Aspectos generales

Como ya fue visto con anterioridad, en el estudio de los lenguajes se distinguen la *gramática* y la *semántica*. La gramática se ocupa del análisis de las estructuras de las frases, y dentro de la gramática, la *morfología* analiza las formas que toman las palabras, la *sintaxis* se ocupa de cómo combinar las palabras para formar frases correctas y la *fonética* trata al lenguaje hablado.

Por su parte, la semántica se ocupa del estudio del significado que es atribuible a expresiones sintácticamente bien formadas. El término *semántica* proviene del griego, *semantikos* (significado relevante), y se refiere al sentido o interpretación de los signos lingüísticos, tales como los símbolos, las palabras y las expresiones.

Fue Noam Chomsky quien en 1956 propuso su *Teoría de las Gramáticas Transformacionales*, en la que presentó la *gramática generativa* y estableció las bases de la *lingüística matemática.* Esta nueva disciplina posicionó a la sintaxis en el centro de la investigación lingüística, cambiando la perspectiva, la orientación y métodos de investigación en el estudio de los lenguajes. Esto no solo revolucionó el estudio de los lenguajes naturales, sino que tuvo un fuerte impacto en las entonces incipientes propuestas referidas a los lenguajes de computación y sus gramáticas. En efecto, las ideas de Chomsky fueron inmediatamente aprovechadas por la naciente *Ciencia de la Computación*, que en esa misma época estaba proponiendo los primeros lenguajes formales y desarrollando sus compiladores. Con este aporte, los lenguajes de computación experimentaron un rápido desarrollo y una sostenida evolución. Estos nuevos lenguajes, denominados de alto nivel, permitieron alcanzar una capacidad de abstracción apropiada para la especificación de datos, de su lógica y control, con independencia de la máquina que tiene a cargo la ejecución de los procesos. Se consolidó así el compilador como la principal herramienta de desarrollo e implementación de programas de computadora.

Resulta aquí oportuno insistir en la distinción entre los lenguajes *naturales* y los *formales*. Los primeros evolucionan permanentemente según los usos y costumbres de los pueblos, y sus gramáticas deben regularmente adaptarse a esta realidad. Por el contrario, los lenguajes *formales*, que tienen a los lenguajes de programación de computadoras como una de las principales aplicaciones, son desarrollados a partir de gramáticas previamente establecidas y no admiten excepciones o desviaciones. Esto se debe a que las gramáticas formales están obligadas a cumplir con el requisito primordial de estructurar mensajes que no ofrezcan dudas, condición esencial para el desarrollo de compiladores y, en general, de todas las aplicaciones informáticas.

Lo dicho no debe hacer pensar que la lingüística computacional solo se ocupa de los lenguajes formales. El procesamiento del lenguaje natural, que incluye la traducción automática, el reconocimiento del habla y el diálogo con sistemas expertos, entre otros, son campos de intenso estudio.

Conceptos de semántica de lenguajes

A partir de lo ya expuesto, puede anticiparse que **se entiende por semántica de un lenguaje, al conjunto de reglas que especifican el significado de toda sentencia sintácticamente correcta, escrita en el mismo**. En el caso de los lenguajes de programación, existen dos formas básicas que permiten describir sus contenidos semánticos: la *especificación natural* y la *especificación formal*.

La especificación natural se basa precisamente en utilizar el lenguaje natural para definir las características semánticas que no sean deducibles de la gramática que describe sintácticamente el lenguaje. En este caso, deben comprobarse tanto las especificaciones semánticas relatadas en lenguaje natural, como las derivadas de la sintaxis del lenguaje de programación. Al no disponerse de reglas fijas, se deben incorporar al compilador procedimientos y/o funciones que son denominadas semánticas y que tienen por finalidad establecer y comprobar esas especificaciones. Dependiendo de la complejidad de las mismas, estas rutinas serán procedimientos independientes o simplemente sentencias intercaladas entre las específicas del análisis sintáctico para incorporarle las condiciones semánticas.

Ejemplo 8.1

En el clásico libro *El Lenguaje de Programación C* de Kernighan y Ritchie, el Apéndice A contiene el "Manual de Consulta de C". En el mismo, se explica cómo se usa el lenguaje y qué significa cada una de sus sentencias. Por ejemplo, dice:

"Proposición Condicional:

Las dos formas de la proposición condicional son

- ***if (****expresión****)*** *proposición*
- ***if (****expresión****)*** *proposición* ***else*** *proposición*

En ambos casos, se evalúa la expresión y si es distinta de cero se ejecuta la primera proposición. En el segundo caso, la segunda sentencia solo se ejecuta cuando la expresión es 0. La ambigüedad del "else" se resuelve asociando cada ***else*** *con el último* ***if*** *libre encontrado, como es usual".*

