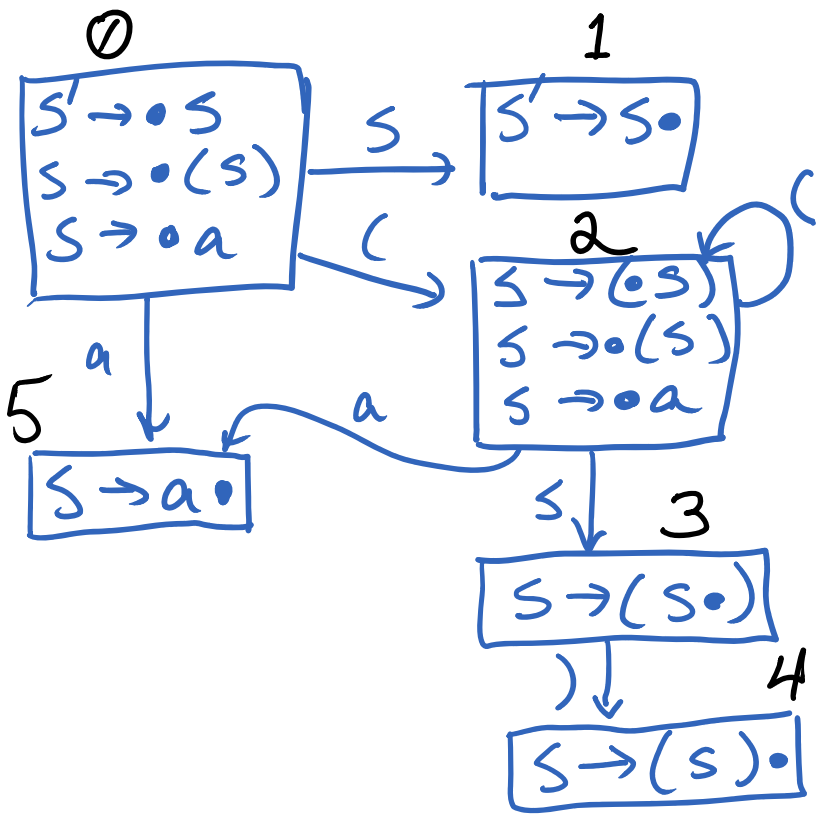
Stack	Input
0	((a))$
$	
0 2	(a))$
$ (	
0 2 2	a))$
$ ((	
0 2 2 5	))$
$ ((a	
0 2 2	))$
$ ((	

pop 1 state

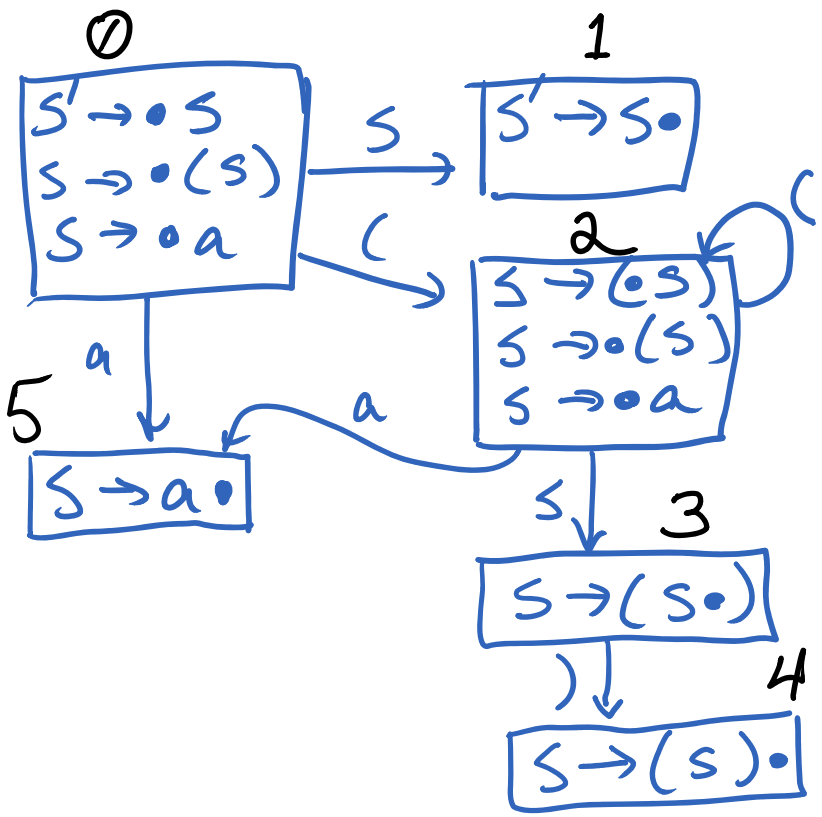
2 —S→ 3



Stack	Input
0	((a))$
$	
0 2	(a))$
$ (	
0 2 2	a))$
$ ((	
0 2 2 5	))$
$ ((a	
0 2 2 3	))$
$ ((S	

