Universidad Nacional de Asunción

Facultad Politécnica

Diseño de Compiladores

Trabajo Práctico

Traductor Dirigido por la Sintaxis

**Integrantes:**

* Cesar Belotto
* Javier López

Tabla de contenido

[**Gramática Asignada** 3](#_Toc136784604)

[BNF que acepte la entrada: 3](#_Toc136784605)

[Árbol sintáctico para la cadena de entrada “43 2451 701” 3](#_Toc136784606)

[Árbol sintáctico para la cadena de entrada no válida “23 4#” 4](#_Toc136784607)

[En el árbol sintáctico definimos las reglas semánticas 4](#_Toc136784608)

[Verificamos que la gramática sea predictiva: 5](#_Toc136784609)

[1- Verificamos que no exista recursividad por la izquierda: No se observa que exista recursividad por la izquierda en el BNF. 5](#_Toc136784610)

[2- Verificamos que no exista Factor Común por la izquierda: Si existe, procedemos a eliminarlo: 5](#_Toc136784611)

[Verificamos si tiene factor común indirecto: 5](#_Toc136784612)

[Finalmente, la gramática queda de la siguiente manera: 6](#_Toc136784613)

[Código fuente: 6](#_Toc136784614)

[Entrada y salida en el código fuente: 10](#_Toc136784615)

[Entrada válida: **43 2451 701** 10](#_Toc136784616)

[Entrada no válida: 45 #5 10](#_Toc136784617)

## **Gramática Asignada**

Escribir un TDS predictivo que recibe una lista de números enteros y retorna la misma lista, pero esta vez con los números invertidos.

Ejemplo:

* Entrada = 43 2451 701
* Salida = 34 1542 107

## BNF que acepte la entrada:

lista → lista nro | nro

nro → nro digito | digito

digito → "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

## Árbol sintáctico para la cadena de entrada: “43 2451 701”

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

## Árbol sintáctico para la cadena de entrada no válida: “23 4#”

Gráfico, Gráfico de cajas y bigotes

Descripción generada automáticamente

## En el árbol sintáctico definimos las reglas semánticas

Diagrama

Descripción generada automáticamente con confianza media

|  |  |  |
| --- | --- | --- |
| BNF | Reglas semánticas | |
| lista → lista nro | lista.x = lista.x + " " + nro.x |
| lista → nro | lista.x = nro.x |
| nro → nro digito | nro.x = digito.x + nro.x |
| nro → digito | nro.x = digito.x |
| digito → "0" | digito.x = "0" |
| … | … |
| digito → "9" | digito.x = "9" |

## Verificamos que la gramática sea predictiva:

### Verificamos que no exista recursividad por la izquierda: Si existe procedemos a eliminarlo:

~~lista → lista nro | nro~~

lista → nro R

R → nro R | e

~~nro → nro digito | digito~~

nro → digito R1

R1 → digito R1 | e

### Verificamos que no exista Factor Común por la izquierda: No existe.

**La gramática queda de la siguiente manera:**

lista → nro R

R → nro R | e

nro → digito R1

R1 → digito R1 | e

digito -> 0 | 1 | … | 9

### Verificamos si tiene factor común indirecto:

|  |  |  |
| --- | --- | --- |
| **BNF** | **Conjunto primero** | |
| lista → nro R | P(lista) = P(nro) = {0,1,…,9} |
| R → nro R | e | P(R) = P(nro)u{e} = {0,1,…,9}u{e} |
| nro → digito R1 | P(nro) = P(digito) = {0,1,…,9} |
| R1 → digito R1 | e | P(R1) = P(digito)u{e}) = {0,1,…,9}u{e} |
| digito -> 0 | 1 | … | 9 | P(digito) = {0}u{1}u…u{9} |

No existe factor común indirecto.

## Finalmente, la gramática queda de la siguiente manera:

|  |  |  |
| --- | --- | --- |
| BNF | Reglas semánticas | |
| lista → nro R | lista.x = nro.x + " " + R.x |
| R → nro R | R.x = nro.x + " " + R.x |
| R → e | R.x = "" |
| nro → digito R1 | nro.x = R1.x + digito.x |
| R1 → digito R1 | R1.x = R1.x + digito.x |
| R1 → e | R1.x = "" |
| digito → "0" | digito.x = "0" |
| … | … |
| digito → "9" | digito.x = "9" |

## Código fuente:

|  |  |  |
| --- | --- | --- |
| BNF | Reglas semánticas | |
| lista → nro R | lista.x = nro.x + " " + R.x |

def lista():

    nroX = nro() #34

    Rx = R() #

    return nroX+" "+Rx

|  |  |  |
| --- | --- | --- |
| BNF | Reglas semánticas | |
| R → nro R | e | R.x = nro.x + " " + R.x |
| R → e | R.x = "" |

def R():

    global caracter

    if (caracter == "0" or caracter == "1" or caracter == "2" or caracter == "3" or caracter == "4" or caracter == "5" or caracter == "6" or caracter == "7" or caracter == "8" or caracter == "9"):

        nroX = nro() #

        Rx = R()

        return nroX+" "+Rx

    else:

        match(" ")

        return ""

|  |  |  |
| --- | --- | --- |
| BNF | Reglas semánticas | |
| nro → digito R1 | nro.x = R1.x + digito.x |

def nro():

    digitoX = digito()

    R1x = R1()

    return R1x+digitoX

|  |  |  |
| --- | --- | --- |
| BNF | Reglas semánticas | |
| R1 → digito R1 | R1.x = R1.x + digito.x |
| R1 → e | R1.x = "" |

def R1():

    global caracter

    if (caracter == "0" or caracter == "1" or caracter == "2" or caracter == "3" or caracter == "4" or caracter == "5" or caracter == "6" or caracter == "7" or caracter == "8" or caracter == "9"):

        digitoX = digito()

        R1x = R1()

        x = R1x+digitoX

        return x

    else:

        match(" ")

        return ""

|  |  |  |
| --- | --- | --- |
| BNF | Reglas semánticas | |
| digito → "0" | digito.x = "0" |
| … | … |
| digito → "9" | digito.x = "9" |

def digito():

    global caracter

    if ( caracter == "0" ):

        match("0")

        return "0"

    elif( caracter == "1" ):

        match("1")

        return "1"

    elif( caracter == "2" ):

        match("2")

        return "2"

    elif( caracter == "3" ):

        match("3")

        return "3"

    elif( caracter == "4" ):

        match("4")

        return "4"

    elif( caracter == "5" ):

        match("5")

        return "5"

    elif( caracter == "6" ):

        match("6")

        return "6"

    elif( caracter == "7" ):

        match("7")

        return "7"

    elif( caracter == "8" ):

        match("8")

        return "8"

    else:

        match("9")

        return "9"

match

def match(t):

    global caracter

    global entrada

    if( len(caracter) > 0):

        if ( t == caracter):

            entrada = entrada[1:]

            caracter = entrada[0:1]

            if(t == " "):

                while( caracter == " "):

                    entrada = entrada[1:]

                    caracter = entrada[0:1]

        else:

            print("Error!, {} no es una entrada valida!".format(caracter))

            exit(200)

## Entrada y salida en el código fuente:

### Entrada válida: **43 2451 701**

Entrada:

Texto

Descripción generada automáticamente

Salida:

Captura de pantalla de computadora

Descripción generada automáticamente

### Entrada no válida: 23 4#

Entrada:

Texto

Descripción generada automáticamente

Salida:

