**CPSC 323 Handout 7 and assignment No.7**

**The LR Parsing table:** L is for left-to-right scan of tokens; the R is for Rightmost derivation.

This is the second method to help you write a compiler. To be able to compare the Predictive Parsing with this method, we use exactly the same grammar to show you how to construct the LR parsing table.

After finishing the project at the end of this handout, you must decide to write your final project using one of these two methods.

**CONSTRUCTING THE LR PARSING TABLE.** Given the following CFG, first we find the members of FIRST and FOLLOW directly from the given CFG which is in BNF format (in this method you don’t have to remove left recursions)

|  |  |  |
| --- | --- | --- |
| **The given CFG in BNF format** | **Members of FIRST set** | **Members of FOLLOW set** |
| Assign number to each production. The numbers will be used in tracing the table   1. E🡪E + T 2. E🡪E – T 3. E🡪 T 4. T🡪T\*F 5. T🡪T/F 6. T🡪F 7. F🡪( E ) 8. F🡪a | FIRST(F) = {a ( }  T🡪 F implies  FIRST( T )=FIRST( F)={ a ( }  E🡪T implies  FIRST (E ) = FIRST(T)= {a ( } | a.FOLLOW( E )= {+ - ) $ }  b. E🡪T implies  FOLLOW(E) are in FOLLOW( T)  In addition \*, / are in FOLLOW ( T)  FOLLOW(T)={ + - ) $ \* / }  c.FOLLOW(F)  T🡪F implies  FOLLOW(T) are in FOLLOW (F)  FOLLOW(F)= { + - ) $ \* / } |

**Notations.**

**a.** For CFG: A🡪•BC, the dot indicates what symbol( either terminal or non-terminal)must be pushed in stack next. Since the first symbol after dot is B, we push B in stack, the grammar becomes A🡪B•C. Now push C in stack, the grammar change to A🡪BC•. Now, there is no more symbol after the dot which indicates there is nothing more to push in stack.

**b.** Given A🡪•BC, which is called A production, because of A on the left-hand-side. Hence, E🡪•E+T, E🡪•E-T, E🡪•T are called **E productions**  ; T🡪•T\*F, T🡪•T/F, T🡪•F are called **T productions**, and F🡪•(E ), F🡪• a are called **F productions.**

Let’s construct the following FA. Introduce a new non-terminal state E’, and production E’🡪•E to construct state I0

|  |
| --- |
| I0  E’🡪•E , because after • is E, we add all E productions ( production with E on the left side)  E🡪•E + T  E🡪•E – T  E🡪•T , after • is T, add all T productions  T🡪• T\*F  T🡪•T/F  T🡪• F , after • is F, add all F productions  F🡪•( E )  F🡪•a |

|  |
| --- |
| Constructing State I1 of FA  In the following productions of I0, there is an E after the • on the right-hand-side, push E in stack, and move • forward to find the members of I1  E’🡪• E push E E’🡪E•  E🡪•E+T push E E🡪E • + T all these new productions will be stored in state I1  E🡪•E-T push E E🡪E • - T |

|  |
| --- |
| Constructing State I2 of FA  There is a T after • in I0, in the following group of productions, Push T in stack and move • forward to find the members of state I2  E🡪•T push T E🡪T•  T🡪• T\*F push T T🡪T•\*F all of these productions stored in state I2 of the FA  T🡪 •T/F push T T🡪T•/F |

|  |
| --- |
| Constructing State I3 of FA  T🡪•F , push F in stack and move dot forward to get T🡪F•, this is the only production in I3 |

|  |
| --- |
| Constructing State I4 of FA  F🡪•( E ) push ( in stack to get F🡪( •E ), since there is an E after •,  add all E productions to this state  E🡪•E + T  E🡪•E – T  E🡪•T , because after• is T, we add all T productions to this state  T🡪• T\*F  T🡪•T/F  T🡪• F , there is F after •, add all F productions to this state  F🡪•( E )  F🡪•a |
| Constructing State I5 of FA  F🡪•a, push a to get F🡪a• , is the only production in I5 |

