Universidad Simón Bolivar
Departamento de Computación y Tecnología de la Información
CI3725 - Traductores e Interpretadores
Enero-Marzo 2016

Proyecto 2

Traductores

Samuel Arleo
10-10969

Sergio Terán
11-11020

09 de Febrero, 2016

1. Formulación e Implementación del analizador Sintáctico

Para implementar el Parser se utiliz el analizador sintctico de Yacc de PLY en Python 3.4. Se aplicaron conceptos de gramticas libres de contexto y gramticas de atributos para reconocer las instrucciones del lenguaje BOT, as como para determinar la existencia de errores de sintaxis. El Lexer creado en la entrega anterior permiti obtener los tokens del archivo de entrada.

La precedencia fue especificada en el siguiente orden, con mayor precedencia los operadores al final de la lista y la misma para aquellos en la misma lnea:

- Disyunción
- Conjunción
- Distinto
- Igual
- Menor o igual
- Mayor o igual
- Menor
- Mayor
- Negación
- Suma, Resta
- Multiplicación, División, Módulo
- Negativo

Para reconocer el símbolo negativo se definió el token TkNegativo y se le asignó mayor precedencia que que cualquier otro operador.

Se tomó como regla el uso de mayúsculas para la especificación de los símbolos no terminales, mientras que los terminales poseen el prefijo Tk de la misma forma que el Lexer.

La gramática posee la siguiente forma:

$$
\begin{array}{rcl}
\text{INICIO} & \longrightarrow & \text{CREATE} \\
& | & \text{EXECUTE} \\
\text{CREATE} & \longrightarrow & \text{TkCreate TYPE TkBot IDENT LISTA\_IDENT} \\
& & \text{COMPORTAMIENTO TkEnd DECLARE} \\
& & \text{EXECUTE} \\
\text{DECLARE} & \longrightarrow & \text{TYPE TkBot IDENT LISTA\_IDENT} \\
& & \text{COMPORTAMIENTO TkEnd DECLARE} \\
& | & \lambda \\
\text{TYPE} & \longrightarrow & \text{TkInt} \\
& | & \text{TkBool} \\
& | & \text{TkChar} \\
\text{LISTA\_IDENT} & \longrightarrow & \text{TkComa IDEN LISTA IDENT} \\
& | & \lambda \\
\text{IDENT} & \longrightarrow & \text{TkIdent} \\
\text{COMPORTAMIENTO} & \longrightarrow & \text{TkOn CONDICION TkDosPuntos INST\_ROBOT} \\
& & \text{TkEnd COMPORTAMIENTO} \\
& | & \lambda \\
\text{CONDICION} & \longrightarrow & \text{TkActivation} \\
& | & \text{TkDeactivation} \\
& | & \text{TkDefault} \\
& | & \text{EXP} \\
\text{EXP} & \longrightarrow & \text{EXP TkConjuncion EXP} \\
& | & \text{EXP TkDisyuncion EXP} \\
& | & \text{EXP TkIgual EXP} \\
& | & \text{EXP TkDistinto EXP} \\
& | & \text{IDENT} \\
& | & \text{LITERAL\_BOOL} \\
& | & \text{TkParAbre EXP TkParCierra} \\
& | & \text{Tkresta EXP \%prec TkNegativo} \\
& | & \text{EXP TkMenor EXP} \\
& | & \text{EXP TkMenorIgual EXP} \\
& | & \text{EXP TkMayor EXP} \\
& | & \text{EXP TkMayorIgual EXP} \\
& | & \text{EXP TkSuma EXP} \\
& | & \text{EXP TkResta EXP} \\
& | & \text{EXP TkMult EXP} \\
& | & \text{EXP TkDiv EXP} \\
& | & \text{EXP TkMod EXP} \\
& | & \text{TkNum} \\
\text{LITERAL\_BOOL} & \longrightarrow & \text{TkTrue} \\
& | & \text{TkFalse} \\
\text{INST\_ROBOT} & \longrightarrow & \text{TkStore EXPRESION TkPunto INST\_ROBOT\_A} \\
& | & \text{TkCollect COLLECT TkPunto INST\_ROBOT\_A}
\end{array}
$$

| | TkDrop EXPRESION TkPunto INST_ROBOT_A
| | DIRECCION TkPunto INST_ROBOT_A
| | DIRECCION EXP TkPunto INST_ROBOT_A
| | READ TkPunto INST_ROBOT_A
| | TkSend TkPunto INST_ROBOT_A
| | TkRecieve TkPunto INST_ROBOT_A

