



CSCI 742 - Compiler Construction

Lecture 14

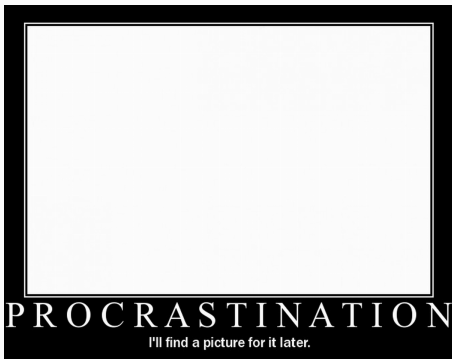
Introduction to Shift/Reduce Parsing

Instructor: Hossein Hojjat

February 16, 2018

LL vs LR

- $LL(k)$ parser predicts which production rule to use by looking at the next k tokens
- $LR(k)$ parser postpone the decision until it has seen input tokens of the entire right-hand side of the production rule
 - more powerful than $LL(k)$ parsers



- Left-to-right scan of input, **R**ightmost derivation
- Can handle left recursion and common prefixes
- As efficient as any top-down parsing
- Complex to implement
- Generally need automatic tools to construct parser from grammar

Example Grammar for LR Parsing

- Left-factoring is unnecessary for bottom-up parsers
- We can use a more “natural” grammar for additive expressions:

$$S \rightarrow E$$

$$E \rightarrow T \mid E + T$$

$$T \rightarrow \text{num} \mid (E)$$

- Consider the input word: `num + (num + num)`

Bottom-up Parsing for Example Grammar

- Bottom-up parsing **reduces** a word to the start symbol by inverting productions
- Productions rules from bottom to top create a rightmost derivation

$\text{num} + (\text{num} + \text{num})$	$T \rightarrow \text{num}$
$T + (\text{num} + \text{num})$	$E \rightarrow T$
$E + (\text{num} + \text{num})$	$T \rightarrow \text{num}$
$E + (T + \text{num})$	$E \rightarrow T$
$E + (E + \text{num})$	$T \rightarrow \text{num}$
$E + (E + T)$	$E \rightarrow E + T$
$E + (E)$	$T \rightarrow (E)$
$E + T$	$E \rightarrow E + T$
E	$S \rightarrow E$
S	

Useful Notation

- **Idea:** Split string into two substrings
- Right substring is still not examined by parser (a string of terminals)
- Left substring has both terminals and non-terminals
- The dividing point is marked by a |
- The | is not part of the string
- Initially, all input is unexamined $|a_1a_2 \cdots a_n$

Shift-Reduce Parsing

- A shift-reduce parser is a form of bottom-up parser whose primary operations are

Shift

$ABC \mid abc \Rightarrow ABCa \mid bc$

- Move $|$ one place to the right
- Shifts a terminal to the left string

Reduce

$Aabc \mid def \Rightarrow AaB \mid def$

- Apply an inverse production at the right end of the left string
- In the example we use production rule $B \rightarrow bc$

Shift-Reduce Parsing

- read (**shift**) tokens until the right hand side of “correct” production has been seen
- (**reduce**) the right hand side to non-terminal then continue
- done when all input read and reduced to start non-terminal

Shift-Reduce Parsing

| num + (num + num) shift

r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		

↑ num + (num + num)

Shift-Reduce Parsing

| num + (num + num) shift
num | + (num + num) reduce (r5)

r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		

num ↑ + (num + num)

Shift-Reduce Parsing

| num + (num + num) shift
num | + (num + num) reduce (r5)
T | + (num + num) reduce (r4)

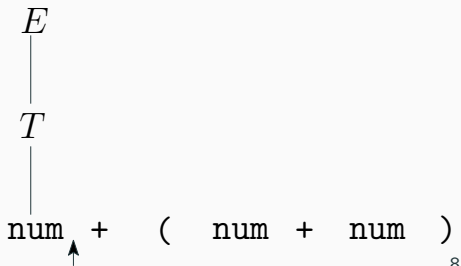
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		

