

College of Engineering Trivandrum

Compiler Design Lab



Abhishek Manoharan

S7 CSE Roll No:2

TVE17CS002

Department of Computer Science

January 18, 2021



Final Exam Report

1 LL(1) Parser

1.1 Question

For the Grammar G given, build an LL (1) parser and check for the acceptance of any given string. If the given grammar is Left Recursive, write the code for removing it. Input:

- (1) Set of terminals, set of non-terminals, productions and start symbol of the left recursive grammar
- (2) Input string to be parsed

1.2 Theory

LL(1)

LL (1) is a Top-down parser, where the 1st L represents that the scanning of the Input will be done from Left to Right manner and second L shows that in this Parsing technique we are going to use Left most Derivation Tree. and finally the 1 represents the number of look ahead, means how many symbols are you going to see when you want to make a decision. Non- Recursive predictive parsing is a table-driven parser.

1.3 Algorithm

Algorithm 1: Algorithm for Shift Reduce parser

```
1 for each production rule A -> a of a grammar G
2   for each terminal a in FIRST(a)    add A    a to M[A,a]
3   If e in FIRST(a)    for each terminal a in FOLLOW(A) add A    a to M[A,a]
4   If e in FIRST(a) and $ in FOLLOW(A)    add A    a to M[A,$]
5   All other undefined entries of the parsing table are error entries.
6
7 If X and a are $    parser halts (successful completion)
8   If X and a are the same terminal symbol (different from $)    parser pops X from the
9   stack, and moves the next symbol in the input buffer.
10  If X is a non-terminal    parser looks at the parsing table entry M[X,a]. If M[X,a] holds
11  a production rule X ->Y1Y2...Yk, it pops X from the stack and pushes Yk,Yk-1,...,Y1 into
12  the stack. The parser also outputs the production rule X->Y1Y2...Yk to represent a step
13  of the derivation.
14  None of the above    error
15  All empty entries in the parsing table are errors.
16  If X is a terminal symbol different from a, this is also an error case.
```

1.4 Code

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 vector<vector<string>> get_production(unordered_map<char, int> &non_term, int *num)
4 {
5     int non = -1;
6     string s;
7     vector<vector<string>> production(100);
8     getline(cin, s);
9     while (s != "")
10    {
11        int non_index;
12        char left = s[0];
13        if (non_term.find(s[0]) == non_term.end())
14        {
15            non_term[s[0]] = ++non;
16            non_index = non;
17        }
18        else
19        {
```

```

20         non_index = non_term[s[0]];
21     }
22     string right = s.substr(4, s.size() - 4);
23     production[non].push_back(right);
24     //cout << "right side " << right << endl;
25     getline(cin, s);
26 }
27 *num = non;
28 return production;
29 }
30 unordered_set<char> split_string(string s)
31 {
32     int n = s.size();
33     unordered_set<char> result;
34     for (int i = 0; i < n; ++i)
35     {
36         if (s[i] != ' ')
37         {
38             result.insert(s[i]);
39         }
40     }
41     return result;
42 }
43 void print_stack(stack<char> check)
44 {
45     string s = "";
46     while (!check.empty())
47     {
48         s = check.top() + s;
49         check.pop();
50     }
51     cout << s;
52 }
53 vector<vector<string>> remove_left(vector<vector<string>> production, unordered_map<char, int>
    &non_term, int *nn)
54 {
55     char extra = 'Z';
56     int non = *nn;
57     //cout << non << " is non terminal count" << endl;
58     vector<vector<string>> no_left(10);
59     for (int i = 0; i < production.size(); ++i)
60     {
61         int flag = 0;
62         for (int j = 0; j < production[i].size(); ++j)
63         {
64             if (non_term.find(production[i][j][0]) != non_term.end() && non_term[production[i]
                ][j][0] == i)
65             {
66                 flag = 1;
67                 cout << "left recursion found in " << production[i][j] << endl;
68                 vector<string> new_prod;
69                 string alpha = production[i][j].substr(1, production[i][j].size() - 1);
70                 cout << "alpha is " << alpha << endl;
71                 for (int k = 0; k < production[i].size(); ++k)
72                 {
73                     string rhs = "";
74                     if (k != j)
75                     {
76                         rhs = production[i][k] + extra;
77                         new_prod.push_back(rhs);
78                         cout << "first production push back" << endl;
79                     }
80                 }
81                 no_left[i] = new_prod;
82                 vector<string> second_prod;
83                 second_prod.push_back(alpha + extra);
84                 second_prod.push_back("#");
85                 cout << "Extra prodcuton pushing back" << endl;
86                 no_left[++non] = second_prod;
87                 non_term[extra] = non;
88                 extra = extra - 1;
89                 cout << "push back ended" << endl;
90             }
91         }
92         if (flag == 0)
93         {

```

