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 las gramáticas $G1_i$ y $G1_d$ dadas:

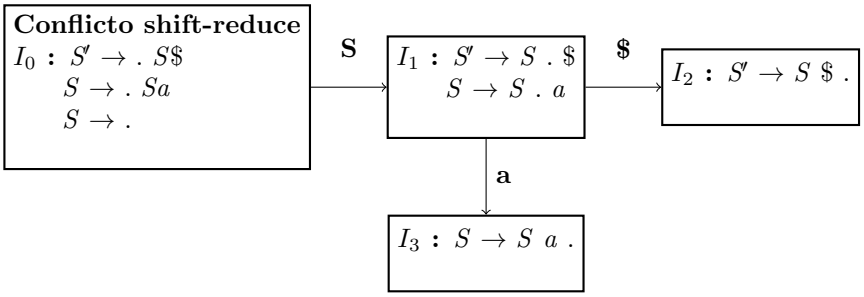
   (a) Para construir el analizador sintáctico *LR(0)* de $G1_i$, modificamos la gramática y queda:

   (i) $S' \to S\$$

   (ii) $S \to Sa$

   (iii) $\quad \mid \lambda$

   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)*.

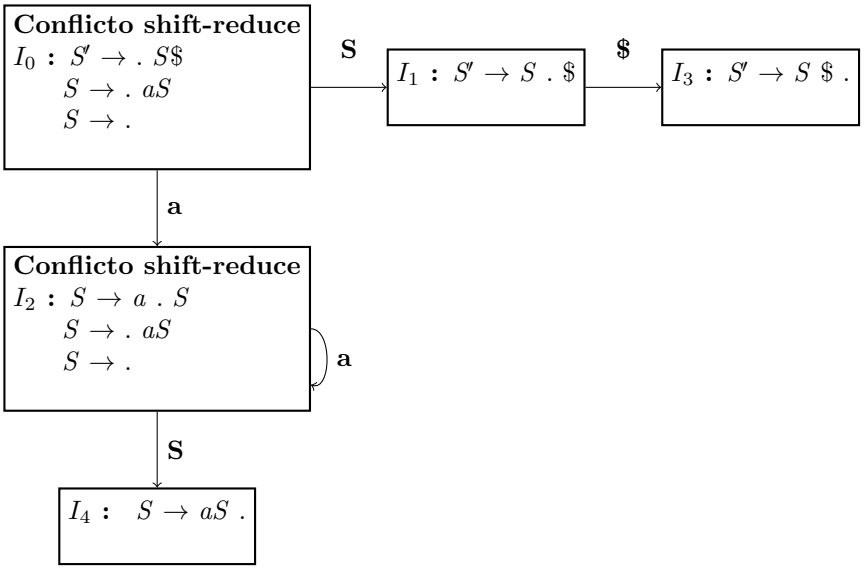
Para construir el analizador sintáctico *LR(0)* de $G1_d$, modificamos la gramática y queda:

(i) $S' \to S\$$

(ii) $S \to aS$

(iii) $\quad | \ \lambda$

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) En términos de espacio, para la eficiencia de los analizadores de ambas gramáticas, tenemos que $G1_i$ tiene sólo 4 estados en su tabla de parseo, mientras que $G1_d$ tiene 5 estados, así, la cantidad de pila y pasos utilizados para el reconocimiento de cualquier palabra perteneciente a *L(a*)* es menor en $G1_i$. En cuanto al tiempo que le tomaría a cada autómata de pila en reconocer una frase de *L(a*)* seguiría siendo menor para el autómata de $G1_i$.

Finalmente, la complejidad es $\mathbf{O(n)}$ para ambos puesto que en ambos casos la complejidad es lineal y depende del tamaño de la frase perteneciente al lenguaje.

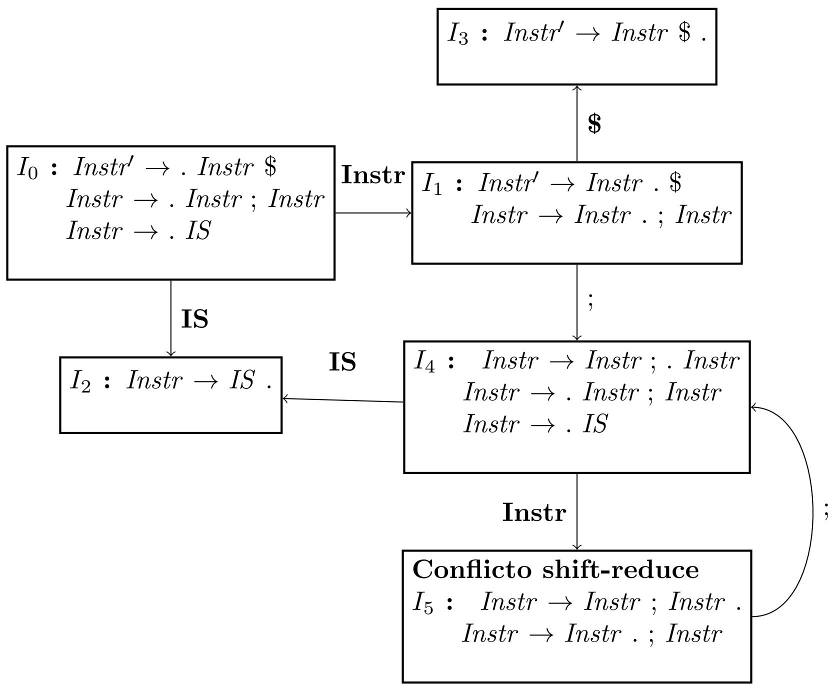
2. (a) Basados en la gramática dada, construimos un analizador sintáctico, empezando por transformar la gramática a:

(i) $Instr' \rightarrow Instr\$$

(ii) $Instr \rightarrow Instr;Instr$

(iii) $\quad\quad | \; IS$

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) En cuanto al tamaño de la pila, el favorecer al **reduce** hace que la pila no sea tan grande y comparta la cantidad de pasos con el caso de favorecer al **shift**. En terminos de complejidad, para una frase IS;$(IS)^n$ vemos que conviene el uso de la segunda alternativa ya que se empilarían menos estados a la pila y por ende se harían menos operaciones de empilar y desempilar, lo que llevaría a una complejidad del tipo **O(n)** al depender de la cantidad de $';IS'$.