ST0244 Lenguajes de Programacion
Sintaxis

Andrés Sicard Ramírez

Universidad EAFIT

Semestre 2024-2

Preliminares

Convenciones

- La numeración (capítulos, teoremas, figuras, páginas, etc) en estas diapositivas corresponde a la numeración del texto guía [Louden y Lambert 2011].
- Los ejemplos que incluyen código fuente están en el repositorio del curso.

Introducción

Sintaxis y semántica

- La sintaxis son las reglas para escribir código fuente en un lenguaje de programación (*well-formed programs*).
- La semántica establece el significado de los programas.

Introducción

Sintaxis y semántica

- La sintaxis son las reglas para escribir código fuente en un lenguaje de programación (*well-formed programs*).
- La semántica establece el significado de los programas.

Pregunta

¿Cuando se está aprendiendo y/o usando un lenguaje de programación son la sintaxis y la semántica igualmente de importantes?

Terminología

Problemas sintácticos y semánticos

Tipo	Estático (*compile-time*)	Dinámico (*run-time*)
Sintaxis	✓	
Semántica	✓	✓

Terminología

Ejemplo (pág. 32)

¿Es `a = b + c;` una instrucción correcta en C++?

Terminología

Ejemplo (pág. 32)

¿Es `a = b + c;` una instrucción correcta en C++?

Algunas preguntas:

1. ¿Tienen `b` y `c` valores? (sí es resuelto en *run-time*, es un problema semántico dinámico, pero sí es resuelto en *compile-time*, es un problema semántico estático).
2. ¿Han `b` y `c` sido declaradas de un tipo que permite la operación `+`? (resuelto en *compile-time*, problema semántico estático).
3. Es la asignación a `a` compatible con el resultado de la expresión `b + c` (resuelto en *compile-time*, problema semántico estático).
4. ¿Tiene la instrucción de asignación la forma correcta? (resuelto en *compile-time*, problema sintáctico).

Terminología

Definición

Un símbolo **terminal** (o **token**) es un símbolo elemental del lenguaje.

Ejemplo

Palabras reservadas (*keywords*), tipos simples, operadores, números, identificadores, entre otros.

Terminología

Definición

Un símbolo **no terminal** (o **categoría sintáctica**) representa una secuencia de símbolos terminales.

Ejemplo

Tipos compuestos, instrucciones, expresiones, sentencias if, aplicación y abstracción de funciones, entre otros.

Formas de Backus-Naur

Definición

La **forma de Backus-Naur** (BNF) es un meta-lenguaje formal para especificar la sintaxis de un lenguaje.

Formas de Backus-Naur

Reglas BNF

Una BNF para un lenguaje es un conjunto de reglas de la forma

$$\langle \text{no terminal} \rangle \quad ::= \quad \texttt{expresión}_1 \mid \texttt{expresión}_2 \mid \ldots \mid \texttt{expresión}_n$$

donde

(i) $\texttt{expresión}_i$ es una cadena de terminales y no terminales,

(ii) el símbolo $::=$ significa que el símbolo no terminal a la izquierda «tiene la forma» de la expresión a la derecha,

(iii) el símbolo $\mid$ significa «o».

Formas de Backus-Naur

Ejemplo

Sea P un conjunto de variables proposicionales (fórmulas atómica) y sea $p \in P$. Podemos definir las fórmulas bien formadas (*well-formed formulae*) de la lógica proposicional por la siguiente BNF:

$$
\begin{aligned}
\langle \text{fórmula} \rangle \quad ::= \quad & \mathsf{p} \\
& \mid \neg \langle \text{fórmula} \rangle \\
& \mid \mathsf{(}\ \langle \text{fórmula} \rangle \wedge \langle \text{fórmula} \rangle\ \mathsf{)} \\
& \mid \mathsf{(}\ \langle \text{fórmula} \rangle \vee \langle \text{fórmula} \rangle\ \mathsf{)} \\
& \mid \mathsf{(}\ \langle \text{fórmula} \rangle \to \langle \text{fórmula} \rangle\ \mathsf{)} \\
& \mid \mathsf{(}\ \langle \text{fórmula} \rangle \leftrightarrow \langle \text{fórmula} \rangle\ \mathsf{)}
\end{aligned}
$$

Observación

Note que la definición de las fórmulas bien formadas es recursiva.

