DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico

Parser para gramática PEGS

25 de noviembre de 2014 — Teoría de Lenguajes de Programación

Grupo 10

Integrante	LU	Correo electrónico
Garbi, Sebastían	179/05	garbyseba@gmail.com
Sarries, Ana	144/02	anasarries@yahoo.com.ar

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

1. Introducción

2. Gramática

Para interpretar el lenguaje se generó la gramática $G = \langle V_n, V_t, P, Programa \rangle$ donde:

- V_n es: {Elemento, Elementoand, Elementobase, Factor, Main, Masreglas, Numero, Prim, Programa, Reglas, Termino, Trans, Unaregla}
- V_t es:{$, &, (,), *, +, -, ., /, :, <, =, >, BALL, BOX, CB, CG, CR, D, NADA, NUM, REGLA, RX, RY, RZ, S, SX, SY, SZ, TX, TY, TZ, [,], ^,| }
- Producciones P:
  $Programa \rightarrow Reglas\ Main\ Masreglas$
  $Reglas \rightarrow \lambda$
  $Reglas \rightarrow Reglas\ Unaregla$
  $Main \rightarrow \$\ =\ Elemento$
  $Main \rightarrow \$.\ =\ Elemento$
  $Masreglas \rightarrow \lambda$
  $Masreglas \rightarrow Masreglas\ Unaregla$
  $Masreglas \rightarrow Masreglas\ Main$
  $Unaregla \rightarrow$ **REGLA** $=\ Elemento$
  $Unaregla \rightarrow$ **REGLA.** $=\ Elemento$
  $Elemento \rightarrow Elemento \mid Elementoand$
  $Elemento \rightarrow Elementoand$
  $Elementoand \rightarrow Elementoand\ \&\ Elementobase$
  $Elementoand \rightarrow Elementobase$
  $Elementobase \rightarrow Prim$
  $Elementobase \rightarrow Elementobase\ :\ Trans$
  $Elementobase \rightarrow [\ Elemento\ ]$
  $Elementobase \rightarrow Elementobase\ \hat{}\ Numero$
  $Elementobase \rightarrow <\ Elemento\ >$
  $Elementobase \rightarrow$ **REGLA**
  $Elementobase \rightarrow \$$
  $Prim \rightarrow$ **BALL**
  $Prim \rightarrow$ **BOX**
  $Prim \rightarrow$ **NADA**
  $Trans \rightarrow$ **RX** $Numero$
  $Trans \rightarrow$ **RY** $Numero$
  $Trans \rightarrow$ **RZ** $Numero$
  $Trans \rightarrow$ **S** $Numero$
  $Trans \rightarrow$ **SX** $Numero$
  $Trans \rightarrow$ **SY** $Numero$
  $Trans \rightarrow$ **SZ** $Numero$
  $Trans \rightarrow$ **TX** $Numero$
  $Trans \rightarrow$ **TY** $Numero$
  $Trans \rightarrow$ **TZ** $Numero$
  $Trans \rightarrow$ **CR** $Numero$
  $Trans \rightarrow$ **CG** $Numero$
  $Trans \rightarrow$ **CB** $Numero$
  $Trans \rightarrow$ **D** $Numero$
  $Numero \rightarrow Numero\ +\ Factor$
  $Numero \rightarrow Numero\ -\ Factor$
  $Numero \rightarrow Factor$
  $Factor \rightarrow Factor\ *\ Termino$

$Factor \rightarrow Factor\ /\ Termino$
$Factor \rightarrow Termino$
$Termino \rightarrow$ **NUM**
$Termino \rightarrow\ +$ **NUM**
$Termino \rightarrow\ -$ **NUM**
$Termino \rightarrow (\ Numero\ )$
$Termino \rightarrow\ +\ (\ Numero\ )$
$Termino \rightarrow\ -\ (\ Numero\ )$

De V_t cabe destacar el token **NADA** que es el que representa al caractér "_", el token **REGLA** que se corresponde con la expresión regular '[a-zA-Z]+' y el token **NUM** con la expresión regular '\d+(\.\d+)?'.
Los demás tokens se corresponden literalmente con su texto en minúscula, por ejemplo *BALL* con la palabra "ball".
Para que estos últimos no sean tokenizados como **REGLA** se usó el ejemplo de palabras reservadas que aparece en la sección **4.3** de la documentación de ply (`http://www.dabeaz.com/ply/ply.html#ply_nn6`).

