Etapa 2: Análisis Sintáctico

Benjamin Amos, Douglas Torres

Enero - Marzo 2016

1 Detalles de Implementacion

Para la implementación de la segunda etapa del proyecto, utilizamos una herramienta, parte de la librería ply, llamada *yacc*, la cual permite el diseño de una gramática para la creación del parser para nuestra lista de tokens, lo cual permitirá el diseño del árbol sintáctico abstracto, como finalidad de esta etapa.

La implementacion del parser fue dividida en dos partes, un archivo para la creación de la gramatica libre de contexto la cual estudiara la sintaxis de un programa escrito en el lenguaje de estudio **BOT** y generara todas las cadenas posibles permitidas por el mismo, y otro archivo en el cual se almacenan las estructuras de datos utilizadas para la creación del árbol sintáctico abstracto, al igual que los métodos que permiten proporcionar la interfaz pedida.

1.1 parser.py

En este archivo se encuentran las reglas gramaticales permitidas por el lenguaje **BOT**. De este modo, se pueden producir las distintas cadenas que acepta el mismo lenguaje. Así mismo, en las reglas, se encuentra la inicialización y agregación de nodos al árbol sintáctico abstracto.

1.2 arboles.py

Este archivo contiene las estructuras de datos utilizadas para la creación del árbol sintáctico abstracto, las cuales se mencionan a continuación:

1. **ArbolInstr**:
   - *CondicionalIf*
   - *IteracionIndef*
   - *Activate*
   - *Deactivate*
   - *Advance*
2. **ArbolBin**
3. **ArbolUn**
4. **Ident**
5. **Bool**
6. **Numero**

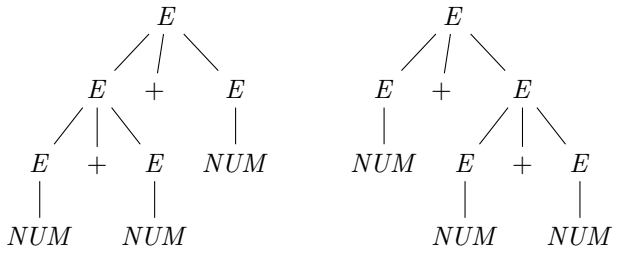
Notemos que solo se encuentran creadas estructuras de datos para el manejo de instrucciones de controlador de **BOT**.

2 Sección Teórico-Práctica

En esta sección se presenta el desarrollo y respuestas para las preguntas propuestas para esta etapa.

1. Basados en la gramática *G1* dada:

   (a) Para demostrar que la frase $NUM{+}NUM{+}NUM$ es ambigua, mostraremos que existen dos árboles de derivación distintos que representan a la frase:

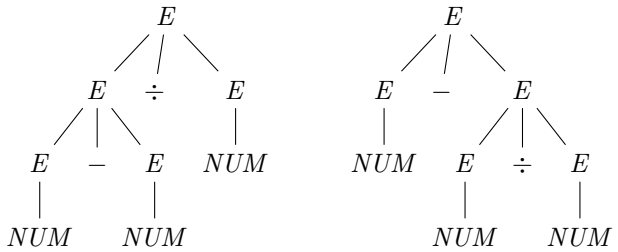


   Notemos entonces que la frase en cuestión es ambigua.

   (b) Sean $Izq(G1)$ y $Der(G1)$ definidas de la siguiente manera:

   - $Izq(G1)$: $Expr \rightarrow Expr + Expr^{'} \mid Expr^{'}$
     $Expr^{'} \rightarrow NUM$
   - $Der(G1)$: $Expr \rightarrow Expr^{'} + Expr \mid Expr^{'}$
     $Expr^{'} \rightarrow NUM$

   (c) En efecto, si importa la forma en la que se asocian las expresiones en esta gramática. Veamos con un ejemplo por qué es relevante. Supongamos que el alfabeto acepta los operadores $-$ y $\div$. Sea la expresión $NUM{-}NUM{\div}NUM$. Construyamos los árboles de derivación respectivos a $Izq(G1)$ y $Der(G1)$ respectivamente:



Notemos que *Izq(G1)* generaría a la expresión $(NUM - NUM) \div NUM$, mientras que *Der(G1)* genera a $NUM - (NUM \div NUM)$

2. Basados en la gramática *G2*, tenemos que:

   (a) Se entiende que la gramática *G2* tiene los mismos problemas de ambigüedad que *G1* ya que, $Expr \equiv Instr$, $+ \equiv\ ;$ y $NUM \equiv IS$. Luego, existen dos o más arboles de derivación diferentes para una frase que reconoce la gramática. Las únicas frases no ambiguas de la gramática *G2* son la frase *IS* y la frase *IS ; IS* ya que sólo hay un árbol para construirlas.

   (b) Dado que la gramática en este caso reconoce secuencias de instrucciones y no operaciones de expresiones, se puede afirmar que no importa el sentido en el que se asocien las expresiones. Para esto, supongamos que existen dos gramáticas *Izq(G2)* y *Der(G2)* tales que:

   - *Izq(G2):* $Instr \to Instr\ ;\ Inst'$
     $\quad\quad\mid\ Instr'$
     $Instr' \to IS$

   - *Der(G2):* $Instr \to Instr'\ ;\ Instr$
     $\quad\quad\mid\ Expr'$
     $Instr' \to IS$

   Dado que los árboles de derivación que se presentan son lo mismos que los de *G1*, notemos que las expresiones que se generarían son $(IS; IS); IS$ y $IS; (IS; IS)$. Siendo éstas instrucciones, no importa el orden en el que se realicen.