Formas de Backus-Naur

Ejemplo

Una BNF describiendo los números enteros, con o sin signo (*p. ej.* -344, 56, +9784, 8, 000).

$$
\begin{aligned}
\langle \text{entero} \rangle &::= \langle \text{signo} \rangle \langle \text{dígitos} \rangle \mid \langle \text{dígitos} \rangle \\
\langle \text{dígitos} \rangle &::= \langle \text{dígito} \rangle \langle \text{dígitos} \rangle \mid \langle \text{dígito} \rangle \\
\langle \text{dígito} \rangle &::= \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \texttt{3} \mid \texttt{4} \mid \texttt{5} \mid \texttt{6} \mid \texttt{7} \mid \texttt{8} \mid \texttt{9} \\
\langle \text{signo} \rangle &::= \texttt{+} \mid \texttt{-}
\end{aligned}
$$

Formas de Backus-Naur

Ejemplo

La siguiente BNF genera las instrucciones de asignación en un lenguaje de programación.

⟨instrucción-asignación⟩ ::= ⟨var⟩ `:=` ⟨expr-aritmética⟩

⟨expr-aritmética⟩ ::= ⟨var⟩ | ⟨const⟩ |
`(` ⟨expr-aritmética⟩⟨op-aritmético⟩⟨expr-aritmética⟩ `)`

⟨op-aritmético⟩ ::= `+` | `-` | `*` | `/`

⟨const⟩ ::= `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9`

⟨var⟩ ::= `a` | `b` | `c` | ... | `x` | `y` | `z`

Formas de Backus-Naur

BNF extendida (EBNF)

La forma BFN se puede extender con las siguientes definiciones adicionales:

i) ítem? o [ítem] significa que el ítem es opcional.

ii) ítem* o {ítem} significa que cero o más ocurrencias del ítem son permitidas.

iii) ítem+ significa que una o más ocurrencias del ítem son permitidas.

iv) Paréntesis pueden ser usados para agrupación.

Formas de Backus-Naur

Ejemplo

Una BNF y una EBNF describiendo los números enteros, con o sin signo.

$$
\begin{aligned}
\langle \text{entero} \rangle &::= \langle \text{signo} \rangle \langle \text{dígitos} \rangle \mid \langle \text{dígitos} \rangle \\
\langle \text{dígitos} \rangle &::= \langle \text{dígito} \rangle \langle \text{dígitos} \rangle \mid \langle \text{dígito} \rangle \\
\langle \text{dígito} \rangle &::= \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \ldots \mid \texttt{9} \\
\langle \text{signo} \rangle &::= \texttt{+} \mid \texttt{-}
\end{aligned}
$$

BNF

$$
\begin{aligned}
\langle \text{entero} \rangle &::= \text{signo}? \ \text{dígito}^{+} \\
\langle \text{dígito} \rangle &::= \texttt{0} \mid \texttt{1} \mid \texttt{2} \mid \ldots \mid \texttt{9} \\
\langle \text{signo} \rangle &::= \texttt{+} \mid \texttt{-}
\end{aligned}
$$

EBNF

Formas de Backus-Naur

Ejemplo

Una BNF y una EBNF describiendo una lista de identificadores.

$$\langle \text{lista-id} \rangle \quad ::= \quad \texttt{id} \ \texttt{;} \ \langle \text{lista-id} \rangle \ \mid \ \texttt{id}$$

BNF

$$\langle \text{lista-id} \rangle \quad ::= \quad \texttt{id} \ (\texttt{;} \ \texttt{id})^*$$

EBNF

Gramáticas independientes del contexto

Descripción

Una **gramática independiente del contexto** es un meta-lenguaje formal para especificar la sintaxis de un lenguaje.

