3.4 BNF Y ALGOL

La sigla BNF corresponde a la frase "Backus Normal Form" (Forma Normal de Backus) o también a la frase "Backus-Naur Form" (Forma de Backus y Naur). Con el breve Manual de Referencia del lenguaje ALGOL (*ALGOrithmic Language*) se publicó, por primera vez en 1960, una descripción formal de la sintaxis de un LP. Esta descripción, similar a las producciones de las GICs, se llamaría luego BNF.

La notación BNF consiste en un conjunto de REGLAS que definen, con precisión, la sintaxis de los componentes y de las estructuras de un LP dado, como hemos hecho en secciones anteriores, pero utilizando GICs. Pero, como veremos en el resto de este capítulo, BNF ha evolucionado incorporando nuevos metasímbolos; en esto radica una diferencia fundamental entre BNF y una Gramática Formal.

Volviendo al origen de BNF, la diferencia entre la notación utilizada para describir las producciones de una GIC y la usada para representar las reglas BNF era mínima: el metasímbolo "operador de producción" comenzó a representarse como `::=` (al que podríamos llamar "operador ES") y los nombres de los noterminales fueron encerrados entre corchetes angulares.

Ejemplo 11 Algunos elementos básicos del lenguaje ALGOL pueden ser enumerados de esta forma:
`<símbolo básico> ::= <letra> | <dígito> | <valor lógico> | <delimitador>`
Estas cuatro reglas BNF se pueden leer de esta forma: "un símbolo básico es una letra o un dígito o un valor lógico o un delimitador".

Cada regla BNF se forma con elementos provenientes de tres conjuntos disjuntos:

– ***metavariables*** o ***noterminales***, que son palabras o frases encerradas entre corchetes angulares (ejemplo: <símbolo básico>, <identificador>).

– ***terminales***, que son los caracteres del alfabeto o palabras del lenguaje sobre los cuales se construye el LP descripto. Se los llama terminales porque no existen reglas de producción para ellos (ejemplo: una palabra reservada es un terminal).

– ***metasímbolos***, que son caracteres o grupos de caracteres que ayudan a representar estas reglas (ejemplo: <>, ::=, |).

En toda regla BNF, encontramos, al igual que en toda producción de una GIC, tres componentes:
(1) el lado izquierdo, formado por un solo noterminal
(2) el lado derecho, formado por terminales y noterminales o, eventualmente, vacío
(3) un metasímbolo, el operador es, que vincula el lado izquierdo con el lado derecho.

Ejemplo 12
`<símbolo básico> ::= <letra>` que se lee: "un Símbolo Básico es una Letra".

* Ejercicio 17 *
Si el lado derecho de una regla BNF es vacío, ¿qué significa? Justifique su respuesta.

Los tres metasímbolos que aparecen en las reglas del Ejemplo 11 y en la BNF general del lenguaje ALGOL son:
(1) **<** y **>** (para encerrar una palabra o frase que es el nombre de un noterminal);

(2) **::=** (que se lee "es" o "corresponde a"), y

(3) **|** (que se lee "ó").

Estos son los únicos metasímbolos que formaban parte de la definición original de BNF.

3.4.1 DOS EJEMPLOS DE BNF EN ALGOL

En ALGOL 60, un identificador debía comenzar con una letra y, si tenía más caracteres, éstos podían ser letras y dígitos decimales. La categoría léxica "Identificador en ALGOL 60" constituye un LR infinito.

Ejemplo 13
Esta es la definición BNF de un identificador en ALGOL 60:

```
<identificador> ::= <letra> |
<identificador> <letra> |
<identificador> <dígito>
<letra> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
            p | q | r | s | t | u | v | w | x | y | z | A | B | C | D |
            E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
            T | U | V | W | X | Y | Z
<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Análisis:
– El conjunto de tres reglas para <identificador>, con dos reglas recursivas a izquierda, equivale a una GQR.
– El conjunto de reglas para <letra> se lee: "*letra* es cualquier letra minúscula o mayúscula del alfabeto inglés".
– El conjunto de reglas para <dígito> se lee: "*dígito* es cualquier dígito decimal (base 10)".

Nota 3
<letra> y <dígito> definen dos LRs finitos, mientras que <identificador> define un LR infinito.

En cuanto al LR infinito de los números enteros, veamos cómo se define en ALGOL.

Ejemplo 14

```
<número entero> ::= <entero sin signo> |
                    + <entero sin signo> |
                    - <entero sin signo>
<entero sin signo> ::= <dígito> | <entero sin signo> <dígito>
<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Análisis:
– Las tres reglas de <número entero> definen al número entero sin signo y con signo.
– El noterminal <entero sin signo>, por medio de una regla recursiva a izquierda y otra de terminación de la recursividad, define una sucesión de uno o más dígitos decimales. En consecuencia, un *número entero* es: (a) una sucesión de dígitos, o (b) una sucesión de dígitos precedida por el signo positivo, o (c) una sucesión de dígitos precedida por el signo negativo.

