Programming Language Principles

University of Florida Gainesville - February 28, 2014: HW1

Michael Kummer: 14218718

a)

	'|'	'cat'	'list'	'+'	'*'	'?'	'('	'a'	'b'	'c'
E0							E0 → E0 '|' E1 E0 → E1	E0 → E0 '|' E1 E0 → E1	E0 → E0 '|' E1 E0 → E1	E0 → E0 '|' E1 E0 → E1
E1							E1 → E2 E1 E1 → E2	E1 → E2 E1 E1 → E2	E1 → E2 E1 E1 → E2	E1 → E2 E1 E1 → E2
E2							E2 → E2 'list' E3 E2 → E3	E2 → E2 'list' E3 E2 → E3	E2 → E2 'list' E3 E2 → E3	E2 → E2 'list' E3 E2 → E3
E3							E3 → E4 '+' E3 → E4 '*' E3 → E4 '?' E3 → E4	E3 → E4 '+' E3 → E4 '*' E3 → E4 '?' E3 → E4	E3 → E4 '+' E3 → E4 '*' E3 → E4 '?' E3 → E4	E3 → E4 '+' E3 → E4 '*' E3 → E4 '?' E3 → E4
E4							E4 → '(' E0 ')'	E4 → 'a'	E4 → 'b'	E4 → 'c'

This is clearly not LL(1) because there is at least one case of multiple production rules within a cell. Left recursive always produces a non LL(1) grammar. E0 and E2 are both left recursive, therefore the grammar is not LL(1).

b)

Original Grammar:

E0 → E0 '|' E1
→ E1
E1 → E2 E1
→ E2
E2 → E2 'list' E3
→ E3
E3 → E4 '+'
→ E4 '*'
→ E4 '?'
→ E4
E4 → '(' E0 ')'
→ 'a'
→ 'b'
→ 'c'

LL(1) Grammar:

E0 → E1 Y3
Y3 → '|' E1 Y3
→
E1 → E2 Y2
Y2 → E1
→
E2 → E3 Y1
Y1 → 'list' E3 Y1
→
E3 → E4 Y0
Y0 → '+'
→ '*'
→ '?'
→
E4 → '(' E0 ')'
→ 'a'
→ 'b'
→ 'c'

c)

(next page)

	⊥	‘|’	‘cat’	‘list’	‘+’	‘*’	‘?’	‘(’	‘)’	‘a’	‘b’	‘c’
E0								E0 → E1 Y3		E0 → E1 Y3	E0 → E1 Y3	E0 → E1 Y3
Y3	Y3 →	Y3 → ‘|’ E1 Y3							Y3 →			
E1								E1 → E2 Y2		E1 → E2 Y2	E1 → E2 Y2	E1 → E2 Y2
Y2	Y2 →							Y2 → E1	Y2 →	Y2 → E1	Y2 → E1	Y2 → E1
E2								E2 → E2 Y1		E2 → E2 Y1	E2 → E2 Y1	E2 → E2 Y1
Y1	Y1 →			Y1 → ‘list’ E3 Y1					Y1 →			
E3								E3 → E4 Y0		E3 → E4 Y0	E3 → E4 Y0	E3 → E4 Y0
Y0	Y0 →				Y0 → ‘+’	Y0 → ‘*’	Y0 → ‘?’		Y0 →			
E4								E4 → ‘(’ E0 ‘)’		E4 → ‘a’	E4 → ‘b’	E4 → ‘c’

d)