```

94         for (int m = 0; m < production[i].size(); ++m)
95             no_left[i].push_back(production[i][m]);
96     }
97 }
98 *nn = non;
99 // cout << "new number of non terminal is : " << non << endl;
100 // for (auto x : no_left)
101 // {
102 //     for (auto y : x)
103 //     {
104 //         cout << y << "\t";
105 //     }
106 //     cout << endl;
107 // }
108 return no_left;
109 }
110 vector<char> find_first(char c, vector<vector<string>> production, vector<vector<char>> &First
111 , unordered_map<char, int> umap)
112 {
113     vector<char> res;
114     if (umap.find(c) == umap.end())
115     {
116         res.push_back(c);
117         return res;
118     }
119     int num = umap[c];
120     if (First[num].size() != 0)
121     {
122         return First[num];
123     }
124     int n = production[num].size();
125     for (int i = 0; i < n; ++i) // iterate through each production
126     {
127         string m = production[num][i];
128         int right_size = m.size();
129         for (int j = 0; j < right_size; ++j) // iterate through each charecter in production
130         {
131             if (umap.find(m[j]) == umap.end()) //if right side of production is a terminal
132             {
133                 if (find(res.begin(), res.end(), m[j]) == res.end())
134                 {
135                     res.push_back(m[j]);
136                 }
137                 break;
138             }
139             else // Non terminal
140             {
141                 vector<char> temp = find_first(m[j], production, First, umap); // finding
142                 first of j th non terminal
143                 //cout << "called for first of " << m[j] << endl;
144                 int first_char = temp.size();
145                 int flag = 1;
146                 for (int k = 0; k < first_char; ++k)
147                 {
148                     if (temp[k] == '#')
149                     {
150                         // cout << "Epsilon found in first of " << m[j] << endl;
151                         flag = 0;
152                     }
153                     if (find(res.begin(), res.end(), temp[k]) == res.end())
154                     {
155                         if (temp[k] != '#')
156                         {
157                             res.push_back(temp[k]);
158                         }
159                         else
160                         {
161                             if (j == right_size - 1)
162                             {
163                                 res.push_back(temp[k]);
164                             }
165                         }
166                     }
167                 }
168             }
169         }
170     }
171     if (flag == 1)

```

```

168         {
169             break;
170         }
171     }
172 }
173 }
174 First[num] = res;
175 return res;
176 }
177 unordered_set<char> find_follow(char c, vector<vector<string>> production, vector<
    unordered_set<char>> &follow, unordered_map<char, int> umap, vector<vector<char>> first)
178 {
179     if (!follow[umap[c]].empty())
180     {
181         return follow[umap[c]];
182     }
183     //cout << "called follow of " << c << endl;
184     unordered_set<char> res;
185     if (umap[c] == 0)
186     {
187         //cout << "added $ in follow of " << c << endl;
188         res.insert('$');
189     }
190     int n = production.size();
191     for (int i = 0; i < n; ++i)
192     {
193         for (auto x : production[i])
194         {
195             // considering each production
196             int m = x.size(); //read rhs charecter by charecter
197             for (int j = 0; j < m; ++j)
198             {
199                 if (x[j] == c) // if we find charecter in right side of production
200                 {
201                     //cout << c << " found in production " << x << endl;
202                     if (j == m - 1)
203                     { //last element
204                         // cout << c << " is the edning charecter" << endl;
205                         char check;
206                         for (auto y : umap)
207                         {
208                             if (y.second == i)
209                             {
210                                 check = y.first;
211                             }
212                             if (check != c)
213                             {
214                                 unordered_set<char> sample = find_follow(check, production, follow
, umap, first);
215                                 for (auto y : sample)
216                                 {
217                                     //cout << y << " inserted in follow of " << c << endl;
218
219                                     res.insert(y);
220                                 }
221                                 //cout << endl;
222                             }
223                         }
224                     }
225                     else
226                     {
227                         for (int k = j + 1; k < m; ++k)
228                         {
229                             int flag = 1;
230                             if (umap.find(x[k]) == umap.end())
231                             { // checking whether char is termi if so add and stop
232                                 // cout << "since found non terminal " << x[k] << "stop here
233
234                                 //<< "in follow of " << c << endl;
235                                 res.insert(x[k]);
236                                 flag = 1; //should stop here
237                             }
238                             else
239                             { // if it is a non terminal then add its first
240                                 int first_b = first[umap[x[k]]].size();
241                                 for (int l = 0; l < first_b; ++l)
242                                 {

```

