

CS 515 Problem Set 2 Sample Solution

Problem 1

1. Canonical collection of sets of LR(1) items.

I_0 : $goal \rightarrow \cdot expr, \$$
 $expr \rightarrow \cdot term + expr, \$$
 $expr \rightarrow \cdot term, \$$
 $term \rightarrow \cdot factor * term, +/\$$
 $term \rightarrow \cdot factor, +/\$$
 $factor \rightarrow \cdot num, */ + /\$$

I_1 : $goal \rightarrow expr \cdot, \$$

I_2 : $expr \rightarrow term \cdot + expr, \$$
 $expr \rightarrow term \cdot, \$$

I_3 : $term \rightarrow factor \cdot * term, +/\$$
 $term \rightarrow factor \cdot, +/\$$

I_4 : $factor \rightarrow num \cdot, */ + /\$$

I_5 : $expr \rightarrow term + \cdot expr, \$$
 $expr \rightarrow \cdot term + expr, \$$
 $expr \rightarrow \cdot term, \$$
 $term \rightarrow \cdot factor * term, +/\$$
 $term \rightarrow \cdot factor, +/\$$
 $factor \rightarrow \cdot num, */ + /\$$

I_6 : $term \rightarrow factor * \cdot term, +/\$$
 $term \rightarrow \cdot factor * term, +/\$$
 $term \rightarrow \cdot factor, +/\$$
 $factor \rightarrow \cdot num, */ + /\$$

I_7 : $expr \rightarrow term + expr \cdot, \$$

I_8 : $term \rightarrow factor * term \cdot, +/\$$

2.

STATE	action				goto		
	+	*	num	\$	expr	term	factor
0			s4		1	2	3
1				acc			
2	s5			r3			
3	r5	s6		r5			
4	r6	r6		r6			
5			s4		7	2	3
6			s4			8	3
7				r2			
8	r4			r4			

3. Yes; there are no conflicts in the parse table.

4.

	STACK	INPUT	ACTION
1	0	num * num + num \$	shift
2	0 num 4	* num + num \$	r6
3	0 factor 3	* num + num \$	shift
4	0 factor 3 * 6	num + num \$	shift
5	0 factor 3 * 6 num 4	+ num \$	r6
6	0 factor 3 * 6 factor 3	+ num \$	r5
7	0 factor 3 * 6 term 8	+ num \$	r4
8	0 term 2	+ num \$	shift
9	0 term 2 + 5	num \$	shift
10	0 term 2 + 5 num 4		\$ r6
11	0 term 2 + 5 factor 3		\$ r5
12	0 term 2 + 5 term 2		\$ r3
13	0 term 2 + 5 expr 7		\$ r2
14	0 expr 1		\$ accept

Problem 2

LR(1) Construction

- $I_0: S' \rightarrow \cdot S, \$$
 $S \rightarrow \cdot Aa, \$$
 $S \rightarrow \cdot bAc, \$$
 $S \rightarrow \cdot dc, \$$
 $S \rightarrow \cdot bda, \$$
 $A \rightarrow \cdot d, a$
- $I_1: S' \rightarrow S \cdot, \$$
- $I_2: S \rightarrow A \cdot a, \$$
- $I_3: S \rightarrow b \cdot Ac, \$$
 $S \rightarrow b \cdot da, \$$
 $A \rightarrow \cdot d, c$
- $I_4: S \rightarrow d \cdot c, \$$
 $A \rightarrow d \cdot, a$
- $I_5: S \rightarrow Aa \cdot, \$$
- $I_6: S \rightarrow bA \cdot c, \$$
- $I_7: S \rightarrow bd \cdot a, \$$
 $A \rightarrow d \cdot, c$
- $I_8: S \rightarrow dc \cdot, \$$
- $I_9: S \rightarrow bAc \cdot, \$$
- $I_{10}: S \rightarrow bda \cdot, \$$

STATE	action					goto	
	a	b	c	d	\$	S	A
0		s3		s4		1	2
1					acc		
2	s5						
3				s7		6	
4	r6		s8				
5					r2		
6			s9				
7	s10	r6					
8					r4		
9					r3		
10					r5		

No conflicts in the parse table. Hence, the grammar is LR(1).

SLR(1) Construction

- $I_0: S' \rightarrow \cdot S$
 $S \rightarrow \cdot Aa$
 $S \rightarrow \cdot bAc$
 $S \rightarrow \cdot dc$
 $S \rightarrow \cdot bda$
 $A \rightarrow \cdot d$
- $I_1: S' \rightarrow S \cdot$
- $I_2: S \rightarrow A \cdot a$
- $I_3: S \rightarrow b \cdot Ac$
 $S \rightarrow b \cdot da$
 $A \rightarrow \cdot d$
- $I_4: S \rightarrow d \cdot c$
 $A \rightarrow d \cdot$
- $I_5: S \rightarrow Aa \cdot$
- $I_6: S \rightarrow bA \cdot c$
- $I_7: S \rightarrow bd \cdot a$
 $A \rightarrow d \cdot$
- $I_8: S \rightarrow dc \cdot$
- $I_9: S \rightarrow bAc \cdot$
- $I_{10}: S \rightarrow bda \cdot$

FOLLOW(S) = {\$}

FOLLOW(A) = {a, c}

STATE	action					goto	
	a	b	c	d	\$	S	A
0		s3		s4		1	2
1					acc		
2	s5						
3				s7		6	
4	r6		s8/r6				
5					r2		
6			s9				
7	s10/r6		r6				
8					r4		
9					r3		
10					r5		

There are two states {4, 7} with shift-reduce conflicts on inputs {c, a}, respectively. Hence, the grammar is not SLR(1).

Problem 3

In `scan.l`, under the rules section, add a regular expression for the “mod” token.

```
mod          {ECHO;return(MOD);}
```

In `parse.y`, under the definition section, add two directives; and under the rules section, extend the grammar rules for `exp`. First, add a declaration for the “mod” token. Second, add a precedence level along with associativity for “mod”. Associativity was unspecified, but precedence must be higher than other operators except exponentiation. Finally, the new grammar rule for “mod” is similar to that for “div”.

```
%token MOD

%left '+' '-'
%left '*' DIV
%left EXP MOD

...

%%

...

exp      : ...
         | ...
         | exp MOD exp {addop('mod');}
         | ...
         ;
```