Compiladores 24-2
Análisis Sintáctico: funciones auxiliares

Lourdes del Carmen González Huesca
luglzhuesca@ciencias.unam.mx

Facultad de Ciencias, UNAM

26 febrero 2024

Análisis sintáctico

Top-down parsing

- Top-down parsing para gramáticas tipo **LL**
  (*scan input from left to right & left-most derivation*)

Análisis sintáctico

Top-down parsing

- Top-down parsing para gramáticas tipo **LL**
  (*scan input from left to right & left-most derivation*)
- Un parser **LL(k)** es también llamado predictivo, en donde se revisan *k* tokens por adelantado
  para una variable o un símbolo no-terminal, el símbolo leído por adelantado determina de forma única la producción a aplicar

Análisis sintáctico

Top-down parsing

- Top-down parsing para gramáticas tipo **LL**
  (*scan input from left to right & left-most derivation*)
- Un parser **LL(k)** es también llamado predictivo, en donde se revisan *k* tokens por adelantado
  para una variable o un símbolo no-terminal, el símbolo leído por adelantado determina de forma única la producción a aplicar
- Un parser es un autómata de pila definido por la tabla de transiciones que se denomina **tabla de parsing**.
  toma una variable y símbolo para devolver una producción o error

Análisis sintáctico

Top-down parsing

- Top-down parsing para gramáticas tipo **LL**
  (*scan input from left to right & left-most derivation*)
- Un parser **LL(k)** es también llamado predictivo, en donde se revisan *k* tokens por adelantado
  para una variable o un símbolo no-terminal, el símbolo leído por adelantado determina de forma única la producción a aplicar
- Un parser es un autómata de pila definido por la tabla de transiciones que se denomina **tabla de parsing**.
  toma una variable y símbolo para devolver una producción o error

determinar los prefijos y sufijos de las variables para predecir producciones

Definición (Función FIRST)

Esta función calcula el conjunto de símbolos terminales que están al inicio de las palabras derivadas de una cadena:

$$\text{FIRST}(\alpha) = \{a \in \Sigma \mid \exists w, \alpha \rightarrow^* aw\}$$

Definición (Función FIRST)

Esta función calcula el conjunto de símbolos terminales que están al inicio de las palabras derivadas de una cadena:

$$\text{FIRST}(\alpha) = \{a \in \Sigma \mid \exists w, \alpha \to^* aw\}$$

Recursivamente, para un símbolo se tiene:

- Si $X \in \Sigma$ entonces $\text{FIRST}(X) = \{X\}$.
- Si X es no-terminal y $X \to Y_1 Y_2 \dots Y_k$ es una producción con $k \geqslant 1$, entonces si $a \in \text{FIRST}(Y_1)$ se tiene $a \in \text{FIRST}(X)$.
  Es decir que $\text{FIRST}(Y_1) \subseteq \text{FIRST}(X)$.

Esta definición se puede generalizar a una cadena.

Definición (Función FOLLOW)

La función FOLLOW para un símbolo no-terminal X calcula el conjunto de símbolos terminales que aparecen en una derivación cualquiera justo despúes de X:

$$\text{FOLLOW}(X) = \{a \in \Sigma \mid A \to vXaw,\ v, w \in \Gamma \cup \Sigma\}$$

Definición (Función FOLLOW)

La función FOLLOW para un símbolo no-terminal X calcula el conjunto de símbolos terminales que aparecen en una derivación cualquiera justo despúes de X:

$$\text{FOLLOW}(X) = \{a \in \Sigma \mid A \to vXaw,\ v, w \in \Gamma \cup \Sigma\}$$

Para esta función se agrega un nuevo símbolo para indicar el final del archivo a procesar mediante $\#$ (o el símbolo EOF).