Gramáticas independientes del contexto

Definición

Una **gramática independiente del contexto** (o **gramática libre de contexto**) (*context-free grammar*) es una estructura

$$G = (\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S}),$$

donde

(i) $\mathcal{N}$ es un conjunto finito de símbolos no terminales,

(ii) $\mathcal{T}$ es un conjunto finito símbolos terminales,

(iii) $\mathcal{P}$ es un conjunto finito de producciones de la forma $A \to \alpha$ con $A \in \mathcal{N}$ y $\alpha \in \{\mathcal{N} \cup \mathcal{T}\}^*$ (conjunto de símbolos terminales y no terminales incluyendo la palabra vacía ϵ),

(iv) $\mathcal{S} \in \mathcal{N}$ es el símbolo de inicio.

Gramáticas independientes del contexto

Ejemplo (gramática para expresiones infijas (§ 2.3.1))

Podemos definir una gramática independiente del contexto $(\mathcal{N}, \mathcal{T}, \mathcal{P}, E)$ para expresiones infijas por

$$\mathcal{N} = \{E, T, F\},$$
$$\mathcal{T} = \{\mathrm{id}, \mathrm{num}, +, -, *, /, (,)\},$$

y el conjunto $\mathcal{P}$ está compuesto por las siguientes producciones:

$$E \to E \ + \ T \qquad T \to T \ * \ F \qquad F \to (\ E\ )$$
$$E \to E \ - \ T \qquad T \to T \ / \ F \qquad F \to \mathrm{id}$$
$$E \to T \qquad T \to F \qquad F \to \mathrm{num}$$

Derivaciones

Definición

Una **sentencia** (*sentence*) de una gramática G es una cadena de símbolos terminales (*tokens*) de G.

Derivaciones

Definición

Una **sentencia** (*sentence*) de una gramática G es una cadena de símbolos terminales (*tokens*) de G.

Ejemplo

Dos sentencias de la gramática para las expresiones infijas son:

(i) $(\ 5 \ * \ x \ ) \ + \ y$

(ii) $) \ 4 \ + \ + \ ($

Derivaciones

Definición

Una **forma sentencial** (*sentential form*) de una gramática G es una cadena de símbolos terminales y no terminales de G.

Derivaciones

Definición

Una **forma sentencial** (*sentential form*) de una gramática G es una cadena de símbolos terminales y no terminales de G.

Ejemplo

Dos formas sentenciales de la gramática para las expresiones infijas son:

(i) $(\; T \; * \; F \;) \; + \; T$

(ii) $(\; 5 \; * \; F \; F$

Derivaciones

Definición

Una **derivación** de una sentencia S en un gramática G es una secuencia de formas sentenciales de G que comienza en el símbolo inicial de G y termina en S.

Derivaciones

Definición

Una **derivación** de una sentencia S en un gramática G es una secuencia de formas sentenciales de G que comienza en el símbolo inicial de G y termina en S.

Observación

Cada forma sentencial en la derivación es obtenida a partir de la anterior reemplazando $A \in \mathcal{N}$ (símbolo no terminal) por $\alpha \in \{\mathcal{N} \cup \mathcal{T}\}^*$ (cadena de símbolos terminales y no terminales), si $A \to \alpha$ es una producción de G.

Derivaciones

Definición

Un sentencia S de una gramática G es **válida** sii existe al menos una derivación para S en G.

Derivaciones

Ejemplo

La sentencia «(5 * x) + y» de la gramática de las expresiones infijas es válida porque tiene la siguiente derivación:

$$
\begin{aligned}
\underline{E} &\Rightarrow_1 \underline{E} \ + \ T \\
&\Rightarrow_3 \underline{T} \ + \ T \\
&\Rightarrow_6 \underline{F} \ + \ T \\
&\Rightarrow_7 (\ \underline{E} \ ) \ + \ T \\
&\Rightarrow_3 (\ \underline{T} \ ) \ + \ T \\
&\Rightarrow_4 (\ \underline{T} \ * \ F \ ) \ + \ T \\
&\Rightarrow_6 (\ \underline{F} \ * \ F \ ) \ + \ T \\
&\Rightarrow_9 (\ 5 \ * \ \underline{F} \ ) \ + \ T \\
&\Rightarrow_8 (\ 5 \ * \ x \ ) \ + \ \underline{T} \\
&\Rightarrow_6 (\ 5 \ * \ x \ ) \ + \ \underline{F} \\
&\Rightarrow_8 (\ 5 \ * \ x \ ) \ + \ y
\end{aligned}
$$

