Syntax Analysis – Part V

(Finish LR(0) Parsing, Start on LR(1) Parsing)

Yongjun Park

Hanyang University

From Last Time: Shift-Reduce Parsing

S → S + E | E
E → num | (S)

derivation	stack	input stream	action
(1+2+(3+4))+5		(1+2+(3+4))+5	shift
(1+2+(3+4))+5	(	1+2+(3+4))+5	shift
(1+2+(3+4))+5	(1	+2+(3+4))+5	reduce E→ num
(E+2+(3+4))+5	(E	+2+(3+4))+5	reduce S→ E
(S+2+(3+4))+5	(S	+2+(3+4))+5	shift
(S+2+(3+4))+5	(S+	2+(3+4))+5	shift
(S+2+(3+4))+5	(S+2	+(3+4))+5	reduce E→ num
(S+E+(3+4))+5	(S+E	+(3+4))+5	reduce S → S+E
(S+(3+4))+5	(S	+(3+4))+5	shift
(S+(3+4))+5	(S+	(3+4))+5	shift
(S+(3+4))+5	(S+(	3+4))+5	shift
(S+(3+4))+5	(S+(3	+4))+5	reduce E→ num
…			

From Last Time: LR Parsing Table Example

We want to derive this in an algorithmic fashion

State	Input terminal					Non-terminals	
	(	)	id	,	$	S	L
1	s3		s2			g4	
2	S→id	S→id	S→id	S→id	S→id		
3	s3		s2			g7	g5
4					accept		
5		s6		s8			
6	S→(L)	S→(L)	S→(L)	S→(L)	S→(L)		
7	L→S	L→S	L→S	L→S	L→S		
8	s3		s2			g9	
9	L→L,S	L→L,S	L→L,S	L→L,S	L→L,S		

Start State and Closure

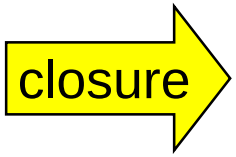
- **Start state**
  - Augment grammar with production: S' → S $
  - Start state of DFA has empty stack: S' → **.** S $
- **Closure of a parser state:**
  - Start with Closure(S) = S
  - Then for each item in S:
    - X → α **.** Y β
    - Add items for all the productions Y → γ to the closure of S: Y → **.** γ

