Andrew López Herrera - 2021062132

Portafolio #2: Gramáticas

Gramática que permita un único main y cero o muchas funciones (main y funciones con una única expresión de asignación o creación)

Inicio -> codigo ::= funciones funcionMain

funciones ::= funcion funciones

funciones ::= funcion

funcionMain ::= ‘int’ ‘main’ ‘(’ parametros ‘)’ cuerpoFuncion

funcion ::= tipo id ‘(’ parametros ‘)’ cuerpoFuncion

parametros ::= parametro ‘,’ parametros

parametros ::= parametro

parametro ::= declaracion

cuerpoFuncion ::= ‘{’ (declaracion | asignación) ‘}’

declaracion ::= tipo id ‘;’

asignacion ::= id ‘=’ literal ‘;’

tipo ::= ‘int’ | ‘boolean’ | ‘char’

literal ::= literalInt | literalBoolean | literalChar

literalInt ::= -?(0|[1-9][0-9]*)

literalBoolean ::= ‘true’ | ‘false’

literalChar ::= '\'' . '\''

id ::= ([A-Z]|[a-z])(_[A-Z][a-z][0-9])+

Andrew López Herrera - 2021062132

Investigación

- Gramáticas Libres de contexto:
  - Definición
  - Notación
  - Derivaciones
  - Gramática Pascal
  - Gramática C
  - Gramática Java

Definición: Las gramáticas libres de contexto son aquellas que pueden generar lenguajes libres o independientes del contexto. Los lenguajes libres de contexto son aquellos que pueden ser representados por autómatas no deterministas o deterministas. También se puede describir como la descripción de la estructura de un lenguaje de programación.

Notación: Para crear gramáticas libres de contexto se utiliza la notación BNF (Backus-Naur Form)

Metasímbolos:

::= se define como

| or

{ } repetición

[ ] opcional

Los terminales entre comillas y negrita, por ejemplo: **'if'**, **'5'**

Derivaciones:

Existen dos formas para restringir la forma en la que se derivan los lenguajes: derivación más a la izquierda y derivación más a la derecha.

Derivación más a la izquierda: siempre reemplaza la variable más a la izquierda por uno de los cuerpos de sus producciones.

Derivación más a la derecha: siempre reemplaza la variable más a la derecha por uno de los cuerpos de sus producciones.

Gramática Pascal: https://www.infor.uva.es/~cvaca/asigs/pascal.html

Gramática C: https://cs.wmich.edu/~gupta/teaching/cs4850/sumII06/The%20syntax%20of%20C%20in%20Backus-Naur%20form.htm

Gramática Java: https://cs.au.dk/~amoeller/RegAut/JavaBNF.html

Andrew López Herrera - 2021062132

Portafolio #2

- Desarrolle los AFN de los lenguajes denotados por las siguientes expresiones regulares y su tabla de transición:

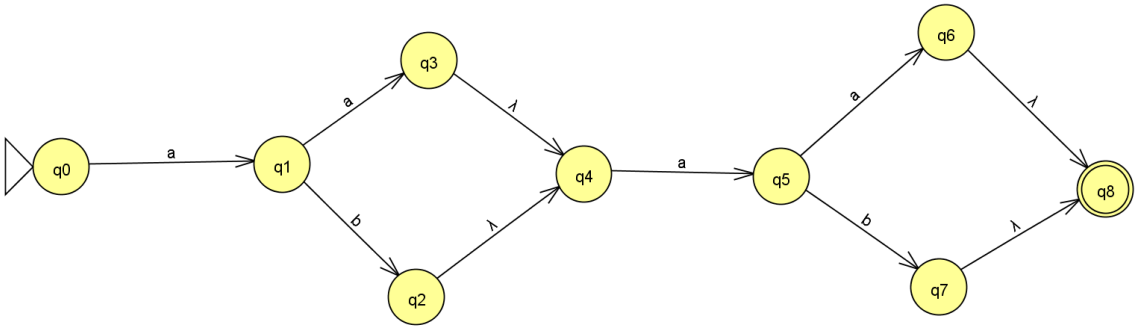
  a(a | b)a(a | b)