* Ejercicio 18 *
Derive el número entero `-123456`. Si desea, puede abreviar el nombre de los noterminales explicando qué significa cada abreviatura utilizada.

A continuación veremos cómo fue descripto el lenguaje Pascal por su creador, Niklaus Wirth, y la primera ampliación del conjunto de metasímbolos en BNF.

3.5 BNF Y EL LENGUAJE PASCAL

Independientemente de la notación que utilicemos y del LP descripto, las BNFs están formadas por:

1) un conjunto de NOTERMINALES
2) un conjunto de TERMINALES
3) un conjunto de METASÍMBOLOS
4) un conjunto de PRODUCCIONES o REGLAS

Seguidamente, veremos una extensión que realizó Niklaus Wirth a la notación BNF utilizada para describir la sintaxis del ALGOL. Por ello, a partir de esta extensión y de otras que analizaremos luego, la sigla BNF fue cambiada por la de EBNF (BNF extendida). No obstante, muchos autores siguen utilizando la sigla BNF también para las BNFs extendidas; nosotros haremos lo mismo.

3.5.1 LA SINTAXIS DEL LENGUAJE PASCAL, SEGÚN WIRTH

En el libro "Pascal – User Manual and Report" (1974), escrito por Niklaus Wirth –creador de este lenguaje– y Kathleen Jensen, se describen con precisión la sintaxis y la semántica del lenguaje Pascal. En esta sección nos ocuparemos de su descripción sintáctica.

En la Introducción del libro mencionado, Wirth dice: "Una formulación de la sintaxis es la tradicional BNF, en la que los *constructos sintácticos* [categorías léxicas y categorías sintácticas]

son denotados por palabras en inglés [en castellano, en nuestro caso] encerradas entre los corchetes angulares **<** y **>**."

Por otro lado, al conjunto de metasímbolos ya existente en ALGOL, Wirth agrega un metasímbolo muy importante y lo describe así: "Las llaves **{** y **}** denotan la posible repetición de los símbolos que encierra, cero o más veces." Es una forma más compacta de describir una secuencia de elementos, como veremos en próximos ejemplos.

Este metasímbolo es la primera extensión que se realizó sobre la BNF utilizada en ALGOL: a los tres metasímbolos ya existentes en ALGOL, Wirth agregó el metasímbolo indicado por un par de llaves para representar lo que conocemos como el operador *clausura de Kleene*, es decir: cero o más veces lo que está encerrado entre las llaves.

A partir de ésta y de otras extensiones posteriores es que algunos autores utilizan la sigla EBNF para referirse a una BNF extendida. Pero, como ya se mencionó, nosotros seguiremos utilizando la sigla BNF, independientemente de las extensiones que se agreguen y de los caracteres utilizados para representar a los metasímbolos.

Veamos, a continuación, algunos ejemplos de la BNF que utiliza Wirth para describir la sintaxis del lenguaje Pascal en su Manual de Referencia original.

Ejemplo 15
La definición de identificador en el Pascal original y con esta BNF extendida es:

```
<identificador> ::= <letra> { <letra o dígito> }
<letra o dígito> ::= <letra> | <dígito>
<letra> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
            p | q | r | s | t | u | v | w | x | y | z | A | B | C | D |
            E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S |
            T | U | V | W | X | Y | Z
<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

* Ejercicio 19 *
(a) Interprete, mediante una frase, la regla <identificador>.
(b) Convierta la regla <identificador> a la notación utilizada en ALGOL.
(c) ¿Cuál notación le parece más conveniente?

Ejemplo 16
La definición de la constante real sin signo en el Pascal original y con esta BNF extendida es la siguiente:

```
<real sin signo> ::=
           <entero sin signo> . <entero sin signo> |
           <entero sin signo> . <entero sin signo> E <factor de escala> |
           <entero sin signo> E <factor de escala>
<entero sin signo> ::= <dígito> { <dígito> }
<factor de escala> ::= <entero sin signo> | <signo> <entero sin signo>
<signo> ::= + | -
<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Análisis:
Como se puede apreciar, estas producciones definen con precisión tanto a las constantes reales en punto fijo como a las constantes reales en punto flotante; para éstas últimas utiliza la letra E. Como se puede apreciar también, el Pascal original tiene una única forma de escribir las constantes reales en punto fijo y tiene dos formas de escribir las constantes reales en punto flotante.