Keep these steps in mind to construct a complete the FA which look like this:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| I0  E’🡪•E  E🡪•E + T  E🡪•E – T  E🡪•T  T🡪• T\*F  T🡪•T/F  T🡪• F  F🡪•( E )  F🡪•a | push E  push T  push F  push (  push a | I1  E’🡪E•  E🡪E• +T  E🡪E• -T | Push +  Push -  \*  /  \*    /  E  T  F  (    a | I6  E🡪E +•T  T🡪•T\*F  T🡪•T/F  T🡪• F  F🡪•( E )  F🡪•a | T  F  (    a | I11  E🡪E+T•  T🡪T• \*F  T🡪T•/F | \*  / | I8 |
| I9 |
| E🡪T• I2  T🡪T•\*F  T🡪T•/ F | I3 |  | |
| I4 |
| T🡪F• I3 | I7  E🡪E -•T  T🡪•T\*F  T🡪•T/F  T🡪• F  F🡪•( E )  F🡪•a | I5 |
| I4  F🡪( •E )  E🡪•E + T  E🡪•E – T  E🡪•T  T🡪• T\*F  T🡪•T/F  T🡪• F  F🡪•( E )  F🡪•a | T    F  (  a | I12  E🡪E -T•  T🡪T• \*F  T🡪T•/F | \*  / | I8 |
| I9 |
| I8  T🡪T\*•F  F🡪• ( E)  F🡪• a | I3 | F  (  a |  |
| I4 |  |
| I5 |  |
| F🡪a• I5 | I9  T🡪T/•F  F🡪•( E )  F🡪 •a | F  (  a | I14  T🡪T/F• | I13  T🡪T\*F• |
| I4 | I4 |
| I5 | I5 |
|  | | | I10  F🡪(E •)  E🡪E •+T  E🡪E •- T | )  +  - | I15  F🡪( E )• |  |
| I6 |  |
| I2 | I7 |  |
| I3 |  | | | |
| I4 |
| I5 |

The LR parser table tend to be large. For simplicity we don’t place productions in each state of the FA; instead we use the state number, and how to go from one state to another by pushing terminals and non-terminals. For example,

I0 E I1 + I6

Means, at state I0, after pushing E in stack we enter state I1. From I1 we enter I6 by pushing + in stack.

I0 E I1 + I6 T I11 \* I8

-

F I3 / I9

T I2

(

I7 I4

a I5 \* I8

F I3

\* T I12 / I9

I8

F I3 F I15

( I4 / ( I4 ( I4

I9

a I5 a I5

a I5 F I13

This state means, we already processed state I4. Hence, we use it instead of connecting the arrow to state I4



E ( I4

I10

a I5

T I2 ) I14

F I3 + I6

( I4 - I7

a

I5

**Rules to construct LR parsing table using the above FA**

The table consist of three groups of columns: **States, Terminals** columns and **Non-Terminal** columns

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **States** | **Terminals** | | | | | | | | | **Non-Terminals** | | |
|  | a | | + | - | \* | / | ( | ) | $ | E | T | F |
| 0  1  2  15 |  | **Rules to complete this part of table**   1. **The Sn entries:** Push a **terminal t** to go form state Ik to state In  |  |  |  | | --- | --- | --- | | State Ik  Ik | Terminal  t  [ k , t ] =Sn | State In    In | | Example for S entries:  At I0, push terminal ( to enter I4,  [0,( ]= S4  At I0, push terminal a to enter I5: [0, a]= S5 | | | | 1. **The Accept entry:**   if E’🡪E• is in Ik, then [k, $ ] = Acc  Example:  E’🡪E• is in I1, [1, $ ]= Acc | | | | 1. **The Rn entries:**   if A🡪P• is in Ik, (• at the end of rule)  and A🡪P is rule # n, then for every a in FOLLOW( A ) , [k , a ]= Rn  Example for R entries:  1.E🡪T• is in I2, and E🡪 T is rule #3. Since FOLLOW(E )= {+, -, ) $ } , therefore  [2,+] = [2, -] = [2,)] = [2,$] = R3  2. F🡪( E )• in I14, and F🡪( E) is rule #7, since FOLLOW(F)= { +,- ), $, \*, / }, therefore  [14,+]=[14,-]=[14,)]=[14,$]=[14,\*]=[15,/]=R7 | | | | | | | | | | | **b.To find n ( constant ) entries for this part of the table.** Push a non-terminal X to go from state Ik to state In     |  |  |  | | --- | --- | --- | | State    Ik | Non-Terminal  X  [k , X]=n | State  In | | Examples for constant entries:  At I0, push T to enter I2, [0,T]=2  At I0, push F to enter I3, [0.F]=3 | | |   **Original rules with numbering**   1. E🡪E + T ,rule #1 2. E🡪E – T ,rule #2 3. E🡪 T ,rule #3 4. T🡪T\*F ,rule #4 5. T🡪T/F ,rule #5 6. T🡪F ,rule #6 7. F🡪( E ) ,rule #7 8. F🡪a ,rule #8 | | |

Step-by-step processes to find the **Sn**, **Rn**, **n**, and **ACC** entries in the first 5 rows of the table

CFG FIRST FOLLOW

|  |  |  |
| --- | --- | --- |
| 1. E🡪E + T 2. E 🡪E – T 3. E 🡪 T 4. T🡪T \* F 5. T🡪 T/F 6. T🡪 F 7. F🡪( E ) 8. F🡪 i | FIRST( E ) ={ ( i }  FIRST (T ) = { ( i }  FIRST( F ) = { ( i } | FOLLOW( E ) = { $ + - ) }  FOLLOW( T ) = { $ + - ) \* / }  FOLLOW( F ) = { $ + - ) \* / } |