  abc | a(b | d)d

  $11^{+}0$

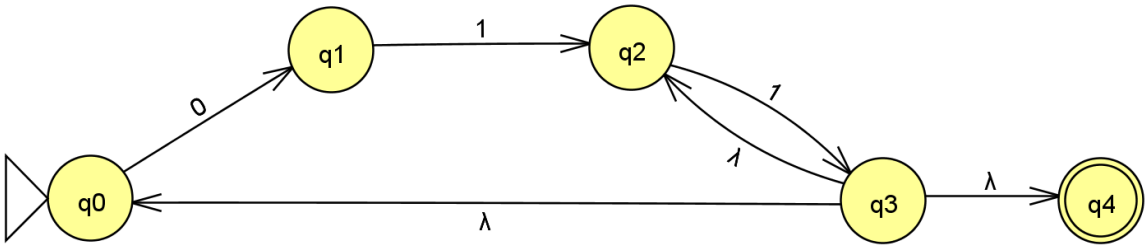
  (011+)+

1. a(a|b)a(a|b)



Andrew López Herrera - 2021062132

2. (011+)+



Andrew López Herrera - 2021062132

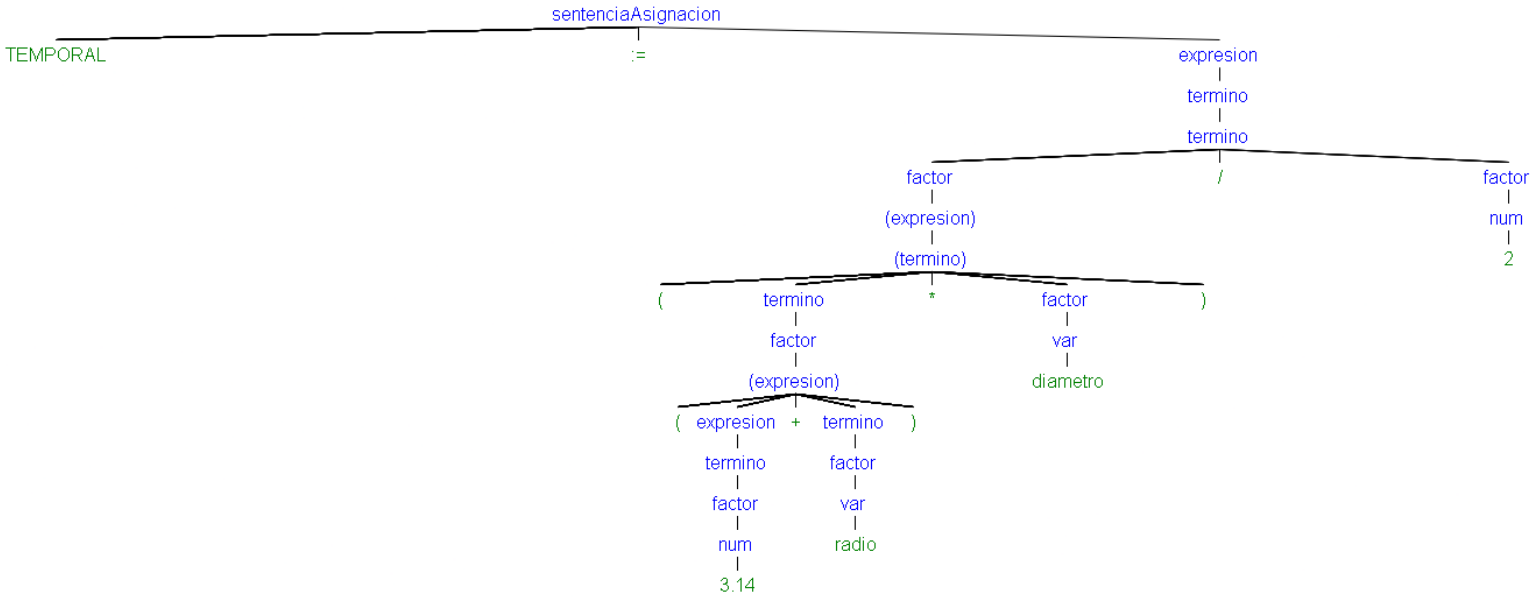
Portafolio #2 Construcción AS

- `varPrincipal:=((3.14+radio)*diametro)/2`
- `TEMPORAL:=(varPrincipal*2)+diametro`
- `Auxiliar:=(TEMPORAL-3.14)*(3.14*2)`