3. Implementación de la solución

La solución consiste de dos grandes pasos.
El primero ocurre mientras se ejecuta el analizador sintáctico, en este paso se "sintetiza" en el **NT** que generó la producción el objeto que va a ser mostrado así también las transformaciones que sufre el mismo. Para las producciones donde se define un nombre de regla[1], por ejemplo "*bola* = *ball*|*box*", utilizamos dos diccionarios globales "reglas" y "finales". Si la regla no es final se define solo en el diccionario "reglas", en cambio si es final (tiene '.') se define en ambos diccionarios.
El segundo paso es al momento de mostrar, en este paso se toma del diccionario "reglas" la definición de **$** y se le ejecuta el método mostrar que se encargará de mostrar todos los elementos que se generaron durante el parseo.

Para la solución se crearon dos jerarquías de clases que interactúan entre sí. Una es la que se refiere a las transformaciones:
De Transformacion heredan TransRX, TransRY, TransRZ, TransT, TransS, TransC, TransD, cada una de ellas representa un tipo de transformación según su nombre lo indica. A su vez las instancias directas de Transformacion se corresponde con la transformación identidad, la cual al ser aplicada no efectúa ningún cambio. La otra jerarquía es la de Elementos: De Elemento heredan Primitiva, Compuesta, ElementoPOT, ElementoCorchete y ElementoREGLA. De Primitiva heredan Ball, Box y Nada, y de Compuesta heredan ElementoOR, ElementoAND y ElementoANDTransformaDirecto.
Las clases del subárbol de Primitiva representan cada una un objeto final que se mostrará en la composición visual final.
Las clases del subárbol de Compuesta implementan un patrón "composite" los cuales reenviaran a sus contenidos los mensajes que reciban.
Las clases ElementoCorchete y ElementoPOT también siguen este patrón pero conteniendo un solo elemento.
La clase ElementoCorchete tiene mensajes especiales para interactuar con ElementoPOT, en caso de que ésta última lo contenga, en otro caso simplemente reenvía los mensajes a su elemento contenido.
Todos los Elementos tienen implementados los métodos para interactuar con ElementoPOT de forma tal que éste los repita sin el comportamiento especial que tiene ElementoCorchete.

[1] Estamos utilizando que $ es un nombre de regla

Teniendo en cuenta que puede haber más de una definición del mismo nombre de regla y que esto es equivalente a tener un '|' entre las dos definiciones, todas las definiciones del diccionario reglas son ElementoOR al cual se van agregando las nuevas definiciones, teniendo así un único valor para la misma clave el cual se encargara de decidir cual de las definiciones aplicar[2].

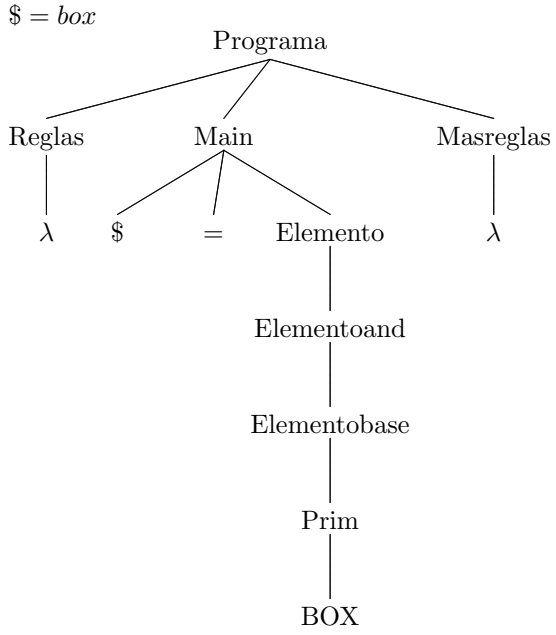
$$\left.\begin{array}{l} Reg = A \\ Reg = B \end{array}\right\} \equiv Reg = A|B \tag{1}$$

Los elementos tienen estados inicializados con la transformación identidad. Cada vez que reciben una transformación se modifica dicho estado.