State 0:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | a | + | - | \* | / | ( | ) | $ | E | T | F |
| 0 | S5 |  |  |  |  | S4 |  |  | 1 | 2 | 3 |

At state I0, push a : [I0,a]=I5, [0,a]=S5, push ( : [I0,( ]=I4, [0,( ]=S4

At state I0, push E: [I0,E]=I1, [0,E] = 1, push T: [I0,T]= I2, [0,T] = 2, push F: [I0,F]=I3, [0,F] = 3

State 1:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | a | + | - | \* | / | ( | ) | $ | E | T | F |
| 1 |  | S6 | S7 |  |  |  |  | **ACC** |  |  |  |

At state I1, push +: [I1, +]=I6, [1,+]=S6, push -: [I1, -] = I7, [1,-] =S7, E’🡪E• is in I1, [1,$]=ACC

State 2:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | a | + | - | \* | / | ( | ) | $ | E | T | F |
| 2 |  | R3 | R3 | S8 | S9 |  | R3 | R3 |  |  |  |

At state I2, push \*: [I2, \*]=I8, [2,\*] =S8, push /: [I2, /]=I9, [2,/ ] =S9,

In I2, E🡪T• and E🡪 T is rule #3, FOLLOW( E)={ S + - ) } Therefore: [2,$]=[2,+]=[2,-]=[2, )]= **R3**

State 3:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | a | + | - | \* | / | ( | ) | $ | E | T | F |
| 3 |  | R6 | R6 | R6 | R6 |  | R6 | R6 |  |  |  |

At state I3, In I3, T🡪F• and T🡪F is rule #6 , FOLLOW(T)={$ + - ) \* / }

Therefore: [3,$] = [3,+] = [3,-]= [3,)] = [3,\*] = [3,/] = **R6**

State 4:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| State | a | + | - | \* | / | ( | ) | $ | E | T | F |
| 4 | S5 |  |  |  |  | S4 |  |  | 10 | 2 | 3 |

At I4, push (: [I4, ( ]=I4, [4,(]=S4, push a: [4, a]=S5, [I4,E]=I10, [4,E]=10, [I4,T]=I2,[4,T]=2, [I4,F]=I3, [4,F] =3

The **R’s** are the same on each row, and the **S’s** are the same on each columns

The LR parser is the most powerful of all other parsers. It can handle the widest variety of CFGs. It works fast, and it detects errors as soon as possible. You can complete the other S, R, and constant entries to construct the complete table.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **States** | **Terminals** | | | | | | | | **Non-terminals** | | |
|  | a | + | - | \* | / | ( | ) | $ | E | T | F |
| 0 | S5 |  |  |  |  | S4 |  |  | 1 | 2 | 3 |
| 1 |  | S6 | S7 |  |  |  |  | **ACC** |  |  |  |
| 2 |  | R3 | R3 | S8 | S9 |  | R3 | R3 |  |  |  |
| 3 |  | R6 | R6 | R6 | R6 |  | R6 | R6 |  |  |  |
| 4 | S5 |  |  |  |  | S4 |  |  | 10 | 2 | 3 |
| 5 |  | R8 | R8 | R8 | R8 |  | R8 | R8 |  |  |  |
| 6 | S5 |  |  |  |  | S4 |  |  |  | 11 | 3 |
| 7 | S5 |  |  |  |  | S4 |  |  |  | 12 | 3 |
| 8 | S5 |  |  |  |  | S4 |  |  |  |  | 13 |
| 9 | S5 |  |  |  |  | S4 |  |  |  |  | 14 |
| 10 |  | S6 | S7 |  |  |  | S15 |  |  |  |  |
| 11 |  | R1 | R1 | S8 | S9 |  | R1 | R1 |  |  |  |
| 12 |  | R2 | R2 | S8 | S9 |  | R2 | R2 |  |  |  |
| 13 |  | R4 | R4 | R4 | R4 |  | R4 | R4 |  |  |  |
| 14 |  | R5 | R5 | R5 | R5 |  | R5 | R5 |  |  |  |
| 15 |  | R7 | R7 | R7 | R7 |  | R7 | R7 |  |  |  |

The following tracing will help you how to write a compiler using this table. As you see the entries in table are blank, Acc, numbers, Sn, Rn. This is how to handle those entries in your tracing:

1. **Blank boxes**: during tracing, if you end up at any blank box that indicates there were something wrong with the expression. ***Reject*** the input
2. **Boxes with number entries**: [ k, X] = n, push k, X, and n in stack
3. **Boxes with Sn entries**: [k, t] = Sn, push k, t, n in stack. t is a non-terminal
4. **Boxes with Rn entries**: [k, t] = Rn,
5. Push k
6. Rule #n: A🡪 B, pop twice as the length of the right hand side
7. Pop for example state m, go to [m, A], A is the left of rule #n

|  |  |  |  |
| --- | --- | --- | --- |
| **State** | **Input string** | Trace **(a+a)/a$** using the LR table |  |
| 1.Push 0 | (a+a)/a$ | 1. E🡪E + T #1 2. E 🡪E – T #2 3. E 🡪 T #3 4. T🡪T \* F #4 5. T🡪 T/F #5 6. T🡪 F #6 7. F🡪( E ) #7 8. F🡪 a #8  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **States** | **Terminals** | | | | | | | | **Non-terminals** | | | |  | a | + | - | \* | / | ( | ) | $ | E | T | F | | 0 | S5 |  |  |  |  | S4 |  |  | 1 | 2 | 3 | | 1 |  | S6 | S7 |  |  |  |  | ACC |  |  |  | | 2 |  | R3 | R3 | S8 | S9 |  | R3 | R3 |  |  |  | | 3 |  | R6 | R6 | R6 | R6 |  | R6 | R6 |  |  |  | | 4 | S5 |  |  |  |  | S4 |  |  | 10 | 2 | 3 | | 5 |  | R8 | R8 | R8 | R8 |  | R8 | R8 |  |  |  | | 13 |  | R4 | R4 | R4 | R4 |  | R4 | R4 |  |  |  | | 14 |  | R5 | R5 | R5 | R5 |  | R5 | R5 |  |  |  | | 15 |  | R7 | R7 | R7 | R7 |  | R7 | R7 |  |  |  | | |
| 2.Pop: 0  **Read: (**  Goto [0,( ]=S4  Push 0, (, 4  Stack:0(4 | a+a)/a$ |
| 3.Pop 4  **Read: a**  Goto [4, a]=S5  Stack:0(4a5 | +a)/a$ |
| 4.Pop: 5  **Read: +**  Goto [5,+]=R8  Push 5  Stack: 0(4a5  #8: **F**  🡪a  Pop 2|a|=2\*1=2  Stack:0(4  Pop:4  [4, **F** ]= 3  Push 4, F, 3  Stack:0(4F3 | a)/a$ |
| 5.Pop: 3  [3,+]=R6  Push 3  Stack: 0(4F3  #6 T🡪F  Pop 2\*|F|=2\*1=2  Stack:0(4  Pop: 4  [4, T]= 2  Push 4, T, 2  Stack:0(4T2 |  |
| 6.Pop: 2  [2, +]=R3  Push 2  Stack:0(4T2  #3 E🡪T  Pop:2\*|T|=2\*1=2  Stack: 0(4  Pop: 4  [4, E]=10  Push 4, E, 10  Stack: 0(4E10 |
| 7.Pop: 10  [10,+]= S6  Push 10, +, 6  Stack: 0(4E10+6 |  | 1. E🡪E + T #1 2. E 🡪E – T #2 3. E 🡪 T #3 4. T🡪T \* F #4 5. T🡪 T/F #5 6. T🡪 F #6 7. F🡪( E ) #7 8. F🡪 a #8 | |
| 8.Pop:6  **Read: a**  [6, a]=S5  Push 6, a, 5  Stack: 0(4E10+6a5 | )/a$ |
| 9.Pop: 5  **Read: )**  [5,) ]=R8  Push 5  #8: F🡪a  Pop 2\*|a|=2\*1=2  Stack: 0(4E10+6  Pop: 6  [6,F]= 3  Push 6,F, 3  Stack:0(4E10+6F3 | /a$ |
|  |
| 10.Pop: 3  [3,)]=R6  Push 3  Stack: 0(4E10+6F3  #6: T🡪F  Pop 2\*|F|=2\*1=2  Stack: 0(4E10+6  Pop: 6  [6, T]= 11  Push 6 ,T , 11  Stack: 0(4E10+6T11 | |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **States** | **Terminals** | | | | | | | | **Non-terminals** | | | |  | a | + | - | \* | / | ( | ) | $ | E | T | F | | 3 |  | R6 | R6 | R6 | R6 |  | R6 | R6 |  |  |  | | 4 | S5 |  |  |  |  | S4 |  |  | 10 | 2 | 3 | | 5 |  | R8 | R8 | R8 | R8 |  | R8 | R8 |  |  |  | | 6 | S5 |  |  |  |  | S4 |  |  |  | 11 | 3 | | 7 | S5 |  |  |  |  | S4 |  |  |  | 12 | 3 | | 8 | S5 |  |  |  |  | S4 |  |  |  |  | 13 | | 9 | S5 |  |  |  |  | S4 |  |  |  |  | 14 | | 10 |  | S6 | S7 |  |  |  | S15 |  |  |  |  | | 11 |  | R1 | R1 | S8 | S9 |  | R1 | R1 |  |  |  | | 12 |  | R2 | R2 | S8 | S9 |  | R2 | R2 |  |  |  | | 15 |  | R7 | R7 | R7 | R7 |  | R7 | R7 |  |  |  | | | |