Closure

S → (L) | id
L → S | L,S

DFA start state

S' → . S $

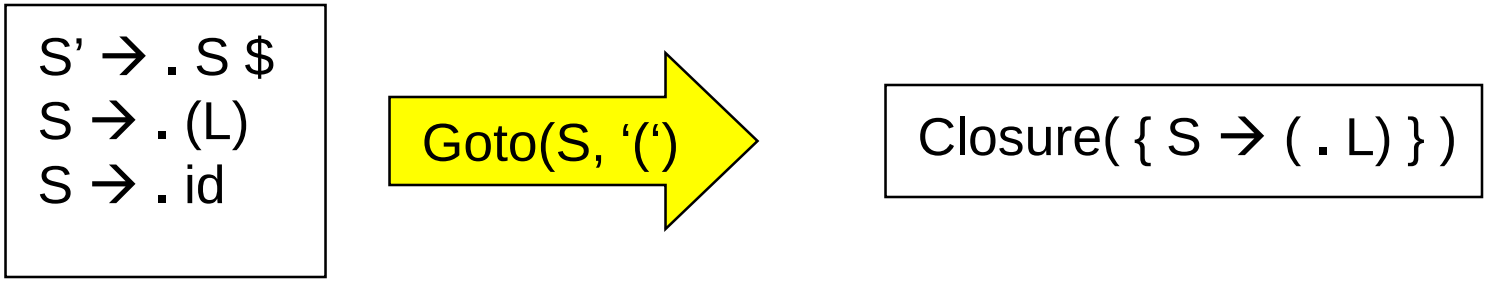


S' → . S $
S → . (L)
S → . id

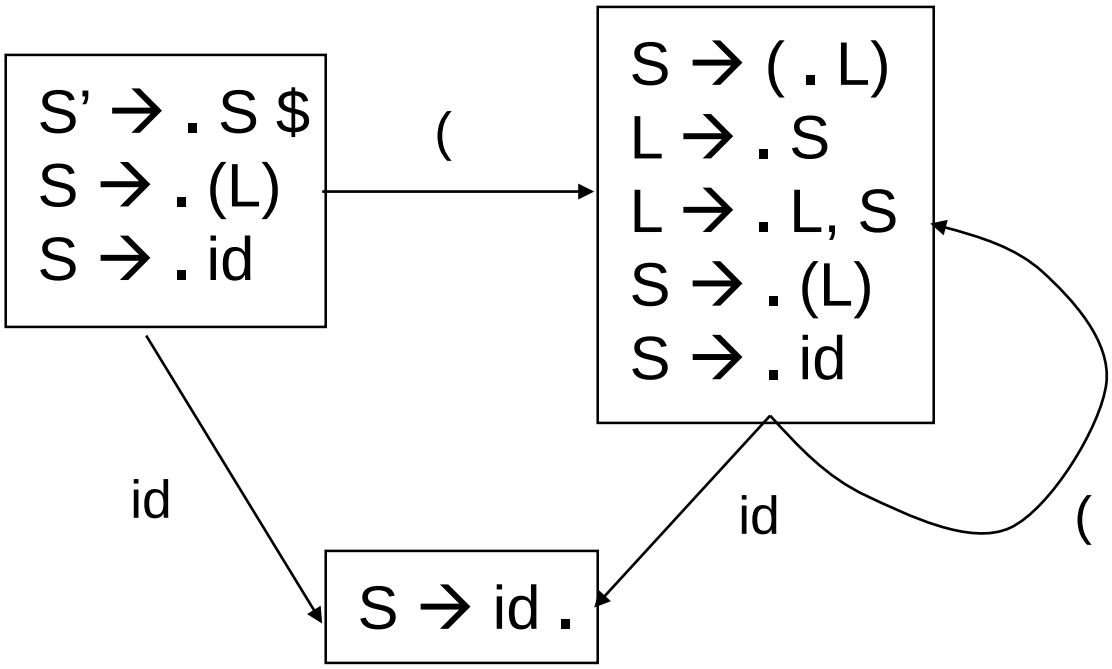
- Set of possible productions to be reduced next
- Closure of a parser state, S:
  - Start with Closure(S) = S
  - Then for each item in S:
    - X → α . Y β
    - Add items for all the productions Y → γ to the closure of S: Y → . γ

The Goto Operation

- **Goto operation = describes transitions between parser states, which are sets of items**
- **Algorithm: for state S and a symbol Y**
  - If the item [X → α . Y β] is in I, then
  - Goto(I, Y) = Closure([X → α Y . β])



