Lenguajes y Máquinas

Silvia Takahashi

29 de julio de 2014

Capítulo 1

Conceptos Básicos

Este capítulo presenta una visión global de la teoría de lenguajes. Primero se dan las definiciones básicas. Luego se describen los conceptos de sintaxis y semántica. Estos conceptos serán cubiertos en más profundidad en los capítulos siguientes.

Dado un lenguaje, se debe poder determinar si una cadena pertenece o no al lenguaje. Note que el problema de determinar si un programa en un lenguaje de programación es correcto sintácticamente o no, es un caso especial de este problema. Otro problema es saber cuál es el significado de cada uno de los elementos del lenguaje. Esto puede querer decir algo tan sencillo como un valor en un dominio dado o su comportamiento sobre una máquina. A lo primero (la forma) se le llama sintaxis y a lo segundo (el significado) se le llama semántica.

1.1. Definiciones Generales

En esta sección se presentan las definiciones básicas de la teoría de lenguajes. Estas definiciones son necesarias para el resto del libro[1]. El lector notará la analogía entre la teoría de cadenas y la teoría de secuencias vista en [Gries94].

Comenzamos con los elementos constructores de los lenguajes. El elemento más sencillo es el alfabeto:

Definición 1. Un **alfabeto** es un conjunto finito de símbolos.

Se debe resaltar lo siguiente de esta definición:

1. Los alfabetos son finitos.
2. Por símbolo no se está haciendo referencia a un solo carácter. Los símbolos se pueden denotar con nombres.

Los siguientes son ejemplos de alfabetos:

- $\{a, b, c\}$
- $\{while, if, else, +, ++, \{, \}, -, =, class, extends, private, public\}$
- $\{sustantivo, verbo, pronombre, adjetivo, adverbio, preposicion\}$
- $\{\alpha_1, \alpha_2, ..., \alpha_n\}$

[1] Las definiciones básicas para el análisis de lenguajes mostradas en esta sección son las que se encuentran en los libros dedicados a teoría de compiladores, teoría de lenguajes, teoría de la computación. Hay pequeñas variaciones en cuanto a la notación. Las referencias [Denning78], [Lewis81] y [López87] fueron usadas como base para esta sección

El siguiente concepto de la teoría de lenguajes es el de *cadena* o *palabra* (o programa!). Dado un alfabeto, podemos formar palabras o cadenas con los símbolos del alfabeto. Por ejemplo, dado el alfabeto $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ podemos formar las siguientes palabras o cadenas: **a, b, ab, cab, bb, aaaa**.

Definición 2. Dado un alfabeto A, una cadena sobre A es una sucesión de longitud finita (mayor o igual a cero) de símbolos del alfabeto. La letra griega *lambda*, λ, se usa para denotar la cadena vacía sobre cualquier alfabeto.

En esta definición también se debe resaltar el hecho que las cadenas son de longitud finita.

Ejemplo 1.1. Dado un alfabeto $A = \{a, b\}$, las siguientes son cadenas sobre A.

$$a, b, aa, ba, bb, ba, aaa, aba, abb, aba, baa, bba, bbb, bba$$

Ejemplo 1.2. Si el alfabeto A es:

$$\{\texttt{while}, \texttt{if}, \texttt{else}, +, ++, \{, \}, -, =, \texttt{class}, \texttt{extends}, \texttt{private}, \texttt{public}, (,), ==, \texttt{var}, \texttt{num}\}$$

, las siguientes son cadenas sobre A.

- `while (var == var) var = var + num`
- `var var var ())(`

Note que, en el ejemplo, la segunda cadena no es una instrucción correcta en Java.

La siguiente definición de cadena es recursiva. Esta definición facilitará la definición de otros conceptos así como demostraciones que usan inducción estructural sobre las cadenas.

Definición 3. Recursivamente se define una cadena sobre un alfabeto A con las siguientes dos proposiciones:

- La cadena vacía denotada por λ es una cadena sobre A.
- Si ω es una cadena sobre A (digamos $\sigma_1\sigma_2\ldots\sigma_n$ para $n \geq 0$ con cada $\sigma_i \in A$) y α es un símbolo del alfabeto A, ($\alpha \in A$) entonces $\alpha\omega$ (i.e., $\alpha\sigma_1\sigma_2\ldots\sigma_n$) es una cadena sobre A.

La longitud de una cadena es el número de símbolos de la misma. Se puede definir este concepto recursivamente sobre la estructura de las cadenas como se muestra a continuación.

Definición 4. Sea ω una cadena sobre A, la longitud de ω denotada $\#.\omega$ se define así:

- Si $\omega = \lambda$ entonces $\#.\omega| = \#.\lambda = 0$
- Si $\omega = \alpha\rho$ con $\alpha \in A$ entonces $\#.\omega = \#.\alpha\rho = 1 + \#.\rho$

También se define recursivamente el concepto de concatenación de cadenas. Intuitivamente la concatenación de cadenas se obtiene pegando cadenas. Dadas dos cadenas: $\alpha = \sigma_1\sigma_2\ldots\sigma_n$ y $\beta = \tau_1\tau_2\ldots\tau_m$ con $n \geq 0$ y $m \geq 0$, la concatenación de α y β denotada $\alpha\beta$ sería: $\sigma_1\sigma_2\ldots\sigma_n\tau_1\tau_2\ldots\tau_m$. Formalmente:

Definición 5. Si ρ y γ son cadenas sobre A entonces la concatenación de ρ y γ denotada $\rho\gamma$ se define recursivamente sobre la estructura de ρ así:

- Si $\rho = \lambda$ entonces $\rho\gamma = \lambda\gamma = \gamma$
- Si $\rho = \alpha\omega$ con $\alpha \in A$ entonces $\rho\gamma = (\alpha\omega)\gamma = \alpha(\omega\gamma)$

Ahora se define el conjunto de todas las palabras de longitud finita que pueden formarse con símbolos de un alfabeto dado A. Este conjunto se denota A^*. Primero se define A^n que denota el conjunto de todas las cadenas de longitud n que se pueden formar con símbolos de A.

Definición 6. El conjunto de todas las cadenas sobre un alfabeto A se denota A^*. Para definir A^* formalmente, primero definimos A^i.