Stack	Input	Table Lookup Value
E0	(a | b)* c? (b a)+ ⊥	E0 → E1 Y3
E1 Y3	(a | b)* c? (b a)+ ⊥	E1 → E2 Y2
E2 Y2 Y3	(a | b)* c? (b a)+ ⊥	E2 → E3 Y1
E3 Y1 Y2 Y3	(a | b)* c? (b a)+ ⊥	E3 → E4 Y0
E4 Y0 Y1 Y2 Y3	(a | b)* c? (b a)+ ⊥	E4 → '(' E0 ')'
'(' E0 ')' Y0 Y1 Y2 Y3	(a | b)* c? (b a)+ ⊥	
E0 ')' Y0 Y1 Y2 Y3	a | b)* c? (b a)+ ⊥	E0 → E1 Y3
E1 Y3 ')' Y0 Y1 Y2 Y3	a | b)* c? (b a)+ ⊥	E1 → E2 Y2
E2 Y2 Y3 ')' Y0 Y1 Y2 Y3	a | b)* c? (b a)+ ⊥	E2 → E3 Y1
E3 Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	a | b)* c? (b a)+ ⊥	E3 → E4 Y0
E4 Y0 Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	a | b)* c? (b a)+ ⊥	E4 → 'a'
'a' Y0 Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	| b)* c? (b a)+ ⊥	
Y0 Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	| b)* c? (b a)+ ⊥	Y0 →
Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	| b)* c? (b a)+ ⊥	Y1 →
Y2 Y3 ')' Y0 Y1 Y2 Y3	| b)* c? (b a)+ ⊥	Y2 →
Y3 ')' Y0 Y1 Y2 Y3	| b)* c? (b a)+ ⊥	Y3 → '|' E1 Y3
'|' E1 Y3 ')' Y0 Y1 Y2 Y3	| b)* c? (b a)+ ⊥	
E1 Y3 ')' Y0 Y1 Y2 Y3	b)* c? (b a)+ ⊥	E1 → E2 Y2
E2 Y2 Y3 ')' Y0 Y1 Y2 Y3	b)* c? (b a)+ ⊥	E2 → E3 Y1
E3 Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	b)* c? (b a)+ ⊥	E3 → E4 Y0
E4 Y0 Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	b)* c? (b a)+ ⊥	E4 → 'b'
'b' Y0 Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	b)* c? (b a)+ ⊥	
Y0 Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	)* c? (b a)+ ⊥	Y0 →
Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	)* c? (b a)+ ⊥	Y1 →
Y2 Y3 ')' Y0 Y1 Y2 Y3	)* c? (b a)+ ⊥	Y2 →
Y3 ')' Y0 Y1 Y2 Y3	)* c? (b a)+ ⊥	Y3 →
')' Y0 Y1 Y2 Y3	)* c? (b a)+ ⊥	
Y0 Y1 Y2 Y3	* c? (b a)+ ⊥	Y0 → '*'
'*' Y1 Y2 Y3	* c? (b a)+ ⊥	
Y1 Y2 Y3	c? (b a)+ ⊥	Y1 →
Y2 Y3	c? (b a)+ ⊥	Y2 → E1
E1 Y3	c? (b a)+ ⊥	E1 → E2 Y2
E2 Y2 Y3	c? (b a)+ ⊥	E2 → E3 Y1
E3 Y1 Y2 Y3	c? (b a)+ ⊥	E3 → E4 Y0
E4 Y0 Y1 Y2 Y3	c? (b a)+ ⊥	E4 → 'c'
'c' Y0 Y1 Y2 Y3	c? (b a)+ ⊥	
Y0 Y1 Y2 Y3	? (b a)+ ⊥	Y0 → '?'
'?' Y1 Y2 Y3	? (b a)+ ⊥	
Y1 Y2 Y3	(b a)+ ⊥	Y1 →
Y2 Y3	(b a)+ ⊥	Y2 → E1
E1 Y3	(b a)+ ⊥	E1 → E2 Y2
E2 Y2 Y3	(b a)+ ⊥	E2 → E3 Y1
E3 Y1 Y2 Y3	(b a)+ ⊥	E3 → E4 Y0
E4 Y0 Y1 Y2 Y3	(b a)+ ⊥	E4 → '(' E0 ')'
'(' E0 ')' Y0 Y1 Y2 Y3	(b a)+ ⊥	
E0 ')' Y0 Y1 Y2 Y3	b a)+ ⊥	E0 → E1 Y3
E1 Y3 ')' Y0 Y1 Y2 Y3	b a)+ ⊥	E1 → E2 Y2
E2 Y2 Y3 ')' Y0 Y1 Y2 Y3	b a)+ ⊥	E2 → E3 Y1
E3 Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	b a)+ ⊥	E3 → E4 Y0
E4 Y0 Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	b a)+ ⊥	E4 → 'b'
'b' Y0 Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	b a)+ ⊥	
Y0 Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	a)+ ⊥	Y0 →
Y1 Y2 Y3 ')' Y0 Y1 Y2 Y3	a)+ ⊥	Y1 →
Y2 Y3 ')' Y0 Y1 Y2 Y3	a)+ ⊥	Y2 → E1
E1 Y3 ')' Y0 Y1 Y2 Y3	a)+ ⊥	E1 → E2 Y2
E2 Y2 Y3 ')' Y0 Y1 Y2 Y3	a)+ ⊥	E2 → E3 Y1