T
|
num ↑ + (num + num)
8

Shift-Reduce Parsing

| num + (num + num) shift
num | + (num + num) reduce (r5)
T | + (num + num) reduce (r4)
E | + (num + num) shift

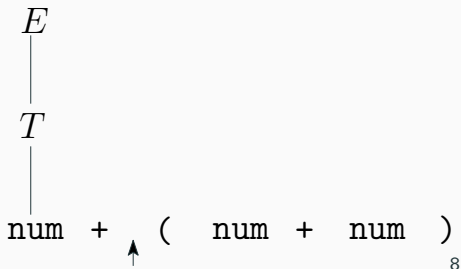
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

| num + (num + num) shift
num | + (num + num) reduce (r5)
T | + (num + num) reduce (r4)
E | + (num + num) shift
E + | (num + num) shift

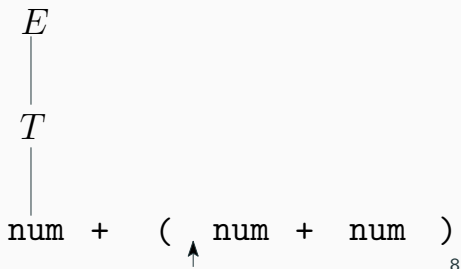
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

| num + (num + num) shift
num | + (num + num) reduce (r5)
T | + (num + num) reduce (r4)
E | + (num + num) shift
E + | (num + num) shift
E + (| num + num) shift

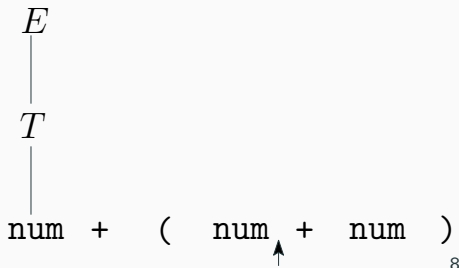
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

| num + (num + num) shift
num | + (num + num) reduce (r5)
T | + (num + num) reduce (r4)
E | + (num + num) shift
E + | (num + num) shift
E + (| num + num) shift
E + (num | + num) reduce (r5)

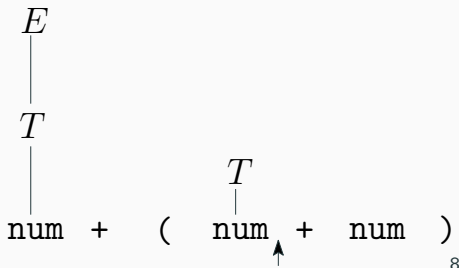
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

| num + (num + num) shift
num | + (num + num) reduce (r5)
T | + (num + num) reduce (r4)
E | + (num + num) shift
E + | (num + num) shift
E + (| num + num) shift
E + (num | + num) reduce (r5)
E + (*T* | + num) reduce (r4)

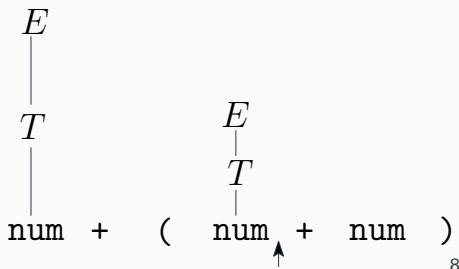
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

| num + (num + num) shift
num | + (num + num) reduce (r5)
T | + (num + num) reduce (r4)
E | + (num + num) shift
E + | (num + num) shift
E + (| num + num) shift
E + (num | + num) reduce (r5)
E + (*T* | + num) reduce (r4)
E + (*E* | + num) shift