- $A^0 = \{\lambda\}$
- $A^k = \{\alpha\omega : \alpha \in A,\ \omega \in A^{k-1}\}$ para $k > 0$
- $A^* = \bigcup_{k=0}^{\infty} A^k$

Ejemplo 1.3. Sea $A = \{a, b, c\}$

- $A^0 = \{\lambda\}$
- $A^1 = \{a, b, c\}$
- $A^2 = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$
- $$A^3 = \{aaa, aab, aac, aba, abb, abc, aca, acb, acc, baa, bab,$$ $$bac, bba, bbb, bbc, bca, bcb, bcc, caa, cab, cac, cba, cbb, cbc, cca, ccb, ccc\}$$

Finalmente, con estas definiciones se puede definir formalmente el concepto de *lenguaje*. Siempre que se habla de lenguaje, se hace con respecto a un alfabeto. Un lenguaje sobre un alfabeto es un conjunto, no necesariamente finito, de cadenas sobre el alfabeto.

Definición 7. Un lenguaje sobre un alfabeto A es un conjunto de cadenas sobre A. Formalmente, un lenguaje sobre A es un subconjunto de A^*.

Ejemplo 1.4. Dado el alfabeto: $A = \{a, b, c\}$ los siguientes son lenguajes sobre A.

- $\lambda, a, b, aa, ba, bb, bac, aaa, caaaaba$
- $\{\omega : \omega \in A^*$, tiene un número par de a's$\}$
- $\{\lambda, a, b, aa, ba, bb, ba, aaa, c\}$
- $\{a^n c b^n : n > 0\}$

Ejemplo 1.5. Dado el alfabeto $A = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +$, los siguientes son lenguajes sobre A.

- $\{123, 34, ++90+\}$
- $L = \{0, 1, 2, 3, 4, 5, 7, 8, 9, 0\} \cup \{\alpha + \beta : \alpha \in L, \beta \in L\}$. Esta última es una definición recursiva.

Dado cualquier alfabeto A, los siguientes son lenguajes sobre A:

- $\{\lambda\}$
- $\emptyset$
- A^*
- A^i para $i \geq 0$

Tambien existe el concepto de concatenación de lenguajes. Esta definición se basa en la definición de concatenación de cadenas. La concatenación de dos lenguajes L_1 y L_2 es el lenguaje que contiene todas las cadenas formadas al concatenar cada una de las cadenas de L_1 con cada una de las cadenas de L_2

Definición 8. Dados dos lenguajes: L_1 y L_2 podemos definir la concatenación de estos lenguajes (denotada L_1L_2), así: $\{\omega\gamma : \omega \in L_1, \gamma \in L_2\}$.

Ejemplo 1.6. Sean: $L_1 = \{a, ab, bb\}$, $L_2 = \{b, a, \lambda\}$, $L_3 = \{ab, aa\}$. Entonces:

- $L_1L_1 = \{aa, aab, abb, aba, abab, abbb, bba, bbab, bbbb\}$
- $L_1L_2 = \{ab, aa, a, abb, aba, bbb, bba, bb\}$
- $L_1L_3 = \{aab, aaa, abab, abaa, bbab, bbaa\}$

Note que como un lenguaje es un conjunto, entonces no tiene elementos repetidos.

Dado un lenguaje L, es útil definir el lenguaje formado concatenando L palabras L cuantas veces se quiera. Para esto se usa el operador $*$, que se introdujo en la definición **??**. Formalmente:

Definición 9. Dado un lenguaje L, $L^* = \{\lambda\} \cup \{\omega_1\omega_2 \dots \omega_n \mid n \geq 0, \omega_i \in L \text{ para todo } 1 \leq i \leq n\}$. Recursivamente:

- $L^0 = \{\lambda\}$
- $L^k = LL^{k-1}$
- $L^* = \bigcup_{k=0}^{\infty} L^k$

1.1.1. Ejercicios

1. Dé ejemplos de cadenas sobre el alfabeto $\{casa, amarilla, es, carro, azul, y, son\}$
2. Dé ejemplos de cadenas sobre el alfabeto $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
3. Usando las definiciones, demuestre las siguientes proposiciones:
   - Para cualquier par de cadenas: ρ y γ, se tiene que: $|\rho\gamma| = |\rho| + |\gamma|$
   - Para cualquier cadena ω, se tiene que $\lambda\omega = \omega\lambda = \omega$.
4. Dé ejemplos de cadenas sobre el alfabeto $\{NUM, +, *, (,)\}$
5. Sea $A = \{a, b\}$, encuentre A^4.
6. Sea $A = \{a\}$, encuentre A^n.
7. Dadas las siguientes afirmaciones indique si son falsas o verdaderas, justificando su respuesta.
   - Dado un alfabeto A, el lenguaje A^* siempre es infinito.
   - Dado un lenguaje L, se puede decir que siempre $\lambda \in L$.
   - Dado un lenguaje L, se puede decir que siempre $\lambda \in L^*$.
8. Sea L un lenguaje con n elementos y M un lenguaje con m elementos, cuantos elementos tiene el lenguaje LM

1.2. Sintáxis

En la sección anterior se usó la notación de conjuntos para definir lenguajes. Otra forma, es definir la forma que tienen las cadenas que pertenecen al lenguaje en lenguaje natural: diciendo cómo son o cómo pueden construirse las cadenas que pertenecen al lenguaje. Si el lenguaje es pequeño y poco complejo, estas metodologías pueden ser aceptables, pero si el lenguaje es muy grande, esto no es viable. Se introduce, entonces, un formalismo para definir lenguajes y para determinar si una cadena pertenece o no a un lenguaje: las gramáticas.

Definición 10. Una gramática es una 4-tupla (N, T, S, P) donde:

- N es un conjunto finito de símbolos llamados **no-terminales**,
- T es un conjunto finito de símbolos llamados **terminales**,
- S es un símbolo denominado **símbolo distinguido** con $S \in N$, y
- P es un conjunto de producciones. Una producción es una regla de la forma: $\rho \to \delta$ con ρ y δ cadenas de longitud mayor o igual a cero. Es decir $\rho \in (N \cup T)^*$, $\delta \in (N \cup T)^*$.

Dada una gramática $G = (N, T, S, P)$, se desea definir el lenguaje generado por G. Se comienza con una definición que muestra cómo se usan las reglas de producción para obtener una cadena a partir de otra. En este caso decimos que una cadena ω es derivable a partir de otra cadena φ. La Definición 11 define el concepto de ser derivable en un paso y la Definición 12 define el concepto de ser derivable en cero o más pasos.

Definición 11. Dada una cadena $\omega \in (N \cup T)^*$ tal que $\omega = \beta\rho\beta'$ con $A \in N$, $\beta \in (N \cup T)^*$, $\beta' \in (N \cup T)^*$, $\rho \in (N \cup T)^*$. Si hay una regla $\rho \to \delta$ en P, se dice que $\beta\delta\beta'$ es **derivable en un paso** de $\beta\rho\beta'$ y escribimos $\beta\rho\beta' \to \beta\delta\beta'$.