* Ejercicio 20 *
(a) Escriba el conjunto de terminales utilizados en la BNF del Ejemplo 16.
(b) Escriba el conjunto de metasímbolos utilizados en la BNF del Ejemplo 16.
(c) Escriba ejemplos de todos los casos de <real sin signo> descriptos por la BNF del Ejemplo 16.

Ejemplo 17
El constructo <programa Pascal> es definido por Wirth de esta manera:

```
<programa Pascal> ::= <encabezamiento> <bloque> .
<encabezamiento> ::= program <identificador> ( <identificador-archivo>
                     {, <identificador-archivo>} ) ;
<identificador-archivo> ::= <identificador>
<bloque> ::= <sección definición de constantes>
             <sección definición de tipos>
             <sección declaración de variables>
             <sección declaración de procedimientos y funciones>
             <sección sentencias>
<sección definición de constantes> ::=
           <vacío> |
           const <definición de constante> { ; <definición deconstante> }
<definición de constante> ::= <identificador> = <constante>
<vacío> ::=
<sección declaración de variables> ::=
           <vacío> |
           var <declaración de variable> { ; <declaración de variable> }
<declaración de variable> ::=
           <identificador> { , <identificador> } : <tipo>
<sección sentencias> ::= <sentencia compuesta>
<sentencia compuesta> ::= begin <sentencia> { ; <sentencia> } end
<sentencia> ::= <sentencia simple> | <sentencia estructurada>
<sentencia simple> ::= <sentencia de asignación> |
                       <sentencia de procedimiento> |
                       <sentencia vacía>
<sentencia de asignación> ::= <variable> := <expresión>
<sentencia de procedimiento> ::= <identificador de procedimiento> | . . .
<sentencia vacía> ::= <vacío>
<sentencia estructurada> ::= . . .
```

* Ejercicio 21 *
Preguntas referidas a la BNF descripta en el Ejemplo 17:
(a) ¿Cómo se representan las palabras reservadas en esta BNF?
(b) ¿Cuántos elementos forman el conjunto de terminales y cuáles son?
(c) ¿Qué significa <vacío>?
(d) ¿Cuántas reglas tiene el noterminal <bloque>?

(continúa en la página siguiente)

(e) De acuerdo a esta BNF, ¿existe en Pascal la sentencia compuesta vacía de acciones (sentencias)? Justifique su respuesta.
(f) ¿Cuántas variables se pueden declarar para un <tipo> determinado?

* Ejercicio 22 *
Escriba dos ejemplos de <sentencia compuesta>.

3.5.2 EXPRESIONES EN PASCAL

A continuación definiremos un subconjunto de las expresiones en Pascal. Analizaremos las prioridades de los operadores en este LP y también veremos cómo la BNF no se ocupa de los tipos de los datos.

```
<expresión> ::= <expresiónSimple> |
                <expresiónSimple> <operadorRelacional> <expresiónSimple>
<operadorRelacional> ::= = | <> | < | <= | > | >=
<expresiónSimple> ::= <término> |
                      <signo> <término> |
                      <expresiónSimple> <operadorAditivo> <término>
<operadorAditivo> ::= + | - | or
<término> ::= <factor> | <término> <operadorMultiplicativo> <factor>
<operadorMultiplicativo> ::= * | / | and
<factor> ::= <variable> | <número> | ( <expresión> ) | not <factor>
. . .
```

Precedencia de los operadores
En el libro de Wirth se lee: "El operador not (aplicado a un operando Booleano) tiene la máxima prioridad. Le siguen los operadores multiplicativos, luego los operadores aditivos y los de mínima prioridad son los operadores relacionales."

Como ya hemos dicho, observe que, en la BNF, los operadores van apareciendo en orden inverso a sus prioridades: los operadores relacionales son los que están más cerca del axioma (expresión, en este caso); a continuación aparecen los operadores aditivos, luego los operadores multiplicativos y, finalmente, el operador not (el de máxima prioridad).

Una curiosidad a destacar es que operadores aritméticos y cierto operador lógico aparecen en el mismo nivel de prioridades, como se observa en dos de las reglas BNF.

* Ejercicio 23 *
Escriba los operadores aritméticos y el operador lógico que aparecen en el mismo nivel.

