CS443: Compiler Construction

Lecture 2: Parsing Algorithms

Recursive descent parsing – simple algorithm for simple grammars

```
parse(A : nonterminal):
  for each production p of A:
    for each terminal/nonterminal in p:
      parse(p)
```

Needs backtracking

Examples:

S ::= a S | b S | ε   Input 1: aaaa   Input 2: aba

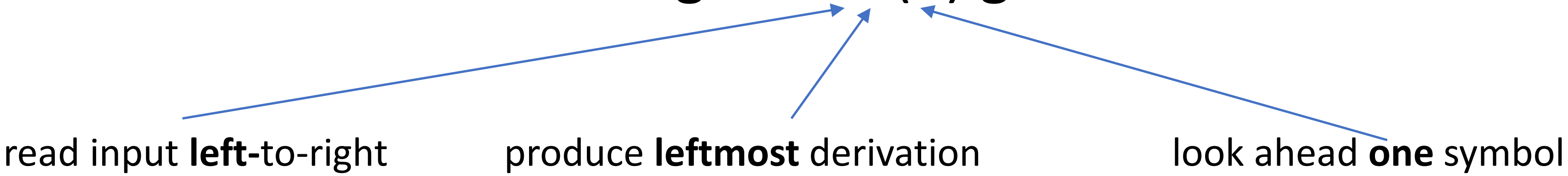
S ::= *S* a | *S* b | ε

Uh oh, left recursion

Predictive parsers make decision based on next input symbol

S ::= a S | b S | ε   Input: aba

- Works without backtracking for “LL(1) grammars”



read input **left-**to-right

produce **leftmost** derivation

look ahead **one** symbol

See Appel, PDB for algorithms for:

- Getting rid of left recursion
- Determining if a grammar is LL(1)
- Building a predictive parser for LL(1) grammars

Shift-reduce parsers work for LR(k) grammars

- Two actions
  - **shift** a symbol onto a **stack**
  - **reduce** some symbols from the top of the stack into a nonterminal

Shift-reduce parsers work for LR(k) grammars

S ::= e$ *e* ::= n | e + n | *e* + (*e*)

end of string

Stack	Input	
	1 + (2 + 3)$	
1	+ (2 + 3)$	Shift
e	+ (2 + 3)$	Reduce n -> e
e +	(2 + 3)$	Shift
e + (	2 + 3)$	Shift
e + (2	+ 3)$	Shift
e + (e	+ 3)$	Reduce n -> e
e + (e +	3)$	Shift
e + (e + 3	)$	Shift
e + (e	)$	Reduce e + n -> e
e + (e)	$	Shift
e	$	Reduce e + (e) -> e
S	$	Reduce e -> S, accept

Parse Tree

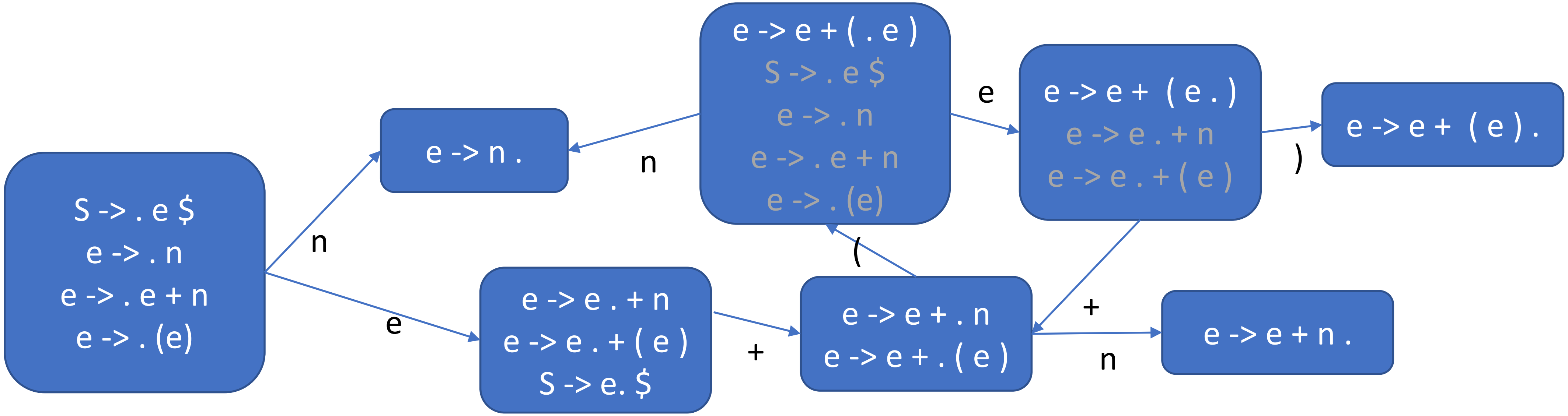
Shift-reduce parsers make decisions based on DFAs

