Autómatas y Lenguajes Formales

Tema 8: Ambigüedad en gramáticas libres de contexto

Dr. Favio Ezequiel Miranda Perea
favio@ciencias.unam.mx

Facultad de Ciencias UNAM[1]

27 de abril de 2019

[1]Con el apoyo del proyecto PAPIME PE102117

Árboles de derivación

GLC

- Los árboles de derivación o árboles sintácticos son un mecanismo para representar las derivaciones de gramáticas libres de contexto.

Árboles de derivación

GLC

- Los árboles de derivación o árboles sintácticos son un mecanismo para representar las derivaciones de gramáticas libres de contexto.
- En compiladores se utilizan para el análisis sintáctico de programas fuente (parsing) y sirven de base para la generación de código.

Árboles de derivación

GLC

- Los árboles de derivación o árboles sintácticos son un mecanismo para representar las derivaciones de gramáticas libres de contexto.
- En compiladores se utilizan para el análisis sintáctico de programas fuente (parsing) y sirven de base para la generación de código.
- Puede ser que dos derivaciones distintas tengan el mismo árbol.

Construcción de árboles de derivación

GLC

Dada una gramática libre de contexto $G = \langle V, T, S, P \rangle$, un árbol de derivación en G se construye como sigue:

Construcción de árboles de derivación

GLC

Dada una gramática libre de contexto $G = \langle V, T, S, P \rangle$, un árbol de derivación en G se construye como sigue:

- La raiz contiene al símbolo inicial S.

Construcción de árboles de derivación

GLC

Dada una gramática libre de contexto $G = \langle V, T, S, P \rangle$, un árbol de derivación en G se construye como sigue:

- La raiz contiene al símbolo inicial S.
- Cada nodo interior contiene una variable

Construcción de árboles de derivación

GLC

Dada una gramática libre de contexto $G = \langle V, T, S, P \rangle$, un árbol de derivación en G se construye como sigue:

- La raiz contiene al símbolo inicial S.
- Cada nodo interior contiene una variable
- Cada hoja contiene un símbolo de $V \cup T \cup \{\varepsilon\}$.

Construcción de árboles de derivación

GLC

Dada una gramática libre de contexto $G = \langle V, T, S, P \rangle$, un árbol de derivación en G se construye como sigue:

- La raiz contiene al símbolo inicial S.
- Cada nodo interior contiene una variable
- Cada hoja contiene un símbolo de $V \cup T \cup \{\varepsilon\}$.
- Si un nodo interior contiene una variable A entonces sus hijos contienen símbolos (de izquierda a derecha) $a_1, \ldots, a_n$ si y sólo si $A \to a_1 a_2 \ldots a_n$ está en P.

Construcción de árboles de derivación

GLC

Dada una gramática libre de contexto $G = \langle V, T, S, P \rangle$, un árbol de derivación en G se construye como sigue:

- La raiz contiene al símbolo inicial S.
- Cada nodo interior contiene una variable
- Cada hoja contiene un símbolo de $V \cup T \cup \{\varepsilon\}$.
- Si un nodo interior contiene una variable A entonces sus hijos contienen símbolos (de izquierda a derecha) $a_1, \ldots, a_n$ si y sólo si $A \to a_1 a_2 \ldots a_n$ está en P.
- La palabra generada se puede leer al leer las hojas de izquierda a derecha.

Árboles de derivación

GLC

- Puede haber mas de un árbol de derivación para una cadena.

Árboles de derivación

GLC

- Puede haber mas de un árbol de derivación para una cadena.
- Lo ideal es que cada cadena tenga sólo un árbol asociado, esto implica que el lenguaje no es ambiguo.

Árboles de derivación

GLC

- Puede haber mas de un árbol de derivación para una cadena.
- Lo ideal es que cada cadena tenga sólo un árbol asociado, esto implica que el lenguaje no es ambiguo.
- Desafortunadamente existen lenguajes ambiguos.

Ambigüedad

GLC

- Una gramática se dice ambigua si existe una palabra w con dos o más árboles de derivación distintos.

Ambigüedad

GLC

- Una gramática se dice ambigua si existe una palabra w con dos o más árboles de derivación distintos.
- En general una palabra puede tener mas de una derivación, pero un sólo árbol, en tal caso no hay ambigüedad.