4. Arboles de derivación

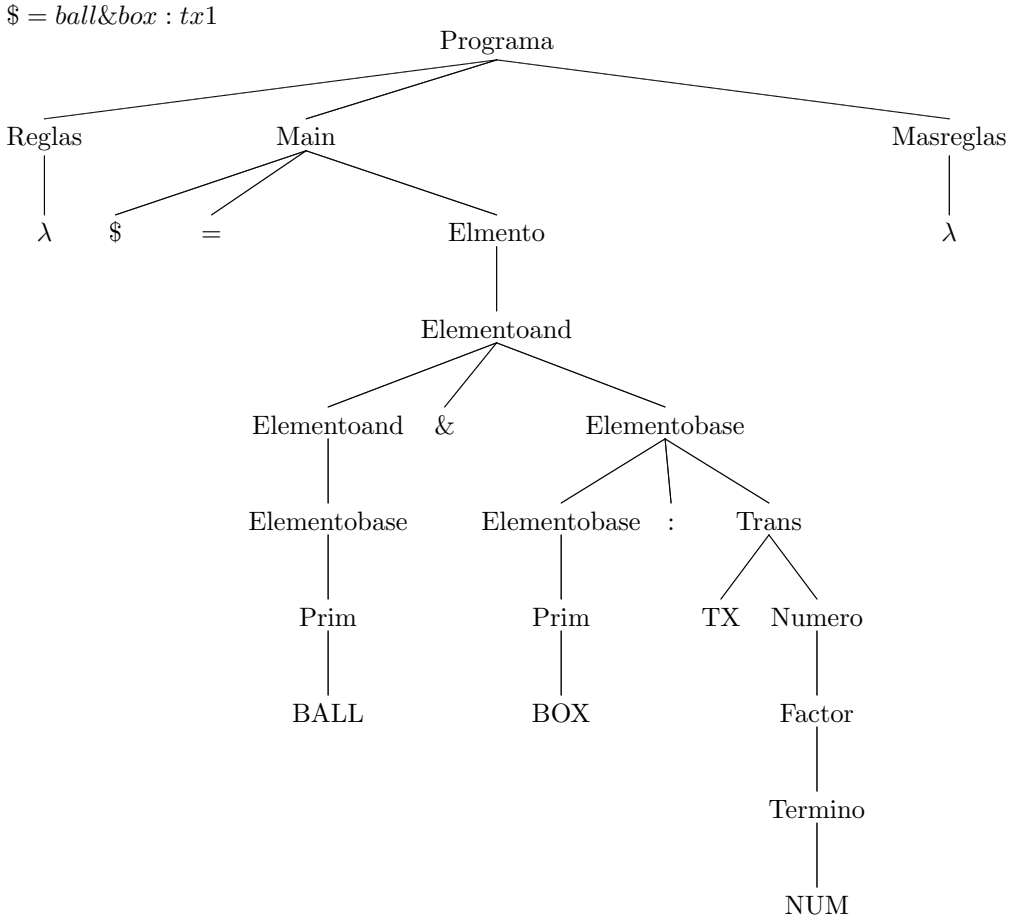
En esta sección se van a presentar algunos ejemplos parseados junto con los arboles de derivación que genera cada entrada.