Goto: Terminal Symbols



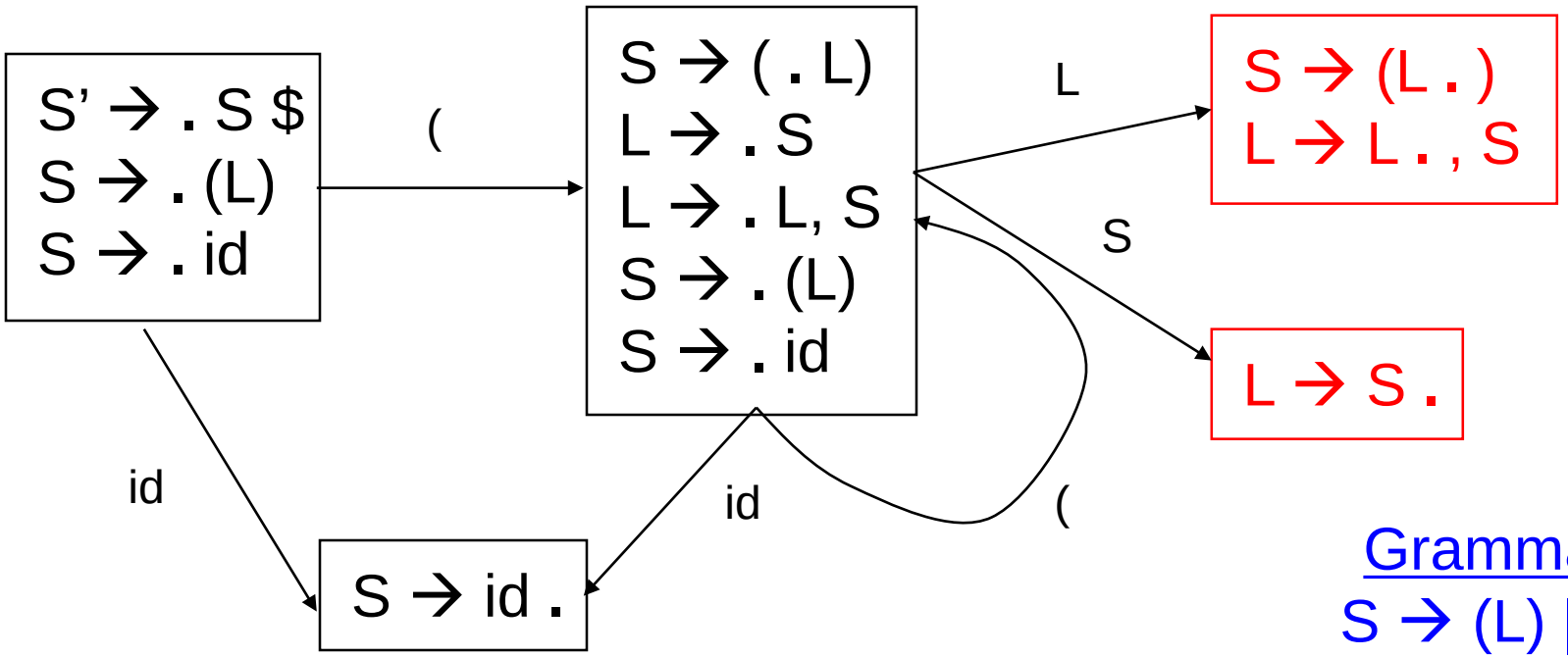
Grammar

S → (L) | id

L → S | L,S

In new state, include all items that have appropriate input symbol just after dot, advance do in those items and take closure