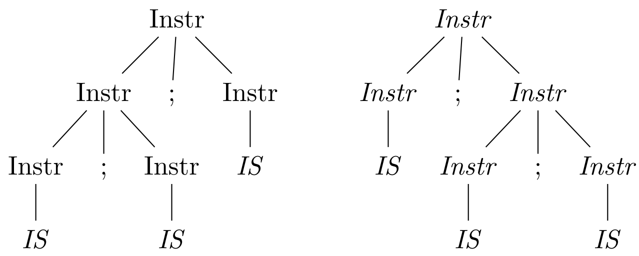
   (c) Veamos una derivación más a la izquierda para la frase $IS; IS; IS$:

   $$Instr \Rightarrow Instr\ ;\ Instr \Rightarrow Instr\ ;\ Instr\ ;\ Instr \Rightarrow Instr\ ;\ Instr\ ;\ IS \Rightarrow Instr\ ;\ IS\ ;\ IS \Rightarrow IS\ ;\ IS\ ;\ IS$$

   Veamos una derivación más a la derecha para la misma frase:

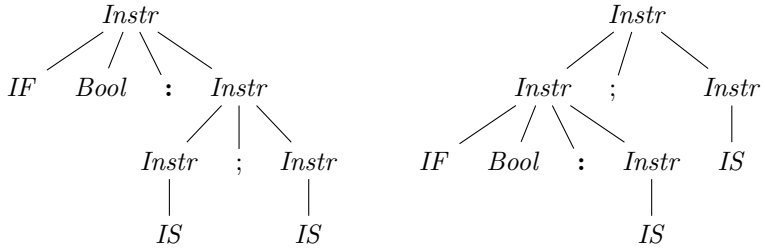
   $$Instr \Rightarrow Instr\ ;\ Instr \Rightarrow Instr\ ;\ Instr\ ;\ Instr \Rightarrow IS\ ;\ Instr\ ;\ Instr \Rightarrow IS\ ;\ IS\ ;\ Instr \Rightarrow IS\ ;\ IS\ ;\ IS$$

   Para ilustrarlo, se muestran dos árboles de derivación diferentes que generan la misma frase:



3. Basados en la gramatica *G3*:

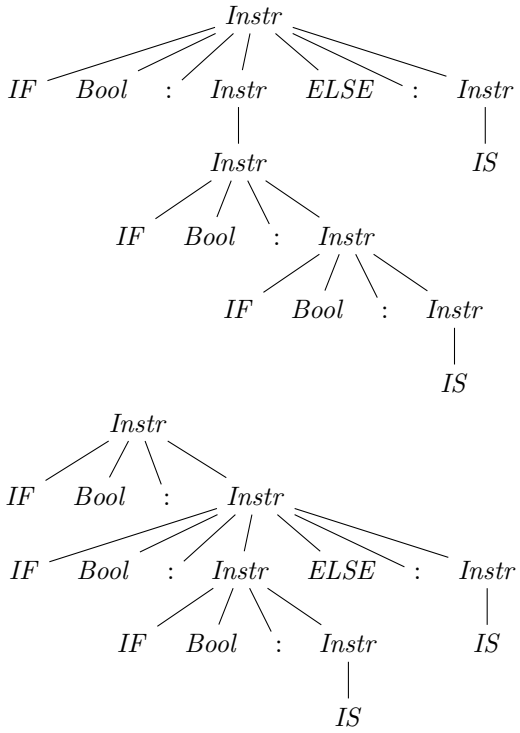
   (a) Mostraremos mediante dos árboles de derivación que la frase *IF Bool* : *IS* ; *IS* es ambigua:



   (b) Una frase *g* de *G3* sin ocurrencias de ; que sea ambigua puede ser:

$$IF\ Bool : IF\ Bool : IF\ Bool : IS\ ELSE : IS$$

   (c) Para mostrar que es ambigua, mostraremos dos árboles de derivación distintos:



   (d) Utilizando llaves, podríamos escribir:

   - Dos interpretaciones de la frase *f* con { y }:
     - *IF Bool* { *IS* ; *IS* }
     - *IF Bool* { *IS* } ; *IS*

- Dos interpretaciones de la frase g con { y }:
  - *IF Bool* { *IF Bool* { *IF Bool* { *IS* } } } *ELSE* { *IS* }
  - *IF Bool* { *IF Bool* { *IF Bool* { *IS* } *ELSE* { *IS* } } }

(e) Utilizando el terminador *end*, podemos escribir:

- Dos interpretaciones de la frase f con { y }:
  - *IF Bool* : *IS* ; *IS end*
  - *IF Bool* : *IS end* ; *IS*
- Dos interpretaciones de la frase g con { y }:
  - *IF Bool* : *IF Bool* : *IF Bool* : *IS end end end ELSE* : *IS end*
  - *IF Bool* : *IF Bool* : *IF Bool* : *IS end ELSE* : *IS end end end*