Aquí se explica en lenguaje natural cómo funciona un condicional en el lenguaje C. Véase que se indica que el valor de la expresión cero es considerado como falso y cualquier otro valor distinto de cero, debe ser considerado verdadero. Además, se indica la forma en que se debe resolver la ambigüedad del **else**, ya que con la especificación dada, una sentencia como:

if (a > 0) if (a < 0) b=2; else b=3;

tiene dos posibles árboles de análisis sintáctico, uno asociando el **else** al primer **if** y el otro asociándolo al segundo. El manual indica que el segundo caso es el correcto, pero esto no está explícito en la regla sintáctica, por lo que deberá ser un procedimiento semántico el que busque el anterior **if** al **else** y efectúe la asociación.

Nótese que nada se dice acerca de la implementación efectiva de esta instrucción de control, por lo cual los constructores de compiladores podrán hacer diversos diseños y tomar sus decisiones al respecto.

Lo expuesto hace que la especificación semántica natural resulte atractiva por ser en principio inteligible, pero la experiencia demuestra que describir por este medio todas las características de un lenguaje de programación de modo preciso es una tarea ardua, muy compleja, difícilmente estandarizable y sensible a omisiones y errores.

Por su parte, las especificaciones formales de la semántica de los lenguajes de programación se efectúan sobre la base de una formulación matemática rigurosa del significado o comportamiento esperado de programas. Numerosos autores, tales como Nielson (1992) y Labra (2003), han puntualizado los diversos casos en que se pone de manifiesto y se justifica la necesidad de hacer especificaciones formales de semántica, entre los que se destacan los siguientes:

a) Diseño de nuevos lenguajes de programación, registrando decisiones sobre construcciones particulares y facilitando la identificación de errores u omisiones.

b) Necesidad de estandarización de los lenguajes mediante la publicación de su semántica de un modo universalmente reconocible.

c) Diseño e implementación de los procesadores de los nuevos lenguajes (compiladores o intérpretes) y de los generadores de compiladores.

d) Necesidad de identificar posibles ambigüedades en las implementaciones de procesadores de lenguajes de programación o en sus documentos descriptivos.

e) Comprensión de los lenguajes por parte de los programadores.

f) Verificación de propiedades de programas a través de pruebas de corrección (verificación) o de la información relacionada con su ejecución.

Debe notarse que el ordenamiento cronológico de los casos presentados está justificado: el diseño de nuevos lenguajes (a), da lugar a la necesidad de su estandarización (b), deben implementarse sus procesadores (c), es esencial asegurar la ausencia de ambigüedades (d), los

lenguajes deben ser difundidos entre sus destinatarios (e), que son los programadores, y finalmente los programas resultantes, que representan el producto esperado, deben poder ser verificados (f).

La enorme importancia práctica del último caso citado, la verificación de programas (f), merece dejar de lado por el momento a la semántica y poner el foco en este tema. Modernamente, está fuera de toda discusión la conveniencia de poder realizar pruebas efectivas de corrección a partir de la inspección del código de los programas, lo que es definido como *verificación estática*. Para ello, el compilador y sus herramientas complementarias desempeñan un papel central. Con este fin, se ha realizado un importante esfuerzo de investigación y se lo continúa haciendo.

La contrapartida es la *verificación dinámica*, que implica la realización de pruebas de corrección en tiempo de ejecución. Aquí es necesario puntualizar que la *verificación dinámica* permite identificar la presencia de errores, pero nunca asegurar la ausencia de ellos. Así, al considerarse el alto valor práctico que indudablemente tiene la *verificación dinámica*, debe tenerse en cuenta que solo es posible asegurar la corrección de los programas a partir de pruebas sobre todas las entradas y condiciones de operación posibles, lo que puede dar lugar a un enorme esfuerzo que en la práctica es inviable.

Para superar esta dificultad, el esfuerzo se viene orientando a obtener suficiente conocimiento sobre el comportamiento de los programas a partir del examen anticipado de sus códigos y no de los resultados de sus ejecuciones. Y se recurre a la *verificación estática* para probar anticipadamente su corrección. Aquí es también muy importante revisar los conceptos de verificación y validación, muchas veces confundidos. Mientras que la *verificación* se ocupa de la corrección de los programas, la *validación* tiene por finalidad la comprobación que éstos satisfacen a sus especificaciones.