Definición 12. Dadas dos cadenas ω_0 y ω_n, decimos que ω_n es derivable (en cero o más pasos) de ω_0, y escribimos $\omega_0 \overset{*}{\Longrightarrow} \omega_n$. Si se cumple alguna de las siguientes condiciones:

- $\omega_0 = \omega_n$
- $\omega_0 \Rightarrow \omega_n$
- Si existen $n-1$ cadenas: $\omega_1, \omega_2, \ldots, \omega_{n-1}$ tales que $\omega_{i-1} \Rightarrow \omega_i$ para $1 \leq i \leq n$. (o equivalentemente, usando una definición recursiva, si existe una cadena ω tal que $(\omega_0 \to \omega) \wedge (\omega \overset{*}{\Longrightarrow} \omega_n)$)

Con estos conceptos, se puede definir el lenguaje generado por una gramática:

Definición 13. Dada una gramática $G = (N, T, S, P)$, el lenguaje generado por G, denotado $L(G)$ es el conjunto de todas las cadenas de terminales que se pueden derivar a partir del símbolo distinguido, es decir $L(G) = \{\omega : \omega \in T^*, S \overset{*}{\Longrightarrow} \omega\}$.

El proceso de derivación describe cómo se llega del símbolo distinguido a la cadena derivada. Los compiladores, o traductores, usan o implementan este proceso de distintas formas.

Definición 14. Dada una gramática G y una cadena $\omega \in L(G)$, una **derivación** para ω es una secuencia de cadenas $\omega_0, \omega_1, \ldots, \omega_n$ tales que $\omega_0 = S$, $\omega_n = \omega$, y $\omega_{i-1} \Rightarrow \omega_i$ para $1 \leq i \leq n$. Generalmente denotamos la derivación así: $\omega_0 \Rightarrow \omega_1 \Rightarrow \omega_2 \ldots \omega_{n-1} \Rightarrow \omega_n$.

Ejemplo 1.7. Una gramática para generar el lenguaje $L = \{a^n b^n c^n : n \geq 0\}$ se puede definir así:

- No terminales: S, Y, Z
- Terminales: a, b, c
- Símbolo distinguido: S
- Reglas:
  1. $S \to \lambda$
  2. $S \to aSYZ$
  3. $aY \to ab$
  4. $bY \to bb$
  5. $Z \to c$
  6. $ZY \to YZ$

Mostramos un ejemplo de una derivación:

$$\begin{aligned}
\underline{S} &\Rightarrow a\underline{S}YZ \\
&\Rightarrow aa\underline{S}YZYZ \\
&\Rightarrow aaa\underline{S}YZYZYZ \\
&\Rightarrow aaa\underline{Y}ZYXYZ \\
&\Rightarrow aaab\underline{ZY}ZYZ \\
&\Rightarrow aaab\underline{Y}ZZYZ \\
&\Rightarrow aaabbZ\underline{ZY}Z \\
&\Rightarrow aaabb\underline{ZY}ZZ \\
&\Rightarrow aaabb\underline{Y}ZZZ \\
&\Rightarrow aaabbb\underline{Z}ZZ \\
&\Rightarrow aaabbbc\underline{Z}Z \\
&\Rightarrow aaabbbcc\underline{Z} \\
&\Rightarrow aaabbbccc
\end{aligned}$$

Comentarios:

- Por las reglas 1 y 2 se puede afirmar que: $S \overset{*}{\Longrightarrow} a^n S(ZY)^n \Rightarrow a^n(ZY)^n$. Por lo tanto se garantiza que hay tantas Z's como a's y tantas Y's como a's.
- se sabe que todas las a's aparecen juntas y al principio de la cadena.
- Las última regla logra:$a^n S(ZY)^n \overset{*}{\Longrightarrow} a^n Y^n Z^n$
- Las reglas 3 y 4 garantizan que las b's aparecen después de las a's y que están seguidas.
- al finalizar la conversión de las Y's en b's se pasa a converitir las Z's en c's con la regla 5.
- Si se aplica la regla 5 cuando aún aparece alguna Y a la derecha, esta nunca podrá convertirse en una b. En el ejemplo, de derivación, no se peude terminar si se hubiese aplicado la regla 5, antes de pasar las Y a la izquierda estas no se habrían podido convertir en b.

Se dice que una gramática G genera una cadena ω si existe una derivación para la cadena con la gramática. Puede que haya una derivación que no llegue a esa cadena, pero con tal de que haya una entonces, se puede decir que la gramática la genera. Para decir que una gramática no genera una cadena, habría que demostrar la imposibilidad que se genere.

Ejemplo 1.8. Mostramos una derivación que no logra llegar a una cadena de terminales.

$$\begin{aligned} S &\Rightarrow aSYZ \\ &\Rightarrow aaSYZYZ \\ &\Rightarrow aaaSYZYZYZ \\ &\Rightarrow aaaYZYXYZ \\ &\Rightarrow aaabZYZYZ \\ &\Rightarrow aaabYZZYZ \\ &\Rightarrow aaabbZZYZ \\ &\Rightarrow aaabbcZYZ \\ &\Rightarrow aaabbcYZZ \\ &\Rightarrow aaabbcYcZ \\ &\Rightarrow aaabbcYcc \end{aligned}$$

Este ejemplo muestra que no se pueden generar b's a la derecha de c's.

Las definiciones presentadas se refieren a las gramáticas generales sin restricciones. Se restringen estas gramáticas estableciendo la forma que pueden tomar las producciones obteniendo: gramáticas dependientes del contexto, gramáticas independientes del contexto y gramáticas regulares. A continuación se definen los distintos tipos de gramáticas. La restricción más débil es la siguiente.

Definición 15. Una gramática es **Dependiente del contexto** si para cualquier producción $\rho \to \delta$, $\#.\rho \leq \#.\delta$.

La gramática del ejemplo no es dependiente del contexto, pues la regla 1 no cumple con la condición que la longitud de la parte izquierda sea menor o igual que la longitud de la parte derecha. Note, entonces, que estas gramáticas no pueden servir para generar lenguajes que contienen la cadena vacía.

Modificamos el lenguaje agegándole un símbolo de final de cadena. Este lenguaje sí se puede generar con una gramática que es dependiente del contexto. Este símbolo de final de cadena, además nos sirve para guiar la generación del símbolo c apartir de Z, sin correr el riesgo de que existan símbolos Y a su derecha.