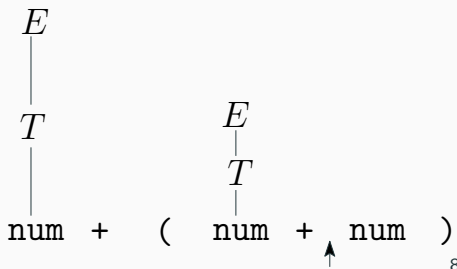
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

| num + (num + num) shift
num | + (num + num) reduce (r5)
T | + (num + num) reduce (r4)
E | + (num + num) shift
E + | (num + num) shift
E + (| num + num) shift
E + (num | + num) reduce (r5)
E + (*T* | + num) reduce (r4)
E + (*E* | + num) shift
E + (*E* + | num) shift

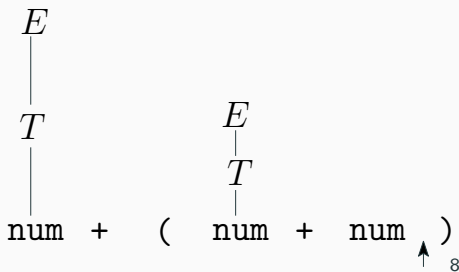
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

num + (num + num)	shift
num + (num + num)	reduce (r5)
T + (num + num)	reduce (r4)
E + (num + num)	shift
E + (num + num)	shift
E + (num + num)	shift
E + (num + num)	reduce (r5)
E + (T + num)	reduce (r4)
E + (E + num)	shift
E + (E + num)	shift
E + (E + num)	reduce (r5)

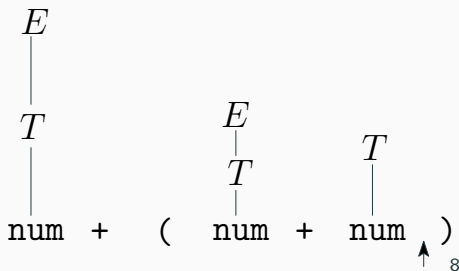
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

num + (num + num)	shift
num + (num + num)	reduce (r5)
T + (num + num)	reduce (r4)
E + (num + num)	shift
E + (num + num)	shift
E + (num + num)	shift
E + (num + num)	reduce (r5)
E + (T + num)	reduce (r4)
E + (E + num)	shift
E + (E + num)	shift
E + (E + num)	reduce (r5)
E + (E + T)	reduce (r2)

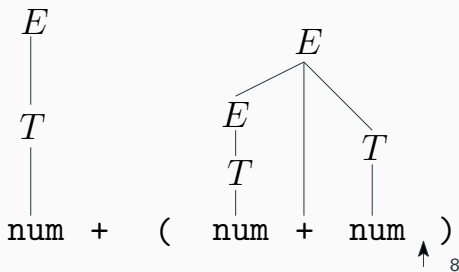
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

num + (num + num)	shift
num + (num + num)	reduce (r5)
T + (num + num)	reduce (r4)
E + (num + num)	shift
$E +$ (num + num)	shift
$E + ($ num + num)	shift
$E + (num$ + num)	reduce (r5)
$E + (T$ + num)	reduce (r4)
$E + (E$ + num)	shift
$E + (E +$ num)	shift
$E + (E + num$)	reduce (r5)
$E + (E + T$)	reduce (r2)
$E + (E$)	shift

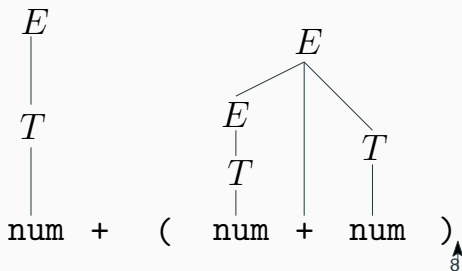
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

num + (num + num)	shift
num + (num + num)	reduce (r5)
<i>T</i> + (num + num)	reduce (r4)
<i>E</i> + (num + num)	shift
<i>E</i> + (num + num)	shift
<i>E</i> + (num + num)	shift
<i>E</i> + (num + num)	reduce (r5)
<i>E</i> + (<i>T</i> + num)	reduce (r4)
<i>E</i> + (<i>E</i> + num)	shift
<i>E</i> + (<i>E</i> + num)	shift
<i>E</i> + (<i>E</i> + num)	reduce (r5)
<i>E</i> + (<i>E</i> + <i>T</i>)	reduce (r2)
<i>E</i> + (<i>E</i>)	shift
<i>E</i> + (<i>E</i>)	reduce (r3)