Retomando ahora el tema del análisis semántico, es necesario reconocer que así como para la especificación formal de la sintaxis de un lenguaje se dispone de la notación BNF (*Backus-Naur Form*), que permite hacer una descripción normalizada de su gramática, para la especificación semántica no hay ningún método estándar de aplicación general y universalmente aceptado. Es así que la descripción del comportamiento de las distintas construcciones de un lenguaje de programación puede ser abordada desde distintos puntos de vista, distinguiéndose dos grandes enfoques:

1. Los que utilizan metalenguajes de especificación semántica.
2. Aquellos que usan gramáticas atribuidas o gramáticas con atributos.

En los primeros, se distinguen a su vez las propuestas que están orientadas a definir modelos de aquellas otras que tienen por finalidad definir propiedades. Lo expuesto puede resumirse en el esquema presentado a continuación en la Figura 8.1.

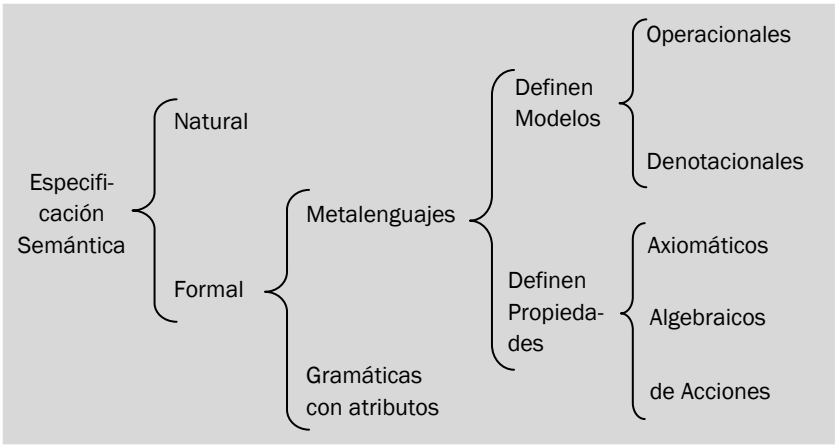

Figura 8.1: Distintos enfoques de la especificación semántica de lenguajes.

Metalenguajes para la especificación semántica de lenguajes

El tratamiento de los metalenguajes de especificación semántica está fuera del alcance de este libro; tanto por el espacio que requeriría hacerlo como también por el objetivo de este capítulo,

que es introductorio. Hay unanimidad en reconocer que la complejidad del tratamiento de la semántica de lenguajes es, al menos, un orden de magnitud mayor que el de la sintaxis, lo que hace imposible tratar estos temas de manera breve e introductoria, a la vez que lograr una presentación inteligible. Por lo expuesto, se optó por presentar una descripción general de estas ideas sin entrar en su esencia, temas sobre los cuales hay excelente bibliografía especializada que se recomienda consultar.

Semántica operacional

Se trata del enfoque semántico más antiguo y es muy utilizado por ser intuitivo, fácilmente aplicable y describir el comportamiento observable. La semántica operacional describe cómo interpretar un programa a través de una secuencia de pasos de cálculo u operaciones. Esto puede hacerse asociando código de máquina (de una máquina real definida o de una máquina abstracta) a cada regla sintáctica, por lo que la descripción es muy cercana a su implementación. También se han propuesto alternativas que utilizan un lenguaje de reglas de inferencia lógicas que representan el significado de cada construcción sintáctica, con lo cual puede usarse la deducción lógica (o una máquina abstracta de reducción que la implemente) para interpretar el significado de un programa. El concepto de semántica operacional se utilizó por primera vez en la definición de la semántica de Algol 68 y posteriormente para construir prototipos de procesadores de lenguajes, como en el caso de la descripción del lenguaje PL/1 (1969).

Semántica denotacional

La principal idea de la semántica denotacional es que el significado de un programa puede describirse en términos de la aplicación de entidades matemáticas (funciones) a sus argumentos (denotación). Así, a través del uso de conjuntos y funciones, se representan la mayoría de los conceptos, tales como los de memoria, variables, tipos, etcétera, alcanzándose un elevado nivel de abstracción. De esta manera, se representan todas las instrucciones del lenguaje con funciones que producen el efecto de haberlas ejecutado. Por tanto, se hace más hincapié en el resultado de la computación que en la forma en la que ésta es llevada a cabo. La semántica denotacional se ha empleado para especificar la semántica completa de lenguajes de programación tales como Ada y Pascal.

Semántica axiomática

La semántica axiomática descansa sobre una teoría formal, donde un conjunto de axiomas, junto con una descripción formal del programa, permiten la deducción del resultado de la ejecución del programa en cualquier ambiente dado, así como la inferencia de propiedades más generales, tales como la corrección, capacidad de finalizar y equivalencia entre programas. El enfoque axiomático, por estar orientado a las necesidades de demostración de propiedades de programas, no constituye un camino natural ni directo a algún método de implementación.

