Interprete de BOT
Etapa 2
Analizador Sintáctico

Meggie Sánchez 11-10939, Jorge Marcano 11-10566

12/02/2016

1. Introducción

En el presente proyecto, se nos ha pedido crear un interprete para el lenguaje llamado "BOT". El desarrollo para el intérprete se dará en 4 etapas: (I) Análisis lexicográfico, (II) Análisis sintáctico y construcción del árbol sintáctico abstracto, (III) Análisis de contexto e (IV) Intérprete final del lenguaje.

Para esta segunda etapa, se ha dado uso de la implementación del analizador lexicográfico para el lenguaje BOT ya existente en la etapa 1. Esta etapa incluye la definición de la gramática del lenguaje BOT y la contrucción del árbol abstracto correspondiente.

Ya dicho anteriormente, ha sido realizado en el lenguaje de programación Python (versión 3.4), con la herramienta ofrecida por este mismo llamada PLY.

2. Implementación

Para la implementación del analizador sintáctico, se utilizó como base los ejemplos mostrados en la documentación de la herramienta PLY.

Además de apoyarnos en la documentación de PLY, para esta etapa tuvimos que generar un Árbol Sintáctico Abstracto de los programas que son pasados como input al analizador sintáctico. Para esto utilizamos dos clases, ArbolExpr() y ArbolInst().

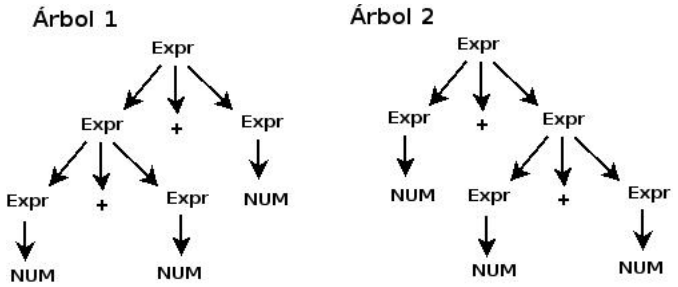
La clase ArbolExpr() es utilizada para generar árboles para todo lo que es considerado una expresión en el lenguaje, desde simples identificadores o elementos de un tipo de variable, hasta expresiones completas, tanto binarias como unarias de tipo aritméticas o booleanas, se crearon dos clases específicas, ArbolUn y ArbolBin para almacenar los casos mencionados anteriormente.

La clase ArbolInst() sirve para generar los árboles para las distintas instrucciones del robot. En este caso, hemos creado una gran cantidad de clases que heredan de ArbolInst y funcionan de la misma manera que ArbolExpr también para almacenar los árboles de cada instrucción específica del lenguaje y su sintaxis, la diferencia entre cada clase es la cantidad de hijos que posee cada árbol y el contenido de cada nodo. La idea es que usando estas clases de árboles, podamos generar un gran árbol para el programa del lenguaje BOT. Este árbol final tendrá un hijo izquierdo en donde se encontrará toda la información de las declaraciones del programa (si las hay) y en el hijo derecho la información de la parte ejecutable del programa. Cada una de estas partes se subdivide posteriormente en árboles de distintos tipos según corresponda, dependiendo de las instrucciones, expresiones o palabras reservadas que se utilizen en el programa.

Es importante mencionar que en la implementación, si el archivo o entrada pasada por parámetro está vacío, el mensaje que arrojará el programa es 'Error sintáctico.'

3. Revisión Teórico-Práctica

3.1. Pregunta 1



a) La frase NUM + NUM + NUM de G1 es ambigua ya que como podemos ver existen dos árboles de derivación distintos que dan como resultado la misma frase.

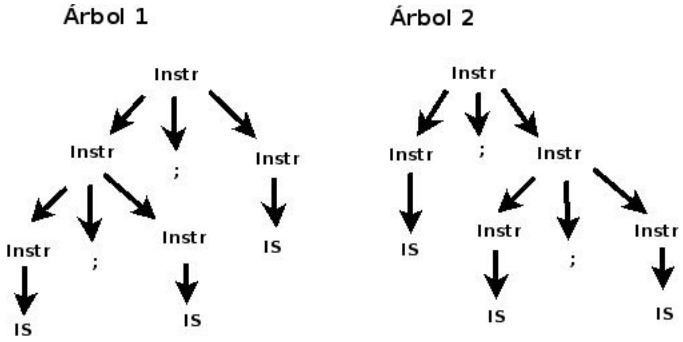
b)

	Izq(G1)	Der(G1)
Expr	-> Expr + T	-> T + Expr
	| T	| T
T	-> NUM	-> NUM

c) Si importa si se asocian las expresiones hacia la izquierda o hacia la derecha. En el caso del operador '-', tenemos como ejemplo: (1-2)-4 = -5 y 1-(2-4) = 3 . Sabemos que -5 es distinto que 3, por lo tanto si importa la asociación.

