Índice general

Índice de figuras

Índice de cuadros

Capítulo 6

Formalidades del Sintáctico

6.1. Gramáticas

6.1.1. Derivaciones

Definición 6.1.1. Se puede decribir la derivación de un símbolo no terminal σ_N para una cadena a través de las reglas de derivación sobre un símbolo inicial de dos formas: *Derivacion por la Izquierda* y *Derivación por la Derecha*.

Corolario 6.1.2. *Las derivaciones pueden se en cero o más pasos:* $\{DERIVACION^*\}$.

Corolario 6.1.3. *Las derivaciones se aplican al analisis sintáctico.*

Definición 6.1.4. Se tiene como *Derivación por la Izquierda* para un símbolo σ_N cuando este es reemplazado siempre y se situa en la parte izquierda de la regla de producción.

Ejemplo 6.1.5. Para la siguiente gramática:

Definición 6.1.6. Se tiene como *Derivación por la Derecha* para un símbolo σ_N cuando este es reemplazado siempre y se situa en la parte derecha de la regla de producción.

Ejemplo 6.1.7. Para la siguiente gramática:

6.1.1.1. Representación mediante Árboles

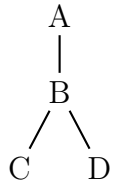

Figura 6.1: Ejemplo genérico de *Árbol de Derivación*.

Definición 6.1.8. Cualquier tipo de derivación puede ser representada gráficamente mediante un *Árbol de Derivación*.

Ejemplo 6.1.9. Para la siguiente gramática: $G = \{\sigma_T, \sigma_N, S, P)$

Dónde:

P :

Para obtener el número 399:

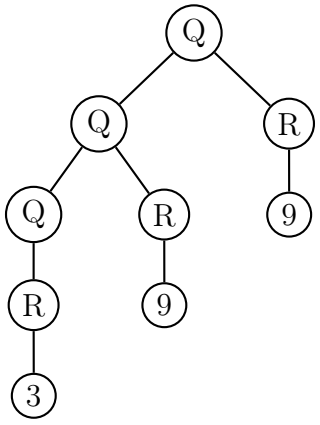

Figura 6.2: *Árbol de Derivación* para el Ejemplo:

$$Q \to QR \to QRR \to RRR \to 3RR \to 39R \to 399 \tag{6.1}$$

6.2. Introducción a los Lenguajes Formales (LFs)

6.2.1. Definiciones

Definición 6.2.1. Un Lenguaje Formal se compone de un conjunto de signos finitos y unas leyes para operar con ellos.

Definición 6.2.2. Al conjunto de símbolos de un lenguaje se les denomina *Alfabeto*, denotado como Σ.

Definición 6.2.3. Al conjunto de leyes que describen al lenguaje se les denomina *sintaxis*.

Corolario 6.2.4. *Por tanto una palabra derivada de un alfabeto pertenecerá (será propio del lenguaje) si cumple las leyes formales del mismo.*

Definición 6.2.5. Para todos los lenguajes existe la palabra vacía, que se denota en este texto mediante el símbolo λ.

Corolario 6.2.6. *Por lo tanto:*

$$|\lambda| = 0 \tag{6.2}$$

Figura 6.3: Relación entre cadenas de caracteres.

Ejemplo 6.2.7. Para el alfabeto $O = \{0, 1\}$ y la palabra p, se dice que dicha palabra pertenece al alfabeto si cumple con la sintaxis:

$$p \ \subset \ O \setminus p_0 = \lambda, \ p_1 = [01], \ p_2 = [0101], \ \ldots, \ p_n = CONCAT_{i=0}^{i=n}[01]_i \equiv [01]^* \tag{6.3}$$

6.2.2. Especificación de los LFs

Los Lenguajes Formales se pueden describir por diversos métodos, sobre los que destacan:

1. Mediante candenas producidas por una gramática de Chomsky. Ver sección (6.3)
2. Por medio de una Expresión Regular. Ver sección (**??**)
3. Por cadenas aceptadas por un Autómata.

6.2.3. ¿Qué diferencia a un Lenguaje Natural (Humano) de un LF?

Para responder a esta pregunta, debemos aclarar que entendemos por Lenguaje Natural. Los Lenguajes Naturales tienen estructuras básicas en común con los Lenguajes Formales (de hecho la especificación formal se basa en el Lenguaje Humano).