Ejemplo 1.9. Una gramática dependiente del contexto para $\{a^n b^n c^n \# : n \geq 0\}$:

- No terminales: S, X, Y, Z
- Terminales: a, b, c, #
- Símbolo distinguido: S
- Reglas:
  1. S → X#
  2. S → #
  3. X → aYZ
  4. X → aXYZ
  5. aY → ab
  6. bY → bb
  7. Z# → c#
  8. Zc → cc
  9. ZY → YZ

Mostramos un ejemplo de una derivación:

Ejemplo 1.10. Derivación de aaabbbccc#:

$$
\begin{aligned}
S &\Rightarrow X\# \\
&\Rightarrow aXYZ\# \\
&\Rightarrow aaXYZYZ\# \\
&\Rightarrow aaaYZYZYZ\# \\
&\Rightarrow aaabZYZYZ\# \\
&\Rightarrow aaabYZZYZ\# \\
&\Rightarrow aaabbZZYZ\# \\
&\Rightarrow aaabbZYZZ\# \\
&\Rightarrow aaabbYZZZ\# \\
&\Rightarrow aaabbbZZZ\# \\
&\Rightarrow aaabbbZZc\# \\
&\Rightarrow aaabbbZcc\# \\
&\Rightarrow aaabbbccc\#
\end{aligned}
$$

El siguiente tipo de gramáticas restringen la forma del lado izquierdo de las reglas de producción: admintiendo únicamente un no-terminal al lado izquierdo.

Definición 16. Una gramática es **Independiente del contexto** si todas las producciones son de la forma $A \to \delta$ donde $A \in N$ y $\delta \in (N \cup T)^*$[2].

Estas gramáticas se denominan independientes del contexto porque podemos reemplazar el no terminal de la izquierda por la parte derecha sin importar el contexto donde se encuentre. Vale la pena notar que el contexto al que se refiere es un concepto puramente sintáctico. Quiere decir que el no terminal A siempre se puede reemplazar por δ, sin importar el contexto (los símbolos que están al lado de A) en que aparezca A. En las gramáticas dependientes del contexto, se permiten regalas de esta forma: $\varphi A \gamma \to \varphi \delta \gamma$, en este caso se puede reemplazar A por δ, si A está en el contexto de φ y γ.

Ejemplo 1.11. La siguiente gramática sirve para generar el lenguaje: $a^n b^n : n \leq 0$. $G = (\{S\}, \{a, b\}, S, \{S \to aSb, S \to \lambda\})$

- Una derivación para aaabbb sería: $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$

Hay una última restricción que puede agregarse a las gramáticas. La idea es restringir la forma que tiene la parte derecha de las reglas:

Definición 17. Una gramática es **regular** si la parte derecha de la regla tiene a lo más un no terminal y este siempre aparece a la izquierda o a la derecha. Es decir, si ocurre una y solo una de las dos condiciones siguientes:

- Todas las producciones son de la forma: $A \to \delta B$ o $A \to \delta$ con $\delta \in T^*$, $A \in N$ y $B \in N$. Es decir, la parte derecha de las reglas tiene a lo más un no-terminal y este aparece siempre a la derecha.
- Todas las producciones son de la forma: $A \to B\delta$ o $A \to \delta$ con $\delta \in T^*$, $A \in N$ y $B \in N$. Es decir, la parte derecha de las reglas tiene a lo más un no-terminal y este aparece siempre a la izquierda.

En el primer caso se dice que la gramática es regular por la derecha y en el segundo se dice que es regular por la izquierda.

Cada una de estas restricciones define, no sólo una clase de gramáticas, también define una clase de lenguajes y describen una jerarquía entre los tipos de lenguajes[3]. Un lenguaje es regular si puede generarse con una gramática regular; un lenguaje es independiente del contexto si puede generarse con una gramática independiente del contexto y un lenguaje es dependiente del contexto si puede generarse con una gramática

[2]Note que en estas gramáticas el lado derecho sí puede ser λ.
[3]Esta jerarquía de lenguajes se denomina la Jerarquía de Chomsky por el lingüista Noam Chomsky

dependiente del contexto [4]. Finalmente los lenguajes definidos con gramáticas sin restricciones son lenguajes recursivamente enumerables. Esta jerarquía también incluye las máquinas con las que se pueden reconocer los lenguajes. Los lenguajes recursivamente enumerables son reconocidos por máquinas de Turing; los lenguajes dependientes del contexto son reconocidos por Máquinas de Turing restringidas linealmente; los lenguajes independientes del contexto son reconocidos por autómatas de pila; y finalmente, los lenguajes regulares son reconocidos por autómatas finitos. Estos últimos serán explicados en profundidad en el Capítulo 2. El capítulo 3 describe brevemente los autómatas de pila. El lector interesado en profundizar en el tema de los otros formalismos puede consultar las referencias que se refieren a la teoría de la calculabilidad.

1.2.1. Gramáticas independientes del contexto

Casi siempre en el análisis de lenguajes se utilizan las gramáticas independientes del contexto o regulares. Como vimos, estas imponen una restricción sobre la forma de las reglas y son, por lo tanto, más fácilmente implementables. Dado que este texto está dedicado principalmente al procesamiento de lenguajes, el resto del texto tratará sólo este tipo de gramáticas.

En esta sección redefiniremos algunos términos para ajustarlos a las gramáticas independientes del contexto.

Definición 18. Una gramática independiente del contexto es una 4-tupla (N, T, S, P) donde:

- N es un conjunto de símbolos **no-terminales**,
- T es un conjunto de símbolos **terminales**,
- S es un símbolo denominado **símbolo distinguido** con $S \in N$, y
- P es un conjunto de **producciones**. Una producción es una regla de la forma: $A \to \delta$ con $A \in N$ y $\delta \in (N \cup T)^*$.

Ahora se darán las definiciones necesarias para determinar el lenguaje generado por la gramática. Al igual que en el caso anterior, comenzamos definiendo la noción de derivable en un paso. Las nociones de derivación y lenguaje generado por una gramática son las mismas que las dadas para las gramáticas generales.

Definición 19. Dada una cadena de $(N \cup T)^*$ de la forma $\beta A \beta'$ con $A \in N$, $\beta \in (N \cup T)^*$, $\beta' \in (N \cup T)^*$, si hay una regla $A \to \delta$ en P, se dice que $\beta\delta\beta'$ es derivable en un paso de $\beta A \beta'$ y escribimos $\beta A \beta' \Rightarrow \beta\delta\beta'$.

Se puede asociar un árbol de sintaxis a cada derivación, informalmente se puede describir la construcción del árbol de sintaxis como sigue:

- La raíz del árbol de sintaxis es el símbolo distinguido.
- Este nodo tiene como hijos los símbolos que quedan al lado derecho de la primera regla usada en la derivación.
- Cada nodo etiquetado con un símbolo no terminal tiene como hijos los símbolos de la parte derecha de la regla de producción que se aplicó para reemplazar este símbolo.
- Los símbolos terminales (así como la cadena vacía) son las hojas del árbol.