Ambigüedad

GLC

- Una gramática se dice ambigua si existe una palabra w con dos o más árboles de derivación distintos.
- En general una palabra puede tener mas de una derivación, pero un sólo árbol, en tal caso no hay ambigüedad.
- Algunas veces se puede suprimir la ambigüedad directamente.

Ambigüedad

GLC

- Una gramática se dice ambigua si existe una palabra w con dos o más árboles de derivación distintos.
- En general una palabra puede tener mas de una derivación, pero un sólo árbol, en tal caso no hay ambigüedad.
- Algunas veces se puede suprimir la ambigüedad directamente.
- Sin embargo no hay un algoritmo para remover ambigüedad.

Ambigüedad

GLC

- Una gramática se dice ambigua si existe una palabra w con dos o más árboles de derivación distintos.
- En general una palabra puede tener mas de una derivación, pero un sólo árbol, en tal caso no hay ambigüedad.
- Algunas veces se puede suprimir la ambigüedad directamente.
- Sin embargo no hay un algoritmo para remover ambigüedad.
- Pero aún, hay lenguajes cuya ambigüedad es inevitable.

Ejemplos

Ambigüedad

$$S \to AA \quad A \to aSa \mid a$$

La palabra a^5 tiene las siguientes derivaciones:

Ejemplos

Ambigüedad

$$S \to AA \quad A \to aSa \mid a$$

La palabra a^5 tiene las siguientes derivaciones:

- $S \to AA \to aA \to aaSa \to aaAAa \to aaaAa \to aaaaa$

Ejemplos

Ambigüedad

$$S \to AA \quad A \to aSa \mid a$$

La palabra a^5 tiene las siguientes derivaciones:

- $S \to AA \to aA \to aaSa \to aaAAa \to aaaAa \to aaaaa$
- $S \to AA \to aSaA \to aAAaA \to aaAaA \to aaaaA \to aaaaa$

Ejemplos

Ambigüedad

$$S \to AA \quad A \to aSa \mid a$$

La palabra a^5 tiene las siguientes derivaciones:

- $S \to AA \to aA \to aaSa \to aaAAa \to aaaAa \to aaaaa$
- $S \to AA \to aSaA \to aAAaA \to aaAaA \to aaaaA \to aaaaa$
- Las dos derivaciones son por la izquierda y generan árboles distintos.

Lenguajes Ambiguos

Ambigüedad

- Un lenguaje L es ambiguo si existe una gramática ambigua G que genera a L.

Lenguajes Ambiguos

Ambigüedad

- Un lenguaje L es ambiguo si existe una gramática ambigua G que genera a L.
- $L = \{a^{2+3i} \mid i \geq 0\}$ es ambiguo.

Lenguajes Ambiguos

Ambigüedad

- Un lenguaje L es ambiguo si existe una gramática ambigua G que genera a L.
- $L = \{a^{2+3i} \mid i \geq 0\}$ es ambiguo.

- Un lenguaje es inherentemente ambiguo si todas las gramáticas que lo generan son ambiguas.

Lenguajes Ambiguos

Ambigüedad

- Un lenguaje L es ambiguo si existe una gramática ambigua G que genera a L.
- $L = \{a^{2+3i} \mid i \geq 0\}$ es ambiguo.

- Un lenguaje es inherentemente ambiguo si todas las gramáticas que lo generan son ambiguas.
- $L = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n, m \geq 1\}$ es inherentemente ambiguo.

$L = \{a^{2+3i} \mid i \geq 0\}$

Lenguajes ambiguos

L es ambiguo por se generado por la gramática ambigua

$$S \to AA \quad A \to aSa \mid a$$

$L = \{a^{2+3i} \mid i \geq 0\}$

Lenguajes ambiguos

L es ambiguo por se generado por la gramática ambigua

$$S \to AA \quad A \to aSa \mid a$$

Sin embargo este lenguaje también es generado por una gramática no ambigua:

$$S \to aa \mid aaU \qquad U \to aaaU \mid aaa$$

$L = \{a^{2+3i} \mid i \geq 0\}$

Lenguajes ambiguos

L es ambiguo por se generado por la gramática ambigua

$$S \to AA \quad A \to aSa \mid a$$

Sin embargo este lenguaje también es generado por una gramática no ambigua:

$$S \to aa \mid aaU \qquad U \to aaaU \mid aaa$$

en este caso la derivación de a^5 es:

$$S \to aaU \to aaaaa$$

$L = \{a^{2+3i} \mid i \geq 0\}$

Lenguajes ambiguos

L es ambiguo por se generado por la gramática ambigua

$$S \to AA \quad A \to aSa \mid a$$

Sin embargo este lenguaje también es generado por una gramática no ambigua:

$$S \to aa \mid aaU \qquad U \to aaaU \mid aaa$$

en este caso la derivación de a^5 es:

$$S \to aaU \to aaaaa$$

por lo tanto L no es un lenguaje inherentemente ambiguo

$L = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n, m \geq 1\}$

Lenguajes inherentemente ambigüos

L es generado por la gramática:

$$S \to AB \mid C \quad A \to aAb \mid ab \quad B \to cBd \mid cd$$
$$C \to aCd \mid aDd \quad D \to bDc \mid bc$$

$L = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n, m \geq 1\}$

Lenguajes inherentemente ambigüos

L es generado por la gramática:

$$S \to AB \mid C \quad A \to aAb \mid ab \quad B \to cBd \mid cd$$
$$C \to aCd \mid aDd \quad D \to bDc \mid bc$$

La cadena *aabbccdd* tiene dos derivaciones por la izquierda:

$L = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n, m \geq 1\}$

Lenguajes inherentemente ambigüos

L es generado por la gramática:

$$S \to AB \mid C \quad A \to aAb \mid ab \quad B \to cBd \mid cd$$
$$C \to aCd \mid aDd \quad D \to bDc \mid bc$$

La cadena *aabbccdd* tiene dos derivaciones por la izquierda:

$S \to AB \to aAbB \to aabbB \to aabbcBd \to aabbccdd$

$L = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n, m \geq 1\}$

Lenguajes inherentemente ambigüos

L es generado por la gramática:

$$S \to AB \mid C \quad A \to aAb \mid ab \quad B \to cBd \mid cd$$
$$C \to aCd \mid aDd \quad D \to bDc \mid bc$$

La cadena *aabbccdd* tiene dos derivaciones por la izquierda:

$S \to AB \to aAbB \to aabbB \to aabbcBd \to aabbccdd$

$S \to C \to aCd \to aaDdd \to aabDcdd \to aabbccdd$

$L = \{a^n b^n c^m d^m \mid n, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n, m \geq 1\}$

Lenguajes inherentemente ambigüos

L es generado por la gramática:

$$S \to AB \mid C \quad A \to aAb \mid ab \quad B \to cBd \mid cd$$
$$C \to aCd \mid aDd \quad D \to bDc \mid bc$$

La cadena $aabbccdd$ tiene dos derivaciones por la izquierda:

$S \to AB \to aAbB \to aabbB \to aabbcBd \to aabbccdd$

$S \to C \to aCd \to aaDdd \to aabDcdd \to aabbccdd$

Probar la ambiguedad inherente es complicado.

Ambigüedad

- Está probado que no puede existir un algoritmo que determine con certeza si una gramática es ambigua o no, y que en tal caso elimine dicha ambigüedad produciendo una gramática no ambigua equivalente a la original.

Ambigüedad

- Está probado que no puede existir un algoritmo que determine con certeza si una gramática es ambigua o no, y que en tal caso elimine dicha ambigüedad produciendo una gramática no ambigua equivalente a la original.
- Es decir, el problema de ambigüedad es indecidible. Lo más que se sabe es que hay ciertas condiciones que determinan ambigüedad pero en caso de no cumplirse éstas nada puede decirse de la gramática en cuestión.

Ambigüedad

- Está probado que no puede existir un algoritmo que determine con certeza si una gramática es ambigua o no, y que en tal caso elimine dicha ambigüedad produciendo una gramática no ambigua equivalente a la original.
- Es decir, el problema de ambigüedad es indecidible. Lo más que se sabe es que hay ciertas condiciones que determinan ambigüedad pero en caso de no cumplirse éstas nada puede decirse de la gramática en cuestión.
- En algunos casos, dada una gramática ambigua, se puede encontrar otra gramática equivalente no ambigua, por ejemplo agregando precedencia de operadores y asociatividad.