```

241         if (first[umap[x[k]]][1] != '#')
242         {
243             res.insert(first[umap[x[k]]][1]);
244             //cout << first[umap[x[k]]][1] << " Added to follow of
" << c << endl;
245         }
246         else //we found an epsilon in first so process should
continues
247         {
248             flag = 0;
249             if (k == m - 1) // first[b] has #
250             {
251                 char check;
252                 for (auto y : umap)
253                 {
254                     if (y.second == i)
255                     {
256                         check = y.first;
257                     }
258                 }
259                 if (check != c)
260                 {
261                     unordered_set<char> sample = find_follow(check
, production, follow, umap, first);
262                     for (auto y : sample)
263                     {
264                         //cout << y << "added to follow of " << c
<< endl;
265                         res.insert(y);
266                     }
267                 }
268             }
269         }
270     }
271     if (flag == 1)
272     {
273         break;
274     }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 return res;
283 }
284 void print_star(int n)
285 {
286     for (int i = 0; i < n; ++i)
287     {
288         cout << "*";
289     }
290     cout << endl;
291 }
292 int main()
293 {
294     int non = -1;
295     string s;
296     vector<vector<string>> production(100);
297     cout << "Enter the productions in the form \"S : r\" " << endl;
298     unordered_map<char, int> non_term;
299     production = get_production(non_term, &non);
300     unordered_set<char> terminals;
301     unordered_set<char> non_terminals;
302     cout << "Non-terminals: ";
303     getline(cin, s);
304     non_terminals = split_string(s);
305     // for (auto x : non_terminals)
306     // {
307     //     cout << x << " ";
308     // }
309     // cout << endl;
310     cout << "Terminals: ";
311     getline(cin, s);
312     terminals = split_string(s);

```

```

313 vector<vector<string>> new_production(100);
314 new_production = remove_left(production, non_term, &non);
315 vector<vector<char>> First(non + 1);
316 cout << "New grammar after removing left recursion" << endl;
317 print_star(20);
318 //cout << "non terminal count is : " << non + 1 << endl;
319 for (auto x : non_term)
320 {
321     if (non_terminals.find(x.first) == non_terminals.end())
322     {
323         non_terminals.insert(x.first);
324     }
325 }
326 for (int i = 0; i <= non; ++i)
327 {
328     char left;
329     for (auto x : non_term)
330     {
331         if (x.second == i)
332         {
333             left = x.first;
334             //cout << x.first << " --> ";
335         }
336     }
337     for (int j = 0; j < new_production[i].size(); ++j)
338     {
339         cout << left << " --> " << new_production[i][j] << "\t";
340     }
341     cout << endl;
342 }
343 for (auto x : non_terminals)
344 {
345     First[non_term[x]] = find_first(x, new_production, First, non_term);
346 }
347 cout << "-----First-----" << endl;
348 for (auto x : non_terminals)
349 {
350     cout << x << ": ";
351     for (auto y : First[non_term[x]])
352     {
353         cout << y << " ";
354     }
355     cout << endl;
356 }
357 vector<unordered_set<char>> follow(non + 1);
358 cout << "Enter the Start symbol: ";
359 char c;
360 cin >> c;
361 cout << "-----Follow-----" << endl;
362 for (auto x : non_terminals)
363 {
364     if (follow[non_term[x]].empty())
365         follow[non_term[x]] = find_follow(x, new_production, follow, non_term, First);
366 }
367 for (auto x : non_terminals)
368 {
369     cout << x << ": ";
370     for (auto y : follow[non_term[x]])
371     {
372         cout << y << " ";
373     }
374     cout << endl;
375 }
376 int terminal_count = terminals.size();
377 //cout << "terminal count is : " << terminal_count << endl;
378 vector<vector<string>> parse_table(100, vector<string>(100));
379 unordered_map<char, int> term_mapp;
380 int ter_count = 0;
381 for (auto x : terminals)
382 {
383     term_mapp[x] = ter_count++;
384     //cout << x << endl;
385 }
386 for (int i = 0; i <= non; ++i)
387 {
388     for (int j = 0; j < new_production[i].size(); ++j)

```

