1. **Shift-Reduce Parsing the Lambda Calculus.**

   We'll look again at the lambda calculus grammar:

   ```
   1. expr : var
   2.      | '(' 'λ' var '.' expr ')'
   3.      | '(' expr expr ')' ;
   4. var  : ID ;
   ```

   (a) Is this grammar LL(1)?

   Answer. No. FIRST(expr2) and FIRST(expr3) both contain '('.

   (b) We'll now use the following LR(1) parsing table to parse some strings with this grammar.

	(	)	.	ID	λ	$	var	expr
0	s1			s2			s4	s3
1	s1			s2	s5		s4	s6
2	r4	r4	r4	r4	r4	r4		
3						s7		
4	r1	r1	r1	r1	r1	r1		
5				s2			s8	
6	s1			s2			s4	s9
7	ACC	ACC	ACC	ACC	ACC	ACC		
8			s10					
9		s11						
10	s1			s2			s4	s12
11	r3	r3	r3	r3	r3	r3		
12		s13						
13	r2	r2	r2	r2	r2	r2		

   Here's how we use this parsing table:

   - Maintain a stack of states. Initialize it with state 0.
   - Use the state on top of the parse stack and the lookahead symbol to find the corresponding action in the parse table.

– If the action is a shift to a new state, push the new state onto the stack and advance the input.

– If the action is a reduce that reduces k symbols, pop k symbols off the stack. If we reduced to a rule A, consult the "A" entry in the state now at the top of the stack.

– If the action is ACC, then we have succeeded.

Use the parsing table above to parse the following:

i. (xx)

Answer.

Iteration	States on stack	Input	Actions
1	0	• (x x) $	s1
2	0 1	(• x x) $	s2
3	0 1 2	(x• x) $	r4
4	0 1 4	(var • x) $	r1
5	0 1 6	(expr • x) $	s2
6	0 1 6 2	(expr x •) $	r4
7	0 1 6 4	(expr var •) $	r1
8	0 1 6 9	(expr expr •) $	s11
9	0 1 6 9 11	(expr expr)• $	r3
10	0 3	expr • $	s7
11	0 3 7	expr $ •	ACC

ii. $(\lambda x.x)$

Answer.

Iteration	States on stack	Input	Actions
1	0	• (λ x . x) $	s1
2	0 1	(• λ x . x) $	s5
3	0 1 5	(λ • x . x) $	s2
4	0 1 5 2	(λ x • . x) $	r4
5	0 1 5 8	(λ var • . x) $	s10
6	0 1 5 8 10	(λ var . • x) $	s2
7	0 1 5 8 10 2	(λ var . x •) $	r4
8	0 1 5 8 10 4	(λ var . var •) $	r1
9	0 1 5 8 10 12	(λ var . expr •) $	s13
10	0 1 5 8 10 12 13	(λ var . expr) • $	r2
11	0 3	expr • $	s7
12	0 3 7	expr $ •	ACC

iii. $(x(\lambda x.x))$

Answer.

Iteration	States on stack	Input	Actions
1	0	$\bullet$ (x (λ x . x)) \$	s1
2	0 1	($\bullet$ x (λ x . x)) \$	s2
3	0 1 2	(x $\bullet$ (λ x . x)) \$	r4
4	0 1 4	(var $\bullet$ (λ x . x)) \$	r1
5	0 1 6	(expr $\bullet$ (λ x . x)) \$	s1
6	0 1 6 1	(expr ($\bullet$ λ x . x)) \$	s5
7	0 1 6 1 5	(expr (λ $\bullet$ x . x)) \$	s2
8	0 1 6 1 5 2	(expr (λ x $\bullet$. x)) \$	r4
9	0 1 6 1 5 8	(expr (λ var $\bullet$. x)) \$	s10
10	0 1 6 1 5 8 10	(expr (λ var .$\bullet$ x)) \$	s2
11	0 1 6 1 5 8 10 2	(expr (λ var .x$\bullet$)) \$	r4
12	0 1 6 1 5 8 10 4	(expr (λ var . var$\bullet$)) \$	r1
13	0 1 6 1 5 8 10 12	(expr (λ var . expr $\bullet$)) \$	s13
14	0 1 6 1 5 8 10 12 13	(expr (λ var . expr)$\bullet$) \$	r2
15	0 1 6 9	(expr expr $\bullet$) \$	s11
16	0 1 6 9 11	(expr expr)$\bullet$ \$	r3
17	0 3	expr $\bullet$ \$	s7
18	0 3 7	expr \$ $\bullet$	ACC

(c) Is this grammar LR(0)?

Answer. Yes. There is no need for a lookahead to disambiguate a shift versus reduce, nor to disambiguate between two reduces.

2. **Altering the Lambda Calculus.**

Suppose we want to add an optional extension that allows raising a `var` to a power. We define the grammar as

```
expr : var
     | var '^' NUM
     | '(' 'λ' var '.' expr ')'
     | '(' expr expr ')' ;
var  : ID ;
```

(a) Is this grammar LR(0)?

Answer. Nope. We don't know whether to reduce 'var' until we check for '^'. It is LR(1).

3

(b) Which state in the parsing table would we need to modify to parse this grammar?

   Answer. We would need to add an extra case for shifting on the lookahead symbol '^' in State 4.

3. **Stack in Shift-Reduce Parsing.**

Suppose it is given that shift-reduce parsing is equivalent to finding the rightmost derivation in reverse. Prove that during shift-reduce parsing, we can only reduce the topmost items in the stack (i.e. we don't need to worry about reducing something in the middle; hence the usage of a stack is justified).

Answer. Suppose the state of the stack is given by $\alpha\beta\gamma$, where the Greek letters are strings of 0 or more symbols and capital Roman letters are non-terminal symbols. Suppose we want to reduce β using the rule $B \to \beta$ when it is not at the top of the stack (i.e. $\gamma \neq \epsilon$). Then γ can be either 1) a string of terminals, or 2) $\gamma = \gamma'A\gamma''$, where A is a non-terminal. In Case 1, we could have reduced β before shifting the terminal symbols in γ, since the terminals in γ don't help in matching the right hand side of a rule. Case 2 is problematic, because shift-reduce parsing finds the rightmost derivation in reverse order, but

$$(\text{input}) \ \Longleftarrow\ \alpha\beta\gamma'A\gamma'' \ \Longleftarrow\ S$$

is not a rightmost derivation if we reduce β using the rule $B \to \beta$. Consequently, in both cases, a reduction in the middle of the stack cannot occur, and we can only reduce the topmost items in the stack.