Dado un árbol de sintaxis, la cadena resultante de la derivación correspondiente es la lista de las hojas del árbol, leídas de izquierda a derecha e ignorando aquellas hojas etiquetadas con λ.

[4] Un lenguaje que contiene la cadena vacía puede ser dependiente del contexto mas no puede ser generado por una gramática dependiente del contexto por la restricción sobre la longitud de la parte derecha. En este caso, se dice que L es un lenguaje dependiente del contexto si y solamente si existe una gramática G tal que $L(G) = L - \{\lambda\}$

Ejemplo 1.12. El árbol asociado a la derivación del ejemplo 1.11 es el siguiente:

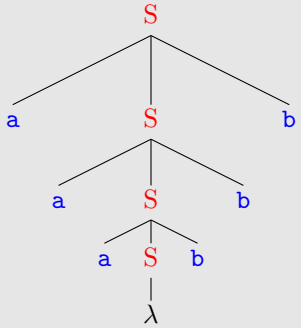


Figura 1.1: Arbol para derivación anbn

La siguiente gramática sirve para generar el lenguaje de las expresiones aritméticas entre números de un sólo dígito: $L = \{0, 1, 2, 3, 4, 5, 7, 8, 9\} \cup \{\alpha \ + \ \beta \ : \ \alpha \in L, \beta \in L\}$

- Símbolos no-terminales: E
- Símbolos terminales: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +
- Símbolo distinguido: E
- Producciones:

1. E → 0
2. E → 1
3. E → 2
4. E → 3
5. E → 4
6. E → 5
7. E → 6
8. E → 7
9. E → 8
10. E → 9
11. E → E+E

Algunas derivaciones para 3 + 7 + 9 serían [5]:

	Derivación 1		Derivación 2
E	⇒ E+E	E	⇒ E+E
	⇒ E + E+E		⇒ E+9
	⇒ 3+E+E		⇒ E+E+9
	⇒ 3+7+E		⇒ 3+E+9
	⇒ 3+7+9		⇒ 3+7+9

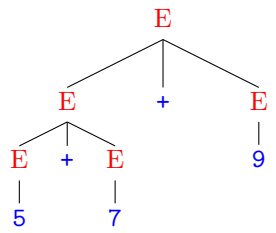


Figura 1.2

El siguiente árbol de sintaxis corresponde a ambas derivaciones.

En cambio: la siguiente derivación tiene un árbol de sintaxis diferente.

Derivación 1

$$\begin{aligned} E &\Rightarrow \underline{E}+E \\ &\Rightarrow 3+\underline{E} \\ &\Rightarrow 3+\underline{E}+E \\ &\Rightarrow 3+7+\underline{E} \\ &\Rightarrow 3+7+9 \end{aligned}$$

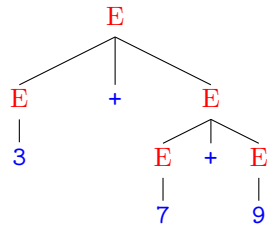


Figura 1.3

Dada una derivación:

$$\omega_0 \Rightarrow \omega_1 \Rightarrow \omega_2 \ldots \omega_{n-1} \Rightarrow \omega_n$$

decimos que es una derivación por la izquierda si en cada paso de la derivación se escoge el elemento no terminal más a la izquierda para ser reemplazado. Note que la Derivación 1 y la Derivación 3 mostradas arriba son ambas derivaciones por la izquierda. En cambio, la Derivación 2 no lo es. Esto lleva a la definición de derivable por la izquierda.

Definición 20. Dada una cadena de $(N \cup T)^*$ de la forma $\beta A \delta$ con $A \in N$, $\beta \in T^*$ y $\delta \in (N \cup T)^*$ si hay una regla $A \to \varphi$ en P, se dice que $\beta\varphi\delta$ es **derivable por la izquierda en un paso** de $\beta A\delta$ y escribimos $\beta A\delta \Rightarrow_L \beta\varpi\delta$.

[5] el no-terminal que se está reemplazando está subrayado

Definición 21. Dadas dos cadenas ω_0 y ω_n, decimos que ω_n es **derivable por la izquierda** (en cero o más pasos) de ω_0, y escribimos $\omega_0 \underset{L}{\overset{*}{\Longrightarrow}} \omega_n$. Si :

- $\omega_0 = \omega_n$
- $\omega_0 \Rightarrow^L \omega_n$
- Si existen $n-1$ cadenas: $\omega_1, \omega_2, \ldots, \omega_{n-1}$ tales que $\omega_{i-1} \Rightarrow_L \omega_i$ para $1 \leq i \leq n$

Análogamente se define el concepto de derivable por la derecha:

Definición 22. Dada una cadena de $(N \cup T)*$ de la forma $\delta A \beta$ con $A \in N$, $\beta \in T*$ y $\delta \in (N \cup T)*$ si hay una regla $A \to \varphi$ en P, se dice que $\delta\varphi\beta$ es **derivable por la derecha en un paso** de $\delta A\beta$ y escribimos $\delta A\beta \Rightarrow^R \delta\varphi\beta$.

Definición 23. Dadas dos cadenas ω_0 y ω_0, decimos que ω_n es derivable por la derecha (en cero o más pasos) de ω_0, y escribimos $\omega_0 \Rightarrow^{R*} \omega_n$. Si:

- $\omega_0 = \omega_n$
- $\omega_0 \Rightarrow^R \omega_n$
- Si existen $n-1$ cadenas: $\omega_1, \omega_2, \ldots, \omega_{n-1}$ tales que $\omega_{i-1} \Rightarrow^{R*} \omega_i$ para $1 \leq i \leq n$

Se puede demostrar que $\omega_{i-1} \Rightarrow^{L*} \omega_i \equiv \omega_{i-1} \Rightarrow^{R*} \omega_i \equiv \omega_{i-1} \Rightarrow^* \omega_i$.
Con estas definiciones se puede definir formalmente el concepto de gramática ambigua.

Definición 24. Se dice que una gramática es ambigua si existe una cadena del lenguaje generado por la gramática que tiene dos derivaciones por la **izquierda** diferentes o equivalentemente dos derivaciones por la **derecha** diferentes o si tiene dos árboles de sintaxis diferentes.

La gramática del ejemplo anterior es ambigua, ya que tiene dos derivaciones por la izquierda diferentes para la cadena 3+7+9.

