

Etapa 3: 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 implementación del parser fue dividida en dos partes, un archivo para la creación de la gramática 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:

- *Condicionallf*
- *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 las gramáticas $G1_i$ y $G1_d$ dadas:

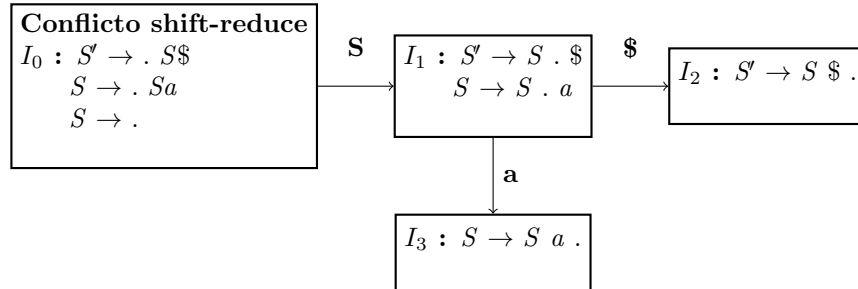
(a) Para construir el analizador sintáctico $LR(0)$ de $G1_i$, modificamos la gramática y queda:

(i) $S' \rightarrow S\$$

(ii) $S \rightarrow Sa$

(iii) λ

Vemos que ocurre un conflicto **shift-reduce** en el estado I_0 , como se ve en los siguientes estados:



Resolvemos el conflicto calculando el *First* y después el *Follow* y usamos este último para saber como resolver el **shift-reduce**:

	FIRST	FOLLOW
S'	λ	$\$$
S	λ	$\$, a$

Luego, procedemos a crear la **Tabla de Parsing**, que queda:

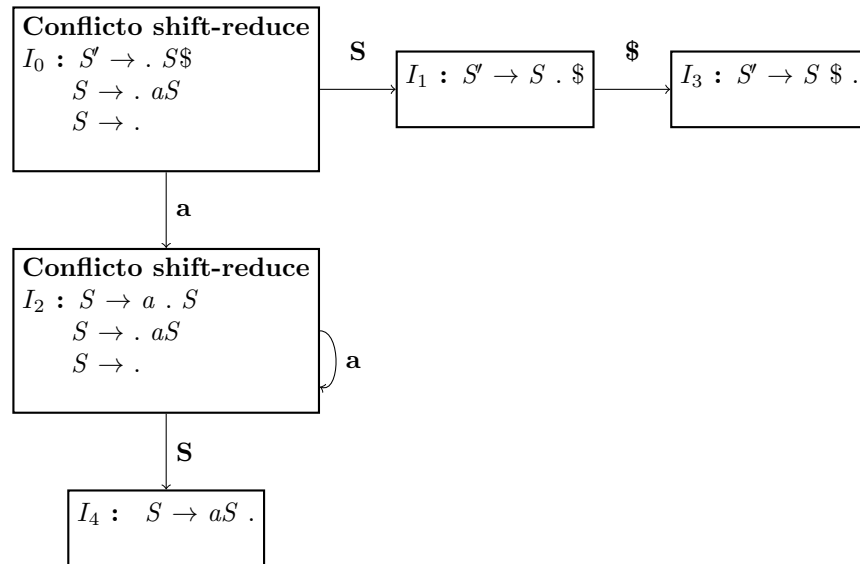
Estado	Acciones		Go to	
	a	\$	S'	S
I_0	reducir(iii)			1
I_1	avanzar(3)	avanzar(2)		
I_2		aceptar		
I_3	reducir(ii)			

Así, vemos que ésta gramática es finalmente $LR(1)$.