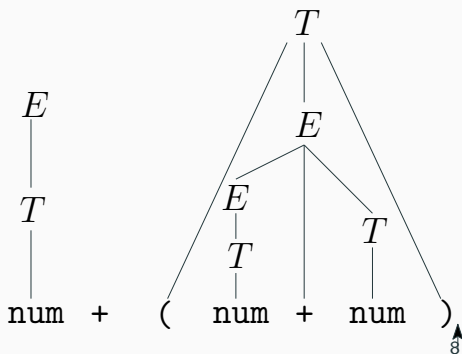
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

num + (num + num)	shift
num + (num + num)	reduce (r5)
T + (num + num)	reduce (r4)
E + (num + num)	shift
$E +$ (num + num)	shift
$E + ($ num + num)	shift
$E + (num$ + num)	reduce (r5)
$E + (T$ + num)	reduce (r4)
$E + (E$ + num)	shift
$E + (E +$ num)	shift
$E + (E + num$)	reduce (r5)
$E + (E + T$)	reduce (r2)
$E + (E$)	shift
$E + (E)$	reduce (r3)
$E + T$	reduce (r2)

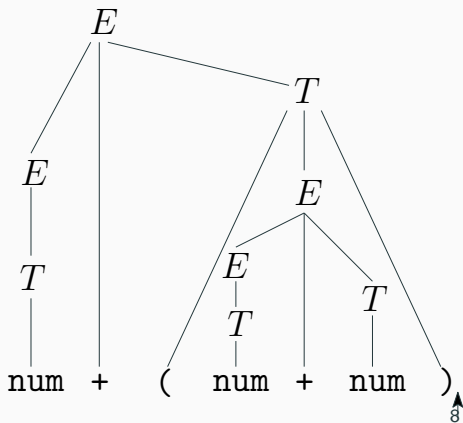
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

num + (num + num)	shift
num + (num + num)	reduce (r5)
T + (num + num)	reduce (r4)
E + (num + num)	shift
$E +$ (num + num)	shift
$E + ($ num + num)	shift
$E + (num$ + num)	reduce (r5)
$E + (T$ + num)	reduce (r4)
$E + (E$ + num)	shift
$E + (E +$ num)	shift
$E + (E + num$)	reduce (r5)
$E + (E + T$)	reduce (r2)
$E + (E$)	shift
$E + (E)$	reduce (r3)
$E + T$	reduce (r2)
E	reduce (r1)

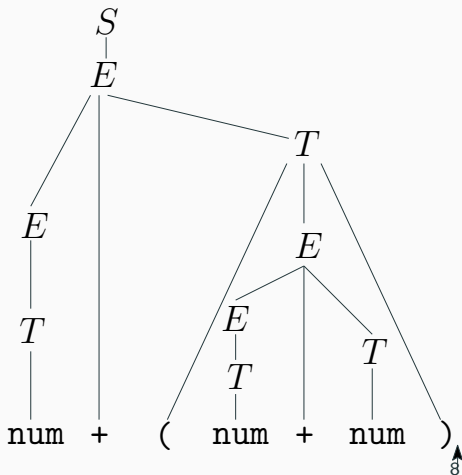
r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



Shift-Reduce Parsing

num + (num + num)	shift
num + (num + num)	reduce (r5)
T + (num + num)	reduce (r4)
E + (num + num)	shift
$E +$ (num + num)	shift
$E + ($ num + num)	shift
$E + (num$ + num)	reduce (r5)
$E + (T$ + num)	reduce (r4)
$E + (E$ + num)	shift
$E + (E +$ num)	shift
$E + (E + num$)	reduce (r5)
$E + (E + T$)	reduce (r2)
$E + (E$)	shift
$E + (E)$	reduce (r3)
$E + T$	reduce (r2)
E	reduce (r1)
S	

r1	$S \rightarrow E$	r4	$E \rightarrow T$
r2	$E \rightarrow E + T$	r5	$T \rightarrow \text{num}$
r3	$T \rightarrow (E)$		