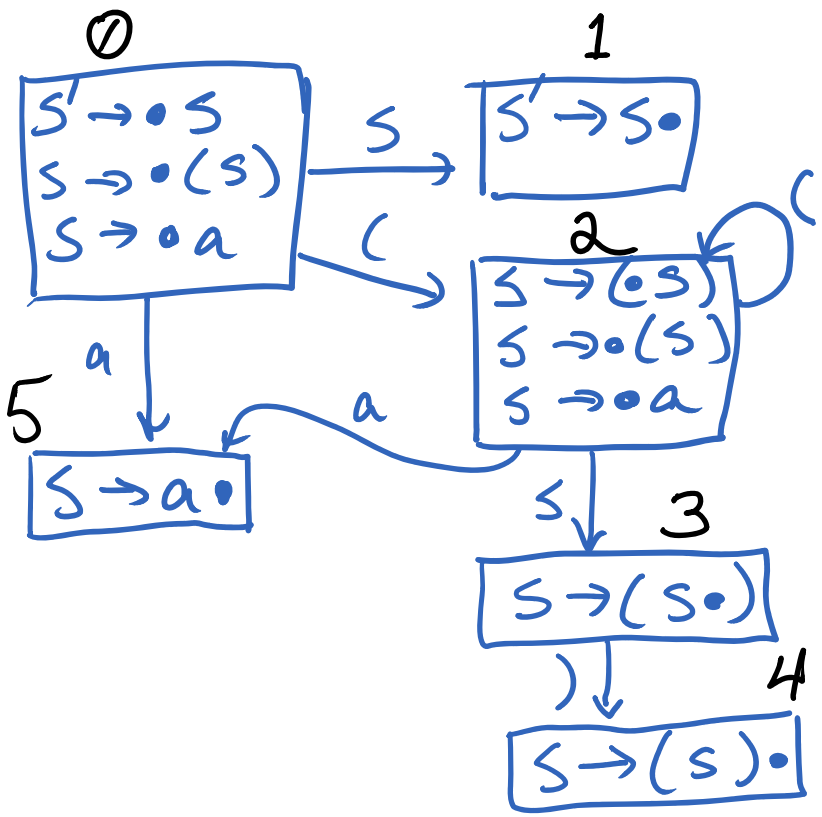


Stack	Input
0	((a))$
$	
0 2	(a))$
$ (	
0 2 2	a))$
$ ((	
0 2 2 5	))$
$ ((a	
0 2 2 3	))$
$ ((S	
0 2 2 3 4	)$
$ ((S)	