El denominador común es la palabra como unidad estructural para construir oraciones. Por ello se tiene un alfabeto Σ para los Lenguajes Naturales, que es finito. La diferencia real entre estas dos formas de lenguajes radica en la polisemia (distintos significados) que tiene una palabra dentro de una oración (semántica) es decir, el significado varía según su posición y el contexto en el que se formula.

Ejemplo 6.2.8. Dados las siguientes palabras:

$$\{Javier, rompió, la, ventana\} \tag{6.4}$$

Se puede construir la frase:

$$Javier\ rompió\ la\ ventana \tag{6.5}$$

que sintáctica y semánticamente es correcta, pero la oración:

$$La\ ventana\ rompió\ Javier \tag{6.6}$$

es sintácticamente correcta pero no semánticamente.

Por supuesto otra característica que diferencia a estos dos lenguajes es que un Lenguaje Formal como el Castellano ha sido perfeccionado a lo largo del tiempo. Con esto decimos que los Lenguajes Naturales evolucionan y están directamente relacionados con el tiempo.

6.3. Jerarquía de Chomsky (JC)

6.3.1. Definiciones

Nivel	*Lenguaje*	*Autómata*	*Producciones*
0	Recursivamente Enumerable (LRE)	Máquina de Turing (MT)	Sin restricciones
1	Dependiente del Contexto (LSC)	Autómata Linealmente Acotado	$\alpha A\beta \to \alpha\gamma B$
2	Independiente del Contexto (LLC)	Autómata a Pila	$A \to \gamma$
3	Regular (LR)	Autómata Finito	$A \to aB$ $A \to a$

Cuadro 6.1: Tabla de relaciona entre: Nivel, Lenguaje, Autómata y Producciones en la JC.

6.3.2. Niveles

La Jeraquía de Chomsky[1] contiene los siguientes niveles.

Gramáticas de tipo 0 (No Restrictivas):

Reglas de producción:

$$P = \{(p_1 \to p_2) \mid p_1 = xAy; p_1 \in \Sigma^+; p_2, x, y \in \Sigma^*; A \in N\} \tag{6.7}$$

Ejemplo 6.3.1.

$$example \tag{6.8}$$

Gramáticas de tipo 1 (Sensibles al Contexto):

Reglas de producción:

$$P = \{(S \to \lambda) \vee (xAy \to xp_2y) \mid p_2 \in \Sigma^+; x, y \in \Sigma^*; A \in N\} \tag{6.9}$$

Ejemplo 6.3.2.

$$example \tag{6.10}$$

Gramáticas de tipo 2 (Libres de Contexto):

Reglas de producción:

$$P = \{(S \to \lambda) \vee (A \to p_2) \mid p_2 \in \Sigma^+; A \in N\} \tag{6.11}$$

Ejemplo 6.3.3.

$$example \tag{6.12}$$

Gramáticas de tipo 3 (Regulares):

Reglas de producción:

$$P = \{(S \to \lambda) \vee (A \to aB) \vee (A \to a) \mid a \in T; A, B \in N\} \tag{6.13}$$

$$P = \{(S \to \lambda) \vee (A \to Ba) \vee (A \to a) \mid a \in T; A, B \in N\} \tag{6.14}$$

Ejemplo 6.3.4.

$$example \tag{6.15}$$

Tipo	*Para producciones de tipo* $p_1 \longrightarrow p_2$
0	$(p_1 \to p_2) \equiv$ No restrictivas.
1	$(S \to \lambda) \vee (xAy \to xp_2y)$
2	$(S \to \lambda) \vee (A \to v)$.
3	$(S \to \lambda) \vee (A \to aB) \vee (A \to Ba) \vee (A \to a)$

Cuadro 6.2: Grados en la JC para producciones de tipo $p_1 \longrightarrow p_2$.

6.4. Descripción de Gramáticas Formales

6.4.1. Backus-Naur Form

Backus[2]-Naur[3] Form se trata una de las dos notaciones más importantes para Gramáticas Libres de Contexto.

John Backus, diseñador de lenguajes en IBM propuso para el Lenguaje de Programación IAL (conocido como ALGOL 58) un meta-lenguaje. Posteriormente con la publicación de ALGOL 60 la fórmula BNF se simplificó y perfeccionó.

