Universidad Nacional Autónoma de México

Facultad de Ciencias

Compiladores

Tarea 2:
Gramaticas

Autores:

Escamilla Soto Cristopher Alejandro
Montiel Manriquez Ricardo

31 de Marzo del 2022

1. Considera la siguiente gramática:

$$S \to a \qquad S \to \textbf{if } b \textbf{ then } S \qquad S \to \textbf{if } b \textbf{ then } S \textbf{ else } S$$

a) Proporciona una derivación, un árbol de sintaxis abstracta y un árbol de sintaxis concreta para la cadena **if** b **then if** b **then** a **else** a.

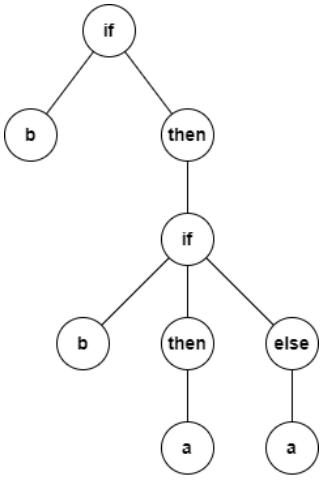
- Derivación:

  $S \to \textbf{if } b \textbf{ then } S$

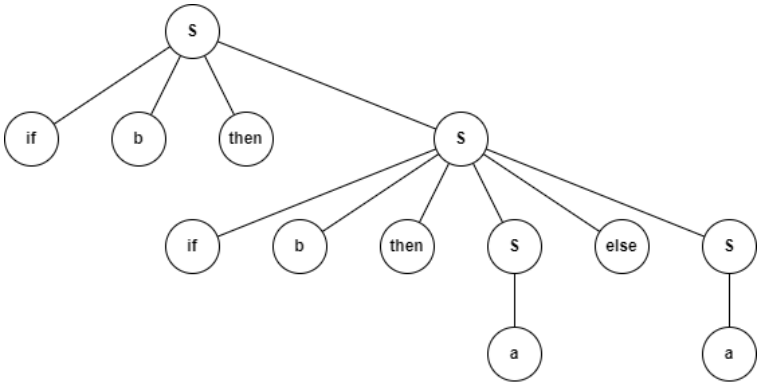
  $S \to \textbf{if } b \textbf{ then if } b \textbf{ then } S \textbf{ else } S$

  $S \to \textbf{if } b \textbf{ then if } b \textbf{ then } a \textbf{ else } a$

- Árbol de sintaxis abstracta:



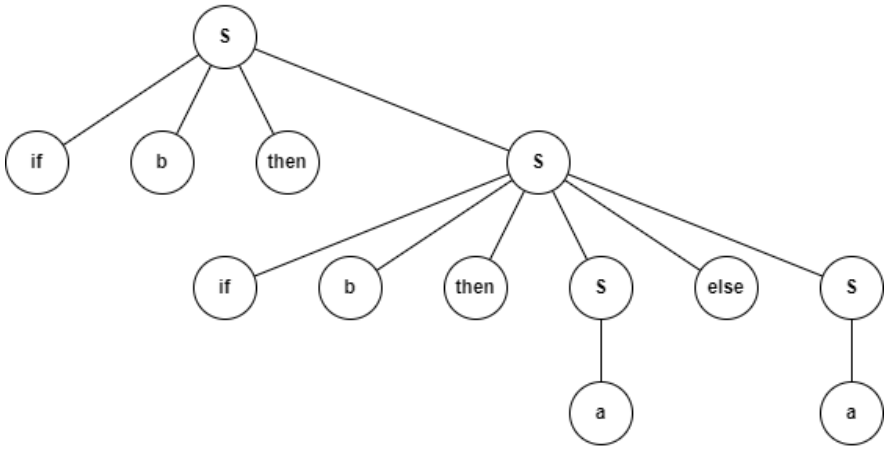
- Árbol de sintaxis concreta:



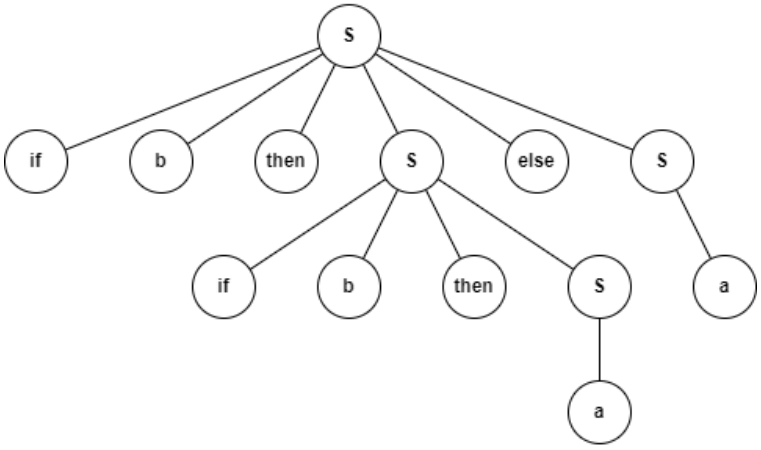
b) Demuestra que esta gramática es ambigua y describe una propuesta detallada para eliminar la ambigüedad.

La gramática es ambigua porque hay dos arboles de derivación para representar la cadena **if** b **then if** b **then** a **else** a.

Árbol 1



Árbol 2



Para eliminar la ambigüedad debemos de hacer que el **else** se empareje con el **if** anterior mas cercano.