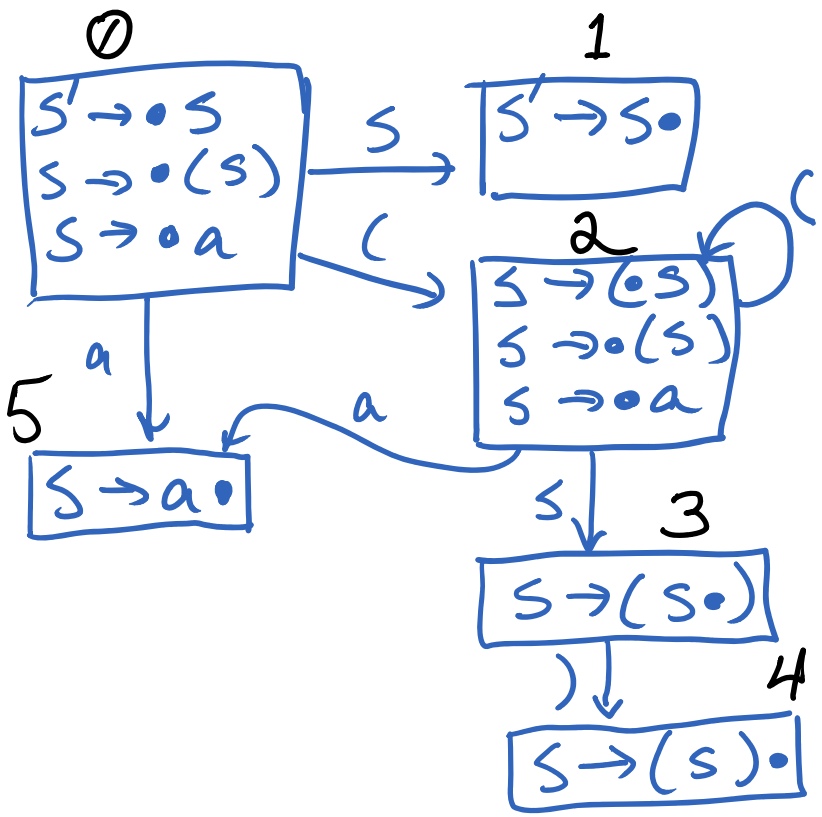


Stack	Input
0	((a))$
$	
0 2	(a))$
$ (	
0 2 2	a))$
$ ((	
0 2 2 5	))$
$ ((a	
0 2 2 3	))$
$ ((S	
0 2 2 3 4	)$
$ ((S)	
0 2	)$
$ (	

pop 3 states



Stack	Input
0	((a))$
$	
0 2	(a))$
$ (	
0 2 2	a))$
$ ((	
0 2 2 5	))$
$ ((a	
0 2 2 3	))$
$ ((S	
0 2 2 3 4	)$
$ ((S)	
0 2 3	)$
$ (S	



Stack	Input
0	((a))$
$	
0 2	(a))$
$ (	a))$
0 2 2	a))$
$ ((	a))$
0 2 2 5	))$
$ ((a	))$
0 2 2 S	))$
$ ((S	))$
0 2 2 3	))$
$ ((S)	))$
0 2 3 4	))$
$ (S))	))$
0 1	))$
$ S	))$