Definición (Función FOLLOW)

La función FOLLOW para un símbolo no-terminal X calcula el conjunto de símbolos terminales que aparecen en una derivación cualquiera justo despúes de X:

$$\text{FOLLOW}(X) = \{a \in \Sigma \mid A \to vXaw,\ v, w \in \Gamma \cup \Sigma\}$$

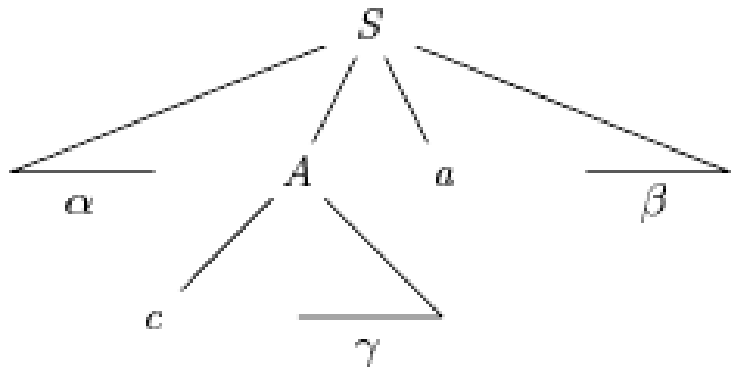
Para esta función se agrega un nuevo símbolo para indicar el final del archivo a procesar mediante $\#$ (o el símbolo EOF).
Recursivamente:

- $\# \in \text{FOLLOW}(S)$ con S el símbolo inicial de la gramática.
- Si $A \to uXw$ entonces todo símbolo en $\text{FIRST}(w)$ está en $\text{FOLLOW}(X)$.
- Si existen producciones $A \to \alpha X$ o $A \to \alpha X \beta$ entonces $\text{FOLLOW}(A) \subseteq \text{FOLLOW}(X)$.

Análisis sintáctico

Funciones auxiliares

Veamos gráficamente cómo se ven los símbolos de estas funciones en un árbol de sintaxis:



Terminal c is in $\textsc{first}(A)$ and a is in $\textsc{follow}(A)$

$$\textsc{First}(\alpha) = \{a \in \Sigma \mid \exists w, \alpha \rightarrow^* aw\}$$

$$\textsc{Follow}(X) = \{a \in \Sigma \mid A \rightarrow vXaw,\ v, w \in \Gamma \cup \Sigma\}$$

Funciones auxiliares

ejemplo

$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid n$$

Funciones auxiliares

ejemplo

$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid n$$

Se puede transformar la gramática anterior en gramáticas equivalentes que no sean ambigüas:

1. Gramática recursiva por la izquierda no-ambigüa

$$\begin{array}{lcl} E & \to & E + T \mid E - T \mid T \\ T & \to & T * F \mid T/F \mid F \\ F & \to & (E) \mid n \end{array}$$

Funciones auxiliares

ejemplo

$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid n$$

Se puede transformar la gramática anterior en gramáticas equivalentes que no sean ambigüas:

1. Gramática recursiva por la izquierda no-ambigüa

$$\begin{array}{lcl} E & \to & E + T \mid E - T \mid T \\ T & \to & T * F \mid T/F \mid F \\ F & \to & (E) \mid n \end{array}$$

2. Eliminación de la recursión por la izquierda

$$\begin{array}{lcl} E & \to & TE' \mid T \\ E' & \to & +TE' \mid -TE' \mid +T \mid -T \\ T & \to & FT' \mid F \\ T' & \to & *FT' \mid /FT' \mid *F \mid /F \\ F & \to & (E) \mid n \end{array}$$

Funciones auxiliares

ejemplo

$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid n$$

Se puede transformar la gramática anterior en gramáticas equivalentes que no sean ambigüas:

1. Gramática recursiva por la izquierda no-ambigüa

$$\begin{array}{lcl} E & \to & E + T \mid E - T \mid T \\ T & \to & T * F \mid T/F \mid F \\ F & \to & (E) \mid n \end{array}$$