La idea es que una sentencia que aparezca entre un **then** y un **else** debe estar emparejada, es decir no debe terminar con un **then** sin emparejar seguido de cualquier sentencia, porque entonces el **else** estaría obligado a concordar con este **then** no emparejado. Así la sentencia emparejada tan solo puede ser una sentencia alternativa **if-then-else** que

no contenga sentencias sin emparejar o cualquier otro tipo de sentencia diferente de la alternativa.

Con esto eliminaremos la ambigüedad y la gramática seria la siguiente:

$S \rightarrow A \mid B$

$A \rightarrow \textbf{if } b \textbf{ then } A \textbf{ else } A \mid a$

$B \rightarrow \textbf{if } b \textbf{ then } S \mid \textbf{if } b \textbf{ then } A \textbf{ else } B$

2. La siguiente gramática describe un lenguaje de consultas simple en donde STRING es un símbolo terminal:

$$\texttt{Session} \rightarrow \texttt{Fact Session}$$
$$\texttt{Session} \rightarrow \texttt{Question}$$
$$\texttt{Session} \rightarrow \texttt{(Session) Session}$$
$$\texttt{Fact} \rightarrow \texttt{!STRING}$$
$$\texttt{Question} \rightarrow \texttt{?STRING}$$

   a) Calcula las funciones First y Follow para los símbolos no-terminales de la gramática.

   - First:

     First (Session → Fact Session) = {!}

     First (Session → Question) = {?}

     First (Session → (Session) Session) = {(}

     $\Longrightarrow$

     First (Session) = {!, ?, (}

     First (Fact) = {!}

     First (Question) = {?}

   - Follow:

     Follow (Session) = {#,)}

     Follow (Fact) = First (Session) = {(, !, ?}

     Follow (Question) = {#,)}

   b) Genera una tabla de parsing predictivo de tipo **LL(1)**.

	String	?	!	(	)	#
Session		Session → Question	Session → Fact Session	Session → (Session) Session		
Fact			Fact → !STRING			
Question		Question → ?STRING				

3. Considera la siguiente gramática:

   $stmt \rightarrow$ **if** $bexpr$ **then** $stmt$ **else** $stmt$ | **while** $bexpr$ **do** $stmt$ | **begin** $stmts$ **end**

   $stmts \rightarrow stmt$; $stmts \mid \varepsilon$

   $bexpr \rightarrow \mathbf{id}_i$

   donde las expresiones booleanas están representadas por terminales de la forma $\mathbf{id}_i$.

   a) Calcula las funciones First y Follow para los símbolos no-terminales de la gramática.

   - First:

     First (stmt) = {if, while, begin}

     First (stmts) = {if, while, begin, ε}

     First (bexpr) = {id_i}

   - Follow:

     Follow (stmt) = {else, ;, #}

     Follow (stmts) = {end, #}

     Follow (bexpr) = {then, do}

   b) Calcula la tabla de parsing predictivo.

	if	then	else	while	do	begin	end	id_i	#
stmt	stmt → if...			stmt → while...		stmt → begin...			
stmts	stmts → stmt...			stmts → stmt...		stmts → stmt...	stmts → ε		stmts → ε
bexpr								bexpr → id_i	

   c) Da una propuesta para manejar los errores en esta gramática.

   Cuando el match ya no pueda hacerse con el elemento top de la pila, podemos rechazar la cadena de entrada y notificar mediante un error.

   Considera la siguiente gramática, alternativa a la anterior:

   $stmt \rightarrow$ **if** $expr$ **then** $stmt$ $stmtTail$ | **while** $expr$ **do** $stmt$ | **begin** $list$ **end** | s

$stmtTail \rightarrow$ **else** $stmt \mid \varepsilon$

$list \rightarrow stmt\ listTail$

$listTail \rightarrow ;\ list \mid \varepsilon$

donde expr y s se consideran terminales para las guardias booleanas y otros enunciados.

d) Explica las diferencias entre ambas gramáticas.

A diferencia de la primera gramatica esta no es libre de contexto, ya que podemos tener 2 derivaciones de la misma cadena:

- Primera derivación:

  $stmt \rightarrow$ **if** $expr$ **then** $stmt\ stmtTail$

  $stmt \rightarrow$ **if** $expr$ **then if** $expr$ **then** $stmt\ stmtTail\ stmtTail$

  $stmt \rightarrow$ **if** $expr$ **then if** $expr$ **then** $stmt\ else\ stmt\ stmtTail$

  $stmt \rightarrow$ **if** $expr$ **then if** $expr$ **then** $stmt\ else\ stmt\ \varepsilon$

  $stmt \rightarrow$ **if** $expr$ **then if** $expr$ **then** $stmt\ else\ stmt$

- Segunda derivación:

  $stmt \rightarrow$ **if** $expr$ **then** $stmt\ stmtTail$

  $stmt \rightarrow$ **if** $expr$ **then if** $expr$ **then** $stmt\ stmtTail\ stmtTail$

  $stmt \rightarrow$ **if** $expr$ **then if** $expr$ **then** $stmt\ \varepsilon\ stmtTail$

  $stmt \rightarrow$ **if** $expr$ **then if** $expr$ **then** $stmt\ stmtTail$

  $stmt \rightarrow$ **if** $expr$ **then if** $expr$ **then** $stmt\ else\ stmt$

e) Muestra la tabla de parsing predictivo.

f) Usa la idea de sincronización de símbolos para resolver errores.

g) Muestra el procesamiento de la cadena **while** e **do begin** s **; if** e **then** s **; end** con la segunda gramática.