Semántica algebraica

Toda teoría formal descripta por un sistema axiomático (términos primitivos, axiomas, definiciones y teoremas) puede ser también descripta por una estructura algebraica (elementos, conjuntos y funciones definidas sobre ellos), pero con un sabor más matemático por las notaciones empleadas. Este método está enfocado a especificar la semántica de los tipos y de las operaciones entre los mismos. Se basa en la especificación de tipos de datos abstractos mediante un conjunto de valores posibles y operaciones efectuadas sobre ellos y se la reconoce como una forma de semántica axiomática basada en las reglas del álgebra. En algún sentido, se acerca al enfoque denotacional.

Semántica de acciones

En este caso, el tratamiento semántico del lenguaje está basado en el concepto de acciones, que son las soportadas por las operaciones habitualmente previstas en los lenguajes de programación. Con este fin, se definen primitivas para la asignación y declaración de identificadores y la combinación de instrucciones mediante control de flujo secuencial, condicional e iterativo. La finalidad fue describir la semántica de lenguajes de un modo que facilite su entendimiento y en este sentido, se acerca al enfoque operacional.

Gramáticas con atributos

Las *gramáticas con atributos* (GA) o *gramáticas atribuidas* deben su nombre a que se apoyan en la asignación de atributos (propiedades) a las distintas construcciones sintácticas de un lenguaje. Las gramáticas son de tipo 2, *independientes de contexto*, a las que se incorporan atributos a sus símbolos terminales y no terminales, las reglas para su evaluación y las condiciones que éstas deben cumplir. La muy estrecha relación con la gramática hace que el análisis semántico que se desprende de este enfoque sea denominado *dirigido por la sintaxis*. Además, cabe destacar que esta circunstancia motivó una gran difusión de las GA, lo que justifica que sean aquí tratadas con mayor detalle que el utilizado en el caso de los *metalenguajes*. Las gramáticas con atributos fueron originalmente propuestas por Donald Knuth en 1968 para definir las especificaciones semánticas de los lenguajes de programación.

Las GA no están directamente ligadas al comportamiento dinámico de los programas y por lo tanto en muchos casos no son reconocidas como una de las formas de especificar formalmente la semántica de lenguajes. Sin embargo, por ser muy versátiles, de un modo directo o indirecto están siempre presentes en cualquier implementación de procesadores de lenguajes formales.

Definición de GA

Una gramática con atributos es una gramática de contexto libre a la que se le ha incorporado un conjunto de atributos (***A***), un conjunto de reglas semánticas (***R***) y un conjunto (***C***) de condiciones, donde los elementos de ***A*** están relacionados con los símbolos terminales y no terminales, mientras que los de ***R*** y ***C*** están asociados a cada una de las reglas de producción de ***P***. Por lo expuesto, una GA es una séptupla que queda definida:

$$\boldsymbol{GA} = (\boldsymbol{\Sigma_T}, \boldsymbol{\Sigma_N}, \boldsymbol{S}, \boldsymbol{P}, \boldsymbol{A}, \boldsymbol{R}, \boldsymbol{C})$$

y cuyas componentes tienen las siguientes características:

a) Los primeros cuatro componentes son los que corresponden a toda gramática formal, a saber: *i*) el alfabeto de símbolos terminales ($\boldsymbol{\Sigma_T}$), *ii*) el alfabeto de símbolos no terminales ($\boldsymbol{\Sigma_N}$), *iii*) el axioma (***S***) y *iv*) el conjunto de reglas de producción (***P***). Naturalmente, las reglas de producción deben responder a las condiciones que son propias de una gramática independiente de contexto y pueden estar expresadas en formas normales, en las que las de Greibach (FNG) y de Chomsky (FNC) son las más habituales.

b) Los atributos son variables que representan determinadas propiedades de los símbolos terminales y no terminales. Denominando $\boldsymbol{\Sigma=\Sigma_T\cup\Sigma_N}$, el quinto componente de la GA es identificado como ***A*(Σ),** representando al conjunto de atributos asociados a los símbolos del alfabeto Σ. Cada atributo se denota con un nombre precedido por un punto y el nombre del símbolo al que está asociado. Por ejemplo, si $\boldsymbol{a\in\Sigma}$ puede haber atributos tales como **a.*tipo*** o **a.*valor***. Además, cada atributo puede tomar un valor cualquiera, que describe propiedades dentro de un rango de valores posibles.

c) A cada regla de producción del conjunto ***P*** (regla sintáctica BNF) se le asocia un número finito de acciones o reglas semánticas que especifican la forma en que se modifican los atributos con la aplicación de la regla sintáctica. Al conjunto de todas las reglas semánticas se lo define como ***R*(*P*)**, que es el sexto componente de la GA.

d) El séptimo y último componente de la GA es un conjunto ***C*(*P*)** de condiciones asociado a cada una de las reglas de producción ***P***. Estas condiciones, que deben ser cumplidas por los valores de los atributos, describen un sublenguaje del lenguaje definido por la sintaxis de la gramática. Nótese que una sentencia sintácticamente correcta también lo será semánticamente, si y solo si todos los atributos satisfacen las condiciones ***C*** del contexto.