2. Eliminación de la recursión por la izquierda

$$\begin{array}{lcl} E & \to & TE' \mid T \\ E' & \to & +TE' \mid -TE' \mid +T \mid -T \\ T & \to & FT' \mid F \\ T' & \to & *FT' \mid /FT' \mid *F \mid /F \\ F & \to & (E) \mid n \\ S & \to & E\# \end{array}$$

agregamos un nuevo inicial para incluir el EOF

Funciones auxiliares

ejemplo

Cálculo de los conjuntos FIRST y FOLLOW

$$
\begin{array}{lcl}
S & \rightarrow & E\# \\
E & \rightarrow & TE' \mid T \\
E' & \rightarrow & +TE' \mid -TE' \mid +T \mid -T \\
T & \rightarrow & FT' \mid F \\
T' & \rightarrow & *FT' \mid /FT' \mid *F \mid /F \\
F & \rightarrow & (E) \mid n
\end{array}
$$

Funciones auxiliares

ejemplo

Cálculo de los conjuntos FIRST y FOLLOW

$$
\begin{array}{lcl}
S & \to & E\# \\
E & \to & TE' \mid T \\
E' & \to & +TE' \mid -TE' \mid +T \mid -T \\
T & \to & FT' \mid F \\
T' & \to & *FT' \mid /FT' \mid *F \mid /F \\
F & \to & (E) \mid n
\end{array}
$$

$\text{FIRST}(Z) = \{a \in \Sigma \mid \exists w, Z \to^* aw\}$ calcular desde el “último” no-terminal

Funciones auxiliares

ejemplo

Cálculo de los conjuntos FIRST y FOLLOW

$$
\begin{array}{lcl}
S & \to & E\# \\
E & \to & TE' \mid T \\
E' & \to & +TE' \mid -TE' \mid +T \mid -T \\
T & \to & FT' \mid F \\
T' & \to & *FT' \mid /FT' \mid *F \mid /F \\
F & \to & (E) \mid n
\end{array}
$$

$\text{FOLLOW}(X) = \{a \in \Sigma \mid A \to vXaw,\ v, w \in \Gamma \cup \Sigma\}$ calcular desde el “primer” no-terminal

Funciones auxiliares

ejemplo

Cálculo de los conjuntos FIRST y FOLLOW

$$
\begin{array}{lcl}
S & \to & E\# \\
E & \to & TE' \mid T \\
E' & \to & +TE' \mid -TE' \mid +T \mid -T \\
T & \to & FT' \mid F \\
T' & \to & *FT' \mid /FT' \mid *F \mid /F \\
F & \to & (E) \mid n
\end{array}
$$

	E	E'	T	T'	F	S
FIRST	$\{(, n\}$	$\{+, -\}$	$\{(, n\}$	$\{*, /\}$	$\{(, n\}$	$-$
FOLLOW	$\{\#,)\}$	$\{\#,)\}$	$\{+, -, \#,)\}$	$\{+, -, \#,)\}$	$\{*, /, +, -, \#,)\}$	$\{\#\}$

Referencias

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman.
Compilers, Principles, Techniques and Tools.
Pearson Education Inc., Second edition, 2007.

[2] H. R. Nielson and F. Nielson.
Semantics with Applications: An Appetizer (Undergraduate Topics in Computer Science).
Springer-Verlag, Berlin, Heidelberg, 2007.

[3] F. Pfenning.
Notas del curso (15-411) Compiler Design.
https://www.cs.cmu.edu/~fp/courses/15411-f14/, 2014.

[4] M. L. Scott.
Programming Language Pragmatics.
Morgan-Kauffman Publishers, Third edition, 2009.

[5] Y. Su and S. Y. Yan.
Principles of Compilers, A New Approach to Compilers Including the Algebraic Method.
Springer-Verlag, Berlin Heidelberg, 2011.

[6] L. Torczon and K. Cooper.
Engineering A Compiler.
Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2011.