Ambigüedad

- Está probado que no puede existir un algoritmo que determine con certeza si una gramática es ambigua o no, y que en tal caso elimine dicha ambigüedad produciendo una gramática no ambigua equivalente a la original.
- Es decir, el problema de ambigüedad es indecidible. Lo más que se sabe es que hay ciertas condiciones que determinan ambigüedad pero en caso de no cumplirse éstas nada puede decirse de la gramática en cuestión.
- En algunos casos, dada una gramática ambigua, se puede encontrar otra gramática equivalente no ambigua, por ejemplo agregando precedencia de operadores y asociatividad.
- Sin embargo existen lenguajes cuya ambigüedad es inevitable.

Paréntesis balanceados

Eliminación de la ambigüedad

- Paréntesis balanceados:

$$S \to \varepsilon \mid (S) \mid SS$$

Paréntesis balanceados

Eliminación de la ambigüedad

- Paréntesis balanceados:

$$S \to \varepsilon \mid (S) \mid SS$$

- Ambigüedad: ()()() tiene dos árboles de derivación

Paréntesis balanceados

Eliminación de la ambigüedad

- Paréntesis balanceados:

$$S \to \varepsilon \mid (S) \mid SS$$

- Ambigüedad: ()()() tiene dos árboles de derivación
- Gramática no ambigua equivalente:

$$S \to \varepsilon \mid (S)S$$

Expresiones aritméticas

Eliminación de la ambigüedad

- Expresiones aritméticas:

$$S \to S + S \mid S * S \mid (S) \mid a$$

donde a es un terminal que representa a los identificadores y constantes.

Expresiones aritméticas

Eliminación de la ambigüedad

- Expresiones aritméticas:

$$S \to S + S \mid S * S \mid (S) \mid a$$

donde *a* es un terminal que representa a los identificadores y constantes.

- Ambigüedad: a + a * a

Expresiones aritméticas

Eliminación de la ambigüedad

- Expresiones aritméticas:

$$S \to S + S \mid S * S \mid (S) \mid a$$

donde a es un terminal que representa a los identificadores y constantes.

- Ambigüedad: a + a * a
- Gramática no ambigua equivalente: se obtiene modelando la precedencia de operadores como sigue:

$$\begin{aligned} S &\to S + T \mid T \\ T &\to T * F \mid F \\ F &\to (S) \mid a \end{aligned}$$

Expresiones condicionales

Eliminación de la ambigüedad

- Expresiones condicionales:

$$\begin{aligned} S &\rightarrow C \mid O \\ C &\rightarrow \texttt{if}\ E\ \texttt{then}\ S \mid \texttt{if}\ E\ \texttt{then}\ S\ \texttt{else}\ S \end{aligned}$$

- Problema del `if` colgante (dangling else, ALGOL 60):

```
if a then if b then s1 else s2
```

- Dos significados:

```
if a then (if b then s1) else s2

if a then (if b then s1 else s2)
```

Expresiones condicionales

Eliminación de la ambigüedad

- Una gramática equivalente no ambigua es:

$$
\begin{array}{rcl}
S & \rightarrow & C \mid O \\
C & \rightarrow & C1 \mid C2 \\
C1 & \rightarrow & \texttt{if}\ E\ \texttt{then}\ C1\ \texttt{else}\ C1 \\
C2 & \rightarrow & \texttt{if}\ E\ \texttt{then}\ S \mid \texttt{if}\ E\ \texttt{then}\ C1\ \texttt{else}\ C2
\end{array}
$$

- Idea: $C1$ genera condicionales dobles (if-then-else) balanceados; $C2$ representa condicionales simples (if-then) y condicionales dobles pero de forma que un if-then sólo figura colgando al final (en el else).

Gramáticas libres de contexto en lenguajes de programación

- El estudio formal de los lenguajes de programación se divide en sintaxis, pragmática y semántica.

Gramáticas libres de contexto en lenguajes de programación

- El estudio formal de los lenguajes de programación se divide en sintaxis, pragmática y semántica.
- La semántica se encarga de definir el significado de las expresiones, enunciados y unidades de programa.

