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

Proyecto 3

Traductores

Samuel Arleo
10-10969

Sergio Terán
11-11020

07 de Marzo, 2016

1. Formulación e Implementación

Para realizar el análisis semántico del lenguaje BOT se utilizó la noción de gramática de atributos, ya que se empleo una variable global tabla (atributo heredado). Fueron necesarios algunos ajustes en la gramática e implementar nuevas estructuras de datos tales como la clase pila, datos y tabla. Algunas de las decisiones de diseo fueron:

* Todas las variables declaradas dentro de la lista de comportamientos de un robot poseen el mismo tipo del robot (incluyendo a la variable asociada al robot me)
* La variable asociada al robot me es distinta para cada robot, as la declaración pudiera haber sido hecha con una lista de robots.
* Las el numero de instrucciones de robot default puede ser maximo 1 para cada robot.
* Una instrucción de robot activation / deactivation no puede aparecer dos veces seguidas en una lista de comportamientos.
* Una instrucción de robot activation no puede ocurrir luego de otro activation
* Una instrucción de robot deactivation no puede ocurrir luego de otro deactivation
* La aceptación de expresiones de la forma 'a' = 'b' fue realizada utilizando la gramática, por lo que un error en este tipo de expresiones sera visto como un error sintáctico. * Las incorporaciones de alcance contaran con el acceso a las variables declaradas en los niveles superiores

2. Revisión Teórico-Practica

Pregunta 1

(a)

(a.1) $G_1 = (\{S\},\{a\},\{S \longrightarrow Sa, S \longrightarrow \lambda\},S)$

Determinemos si la gramática

$$\begin{aligned} S &\longrightarrow Sa \\ S &\longrightarrow \lambda \end{aligned}$$

Es LR(1) y construyamos su analizador sintáctico. Comenzamos por aumentar la gramatica con el símbolo S' y agregando el símbolo \$ al final de la primera entrega. A demás enumeramos las producciones.

(i) $S' \longrightarrow S\$$
(ii) $S \longrightarrow Sa$
(iii) $S \longrightarrow \lambda$

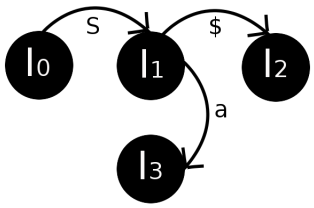
Construimos los conjuntos FIRST y FOLLOW para los simbolos no terminales:

$\text{FIRST}(S') = \{ \lambda , a \}$
$\text{FIRST}(S) = \{ \lambda , a \}$
$\text{FOLLOW}(S') = \{ \$ \}$
$\text{FOLLOW}(S) = \{ a, \$ \}$

El conjunto de clauduras nos queda:

$$\begin{aligned} I_0 :\quad & S' \longrightarrow \cdot S\$ \\ & S \longrightarrow \cdot Sa \\ & S \longrightarrow \cdot \\ I_1 :\quad & S' \longrightarrow S\cdot\$ \\ & S \longrightarrow S\cdot a \\ I_2 :\quad & S' \longrightarrow S\$\cdot \\ I_3 :\quad & S \longrightarrow Sa\cdot \end{aligned}$$

Construimos el automata de prefijos viables, que nos queda de la forma:



Ahora podemos constriur la tabla de parsing $SLR(1)$:

	Acciones		Goto	
	a	$	S'	S
I_0	r(iii)	r(iii)		1
I_1	s(3)	s(2)		
I_2		*acc*		
I_3	r(ii)	r(ii)		

(a.2) G_1 = ({S},{a},{S $\longrightarrow$ aS,S $\longrightarrow$ λ},S)

Determinemos si la gramática

$$\begin{aligned} \text{S} &\longrightarrow \text{aS} \\ \text{S} &\longrightarrow \lambda \end{aligned}$$

Es LR(1) y construyamos su analizador sintáctico. Comenzamos por aumentar la gramatica con el símbolo S' y agregando el símbolo \$ al final de la primera entrega. A demás enumeramos las producciones.