Goto: Non-terminal Symbols



Grammar

S → (L) | id

L → S | L,S

same algorithm for transitions on non-terminals

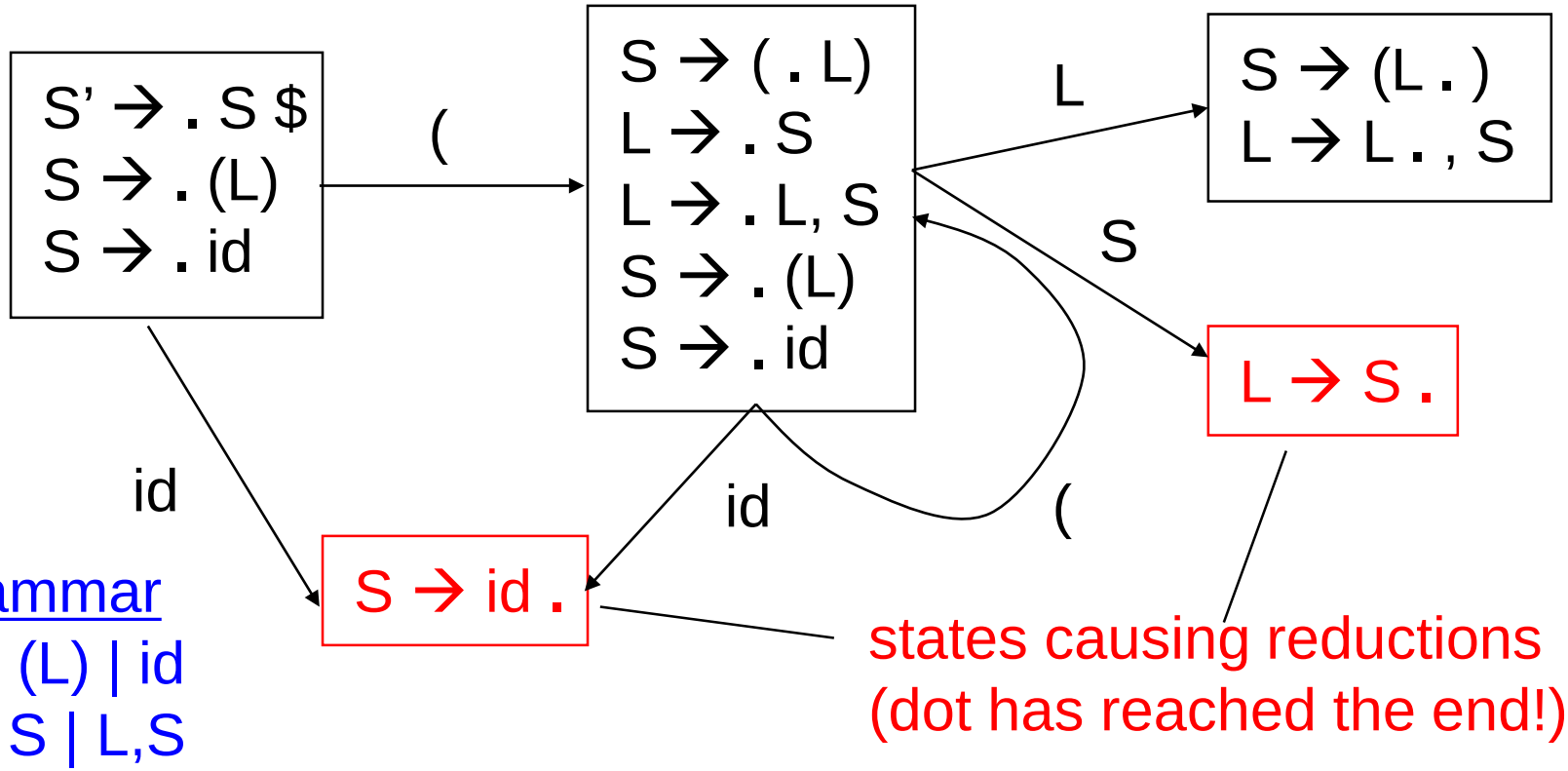
Class Problem

E' → E
E → E + T | T
T → T * F | F
F → (E) | id

1. If I = { [E' → **.** E]}, then Closure(I) = ??

2. If I = { [E' → E .], [E → E . + T] }, then Goto(I,+) = ??

Applying Reduce Actions



Grammar

S → (L) | id

L → S | L,S

Pop RHS off stack, replace with LHS X (X → β), then rerun DFA (e.g., (x))

