SSL

OBJETIVOS: Formar en las metodologías, técnicas y lenguajes de programación, como herramientas básicas para el desarrollo de software y el estudio de disciplinas que permitan crear nuevas tecnologías.

Asignatura: SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES

Departamento: Ingeniería en Sistemas de Información
Bloque: Tecnologías Básicas
Área: Programación
Horas/semana: 8 (cuatrimestral)
Horas/Año: 128 (en un cuatrimestre)

OBJETIVOS:

- Conocer los elementos propios de la sintaxis y semántica de los lenguajes de programación.
- Conocer los lenguajes formales y autómatas.
- Comprender conceptos y procedimientos de las gramáticas independientes (libres) de contexto y gramáticas regulares para especificar la sintaxis de los lenguajes de programación.
- Utilizar distintos tipos de autómatas y distintos tipos de notaciones gramaticales.
- Comprender el procesamiento de lenguajes y, en particular, el proceso de compilación.

El nombre de esta asignatura, "SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES" (SSL), se refiere a los LENGUAJES DE PROGRAMACIÓN.

En Informática, la sintaxis de un Lenguaje de Programación (LP) es el conjunto de reglas que define la correcta escritura de un programa en ese LP.

Ejemplo
Sea, en el lenguaje C, la declaración: int `a, b;`
Entonces, la sentencia `a = b + 1;` es sintácticamente correcta.
En cambio, `a := b 1 +;` no lo es.

Los LP están formados, básicamente, por un conjunto de Lenguajes Formales.

Entonces
DEFINICIONES BÁSICAS E INTRODUCCIÓN A LENGUAJES FORMALES

Los LF están formados por PALABRAS, que son CADENAS constituidas por SÍMBOLOS de un ALFABETO.

SÍMBOLOS y ALFABETOS

Un SÍMBOLO es el elemento constructivo básico, indivisible, a partir de la cual se forman los alfabetos.
Un ALFABETO es un conjunto finito de símbolos. **Σ** (sigma).
Ejemplo 1
La letra **a** es un símbolo o carácter que forma parte del alfabeto español, del alfabeto inglés, etc.
Los símbolos **>**, **=** y **+** son elementos del alfabeto de los operadores de los lenguajes Pascal y ANSI C .
El alfabeto **Σ = {0, 1}** proporciona los caracteres utilizados en la construcción de los números binarios.
Los números enteros con signo en base 10 se construyen ***Σ = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -, +}***
El "código ASCII" proporciona la representación interna de los caracteres que constituyen el alfabeto más utilizado en computación.

CADENAS: secuencia finita de caracteres tomados de cierto alfabeto y colocados uno a continuación de otro.
abac (se lee "a-b-a-c") es una cadena formada con caracteres del alfabeto {a, b, c}.
101110 ("uno-cero-uno-uno-uno-cero") es una cadena construida con caracteres del alfabeto {0, 1}.
a es una cadena formada por un solo símbolo de cualquier alfabeto que contenga el carácter a.

➔ Para los conocedores del Lenguaje ANSI C: recuerden la diferencia que existe entre 'a' y "a". Es la misma diferencia que existe entre un símbolo de un alfabeto y una cadena compuesta por ese único símbolo.

LONGITUD DE UNA CADENA |S| es la cantidad de caracteres que la componen.
La longitud de la cadena **abac** es: |abac| = 4.
CADENA VACÍA se simboliza con ε (épsilon), es la cadena que no tiene caracteres. → (|ε| = 0).
Este símbolo **ε** no forma parte de ningún alfabeto. Algunos autores utilizan λ (lambda) .

LA POTENCIACIÓN DE UN SÍMBOLO
La cadena **aaaaabbbbbbb** → a^5 b^7 *El* supraíndice *indica el número de veces que aparece el símbolo "potenciado".*

CONCATENACIÓN DE DOS CADENAS :aplicada a cadenas ($S_1 \bullet S_2$) produce una nueva cadena formada por los caracteres de la primera cadena seguidos inmediatamente por los caracteres de la segunda cadena.
➔ La concatenación NO ES CONMUTATIVA (excepto en casos muy especiales).
a•b = **ab**, mientras que **b•a** = **ba**. Como **ab** ≠ **ba**, se comprueba que la concatenación no es conmutativa.
Si ambas cadenas son idénticas, entonces la concatenación sí es conmutativa, *también* cuando ambas cadenas están compuestas por una repetición del mismo carácter. La cadena vacía (**ε**) es la IDENTIDAD para la concatenación. S•ε = ε•S = S.

POTENCIACIÓN DE UNA CADENA si S es una cadena, entonces S^n (con n ≥ 1 y entero) representa la cadena que resulta de concatenar la cadena S, consigo misma, n-1 veces; es decir: la cadena final resulta de repetir la cadena original n veces. Por lo tanto: S^1 = S, S^2 = SS, S^3 = SSS, etc. $\mathbf{S^0}$ es **ε** (la cadena vacía), para cualquier cadena S.

(abc)^0 = **(1234)^0** = ε; $\varepsilon^n = \varepsilon$ *para cualquier valor entero de* $n \geq 0$. ¿Por qué?

PREFIJO, SUFIJO Y SUBCADENA DE UNA CADENA
1º Un PREFIJO de una cadena es una secuencia de cero o más caracteres iniciales de esa cadena.
Sea la cadena **abcd**. Entonces, sus prefijos son: **ε** (la cadena vacía), **a**, **ab**, **abc** y **abcd** (la cadena completa).
2º Un SUFIJO de una cadena es una secuencia de cero o más caracteres finales de esa cadena.
Sea la cadena **abcd**. Entonces, sus sufijos son: **ε**, **d**, **cd**, **bcd** y **abcd**.
3º Una SUBCADENA de una cadena es una secuencia de caracteres que se obtiene eliminando cero o más caracteres iniciales y cero o más caracteres finales de esa cadena.
Sea la cadena **abcd**. Entonces, sus subcadenas son: **abcd**, **bcd**, **cd**, **d**, **abc**, **ab**, **a**, **bc, b, c** y **ε**.