Producciones

$$
\begin{aligned}
E &\to E \ + \ T && (1) \\
E &\to E \ - \ T && (2) \\
E &\to T && (3) \\
T &\to T \ * \ F && (4) \\
T &\to T \ / \ F && (5) \\
T &\to F && (6) \\
F &\to (\ E \ ) && (7) \\
F &\to \mathrm{id} && (8) \\
F &\to \mathrm{num} && (9)
\end{aligned}
$$

Derivaciones

Definición

El **lenguaje** de una gramática G, denotado $L(G)$, es el conjunto de sentencias válidas de G.

Derivaciones

Tipos de derivaciones

(i) **Derivación más a la izquierda** (*left-most derivation*): Siempre reemplaza el símbolo no terminal más a la izquierda).

(ii) **Derivación más a la derecha** (*right-most derivation*): Siempre reemplaza el símbolo no terminal más a la derecha.

Derivaciones

Ejemplo

Derivaciones más a la izquierda y más a la derecha para la sentencia «(5 * x) + y ».

$$
\begin{aligned}
\underline{E} &\Rightarrow_1 \underline{E} + T \\
&\Rightarrow_3 \underline{T} + T \\
&\Rightarrow_6 \underline{F} + T \\
&\Rightarrow_7 (\underline{E}) + T \\
&\Rightarrow_3 (\underline{T}) + T \\
&\Rightarrow_4 (\underline{T} * F) + T \\
&\Rightarrow_6 (\underline{F} * F) + T \\
&\Rightarrow_9 (5 * \underline{F}) + T \\
&\Rightarrow_8 (5 * x) + \underline{T} \\
&\Rightarrow_6 (5 * x) + \underline{F} \\
&\Rightarrow_8 (5 * x) + y
\end{aligned}
$$

Más a la izquierda

$$
\begin{aligned}
\underline{E} &\Rightarrow_1 E + \underline{T} \\
&\Rightarrow_6 E + \underline{F} \\
&\Rightarrow_8 \underline{E} + y \\
&\Rightarrow_3 \underline{T} + y \\
&\Rightarrow_6 \underline{F} + y \\
&\Rightarrow_7 (\underline{E}) + y \\
&\Rightarrow_3 (\underline{T}) + y \\
&\Rightarrow_4 (T * \underline{F}) + y \\
&\Rightarrow_8 (\underline{T} * x) + y \\
&\Rightarrow_6 (\underline{F} * x) + y \\
&\Rightarrow_9 (5 * x) + y
\end{aligned}
$$

Más a la derecha

$$
\begin{aligned}
E &\to E + T && (1) \\
E &\to E - T && (2) \\
E &\to T && (3) \\
T &\to T * F && (4) \\
T &\to T / F && (5) \\
T &\to F && (6) \\
F &\to (E) && (7) \\
F &\to \mathrm{id} && (8) \\
F &\to \mathrm{num} && (9)
\end{aligned}
$$

Producciones

Derivaciones

Expresiones prefijas

En las expresiones prefijas el operador aparece antes que los operandos.