accept

SLR(1) Parsing

$S' \to S$

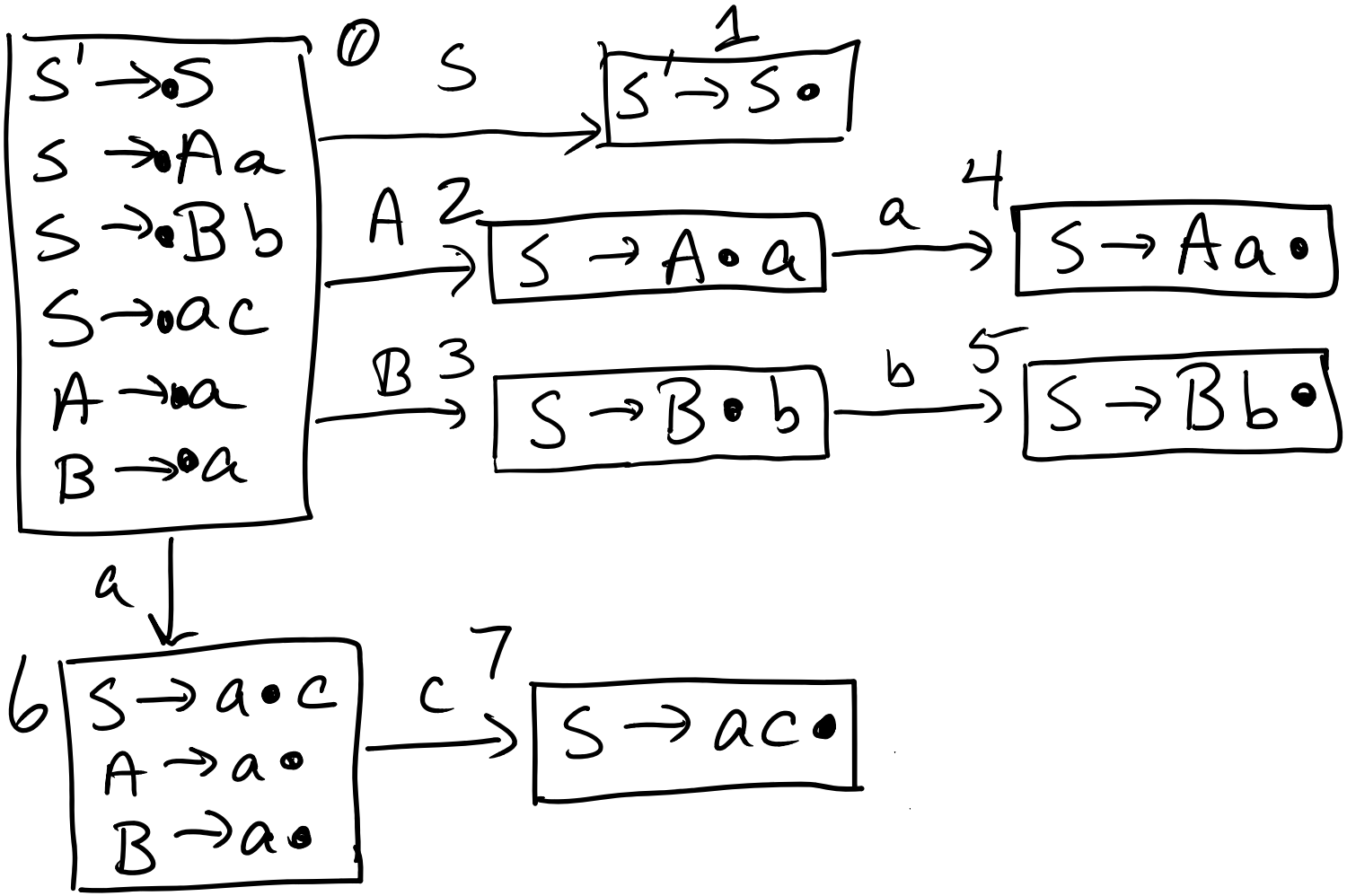
$S \to Aa$

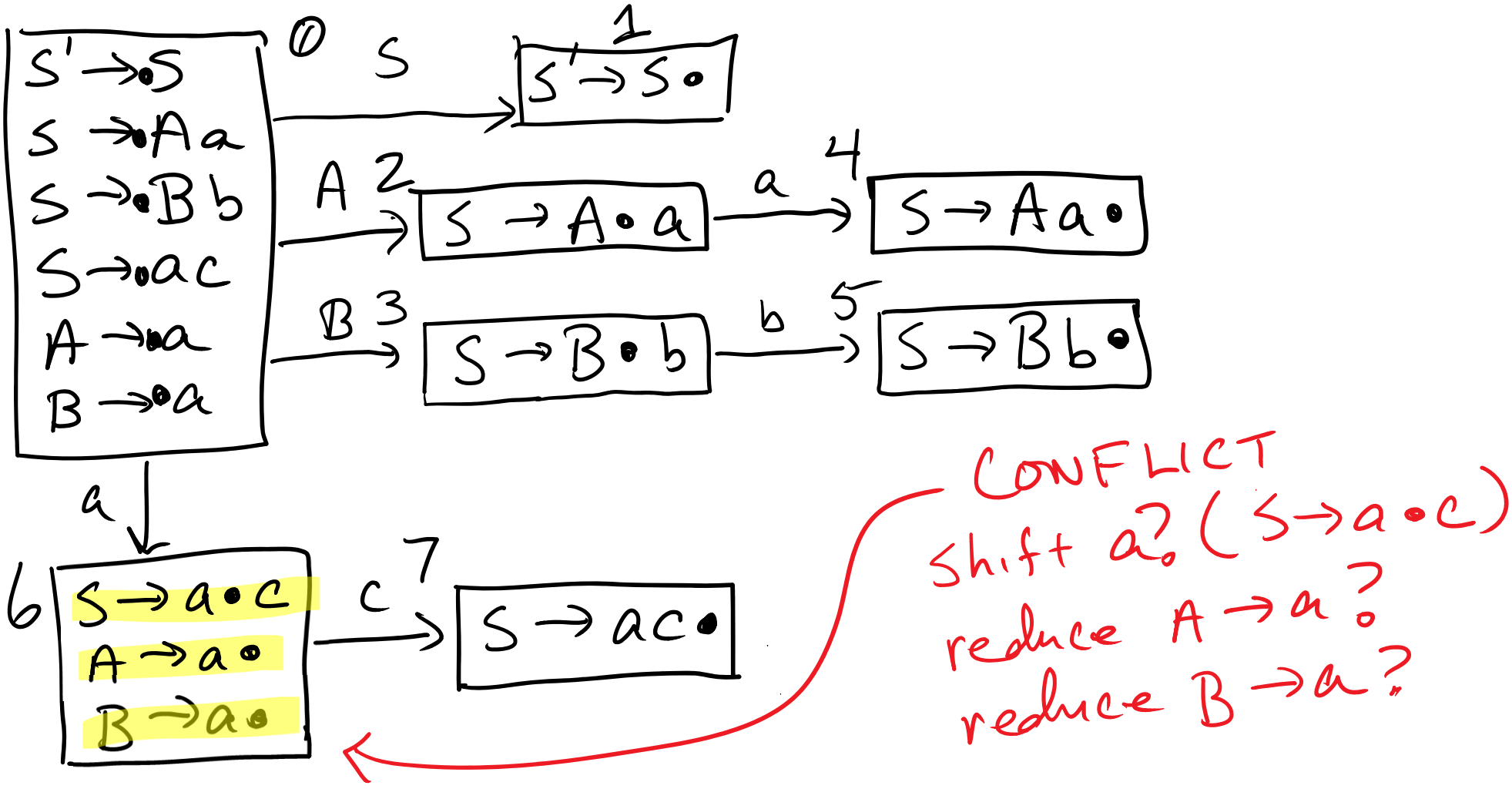
$S \to Bb$

$S \to ac$

$A \to a$

$B \to a$


0
S' → •S
S → •A a
S → •B b
S → •a c
A → •a
B → •a
S
1
S' → S•
A
2
S → A • a
a
4
S → A a •
B
3
S → B • b
b
5
S → B b •
a
6
S → a • c
A → a •
B → a •
c
7
S → a c •


0
S' → •S
S → •A a
S → •B b
S → •a c
A → •a
B → •a
S
1
S' → S•
A
2
S → A • a
a
4
S → A a •