- **TEMPORAL := ((3.14+radio)*diametro)/2**

1. sentenciaAsignacion
2. expresion
3. termino
4. termino/factor
5. factor/num
6. (expresion)/2
7. (termino)/2
8. (termino*factor)/2
9. (factor*var)/2
10. ((expresion)*diametro)/2
11. ((expresion+termino)*diametro)/2
12. ((termino+factor)*diametro)/2
13. ((factor+var)*diametro)/2
14. ((num+radio)*diametro)/2
15. ((3.14+radio)*diametro)/2

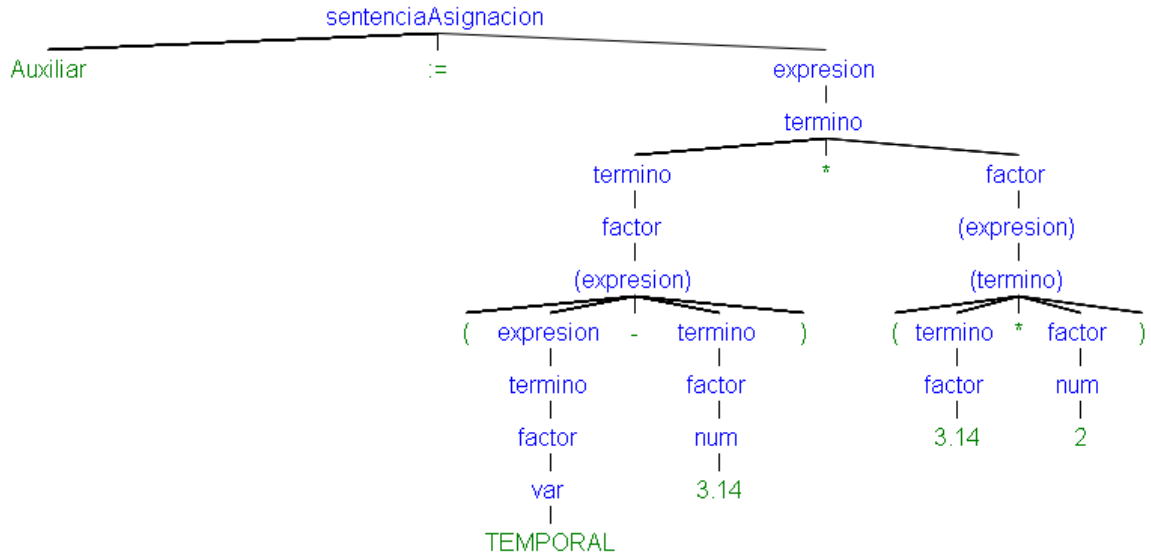
Andrew López Herrera - 2021062132



Andrew López Herrera - 2021062132

- Auxiliar:=(TEMPORAL-3.14)*(3.14*2)

1. sentenciaAsignacion
2. expresion
3. termino
4. termino*factor
5. factor*(expresion)
6. (expresion)*(termino)
7. (expresion-termino)*(termino*factor)
8. (termino-factor)*(factor*num)
9. (factor-num)*(num*2)
10. (var-3.14)*(3.14*2)
11. (TEMPORAL -3.14)*(3.14*2)



Portafolio #2: AS

- Gramática de curso

```
func:int:main _ miFunc(hola,'a'); return; _

func:int:miFunc_
        param:int:dif;
        loc:int:a;
        param:char:otra;
        glob:string:str="Hola $%&/#$&) mundo";
        loc:int:var=34 + id ^ miFunc2 (func3(34,id));
        if ((var+23) > 45 && true || otra == dif)_
                miV = (var+23) > 45 && true || (otra == otrFunc()) ;
        _
        for:var:in:range(5) _ print("Hola mundo"); _
```

1. inicio
2. decS
3. dec
4. func
5. 'func' separador tipo separador id '_' bloque '_'
6. 'func' : 'int' : miFunc '_' bloque '_'
7. 'func' : 'int' : miFunc '_'
   bloque
'_'
8. 'func' : 'int' : miFunc '_'
   bloque
   expr
'_'

9. 'func' : 'int' : miFunc '_'
   bloque
   expr
   estructura
'_'
10. 'func' : 'int' : miFunc '_'
   bloque
   expr
   estructura
   for
'_'
11. 'func' : 'int' : miFunc '_'
   bloque
   expr
   creacionAsign
   if
   'for' separador id separador 'in' separador ‘range’ ‘(’ paramRange ‘)’ ‘_’
      bloque
   ‘_’
'_'
12. 'func' : 'int' : miFunc '_'
   bloque
   expr
   creacionAsign
   'loc' separador tipo separador id '=' exprP finExpr
   'if' '(' exprRelLog ')' '_'

bloque

'_'