Para construir el analizador sintáctico $LR(0)$ de $G1_d$, modificamos la gramática y queda:

- (i) $S' \rightarrow S\$$
- (ii) $S \rightarrow aS$
- (iii) $\epsilon \mid \lambda$

Vemos que ocurre un conflicto **shift-reduce** en los estados I_0 y I_2 , como se ve en los siguientes estados:



Resolvemos el conflicto calculando el *First* y después el *Follow* y usamos este último para saber como resolver el **shift-reduce**:

	FIRST	FOLLOW
S'	a,λ	\$
S	a,λ	\$

Luego, procedemos a crear la **Tabla de Parsing**, que queda:

Estado	Acciones		Go to	
	a	\$	S'	S
I_0	avanzar(2)	reducir(iii)		1
I_1		avanzar(3)		
I_2	avanzar(2)	reducir(iii)		4
I_3		aceptar		
I_4	reducir(ii)			

Así, vemos que ésta gramática es finalmente $LR(1)$.

- (b) Para evaluar la eficiencia en cuanto a espacio y tiempo de las gramáticas pedidas se hicieron los reconocimientos de frases tales como a , aa y aaa para cada una de las gramáticas.

Con esto se logró identifica un patrón tal que para $G1_i$ se usan $2n+3$ pasos en la cantidad de pasos que se necesitan para reconocer cualquier frase de $\mathbf{L(a^*)}$, así como que empila hasta 3 items en la pila.

Para el caso de $G1_d$ vemos que se usan la misma cantidad de pasos $2n+3$ pero en la pila se llegan a empilar hasta 4 items, por lo que en terminos de espacio es más eficiente usar $G1_i$ pero en orden de complejidad ($\mathbf{O(n)}$) es exactamente la misma para ambas gramáticas. El orden de complejidad viene dado por la cantidad de pasos necesarios para reconocer cada palabra del lenguaje y vemos que depende de n que es la cantidad de lestras a en la frase, luego, ésta complejidad es lineal.

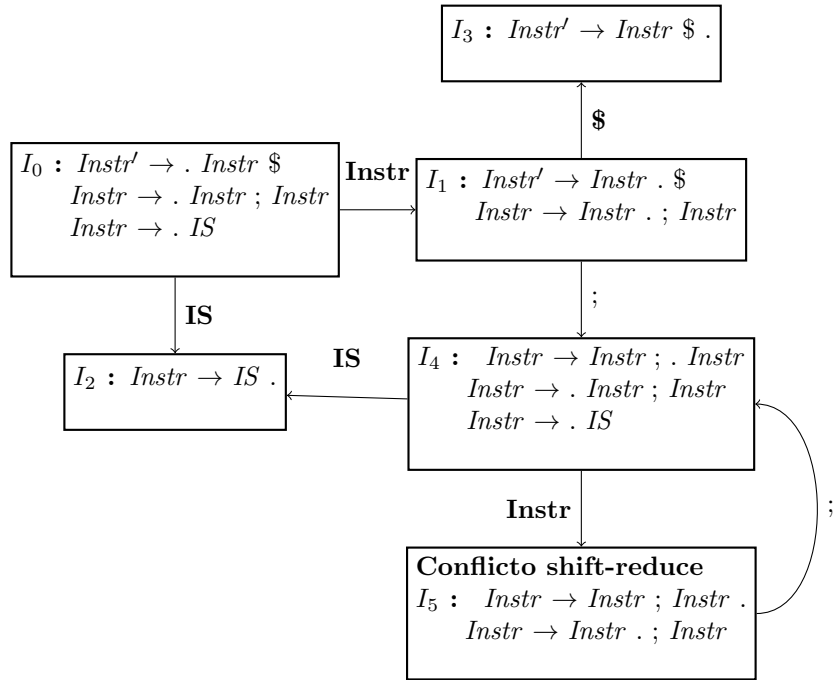
2. (a) Basados en la gramática dada, construimos un analizador sintáctico, empezando por transformar la gramática a:

$$(i) \text{ Instr}' \rightarrow \text{Instr}\$$$

$$(ii) \text{ Instr} \rightarrow \text{Instr};\text{Instr}$$

$$(iii) \quad \quad \quad | \text{ IS}$$

Vemos que ocurre un conflicto **shift-reduce** en el estado I_5 , como se ve en los siguientes estados:



Tratamos de resolver el conflicto calculando el *First* y después el *Follow* y usamos este último para saber como resolver el **shift-reduce**:

	FIRST	FOLLOW
Instr'	IS	\$
Instr	IS	;\$

Luego vemos que en la **Tabla de Parsing** hay un conflicto pues con ; puedo tanto avanzar como reducir al mismo tiempo y no hay forma de decidir exactamente cuál seguir siempre, luego, no es *LR(1)*.