(i) S' $\longrightarrow$ S\$
(ii) S $\longrightarrow$ aS
(iii) S $\longrightarrow$ λ

Construimos los conjuntos FIRST y FOLLOW para los simbolos no terminales:

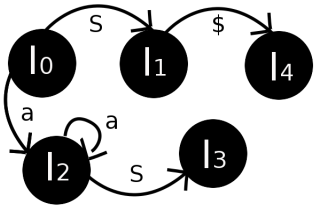
FIRST(S') = FIRST(S) = { λ , a }
FOLLOW(S') = FOLLOW(S) = { \$ }

El conjunto de clauduras nos queda:

$$\begin{aligned} I_0 \ : \quad & S' \longrightarrow \cdot\text{S\$} \\ & S \longrightarrow \cdot\text{aS} \\ & S \longrightarrow \cdot \\ I_1 \ : \quad & S' \longrightarrow \text{S}\cdot\text{\$} \\ I_2 \ : \quad & S \longrightarrow \text{a}\cdot\text{S} \\ & S \longrightarrow \cdot\text{aS} \\ & S \longrightarrow \cdot \end{aligned}$$

I_3 : $S \longrightarrow$ aS$\cdot$
I_4 : $S' \longrightarrow$ S\$ $\cdot$

Construimos el automata de prefijos viables, que nos queda de la forma:



Ahora podemos constriur la tabla de parsing $SLR(1)$:

	Acciones		Goto	
	a	\$	S'	S
I_0	s(2)	r(iii)		1
I_1		s(4)		
I_2	s(2)	r(iii)		3
I_3		r(ii)		
I_4		*acc*		

(b) Comparando las tablas de $G1_i$ y $G1_d$ vemos que la tabla de parsing de $G1_i$ contiene menos filas que $G1_d$.

En la pila, $G1_i$, consume un maximo de 3 items en la pila, en el caso especifico de la frase *aaa* la pila para $G1_d$ puede llegar a tener hasta 4 items, por lo que podemos ver que $G1_i$ es mas eficiente en cuanto al consumo de memoria por parte de la pila.

a demás vemos que para reconocer las frases, la cantidad de movimientos realizados por ambos es de $2 * n + 3$ po lo que el tiempo de ejecucion es de orden $O(n)$ donde n es la cantidad de 'a' en la frase.

Pregunta 2

(a)
(i) $S' \longrightarrow Instr$
(ii) $Instr \longrightarrow Instr$; $Instr$
(iii) $Instr \longrightarrow$ IS

FIRST(S') = FIRST($Instr$) = { IS }
FOLLOW(S') { \$ }
FOLLOW($Instr$) { ;,\$, IS }

$$
\begin{array}{lll}
I_0 & : & S' \longrightarrow \cdot Instr\ \$ \\
 & & Instr \longrightarrow \cdot Insrr\ ;\ Instr \\
 & & Instr \longrightarrow \cdot\ \text{IS} \\
I_1 & : & S' \longrightarrow Instr\cdot\ \$ \\
 & & Instr \longrightarrow Instr\cdot\ ;\ Instr \\
I_2 & : & Instr \longrightarrow \text{IS}\cdot \\
I_3 & : & S' \longrightarrow Instr\$\cdot \\
I_4 & : & Instr \longrightarrow Instr\ ;\ \cdot\ Instr \\
 & & Instr \longrightarrow \cdot Instr\ ;\ Instr \\
 & & Instr \longrightarrow \cdot\text{IS} \\
I_5 & : & Instr \longrightarrow Instr\ ;\ Instr\ \cdot \\
 & & Instr \longrightarrow Instr\cdot\ ;\ Instr
\end{array}
$$

En la regla I_5 vemos que existe un conflicto

(b) Este conflicto, del tipo $shift/reduce$, lo intentaremos solucionar usando el algoritmo de SLR(1), apoyandonos con los FIRST y FOLLOW que ya hemos calculado.