'for' : var : 'in' : 'range' ‘(’ intN ‘)’ ‘_’

expr

‘_’

'_'

13. 'func' : 'int' : miFunc '_'

bloque

expr

param

creacionAsignGlob

'loc' : int : var '=' exprArit ;

'if' '(' exprLog_I ')' '_'

expr

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

exprUni

‘_’

'_'

14. 'func' : 'int' : miFunc '_'

expr

creacion

'param' separador tipo separador id finExpr

'glob' separador tipo separador id ‘=’ exprP finExpr

'loc' : int : var '=' exprArit operadorArit operandoArit ;

'if' '(' exprLog_I operadorLog operandoLog ')' '_'

asign

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

exprP finExpr

‘_’

'_'

15. 'func' : 'int' : miFunc '_'

param

'loc' separador tipo separador id finExpr

'param' separador tipo separador id finExpr

'glob' ':' 'string ':' 'str' ‘=’ exprP finExpr

'loc' ':' 'int' ':' var '=' exprArit '^' invocaFunc ';'

'if' '(' exprLog_I operadorLog operandoLog '||' exprRel_I ')' '_'

id ‘=’ exprP finExpr

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

exprArit ';'

‘_’

'_'

16. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ exprArit ';'

'loc' ':' 'int' ':' var '=' exprArit operadorArit operandoArit '^' id ‘(’ parametrosInv ‘)’ ';'

'if' '(' operandoLog '&&' operandoRel '||' operandoRel operadorRel operandoRel ')' '_'

id ‘=’ exprP finExpr

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

operandoArit ';'

‘_’

'_'

16. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ operandoArit ';'

'loc' ':' 'int' ':' var '=' operandoArit '+' id '^' 'miFunc2' ‘(’ paramInv ‘)’ ';'

'if' '(' exprRel_I '&&' exprArit '||' exprArit '==' exprArit ')' '_'

id ‘=’ exprRelLog ';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

invocaFunc ';'

‘_’

'_'

17. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ literal ';'

'loc' ':' 'int' ':' var '=' literal '+' 'id' '^' 'miFunc2' ‘(’ exprP ‘)’ ';'

'if' '(' operandoRel operadorRel operandoRel '&&' operandoArit '||' operandoArit '==' operandoArit ')' '_'

id '=' exprLog_I ';'

'_'

'for' : var : 'in' : 'range' '(' 5 ')' '_'

id '(' parametrosInv ')' ';'

'_'

'_'

18. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' '=' stringN ';'

'loc' ':' 'int' ':' var '=' intN '+' 'id' '^' 'miFunc2' '(' exprArit ')' ';'

'if' '(' exprArit '>' exprArit '&&' literal '||' id '==' id ')' '_'

id '=' exprLog_I operadorLog operandoLog ';'

'_'

'for' : var : 'in' : 'range' '(' 5 ')' '_'

'print'' '(' paramInv ')' ';'

'_'

'_'

19. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' '=' "Hola $%&/#$&) mundo" ';'

'loc' ':' 'int' ':' var '=' 34 '+' 'id' '^' 'miFunc2' ‘(’ operandoArit ‘)’ ';'

'if' '(' exprArit operadorArit operandoArit '>' operandoArit '&&' boolN '||' 'otra '==' 'dif ')' '_'

id ‘=’ exprLog_I operadorLog exprRel_I';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

'print'' ‘(’ exprP ‘)’ ';'

‘_’

'_'

20. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ "Hola $%&/#$&) mundo" ';'

'loc' ':' 'int' ':' var '=' 34 '+' 'id' '^' 'miFunc2' ‘(’ invocaFunc ‘)’ ';'

'if' '(' operandoArit '+' literal '>' literal '&&' true '||' 'otra '==' 'dif ')' '_'

id '=' exprLog_I operadorLog operandoRel ‘||’ operandoRel operadorRel operandoRel ';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

'print'' ‘(’ exprArit ‘)’ ';'

‘_’

'_'

21. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ "Hola $%&/#$&) mundo" ';'

'loc' ':' 'int' ':' var '=' 34 '+' 'id' '^' 'miFunc2' ‘(’ id '(' parametrosInv ')' ')' ';'

'if' '(' id '+' literal '>' literal '&&' 'true' '||' 'otra '==' 'dif ')' '_'

id '=' operandoLog_I '&&' exprArit ‘||’ exprArit '== exprArit ';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