Gramáticas libres de contexto en lenguajes de programación

- El estudio formal de los lenguajes de programación se divide en sintaxis, pragmática y semántica.
- La semántica se encarga de definir el significado de las expresiones, enunciados y unidades de programa.
- La pragmática define la implementación del lenguaje basada en la metodología y estrategias de programación deseadas

Gramáticas libres de contexto en lenguajes de programación

- El estudio formal de los lenguajes de programación se divide en sintaxis, pragmática y semántica.
- La semántica se encarga de definir el significado de las expresiones, enunciados y unidades de programa.
- La pragmática define la implementación del lenguaje basada en la metodología y estrategias de programación deseadas
- La sintaxis se encarga de definir la forma de las expresiones y enunciados de un lenguaje y se sirve fundamentalmente de los conceptos y herramientas de nuestro curso.

Gramáticas libres de contexto en lenguajes de programación

- El estudio formal de los lenguajes de programación se divide en sintaxis, pragmática y semántica.
- La semántica se encarga de definir el significado de las expresiones, enunciados y unidades de programa.
- La pragmática define la implementación del lenguaje basada en la metodología y estrategias de programación deseadas
- La sintaxis se encarga de definir la forma de las expresiones y enunciados de un lenguaje y se sirve fundamentalmente de los conceptos y herramientas de nuestro curso.
- Antes del proceso de evaluación, un compilador e intérprete necesita realizar los procesos de análisis léxico y sintáctico, describimos a continuación a grandes rasgos.

Análisis léxico y sintáctico

- Análisis léxico: se encarga de transformar el programa fuente en una lista de unidades sintácticas de bajo nivel llamadas lexemas, los cuales se clasifican en distintas categorías llamadas *tokens*, como pueden ser identificadores, constantes, separadores, etc.

Análisis léxico y sintáctico

- Análisis léxico: se encarga de transformar el programa fuente en una lista de unidades sintácticas de bajo nivel llamadas lexemas, los cuales se clasifican en distintas categorías llamadas *tokens*, como pueden ser identificadores, constantes, separadores, etc.
- El análisis léxico se sirve fundamentalmente de expresiones regulares para su definición y reconocimiento.

Análisis léxico y sintáctico

- Análisis léxico: se encarga de transformar el programa fuente en una lista de unidades sintácticas de bajo nivel llamadas lexemas, los cuales se clasifican en distintas categorías llamadas *tokens*, como pueden ser identificadores, constantes, separadores, etc.
- El análisis léxico se sirve fundamentalmente de expresiones regulares para su definición y reconocimiento.
- Análisis sintáctico: se encarga de transformar la lista de lexemas en un programa objeto, el cual es una expresión válida de la llamada sintaxis abstracta del lenguaje. Este programa es esencialmente un árbol de derivación dictado por una gramática libre de contexto que define al lenguaje de programación.

Análisis léxico y sintáctico

- Análisis léxico: se encarga de transformar el programa fuente en una lista de unidades sintácticas de bajo nivel llamadas lexemas, los cuales se clasifican en distintas categorías llamadas *tokens*, como pueden ser identificadores, constantes, separadores, etc.
- El análisis léxico se sirve fundamentalmente de expresiones regulares para su definición y reconocimiento.
- Análisis sintáctico: se encarga de transformar la lista de lexemas en un programa objeto, el cual es una expresión válida de la llamada sintaxis abstracta del lenguaje. Este programa es esencialmente un árbol de derivación dictado por una gramática libre de contexto que define al lenguaje de programación.
- Por lo tanto el análisis sintáctico es esencialmente una forma del problema de la pertenencia en gramáticas libres de contexto.

Forma de Backus-Naur

- Las gramáticas libres de contexto para lenguajes de programación suelen escribirse en la forma de Backus-Naur o BNF.

Forma de Backus-Naur

- Las gramáticas libres de contexto para lenguajes de programación suelen escribirse en la forma de Backus-Naur o BNF.
- Este método de definición de gramáticas fue introducido por John Backus para el lenguaje ALGOL 58 en 1959 y fue mejorado por Peter Naur para la definición de ALGOL 60.