INST_ROBOT_A ⟶ INST_ROBOT
| | λ

EXPRESION ⟶ EXP
| | TkCaracter

DIRECCION ⟶ TkLeft
| | TkRight
| | TkUp
| | TkDown

COLLECT ⟶ TkAs IDENT
| | λ

READ ⟶ TkRead
| | Tk Read TkAs IDENT

EXECUTE ⟶ TkExecute INST_CONTROLADOR TkEnd

INST_CONTROLADOR ⟶ TkActivate IDENT LISTA_IDENT
TkPunto INST_CONTROLADOR_A
| | TkAdvance IDENT LISTA_IDENT TkPunto
INST_CONTROLADOR_A
| | TkDeactivate IDENT LISTA_IDENT
TkPunto INST_CONTROLADOR_A
| | TkIf EXP TkDosPuntos CONTENIDO TkEnd
INST_CONTROLADOR_A
| | TkWhile EXP TkDosPuntos INST_CONTROLADOR
TkEnd INST_CONTROLADOR_A
| | INICIO

CONTENIDO ⟶ INST_CONTROLADOR
| | INST_CONTROLADOR TkElse TkDosPuntos
INST_CONTROLADOR

INST_CONTROLADOR_A ⟶ INST_CONTROLADOR
| | λ

La impresión del árbol se llevó a cabo mediante la creación de árboles. La clase principal arbol posee los atributos type que permite conocer el tipo de instrucción o expresión que almacena el nodo y una lista llamada hijos que contiene a los nodos del nivel siguiente del árbol. Los métodos de esta clase fueron creados para imprimir las hojas dependiendo del tipo de estas. Para evitar mostrar por la salida estándar las instrucciones de robot se utilizó el parámetro imprimir que cambia a True cuando una hoja es de tipo **EXECUTE**, y se vuelve a colocar en False cuando la funcin es aplicada sobre un nodo es del tipo **INSTRUCCIONES_ROBOT**.

2. Revisión Teórico-Practica

Pregunta 1

(a) Para demostrar que la frase es ambigua, basta encontrar dos derivaciones *leftmost* para la misma frase.

$$\textbf{Expr} \Longrightarrow \textbf{Expr} + \text{Expr} \Longrightarrow \textbf{Expr} + \text{Expr} + \text{Expr} \Longrightarrow \text{NUM} + \textbf{Expr} + \text{Expr} \Longrightarrow \text{NUM} + \text{NUM} + \textbf{Expr} \Longrightarrow \text{NUM} + \text{NUM} + \text{NUM} \quad (1.1)$$

$$\textbf{Expr} \Longrightarrow \textbf{Expr} + \text{Expr} \Longrightarrow \text{NUM} + \textbf{Expr} \Longrightarrow \text{NUM} + \textbf{Expr} + \text{Expr} \Longrightarrow \text{NUM} + \text{NUM} + \textbf{Expr} \Longrightarrow \text{NUM} + \text{NUM} + \text{NUM} \quad (1.2)$$

Las derivaciones *leftmost* (1.1) y (1.2) producen la misma frase, luego podemos decir que la frase $NUM+NUM+NUM$ de G_1 es ambigua.

