Top-Down Parser

Non-Recursive Predictive Parser

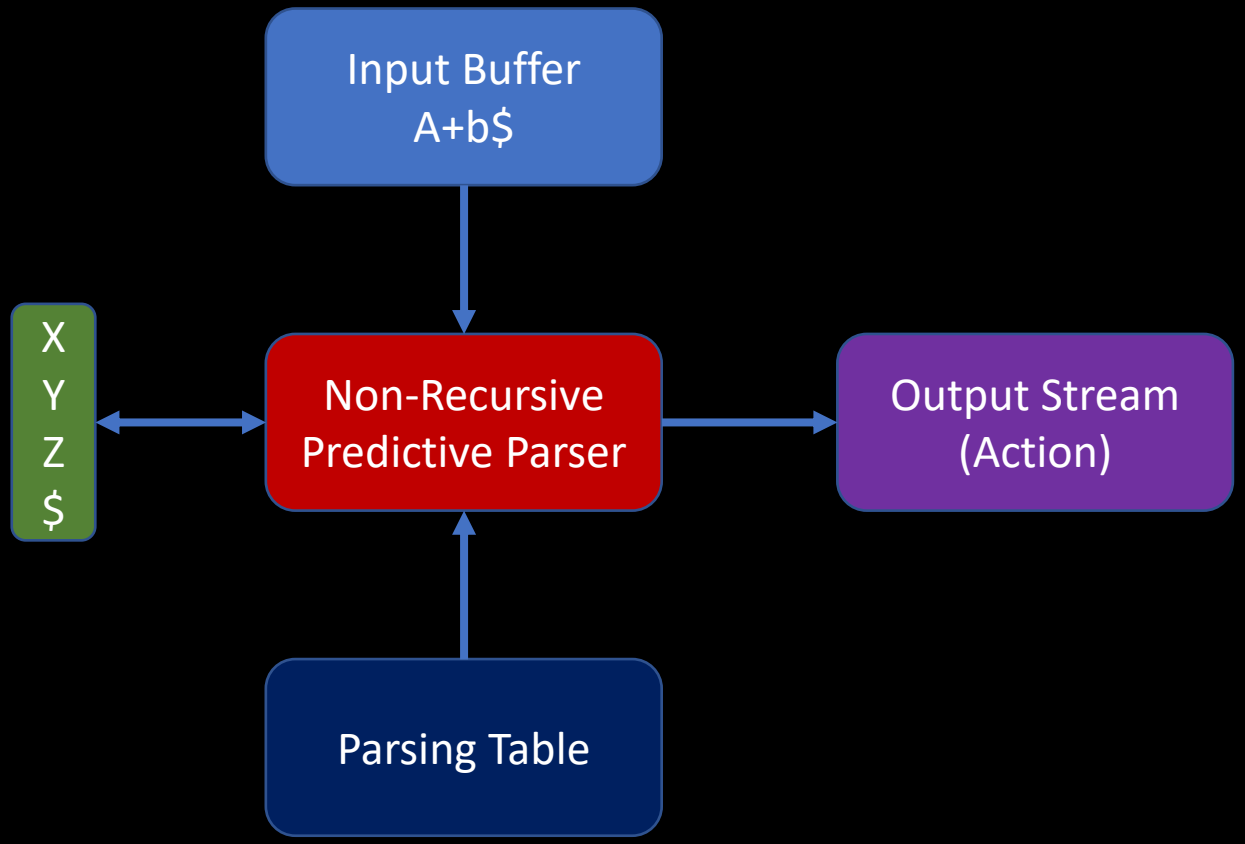
LL(1) Parser

Prepared by : Ahmed A. Alsamman

University of Mosul / College of computer Science and mathematics

Computer Science Department

Block Diagram



Construction of LL(1) Parsing Table

Grammar G:

E → TE'

E'→+TE'|ϵ

T→FT'

T'→*FT'| ϵ

F→(E)|id

First of G
First (E)= First(T)= First(F)= {(, id}
First (E') = {+, ϵ}
First (T) = { (, id}
First (T') = {*, ϵ}
First (F): = {(, id}

Follow of G
Follow (E)= {$,)}
Follow (E') = Follow (E) = {$,)}
Follow (T) = First (E') + Follow (E)= {+,$,)}
Follow (T') = Follow (T) = {+,$,)}
Follow (F): = First(T') + Follow (T)= {*,+,$,)}