BNF se trata de un conjunto de reglas derivativas del tipo: `<symbol>::= _expression_`

Dónde:

1. `symbol`: Es un símbolo No Terminal.
2. `_expression_`: Consiste en un conjunto de símbolos o de secuencias (separadas por el carácter '|') donde el símbolo "más a la izquierda" es el Terminal.
3. `'::='`: Operador de asignación. Indica que el símbolo de la izquierda es sustituido por la expresión de la derecha.

```
<syntax>         ::= <rule> | <rule> <syntax>
<rule>           ::= <opt-whitespace> "<" <rule-name> ">" <opt-whitespace> "::="
  <opt-whitespace> <expression> <line-end>
<opt-whitespace> ::= " " <opt-whitespace> | ""
<expression>     ::= <list> | <list> "|" <expression>
<line-end>       ::= <opt-whitespace> <EOL> | <line-end> <line-end>
<list>           ::= <term> | <term> <opt-whitespace> <list>
<term>           ::= <literal> | "<" <rule-name> ">"
<literal>        ::= '"' <text> '"' | "'" <text> "'"
```

EBNF: Existen distintas variantes sobre BNF. Las más popular es Extended Backus-Naur Form (EBNF) que incopora operadores de Expresiones Regulares como:

i. a^+: Repetir a de 1 a n veces.

ii. a^*: Repetir a de 0 a n veces.

6.4.2. Wijngaarden Form

Var Wijngaarden Form (también conocida como vW-grammar p W-grammar) se trata de una técnica para definir Gramáticas Libres de Contexto en un número finito de reglas.

Las W-grammars se basan en la idea de que los símbolos No Terminales intercambian información entre los nodos y el árbol de "parseo".

El primer uso de estas gramáticas fue en ALGOL 68.

6.5. Especificaciones

6.6. Reconocimiento

6.7. Definiciones

Definición 6.7.1. La función de un *analizador sintáctico* es la de relacionar la cadena de *tokens* elaborada por el *analizador léxico* y la de comprobar que la secuencia de estos *tokens* se corresponden con patrones sintácticos (las reglas) del lenguaje.

Corolario 6.7.2. *El analizador sintáctico es el encargado de elaborar el árbol de análisis del código fuente sobre el que trabajaran el resto de fases de los compiladores.*

Definición 6.7.3. El *analizador sintáctico* es capaz de detectar errores en segunda fase, es decir, en la correspondencia entre *token* y *patrón sintáctico*.

Corolario 6.7.4. *Al contrario que ocurre con los errores léxicos, los errores sintácticos tiene una gran cosistecia*[1].

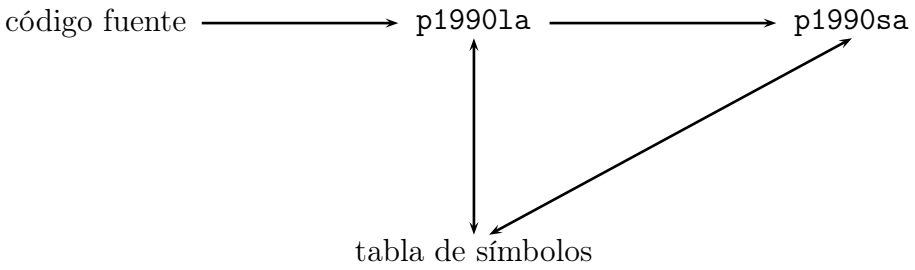

Figura 6.4: Relación entre el *analizador léxico*, *analizador sintáctico* y el programa fuente.

6.7.1. Automátas LL(1)

6.7.2. Conjunto de los Primeros

Partiendo de una gramáctica: $G = (\Sigma_T, \Sigma_N, S, P)$

Definición 6.7.5. Si α es una forma sentencial compuesta por una concatenación de símbolos $PRIM(\alpha)$ es el conjunto de terminales o λ que pueden aparacer iniciando las cadenas que pueden derivar de *alpha*.

Formalidad 6.7.6. $a \in PRIM(\alpha)$ si $a \in (\Sigma_T \cup \{\lambda\}) \wedge \alpha \Longrightarrow^* a\beta$

Reglas 6.7.7. Para calcular el conjunto de los primeros tenemos:

1. Si $\alpha \equiv \lambda \Rightarrow PRIM\{\lambda\} = \{\lambda\}$.
2. Si $\alpha \in (\Sigma_T \cup \Sigma_N)^+ \Rightarrow \alpha = a_1, a_2, \ldots, a_n$ demuestra:
   - i) Si $a_1 \equiv a \in \Sigma_T \Rightarrow PRIM(\alpha) = \{a\}$.
   - ii) Si $a_1 \equiv A \in \Sigma_N$ para:
     1) $PRIM(A) = \cup_{i=1}^{n} PRIM(\alpha_i) \backslash \alpha_i \in P$
     2) Si $PRIM(A) \setminus \lambda \in PRIM(A) \wedge A$ no es el último símbolo de $\alpha \Rightarrow PRIM(\alpha) = (PRIM(A) - \{\lambda\}) \cup PRIM(a_2, a_3, \ldots, a_n)$
     3) Si A es el último símbolo de $\alpha \vee \lambda \notin PRIM(A) \Rightarrow PRIM(\alpha) = PRIM(A)$

Ejemplo 6.7.8.

$$example \tag{6.16}$$

Ejemplo 6.7.9.

$$example \tag{6.17}$$

[1]Están perfectamente definidos en el Lenguaje de Programación.

6.7.3. Conjunto de los Siguientes

Nota: Partiendo de la gramática (Referencia a la gramática 2.3.2).

Definición 6.7.10. Si A es un símbolo inicial no terminal de la gramática, $SIG(A)$ es el conjunto de terminales + {\$} que pueden aparecer a continuación de A en alguna forma sentencial derivada del símbolo inicial.

Formalidad 6.7.11. $a \in SIG(A)$ si $a \in (\Sigma_{TN} \cup \{\$\}) \wedge \exists \alpha, \beta \backslash S \Longrightarrow^* \alpha A a \beta$

Reglas 6.7.12. Para calcular el conjunto de los siguientes tenemos:

1. Partimos de que: $SIG(A) = \varnothing$
2. Si A es símbolo inicial $\Rightarrow SIG(A) = SIG(A) \cup \{\$\}$
3. Dada la regla: $B \rightarrow \alpha A \beta \Rightarrow SIG(A) = SIG(A) \cup (PRIM(\beta) - \{\lambda\})$
4. Dada la regla: $B \rightarrow \alpha A \vee B \rightarrow \alpha A \beta \setminus \lambda \in PRIM(\beta) \Rightarrow SIG(A) = SIG(A) \cup SIG(B)$
5. Repetir los pasos 3 y 4 hasta que no se puedan añadir más símbolos a $SIG(A)$

Ejemplo 6.7.13.

$$example \tag{6.18}$$

6.7.4. Conjuntos de Predicción

Definición 6.7.14. Para una gramática ASDP con símbolo no terminal σ_N se debe consultar el símbolo de entrada y buscar en cada regla de dicho símbolo. Si los conjuntos de producción son disjuntos, el AS podrá construir una derivación hacia la izquierda de la cadena de entrada.

Formalidad 6.7.15. $PRED(A \rightarrow \alpha) \Rightarrow$

1. Si $\alpha \in PRIM(\alpha) \Rightarrow (PRIM(\alpha) - \{\lambda\} \cup SIG(A)$
2. Si no $\Rightarrow PRIM(\alpha)$

Ejemplo 6.7.16.

$$example \tag{6.19}$$

Notas del capítulo

[1]**Avram Noam Chomsky** es unos de los mayores liguistas del siglo XX. Nació en Filadelfia el 7 de diciembre de 1928. A través de sus estudios sobre la formalidad de los lenguajes enuncia su teoría sobre "La adquisición individual" dónde intenta dar explicación a las formalidades de los lenguajes naturales a través de representaciones formales.

[2]**John Backus** fue un importante científico de computación nacido en el estado de Filadelfia (EEUU), el 3 de diciembre de 1924. Es prestigioso ganador del Premio Turing en el año 1977 debido en gran parte a sus trabajos sobre especificación de lenguajes de alto nivel.

Backus estuvo dentro del primer proyecto de FORTRAN, el primer lenguaje de alto nivel en la hitoria de la computación. Además su notación sobre gramáticas sentó las bases para ALGOL.

Su famosa notación es Backus Naur Form (BNF) que describe un autómata a partir del un conjunto de símbolos (ver Definiciones: [**??**, **??**]).

[3]**Peter Naur** es un prestigio científico danés nacido el 25 de octubre de 1928 ganador del Premio Turing en 2005. Su trabajo más representativo consiste en sentar junto a John Backus la notación para especificación de autómatas para lenguajes formales.

Bibliografía

[vdHea05] Jan-Jaap van der Heijden et al. *The GNU Pascal Manual*. GPC Web, 2005.