E3 Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	a)+ ⊥	E3 → E4 Y0
E4 Y0 Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	a)+ ⊥	E4 → ‘a’
‘a’ Y0 Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	a)+ ⊥	
Y0 Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	)+ ⊥	Y0 →
Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	)+ ⊥	Y1 →
Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	)+ ⊥	Y2 →
Y3 ‘)’ Y0 Y1 Y2 Y3	)+ ⊥	Y3 →
‘)’ Y0 Y1 Y2 Y3	)+ ⊥	
Y0 Y1 Y2 Y3	+ ⊥	Y0 → ‘+’
‘+’ Y1 Y2 Y3	+ ⊥	
Y1 Y2 Y3	⊥	Y1 →
Y2 Y3	⊥	Y2 →
Y3	⊥	Y3 →
-	⊥	

Stack	Input	Table Lookup Value
E0	(a* | b)* list c a+ ⊥	E0 → E1 Y3
E1 Y3	(a* | b)* list c a+ ⊥	E1 → E2 Y2
E2 Y2 Y3	(a* | b)* list c a+ ⊥	E2 → E3 Y1
E3 Y1 Y2 Y3	(a* | b)* list c a+ ⊥	E3 → E4 Y0
E4 Y0 Y1 Y2 Y3	(a* | b)* list c a+ ⊥	E4 → ‘(’ E0 ‘)’
‘(’ E0 ‘)’ Y0 Y1 Y2 Y3	(a* | b)* list c a+ ⊥	
E0 ‘)’ Y0 Y1 Y2 Y3	a* | b)* list c a+ ⊥	E0 → E1 Y3
E1 Y3 ‘)’ Y0 Y1 Y2 Y3	a* | b)* list c a+ ⊥	E1 → E2 Y2
E2 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	a* | b)* list c a+ ⊥	E2 → E3 Y1
E3 Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	a* | b)* list c a+ ⊥	E3 → E4 Y0
E4 Y0 Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	a* | b)* list c a+ ⊥	E4 → ‘a’
‘a’ Y0 Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	a* | b)* list c a+ ⊥	
Y0 Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	* | b)* list c a+ ⊥	Y0 → ‘*’
‘*’ Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	* | b)* list c a+ ⊥	
Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	| b)* list c a+ ⊥	Y1 →
Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	| b)* list c a+ ⊥	Y2 →
Y3 ‘)’ Y0 Y1 Y2 Y3	| b)* list c a+ ⊥	Y3 → ‘|’ E1 Y3
‘|’ E1 Y3 ‘)’ Y0 Y1 Y2 Y3	| b)* list c a+ ⊥	
E1 Y3 ‘)’ Y0 Y1 Y2 Y3	b)* list c a+ ⊥	E1 → E2 Y2
E2 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	b)* list c a+ ⊥	E2 → E3 Y1
E3 Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	b)* list c a+ ⊥	E3 → E4 Y0
E4 Y0 Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	b)* list c a+ ⊥	E4 → ‘b’
‘b’ Y0 Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	b)* list c a+ ⊥	
Y0 Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	)* list c a+ ⊥	Y0 →
Y1 Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	)* list c a+ ⊥	Y1 →
Y2 Y3 ‘)’ Y0 Y1 Y2 Y3	)* list c a+ ⊥	Y2 →
Y3 ‘)’ Y0 Y1 Y2 Y3	)* list c a+ ⊥	Y3 →
‘)’ Y0 Y1 Y2 Y3	)* list c a+ ⊥	
Y0 Y1 Y2 Y3	* list c a+ ⊥	Y0 → ‘*’
‘*’ Y1 Y2 Y3	* list c a+ ⊥	
Y1 Y2 Y3	* list c a+ ⊥	Y1 → ‘list’ E3 Y1
‘list’ E3 Y1 Y2 Y3	list c a+ ⊥	
E3 Y1 Y2 Y3	c a+ ⊥	E3 → E4 Y0
E4 Y0 Y1 Y2 Y3	c a+ ⊥	E4 → ‘c’
‘c’ Y0 Y1 Y2 Y3	c a+ ⊥	
Y0 Y1 Y2 Y3	a+ ⊥	Y0 →
Y1 Y2 Y3	a+ ⊥	Y1 →
Y2 Y3	a+ ⊥	Y2 → E1
E1 Y3	a+ ⊥	E1 → E2 Y2
E2 Y2 Y3	a+ ⊥	E2 → E3 Y1

