TRABAJO PRÁCTICO DISEÑO DE COMPILADORES

Grupo N° 2

Camila Alderete

Máxima Ayala

Profesor

Sergio Andrés Aranda Zeman

UNIVERSIDAD NACIONAL DE ASUNCIÓN

FACULTAD POLITÉCNICA

San Lorenzo, Paraguay

Junio, 2023

TRADUCCIÓN DIRIGIDA POR LA SINTAXIS

Un Traductor Dirigido por la Sintaxis (TDS), es un procesador de lenguaje donde al analizador sintáctico toma el control de todo el proceso de compilación. Es así como dado un lenguaje de entrada, el traductor implementa los pasos necesarios para obtener a partir de un lenguaje de entrada, uno de salida que por lo general tiene el mismo nivel de complejidad.

ENUNCIADO DEL PROBLEMA

Escribir un TDS predictivo que recibe una lista de palabras formadas con el alfabeto S = {a, b, ..., z} que retorna la cantidad de palabras en las que las vocales se encuentran en igual o mayor cantidad que las consonantes y cuál es la mayor cantidad de vocales en una o más palabras.
Ejemplo: casa perro murcielago retorna: 2, 5

1- BNF

```
S -> cadena
cadena -> letraR
R -> letraR | ' 'letraR | Ɛ
letra -> a|b| ... | z
```

Nota: El espacio esta definido en la gramática. Gestión de errores lanza warnings por espacios redundantes.

2- BNF Y REGLAS SEMÁNTICAS

BNF	REGLAS SEMÁNTICAS

`S -> cadena`	`S.v= 0`
	`S.c = 0`
	`S.p= 0`
	`S.max_v =0`
	
	`cadena.v = S.v`
	`cadena.c = S.c`
	`cadena.p = S.p`
	`cadena.max_v = S.max_v`
	
	`S.p = cadena.p_sin`
	`S.max_v = cadena.max_v_sin`
	
	`print (S.palabras)`
	`print (S.max_vocales)`

```
cadena -> letraR    letra.v = cadena.v
                    letra.c = cadena.c

                    cadena.v_val = letra.v_sin
                    cadena.c_val = letra.c_sin

                    R.v = cadena.v_val
                    R.c = cadena.c_val
                    R.p = cadena.p
                    R.max_v = cadena.max_v

                    cadena.p_sin = R.p_sin
                    cadena.max_v_sin = R.max_v_sin



R -> letraR1        letra.v = R.v
                    letra.c = R.c

                    R.v_val = letra.v_sin
                    R.c_val = letra.c_sin

                    R1.v = R.v_val
                    R1.c = R.c_val
                    R1.p = R.p
                    R1.max_v = R.max_v

                    R.p_sin = R1.p_sin
                    R.max_v_sin = R1.max_v_sin


R -> '  'letraR1    If( R.v >= R.c and (R.v != 0 or R.v != 0) )
                        R.p = R.p + 1

                    If (R.v > R.max_v)
                        R.max_v = R.v


                    R.v =0
                    R.c=0


                    letra.v = R.v
                    letra.c = R.c

                    R.v_val = letra.v_sin
                    R.c_val = letra.c_sin

                    R1.v = R.v_val
```

```
                    R1.c = R.c_val
                    R1.p = R.p
                    R1.max_v = R.max_v

                    R.p_sin= R1.p_sin
                    R.max_v_sin = R1.max_v_sin


R -> ε              If( R.v >= R.c and (R.v!=0 or R.v != 0) )
                        R.p = R.p + 1

                    If (R.v > R.max_v)
                        R.max_v = R.V

                    R.p_sin = R.p
                    R.max_v_sin = R.max_v



letra -> a          letra.v = letra.v + 1
                    letra.v_sin = letra.v

letra -> b          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> c          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> d          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> e          letra.v = letra.v + 1
                    letra.v_sin = letra.v

letra -> f          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> g          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> h          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> i          letra.v = letra.v + 1
                    letra.v_sin = letra.v

letra -> j          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> k          letra.c = letra.c + 1
                    letra.c_sin = letra.c
```

```
letra -> l          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> m          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> n          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> ñ          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> o          letra.v = letra.v + 1
                    letra.v_sin = letra.v

letra -> p          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> q          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> r          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> s          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> t          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> u          letra.v = letra.v + 1
                    letra.v_sin = letra.v

letra -> v          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> w          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> x          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> y          letra.c = letra.c + 1
                    letra.c_sin = letra.c

letra -> z          letra.c = letra.c + 1
                    letra.c_sin = letra.c
```

