Tutorial 02: CS31003: Compilers:

[M-03] Syntax Analysis or Parsing

Indranil Sengupta
Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

isg@iitkgp.ac.in
ppd@cse.iitkgp.ac.in

September 19, 2020

Doubts from the Week

Problem: Infix → Postfix

Convert

$$10 + 3 * 5/(16 - 4)$$

to postfix and evaluate

Solution: Infix → Postfix

Given : $10 + 3 * 5/(16 - 4)$

Current Symbol	Operator Stack	Postfix String
10		10
+	+	10
3	+	10 3
*	+ *	10 3
5	+ *	10 3 5
/	+ /	10 3 5 *
(	+ / (	10 3 5 *
16	+ / (	10 3 5 * 16
-	+ / (-	10 3 5 * 16
4	+ / (-	10 3 5 * 16 4
)	+ /	10 3 5 * 16 4 -
		10 3 5 * 16 4 - / +

Solution: Postfix Evaluation

Given : $10\ 3\ 5 * 16\ 4 - / +$

Current Symbol	**Stack content**	**Operation**
10	10	
3	10 3	
5	10 3 5	
*	10 15	3 * 5 = 15
16	10 15 16	
4	10 15 16 4	
-	10 15 12	16 - 4 = 12
/	10 1.25	15 / 12 = 1.25
+	11.25	10 + 1.25 = 11.25
	11.25	

Problem: Grammar and Derivation

Given $G = <\{\textbf{id}, +, *, (,)\}, \{E, T, F\}, E, P>$ where P is:

$$
\begin{array}{llll}
1: & E & \rightarrow & E + T \\
2: & E & \rightarrow & T \\
3: & T & \rightarrow & T * F \\
4: & T & \rightarrow & F \\
5: & F & \rightarrow & (E) \\
6: & F & \rightarrow & \textbf{id}
\end{array}
$$

Write the Left-most and Right-most derivations for the sentence
(id + id) * id

Solution: Grammar and Derivation

Left-most derivation for: $\mathbf{(id + id) * id}$

$$
\begin{array}{lllll}
E\ \$ & \Rightarrow & \underline{T}\ \$ & \Rightarrow & \underline{T} * F\ \$ \\
 & \Rightarrow & \underline{F} * F\ \$ & \Rightarrow & \underline{(E)} * F\ \$ \\
 & \Rightarrow & (\ \underline{E} + T) * F\ \$ & \Rightarrow & (\ \underline{T} + T) * F\ \$ \\
 & \Rightarrow & (\ \underline{F} + T) * F\ \$ & \Rightarrow & (\ \mathbf{id} + \underline{T}\ ) * F\ \$ \\
 & \Rightarrow & (\ \mathbf{id} + \underline{F}\ ) * F\ \$ & \Rightarrow & (\ \mathbf{id} + \mathbf{id}\ ) * \underline{F}\ \$ \\
 & \Rightarrow & (\ \mathbf{id} + \mathbf{id}\ ) * \mathbf{id}\ \$ & &
\end{array}
$$

Right-most derivation for: $\mathbf{(id + id) * id}$

$$
\begin{array}{lllll}
E\ \$ & \Rightarrow & \underline{T}\ \$ & \Rightarrow & T * \underline{F}\ \$ \\
 & \Rightarrow & \underline{T} * \mathbf{id}\ \$ & \Rightarrow & \underline{F} * \mathbf{id}\ \$ \\
 & \Rightarrow & \underline{(E)} * \mathbf{id}\ \$ & \Rightarrow & (E + \underline{T}\ ) * \mathbf{id}\ \$ \\
 & \Rightarrow & (E + \underline{F}\ ) * \mathbf{id}\ \$ & \Rightarrow & (\ \underline{E} + \mathbf{id}\ ) * \mathbf{id}\ \$ \\
 & \Rightarrow & (\ \underline{T} + \mathbf{id}\ ) * \mathbf{id}\ \$ & \Rightarrow & (\ \underline{F} + \mathbf{id}\ ) * \mathbf{id}\ \$ \\
 & \Rightarrow & (\ \mathbf{id} + \mathbf{id}\ ) * \mathbf{id}\ \$ & &
\end{array}
$$

Problem: Recursive Descent Parser

Write recursive descent parsers for the following grammar:

$$\begin{array}{lcl} A & \rightarrow & id \\ A & \rightarrow & (\ B\ ) \\ B & \rightarrow & int \\ B & \rightarrow & A \end{array}$$

Solution: Recursive Descent Parser

```
int main() {
    l = getchar();  // l is lookahead
    A();            // A is a start symbol
    // If l = $, it represents the end of the string
    if (l == ‘$’) printf("Parsing Successful");
    else printf("Error");
}
A() { // Definition of A, as per the given production
    if ( l == ’i’ ){
        match(’i’); match(’d’);        // A -> id
    }
    else if ( l == ’(’ ){
        match(’(’); B(); match(’)’);   // A-> ( B )
    }
}
B() { // Definition of B as per the given production
    if ( l == ’i’ ){
        match(’i’);
        if ( l == ’d’ ){        // This follows B -> A -> id production
            match(’d’);
        } else if ( l == ’n’ ){ // This is for normal B -> int
            match(’n’); match(’t’);
        }
    } else A();                 // B -> A
}
match(char t) { // Match function
    if (l == t) l = getchar();
    else printf("Error");
}
```

Problem: Removing Left Recursion

1. Remove left-recursion in the following grammar and compare the parse trees for strings *caaba* and *dbbaa* before and after recursion removal:

$$S \to S\ a \mid S\ b \mid c \mid d$$

2. Remove left-recursion in the following grammar and compare the parse trees for string *abdabed* before and after recursion removal:

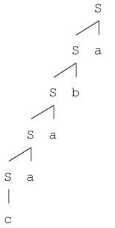
$$A \to A\ B\ d \mid A\ a \mid a$$

$$B \to B\ e \mid b$$

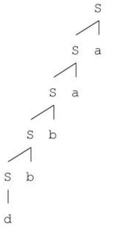
Solution: Removing Left Recursion

```
S -> Sa | Sb | c | d
```

```
S -> caaba
```



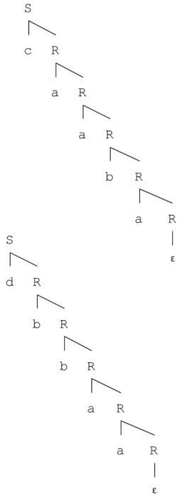
```
S -> dbbaa
```



```
S -> cR | dR
R -> aR | bR | ε
```

```
S -> caaba
```

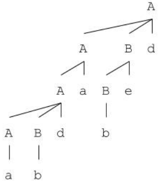
```
S -> dbbaa
```



Solution: Removing Left Recursion

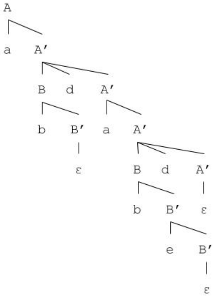
```
A -> ABd | Aa | a
   B -> Be | b
```

A -> abdabed



```
      A -> aA'
 A' -> BdA' | aA' | ε
      B -> bB'
   B' -> eB' | ε
```

A -> abdabed



Problem: Removing Ambiguity

Consider the ambiguous grammar G for regular expressions:

$$R \rightarrow R \ + \ R \mid R \ . \ R \mid R \ * \ \mid a \mid b$$

where $*$ is Kleen closure, . is concatenation, and $+$ is union operators. The priority order is:

$$(a, b) > * > . > +$$

where operators . and $+$ are both left associative.
Rewrite G as an equivalent unambiguous grammar.