E3 Y1 Y2 Y3	a+ ⊥	E3 → E4 Y0
E4 Y0 Y2 Y3	a+ ⊥	E4 → ‘a’
‘a’ Y0 Y2 Y3	a+ ⊥	
Y0 Y2 Y3	+ ⊥	Y0 → ‘+’
‘+’ Y2 Y3	+ ⊥	
Y2 Y3	⊥	Y2 →
Y3	⊥	Y3 →
-	⊥	

e)

The following successfully parses the given inputs.

```
#include <stdio.h>
#include <iostream>
#include <string>
#include <string.h>
#include <sstream>
#include <stdlib.h>     /* exit, EXIT_FAILURE */

 using namespace std;

//std::string myData("( a * | b ) * list c a +");
std::string myData("( a | b ) * c ? ( b a ) +");
std::istringstream tokens(myData);
std::string NT;

void error();
void scan();
void read(std::string);
void write(std::string);

void E0();
void E1();
void E2();
void E3();
void E4();

void Y3();
void Y2();
void Y1();
void Y0();

void error(){
    cout << "Unexpected Token" << NT << endl;
    exit(EXIT_FAILURE);
}

void scan(){
    getline(tokens, NT, ‘ ’);
}

void read(std::string T){
    if (T != NT) error();
    cout << NT << endl;
    scan();
}
```

```
void write(std::string rule){
    cout << rule << endl;
}

void E0(){
    write("E0 -> E1 Y3");
    E1();
    Y3();
}

void Y3(){
    if (NT == "|"){
        write("Y3 -> ‘|’ E1 Y3");
        read("|");
        E1();
        Y3();
    }
    else {
        write("Y3 ->");
    }
}

void E1(){
    write("E1 -> E2 Y2");
    E2();
    Y2();
}

void Y2(){
    if (NT == "(" || NT == "a" || NT == "b" || NT == "c"){
        write("Y2 -> E1");
        E1();
    }
    else {
        write("Y2 ->");
    }
}

void E2(){
    write("E2 -> E3 Y1");
    E3();
    Y1();
}

void Y1(){
    if (NT == "list"){
        write("Y1 -> E3 Y1");
        read("list");
        E3();
        Y1();
    }
    else{
        write("Y1 ->");
    }
}

void E3(){
    write("E3 -> E4 Y0");
    E4();
    Y0();
}
```