'print'' ‘(’ literal ‘)’ ';'

‘_’

'_'

22. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ "Hola $%&/#$&) mundo" ';'

'loc' ':' 'int' ':' var '=' 34 '+' 'id' '^' 'miFunc2' ‘(’ 'fucn3' '(' parametrosInv ',' paramInv ')' ')' ';'

'if' '(' var '+' literal '>' literal '&&' 'true' '||' 'otra '==' 'dif ')' '_'

id '=' exprRel_I '&&' operandoArit || operandoArit '== operandoArit ';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

'print'' ‘(’ stringN ‘)’ ';'

‘_’

'_'

23. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ "Hola $%&/#$&) mundo" ';'

'loc' ':' 'int' ':' var '=' 34 '+' 'id' '^' 'miFunc2' ‘(’ 'fucn3' '(' paramInv ',' exprP ')' ')' ';'

'if' '(' var '+' intN '>' literal '&&' 'true' '||' 'otra '==' 'dif ')' '_'

id '=' operandoRel operadorRel operandoRel '&&' literal || id '==' invocaFunc ';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

'print'' ‘(’ '"Hola mundo"' ‘)’ ';'

‘_’

'_'

24. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ "Hola $%&/#$&) mundo" ';'

'loc' ':' 'int' ':' var '=' 34 '+' 'id' '^' 'miFunc2' ‘(’ 'fucn3' '(' exprP ',' exprArit ')' ')' ';'

'if' '(' var '+' 23 '>' intN '&&' 'true' '||' 'otra '==' 'dif ')' '_'

miV '=' exprArit ‘>’ exprArit '&&' boolN || otra '==' id ‘(’ ‘)’ ';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

'print'' ‘(’ '"Hola mundo"' ‘)’ ';'

‘_’

'_'

25. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ "Hola $%&/#$&) mundo" ';'

'loc' ':' 'int' ':' var '=' 34 '+' 'id' '^' 'miFunc2' ‘(’ 'fucn3' '(' exprArit ',' operandoArit ')' ')' ';'

'if' '(' var '+' 23 '>' 45 '&&' 'true' '||' 'otra '==' 'dif ')' '_'

miV '=' exprArit operadorArit operandoArit '>' literal '&&' 'true' || otra '==' 'otr' ‘(’ ‘)’ ';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

'print'' ‘(’ '"Hola mundo"' ‘)’ ';'

‘_’

'_'

26. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ "Hola $%&/#$&) mundo" ';'

'loc' ':' 'int' ':' var '=' 34 '+' 'id' '^' 'miFunc2' ‘(’ 'fucn3' '(' operandoArit ',' id ')' ')' ';'

'if' '(' var '+' 23 '>' 45 '&&' 'true' '||' 'otra '==' 'dif ')' '_'

miV '=' operandoArit '+' literal '>' intN '&&' 'true' || otra '==' 'otr' ‘(’ ‘)’ ';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

'print'' ‘(’ '"Hola mundo"' ‘)’ ';'

‘_’

'_'

27. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ "Hola $%&/#$&) mundo" ';'

'loc' ':' 'int' ':' var '=' 34 '+' 'id' '^' 'miFunc2' ‘(’ 'fucn3' '(' literal ',' 'id' ')' ')' ';'

'if' '(' var '+' 23 '>' 45 '&&' 'true' '||' 'otra '==' 'dif ')' '_'

miV '=' id '+' intN '>' 45 '&&' 'true' || otra '==' 'otr' ‘(’ ‘)’ ';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

'print'' ‘(’ '"Hola mundo"' ‘)’ ';'

‘_’

'_'

28. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ "Hola $%&/#$&) mundo" ';'

'loc' ':' 'int' ':' var '=' 34 '+' 'id' '^' 'miFunc2' ‘(’ 'fucn3' '(' intN ',' 'id' ')' ')' ';'

'if' '(' var '+' 23 '>' 45 '&&' 'true' '||' 'otra '==' 'dif ')' '_'

miV '=' var '+' 23 '>' 45 '&&' 'true' || otra '==' 'otr' ‘(’ ‘)’ ';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

'print'' ‘(’ '"Hola mundo"' ‘)’ ';'

‘_’

'_'

29. 'func' : 'int' : miFunc '_'

'param' separador tipo separador id finExpr

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ "Hola $%&/#$&) mundo" ';'