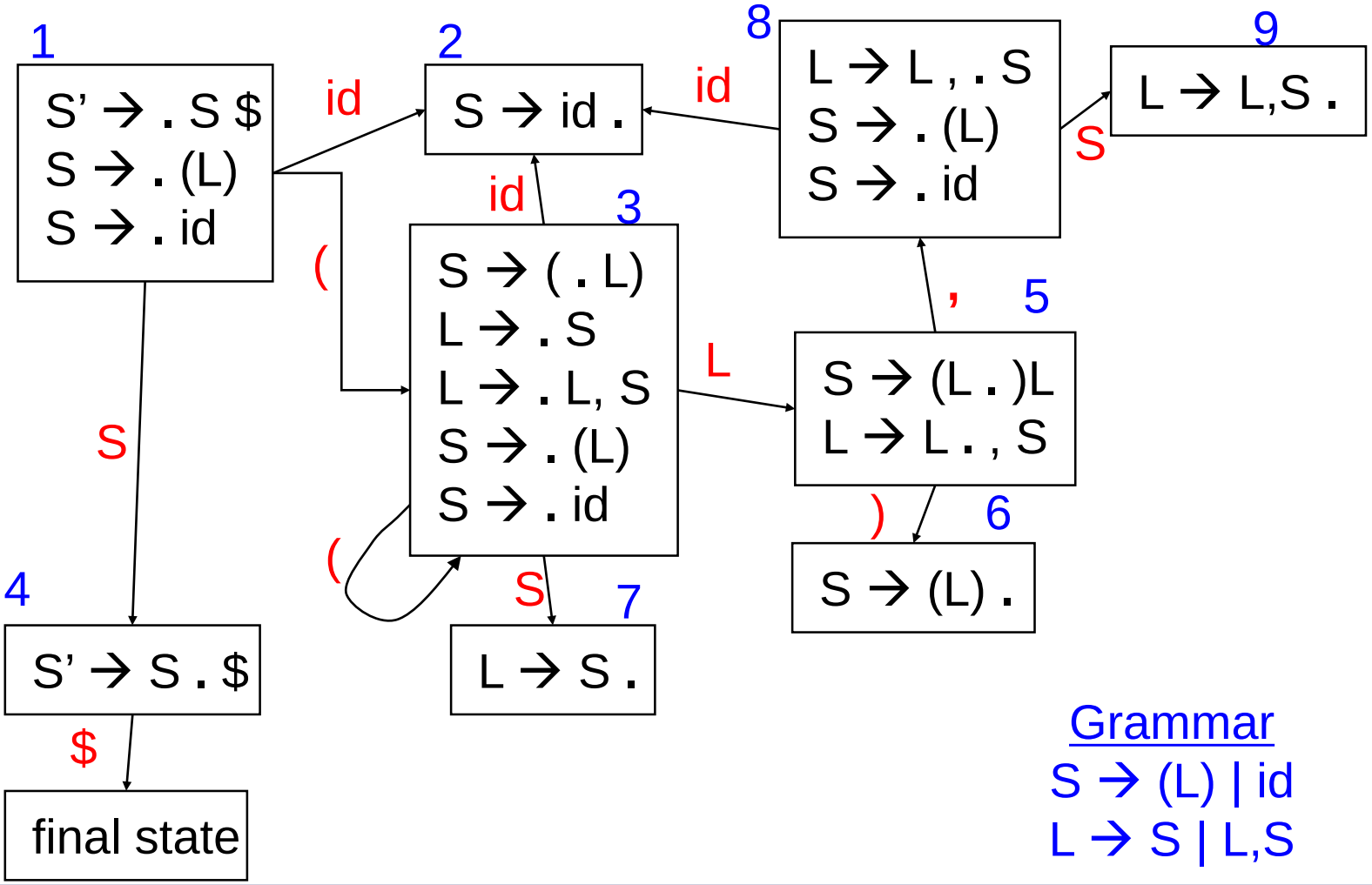
Reductions

- **On reducing X → β with stack αβ**
  - Pop β off stack, revealing prefix α and state
  - Take single step in DFA from top state
  - Push X onto stack with new DFA state

- **Example**

derivation	stack	input	action
((a),b) ←	1 (3 (3	a),b)	shift, goto 2
((a),b) ←	1 (3 (3 a 2	),b)	reduce S → id
((S),b) ←	1 (3 (3 S 7	),b)	reduce L → S

Full DFA



Grammar

S → (L) | id

L → S | L,S

Parsing Example ((a),b)

S → (L) | id
L → S | L,S

derivation	stack	input	action
((a),b) ←	1	((a),b)	shift, goto 3
((a),b) ←	1(3	(a),b)	shift, goto 3
((a),b) ←	1(3(3	a),b)	shift, goto 2
((a),b) ←	1(3(3a2	),b)	reduce S→id
((S),b) ←	1(3(3S7	),b)	reduce L→S
((L),b) ←	1(3(3L5	),b)	shift, goto 6
((L),b) ←	1(3(3L5)6	,b)	reduce S→(L)
(S,b) ←	1(3S7	,b)	reduce L→S
(L,b) ←	1(3L5	,b)	shift, goto 8
(L,b) ←	1(3L5,8	b)	shift, goto 9
(L,b) ←	1(3L5,8b2	)	reduce S→id
(L,S) ←	1(3L8,S9	)	reduce L→L,S
(L) ←	1(3L5	)	shift, goto 6
(L) ←	1(3L5)6		reduce S→(L)
S ←	1S4	$	done

Building the Parsing Table

- **States in the table = states in the DFA**
- **For transition S → S' on terminal C:**
  - Table[S,C] += Shift(S')
- **For transition S → S' on non-terminal N:**
  - Table[S,N] += Goto(S')
- **If S is a reduction state X → β then:**
  - Table[S,*] += Reduce(X → β)