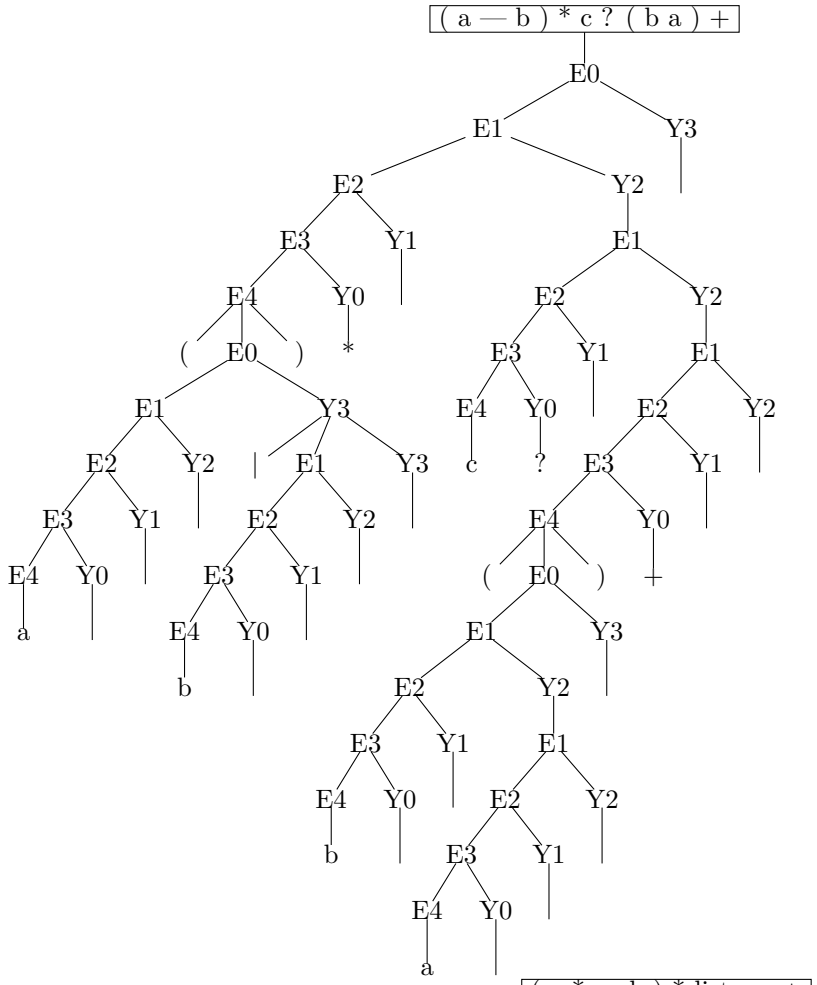
```
void Y0(){
    if (NT == "+" || NT == "*" || NT == "?"){
        write("Y2 -> + | * | ?");
        read(NT);
    }
    else {
        write("Y0 ->");
    }
}

void E4(){
    if (NT == "("){
        write("E4 -> (E0)");
        read("(");
        E0();
        read(")");
    }
    else if (NT == "a"){
        write("E4 -> ‘a’");
        read("a");
    }
    else if (NT == "b"){
        write("E4 -> ‘b’");
        read("b");
    }
    else if (NT == "c"){
        write("E4 -> ‘c’");
        read("c");
    }
    else {error();}
}

int main ()
{
    cout << "( a | b ) * c ? ( b a ) +" << endl;
    cout << "( a * | b ) * list c a +" << endl;
    scan();
    E0();
    cout << "Operation Successful" << endl;
    return 0;
}
```