Ejercicios

1. Dadas las siguientes afirmaciones, indique si son ciertas o falsas justificando su respuesta.
   a) Dada una gramática $G = (N, T, S, P)$ y una cadena $\omega = \sigma_0 sigma_1 \ldots sigma_n$ tal que $\omega \in L(G)$ entonces puede existir un σ_i con $1 \leq i \leq n$ tal que $\sigma_i \notin T$.
   b) Dada la siguiente gramática $G = (S, A, B, a, b, S, \{S \to AB, A \to a, B \to b, A \to aA, b \to bB\}$, podemos decir que G es ambigua pues la cadena ab tiene las siguientes dos derivaciones:
      - $S \Rightarrow AB \Rightarrow aB \Rightarrow ab$
      - $S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$
   c) Dado un lenguaje L se puede construir más de una gramática que lo genera.
   d) Dado un alfabeto T, la siguiente gramática $G = (\{S\}, T, S, \{S \to \lambda\} \cup \{a : T \mid \ : S \to aS\})$ se puede decir que $L(G) = T^*$.
   e) Dada una gramática $G = (N, T, S, P)$ y una cadena ω tal que $\omega \in L(G)$ entonces en el árbol de sintaxis que representa la derivación de ω puede haber hojas etiquetadas con símbolos no-terminales.
   f) Dada la siguiente gramática $G = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to aA, B \to bB\})$ $L(G) = \emptyset$
   g) Dada la siguiente gramática $G = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to aA, B \to bB\})$ $L(G) = \{\lambda\}$
2. Dadas las siguientes gramáticas clasifíquelas como regulares, independientes del contexto o dependientes del contexto. Indique sólo la clasificación más restrictiva.
   a) $G = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to a, B \to b, A \to aA, B \to bB\})$
   b) $G = (\{S, A, B\}, \{a, b\}, S, \{S \to A, S \to B, A \to a, B \to b, A \to aA, B \to bB\})$
   c) $G = (\{S, A, B\}, \{a, b\}, S, \{S \to aAa, Aa \to bBa, B \to b, A \to aAa, B \to bB\})$

1.2.2. Expresiones Regulares

Para los lenguajes regulares, existe otro formalismo para describirlos además de las gramáticas regulares: las expresiones regulares. Estas son un formalismo para representar lenguajes. Dado un alfabeto, las expresiones regulares describen conjuntos de cadenas del alfabeto. Por lo tanto, describen lenguajes sobre el alfabeto.

Definición 25. Dado un alfabeto $A = \sigma_1, \ldots, \sigma_n$ una expresión regular sobre el alfabeto A se define, recursivamente, así.

- $\emptyset$ es una expresión regular sobre el alfabeto y representa el lenguaje vacío, es decir $\{\}$.
- λ es una expresión regular sobre el alfabeto y representa el lenguaje que sólo incluye la cadena la cadena vacía, es decir $\{\lambda\}$.
- Para todo σ_i tal que $\sigma_i \in A$, σ_i es una expresión regular sobre el alfabeto y representa el lenguaje $\{\sigma_i\}$
- Si α y β son expresiones regulares que representan los lenguajes L_α y L_β respectivamente, entonces las siguientes también son expresiones regulares.
  - $(\alpha \mid \beta)$ representa el lenguaje $L_\alpha \cup L_\beta$
  - $(\alpha\beta)$ representa el lenguaje $L_\alpha \cdot L_\beta$
  - (α^*) representa el lenguaje L_α^*
  - Solamente estas son expresiones regulares.

Los paréntesis se omiten a no ser que ayude a la claridad. Se supone que $*$ tiene la máxima precedencia y $|$ la mínima.

Ejemplo 1.13. Las cadenas sobre $\{c, a, t\}$ que contienen la subcadena cat: $(c|a|t)^*cat(c|a|t)^*$

Ejemplo 1.14. Las cadenas sobre $\{c, a, t\}$ que terminan con la subcadena cat: $(c|a|t)^*cat$

Ejemplo 1.15. Las cadenas sobre $\{c, a, t\}$ que comienzan con la subcadena cat: $cat(c|a|t)^*$

Ejemplo 1.16. Las cadenas sobre $\{a, b\}$ que contienen un número de a's que es múltiplo de 3: $((b^*ab^*ab^*a)|b)^*$

Ejemplo 1.17. Las cadenas sobre $\{a, b\}$ que contienen un número de a's módulo 3 es uno: $((b^*ab^*ab^*a)|b)^*ab^*$

Ejemplo 1.18. Las cadenas sobre $\{a, b\}$ que contienen un número de a's módulo 3 es dos:

$$ab^*((b^*ab^*ab^*a)|b)^*ab^*$$

Ejemplo 1.19. Las cadenas sobre $\{a, b\}$ que contienen un número de a's que no es múltiplo de tres:

$$(((b^*ab^*ab^*a)|b)^*ab^*)|(ab^*((b^*ab^*ab^*a)|b)^*ab^*)$$

Ejemplo 1.20. Las cadenas sobre $\{a, b\}$ que no contienen aba como subcadena. Esto es un poco más complicado ya que no existe el símbolo de negación en las expresiones regulares: $((aa^*bb)|b)^*(((aa^*)(b|\lambda))|\lambda)$

Algunos de los ejemplos de arriba no son muy intuitivos. No es sencillo ver cómo construir una expresión regular a partir de una descripción en palabras del lenguaje. Abajo se presentan algunos ejemplos que intentan mostrar cómo construir una expresión regular a partir de una descripción.

Ejemplo 1.21. Las cadenas sobre $\{0,1,2,3\}$ de la forma $\sigma_1\sigma_2\ldots\sigma_n$, tales que $n \geq 0$ y tales que para todo i, $1 < i \leq n$ se cumple que $\sigma_{i+1} = (\sigma_i + 1)\ mod\ 4$. Ejemplos de cadenas que pertenecen a este lenguaje serían $0123, 3012, 012301230123, 01, 2301$.
En primer lugar se podría pensar que una expresión válida sería:

$$(0123)^*$$

El problema de esta expresión es que sólo contempla el caso en que la cadena comienza con 0, termina en 3 y tiene longitud múltiplo de 4.
Se comienza arreglando el problema de la longitud. Primero, para que la cadena tenga longitud, *long*, tal que $long\ mod\ 4 = 1$. Entonces se puede concatenar 3 a $(0123)^*$, obteniendo $3(0123)^*$. Si se quiere que $long\ mod\ 3 = 2$, se obtiene $23(0123)^*$. Para $long\ mod\ 4 = 3$, $123(0123)^*$. Combinando estas expresiones se obtiene:

$$(3|23|123|\lambda)(0123)^*$$

Aún hay problemas pues esta expresión regular sólo cubre los casos en que la cadena termina en 3. Las siguientes expresiones regulares contemplan los casos en que la cadena termina en 0, 1 y 2:

- $(0|30|230|\lambda)(1230)^*$
- $(1|01|301|\lambda)(2301)^*$
- $(2|12|012|\lambda)(3012)^*$

Para obtener la expresión completa se hace la disyunción de las cuatro expresiones obteniendo:

$$(3|23|123|\lambda)(0123)^* \mid (0|30|230|\lambda)(1230)^* \mid (1|01|301|\lambda)(2301)^* \mid (2|12|012|\lambda)(3012)^*$$

Ejercicios

1. Defina una expresión regular que reconozca el lenguaje sobre $\{0,1\}$ que representa un número binario par.
2. Defina una expresión regular sobre $\{0,1\}$ donde la paridad de número de 0's es igual a la paridad de números de 1's.
3. Defina una expresión regular sobre $\{0,1,2\}$ tal que el último símbolo de la cadena representa la suma de los dígitos que han salido hasta ahora módulo 3.
4. Defina una expresión regular sobre $\{0,1\}$ donde el último símbolo indica la paridad del resto de la cadena.
5. Defina una expresión regular sobre $\{a,b\}$ tal que contiene la cadena *abba* y la cadena *bab*. Note que las siguientes cadenas hacen parte del lenguaje: *babba* y *abbab*.
6. Defina una expresión regular que contiene la cadena aba siempre que comience con la cadena *bba*. Si no comienza con *bba*, no tiene que contener *aba*.
7. Defina una expresión regular para denotar números en notación de punto flotante.
8. Defina una expresión regular que defina el lenguaje de las cadenas compuestas de sub-cadenas de longitud 4 donde el cuarto carácter de cada sub-cadena indica su paridad.

1.3. Semántica: un primer vistazo

En esta sección se explicará brevemente el concepto de semántica. Este tema se retomará en el capitulo de Semántica y Traducción.

Como se vio en la sección anterior, la sintaxis define la forma que deben tener las cadenas que pertenecen a un lenguaje; la semántica define su significado. En el caso concreto de los lenguajes de programación, el significado de un lenguaje puede ser el valor resultante después de la ejecución del mismo o su traducción a código de máquina. Por ejemplo, para un lenguaje como HTML, su semántica es la visualización en la pantalla; para un programa escrito en Java su significado sería su traducción a ByteCode. En los primeros lenguajes de programación se definía la semántica usando lenguaje natural. Esto ocasionaba problemas ya que distintas implementaciones del lenguaje a veces tenían como resultado interpretaciones distintas de la semántica. Se quiere poder definir la semántica del lenguaje de una forma clara y sin ambigüedades.

La semántica de muchos lenguajes de programación se define en forma operacional. Es decir, se explica cómo se ejecutaría el programa en una máquina dada. Es bastante útil para la implementación de los lenguajes de programación imperativos. Pero para un lector que no esté familiarizado con la estructura del computador puede ser difícil de entender. Otra forma de definir el significado de los lenguajes es la semántica axiomática. Este enfoque también se usa más frecuentemente para los lenguajes imperativos. La idea es definir aserciones que hablan del estado del programa (el valor de las variables). La semántica denotacional se usa para traducir una sentencia escrita en un lenguaje a otro dominio (que a su vez podría ser un lenguaje).

1.4. Visión global del problema

En las secciones anteriores se dieron los fundamentos para el análisis de lenguajes. Para el lector puede no ser clara su aplicabilidad al desarrollo de software ni tampoco claro cómo se podría implementar fácilmente un analizador de lenguajes. En esta sección se presenta un ejemplo completo para el análisis de un programa sencillo.

Considere el siguiente ejemplo: Tenemos un robot que vive en un mundo de una dimensión. El robot puede moverse a la izquierda o a la derecha o puede ir a una posición dada de su mundo. Podemos manejar el robot con los siguientes comandos: $move(Dir, steps)$ o $goto(pos)$.

Una gramática para las instrucciones del robot serían:

- No terminales: I, Dir, Val
- Terminales: `move`, `goto`, (,), `INT`, “,”, `right`, `left`, −
- Símbolo distinguido: I
- Reglas:
  - I → `move`(Dir, `INT`)
  - I → `goto`(Val)
  - Dir → `right`
  - Dir → `left`
  - Val → `INT`
  - Val → −`INT`

Suponga que tenemos la cadena $move(right, 12)$. El problema es que esta cadena se compone de caracteres, no de los símbolos terminales sobre los cuales se definió la gramática. Por lo tanto, lo primero que debemos hacer es dividir la cadena en los elementos sintácticos del lenguaje: $< MOVE >$. ”(“, $< RIGHT >$, $< ENTERO$, “12” $>$, “)”. Luego sí vemos si estos elementos conforman una instrucción bien formada. El primer paso es el análisis léxico. El segundo es el análisis sintáctico. Para el segundo paso usamos la gramática que definimos arriba. Para el primer paso debemos definir las expresiones regulares que definen los tokens.

Podemos definirla así:

- $Token$ → `move`
- $Token$ → `goto`

- $Token \rightarrow$ `right`
- $Token \rightarrow$ `left`
- $Token \rightarrow$ (
- $Token \rightarrow$)
- $Token \rightarrow$,
- $Token \rightarrow$ `Num`
- $Token \rightarrow -$
- $Num \rightarrow$ (`Dig`)$^+$

El analizador léxico reconoce tokens (elementos sintácticos). El proceso de análisis léxico no sólo debe decir si está bien o no, debe dar como resultado lo que reconoce.

Algo interesante es que los lenguajes usados para definir la forma que tienen los tokens son, por lo general, lenguajes regulares. Mientras que el lenguaje que define la sintaxis del lenguaje en sí generalmente es un lenguaje independiente del contexto.

El resultado del análisis sintáctico es simplemente éxito o fracaso si sólo se está verificando la corrección sintáctica de la instrucción. Si estamos interesados en hacer un análisis semántico hay varias posibilidades. Podemos hacer un intérprete que simplemente lea una instrucción y la ejecute sobre una clase que implemente el robot. También podemos calcular un valor que en nuestro ejemplo puede ser la posición del robot suponiendo que conocemos la posición de la que parte.

Existen herramientas que nos ayudan a construir aplicaciones que hacen análisis sintáctico. Estas herramientas además pueden usarse para hacer la interpretación o para hacer cálculos de valores. En este libro, la herramienta que usaremos será JavaCC. Para el ejemplo de arriba, el siguiente archivo de JavaCC define los tokens y la gramática y realiza acciones para calcular la posición del Robot.

