Homework 6 Predictive Parsing Table

Names: Richard Gresham

Names: Sean McCarthy

1 a). Eliminate left-recursions

E🡪E + T

E🡪T

T🡪T \* F

T🡪F

F🡪(E)

F🡪 a

E 🡪 TE’

E’ 🡪 +TE’

E’ 🡪 λ

T 🡪 FT’

T’ 🡪 \*FT’

T’ 🡪 λ

F 🡪 (E)

F 🡪 a

1b).

|  |  |  |
| --- | --- | --- |
| State | FIRST | FOLLOW |
| E | ( a | ) $ |
| E’ | + λ | ) $ |
| T | ( a | + ) $ |
| T’ | \* λ | + ) $ |
| F | ( a | \* + ) $ |

Finding First:

FIRST(E) = FIRST(T) = ( U a

FIRST (E’) = + U λ

FIRST(T) = FIRST(F) = ( U a

FIRST(T’) = \* U λ

FIRST(F) = ( U a

Finding FOLLOW:

FOLLOW(E) = ) U $

FOLLOW(E’) = FOLLOW(E’) U FOLLOW(E) = FOLLOW(E)

FOLLOW(T) = (FIRST(E’) – λ) + FOLLOW(E’) U (FIRST(E’) – λ) + FOLLOW(E) = + U ) $

FOLLOW(T’) = (FOLLOW(T)) + (FOLLOW(T’)) = +)$

FOLLOW(F) = (FIRST(T’) – λ) + FOLLOW(T) U (FIRST(T’) – λ) + FOLLOW(T’) = \* + ) $

c). Construct a Predictive Parsing Table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | a | + | \* | ( | ) | $ |
| E | TE’ |  |  | TE’ |  |  |
| E’ |  | +TE’ |  |  | λ | λ |
| T | FT’ |  |  | FT’ |  |  |
| T’ |  | λ | \*FT’ |  | λ | λ |
| F | a |  |  | (E) |  |  |

Rule(i) For Each A🡪 B if x if in FIRST of B, then [A,x] = B

E 🡪 TE’, FIRST(TE’) = FIRST(T) = {( a) = [E, ( ] = [E, a] = TE’

E’ 🡪 +TE’, FIRST(+TE’) = FIRST(+) = {+} = [E’, +] = +TE’

T 🡪 FT’, First(FT’) = FIRST(F) = { ( a } = [ T, ( ] = [T, a] = FT’

T’ 🡪 \*FT’ , FIRST(\*FT’) = FIRST(\*) = {\*} = [T’, \*] = \*FT’

F 🡪 ( E ), FIRST((E)) = FIRST(() = {(} = [F, ( ] = (E)

Rule(ii): For each A🡪λ , [A, y]= λ for each y in FOLLOW(A)

E’ 🡪 λ, FOLLOW(E’) = { ) $ } = [E’, )] = [E’, $] = λ

T’ 🡪 λ, FOLLOW(T’) = {+ ) $} = [T’, ) ] = [T’ , +] = [T’ , $] = λ

Rule (iii) For each A🡪 α, where α is terminal, then [A, α] = α

F 🡪 a , [F, a] = a

d) Trace (a + a)$ by showing the content of the stack and the leftover of the input string during tracing.