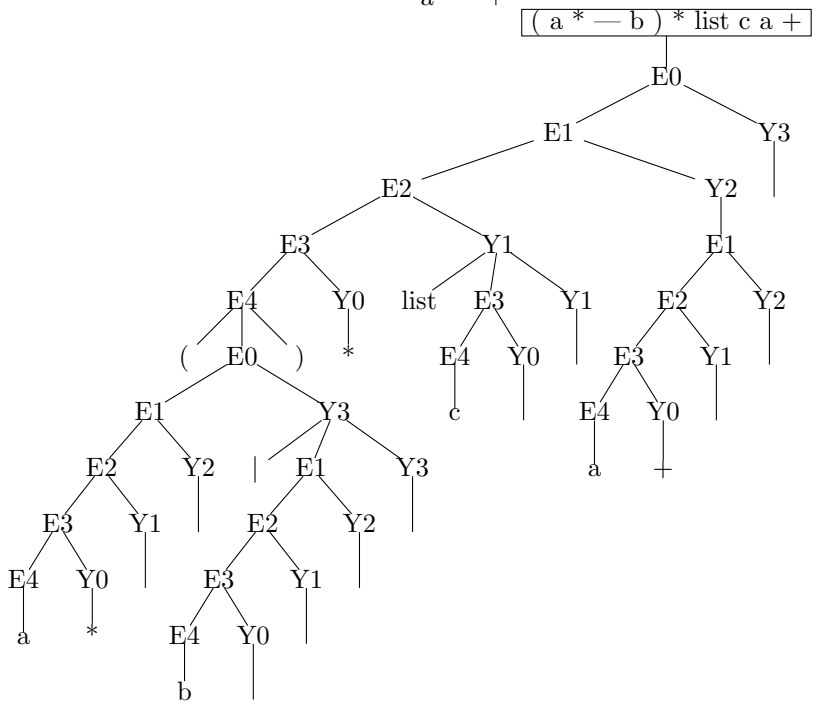
Top Down: This is the result from running the previously linked code for the given inputs using the modified grammar in a Top-Down fashion.

```
( a | b ) * c ? ( b a ) +
E0 -> E1 Y3
E1 -> E2 Y2
E2 -> E3 Y1
E3 -> E4 Y0
E4 -> (E0)
(
E0 -> E1 Y3
E1 -> E2 Y2
E2 -> E3 Y1
E3 -> E4 Y0
E4 -> ‘a’
a
Y0 ->
Y1 ->
```

```
Y2 ->
Y3 -> ‘|’ E1 Y3
|
E1 -> E2 Y2
E2 -> E3 Y1
E3 -> E4 Y0
E4 -> ‘b’
b
Y0 ->
Y1 ->
Y2 ->
Y3 ->
)
Y0 -> + | * | ?
*
Y1 ->
Y2 -> E1
E1 -> E2 Y2
E2 -> E3 Y1
E3 -> E4 Y0
E4 -> ‘c’
c
Y0 -> + | * | ?
?
Y1 ->
Y2 -> E1
E1 -> E2 Y2
E2 -> E3 Y1
E3 -> E4 Y0
E4 -> (E0)
(
E0 -> E1 Y3
E1 -> E2 Y2
E2 -> E3 Y1
E3 -> E4 Y0
E4 -> ‘b’
b
Y0 ->
Y1 ->
Y2 -> E1
E1 -> E2 Y2
E2 -> E3 Y1
E3 -> E4 Y0
E4 -> ‘a’
a
Y0 ->
Y1 ->
Y2 ->
Y3 ->
)
Y0 -> + | * | ?
+
Y1 ->
Y2 ->
Y3 ->
Operation Successful

( a * | b ) * list c a +
E0 -> E1 Y3
E1 -> E2 Y2
E2 -> E3 Y1
E3 -> E4 Y0
```

```
E4 -> (E0)
(
E0 -> E1 Y3
E1 -> E2 Y2
E2 -> E3 Y1
E3 -> E4 Y0
E4 -> ‘a’
a
Y0 -> + | * | ?
*
Y1 ->
Y2 ->
Y3 -> ‘|’ E1 Y3
|
E1 -> E2 Y2
E2 -> E3 Y1
E3 -> E4 Y0
E4 -> ‘b’
b
Y0 ->
Y1 ->
Y2 ->
Y3 ->
)
Y0 -> + | * | ?
*
Y1 -> ‘list’ E3 Y1
list
E3 -> E4 Y0
E4 -> ‘c’
c
Y0 ->
Y1 ->
Y2 -> E1
E1 -> E2 Y2
E2 -> E3 Y1
E3 -> E4 Y0
E4 -> ‘a’
a
Y0 -> + | * | ?
+
Y1 ->
Y2 ->
Y3 ->
Operation Successful
```