```

389     {
390         for (int k = 0; k < new_production[i][j].size(); ++k)
391         {
392             int flag = 0;
393             if (terminals.find(production[i][j][k]) != terminals.end())
394             {
395                 if (production[i][j][k] != '#')
396                 {
397                     parse_table[i][term_mapp[production[i][j][k]]] = production[i][j];
398                     //cout << production[i][j] << " added to " << i << production[i][j][k]
399
400                     break;
401                 }
402                 else
403                 {
404                     flag = 1;
405                 }
406             }
407             else
408             {
409                 vector<char> its_first = First[non_term[production[i][j][k]]];
410                 for (int m = 0; its_first.size(); ++m)
411                 {
412                     if (its_first[m] != '#')
413                     {
414                         parse_table[i][term_mapp[its_first[m]]] = production[i][j];
415                     }
416                     else
417                     {
418                         flag = 1;
419                     }
420                 }
421             }
422             if (flag == 1)
423             {
424                 unordered_set<char> its_follow = follow[i];
425                 for (auto x : its_follow)
426                 {
427                     if (x != '$')
428                     {
429                         parse_table[i][term_mapp[x]] = production[i][j];
430                     }
431                     else
432                     {
433                         parse_table[i][term_mapp['#']] = production[i][j];
434                     }
435                 }
436             }
437         }
438     }
439 }
440
441 cout << "Parsing table" << endl;
442 cout << "\t";
443 for (auto x : terminals)
444 {
445     if (x != '#')
446     {
447         cout << x << "\t";
448     }
449     else
450     {
451         cout << '$' << "\t";
452     }
453 }
454 cout << endl;
455 for (int i = 0; i <= non; ++i)
456 {
457     char left;
458     for (auto x : non_term)
459     {
460         if (x.second == i)
461         {
462             left = x.first;
463             //cout << x.first << " --> ";
464         }
465     }
466     cout << left << "\t";
467     for (int j = 0; j < ter_count; ++j)
468     {
469         cout << parse_table[i][j] << "\t";
470     }
471 }

```



```

464     }
465     cout << endl;
466 }
467 cout << "Enter the string to be parsed" << endl;
468 string input;
469 cin >> input;
470 stack<char> parsing;
471 input = input + '$';
472 parsing.push('$');
473 parsing.push(c);
474 int keep = 0;
475 cout << "stack\tInput\toutput" << endl;
476 print_stack(parsing);
477 cout << "\t" << input << endl;
478 for (int i = 0; i < 100; ++i)
479 {
480     char molil = parsing.top();
481     //cout << molil << " is in top " << endl;
482     if (terminals.find(parsing.top()) != terminals.end())
483     {
484         //cout << "terminal found" << endl;
485         if (parsing.top() == input[keep])
486         {
487             parsing.pop();
488             keep++;
489             print_stack(parsing);
490             cout << "\t" << input.substr(keep, input.size() - keep) << endl;
491         }
492         else
493         {
494             cout << "invalid input parsing cant be done" << endl;
495             break;
496         }
497     }
498     else
499     {
500         if (input[keep] == '$' && molil == '$')
501         {
502             cout << "parsing finished successfully" << endl;
503             break;
504         }
505         //cout << "Non terminal in top of stack" << endl;
506         string action = parse_table[non_term[parsing.top()]] [term_mapp[input[keep]]];
507         if (action == "")
508         {
509             cout << "invalid input no productuon present" << endl;
510             break;
511         }
512         else
513         {
514             if (action == "#")
515             {
516                 parsing.pop();
517                 print_stack(parsing);
518                 cout << "\t" << input.substr(keep, input.size() - keep) << "\t" << molil
519                 << "-->" << action << endl;
520                 continue;
521             }
522             parsing.pop();
523             for (int j = action.size() - 1; j >= 0; --j)
524             {
525                 parsing.push(action[j]);
526             }
527             print_stack(parsing);
528             cout << "\t" << input.substr(keep, input.size() - keep) << "\t" << molil << "-->"
529             << action << endl;
530         }
531     }
532 }

```

Code for LL parser

1.5 Output