'loc' ':' 'int' ':' var '=' 34 '+' 'id' '^' 'miFunc2' ‘(’ 'fucn3' '(' 34 ',' 'id' ')' ')' ';'

'if' '(' var '+' 23 '>' 45 '&&' 'true' '||' 'otra '==' 'dif ')' '_'

miV '=' var '+' 23 '>' 45 '&&' 'true' || otra '==' 'otr' ‘(’ ‘)’ ';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

'print'' ‘(’ '"Hola mundo"' ‘)’ ';'

‘_’

'_'

30. 'func' : 'int' : miFunc '_'

'param' ';' 'int' ';' 'dif' ';'

'loc' ':' 'int'' ':' 'a' ';'

'param' ':' 'char' ':' 'otra' ';'

'glob' ':' 'string ':' 'str' ‘=’ "Hola $%&/#$&) mundo" ';'

'loc' ':' 'int' ':' var '=' 34 '+' 'id' '^' 'miFunc2' ‘(’ 'fucn3' '(' 34 ',' 'id' ')' ')' ';'

'if' '(' var '+' 23 '>' 45 '&&' 'true' '||' 'otra '==' 'dif ')' '_'

miV '=' var '+' 23 '>' 45 '&&' 'true' || otra '==' 'otr' ‘(’ ‘)’ ';'

'_'

'for' : var : 'in' : 'range' ‘(’ 5 ‘)’ ‘_’

'print'' ‘(’ '"Hola mundo"' ‘)’ ';'

‘_’

'_'

El árbol quedó tan grande que no se puede dibujar

[inicio

[decS

[dec
[func
[func]
[separador
[:]
]
[tipo
[int]
]
[separador
[:]
]
[id
[miFunc]
]
[_]
[bloque
[bloque
[bloque
[bloque
[bloque
[bloque
[bloque
[bloque
[expr
[param
[param]

[separador
[:]
]
[tipo
[int]
]
[separador
[:]
]
[id
[dif]
]
[finExpr
[;]
]
]
]
]
[expr
[creacion
[loc]
[separador
[:]
]
[tipo
[int]

```
      ]
      [separador
       [:]
      ]
      [id
       [a]
      ]
      [finExpr
       [;]
      ]
    ]
  ]
]
[expr
 [param
  [param]
  [separador
   [;]
  ]
  [tipo
   [char]
  ]
  [separador
   [;]
  ]
  [id
```

```
        [otra]
       ]
       [finExpr
        [;]
       ]
      ]
     ]
    ]
    [expr
     [creacionAsign
      [param
       [param]
       [separador
        [:]
       ]
       [tipo
        [str]
       ]
       [separador
        [:]
       ]
       [id]
       [\=]
       [exprP
        [exprArit
         [operandoArit
```

```
            [literal
              [stringN
                ["Hola mundo"]
              ]
            ]
          ]
        ]
      ]
      [finExpr
        [;]
      ]
    ]
  ]
 ]
]
[expr
 [creacionAsign
  [loc]
  [separador
   [:]
  ]
  [tipo
   [int]
  ]
  [separador
   [:]
```

]

[id

[var]

]

[\=]

[exprP

[exprArit

[exprArit

[exprArit

[exprArit

[operandoArit

[literal

[intN

[34]

]

]

]

]

[operadorArit

[\+]

]

[operandoLog

[id

[id]

]

]

]

[operadorArit

[\^]

]

[operandoArit

[invocaFunc

[id

[miFunc2]

]

[(]

[parametrosInv

[paramInv

[exprP

[exprArit

[operandoArit

[invocaFunc

[id

[func3]

]

[(]

[parametrosInv

[parametrosInv

[paramInv

[exprP

[exprArit

[operadorArit

[literal

[intN
[34]
]
]
]
]
]
]
[,]
[paramInv
[exprP
[exprArit
[operadorArit
[id
[id]
]
]
]
]
]
]
[)]
]
]
]

]

]

]

[)]

[;]

]

]

]

]

[finExpr

[;]

]

]

]

]

[expr

[estructura

[if

[if]

[(]

[exprRelLog

[exprLog_I

[exprLog_I

[exprLog_I

[operandoLog

[exprRel_I

```
[operandoRel
 [exprArit
  [exprArit
   [operandoArit
    [id
     [var]
    ]
   ]
  ]
  [operadorArit
   [\+]
  ]
  [operandoArit
   [literal
    [intN
     [23]
    ]
   ]
  ]
 ]
]
[operadorRel
 [\>]
]
[operadorRel
 [exprArit
```