(a — b) * c ? (b a) +
E0
E1
Y3
E2
Y2
E3
Y1
E1
E4
Y0
E2
Y2
(
E0
)
*
E3
Y1
E1
E1
Y3
E4
Y0
E2
Y2
E2
Y2
|
E1
Y3
c
?
E3
Y1
E3
Y1
E2
Y2
E4
Y0
E4
Y0
E3
Y1
(
E0
)
+
a
E4
Y0
E1
Y3
b
E2
Y2
E3
Y1
E1
E4
Y0
E2
Y2
b
E3
Y1
E4
Y0
a


(a * — b) * list c a +
E0
E1
Y3
E2
Y2
E3
Y1
E1
E4
Y0
list
E3
Y1
E2
Y2
(
E0
)
*
E4
Y0
E3
Y1
E1
Y3
c
E4
Y0
E2
Y2
|
E1
Y3
a
+
E3
Y1
E2
Y2
E4
Y0
E3
Y1
a
*
E4
Y0
b

Bottom Up: The following output comes from moving the write statements to the end of the executions. The read function prints the token that was just read. This shows that the tokens are read before their corresponding rule is written.

```
( a | b ) * c ? ( b a ) +
(
a
E4 -> 'a'
Y0 ->
E3 -> E4 Y0
Y1 ->
E2 -> E3 Y1
Y2 ->
E1 -> E2 Y2
|
b
E4 -> 'b'
Y0 ->
E3 -> E4 Y0
Y1 ->
E2 -> E3 Y1
Y2 ->
E1 -> E2 Y2
Y3 ->
Y3 -> '|' E1 Y3
E0 -> E1 Y3
)
E4 -> (E0)
*
Y0 -> + | * | ?
E3 -> E4 Y0
Y1 ->
E2 -> E3 Y1
c
E4 -> 'c'
?
Y0 -> + | * | ?
E3 -> E4 Y0
Y1 ->
E2 -> E3 Y1
(
b
E4 -> 'b'
Y0 ->
E3 -> E4 Y0
Y1 ->
E2 -> E3 Y1
a
E4 -> 'a'
Y0 ->
E3 -> E4 Y0
Y1 ->
E2 -> E3 Y1
Y2 ->
E1 -> E2 Y2
Y2 -> E1
E1 -> E2 Y2
Y3 ->
E0 -> E1 Y3
)
E4 -> (E0)
+
Y0 -> + | * | ?
```

```
E3 -> E4 Y0
Y1 ->
E2 -> E3 Y1
Y2 ->
E1 -> E2 Y2
Y2 -> E1
E1 -> E2 Y2
Y2 -> E1
E1 -> E2 Y2
Y3 ->
E0 -> E1 Y3
Operation Successful

( a * | b ) * list c a +
(
a
E4 -> ‘a’
*
Y0 -> + | * | ?
E3 -> E4 Y0
Y1 ->
E2 -> E3 Y1
Y2 ->
E1 -> E2 Y2
|
b
E4 -> ‘b’
Y0 ->
E3 -> E4 Y0
Y1 ->
E2 -> E3 Y1
Y2 ->
E1 -> E2 Y2
Y3 ->
Y3 -> ‘|’ E1 Y3
E0 -> E1 Y3
)
E4 -> (E0)
*
Y0 -> + | * | ?
E3 -> E4 Y0
list
c
E4 -> ‘c’
Y0 ->
E3 -> E4 Y0
Y1 ->
Y1 -> ‘list’ E3 Y1
E2 -> E3 Y1
a
E4 -> ‘a’
+
Y0 -> + | * | ?
E3 -> E4 Y0
Y1 ->
E2 -> E3 Y1
Y2 ->
E1 -> E2 Y2
Y2 -> E1
E1 -> E2 Y2
Y3 ->
```

```
E0 -> E1 Y3
Operation Successful
```

f)