3- Gramática predictiva

```
S -> cadena
cadena -> letraR
R -> letraR | ' 'letraR | Ɛ
letra -> a|b| ... | z
```

Para comprobar que una gramática es predictiva se debe evaluar que estas siguientes condiciones sean cumplidas:

Factor Común

Podemos observar que la gramática no cuenta con factor común por la izquierda en ninguna de sus producciones.

Recursión por la izquierda directa e indirecta

```
i = 1
S -> cadena   No tiene recursión directa
i = 2
cadena -> letraR   No tiene recursión directa
i = 3
R -> letraR | ' 'letraR | Ɛ    No tiene recursión directa
i = 4
letra -> a | b | ... | z   No tiene recursión directa
```

Como podemos observar, no existe recursión directa o indirecta.

Conjunto Primero y Conjunto Siguiente

```
P(S) = {a,b,…z}
P(cadena) = {a,b,z}
P(R) = {a,…,z} U {' '} u {Ɛ}
P(letra) = {a} U {b} U … U {z}
```

Como tenemos a Ɛ en el conjunto Primero de R es necesario evaluar el conjunto siguiente

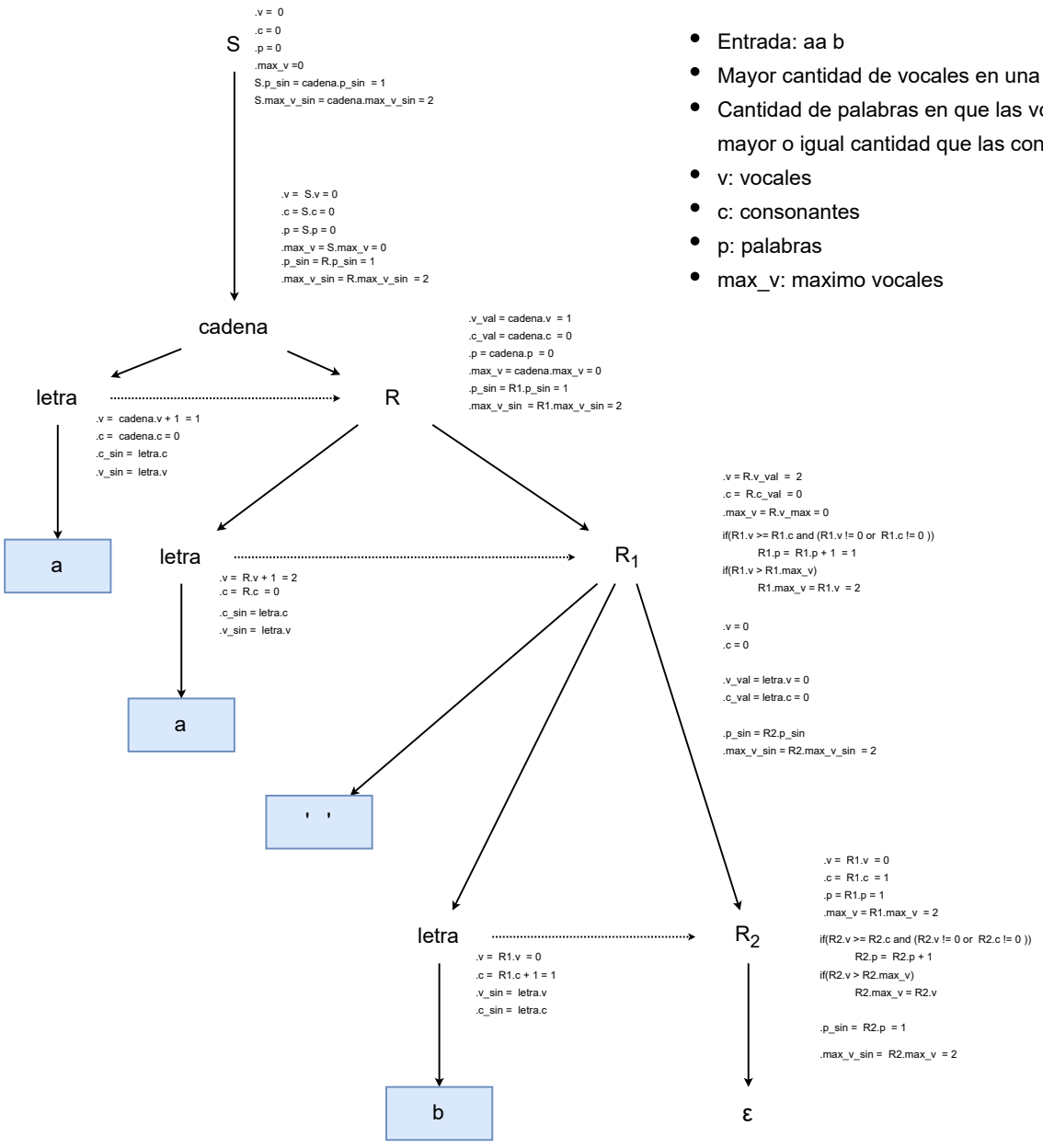
```
S(R)= S(cadena) = {$}
S(cadena) = S(S) = {$}
S(S) = {$}
```