Atributos

Como fue anticipado, los atributos representan propiedades de los símbolos terminales y no terminales de una gramática y su naturaleza varía según la información que contienen y la fase de procesamiento en la que se encuentren. También dependen del momento en el que los atributos

son calculados: si lo son previo a la ejecución de la aplicación son considerados estáticos y si sus valores se obtiene en tiempo de ejecución son reconocidos como dinámicos.

Algunos de los ejemplos típicos de atributos en una gramática son: *i*) el tipo de una variable, *ii*) su valor, *iii*) su dirección asignada de memoria, *iv*) el número de argumentos de una función, *v*) los tipos de estos argumentos, *vi*) el identificador de clase, *vii*) los punteros a entradas, etcétera.

Al operarse los símbolos del alfabeto Σ a través de las reglas de producción, las propiedades representadas por los atributos deben ser evaluadas, es decir, asignadas, confirmadas o alteradas, lo que representa el principal problema de las gramáticas atribuidas. Dependiendo de las características que presenten las gramáticas se dispone para esta evaluación de dos mecanismos, que son la herencia y la síntesis. Las características de ambas son presentadas a continuación.

Aquí cabe acotar que cuando es posible evaluar todos los atributos de los símbolos de todas las sentencias de una gramática, se dice que es una gramática *bien definida* o *no circular.* Esto implica poder reconocer el orden en el que han de ser ejecutadas las reglas o acciones semánticas, para todas las sentencias, con el fin de poder asignar correctamente los valores a los atributos de todos sus símbolos. Resulta necesario destacar que el matemático M. Jazayeri (1975) propuso un algoritmo que permite comprobar anticipadamente si una gramática está *bien definida*, pero este algoritmo es de aplicación limitada por tener el inconveniente de que su complejidad temporal es exponencial. Es decir, las gramáticas complejas pueden conducir a considerarlo un problema intratable.

Atributos sintetizados

Conviene aquí tomar como referencia el árbol de derivación sintáctica que representa el proceso de definición de las sentencias de toda gramática. En el caso de los atributos sintetizados, las evaluaciones en cada nodo se realizan a partir de los atributos de sus nodos hijos u hojas del árbol. Esto significa que se avanza en el árbol desde abajo hacia arriba, hasta alcanzar el axioma, lo que corresponde a un proceso de reducción de una sentencia. En otras palabras, los atributos del antecedente de cada regla de producción de la gramática son calculados exclusivamente a partir de los atributos de los símbolos gramaticales del consecuente o parte derecha. Al conjunto de atributos sintetizados, se lo denomina $\boldsymbol{AS(\Sigma)}$. Para comenzar, es necesario disponer de los valores de los atributos de los símbolos terminales, que representan las hojas del árbol, los que son suministrados por el analizador léxico.

Cuando todos los atributos asociados con los símbolos gramaticales son sintetizados se dice que se trata de una *gramática S-Atribuida*, y aquí son muy eficaces los analizadores ascendentes ya que permiten evaluar los atributos al mismo tiempo que se analiza la sentencia.

Considérese como ejemplo la regla de producción **Z := bDe** en la que los atributos de los símbolos **b**, **D** y **e** permitirán sintetizar los atributos del símbolo no terminal **Z**.

En resumen: *i*) resulta conveniente evaluar los atributos sintetizados con un analizador sintáctico ascendente conforme la entrada es analizada, *ii*) el analizador sintáctico puede conservar en su pila los valores de los atributos sintetizados asociados a los símbolos gramaticales y *iii*) en cada reducción se calculan los valores de los nuevos atributos sintetizados a partir de los atributos de la pila para los símbolos gramaticales del lado derecho de la producción.

Atributos heredados

Los atributos en una derivación directa son heredados, si los mismos correspondientes a los símbolos del consecuente de una regla de producción son evaluados a partir de los atributos del símbolo del antecedente o de los atributos de los demás símbolos del consecuente, siempre que esos símbolos aparezcan a la izquierda en la producción. Esto significa que, en el árbol de derivación sintáctica, los atributos son heredados si se calculan a partir de los atributos del padre de cada nodo o de los nodos hermanos, donde el orden de evaluación de los atributos se corresponde con el orden en el que se “crean” los nodos de un árbol de análisis sintáctico.

Se brinda como ejemplo el caso de la producción $\mathbf{Z := b_1 b_2 b_3 .. b_n}$, donde los atributos de cada símbolo $\mathbf{b_k}$ se heredan del símbolo no terminal **Z** o de los símbolos $\mathbf{b_1, b_2, ..., b_{k-1}}$. A su vez, los atributos de estos últimos pueden haberse obtenido por herencia o síntesis.

Al conjunto de atributos heredados, se los denomina $\boldsymbol{AH(\Sigma)}$ y las gramáticas que contienen atributos tanto heredados como sintetizados son denominas *L-atribuidas.* De acuerdo a esta definición, las gramáticas *S-atribuidas* son un subconjunto de las *L-atribuidas*. En efecto, toda gramática *S-atribuida* que no contenga atributos heredados es *L-atribuida.* Es importante destacar que el ya citado Donald E. Knuth demostró que toda gramática *L-atribuida* podía ser convertida a otra gramática *S-atribuida* que reconozca el mismo lenguaje, para lo cual definió un esquema de traducción de atributos heredados a atributos sintetizados.

Acciones semánticas

Dependiendo del tipo de instrucción, las acciones semánticas que están incluidas en el conjunto de reglas $\boldsymbol{R(P)}$ tienen distintas finalidades y pueden agruparse según se presenta en la Tabla 8.1.

TIPO DE INSTRUCCIÓN	ACCIONES SEMÁNTICAS
Declaración	obtención de tipos de la tabla de símbolos
Ejecutables	completar la definición de tipos de los símbolos
Funciones y procedimientos	comprobación del número, orden y tipo de argumentos
Identificación de variables	comprobación de identificación previo a su uso
Etiquetas	comprobación de repetición y validación
Constantes	comprobación de intentos de alteración de valores
Conversiones	verificación de equivalencias
Sobrecarga de operadores	identificación y resolución

Tabla 8.1: Acciones semánticas agrupadas por tipo de instrucción.

Ejemplo 8.2

A título de ejemplo, se citan las siguientes acciones semánticas que corresponden a sentencias ejecutables, resumidas en la Tabla 8.2:

Tabla 8.2: Acciones semánticas asociadas a ciertas reglas de producción.

REGLA SINTÁCTICA BNF	ACCIÓN SEMÁNTICA
A := B + C	A.valor = B.valor + C.valor
	A.tipo = mayor_tipo(B.tipo, C.tipo)
A := B * C	A.valor = B.valor * C.valor
	A.tipo = mayor_tipo(B.tipo, C.tipo)

Condiciones

El conjunto de condiciones que es definido sobre las producciones $\boldsymbol{C(P)}$ regulan la validez y alcance de las acciones semánticas sobre los atributos, estableciendo límites claros a la aplicación de las acciones según sea la naturaleza de la regla sintáctica. Estas condiciones están activas tanto en tiempo de compilación como de ejecución, ya que muchas de ellas pueden verificarse correctamente en forma estática y no cumplirse dinámicamente por alteración de los

valores asociados a los símbolos. El conjunto de estas condiciones describen un sub-lenguaje del lenguaje definido por la sintaxis de la gramática, donde los atributos de una sentencia sintácticamente correcta deben satisfacer todas las condiciones ***C*** que le correspondan, para ser considerada semánticamente válida.

Ejemplo 8.3

En la Tabla 8.3, se presentan como ejemplo las condiciones que corresponden a los operadores de división para el caso del lenguaje Pascal, que como se sabe son diferentes según se trate de una división de enteros o de números reales:

REGLA SINTÁCTICA BNF	CONDICIÓN SEMÁNTICA
A := B / C	C.valor > 0
	A.tipo, B.tipo = real, C.tipo = real o entero
A := B div C	C.valor > 0
	A.tipo, B.tipo, C.tipo = entero

Tabla 8.3: Condiciones semánticas asociadas a las producciones de división.

Presentados ya los componentes de las gramáticas con atributos, veremos en el Ejemplo 8.4 un fragmento de una gramática de algún lenguaje de programación, correspondiente a la definición de la sintaxis y la semántica de una asignación que nos permitirá mostrar cómo se puede utilizar una GA para la determinación de algunos atributos simples.

Ejemplo 8.4

Se propone una pequeña gramática libre de contexto para definir una asignación muy simple. El alfabeto de símbolos terminales de esta gramática lo componen el operador de asignación (=), los símbolos de operaciones aritméticas de adición (+) y multiplicación (*) y los tokens *variable* y *número*. En estos últimos dos casos, se supondrá que los valores de sus atributos han sido determinados previamente por el análisis léxico, por algún anterior proceso sintáctico o en tiempo de ejecución, ya que en el lenguaje descripto el tipo de las variables es determinado en las asignaciones (no requieren declaraciones previas).