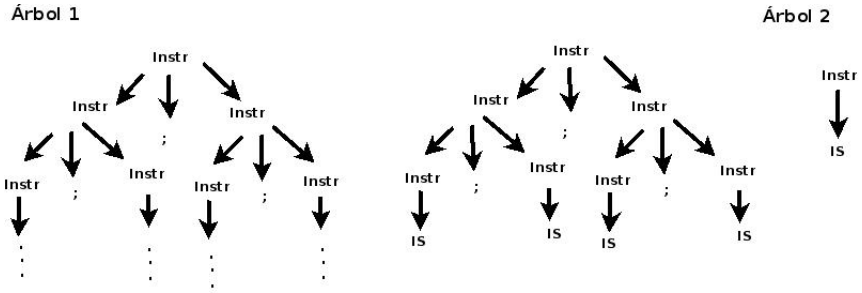
En el caso del operador '/', tenemos como ejemplo: $8/(4/2) = 4$ y $(8/4)/2 = 1$. Sabemos que 4 es distinto de 1, por lo tanto también importa la asociación.

3.2. Pregunta 2



a) Si, G2 presenta los mismos problemas de ambiguedad que G1. Debido a la misma razón ya explicada en la pregunta 1b).

Las únicas frases no ambiguas de G2 son:



Donde sabemos que en el árbol 1, son todos los árboles que siempre terminen con un número par de IS, ejemplificado en el árbol del medio del dibujo.

b) No importa si la ambiguedad se resuelve con asociación hacia la izquierda o hacia la derecha.

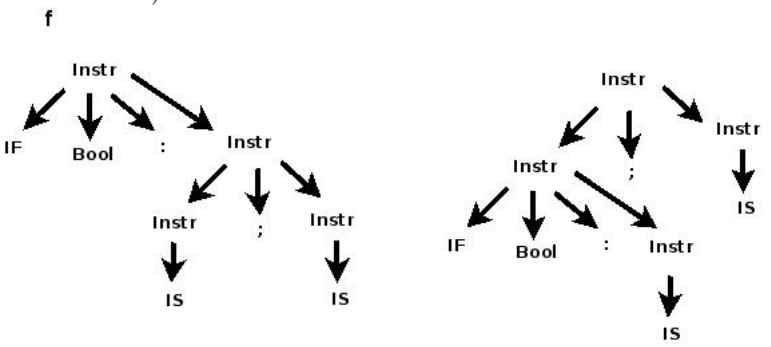
c) Para la frase IS;IS;IS proponemos una derivación más a la izquierda y una derivación más a la derecha:

$Leftmost : Inst => Instr; Instr => IS; Instr => IS; Instr; Instr => IS; IS; Instr => IS; IS; IS$

$Rightmost : Inst => Instr; Instr => Instr; IS => Instr; Instr; IS => Instr; IS; IS => IS; IS; IS$

3.3. Pregunta 3

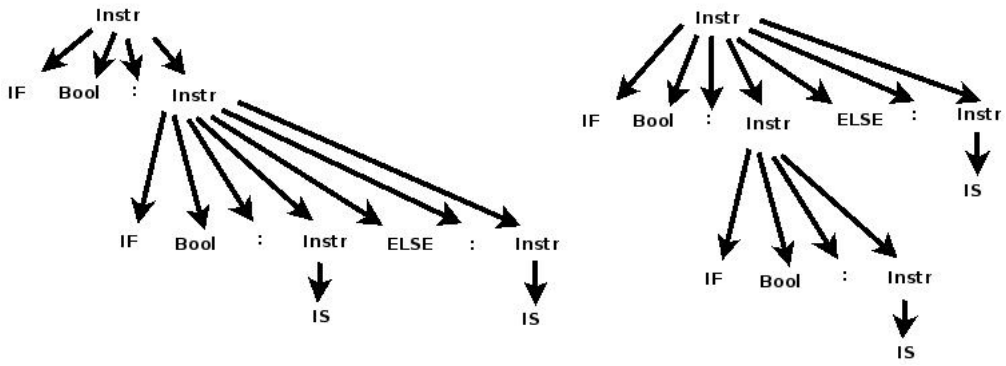
a)



f es una frase ambigua de G3, como podemos observar en la imagen.

b)

g



c) Las dos interpretaciones de f las escribiríamos como:
Para JAVA, C, C++: IF Bool { IS ; IS }
Para Pascal: IF Bool begin IS ; IS end

Las dos interpretaciones de g las escribiríamos como:
Para JAVA, C, C++: IF Bool { IF Bool { IS } ELSE { IS } }
Para Pascal: IF Bool begin IF Bool begin IS end ELSE begin IS end end

d) Las dos interpretaciones de f las escribiríamos como:
IF Bool : IS IS end

Las dos interpretaciones de g las escribiríamos como:
IF Bool : IF Bool : IS ELSE : IS end end

Referencias

[1] David M. Beazley - Documentación de Python-PLY http://www.dabeaz.com/ply/

[2] T. Sudkamp. Languages and Machines . Second Edition. Addison-Wesley, 1997.