Forma de Backus-Naur

- Las gramáticas libres de contexto para lenguajes de programación suelen escribirse en la forma de Backus-Naur o BNF.
- Este método de definición de gramáticas fue introducido por John Backus para el lenguaje ALGOL 58 en 1959 y fue mejorado por Peter Naur para la definición de ALGOL 60.
- Este sistema notacional para definir lenguajes libres de contexto sigue las siguientes convenciones:

Forma de Backus-Naur

- Las gramáticas libres de contexto para lenguajes de programación suelen escribirse en la forma de Backus-Naur o BNF.
- Este método de definición de gramáticas fue introducido por John Backus para el lenguaje ALGOL 58 en 1959 y fue mejorado por Peter Naur para la definición de ALGOL 60.
- Este sistema notacional para definir lenguajes libres de contexto sigue las siguientes convenciones:
  - El símbolo de reescritura $\rightarrow$ se reemplaza con $::=$.

Forma de Backus-Naur

- Las gramáticas libres de contexto para lenguajes de programación suelen escribirse en la forma de Backus-Naur o BNF.
- Este método de definición de gramáticas fue introducido por John Backus para el lenguaje ALGOL 58 en 1959 y fue mejorado por Peter Naur para la definición de ALGOL 60.
- Este sistema notacional para definir lenguajes libres de contexto sigue las siguientes convenciones:
  - El símbolo de reescritura $\rightarrow$ se reemplaza con $::=$.
  - El símbolo $|$ significa *o* y abreviar la definición de producciones de una misma variable.

Forma de Backus-Naur

- Las gramáticas libres de contexto para lenguajes de programación suelen escribirse en la forma de Backus-Naur o BNF.
- Este método de definición de gramáticas fue introducido por John Backus para el lenguaje ALGOL 58 en 1959 y fue mejorado por Peter Naur para la definición de ALGOL 60.
- Este sistema notacional para definir lenguajes libres de contexto sigue las siguientes convenciones:
  - El símbolo de reescritura $\rightarrow$ se reemplaza con $::=$.
  - El símbolo $|$ significa *o* y abreviar la definición de producciones de una misma variable.
  - Las variables se escriben entre paréntesis triangulares y por lo general utilizan nombres largos que ayuden a la descripción de las categorías del lenguaje.

Gramáticas en forma de Backus-Naur

Ejemplos

- Lenguaje de paréntesis balanceados

$$< parent\_balanc > \quad ::= \quad \varepsilon \mid (< parent\_balanc >) \mid < parent\_balanc >< parent\_balanc >$$

Gramáticas en forma de Backus-Naur

Ejemplos

- Lenguaje de paréntesis balanceados

$$\begin{aligned} < parent\_balanc > \quad ::= \quad & \varepsilon \mid \\ & (< parent\_balanc >) \mid \\ & < parent\_balanc >< parent\_balanc > \end{aligned}$$

- Expresiones aritméticas:

$$\begin{aligned} < expr > \quad ::= \quad & < expr >< op >< expr > \mid \\ & (< expr >) \mid < id > \\ < op > \quad := \quad & + \mid - \mid * \mid / \\ < id > \quad := \quad & a \mid b \mid c \end{aligned}$$

Bloques de asignación de expresiones aritméticas:

`begin` a := b/c ; b := a*(b+c) `end`

$$
\begin{aligned}
< \textit{programa} > &::= \texttt{begin} < \textit{sec\_enunc} > \texttt{end} \\
< \textit{sec\_enunc} > &::= < \textit{enunc} > \mid < \textit{enunc} > ; < \textit{sec\_enunc} > \\
< \textit{enunc} > &::= < \textit{id} > := < \textit{expr} > \\
< \textit{expr} > &::= < \textit{expr} > < \textit{op} > < \textit{expr} > \mid \\
&\quad (< \textit{expr} >) \mid < \textit{id} > \\
< \textit{op} > &::= + \mid - \mid * \mid / \\
< \textit{id} > &::= a \mid b \mid c
\end{aligned}
$$

Gramáticas en forma de Backus-Naur

Ejemplos

- Expresiones condicionales:

$$
\begin{aligned}
\langle \textit{enunc} \rangle &::= \langle \textit{condicional} \rangle \mid \langle \textit{otras} \rangle \\
\langle \textit{condicional} \rangle &:= \texttt{if} \ \langle \textit{expr} \rangle \ \texttt{then} \ \langle \textit{enunc} \rangle \mid \\
&\quad\ \texttt{if} \ \langle \textit{expr} \rangle \ \texttt{then} \ \langle \textit{enunc} \rangle \\
&\quad\ \texttt{else} \ \langle \textit{enunc} \rangle
\end{aligned}
$$