Las reglas sintácticas de este fragmento son las siguientes:

(1) <ASIGNACION> := variable = <EXPRESION>

(2) <EXPRESION> := <EXPRESION> <OPERADOR> <EXPRESION>

(3) <EXPRESION> := variable

(4) <EXPRESION> := número

(5) <OPERADOR> := +

(6) <OPERADOR> := **

Claramente, los símbolos no terminales son los las palabras encerradas entre corchetes angulares y tomaremos como axioma de este fragmento al no terminal <ASIGNACION>.

Además, la gramática es complementada con los siguientes atributos:

número.tipo	Tipo del número (entero o real)
número.valor	Valor que representa el número
variable.tipo	Tipo de la variable
variable.valor	Valor que toma la variable
variable.nombre	Nombre que toma la variable
<EXPRESION>.tipo	Tipo de la expresión
<EXPRESION>.valor	Valor después de evaluar la expresión

<OPERADOR>.tipo Tipo del operador (entero o real)

<OPERADOR>.clase Clase del operador (adición o multiplicación)

A partir de aquí, los procesos de derivación de sentencias son representados con árboles sintácticos que deben incluir los atributos. Los valores de atributos de número se suponen determinados por el analizador léxico y el de las variables que se usen en el lado derecho de las asignaciones, ya estipulados por anteriores pasos del análisis sintáctico (recuerde que solo estamos analizando un fragmento de la gramática del lenguaje).

Para ilustrar el funcionamiento, se toma el caso de una expresión que representa el tiempo transcurrido desde algún evento pasado, medido en horas y se muestra en la Figura 8.2 su árbol de análisis sintáctico con información semántica asociada.

tiempoEnHoras = 24 * dias + horaActual,

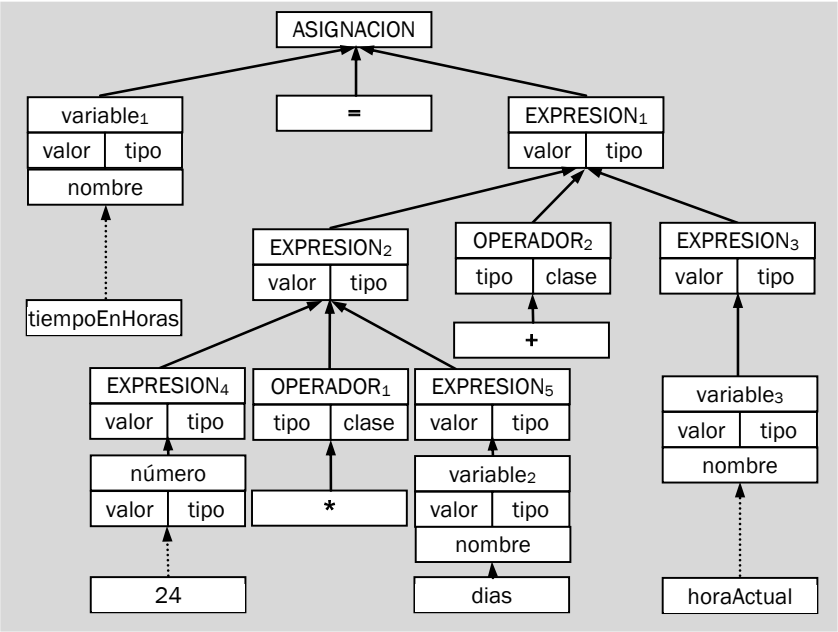

Figura 8.2: Árbol de derivación sintáctica con atributos.

Una vez definido el conjunto de los atributos, la gramática debe ser ampliada con las acciones o reglas semánticas que especifican la manera en la que se modifican los atributos, quedando expresada como se muestra a continuación:

(1) <ASIGNACION> := variable = <EXPRESION>

```
  { variable.valor = <EXPRESION>.valor
    variable.tipo  = <EXPRESION>.tipo
  }
```