B
3
S → B • b
b
5
S → B b •
a
6
S → a • c
A → a •
B → a •
c
7
S → a c •
CONFLICT
shift a? (S → a • c)
reduce A → a?
reduce B → a?

Conflicts

Shift/reduce

$A \to \alpha \bullet a\, \beta$ – shift?

$B \to \gamma \bullet$ – or reduce?

Reduce/reduce

$A \to \alpha \bullet$

$B \to \beta \bullet$

} reduce which?

Idea: Use lookaheads to resolve conflicts (works sometimes).

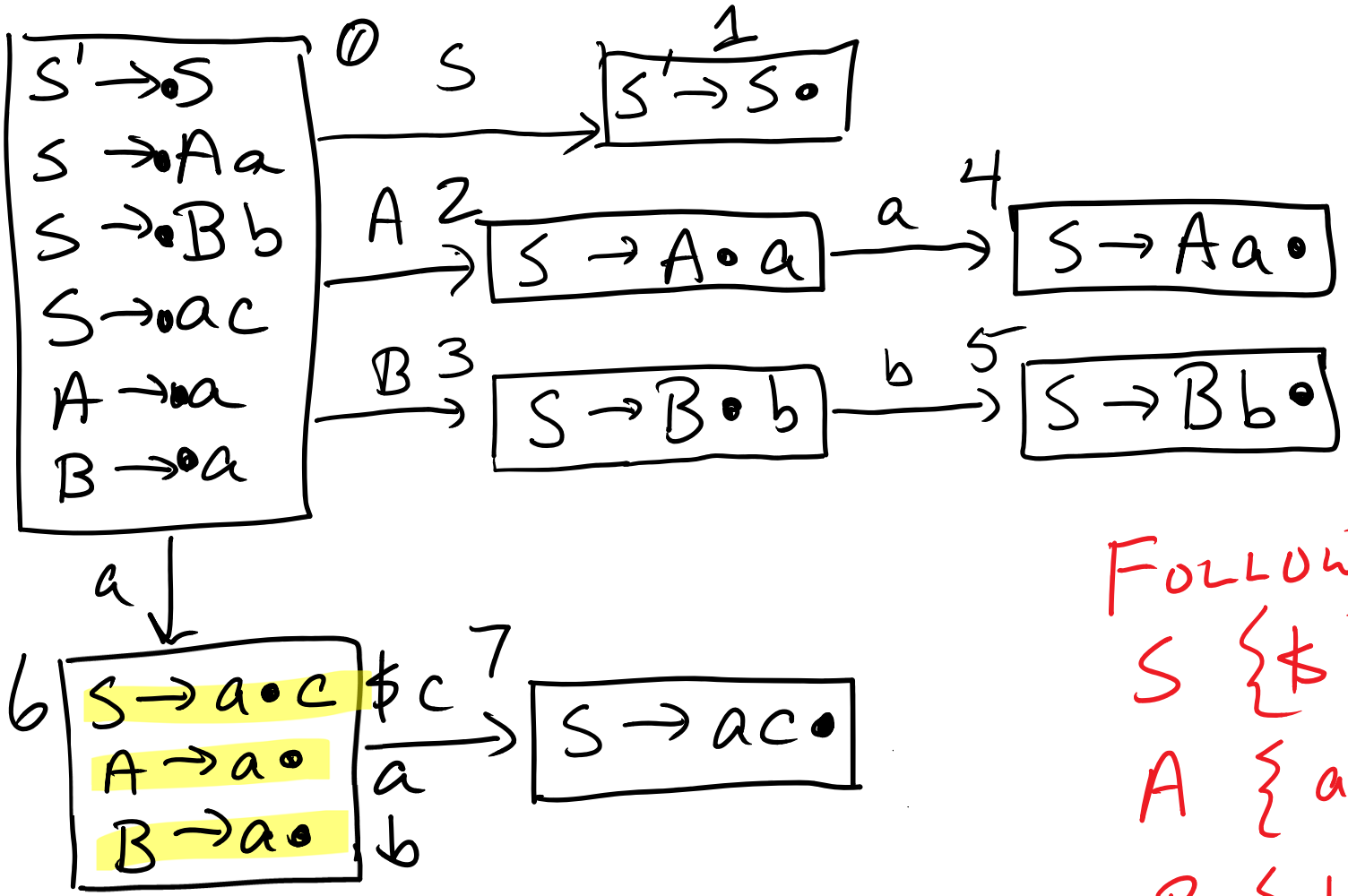
SLR(1)