| 11.Pop: 11  [11, ) ]= R1  Push 11  Stack: 0(4E10+6T11  #1 E🡪E+T  Pop 2\*|E+T|=2\*3=6  Stack: 0(4  Pop: 4  [4,E]=10  Push 4, E, 10  Stack: 0(4E10 |
| 12.Pop:10  [10,)]=S15  Push: 0(4E10)15 |
| 13.Pop:15  **Read: /**  [15, /]=R7  Push 15  Stack: 0(4E10)15  #7: F🡪( E )  Pop 2\*( E)|=2\*3=6  Stack: 0  Pop: 0  [0,F]=3  Push 0,F,3  Stack: 0F3 | |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **States** | **Terminals** | | | | | | | | **Non-terminals** | | | |  | a | + | - | \* | / | ( | ) | $ | E | T | F | | 0 | S5 |  |  |  |  | S4 |  |  | 1 | 2 | 3 | | 1 |  | S6 | S7 |  |  |  |  | ACC |  |  |  | | 2 |  | R3 | R3 | S8 | S9 |  | R3 | R3 |  |  |  | | 3 |  | R6 | R6 | R6 | R6 |  | R6 | R6 |  |  |  | | 4 | S5 |  |  |  |  | S4 |  |  | 10 | 2 | 3 | | 5 |  | R8 | R8 | R8 | R8 |  | R8 | R8 |  |  |  | | 9 | S5 |  |  |  |  | S4 |  |  |  |  | 14 | | 10 |  | S6 | S7 |  |  |  | S15 |  |  |  |  | | 11 |  | R1 | R1 | S8 | S9 |  | R1 | R1 |  |  |  | | 12 |  | R2 | R2 | S8 | S9 |  | R2 | R2 |  |  |  | | 13 |  | R4 | R4 | R4 | R4 |  | R4 | R4 |  |  |  | | 14 |  | R5 | R5 | R5 | R5 |  | R5 | R5 |  |  |  | | 15 |  | R7 | R7 | R7 | R7 |  | R7 | R7 |  |  |  | | | |
| 14.Pop: 3  [3, /]=R6  Push 3  Stack: 0F3  #6: T🡪F  Pop 2\*|F|=2  Stack: 0  Pop: 0  [0, T]= 2  Push 0,T, 2  Stack: 0T2 |
| 15.Pop: 2  [2, / ]= S9  Push 2, /, 9  Stack: 0T2 /9  16. Pop: 9  **Read: a**  [9,a]=S5  Push: 9,a,5  Stack: 0T2/9a5 | a$  $ | 1. E🡪E + T #1 2. E 🡪E – T #2 3. E 🡪 T #3 4. T🡪T \* F #4 5. T🡪 T/F #5 6. T🡪 F #6 7. F🡪( E ) #7 8. F🡪 a #8 | |
| 17.Pop: 5  **Read: $**  [5,$]=R8  Push 5  Stack: 0T2/9a5  #8: F🡪a  Pop 2\*|a|=2  Stack 0T2\*9  pop 9  [9, F]=14  Push 9, F, 14  Stack: 0T2/9F14 |  |
| 1. E🡪E + T #1 2. E 🡪E – T #2 3. E 🡪 T #3 4. T🡪T \* F #4 5. T🡪 T/F #5 6. T🡪 F #6 7. F🡪( E ) #7 8. F🡪 a #8 | |
| 18.Pop: 14  [14,$]=R5  Push 14  Stack 0T2/9F14  #5 : T🡪T/F  Pop 2\*|T/F|=2\*3=6  Stack: 0  Pop: 0  [0, T]= 2  Push: 0T2 |  | |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **States** | **Terminals** | | | | | | | | **Non-terminals** | | | |  | a | + | - | \* | / | ( | ) | $ | E | T | F | | 0 | S5 |  |  |  |  | S4 |  |  | 1 | 2 | 3 | | 1 |  | S6 | S7 |  |  |  |  | ACC |  |  |  | | 2 |  | R3 | R3 | S8 | S9 |  | R3 | R3 |  |  |  | | 5 |  | R8 | R8 | R8 | R8 |  | R8 | R8 |  |  |  | | 12 |  | R2 | R2 | S8 | S9 |  | R2 | R2 |  |  |  | | 13 |  | R4 | R4 | R4 | R4 |  | R4 | R4 |  |  |  | | 14 |  | R5 | R5 | R5 | R5 |  | R5 | R5 |  |  |  | | 15 |  | R7 | R7 | R7 | R7 |  | R7 | R7 |  |  |  | | |
| 19.Pop: 2  [2, $]=R3  Push 2  Stack: 0T2  #3: E🡪T  Pop 2\*|T|=2  Stack: 0  Pop: 0  [0, E]= 1  Push 0, E, 1  Stack: 0E1  Pop: 1  **[1, $]= Acc** |  |