Non Terminals	Input Symbols					
	id	+	*	(	)	$
E	E → TE'			E → TE'		
E'		E'→+TE'			E'→ϵ	E'→ϵ
T	T→FT'			T→FT'		
T'		T'→ϵ	T'→*FT'		T'→ϵ	T'→ϵ
F	F→id			F→(E)		

Parsing Algorithm

Let **M** be the parsing table, **X** be the top of the stack and **a** be the current input symbol pointed by **ip**

Repeat

if terminal (X)

If X=a pop X , advance ip

else error

Else Go to the parsing table

if M[X,a] = X→Y1 Y2….. Yk -1,Yk

Pop X, push right hand side in reversed order Yk,Yk-1,….Y2 Y1

Else if M[X,a] =epsilon pop X

Else (M[X,a] =null) error

Until X=$

```
1  //################### Exp.5: TOP-DOWN PARSER (LL1) ###################
2  //################### Coded BY: AHMED ALI ALSAMMAN ###################
3  //################## Input: string of input symbols ##################
4  //############ Output: parsing result (Success, Failure) ############
5  using System;
6  using System.Collections.Generic;
7  using System.Linq;
8  using System.Text;
9  using System.Threading.Tasks;
10 using System.Collections; // for stack
11
12 namespace TopDownParser_LL1
13 {
14     class Program
15     {
16
17         static string[] terminals =  new string[] {"i", "+", "*", "(", ")", "$"};
18
19         static void Main(string[] args)
20         {
21              string[] nonTerminals = new string[] {"E", "E'", "T", "T'", "F"};
22              string[,] parsTab = new string[,]{
23             //-------------------------------------------------------
24             //   id       +       *       (       )       $
25             //-------------------------------------------------------
26               {"TE'",   null,   null,  "TE'",   null,   null},   //E
27               { null, "+TE'",   null,   null,  "eps",  "eps"},   //E'
28               {"FT'",   null,   null,  "FT'",   null,   null},   //T
29               { null,  "eps", "*FT'",   null,  "eps",  "eps"},   //T'
30               {"i"  ,   null,   null,  "(E)",   null,   null},   //F
31           };
32
33          Stack myStack = new Stack();
34
35          while (true) // infinite loop
36           {
37             int next = 0;
38             string top = null;
39             bool err = false;
40             string ParsTabCell = null;
41             myStack.Push("$");
42             myStack.Push("E");
43             string input = Console.ReadLine();
44             if (input != null)
45               {
46                if (validTerminals(input))
47                   {
48                     input += '$';
49
50
51
52
```

```
53
54    do
55     {
56      top = (string)myStack.Peek();
57      //case 1: top of the stack is a terminal
58      if (terminals.Contains(top))
59       if (top == input[next].ToString()) // case 1-1: matched
60        {
61         next++;
62         myStack.Pop();
63        }
64       else // case 1-2: not matched
65        {
66         err = true;
67         break;
68        }
69
70     // case 2: Top of the stack is a non terminal
71     else
72      {
73       int row = Array.IndexOf(nonTerminals, top);
74       int col = Array.IndexOf(terminals, input[next].ToString());
75       ParsTabCell = parsTab[row, col];
76       if (ParsTabCell==null) // case 2-1: cell=null
77         {
78           err = true;
79           break;
80         }
81
82       else if (ParsTabCell == "eps") // case 2-2: cell= epsilon
83         myStack.Pop(); // epsilon will remove the top of the stack
84         else // case 2-3: cell!=epsilon, push it into the stack in reversed order
85           {
86             myStack.Pop(); // remove the top of the stack;
87             for (int i = ParsTabCell.Length - 1; i >= 0; i--)
88              {
89                if (ParsTabCell[i] == '\'') // current char is '
90  {
91                  myStack.Push(ParsTabCell.Substring(i - 1, 2)); // push the last 2 ↵
                       characters
92                  i--; // skip the next char.
93                  }
94                else
95                  myStack.Push(ParsTabCell[i].ToString()); // push the current      ↵
                      char only
96              }
97           }
98       }
99     } while (top != "$");
100
101
102
```

```
103
104         if (err) // terminals did not match or no production found.
105            Console.WriteLine("Parsing Failed");
106            else
107            Console.WriteLine("Parsing Succeeded");
108               }
109                 else // input contains invalid characters
110                 Console.WriteLine("Parsing Failed");
111                 } //closing of if (input != null)
112              } // closing of infinite loop
113         } // closing of Main
114
115    static bool validTerminals(string input)
116         {
117           foreach(char c in input)
118               if(!terminals.Contains(c.ToString()))
119                   return false;
120           return true;
121         }
122     }
123 }
```