DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Teoria de lenguajes

Trabajo práctico

Resumen

El presente trabajo se analiza una gramatica de atributos y se genera el codigo para poder generar un archivo con extension svg que contenga la imagen de una formula en formato latex ingresada partiendo de una gramatica ambigua.

Integrante	LU	Correo electrónico
Jabalera Gasperi, Fernando	56/09	fgasperijabalera@gmail.com
Russo, Christian Sebastián	679/10	christian.russo8@gmail.com
Delgado, Alejandro Nahuel	601/11	nahueldelgado@gmail.com

Palabras claves:

TDS, Gramatica, Atributos, Ambiguedad

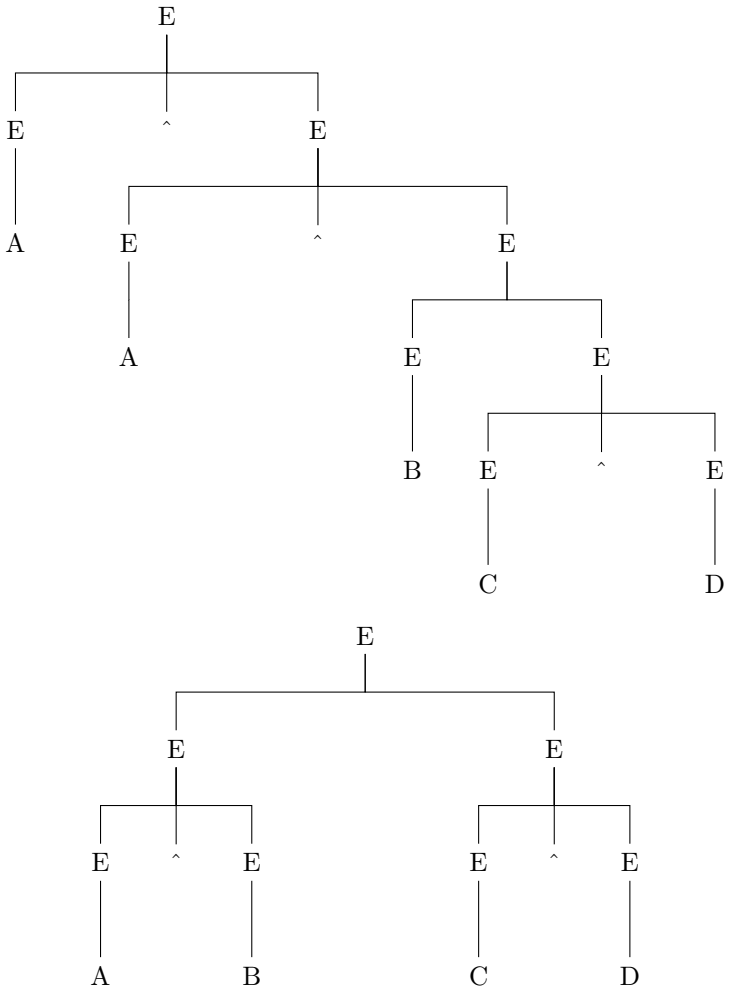
Índice

1. Desambiguando la gramatica

Para utilizar un parser LR lo primero que tuvimos que hacer es desambiguar la gramatica, dado que nuestra gramatica original era:

$$
\begin{aligned}
E \rightarrow \quad & E\ E \\
| \ & E\ \hat{}\ E \\
| \ & E\ \_\ E \\
| \ & E\ \hat{}\ E\ \_\ E \\
| \ & E\ \_\ E\ \hat{}\ E \\
| \ & E\ /\ E \\
| \ & (\ E\ ) \\
| \ & \{\ E\ \} \\
| \ & 1
\end{aligned}
$$

Y esta **es ambigua** dado que tenemos mas de un arbol de derivacion, por ejemplo para la cadena A ^ BC ^ D se la puede derivar de estas maneras:



Entonces para desambiguarla propusimos la siguiente tabla de precedencias y asociaciones.

Operacion	Precedencia	Asociatividad
/	0	Izq
concat	1	Izq
^ y_	2	-
{ } y ()	3	-

quedandonos una gramatica **no ambigua** de la siguiente forma:

E → E / C
E → C
C → CI
C → I
I → A^ A
I → A _ A
I → A^ A _ A
I → A_ A ^ A
I → A
A → { E }
A → (E)
A → l

2. Analizador Lexico

Para poder generar los Tokens utilizamos la herramienta **ply**.