Computed LR Parsing Table

State	Input terminal					Non-terminals	
	(	)	id	,	$	S	L
1	s3		s2			g4	
2	S→id	S→id	S→id	S→id	S→id		
3	s3		s2			g7	g5
4					accept		
5		s6		s8			
6	S→(L)	S→(L)	S→(L)	S→(L)	S→(L)		
7	L→S	L→S	L→S	L→S	L→S		
8	s3		s2			g9	
9	L→L,S	L→L,S	L→L,S	L→L,S	L→L,S		

blue = shift red = reduce

LR(0) Summary

- **LR(0) parsing recipe:**
  - Start with LR(0) grammar
  - Compute LR(0) states and build DFA:
    - Use the closure operation to compute states
    - Use the goto operation to compute transitions
  - Build the LR(0) parsing table from the DFA
- **This can be done automatically**

Class Problem

Generate the DFA for the following grammar

S → E + S | E
E → num

LR(0) Limitations

- **An LR(0) machine only works if states with reduce actions have a single reduce action**
  - Always reduce regardless of lookahead
- **With a more complex grammar, construction gives states with shift/reduce or reduce/reduce conflicts**
- **Need to use lookahead to choose**

OK

$L \rightarrow L , S .$

shift/reduce

$L \rightarrow L , S .$
$S \rightarrow S . , L$

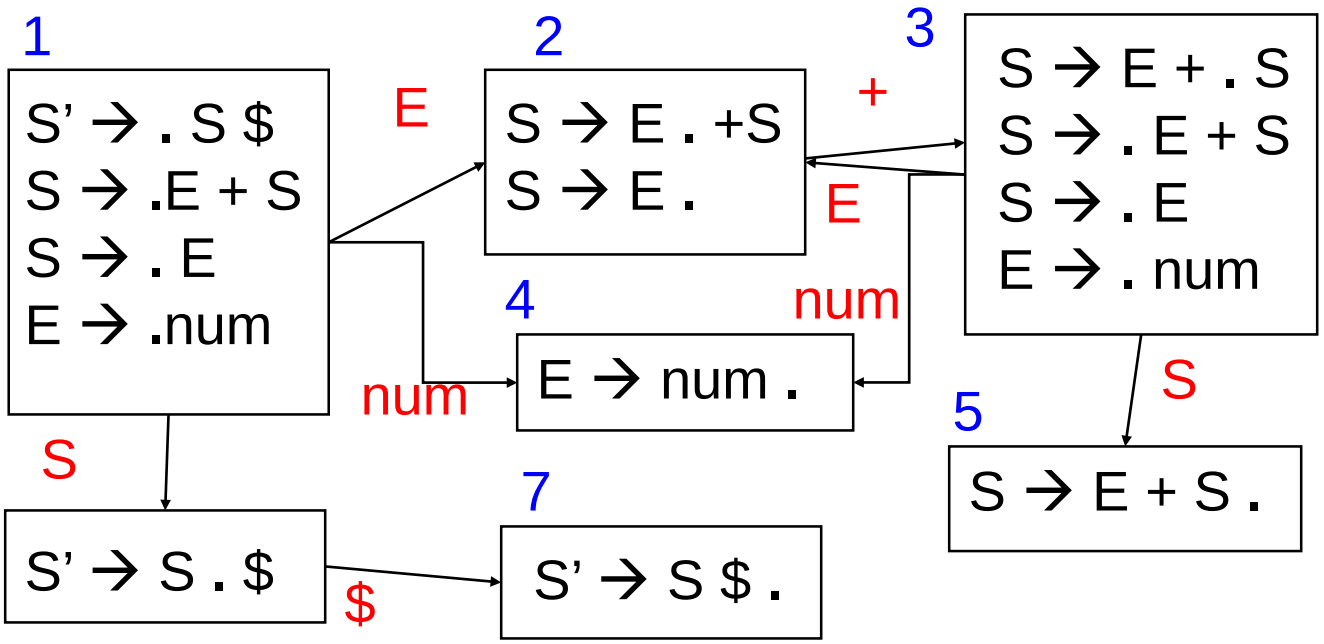
reduce/reduce

$L \rightarrow S , L .$
$L \rightarrow S .$

A Non-LR(0) Grammar

- **Grammar for addition of numbers**
  - S → S + E | E
  - E → num
- **Left-associative version is LR(0)**
- **Right-associative is not LR(0) as you saw with the previous class problem**
  - S → E + S | E
  - E → num