(2) <EXPRESION>$_1$:= <EXPRESION>$_2$ <OPERADOR> <EXPRESION>$_3$

```
  { <OPERADOR>.tipo = Mayor_Tipo(<EXPRESION>2.tipo,
                    <EXPRESION>3.tipo)
    <EXPRESION>1.tipo = <OPERADOR>.tipo
    if (<OPERADOR>.tipo=='F' && <EXPRESION>2.tipo=='I') {
          <EXPRESION>2.tipo  = 'F';
          <EXPRESION>2.valor = FLOAT(<EXPRESION>2.valor);
      }
    if (<OPERADOR>.tipo=='F' && <EXPRESION>3.tipo=='I') {
          <EXPRESION>3.tipo  = 'F';
          <EXPRESION>3.valor = FLOAT(<EXPRESION>3.valor);
      }
    switch (<OPERADOR>.tipo) {
      'I': <EXPRESION>1.valor = op_entera(<OPERADOR>.clase,
          <EXPRESION>2.valor, <EXPRESION>3.valor); break;
```

```
      'F': <EXPRESION>1.valor = op_real(<OPERADOR>.clase,
           <EXPRESION>2.valor, <EXPRESION>3.valor); break;
    }
  }
(3) <EXPRESION> := variable
  { <EXPRESION>.valor = variable.valor
    <EXPRESION>.tipo  = variable.tipo
  }
(4) <EXPRESION> := número
  { <EXPRESION>.valor = número.valor
    <EXPRESION>.tipo  = número.tipo
  }
(5) <OPERADOR> := +    { <OPERADOR>.clase = '+' }
(6) <OPERADOR> := *    { <OPERADOR>.clase = '*' }
```

En las reglas semánticas propuestas, se utilizó la función de librería FLOAT (que transforma números enteros en números reales) y funciones propias auxiliares, que son definidas a continuación usando sintaxis de lenguaje C:

```
char Mayor_Tipo(char tipo1, char tipo2)
{ if (tipo1 == 'F' || tipo2 == 'F') return 'F';
  else                              return 'I';
}
int op_entera(char op, int valor1, int valor2)
{
  switch (op) {
    '+': return (valor1 + valor2);
    '*': return (valor1 * valor2);
  }
}
float op_real(char op, float valor1, float valor2)
{
  switch (op) {
    '+': return (valor1 + valor2);
    '*': return (valor1 * valor2);
  }
}
```

Considerando el proceso de compilación de las asignaciones que ejemplificamos, el analizador léxico devuelve como se indicó, los *tokens* que son los terminales de la gramática y en el caso de número, el valor de sus atributos *tipo* y *valor*.

El analizador sintáctico verifica que la frase en la que intervienen esos *tokens* recibidos, esté correctamente escrita de acuerdo a la gramática, obteniendo un árbol derivación sintáctico en cuyas hojas se lee:

variable$_1$ = numero * variable$_2$ + variable$_3$

Este árbol es tomado por el analizador semántico que determinará, para cada símbolo, el valor de sus atributos. A partir de esta información, los atributos se propagan según las reglas y acciones semánticas definidas con anterioridad, obteniéndose progresivamente los valores según se muestra en la Figura 8.3.

Nótese que el atributo valor de las variables es desconocido durante la compilación. Las variables tendrán el valor conocido recién en tiempo de ejecución y el árbol de la Figura 8.2 muestra cómo se llevará a cabo el cálculo de los atributos de la asignación.

```
numero.tipo          = 'I'   (del análisis léxico)
numero.valor         = 24    (del análisis léxico)
variable2.tipo       = determinado en tiempo de ejecución.
                       Sea 'F'.
variable2.nombre     = días  (del análisis léxico)
variable2.valor      = determinado en tiempo de ejecución.
                       Sea m.
variable3.tipo       = determinado en tiempo de ejecución.
                       Sea 'F'.
variable3.nombre     = horaActual (del análisis léxico)
variable3.valor      = determinado en tiempo de ejecución.
                       Sea n.
<EXPRESION>4.tipo    = 'I'    (según semántica de la regla 4)
<EXPRESION>4.valor   = 24     (según semántica de la regla 4)
<OPERADOR>1.clase    = '*'    (según semántica de la regla 6)
<EXPRESION>5.tipo    = 'F'    (según semántica de la regla 3)
<EXPRESION>5.valor   = m      (según semántica de la regla 3)
<OPERADOR>1.tipo     = 'F'    (según semántica de la regla 2)
<EXPRESION>2.tipo    = 'F'    (según semántica de la regla 2)
<EXPRESION>2.valor   = 24*m   (según semántica de la regla 2)
<OPERADOR>2.clase    = '+'    (según semántica de la regla 5)
<EXPRESION>3.tipo    = 'F'    (según semántica de la regla 3)
<EXPRESION>3.valor   = n      (según semántica de la regla 3)
<OPERADOR>2.tipo     = 'F'    (según semántica de la regla 2)
<EXPRESION>1.tipo    = 'F'    (según semántica de la regla 2)
<EXPRESION>1.valor   = 24*m+n (según semántica de la regla 2)
variable1.tipo       = 'F'    (según semántica de la regla 1)
variable1.nombre     = tiempoEnHoras (del análisis léxico)
variable1.valor      = 24*m+n (según semántica de la regla 2)
```

Figura 8.3: Asignación de atributos y propagación.