Como el conjunto primero no tiene intersección entre si en cada producción, y como el conjunto primero es disjunto al conjunto siguiente de en la producción R, observamos que la gramática cumple con este criterio.

4- Árbol Sintáctico

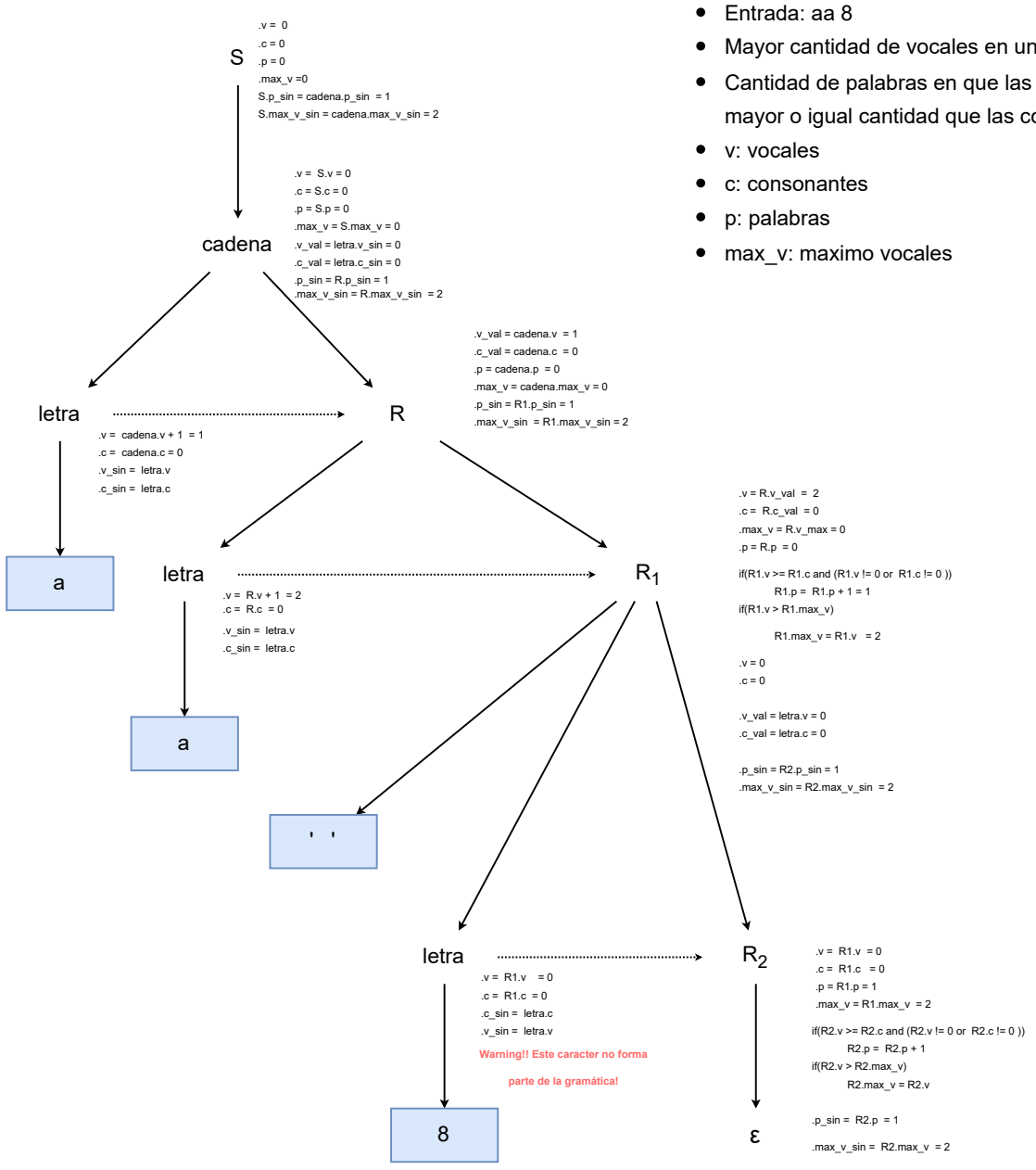
Entrada válida

- Entrada: aa b
- Mayor cantidad de vocales en una o mas palabras: 2
- Cantidad de palabras en que las vocales se encuentran en mayor o igual cantidad que las consonantes: 1
- v: vocales
- c: consonantes
- p: palabras
- max_v: maximo vocales



Entrada inválida

- Entrada: aa 8
- Mayor cantidad de vocales en una o mas palabras: 2
- Cantidad de palabras en que las vocales se encuentran en mayor o igual cantidad que las consonantes: 1
- v: vocales
- c: consonantes
- p: palabras
- max_v: maximo vocales