Otra curiosidad del Lenguaje de Programación Pascal surge en su comparación con el lenguaje matemático. Si en matemáticas escribimos la expresión lógica a > 24 y b < 5, no necesitamos utilizar paréntesis porque los operadores relacionales tienen mayor prioridad que los operadores lógicos o Booleanos. En cambio, vemos que en Pascal los operadores relacionales tienen la mínima prioridad; por lo tanto, los paréntesis serán imprescindibles.

Veamos cómo sería el proceso de derivación a izquierda para esa misma expresión lógica en Pascal; se abrevian los nombres de los noterminales para mayor simplicidad:

Ejemplo 18

```
<E>
<ES>
<T>
<T> <OM> <F>
<F> <OM> <F>
(<E>) <OM> <F>
(<ES> <OR> <ES>) <OM> <F>
(<T> <OR> <ES>) <OM> <F>
(<F> <OR> <ES>) <OM> <F>
(<V> <OR> <ES>) <OM> <F>
(A <OR> <ES>) <OM> <F>
(A > <ES>) <OM> <F>
(A > <T>) <OM> <F>
(A > <F>) <OM> <F>
(A > <N>) <OM> <F>
(A > 24) <OM> <F>
(A > 24) and <F>
(A > 24) and (<E>)
(A > 24) and (<ES> <OR> <ES>)
(A > 24) and (<T> <OR> <ES>)
(A > 24) and (<F> <OR> <ES>)
(A > 24) and (<V> <OR> <ES>)
(A > 24) and (B <OR> <ES>)
(A > 24) and (B < <ES>)
(A > 24) and (B < <T>)
(A > 24) and (B < <F>)
(A > 24) and (B < <N>)
(A > 24) and (B < 5)
```

* Ejercicio 24 *
Encuentre otro proceso de derivación a izquierda para la misma expresión lógica.

Ejemplo 19
Analicemos otra derivación que utiliza solo el operador or:

```
<E>
<ES>
<ES> <OA> <T>
<T> <OA> <T>
<F> <OA> <T>
<V> <OA> <T>
A <OA> <T>
A or <T>
A or <F>
A or <V>
A or B
```

La expresión obtenida es, supuestamente, "sintácticamente correcta" porque la pudimos derivar de la BNF oficial del lenguaje Pascal. Sin embargo, esta expresión solo tiene sentido si ambas variables son Booleanas.

Ejemplo 20
Dos situaciones peores que la anterior:

```
. . .
A or <F>
A or <N>
A or 12
```

y

```
<E>
<ES>
<T>
<F>
not <F>
not 436
```

Ambas expresiones obtenidas son, supuestamente, "sintácticamente correctas" porque las pudimos derivar de la BNF oficial del lenguaje Pascal.

* Ejercicio 25 *
Las dos derivaciones del Ejemplo 20, ¿producen expresiones "sintácticamente correctas" en Pascal? Justifique su respuesta. ¿Qué podría hacer para mejorar esta situación?

3.5.3 SENTENCIAS CON CONDICIONES BOOLEANAS

Las descripciones en BNF no siempre son exactas desde el punto de vista del programador. Ya lo hemos visto en los ejemplos 19 y 20. Estas "fallas" del BNF requieren el agregado de una RESTRICCIÓN en Lenguaje Natural que brinde máxima precisión a la situación descripta.

Veamos qué sucede con la descripción de sentencias como while e if. En el Manual de Referencia original de Pascal, Wirth las describe de esta manera:

```
<sentencia if> ::= if <expresión> then <sentencia> |
                   if <expresión> then <sentencia> else <sentencia>
<sentencia while> ::= while <expresión> do <sentencia>
```

Como observamos, tanto para la sentencia if como para la sentencia while la condición está representada mediante <expresión>, aunque sabemos que no puede ser una expresión cualquiera..

Ejemplo 21
De acuerdo a las definiciones escritas en BNF, una sentencia como while 2+3 do a := 5 sería una sentencia sintácticamente correcta ya que 2+3 es una <expresión> válida. Sin embargo, sabemos que la condición de un while en Pascal debe ser una expresión BOOLEANA.

➔ Conclusión importante: En algunos casos, no solo se debe conocer la definición en BNF del constructo, sino también las RESTRICCIONES que aparecen escritas en lenguaje natural. En el caso del libro de Wirth, algunas de estas definiciones en BNF tienen aclaraciones en las secciones del libro donde se trata el correspondiente constructo. En el caso de la <sentencia while>, por ejemplo, luego de su definición en BNF se aclara que <expresión> debe ser de tipo Boolean; esta restricción es tan importante como la misma definición BNF y, por ello, no pueden ser utilizadas separadamente.

* Ejercicio 26 *

Modifique la escritura de la BNF para if y para while de tal forma que no requieran una aclaración en lenguaje natural.