- $\$ = box$

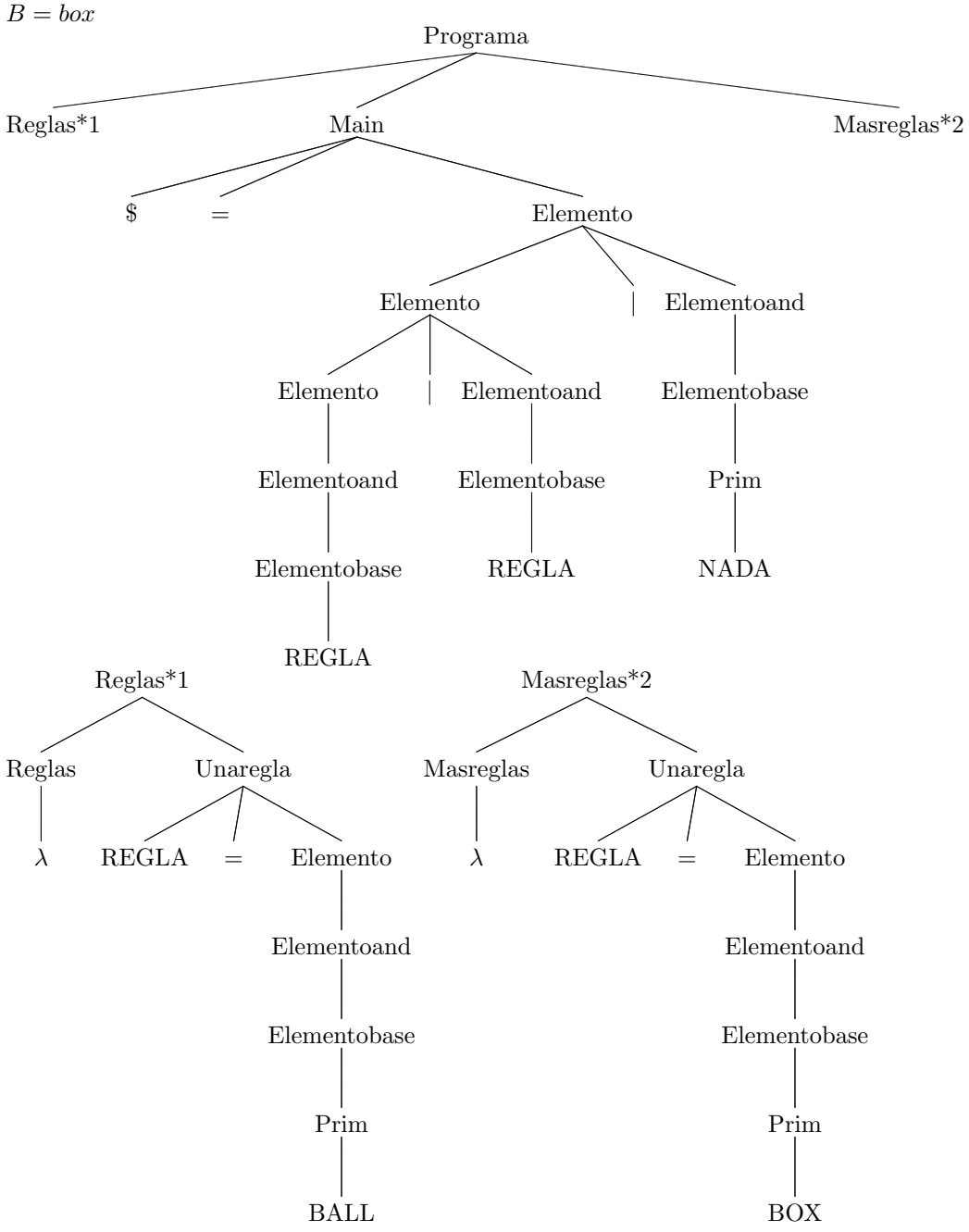


[2]Al ser tratada cada definición como Elementos diferentes, ésto no afecta a los '|' internos de cada definición

- $\$ = ball\&box : tx1$


Programa
Reglas
Main
Masreglas
λ
$
=
Elmento
λ
Elementoand
Elementoand
&
Elementobase
Elementobase
Elementobase
:
Trans
Prim
Prim
TX
Numero
BALL
BOX
Factor
Termino
NUM

- $A = ball$
  $\$ = A|B|\_$
  $B = box$



En el directorio Derivaciones se puede encontrar las listas de producciones utilizadas para generar cada uno de los ejemplos del directorio TL2014C2-TP-Ejemplos. Estas listas conforman las derivaciones mas a la derecha invertidas generadas por el parser LALR

5. Algunos resultados

5.1. Resultados válidos

5.2. Resultados inválidos

6. Ejecución

El programa debe ser ejecutado de la siguiente manera:
python parser.py ruta/al/archivo.peg

En el archivo parser.py hay tres variables boolenas globales útiles para mostrar más información:
mostrarTokens Imprime la lista de Tokens que generó el analizador léxico
mostrarEjes Muestra en la composición visual los ejes X(Rojo) Y(Verde) Z(Azul) centrado en el origen desde -5 a +5 en cada eje
mostrarProducciones Muestra la lista de producciones generada por el parser LALR

7. Detalle de requerimientos

Para la implementación del trabajo práctico se utilizaron las bibliotecas de Python:

- sys
- numpy
- math
- copy
- visual (VPython)
- random
- ply
- re

8. Desiciones

9. Conclusión