5- Código fuente del TDS

```
# VENTANITA
def inicio(lista):
    # prepara las variables necesarias para manejar la entrada

    global input, salida, entrada, i, warning

    i = 0

    entrada = lista
    input = entrada[i]
    salida = ''
    warning = ''

    return S()


def S():
    global entrada, salida, warning

    print('\nEntrada: ' + entrada)

    v = 0
    c = 0
    p = 0
    max_v = 0

    p_sin, max_v_sin = cadena(v, c, p, max_v)

    # para la ventanita
    resultado = 'Entrada procesada: ' + salida + "\n" + \
                'Palabras: ' + str(p_sin) + "\n" + \
                'Cantidad maxima de vocales: ' + str(max_v_sin)
+ "\n" + \
                warning

    return resultado


def cadena(v, c, p, max_v):
    v_val, c_val = letra(v, c)
    p_sin, max_v_sin = R(v_val, c_val, p, max_v)

    return p_sin, max_v_sin


def R(v, c, p, max_v):
    global input

    if input == 'a' or input == 'b' or input == 'c' or input ==
'd' or input == 'e' or input == 'f' or input == 'g' or input ==
'h' or input == 'i' or input == 'j' or input == 'k' or input ==
'l' or input == 'm' or input == 'n' or input == 'ñ' or input ==
'o' or input == 'p' or input == 'q' or input == 'r' or input ==
's' or input == 't' or input == 'u' or input == 'v' or input ==
'w' or input == 'x' or input == 'y' or input == 'z':

        v_val, c_val = letra(v, c)
        p_sin, max_v_sin = R(v_val, c_val, p, max_v)
```

```
        return p_sin, max_v_sin

    elif input == ' ':  # fin de una cadena, inicia otra

        match(' ')

        # cantidad de palabras en que las vocales se encuentran
en mayor o igual cantidad que las consonantes
        if v >= c and (
                v != 0 or c != 0):  # si se proceso una cadena
valida v o c no deben ser 0 al mismo tiempo. Ej cadena invalida:
23
            p = p + 1

        # mayor cantidad de vocales en una o mas palabras
        if v > max_v:
            max_v = v

        v = 0
        c = 0

        v_val, c_val = letra(v, c)
        p_sin, max_v_sin = R(v_val, c_val, p, max_v)

        return p_sin, max_v_sin

    else:  # fin de entrada

        # cantidad de palabras en que las vocales se encuentran
en mayor o igual cantidad que las consonantes
        if v >= c and (
                v != 0 or c != 0):  # si se proceso una cadena
valida v o c no deben ser 0 al mismo tiempo. Ej cadena invalida:
23
            p = p + 1

            # mayor cantidad de vocales en una o mas palabras
        if v > max_v:
            max_v = v

        # para que concuerde con las reglas semanticas
        p_sin = p
        max_v_sin = max_v

        return p_sin, max_v_sin


def letra(v, c):
    global input

    if input == 'a':
        match('a')
        v = v + 1
    elif input == 'b':
        match('b')
        c = c + 1
    elif input == 'c':
        match('c')
        c = c + 1
    elif input == 'd':
        match('d')
        c = c + 1
```

```
    elif input == 'e':
        match('e')
        v = v + 1
    elif input == 'f':
        match('f')
        c = c + 1
    elif input == 'g':
        match('g')
        c = c + 1
    elif input == 'h':
        match('h')
        c = c + 1
    elif input == 'i':
        match('i')
        v = v + 1
    elif input == 'j':
        match('j')
        c = c + 1
    elif input == 'k':
        match('k')
        c = c + 1
    elif input == 'l':
        match('l')