```
abhishek@hephaestus:~/Desktop/S7/CD LAB/Exam$ ./a.out
Enter the productions in the form "S : r"
S : aBa
B : #
B : bB

Non-terminals: S B
Terminals: a b #
New grammer after removing left recursion
*****
S --> aBa
B --> # B --> bB
-----First-----
B: # b
S: a
Enter the Start symbol: S
-----Follow-----
B: a
S: $
Parsing table
      $      b      a
S      aBa
B      bB      #
Enter the string to be parsed
abba
stack  Input  output
$S     abba$
$aBa   abba$   S-->aBa
$aB     bba$
$aBb    bba$   B-->bB
$aB     ba$
$aBb    ba$   B-->bB
$aB     a$
$a      a$     B-->#
$       $
parsing finished successfully
abhishek@hephaestus:~/Desktop/S7/CD LAB/Exam$
```

Enter the productions in the form "S : r"

E : E+T

E : T

T : T*F

T : F

F : (E)

F : i

Non-terminals: E T F

Terminals: + * i () #

left recursion found in E+T

alpha is +T

first production push back

Extra prodcuton pushing back

push back ended

left recursion found in T*F

alpha is *F

first production push back

Extra prodcuton pushing back

push back ended

New grammer after removing left recursion

E --> TZ

T --> FY

F --> (E) F --> i

Z --> +TZ Z --> #

Y --> *FY Y --> #

-----First-----

Z: + #

Y: * #

F: (i

T: (i

E: (i

Enter the Start symbol: E

-----Follow-----

Z: \$)

Y: +) \$

F: +) \$ *

T: \$) +

```

abhishek@hephaestus:~/Desktop/S7/CD LAB/Exam$ ./a.out
Enter the productions in the form "S : r"
S : (L)
S : a
L : SB
B : ,SB
B : #

Non-terminals: S L B
Terminals: ( ) a , #
New grammer after removing left recursion
*****
S --> (L)      S --> a
L --> SB
B --> ,SB      B --> #
-----First-----
B: , #
L: ( a
S: ( a
Enter the Start symbol: S
-----Follow-----
B: )
L: )
S: ) , $

```

```

abhishek@hephaestus:~/Desktop/S7/CD LAB/Exam$ ./a.out
Enter the productions in the form "S : r"
S : aBa
B : #
B : bB

```

```

Non-terminals: S B
Terminals: a b #
New grammer after removing left recursion

```

```

S --> aBa
B --> # B --> bB

```

-----First-----

```

B: # b
S: a

```

Enter the Start symbol: S

-----Follow-----

```

B: a
S: $

```

Parsing table

	\$	b	a
S			aBa
B		bB	#

Enter the string to be parsed

abba

stack	Input	output
\$S	abba\$	

```
$aBa    abba$    S-->aBa
$aB      bba$
$aBb     bba$    B-->bB
$aB      ba$
$aBb     ba$    B-->bB
$aB      a$
$a       a$      B-->#
$        $
parsing finished successfully
```

1.6 Result

Implemented the program to construct a LL(1) parser. It was compiled using g++ version 9.3.0, and executed in Ubuntu 20.04 and the above output was obtained.