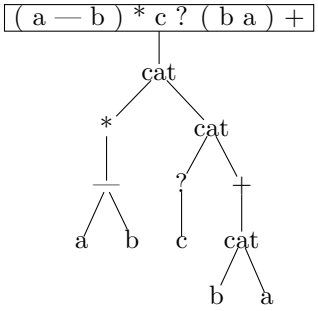
Building the Abstract Syntax Tree

```
#include <stdio.h>
#include <iostream>
#include <string>
#include <string.h>
#include <sstream>
#include <stdlib.h>     /* exit, EXIT_FAILURE */
#include <vector>

 using namespace std;

//std::string myData("( a * | b ) * list c a +");
std::string myData("( a | b ) * c ? ( b a ) +");
std::istringstream tokens(myData);
std::string NT;
std::vector<std::string> tree;

void error();
void scan();
void read(std::string);
void write(std::string);
void treeList();
void buildTree(std::string, int, bool);

void E0();
void E1();
void E2();
void E3();
void E4();

void Y3();
void Y2();
void Y1();
void Y0();

void error(){
    cout << "Unexpected Token" << NT << endl;
    cout << "Exiting... Parse Unsuccessful" << endl;
    exit(EXIT_FAILURE);
}

void scan(){
    getline(tokens, NT, ' ');
}

void read(std::string T){
    if (T != NT) error();
    scan();
}

void write(std::string rule){
    cout << rule << endl;
}

void treeList(){
```

```
    std::string n = "";
    for(int i=0; i<tree.size(); i++){
        n += tree[i];
    }
    cout << n << endl;
}

void buildTree(std::string selectSet, int a, bool isParent){
    cout << "BUILDING NODE " << selectSet << ", " << a << endl;
    std::string nodes = "";
    if (isParent){
        std::string gatherNodes = "";
        for (int i = 0; i < a; i++){
            gatherNodes = tree.back() + gatherNodes;
            tree.pop_back();
        }
        std::string node = "["+selectSet+gatherNodes+"]";
        tree.push_back(node);
    }
    else {
        std::string node = "["+selectSet+"]";
        tree.push_back(node);
    }
}

void E0(){
    E1();
    while (NT == "|"){
        read("|");
        E1();
        buildTree("|", 2, true);
    }
}

void E1(){
    E2();
    if (NT == "(" || NT == "a" || NT == "b" || NT == "c"){
        E1();
        buildTree("cat", 2, true);
    }
    else {
    }
}

void E2(){
    E3();
    while (NT == "list"){
        read("list");
        E3();
        buildTree("list", 2, true);
    }
}

void E3(){
    E4();
    if (NT == "+" || NT == "*" || NT == "?"){
        std::string s = NT;
        read(NT);
        buildTree(s, 1, true);
    }
    else {
```

```
    }
}

void E4(){
    if (NT == "("){
        read("(");
        E0();
        read(")");
    }
    else if (NT == "a"){
        read("a");
        buildTree("a", 1, false);
    }
    else if (NT == "b"){
        read("b");
        buildTree("b", 1, false);
    }
    else if (NT == "c"){
        read("c");
        buildTree("c", 1, false);
    }
    else {error();}
}

int main ()
{
    cout << "( a | b ) * c ? ( b a ) +" << endl;
    cout << "( a * | b ) * list c a +" << endl;
    scan();
    E0();
    cout << tree[0] << endl;
    cout << "Operation Successful" << endl;
    return 0;
}
```

```
( a | b ) * c ? ( b a ) +
BUILDING NODE a, 1
BUILDING NODE b, 1
BUILDING NODE |, 2
BUILDING NODE *, 1
BUILDING NODE c, 1
BUILDING NODE ?, 1
BUILDING NODE b, 1
BUILDING NODE a, 1
BUILDING NODE cat, 2
BUILDING NODE +, 1
BUILDING NODE cat, 2
BUILDING NODE cat, 2
[cat[*[|[a][b]]][cat[?[c]][+[cat[b][a]]]]]
```



```
( a * | b ) * list c a +
BUILDING NODE a, 1
BUILDING NODE *, 1
BUILDING NODE b, 1
BUILDING NODE |, 2
BUILDING NODE *, 1
BUILDING NODE c, 1
BUILDING NODE list, 2
BUILDING NODE a, 1
BUILDING NODE +, 1
BUILDING NODE cat, 2
[cat[list[*[|[*[a]][b]]][c]][+[a]]]
```