Autómatas y Lenguajes Formales

Tema 10: Ambigüedad y gramáticas libres de contexto en lenguajes de programación

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

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.

Gramatícas libres de contexto en lenguajes de programación

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

Gramatícas 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.

Gramatícas 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

Gramatícas 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.

Gramatícas 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:

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

- Problema del `if` colgante:

```
if false then if false then 0 else 1
```

- Dos significados:

```
if false then (if false then 0) else 1
if false then (if false then 0 else 1)
```

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:

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

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

Expresiones aritméticas

Eliminación de la ambigüedad

- Expresiones aritméticas:

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

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

- Ambigüedad: a + a * a

Expresiones aritméticas

Eliminación de la ambigüedad

- Expresiones aritméticas:

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

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:

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

Expresiones condicionales

Eliminación de la ambigüedad

- Expresiones condicionales:

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

- Una gramática equivalente no ambigua es:

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

Expresiones condicionales

Eliminación de la ambigüedad

- Expresiones condicionales:

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

- 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).

Ejercicio presencial

Muestre que la siguiente gramática es ambigua:

$$S \rightarrow AS \mid \varepsilon$$

$$A \rightarrow A1 \mid 0A1 \mid \varepsilon$$

Ejercicio del tema anterior

Transformar la siguiente gramática a la forma normal de Chomsky:

$$
\begin{aligned}
S &\rightarrow 0S1 \mid A \mid AB \\
A &\rightarrow 1A0 \mid S \mid \varepsilon \\
B &\rightarrow 0B \mid 1C \\
C &\rightarrow 0C \mid 0 \mid \varepsilon \\
D &\rightarrow 0C \mid 1D \mid F \\
F &\rightarrow 1F \mid \varepsilon
\end{aligned}
$$