Ejemplo

$$4 + (a - b) * x \quad \text{(expresión infija)}$$
$$+ 4 * - a\, b\, x \quad \text{(expresión prefija)}$$

Derivaciones

Ejemplo (prefix expresiones gramática (§ 2.4.3))

Podemos definir una gramática independiente del contexto $(\mathcal{N}, \mathcal{T}, \mathcal{P}, E)$ para expresiones infijas por

$$\mathcal{N} = \{E\},$$
$$\mathcal{T} = \{\mathrm{id}, \mathrm{num}, +, -, *, /\},$$

y el conjunto $\mathcal{P}$ está compuesto por las siguientes producciones:

$$E \to + \; E \; E \mid - \; E \; E \mid * \; E \; E \mid / \; E \; E \mid \mathrm{id} \mid \mathrm{num}$$

Árboles de análisis sintáctico

Definición

Sea G una gramática. Un **árbol de análisis sintáctico** (*parser tree*) es una representación gráfica de la derivación de una sentencia de $L(G)$.

Árboles de análisis sintáctico

Definición

Sea G una gramática. Un **árbol de análisis sintáctico** (*parser tree*) es una representación gráfica de la derivación de una sentencia de $L(G)$.

Propiedades

Un árbol de análisis sintáctico de una sentencia de $L(G)$ tiene las siguientes propiedades [Aho, Lam, Sethi y Ullman 2006, pág. 45]:

(i) «La raíz se etiqueta con el símbolo inicial.»

(ii) «Cada hoja se etiqueta con un terminal, o con ϵ.»

(iii) «Cada nodo interior se etiqueta con un no terminal.»

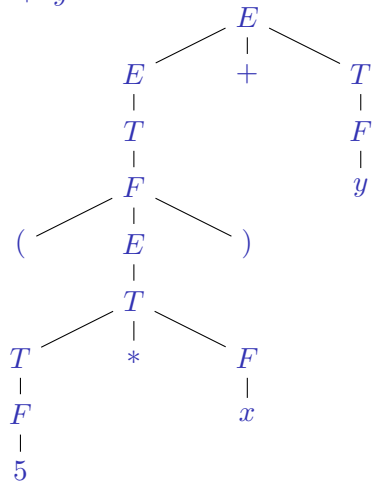
(iv) «Si A es el no terminal que etiqueta a cierto nodo interior, y $X_1, X_2, \ldots, X_n$ son las etiquetas de los hijos de ese nodo de izquierda a derecha, entonces debe haber una producción $A \to X_1, X_2, \ldots, X_n$.»

Árboles de análisis sintáctico

Ejemplo

Árbol de análisis sintáctico para la sentencia $(5 * x) + y$.

$$
\begin{aligned}
\underline{E} &\Rightarrow \underline{E} + T \\
&\Rightarrow \underline{T} + T \\
&\Rightarrow \underline{F} + T \\
&\Rightarrow (\underline{E}) + T \\
&\Rightarrow (\underline{T}) + T \\
&\Rightarrow (\underline{T} * F) + T \\
&\Rightarrow (\underline{F} * F) + T \\
&\Rightarrow (5 * \underline{F}) + T \\
&\Rightarrow (5 * x) + \underline{T} \\
&\Rightarrow (5 * x) + \underline{F} \\
&\Rightarrow (5 * x) + y
\end{aligned}
$$



Árboles sintáctico abstractos

Definición

Un **árbol sintáctico abstracto** (*abstract syntax tree*) es un árbol de análisis sintáctico sin información esencial requerida para evaluar (generar código en la compilación o ejecutar en la interpretación) la sentencia (pág. 38):

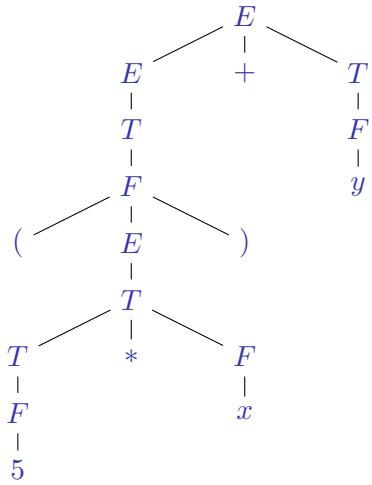
i) «*Non-terminal nodes in the tree are replaced by nodes that reflect the part of the sentencia they represent.*»

ii) «*Unit productions in the tree are collapsed.*»

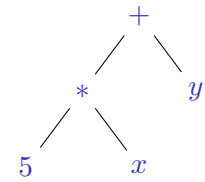
Árboles sintáctico abstractos

Ejemplo

Árbol de análisis sintáctico y árbol sintáctico abstracto (los nodos interiores representan operadores y las hojas representan operandos) para la sentencia $(5 * x) + y$.