- Left substring can be implemented by a stack
- Top of the stack is the |
- **Shift** pushes a terminal on the stack
- **Reduce** pops 0 or more symbols off of the stack (production RHS) and pushes a non-terminal on the stack (production LHS)

Shift-reduce Parsing with stack

derivation	stack	input stream	action
$\text{num} + (\text{num} + \text{num}) \leftarrow$		$\text{num} + (\text{num} + \text{num})$	shift
$\text{num} + (\text{num} + \text{num}) \leftarrow$	num	$+(\text{num} + \text{num})$	reduce $T \rightarrow \text{num}$
$T + (\text{num} + \text{num}) \leftarrow$	T	$+(\text{num} + \text{num})$	reduce $E \rightarrow T$
$E + (\text{num} + \text{num}) \leftarrow$	E	$+(\text{num} + \text{num})$	shift
$E + (\text{num} + \text{num}) \leftarrow$	$E +$	$(\text{num} + \text{num})$	shift
$E + (\text{num} + \text{num}) \leftarrow$	$E + ($	$\text{num} + \text{num})$	shift
$E + (\text{num} + \text{num}) \leftarrow$	$E + (\text{num}$	$+\text{num})$	reduce $T \rightarrow \text{num}$
$E + (T + \text{num}) \leftarrow$	$E + (T$	$+\text{num})$	reduce $E \rightarrow T$
$E + (E + \text{num}) \leftarrow$	$E + (E$	$+\text{num})$	shift
$E + (E + \text{num}) \leftarrow$	$E + (E +$	$\text{num})$	shift
$E + (E + \text{num}) \leftarrow$	$E + (E + \text{num}$)	reduce $E \rightarrow T$
$E + (E + T) \leftarrow$	$E + (E + T$)	reduce $E \rightarrow E + T$
$E + (E) \leftarrow$	$E + (E$)	shift
$E + (E) \leftarrow$	$E + (E)$		reduce $T \rightarrow (E)$
$E + T \leftarrow$	$E + T$		reduce $E \rightarrow E + T$
E	E		reduce $S \rightarrow E$ 10

Shift-Reduce Conflicts

- shift-reduce conflict:
when it is possible to both shift or reduce

Example:

$$S \rightarrow \text{if } E \text{ then } S$$
$$| \text{ if } E \text{ then } S \text{ else } S$$

- Consider step $\text{if } E \text{ then } S | \dots$
- If `else` follows then we can shift or reduce

Reduce-Reduce Conflicts

- reduce-reduce conflict:
when it is possible to reduce by two different productions

Example:

$$\text{r1} \quad S \rightarrow E$$

$$\text{r2} \quad E \rightarrow E + T$$

$$\text{r3} \quad T \rightarrow (E)$$

$$\text{r4} \quad E \rightarrow T$$

$$\text{r5} \quad T \rightarrow \text{num}$$

- Consider step $E + T \mid$
- We can reduce by $E \rightarrow T$ giving $E + E$
- A fatal mistake! No way to reduce to the start symbol anymore

Problem

- How do we know which action to take: shift or reduce, which production?
- Sometimes can reduce but should not
 - For example $X \rightarrow \epsilon$ can always be reduced
- Sometimes can reduce in more than one way

Action Selection Problem

- Given stack σ and look-ahead symbol b , should parser:
- **shift** b onto the stack (making it σb)
- **reduce** some production $X \rightarrow \gamma$ assuming stack has the form $\alpha\gamma$ (making it αX)

- Use a set of parser states
- Use a stack of states
- Use a parsing table to:
 - Determine what action to apply (shift/reduce)
 - Determine the next state
- The parser actions can be precisely determined from the table