Lo primero que necesitamos fue definir los posibles tokens y sus expresiones regulares.

tokens = (SUPERI,
SUBI,
DIVIDE,
LPAREN,
RPAREN,
LBRACE,
RBRACE,
OPERAND)

t_SUPERI = ^
t_SUBI = _
t_DIVIDE = /
t_LPAREN = (
t_RPAREN =)
t_LBRACE = {
t_RBRACE = }

Nota 1: SUPERI y SUBI hacen referencia a super indice y sub indice respectivamente.

Nota 2: Utilizamos t_ como prefijo de las expreciones regulares porque es asi como **ply** las interpreta

Una vez definidas estas dos cosas **ply** las interpreta y nos genera los tokens correspondientes.

3. Traduccion Dirigida por Sintaxis

3.1. Atributos

Para hacer la Traduccion Dirigada por Sintaxis, lo primero que hacemos es definir los atributos de cada símbolo Terminal y No Terminal.

Atributo	Tipo	-
E.x	int	Heredado
E.y	int	Heredado
E.size	int	Heredado
E.h_top	int	Sintetizado
E.h_buttom	int	Sintetizado
E.ancho	int	Sintetizado
C.x	int	Heredado
C.y	int	Heredado
C.size	int	Heredado
C.h_top	int	Sintetizado
C.h_buttom	int	Sintetizado
C.ancho	int	Sintetizado
I.x	int	Heredado
I.y	int	Heredado
I.size	int	Heredado
I.h_top	int	Sintetizado
I.h_buttom	int	Sintetizado
I.ancho	int	Sintetizado
A.x	int	Heredado
A.y	int	Heredado
A.size	int	Heredado
A.h_top	int	Sintetizado
A.h_buttom	int	Sintetizado
A.ancho	int	Sintetizado

3.2. Explicacion

Luego la TDS que escribimos es la siguiente:

$\mathbf{S} \rightarrow$ {E.x = 0, E.y = 0, E.size = 1} **E**

$\mathbf{E}_1 \rightarrow$ $\{E_2.x = E_1.x + E_1.ancho/2 - E_2.ancho/2,$
$E_2.y = E_1.y - E_1.h\_bottom - c * E_1.Size, E_2.Size = E_1.Size\}\mathbf{E}_2/$
$\{C.x = E_1.x + E_1.ancho/2 - C.ancho/2, C.y = E_1.y + E_1.h\_top + c * E_1.Size, C.Size = E_1.Size\}\mathbf{C}$
$\{E_1.h\_top = E_2.h\_top + E_2.h\_bottom, E_1.h\_bottom = C.h\_bottom + C.h\_top, E_1.ancho = max(c.ancho$

$\mathbf{E} \rightarrow$ **C**

$\mathbf{C}_0 \rightarrow$ $\{C_1.x = C_0.x, C_1.y = C_0.y, C_1.Size = C_0.Size\}\mathbf{C}_1\{I.x = ?, I.y = ?, I.Size = ?\}\mathbf{I}$
$\{C_0.h\_top = max(C_1.h\_top, C_1.h\_bottom), C_0.h\_bottom = max(C_1.h\_top, C_1.h\_bottom),$
$C_0.ancho = C_1.ancho + I.ancho\}$

$\mathbf{C} \rightarrow$ **I**

$\mathbf{I} \rightarrow$ $\{A_1.x = I.x, A_1.y = I.y-((B.h\_top + B.h\_bottom) * 0{,}5), A_1.Size = I.Size\}\mathbf{A}_1$
ˆ $\{A_2.x = A_1.ancho + A_1.x, A_2.y = A_1.y, A_2.Size = A_1.Size * 0{,}7\}\mathbf{A}_2$
$\{I.h\_top = A_1.h\_top + A_2.h\_top * 0{,}5, I.h\_bottom = 0, I.ancho = A_1.ancho + A_2.ancho\}$

$\mathbf{I} \rightarrow$ $\{A_1.x = I.x, A_1.y = I.y-((B.h\_top + B.h\_bottom) * 0{,}5), A_1.Size = I.Size\}\mathbf{A}_1$
_$\{A_2.x = A_1.ancho + A_1.x, A_2.y = A_1.y, A_2.Size = A_1.Size * 0{,}7\}\mathbf{A}_2$
$\{I.h\_top = A_1.h\_top + A_2.h\_top * 0{,}5, I.h\_bottom = 0, I.ancho = A_1.ancho + A_2.ancho\}$

$\mathbf{I} \rightarrow$ **A**ˆ **A** _ **A**

$\mathbf{I} \rightarrow$ **A**_ **A**ˆ **A**

$\mathbf{I} \rightarrow$ {A.x = I.x, A.y = I.y, A.Size = I.Size}**A**
{I.h_top = A.h_top, I.h_bottom = A.h_bottom, I.ancho = A.ancho}