LR(0) Parsing Table



Grammar

S → E + S | E

E → num

Shift or reduce in state 2?

	num	+	$	E	S
1	s4			g2	g6
2	S→E	s3/S→E	S→E		

Solve Conflict With Lookahead

- **3 popular techniques for employing lookahead of 1 symbol with bottom-up parsing**
  - SLR – Simple LR
  - LALR – LookAhead LR
  - LR(1)
- **Each as a different means of utilizing the lookahead**
  - Results in different processing capabilities

SLR Parsing

- **SLR Parsing = Easy extension of LR(0)**
  - For each reduction X → β, look at next symbol C
  - Apply reduction only if C is in FOLLOW(X)
- **SLR parsing table eliminates some conflicts**
  - Same as LR(0) table except reduction rows
  - Adds reductions X → β only in the columns of symbols in FOLLOW(X)

Example: FOLLOW(S) = {$}

Grammar
S → E + S | E
E → num

	num	+	$	E	S
1	s4			g2	g6
2		s3	S→E		

SLR Parsing Table

- **Reductions do not fill entire rows as before**
- **Otherwise, same as LR(0)**

Grammar
S → E + S | E
E → num

	num	+	$	E	S
1	s4			g2	g6
2		s3	S→E		
3	s4			g2	g5
4		E→num	E→num		
5			S→E+S		
6			s7		
7			accept		

Class Problem

Consider:

S → L = R
S → R
L → *R
L → ident
R → L

Think of L as l-value, R as r-value, and * as a pointer dereference

When you create the states in the SLR(1) DFA, 2 of the states are the following:

S → L . = R R → L .	S → R .

Do you have any shift/reduce conflicts? (Not as easy as it looks)

LR(1) Parsing

- **Get as much as possible out of 1 lookahead symbol parsing table**
- **LR(1) grammar = recognizable by a shift/reduce parser with 1 lookahead**
- **LR(1) parsing uses similar concepts as LR(0)**
  - Parser states = set of items
  - LR(1) item = LR(0) item + lookahead symbol possibly following production
    - LR(0) item: S → . S + E
    - LR(1) item: S → . S + E , +
    - Lookahead only has impact upon REDUCE operations, apply when lookahead = next input

LR(1) States

- **LR(1) state = set of LR(1) items**
- **LR(1) item = (X → α . β , y)**
  - Meaning: α already matched at top of the stack, next expect to see β y
- **Shorthand notation**
  - (X → α . β , {x1, …, xn})
  - means:
    - (X → α . β , x1)
    - . . .
    - (X → α . β , xn)

S → S . + E +,$
S → S + . E num

- **Need to extend closure and goto operations**

LR(1) Closure

- **LR(1) closure operation:**
  - Start with Closure(S) = S
  - For each item in S:
    - X → α . Y β , z
    - and for each production Y → γ , add the following item to the closure of S: Y → . γ , FIRST(βz)
  - Repeat until nothing changes
- **Similar to LR(0) closure, but also keeps track of lookahead symbol**

LR(1) Start State

- **Initial state: start with (S' → . S , $), then apply closure operation**
- **Example: sum grammar**

S' → S $
S → E + S | E
E → num

S' → . S , $

S' → . S , $
S → . E + S , $
S → . E , $
E → . num , +,$

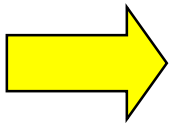
LR(1) Goto Operation

- **LR(1) goto operation = describes transitions between LR(1) states**
- **Algorithm: for a state S and a symbol Y (as before)**
  - If the item [X → α . Y β] is in I, then
  - Goto(I, Y) = Closure([X → α Y . β])

S1

S → E . + S , $
S → E . , $

Goto(S1, '+')



S2

Closure({S → E + . S , $})

Grammar:
S' → S$
S → E + S | E
E → num

Class Problem

1. Compute: Closure(I = {S → E + . S , $})
2. Compute: Goto(I, num)
3. Compute: Goto(I, E)

S' → S $
S → E + S | E
E → num