(b) Izq(G_1): ({Expr},{+,**NUM**},$P1_{Izq}$,Expr)
Donde $P1_{Izq}$:

Expr	$\longrightarrow$	Expr + **NUM**
	|	**NUM**

Der(G_1):({Expr} , {+ , **NUM**} , $P1_{Der}$, Expr)
Donde $P1_{Der}$:

Expr	$\longrightarrow$	**NUM** + Expr
	|	**NUM**

(c) En el caso del + la asociatividad no afecta el resultado final. Al contrario, en el caso del − y el / el resultado final se ve afectado dependiendo de la asociatividad usada.

Pregunta 2

(a) Podemos afirmar que G_2 presenta los mismos problemas de ambiguedad que G_1.

Las únicas frases no ambiguas serian:

1) IS;IS
2) IS

(b) Si importa, ya que dependiendo de la asociatividad usada, el orden en el que se ejecuten las instrucciones puede variar, y por ende el estado final de las variables puede no ser el mismo.

(c) *Leftmost*:

$$\textbf{Instr} \Longrightarrow \textbf{Instr} \ ; \text{Instr} \Longrightarrow \textbf{Instr} \ ; \text{Instr} \ ; \text{Instr} \Longrightarrow \text{IS} \ ; \textbf{Instr} \ ; \text{Instr} \Longrightarrow \text{IS} \ ; \textbf{Instr} \ ; \text{Instr} \Longrightarrow \text{IS} \ ; \text{IS} \ ; \textbf{Instr} \Longrightarrow \text{IS} \ ; \text{IS} \ ; \text{IS}$$

Rightmost:

Instr $\Longrightarrow$ Instr ; **Instr** $\Longrightarrow$ Instr ; Instr ; **Instr** $\Longrightarrow$ Instr ; **Instr** ; IS $\Longrightarrow$ **Instr** ; IS ; IS $\Longrightarrow$ IS ; IS ; IS

Pregunta 3

(a) Para mostrar que la frase f es ambigua basta con hallar dos derivaciones $leftmost$ distintas que la generen:

Instr $\Longrightarrow$ IF Bool: **Instr** $\Longrightarrow$ IF Bool: **Instr** ; Instr $\Longrightarrow$ IF Bool: IS ; **Instr** $\Longrightarrow$ IF Bool: IS ; IS (3.1)

Instr $\Longrightarrow$ **Instr** ; Instr $\Longrightarrow$ IF Bool: **Instr** ; Instr $\Longrightarrow$ IF Bool: IS ; **Instr** $\Longrightarrow$ IF Bool: IS ; IS (3.2)

Podemos ver que las derivaciones (3.1) y (3.2) generan la misma frase f, luego podemos decir que f es una frase ambigua en G_3

(b) Consideremos la frase g: IF Bool: IF Bool: **IS** ELSE **IS**
Y consideremos las siguientes derivaciones $leftmost$:

Instr $\Longrightarrow$ IF Bool:**Instr** ELSE Instr $\Longrightarrow$ IF Bool: IF Bool: **Instr** ELSE Instr $\Longrightarrow$ IF Bool: IF Bool: **IS** ELSE **Instr** $\Longrightarrow$ IF Bool: IF Bool: **IS** ELSE **IS** (3.3)

Instr $\Longrightarrow$ IF Bool:**Instr** $\Longrightarrow$ IF Bool: IF Bool: **Instr** ELSE Instr $\Longrightarrow$ IF Bool: IF Bool: IS ELSE **Instr** $\Longrightarrow$ IF Bool: IF Bool: IS ELSE **IS** (3.4)

Vemos que las derivaciones (3.3) y (3.4) generan la misma frase g, luego la frase g es ambigua en G_3.

(c) f: IF Bool: { IS ; IS }
f: IF Bool: { IS } ; IS

g: IF Bool:{ IF Bool: { IS } ELSE { IS }}
g: IF Bool:{ IF Bool: { IS }} ELSE { IS }

(d) f: IF Bool: IS end; IS
f: IF Bool: IS ; IS end

g: IF Bool: IF Bool: IS end ELSE IS end
g: IF Bool: IF Bool: IS ELSE IS end end