**Factoring:** Consider the following CFG

S🡪 AB | AC, both starts with A on the right-hand-side. Factor the right-hand-side by A to get

AB+AC= A( B + C)= AQ where Q🡪B|C

Now, the original grammar becomes:

S🡪AQ

Q🡪B | C

**Example . Trace (a)a$**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **State** | | | **Input string** | | | | **State** | | | | **Input string** | | | |
| Push 0 | | | (a)a$ | | | |  | | | |  | | | |
| Pop:0  **Read:(**  Goto [0,(]=S4  Push:0 ( 4  Stack:0(4 | | | a)a$ | | | | Stack:0(4F3  Rule 6:T🡪F  Pop 2\*|F|=2  Stack:0(4  Pop:4  Goto [4,T]=2  Push 4,T,2  Stack: 0(4T2 | | | |  | | | |
| Pop:4  **Read: a**  Goto [4,a]=S5  Push:4,a,5  Stack:0(4a5 | | | )a$ | | | | Pop: 2  Goto [2,)]=R3  Push 2  Stack:0(4T2  Rule#3: E🡪T  Pop:2\*|T|=2  Stack: 0(4  Pop:4  Goto [4,E]=10  Push 4,E,10  Stack:0(4E10 | | | |  | | | |
| Pop:5  **Read: )**  Goto [5,)]=R8  Push 5 | | | a$ | | | | Pop:10  Goto [10,)]=S15  Psuh:10 ) 15  Stack:0(4E10)15 | | | |  | | | |
| Rule 8: F🡪a  Pop 2\*|a|=2  Stack:0(4  Pop:4  Goto [4,F]=3 | | |  | | | | Pop:15  **Read:a**  Goto [15,a]= empty | | | | $  **Reject** | | | |
| Push:4,F,3  Stack: 0(4F3  Pop: 3  Goto [3,)]=R6  Push 3 | | |  | | | |  | | | |  | | | |
| **State** | **i** | **+** | | **-** | **\*** | **/** | | **(** | **)** | **$** | | **E** | **T** | **F** | |
| 0 | S5 |  | |  |  |  | | S4 |  |  | | 1 | 2 | 3 | |
| 1 |  | S6 | | S7 |  |  | |  |  | ACC | |  |  |  | |
| 2 |  | R3 | | R3 | S8 | S9 | |  | R3 | R3 | |  |  |  | |
| 3 |  | R6 | | R6 | R6 | R6 | |  | R6 | R6 | |  |  |  | |
| 4 | S5 |  | |  |  |  | | S4 |  |  | | 10 | 2 | 3 | |
| 5 |  | R8 | | R8 | R8 | R8 | |  | R8 | R8 | |  |  |  | |
| 15 |  | R7 | | R7 | R7 | R7 | |  | R7 | R7 | |  |  |  | |

**Computer Science 323**

**Project No. 7 (LR Parser) Names: Richard Gresham**

**Names: Sean McCarthy**

Given the following CFG and the LR Parsing table. Write a program to trace the input strings

(1) (i+i)\*i$ (2) (i\*)$.

CFG FIRST FOLLOW

|  |  |  |
| --- | --- | --- |
| 1. E🡪E + T 2. E 🡪E – T 3. E 🡪 T 4. T🡪T \* F 5. T🡪 T/F 6. T🡪 F 7. F🡪( E ) 8. F🡪 i | FIRST( E ) ={ ( i }  FIRST (T ) = { ( i }  FIRST( F ) = { ( i } | FOLLOW( E ) = { $ + - ) }  FOLLOW( T ) = { $ + - ) \* / }  FOLLOW( F ) = { $ + - ) \* / } |

LR Parsing table for the above CFG as we did it step-by-step in the handout look like this:

Terminals Non-terminals

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **State** | **i** | **+** | **-** | **\*** | **/** | **(** | **)** | **$** | **E** | **T** | **F** |
| 0 | S5 |  |  |  |  | S4 |  |  | 1 | 2 | 3 |
| 1 |  | S6 | S7 |  |  |  |  | **ACC** |  |  |  |
| 2 |  | R3 | R3 | S8 | S9 |  | R3 | R3 |  |  |  |
| 3 |  | R6 | R6 | R6 | R6 |  | R6 | R6 |  |  |  |
| 4 | S5 |  |  |  |  | S4 |  |  | 10 | 2 | 3 |
| 5 |  | R8 | R8 | R8 | R8 |  | R8 | R8 |  |  |  |
| 6 | S5 |  |  |  |  | S4 |  |  |  | 11 | 3 |
| 7 | S5 |  |  |  |  | S4 |  |  |  | 12 | 3 |
| 8 | S5 |  |  |  |  | S4 |  |  |  |  | 13 |
| 9 | S5 |  |  |  |  | S4 |  |  |  |  | 14 |
| 10 |  | S6 | S7 |  |  |  | S15 |  |  |  |  |
| 11 |  | R1 | R1 | S8 | S9 |  | R1 | R1 |  |  |  |
| 12 |  | R2 | R2 | S8 | S9 |  | R2 | R2 |  |  |  |
| 13 |  | R4 | R4 | R4 | R4 |  | R4 | R4 |  |  |  |
| 14 |  | R5 | R5 | R5 | R5 |  | R5 | R5 |  |  |  |
| 15 |  | R7 | R7 | R7 | R7 |  | R7 | R7 |  |  |  |

def main():

    print("This is my LR parsing Table program!")

    uinput = input("Please enter in a trace input: ")

    validation = LRparsing(uinput)

    validation.validationcheck()

    if validation.\_ar == False:

        print("REJECTED")

    else:

        print("ACCEPTED")

class LRparsing:

    def \_\_init\_\_(self,input):

        self.\_userinput = input

        self.\_ar = True    #i    #+      #-        #\*       #/      #(     # )      #$      #E   #T   #F

        self.\_LRparse = [["S5", "blank", "blank", "blank", "blank", "S4", "blank", "blank", "1", "2", "3"], #0

                         ["blank", "S6", "S7", "blank", "blank", "blank", "blank", "ACC", "blank", "blank", "blank"], #1

                         ["blank", "R3", "R3", "S8", "S9", "blank", "R3", "R3", "blank", "blank", "blank"], #2

                         ["blank", "R6", "R6", "R6", "R6", "blank", "R6", "R6", "blank", "blank", "blank"], #3

                         ["S5","blank","blank","blank","blank","S4","blank","blank","10","2", "3"], #4

                         ["blank","R8", "R8", "R8", "R8", "blank", "R8", "R8","blank","blank","blank"], #5

                         ["S5","blank","blank","blank","blank","S4","blank","blank","blank","11","3"], #6

                         ["S5","blank","blank","blank","blank","S4","blank","blank","blank","12", "3"], #7

                         ["S5", "blank","blank","blank","blank","S4", "blank","blank","blank","blank","13"], #8

                         ["S5","blank","blank","blank","blank","S4","blank","blank","blank","blank","14"], #9

                         ["blank", "S6", "S7","blank","blank","blank","S15", "blank","blank","blank","blank"], #10

                         ["blank", "R1", "R1", "S8", "S9", "blank", "R1", "R1", "blank","blank","blank"], #11

                         ["blank", "R2", "R2", "S8", "S9", "blank", "R2", "R2", "blank", "blank", "blank"], #12

                         ["blank", "R4", "R4", "R4", "R4", "blank", "R4", "R4", "blank", "blank", "blank"], #13

                         ["blank", "R5", "R5", "R5", "R5", "blank", "R5", "R5", "blank", "blank", "blank"], #14

                         ["blank", "R7", "R7", "R7", "R7", "blank", "R7", "R7", "blank", "blank", "blank"]] #15

        self.Switcher = { #our dictionary of columns/ tells me what column to go to.

                "i": 0,

                "+": 1,

                "-": 2,

                "\*": 3,

                "/": 4,

                "(": 5,

                ")": 6,

                "$": 7,

                "E": 8,

                "T": 9,

                "F": 10,

            }

        self.CFG = { #dictionary of our Rules from 1-8 for our Rn.

            1 : ("E", "E+T"),

            2 : ("E", "E-T"),

            3 : ("E", "T"),

            4 : ("T", "T\*F"),

            5 : ("T","T/F"),

            6 : ("T", "F"),

            7 : ("F", "(E)"),

            8 : ("F","i"),

        }

    def validationcheck(self):

        acceptfound = False

        tracinginput = self.\_userinput

        stacktable = []

        stacktable.append("0")

        Rcheck = False

        Rnum = "-1"

        for i in tracinginput:

            #print("Read: ", i)

            column = self.Switcher.get(i, -1)

            if column == -1:

                self.\_ar = False

                break

            #print("column: ",column)

            looper = True

            while(looper):

                if len(stacktable) == 0:

                    self.\_ar = False

                    looper = False

                    break

                #print("my stack:", \*stacktable)

                value = stacktable.pop()

                #print("pop: ", value)

                if Rcheck == True:

                    Rvalue = self.Switcher.get(Rnum, -1)

                    #print("GOTO:[", value,",", Rnum,"]" )

                    Checker = self.\_LRparse[int(value)][Rvalue]

                    Rcheck = False

                else:

                    #print("GOTO:[", value,",", i,"]" )

                    Checker = self.\_LRparse[int(value)][column]

                if Checker[0] == "S":

                    newChecker = Checker[1:]

                    looper = False

                    i1 = int(newChecker)

                    stacktable.append(value)

                    stacktable.append(i)

                    stacktable.append(i1)

                    #print("Found: ", i)

                elif Checker[0] == "R":

                    newChecker = Checker[1:]

                    stacktable.append(value)

                    #print("checker", newChecker)

                    cfgmember = self.CFG.get(int(newChecker), "not there?") #gets my tuple from the dictionary

                    popamount = 2 \* len(cfgmember[1])

                    while popamount != 0:

                        stacktable.pop()

                        popamount = popamount - 1

                    Rcheck = True

                    Rnum = cfgmember[0]

                elif Checker == "blank":

                    #print("blank found at Row: ", value, "Column: ", column)

                    self.\_ar = False

                    looper = False

                elif Checker == "ACC":

                    self.\_ar = True

                    looper = False

                    acceptfound = True

                else:

                    #print("this is the value: ", Checker)

                    stacktable.append(value)

                    #print("this is our E,T, or F", Rvalue)

                    if Rvalue == 8:

                        stacktable.append("E")

                    elif Rvalue == 9:

                        stacktable.append("T")

                    elif Rvalue == 10:

                        stacktable.append("F")

                    #print("Checker")

                    stacktable.append(Checker)

            if self.\_ar == False:

                break

            if acceptfound == True:

                break

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Text

Description automatically generated

**Computer Science 323**

**Quiz No.7 Name Richard Gresham**

Given the following CFG and the LR Parsing table.

CFG FIRST FOLLOW

|  |  |  |
| --- | --- | --- |
| 1. E 🡪E – T 2. E 🡪 T 3. T🡪T \* F 4. T🡪 T/F 5. T🡪 F 6. F🡪( E ) 7. F🡪 i | FIRST( E ) ={ ( i }  FIRST (T ) = { ( i }  FIRST( F ) = { ( i } | FOLLOW( E ) = { $ + - ) }  FOLLOW( T ) = { $ + - ) \* / }  FOLLOW( F ) = { $ + - ) \* / } |

1. Find the content of each box after pushing E, - , T

E 🡪E - • T

T 🡪 •T\*F

T 🡪 •T/F

T 🡪 •F

F 🡪 •(E)

F 🡪 •i

E’ 🡪 E•

E 🡪 E• - T

I0

E’🡪•E

E🡪•E-T

E🡪•T

T🡪•T\*F

T🡪•T/F

T🡪•F

F🡪•(E)

F🡪•i

E 🡪 E – T •

T 🡪 T•\*F

T 🡪T•/F

E - T

1. Find the content of each box after pushing T,/,and (

F 🡪 (•E)

E🡪•E – T

E 🡪•T

T 🡪•T/F

T 🡪 •T\*F

T 🡪 •F

F 🡪 •(E)

F 🡪•i

T 🡪 T/•F

F🡪• (E)

F🡪• i

E 🡪 T•

T 🡪T•\*F

T 🡪 T•/F

E’🡪•E

E🡪•E-T

E🡪•T

T🡪•T\*F

T🡪•T/F

T🡪•F

F🡪•(E)

F🡪•i

T / (

1. Use the following FA to complete the parsing table as much as possible using Sn, Rn, and n

I0 I1 I2

F🡪(E•)

E🡪E•-T

T🡪F•

E’🡪•E

E🡪•E-T

E🡪•T

T🡪•T\*F

T🡪•T/F

T🡪•F

F🡪•(E)

F🡪•i

F

I3  E

F🡪(•E)

E🡪•E-T

E🡪•T

T🡪•T\*F

T🡪•T/F

T🡪•F

F🡪•(E)

F🡪•i

|  |  |  |
| --- | --- | --- |
| State | FIRST | FOLLOW |
| E | i, ( | ), - ,$ |
| T | i, ( | ), -, $,\*,/ |
| F | i, ( | ),-,S,\*,/ |

, F

(



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| state | ( | ) | - | $ | \* | / | E | F | T |
| 0 | S3 |  |  |  |  |  |  | 1 |  |
| 1 |  | R13 | R13 | R13 | R13 | R13 |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  | 2 | 1 |  |

At state I0, push F: [I0, F] = I1, [0,F] = 1,

At state i0 push (: [I0, (] = I3, [0, () = S3

At state i1 T 🡪F•, thus it is a RN entry.

Follow(T) = ) - $ \* / = [1,)], [1, - ], [1, $], [1,\*], [1,/] = R13

At state I2 nothing happens.

At state I3, push E = [i3, E] = I2, [3,E] = 2

At state I3, push F = [i3, F] = I1, [3, F] = 1