$\mathbf{A} \rightarrow$ **{** {E.x = A.x, E.y = A.y, E.Size = A.Size}**E**
{A.h_top = E.h_top, A.h_bottom = E.h_bottom, A.ancho = E.ancho} **}**

$\mathbf{A} \rightarrow$ **(**{E.x = A.x, E.y = A.y, E.Size = A.Size}**E**
{A.h_top = E.h_top, A.h_bottom = E.h_bottom, A.ancho = E.ancho}**)**

$\mathbf{A} \rightarrow$ $\mathbf{L}\{A.h\_top = k_1 * A.size, A.h\_bottom = 0, A.ancho = k_2 * A.size.\}$

3.3. Implementacion TDS

3.3.1. Reglas Sintacticas

Para implementar las Reglas Sintacticas **ply** nos pide que las definamos de una forma especial, como por ejemplo:

Algorithm 1 Ejemplo de Regla Sintactica

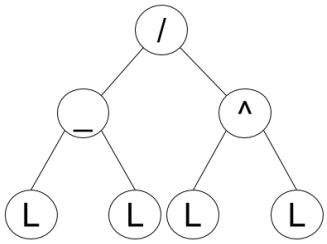
```
def p_expression_divide(p):
'expression : expression DIVIDE concatenation'
p[0] = DivisionNode(p[1], p[3])
```

En el ejemplo se puede ver la regla sintactica para la division. Entre ' ' definimos la produccion para la queremos esta regla y luego con el parametro P hacemos referencia a cada Terminal/No Terminal de la produccion. Utilizamos DivisionNode que es una clase de nuestro Arbol Sintactico

3.3.2. Arbol Sintactico

Para poder implementar la TDS generamos el una estructura de **Arbol Sintactico**, en la cual tenemos la clase nodo y cada nodo hace referencia a sus hijos y a su padre. Definimos nodos unarios, binarios y ternarios para utilizar dependiendo la cantidad de hijos que necesitabamos.

Con esto logramos generar un **Arbol Sintactico** como por ejemplo para la cadena A_B/C^D, el arbol quedaria:



3.4. Recorrido del Arbol Sintactico

Para recorrer el Arbol Sintactico lo que hacemos es separar en tres etapas:

- Bajar1: En esta funcion lo que tenemos que hacer es heredar Size hasta las hojas entonces desde el nodo raiz vamos pasando el atributo **size** hasta llegar a una hoja
- Subir: Una vez con Size en las hojas, sintetizamos **width**, **heightTop**, **heighBot**, hasta la raiz del arbol.
- Bajar2: Por ultimo, nos queda heredar **x** e **y** entonces, los bajamos hasta las hojas.

Una vez terminado estos tres pasados tenemos los atributos acomodados correctamente en el arbol sintactico.

4. Geracion del SVG

Para generar el archivo SVG utilizamos la libreria de python **svgwrite**[1][2]. La forma de utilizarla es la siguiente:

[1] https://pypi.python.org/pypi/svgwrite
[2] Para installar correr: pip install svgwrite

Algorithm 2 Ejemplo de uso de svgwrite

```
# Creamos la nueva imagen con los atributos correspondientes
img = svgwrite.Drawing(filename = "nombre.svg", atributos)
# Agregamos un atributo Group es decir el tag g
group = img.g(atributos)
# Dentro del Group agregamos los tags que querramos, como un Text
group.add(img.Text(atributos))
# Agregamos un tag Line
group.add(img.Line(atributos))
# Se agrega el Group dentro de la imagen
img.add(group)
# Se guarda la imagen
img.save()
```

En particular, en nuestro codigo, la creacion del archivo .svg, la generacion del Group, agregar el Group al svg y guardar el archivo, lo estamos haciendo en **tp.py**. Luego para cada tag se encargan las propias producciones:

- la produccion del operando (OperandNode) escribe en el svg el tag text con los atributos correspondientes
- La produccion de los parentesis (ParenthesisNode) escribe en el svg el tag especial para los parentesis
- La produccion de la division (DivisionNode) agrega el tag Line para la linea devisora

De esta forma estamos generando el archivo .svg llamdo figura.svg.

Nota: Con cada ejecucion del codigo se sobreescribe la imagen anterior con la nueva.

5. Modo de Uso

5.1. Paquetes Necesarios

Para correr nuestro codigo es necesario instalar **svgwriter** de la siguiente forma

```
$ pip install svgwrite
```

5.2. Ejecucion

Parados en la carpeta **src** del proyecto correr:

```
$ python tp.py
parse >> cadena
```

remplazando **cadena** por la ecuacion de entrada. De esta forma se genera el archivo **figura.svg** en la carpeta **src** del proyecto.