Árbol de análisis sintáctico



Árbol sintáctico abstracto

Ambigüedad en las gramáticas

Definición

Un gramática G es **ambigua** sii existe una sentencia en $L(G)$ que tiene más de un árbol de análisis sintáctico.

Ambigüedad en las gramáticas

Ejemplo

Dada la gramática $(\mathcal{N}, \mathcal{T}, \mathcal{P}, E)$ donde

$$\mathcal{N} = \{E\},$$
$$\mathcal{T} = \{*, +, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\},$$

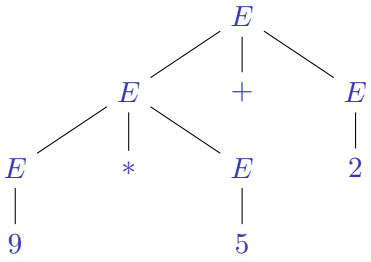
$$E \to E \ * \ E \mid E \ + \ E$$
$$E \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

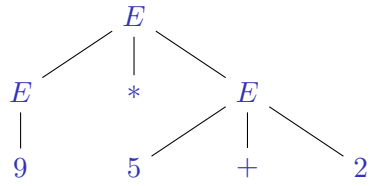
(continua en la próxima diapositiva)

Ambigüedad en las gramáticas

Ejemplo (continuación)

La sentencia $9 * 5 + 2$ tiene dos árboles de análisis sintáctico.





Limitaciones de las definiciones sintácticas

Algunas limitaciones

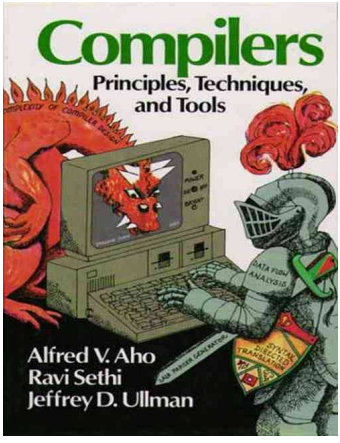
- La sintaxis de un lenguaje de programación es una descripción incompleta de éste (*p. ej.* `5 + 4/0`).
- «The set of programs in any interesting lenguaje is not context-free.» (pág. 50) (*p. ej.* `a + b`)
- Un gramática independiente del contexto no especifica la semántica de un lenguaje de programación.

Limitaciones de las definiciones sintácticas

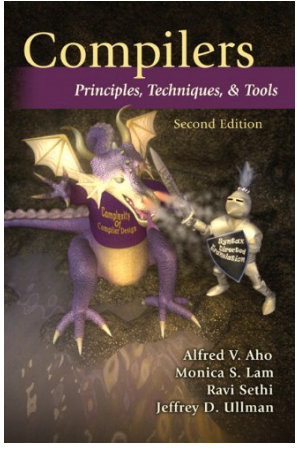
Ejemplo (context-sensitive issues (págs. 50–1))

- «*In an array declaration in C++, the array size must be a non-negative value.*»
- «*Operands for the && operation must be boolean in Java.*»
- «*In a method definition, the return value must be compatible with the return type in the method declaration.*»
- «*When a method is called, the actual parameters must match the formal parameter types.*»

El libro del drágon



(Primera edición, 1986)



(Segunda edición, 2006)

Referencias


Aho, Alfred V., Lam, Monica S., Sethi, Ravi y Ullman, Jeffrey D. [1986] (2006). Compilers: Principles, Techniques, & Tools. 2.ª ed. Addison-Wesley (vid. págs. 34, 35).


Louden, Kenneth C. y Lambert, Kenneth A. [1993] (2011). Programming Languages. Principles and Practice. 3.ª ed. Cengage Learning (vid. pág. 2).