|  |  |
| --- | --- |
| STACK | INPUT |
| 1. Push $, Push E   Stack: $E | (a + a)$ |
| 1. Pop: E   Read: (  Go to [E, (] = TE’  Push E’, Push T  Stack: $E’T | a + a)$ |
| 1. Pop: T   Go to [T,(] = FT’  Push T’, Push F  Stack: $E’T’F | a + a)$ |
| 1. Pop: F   Go to [F,(] =( E )  Push ), Push E, Push (  Stack: $E’T’)E( | a + a)$ |
| 1. Pop: (   Match with input (  Stack: $E’T’)E | a + a)$ |
| 1. Pop: E   Read: a  Goto [E,a] = TE’  Push E’ Push T  Stack: $E’T’)E’T | + a)$ |
| 1. Pop T   Go to [T, a] = FT’  Push T’ Push F  Stack: $E’T’)E’T’F | + a)$ |
| 1. Pop F   Go to [F, a] = a  Push a  Stack: $E’T’)E’T’a | + a)$ |
| 1. Pop a   Match with input a  Stack: $E’T’)E’T’ | + a)$ |
| 1. Pop: T’   Read: +  Goto [T’, +] = λ  Stack: $E’T’)E’ | a)$ |
| 1. Pop: E’   Goto [E’, +] =+TE’  Push E’ Push T Push +  Stack: $E’T’)E’T+ | a)$ |
| 1. Pop: +   Match with input +  Stack: $E’T’)E’T | a)$ |
| 1. Pop: T   Read: a  Goto [T,a] = FT’  Push T’ Push F  Stack: $E’T’)E’T’F | )$ |
| 1. Pop: F   Goto [F,a] = a  Push a  Stack: $E’T’)E’T’a | )$ |
| 1. Pop: a   Matches with input a  Stack: $E’T’)E’T’ | )$ |
| 1. Pop T’   Read: )  Go to [T’, )] = λ  Stack: $E’T’)E’ | $ |
| 1. Pop: E’   Go to [E’, )] = λ  Stack: $E’T’) | $ |
| 1. Pop: )   Matches with input )  Stack: $E’T’ | $ |
| 1. Pop: T’   Read: $  Go to [T’, $] = λ  Stack: $E’ |  |
| 1. Pop: E’   Go to [E’, $] = λ  Stack: $ |  |
| 1. Pop: $   Matches with input $  Stack :  Thus we are done and it is accepted. | ACCEPTED |

Programming Part:

1. Given the following CFG and the Predictive Parsing table. Write a program to trace input strings **(1) (i+i)\*i$ ,(2) i\*(i-i)$ , (3) i(i+i)$** . Show the content of the stack after each match.

**E 🡪 E+T; E🡪 E- T ;E🡪 T; T🡪T\*F; T🡪 T/F; T🡪F; F🡪 i; F🡪( E )**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| states | i | + | - | \* | / | ( | ) | $ |
| E | TQ |  |  |  |  | TQ |  |  |
| Q |  | +TQ | -TQ |  |  |  | λ | λ |
| T | FR |  |  |  |  | FR |  |  |
| R |  | λ | λ | \*FR | /FR |  | λ | λ |
| F | i |  |  |  |  | (E ) |  |  |

1). Accepted

2). Accepted

3). Rejected

Answer at end on program.

Problem 2:

**S🡪aW W🡪 = E**

**E🡪 E+T**

**E🡪 E- T**

**E🡪 T**

**T🡪 T\*F ,**

**T 🡪T/F**

**T🡪F**

**F🡪a**

**F🡪 ( E )**

**Remove left hand recursions:**

**S 🡪 aW**

**W 🡪 = E**

**E 🡪 TQ**

**Q 🡪 +TQ**

**Q 🡪 -TQ**

**Q 🡪** λ

**T 🡪 FR**

**R 🡪 \*FR**

**R 🡪 /FR**

**R 🡪** λ

F 🡪 a

F 🡪(E)

|  |  |  |
| --- | --- | --- |
| **State** | **FIRST** | **FOLLOW** |
| S | a | $ |
| W | **=** | $ |
| E | a ( | ) $ |
| Q | **+ -** λ | ) $ |
| T | a ( | + - ) $ |
| R | **\* /** λ | + - ) $ |
| F | a ( | \* / + - ) $ |

First of Problems:

FIRST(S) = a

FIRST(W) = =

FIRST(E) = FIRST(T) = a U (

FIRST(Q) = + U – U λ

FIRST(T) = FIRST(F) = a U (

FIRST(R) = \* U / U λ

FIRST(F) = a U (

FOLLOW of Problems:

FOLLOW(S) = $

FOLLOW(W) = FOLLOW(S) = $

FOLLOW(E) = ) U FOLLOW(W) = ) $

FOLLOW(Q) = FOLLOW(E) U FOLLOW(Q) = ) $

FOLLOW T = (FIRST(Q) – λ) U FOLLOW(E) U FOLLOW(Q) = + - ) $

FOLLOW(R)= FOLLOW(T) = + - ) $

FOLLOW(F) = (FIRST(R) - λ) U Follow(T) U FOLLOW(R) = \* / + - ) $

c). construct a predictive Parsing table

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | a | = | + | - | / | \* | ( | ) | $ |
| S | aW |  |  |  |  |  |  |  |  |
| W |  | =E |  |  |  |  |  |  |  |
| E | TQ |  |  |  |  |  | TQ |  |  |
| Q |  |  | +TQ | -TQ |  |  |  | λ | λ |
| T | FR |  |  |  |  |  | FR |  |  |
| R |  |  | λ | λ | /FR | \*FR |  | λ | λ |
| F | a |  |  |  |  |  | (E) |  |  |

**S 🡪 aW ,** First(aW) = {a} = [S, a] = aW

**W 🡪 = E ,** First(=E) = {=} = [W, =] = =E

**E 🡪 TQ ,** First(TQ) = First(T) = { a (} = [E, a] = [E, (] = TQ

**Q 🡪 +TQ,** First(+TQ) = FIRST(+) = {+} = [Q, +] = +TQ

**Q 🡪 -TQ,** First(-TQ) = FIRST(-) = {-} = [Q, -] = -TQ

**T 🡪 FR,** FIRST(FR) = FIRST(F) = {a (} = [T, a] = [T, (] = FR

**R 🡪 \*FR,** FIRST(\*FR) = FIRST(\*) = {\*} = [R, \*] = \*FR

**R 🡪 /FR,** FIRST(/FR) = FIRST(/) = {/} = [R, /] = /FR

F 🡪(E), FIRST((E)) = FIRST(() = {(} = [F, () = (E)

Rule ii:

**Q 🡪** λ **,** FOLLOW(Q) = ) $ = [Q, )] = [Q, $] = λ

**R 🡪** λ, FOLLOW(R) = + - ) $ = [R, +] = [R, - ] = [R, )] = [R, $] = λ

RULE iii:

F 🡪 a, {a} = [F, a] = a

Now Onto Program

def main():

    print("this is my Predictive table program.")

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

    table = input("if problem 1 table press 1, else press 2: ")

    print(table)

    if table == '1':

        validation = Table(uinput)

        validation.validationcheckp1()

        if validation.\_ar == False:

            print("REJECTED")

        else:

            print("ACCEPTED")

    if table == '2':

        print('hi')

        validation = Table(uinput)

        validation.validationcheckp2()

        if validation.\_ar == False:

            print("REJECTED")

        else:

            print("ACCEPTED")

class Table:

    def \_\_init\_\_(self,input):

        self.\_userinput = input

        self.\_ar = True

                        #i     #+       #-       # \*      # /      #(     #)       #$

        self.PTable1 =[["TQ", "blank", "blank", "blank", "blank", "TQ" , "blank", "blank"], #E

                     ["blank", "+TQ","-TQ", "blank", "blank", "blank", "lambda", "lambda"],#Q

                     ["FR", "blank", "blank", "blank", "blank", "FR", "blank", "blank"],   #T

                     ["blank", "lambda", "lambda", "\*FR", "/FR", "blank", "lambda", "lambda"], #R

                     ["i", "blank", "blank","blank","blank", "(E)","blank","blank"]] #F

                        #a     # =     # +       # -      #/        # \*     # (      # )      # $

        self.PTable2 =[["aW", "blank", "blank", "blank", "blank", "blank", "blank", "blank", "blank"], #S