        c = c + 1
    elif input == 'm':
        match('m')
        c = c + 1
    elif input == 'n':
        match('n')
        c = c + 1
    elif input == 'ñ':
        match('ñ')
        c = c + 1
    elif input == 'o':
        match('o')
        v = v + 1
    elif input == 'p':
        match('p')
        c = c + 1
    elif input == 'q':
        match('q')
        c = c + 1
    elif input == 'r':
        match('r')
        c = c + 1
    elif input == 's':
        match('s')
        c = c + 1
    elif input == 't':
        match('t')
        c = c + 1
    elif input == 'u':
        match('u')
        v = v + 1
    elif input == 'v':
        match('v')
        c = c + 1
    elif input == 'w':
        match('w')
        c = c + 1
    elif input == 'x':
        match('x')
```

```
        c = c + 1
    elif input == 'y':
        match('y')
        c = c + 1

    else:

        anterior = input  # en match el input se va actualizar y
pierdo el valor anterior

        match('z')

        # con gestion de errores la ejecucion continua y puede
caer cualquier caracter aqui, por eso se coloca el if
        if anterior == 'z':
            c = c + 1

    return v, c


def match(t):
    global input, salida

    if input == t:
        # print('match con '+input)
        salida = salida + input
        avanzar_input()
    else:
        print_warning(input)
        manejo_errores()


def avanzar_input():
    global input, entrada, i, salida

    i = i + 1  # avanza indice

    if i < len(entrada):  # todavia hay caracteres a procesar

        input = entrada[i]

        if not pertenece_a_alfabeto(input):
            print_warning(input)
            manejo_errores()

    else:
        input = ''  # fin de cadena


def manejo_errores():
    # ignora caracteres extranhos y avanza el input hasta encon-
trar uno que pertenezca al alfabeto

    global input
    global entrada
    global i
    global salida

    if i == len(entrada) - 1:  # ultima posicion, ya no se puede
avanzar
        input = ''
```

```
    # mientras haya caracter a procesar y este no pertenezca al
alfabeto
    while i < len(entrada) - 1 and pertenece_a_alfabeto(en-
trada[i]) == False:
        i = i + 1
        input = entrada[i]

        if (pertenece_a_alfabeto(input) == False):

            print_warning(input)

            if i == len(entrada):
                input = ''


def pertenece_a_alfabeto(caracter):
    if caracter == 'a' or caracter == 'b' or caracter == 'c' or
caracter == 'd' or caracter == 'e' or caracter == 'f' or carac-
ter == 'g' or caracter == 'h' or caracter == 'i' or caracter ==
'j' or caracter == 'k' or caracter == 'l' or caracter == 'm' or
caracter == 'n' or caracter == 'ñ' or caracter == 'o' or carac-
ter == 'p' or caracter == 'q' or caracter == 'r' or caracter ==
's' or caracter == 't' or caracter == 'u' or caracter == 'v' or
caracter == 'w' or caracter == 'x' or caracter == 'y' or carac-
ter == 'z' or caracter == ' ':
        return True
    else:
        return False


def print_warning(caracter):
    global warning

    if caracter != '':  # evita mostrar warning del final de ca-
dena

        print("warning: '" + caracter + "'")

        # concatenacion del resultado para ventanita
        warning = warning + "warning: '" + caracter + "'\n"
```

Enlace gitlab: https://github.com/CamilaAlderete/tp-compiladores/tree/master