Universidad de Concepción
Facultad de Ingeniería
Departamento de Ingeniería Informática y Ciencias de la Computación

Práctica 14 Solución

Teoría de Computación (503306)

Profesor: María Angélica Pinninghoff
Ayudante: Diego Palma

Para ver detalles sobre cómo se abordan este tipo de problemas, en la página 253 del pdf del libro *compilers* se encuentra la descripción del *Bottom-Up Parsing*. La solución propuesta en este solucionario está basada en lo visto en clases y diapositivas, por lo que se utilizará la misma notación (es un poco diferente a la del libro, pero equivalente).

El primer paso a realizar es normalizar la gramática (este concepto de *augmented grammar* se encuentra explicado en la página 266 del pdf), para ello agregamos una nueva regla relacionada al símbolo inicial, y reescribimos la gramática como sigue:

$$\begin{array}{ll} (0) & S' \to S \\ (1) & S \to AB \\ (2) & A \to aA \\ (3) & A \to x \\ (4) & B \to bB \\ (5) & B \to c \end{array}$$

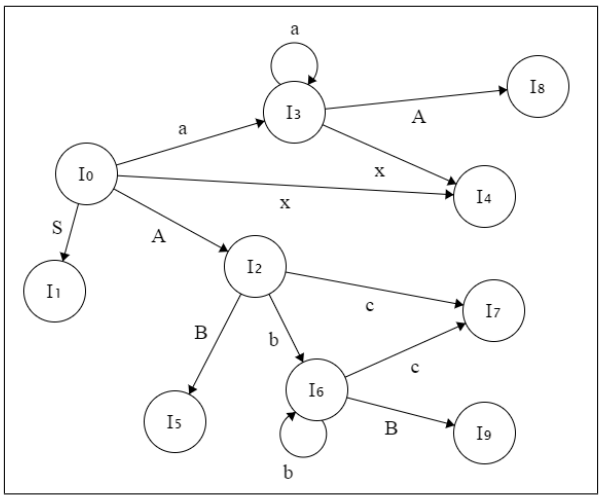
notar que enumeraremos las reglas, y esta enumeración es importante pues se utilizará en pasos posteriores.

Luego, hay que calcular el *closure* de cada *ítemset*. El concepto de *ítemset*, y el cálculo y definición de *closure* (y el algoritmo para obtenerlo) se encuentran explicados detalladamente en la página 266 del pdf. La razón de encontrar estos conjuntos, es que definen los estados de un automatón que toma acciones *shift*, *reduce*, *accept*, y *error* para el proceso de parseo (*shift reduce parsing*).

$$\begin{array}{lll} I_0\text{:} & \{S' \to S, \quad S \to \cdot AB, \quad A \to \cdot aA, \quad A \to \cdot x\} & \\ I_1\text{:} & \{S' \to S\cdot\} & \\ I_2\text{:} & \{S \to A \cdot B, \quad B \to \cdot bB, \quad B \to \cdot c\} & \\ I_3\text{:} & \{A \to a \cdot A, \quad A \to \cdot aA, \quad A \to \cdot x\} & \\ I_4\text{:} & \{A \to x\cdot\} & (\textit{reduce}) \\ I_5\text{:} & \{S \to AB\cdot\} & (\textit{reduce}) \\ I_6\text{:} & \{B \to b \cdot B, \quad B \to \cdot bB, \quad B \to \cdot c\} & \\ I_7\text{:} & \{B \to c\cdot\} & (\textit{reduce}) \\ I_8\text{:} & \{A \to aA\cdot\} & (\textit{reduce}) \\ I_9\text{:} & \{B \to bB\cdot\} & (\textit{reduce}) \end{array}$$

Se deben observar dos cosas, se comenzó por cada regla en orden para crear los conjuntos *closure*, y en orden dependiendo de los ítemes generados por cada regla, se fueron creando los siguientes estados. Es por ello que el orden de las reglas es importante. Como no existen conflictos *shift-reduce*, *shift-shift* ni *reduce-reduce*, se concluye que la gramática es $LR(0)$.

Si los conjuntos anteriores se representarán como un automatón, se obtendría lo siguiente:



El paso siguiente es construir la tabla *Goto-Action*, la cual queda como:

Estado	Action					GOTO		
	a	b	c	x	$	S	A	B
0	s_3			s_4		1	2	
1					OK			
2		s_6	s_7					5
3	s_3			s_4			8	
4	r_3	r_3	r_3	r_3	r_3			
5	r_1	r_1	r_1	r_1	r_1			
6		s_6	s_7					9
7	r_5	r_5	r_5	r_5	r_5			
8	r_2	r_2	r_2	r_2	r_2			
9	r_4	r_4	r_4	r_4	r_4			

Se debe notar que esta tabla básicamente representa las transiciones del automatón.

Para reconocer el string `aaxbc$`, el stack comienza con el símbolo s_0 representando que se encuentra en el estado inicial. Los pasos a seguir son los siguientes:

Stack	Entrada	Acción
s_0	aaxbc$	s_3
$s_0 s_3$	axbc$	s_3
$s_0 s_3 s_3$	xbc$	s_4
$s_0 s_3 s_3 s_4$	bc$	$r_3(A \to x)$
$s_0 s_3 s_3 A_8$	bc$	$r_2(A \to aA)$
$s_0 s_3 A_8$	bc$	$r_2(A \to aA)$
$s_0 A_2$	bc$	s_6
$s_0 A_2 s_6$	c$	s_7
$s_0 A_2 s_6 s_7$	$	$r_5(B \to c)$
$s_0 A_2 s_6 B_9$	$	$r_4(B \to bB)$
$s_0 A_2 B_5$	$	$r_1(S \to AB)$
$s_0 S_1$	$	OK
		Acepta

Se observa como paso a paso se hace *push* cuando se realiza la acción *shift*, y se realiza *pop* al stack en cada *reduce*. Finalmente el string es aceptado.