                       ["blank", "=E", "blank", "blank", "blank", "blank", "blank", "blank", "blank"], #W

                       ["TQ", "blank", "blank", "blank", "blank", "blank", "TQ", "blank", "blank"],    #E

                       ["blank", "blank", "+TQ", "-TQ", "blank", "blank", "blank", "lambda", "lambda"],#Q

                       ["FR", "blank", "blank", "blank", "blank", "blank", "FR", "blank", "blank"],    #T

                       ["blank", "blank", "lambda", "lambda", "/FR", "\*FR", "blank", "lambda", "lambda"], #R

                       ["a", "blank", "blank", "blank", "blank", "blank", "(E)", "blank", "blank"]] #F

    def validationcheckp1(self):

        tracinginput = self.\_userinput

        stackinput = []

        stackinput.append("$")

        stackinput.append("E")

        for i in tracinginput:

            #print("read: " , i)

            row = -1

            column = -1

            looper = True

            while(looper):

                if len(stackinput) == 0:

                    self.\_ar = False

                    looper = False

                value = stackinput.pop()

                #checks if value is i if not checks row and column number for next value, if it doesn't exist on the parsing table it returns rejected.

                if i == value:

                   # print("Value:", value)

                   # print("Match with input: ", i)

                    looper = False

                else:

                    #GO To EX:[A, x]

                    if value == "E":

                        row = 0

                    elif value == "Q":

                        row = 1

                    elif value == "T":

                        row = 2

                    elif value == "R":

                        row = 3

                    elif value == "F":

                        row = 4

                    else:

                        print("falls through", value)

                        self.\_ar = False

                        break

                    if i == "i":

                        column = 0

                    elif i == "+":

                        column = 1

                    elif i == "-":

                        column = 2

                    elif i == "\*":

                        column = 3

                    elif i == "/":

                        column = 4

                    elif i == "(":

                        column = 5

                    elif i == ")":

                        column = 6

                    elif i == "$":

                        column = 7

                    parseinput = self.PTable1[row][column]

                    if parseinput == "blank":

                        #print(parseinput)

                        self.\_ar = False

                        break

                    elif parseinput != "lambda":

                        for j in reversed(parseinput):

                            stackinput.append(j)

            if self.\_ar == False:

                break

    def validationcheckp2(self):

        tracinginput = self.\_userinput

        stackinput = []

        stackinput.append("$")

        stackinput.append("S")

        for i in tracinginput:

            row = -1

            column = -1

            looper = True

            while(looper):

                if len(stackinput) == 0:

                    self.\_ar = False

                    looper = False

                value = stackinput.pop()

                if i == value:

                    #print("Value:", value)

                    #print("Match with input: ", i)

                    looper = False

                else:

                    if value == "S":

                        row = 0

                    elif value == "W":

                        row = 1

                    elif value == "E":

                        row = 2

                    elif value == "Q":

                        row = 3

                    elif value == "T":

                        row = 4

                    elif value == "R":

                        row = 5

                    elif value == "F":

                        row = 6

                    else:

                        #print("falls through", value)

                        self.\_ar = False

                        break

                    if i == "a":

                        column = 0

                    elif i == "=":

                        column = 1

                    elif i == "+":

                        column = 2

                    elif i == "-":

                        column = 3

                    elif i == "/":

                        column = 4

                    elif i == "\*":

                        column = 5

                    elif i == "(":

                        column = 6

                    elif i == ")":

                        column = 7

                    elif i == "$":

                        column = 8

                    parseinput = self.PTable2[row][column]

                    if parseinput == "blank":

                        #print(parseinput)

                        self.\_ar = False

                        break

                    elif parseinput != "lambda":

                        for j in reversed(parseinput):

                            stackinput.append(j)

            if self.\_ar == False:

                break

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

    main()

Problem 1 output:

Text

Description automatically generated

Problem 2 output:

Text

Description automatically generated