- (b) Aún cuando existe un conflicto **shift-reduce** en el estado I_5 para el símbolo ; construiremos la **Tabla de Parsing** con conflictos, la cual quedaría de la siguiente forma:

Estado	Acciones			Go to	
	;	IS	\$	Instr'	Instr
I_0		avanzar(2)			1
I_1	avanzar(4)		avanzar(3)		
I_2	reducir(iii)				
I_3			aceptar		
I_4		avanzar(2)			5
I_5	avanzar(4) // reducir(ii)		reducir(ii)		

- (c) Para el reconocimiento de la frase *IS;IS;IS* priorizaremos al shift y después al reduce:

Priorizando el shift (avanzar):

Pila	Entrada	Acción
I_0	IS;IS;IS\$	avanzar(2)
$I_2 I_0$;IS;IS\$	reducir(iii)
$I_1 I_0$;IS;IS\$	avanzar(4)
$I_4 I_1 I_0$	IS;IS\$	avanzar(2)
$I_2 I_4 I_1 I_0$;IS\$	reducir(iii)
$I_5 I_4 I_1 I_0$;IS\$	avanzar(4)
$I_4 I_5 I_4 I_1 I_0$	IS\$	avanzar(2)
$I_2 I_4 I_5 I_4 I_1 I_0$	\$	reducir(iii)
$I_5 I_4 I_5 I_4 I_1 I_0$	\$	reducir(ii)
$I_5 I_4 I_1 I_0$	\$	reducir(ii)
$I_1 I_0$	\$	avanzar(3)
$I_3 I_1 I_0$	\$	aceptar

Priorizando el reduce (reducir):

Pila	Entrada	Acción
I_0	IS;IS;IS\$	avanzar(2)
$I_2 I_0$;IS;IS\$	reducir(iii)
$I_1 I_0$;IS;IS\$	avanzar(4)
$I_4 I_1 I_0$	IS;IS\$	avanzar(2)
$I_2 I_4 I_1 I_0$;IS\$	reducir(iii)
$I_5 I_4 I_1 I_0$;IS\$	reducir(ii)
$I_1 I_0$;IS\$	avanzar(4)
$I_4 I_1 I_0$	IS\$	avanzar(2)
$I_2 I_4 I_1 I_0$	\$	reducir(iii)
$I_5 I_4 I_1 I_0$	\$	reducir(ii)
$I_1 I_0$	\$	avanzar(3)
$I_3 I_1 I_0$	\$	aceptar

Como vemos en el primer caso se asocia a la izquierda y en la segunda a la derecha pero en verdad vemos que es indiferente pues existe una ambigüedad en el lenguaje al existir dos árboles sintácticos para la misma gramática.

- (d) La eficiencia de ambas alternativas se determinó a través del reconocimiento de las frases IS;IS;IS;IS, IS;IS;IS;IS;IS, adicional a la frase pedida en la pregunta 2, sección (c), esto con el fin de buscar patrones que ayuden a determinar cuál es más eficiente.

En cuanto al tamaño de la pila, el favorecer al **reduce** hace que la pila no sea tan grande (máximo 4 items para los casos probados) como sí es el caso de favorecer el **shift** (8 items y 10 items respectivamente).

En terminos de complejidad, para una frase $IS;(IS)^n$ vemos que favorecer al **reduce** o al **shift** es indiferente pues la cantidad de pasos necesarios para reconocer la misma palabra es igual en ambos casos pero en caso de favorecer el reduce se usa mucho menos espacio para la pila, a diferencia del caso del shift. Luego, con todo lo anterior podemos decir que la complejidad es $O(n)$, lineal, puesto que la cantidad de pasos necesarios para reconocer una palabra en el lenguaje viene dada por $4(n+1)$ y depende de n, que es la cantidad de ' IS' ' en la frase.