LR(1)

LALR(1)

SLR(1) – "Simple LR"

Idea: Use FOLLOW sets



FOLLOW

S {$}

A {a}

B {b}

Distinguish all LR(0) conflicts in state 6.

XLR(1) Parse Tables

ACTION [state, terminal]

$$= \begin{cases} \text{shift}\langle \text{state} \rangle \\ \text{reduce}\langle \text{production} \rangle \end{cases}$$

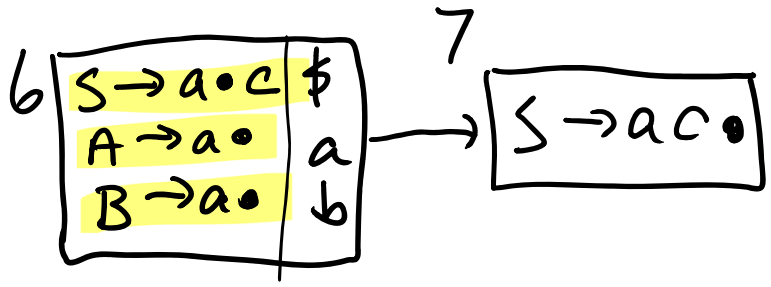
Says what to shift or reduce on next input.

GOTO [state, nonterminal]

= state

Next state on reduction of $A \to \alpha \cdot$

0 S' → S
1 S → Aa
2 S → Bb
3 S → ac
4 A → a
5 B → a



	ACTION				GOTO		
	a	b	c	$	S	A	B
0	s6				1	2	3
1				acc			
2	s4						
3		s5					
4				r1			
5				r2			
6	r4	r5	s7				
7				r3			

0 S'→S
1 S→Aa
2 S→Bb
3 S→ac
4 A→a
5 B→a

	ACTION				GOTO		
	a	b	c	$	S	A	B
0	s6				1	2	3
1				acc			
2	s4						
3		s5					
4				r1			
5				r2			
6	r4	r5	s7				
7				r3			

stack	input	
0	ab$	s6
$		
0 6	b$	r5
$ a		
0 3	b$	s5
$ B		
0 3 5	$	r2
$ B b		
0 1	$	
$ S		

0 S'→S
1 S→Aa
2 S→Bb
3 S→ac
4 A→a
5 B→a

	ACTION				GOTO		
	a	b	c	$	S	A	B
0	s6				1	2	3
1				acc			
2	s4						
3		s5					
4				r1			
5				r2			
6	r4	r5	s7				
7				r3			

stack	input	
0	ab$	s6
0 6	b$	r5
0 a		
0 3	b$	s5
0 B		
0 3 5	$	r2
0 B b		
0 1	$	acc
0 S		