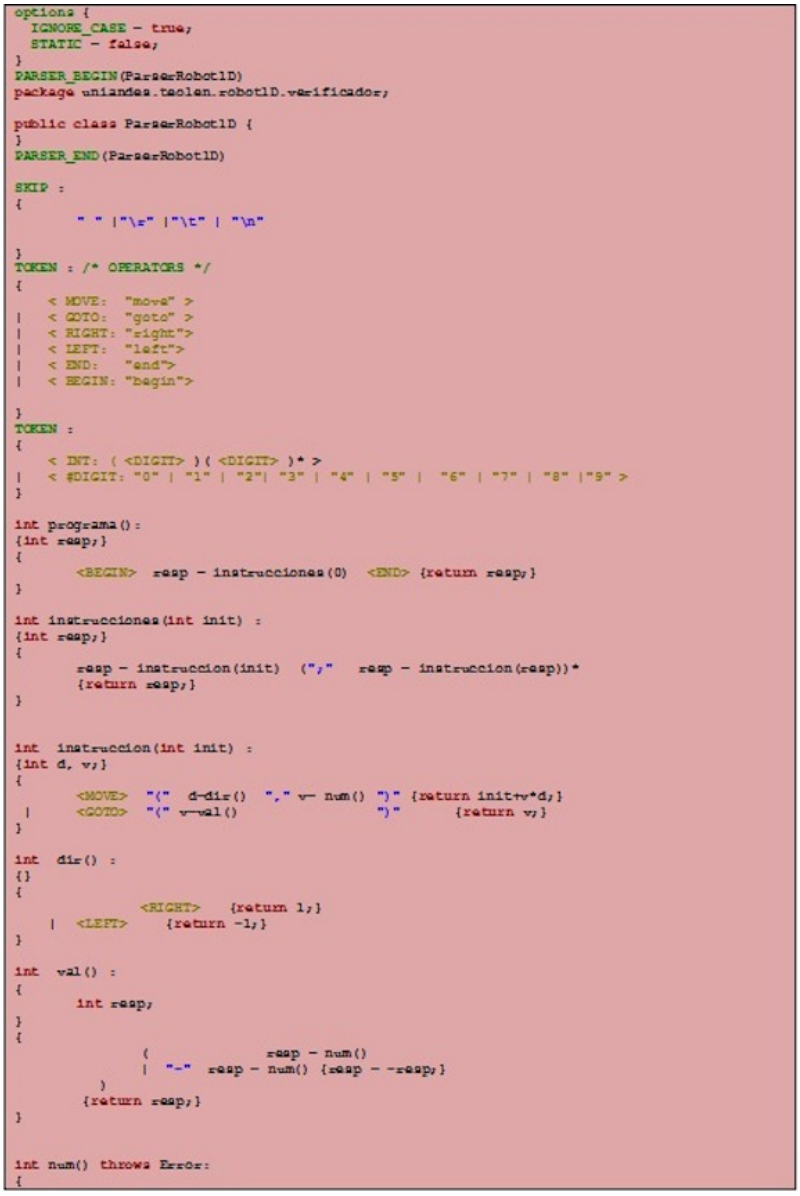
Seguramente, en este punto al lector le costará trabajo entender la sintaxis particular de JavaCC. Sin embargo, debe ser clara la relación que tiene con la definición de tokens y reglas que se menciona arriba. Note que a las reglas se les pueden pasar parámetros y éstas a su vez pueden retornar valores. En este caso la regla usada para la instrucción recibe como parámetro la instrucción donde está y retorna la posición donde queda.

Si lo que se quiere es manejar un Robot definido con la clase Robot1D que tiene métodos para cambiar la posición del Robot, lo que habría que cambiar serían las acciones realizadas al definir las reglas. También se tendría que tener acceso a los métodos del Robot. Digamos que el Robot1D tiene solo dos métodos:

- `void move(int steps)`: Si `steps` es positivo se mueve a la derecha, si es negativo, a la izquierda.
- `void go(int pos)`: Va a la posición `pos`.

Los cambios que se deben efectuar son: agregar un atributo a la clase Robot1D, así como un método setRobot. Además, se cambia la implementación de las reglas para las instrucciones. En la figura de abajo sólo se muestra lo que cambia.

El resto del libro estará dedicado a entrar en detalle en cada uno de los temas tratados en esta sección.



```
options {
  IGNORE_CASE = true;
  STATIC = false;
}
PARSER_BEGIN(ParserRobot1D)
package uniandes.taolen.robot1D.verificador;

public class ParserRobot1D {
}
PARSER_END(ParserRobot1D)

SKIP :
{
       " " |"\r" |"\t" | "\n"

}
TOKEN : /* OPERATORS */
{
    < MOVE:  "move" >
|   < GOTO:  "goto" >
|   < RIGHT: "right">
|   < LEFT:  "left">
|   < END:   "end">
|   < BEGIN: "begin">

}
TOKEN :
{
    < INT: ( <DIGIT> )( <DIGIT> )* >
|   < #DIGIT: "0" | "1" | "2"| "3" | "4" | "5" |  "6" | "7" | "8" |"9" >
}

int programa():
{int resp;}
{
       <BEGIN>  resp = instrucciones(0)  <END> {return resp;}
}

int instrucciones(int init) :
{int resp;}
{
       resp = instruccion(init)  (","   resp = instruccion(resp))*
       {return resp;}
}


int  instruccion(int init) :
{int d, v;}
{
       <MOVE>  "("  d=dir()  "," v= num() ")" {return init+v*d;}
 |     <GOTO>  "(" v=val()                ")"       {return v;}
}

int  dir() :
{}
{
              <RIGHT>   {return 1;}
    |  <LEFT>     {return -1;}
}

int  val() :
{
       int resp;
}
{
              (              resp = num()
              | "-"  resp = num() {resp = -resp;}
        )
       {return resp;}
}


int num() throws Error:
{
```

Figura 1.4

```
PARSER_BEGIN(ParserRobot1D)
package uniandes.teolen.robot1D.verificador;


public class ParserRobot1D {
Robot1D robot;

Void setRobot(Robot1D r) {
      robot = r;
}
}

PARSER_END(ParserRobot1D)
….

void programa():
{}
{
      <BEGIN>  {robot.go(0);}  instrucciones()  <END> {return resp;}
}

void instrucciones() :
{}
{
      instruccion()  (";"   instruccion())*
      {}
}


int  instruccion(int init) :
{int d, v;}
{
      <MOVE>  "("  d=dir()  "," v= num() ")" {robot.move(v*d);}
 |    <GOTO>  "(" v=val()                ")"    { robot.go(v);}
}


}
```

Figura 1.5