	Acciones			Goto	
	;	IS	$	S'	$Instr$
I_0		s(2)			1
I_1	s(4)		s(3)		
I_2	r(iii)	r(iii)	r(iii)		
I_3			acc		
I_4		s(2)			
I_4	**s(4)/r(ii)**	r(ii)	r(ii)		

(c) Secuencia de reconocimiento para la frase IS;IS;IS, dando prioridad al $shift$ en el conflicto $shift/reduce$

Pila	Entrada	Acción	Salida
I_0	IS;IS;IS$	s(2)	
$I_2\ I_0$	;IS;IS$	r(iii)	(iii)
$I_1\ I_0$	;IS;IS$	s(4)	(iii)
$I_4\ I_1\ I_0$	IS;IS$	s(2)	(iii)
$I_2\ I_4\ I_1\ I_0$	;IS$	r(iii)	(iii), (iii)
$I_5\ I_4\ I_1\ I_0$	;IS$	s(4)	(iii), (iii)
$I_4\ I_5\ I_4\ I_1\ I_0$	IS$	s(2)	(iii), (iii)
$I_2\ I_4\ I_5\ I_4\ I_1\ I_0$	$	r(iii)	(iii), (iii), (iii)
$I_5\ I_4\ I_5\ I_4\ I_1\ I_0$	$	r(ii)	(ii), (iii), (iii), (iii)
$I_5\ I_4\ I_1\ I_0$	$	r(ii)	(ii), (ii), (iii), (iii), (iii),
$I_1\ I_0$	$	s(3)	(ii), (ii), (iii), (iii), (iii),
$I_3\ I_1\ I_0$	$	*acc*	(ii), (ii), (iii), (iii), (iii),

Secuencia de reconocimiento para la frase IS;IS;IS, dando prioridad al *reduce* en el conflicto $shift/reduce$

Pila	Entrada	Acción	Salida
I_0	IS;IS;IS$	s(2)	
$I_2\ I_0$	;IS;IS$	r(iii)	(iii)
$I_1\ I_0$	;IS;IS$	s(4)	(iii)
$I_4\ I_1\ I_0$	IS;IS$	s(2)	(iii)
$I_2\ I_4\ I_1\ I_0$	;IS$	r(iii)	(iii), (iii)
$I_5\ I_4\ I_1\ I_0$	;IS$	r(ii)	(ii), (iii), (iii)
$I_1\ I_0$	;IS$	s(4)	(ii), (iii), (iii)
$I_4\ I_1\ I_0$	IS$	s(2)	(ii), (iii), (iii)
$I_2\ I_4\ I_1\ I_0$	$	r(iii)	(iii), (ii), (iii), (iii)
$I_5\ I_4\ I_1\ I_0$	$	r(ii)	(ii), (iii), (ii), (iii), (iii)
$I_1\ I_0$	$	s(3)	(ii), (iii), (ii), (iii), (iii)
$I_3\ I_1\ I_0$	$	*acc*	(ii), (iii), (ii), (iii), (iii)

Podemos concluir que a pesar de que en ambos casos se genera la misma frase sin asociatividades 'explicitas', al priorizar $shift$ nos genera una asociatividad 'inplicita' a la izquierda, mientras que al priorizar *reduce* nos genera una asociatividad 'inplicita' a la derecha.

(d) Para comparar la eficiencia de ambas opciones, reconocimos frase de tamano $IS(;IS)^2$, $IS(;IS)^3$, $IS(;IS)^4$ y $IS(;IS)^5$ para esrudiar sus comportamientos.

Comparando el comportamiento de las pilas en ambas alternativas, vemos que dando prioridad al $shift$ el tamano maximo de la pila es $2 * (n + 1)$ para frases de la forma $IS(IS)^n$ mientras que para el

reduce el tamano maximo de la pila es 4 de manera constante.

Por otro lado vemos que ambas alternativas cuestan $4(n+1)$ pasos para aceptar la frase, luego, en cuestion de tiempo son de orden $O(n)$.

Podemos concluir que en cuestion de eficiencia la alternativa de *reduce* es mas conveniente ya que cuesta la misma cantidad de tiempo, pero utiliza menos memoria en su pila.