Solution: Removing Ambiguity

Removing Ambiguity:

$$\begin{array}{llll} 1: & E & \rightarrow & E + C \mid C \\ 2: & C & \rightarrow & C \,.\, K \mid K \\ 3: & K & \rightarrow & K * F \mid F \\ 4: & F & \rightarrow & a \mid b \end{array}$$

Removing left-recursion:

$$\begin{array}{llll} 1: & E & \rightarrow & CE' \\ 2: & E' & \rightarrow & +CE' \mid \epsilon \\ 3: & C & \rightarrow & KC' \\ 4: & C' & \rightarrow & .KC' \mid \epsilon \\ 5: & K & \rightarrow & FK' \\ 6: & K' & \rightarrow & *FK' \mid \epsilon \\ 7: & F & \rightarrow & a \mid b \end{array}$$

Problem: Shift-Reduce Parsing

For grammar G_1:

$$\begin{array}{llcl}
1: & E & \rightarrow & E + T \\
2: & E & \rightarrow & T \\
3: & T & \rightarrow & T * F \\
4: & T & \rightarrow & F \\
5: & F & \rightarrow & (E) \\
6: & F & \rightarrow & \textbf{id}
\end{array}$$

Parse the following strings using the SR Parsing Table:

1. **id + id * id**
2. **id + * id**

Solution: Shift-Reduce Parsing: Parse Table

State	Action						GO TO		
	id	+	*	(	)	$	*E*	*T*	*F*
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Solution: Shift-Reduce Parsing

step	stack	symbols	Input	Act.
1	0		**id + id * id $**	s5
2	0 5	**id**	**+ id * id $**	r6
3	0 3	F	**+ id * id $**	r4
4	0 2	T	**+ id * id $**	r2
5	0 1	E	**+ id * id $**	s6
6	0 1 6	E +	**id * id $**	s5
7	0 1 6 5	E + **id**	*** id $**	r6
8	0 1 6 3	E + F	*** id $**	r4
9	0 1 6 9	E + T	*** id $**	s7
10	0 1 6 9 7	E + T *	**id $**	s5
11	0 1 6 9 7 5	E + T * **id**	**$**	r6
12	0 1 6 9 7 10	E + T * F	**$**	r3
13	0 1 6 9	E + T	**$**	r1
14	0 1	E	**$**	acc

Solution: Shift-Reduce Parsing

step	stack	symbols	Input	Act.
1	0		**id** + * **id $**	s5
2	0 5	**id**	+ * **id $**	r6
3	0 3	F	+ * **id $**	r4
4	0 2	T	+ * **id $**	r2
5	0 1	E	+ * **id $**	s6
6	0 1 6	E +	* **id $**	err

Practice Problems

1. Update the Operator Precedence Table to include the exponentiation operator $**$ where $**$ has higher precedence over $*$ and $/$, and is right-associative, then convert $5 * 3 * *(4 - 2)$ to postfix and evaluate it.

2. Given $P = \begin{array}{lcl} E & \rightarrow & E + T \mid T \\ T & \rightarrow & T \; \text{*} \; F \mid F \\ F & \rightarrow & (E) \mid \textbf{id} \end{array}$

   write the Left-most and Right-most derivations for **id * (id + id)**

3. Write recursive descent parsers for the following grammar:

   $$\begin{array}{lcl} L & \rightarrow & E \; R \\ E & \rightarrow & id \\ R & \rightarrow & ; \; L \mid \epsilon \end{array}$$

4. Remove left-recursion in the following grammar and compare the parse trees for string $(a, (a, a))$ before and after recursion removal:

   $$\begin{array}{lcl} S & \rightarrow & (\; L \;) \mid a \\ L & \rightarrow & L \; , \; S \mid S \end{array}$$

5. Given $P = \begin{array}{lcl} E & \rightarrow & E + T \mid T \\ T & \rightarrow & T \; \text{*} \; F \mid F \\ F & \rightarrow & (E) \mid \textbf{id} \end{array}$

   parse the following strings using the SR Parsing Table:

   1. **(id + id) * id**
   2. **id + id id**