[operandoArit
[literal
[intN
[45]
]
]
]
]
]
]
]
[operadorLog
[&&]
]
[operandoLog
[operandoRel
[exprArit
[operadorArit
[literal
[boolN
[true]
]
]
]
]
]

]

]

]

[operadorLog

[||]

]

[operadorLog

[exprRel_I

[operandoRel

[exprArit

[operandoArit

[id

[otra]

]

]

]

]

[operadorRel

[\=\=]

]

[operandoRel

[exprArit

[operandoArit

[id

[dif]

]

```
          ]
         ]
        ]
       ]
      ]
     ]
    ]
   ]
   [)]
   [\_]
   [bloque
    [expr
     [asign
      [id
       [id]
      ]
      [\=]
      [exprP
       [exprRelLog
        [exprLog\_l
         [exprLog\_l
          [exprLog_l
           [operadorLog_l
            [exprRel_l
             [operandoRel
              [exprArit
               [exprArit
```

[operandoArit
[id
[var]
]
]
]
[operadorArit
[+]
]
[operandoArit
[literal
[intN
[23]
]
]
]
]
]
[operadorRel
[>]
]
[operandoRel
[exprArit
[operadorArit
[literal
[intN

[45]
]
]
]
]
]
]
]
]
[operadorLog
[&&]
]
[operandoRel
[exprArit
[operandoArit
[literal
[boolN
[true]
]
]
]
]
]
]
[operadorLog
[||]

]

[operandoLog

[exprRel_

[operandoRel

[exprArit

[operandoArit

[id

[otra]

]

]

]

]

[operadorRel

[\=\=]

]

[operandoRel

[exprArit

[operandoArit

[invocaFunc

[id]

[(]

[)]

[;]

]

]

]

```
                ]
               ]
              ]
             ]
            ]
           ]
           [finExpr
            [;]
           ]
          ]
         ]
        ]
        [\_]
       ]
      ]
     ]
    ]
    [expr
     [estructura
      [for
       [for]
       [separador
        [:]
       ]
       [id
        [var]
```

]

[separador

[:]

]

[in]

[separador

[in]

]

[range]

[(]

[paramRange

[intN

[5]

]

]

[)]

[_]

[bloque

[expr

[exprUni

[exprP

[exprArit

[operandoArit

[invocaFunc

[id

[print]

```
                ]
                [(]
                [parametrosInv
                 [paramInv
                  [exprP
                   [exprArit
                    [literal
                     [stringN
                      ["Hola mundo"]
                     ]
                    ]
                   ]
                  ]
                 ]
                ]
                [)]
               ]
              ]
             ]
            ]
            [finExpr
             [;]
            ]
           ]
          ]
         ]
```

```
            [\_]
          ]
         ]
        ]
       ]
       [\_]
     ]
   ]
 ]
]
```

Andrew López Herrera - 2021062132

- Investigar técnicas de automatización de autómatas:
  - Minimización de Estados
  - Combinación de Estados Equivalentes
  - Otros

- **Minimización de estados**

Este es un proceso de que permite encontrar un autómata finito M', a partir de otro autómata finito M, con las siguientes propiedades:

1. Si M y M' comienzan por sus estados iniciales, producirán las mismas salidas para las mismas entradas.
2. M' puede tener menos estados que M o la misma cantidad de estados.
3. M' puede tener estados inalcanzables, es decir, no pueden ser alcanzados desde el estado de inicio, estados se remueven.

- **Combinación con estados equivalentes**

Dos estados si y sj de M son equivalentes si para toda $\alpha \in \Sigma^*$, $fO(si , \alpha) = fO(sj , \alpha)$, donde Σ^* denota el conjunto de cadenas de longitud finita sobre el alfabeto de entrada. Es decir, son iguales si un estado en la posición (i, j) de la matriz de estados es igual al estado en la posición (j, i).

- **Otros**

El problema de minimizar autómatas se reduce al problema de encontrar dichas clases de equivalencia. El problema se resuelve de manera iterativa, identificando lo que se conoce como clases de estados k-equivalentes. Dos estados si y sj de M son kequivalentes si para todo $\alpha \in \Sigma^*$, tal que $|\alpha| \leq k$, $fO(si , \alpha) = fO(sj , \alpha)$