- Edges: terminals + nonterminals on stack
- Just treat the stack as a stack of states (can reconstruct orig. stack)
- Transition table has two parts:
    - ACTION(state, terminal)
        - s*n* – shift state *n* onto stack
        - r*n* – reduce using rule *n*
        - a – accept
        - error (leave the table blank
    - GOTO(state, nonterminal)
        - next state

Building the DFA (see books for details)

- Items: Productions with a . indicating where we are
  - e.g. e -> e + . n
- DFA states = sets of items

S -> . e $
S -> e . $
e -> . n
e -> n .
e -> . e + n
e -> e . + n
e -> e + . n
e -> e + n .
e -> . e + (e)
e -> e . + (e)
e -> e + . (e)
e -> e + (. e)
e -> e + (e .)
e -> e + (e) .



S -> . e $
e -> . n
e -> . e + n
e -> . (e)

e -> n .

e -> e . + n
e -> e . + (e)
S -> e. $

e -> e + (. e)
S -> . e $
e -> . n
e -> . e + n
e -> . (e)

e -> e + . n
e -> e + . (e)

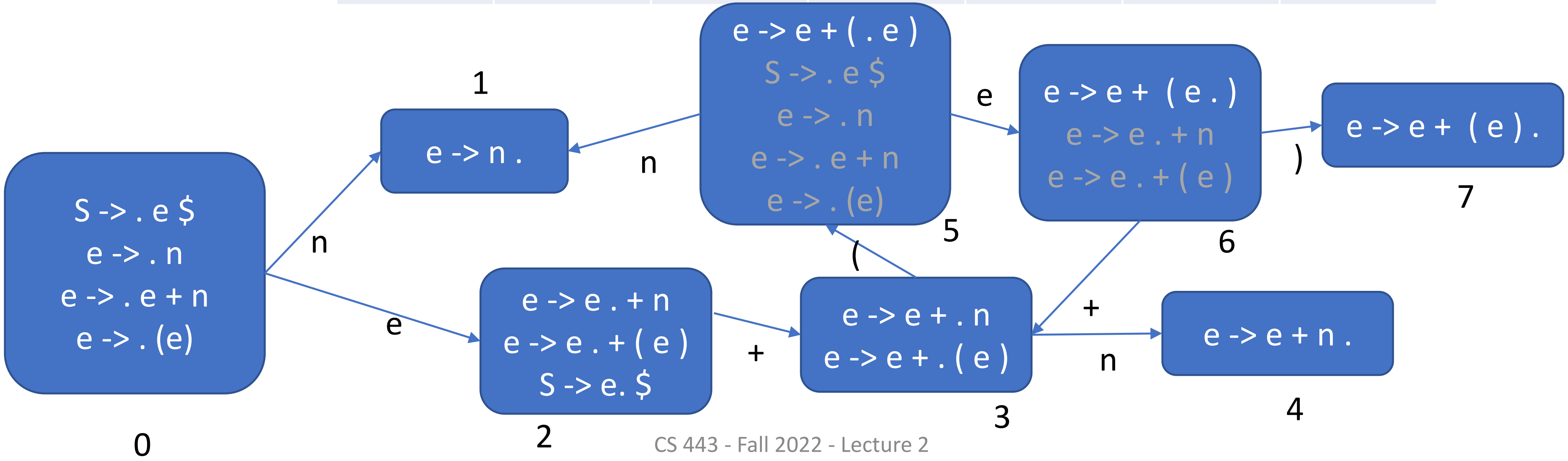
e -> e + (e .)
e -> e . + n
e -> e . + (e)

e -> e + (e) .

e -> e + n .

0 S -> e $
1 e -> n
2 e -> e + n
3 e -> e + (e)

	n	+	(	)	$	e
0	s1					2
1	r1	r1	r1	r1	r1	
2		s3			a	
3	s4		s5			
4	r2	r2	r2	r2	r2	
5	s1					6
6		s3		s7		
7	r3	r3	r3	r3	r3	



S -> . e $
e -> . n
e -> . e + n
e -> . (e)

e -> n .

e -> e . + n
e -> e . + (e)
S -> e. $

e -> e + . n
e -> e + . (e)

e -> e + n .

e -> e + (. e)
S -> . e $
e -> . n
e -> . e + n
e -> . (e)

e -> e + (e .)
e -> e . + n
e -> e . + (e)

e -> e + (e) .