LENGUAJES NATURALES Y LENGUAJES FORMALES

LENGUAJE NATURAL: lenguaje hablado y/o escrito que es utilizado por los seres humanos para comunicarse.
Los Lenguajes Naturales tienen tres características fundamentales:

1º **Evolucionan** con el paso del tiempo, incorporando nuevos términos y nuevas reglas gramaticales para mejorar la comunicación;

2º Sus **Reglas Gramaticales** surgen después que el lenguaje para poder explicar su estructura, es decir: su sintaxis;

3º El **significado** (o sea, la semántica) de cada palabra y de cada oración de un Lenguaje Natural es, en general, más importante que su composición sintáctica.

LENGUAJE FORMAL (LF) es un conjunto de cadenas formadas con los caracteres de un alfabeto dado, y tiene una característica fundamental: es un lenguaje abstracto, en el que el término FORMAL solo estamos interesados en la *forma* de una cadena (es decir, su sintaxis) y no en su *significado* (semántica).

LENGUAJES FORMALES están definidos por reglas gramaticales preestablecidas y se deben ajustar rigurosamente a ellas. En consecuencia, un LF *nunca puede evolucionar*.

PALABRA "si una *cadena* pertenece a un determinado lenguaje, es una PALABRA de ese lenguaje". La pertenencia de una cadena a un LF le da la propiedad de ser PALABRA de ese lenguaje en particular.

Descripción por enumeración: sea el lenguaje $L = \{\mathbf{1}, \mathbf{10}, \mathbf{100}, \mathbf{1000}, \mathbf{10000}\}$. Utilizando el operador supraíndice, este lenguaje puede ser descripto por comprensión, en forma más compacta, así: $L = \{\mathbf{10}^n / 0 \leq n \leq 4\}$ sobre el alfabeto {0, 1}.
Entonces, la cadena **100** ("uno-cero-cero") es una palabra del lenguaje L, mientras que **1101** ("uno-uno-cero-uno") es una cadena construida con caracteres del mismo alfabeto pero no es una palabra de L, ya que no pertenece a este lenguaje.

Nota 5: Un LF puede ser descripto por enumeración, por comprensión, mediante una frase en un lenguaje natural (castellano, en nuestro caso) o mediante otras formas especiales que veremos más adelante.

Ejemplo 30
Sea el lenguaje L del Ejemplo anterior. Lo podemos describir mediante una frase en castellano (lenguaje natural), de la siguiente manera: "L es un lenguaje de 5 palabras, cada una de las cuales comienza con un símbolo **1** que es seguido por una secuencia de entre cero y cuatro **0**s".

Si analizamos las tres formas de describir este lenguaje, seguramente consideraremos que son más simples las dos que figuran en el Ejemplo 29. Sin embargo, hay muchos casos en los que la dificultad para describir el lenguaje con "simbología matemática" es evidente; en estos casos, debemos utilizar una frase en lenguaje natural, evitando todo tipo de ambigüedad.

Ejemplo 31
El lenguaje $L = \{\varepsilon, \mathbf{b}, \mathbf{bb}, \mathbf{bbb}, \mathbf{bbbb}, \mathbf{bbbbb}, \mathbf{bbbbbb}, \mathbf{bbbbbbb}, \mathbf{bbbbbbbb}\}$ se comprende mejor si sus palabras se describen por comprensión, así: $L = \{\mathbf{b}^i / 0 \leq i \leq 8\}$.

Ejemplo 32
El lenguaje $L = \{\mathbf{ab}^i / 1 \leq i \leq 3000\}$ está formado por 3000 palabras; cada una de ellas comienza con la letra a, la cual es seguida de "entre 1 y 3000 bes". ¡Qué complicado sería describir y leer este lenguaje por enumeración!

Ejemplo 33
El lenguaje $L = \{\mathbf{a}^{2i}\mathbf{b}^i / 0 \leq i \leq 2\}$ está formado por tres palabras: si $i = 0$, $\mathbf{a}^{2*0}\mathbf{b}^0 = \mathbf{a}^0\mathbf{b}^0$ (ausencia de aes y ausencia de bes) representa la palabra vacía (**ε**); si $i = 1$, la palabra es $\mathbf{a}^{2*1}\mathbf{b}^1 = \mathbf{aab}$; y si $i = 2$, la palabra es $\mathbf{a}^{2*2}\mathbf{b}^2 = \mathbf{aaaabb}$.
En consecuencia, si enumeramos todas las palabras, el lenguaje L es {**ε**, **aab**, **aaaabb**}.

1.3.2 PROPIEDADES DE LAS PALABRAS

Dado que una PALABRA es una cadena que pertenece a un determinado lenguaje, todos los conceptos sobre *cadena* explicados anteriormente se aplican también a las palabras.
Por lo tanto, hablaremos de: longitud de una palabra, palabra vacía, concatenación de dos o más palabras, potenciación de una palabra, prefijos y sufijos de una palabra, subpalabras, con el mismo significado ya visto.

Ejemplo 34
Sea el lenguaje $L = \{(\mathbf{abc})^n / 0 \leq n \leq 3\} = \{\varepsilon, \mathbf{abc}, \mathbf{abcabc}, \mathbf{abcabcabc}\}$.
Entonces: la longitud de la palabra **abc** es $|\mathbf{abc}| = 3$.
La palabra vacía **ε** es un miembro de este lenguaje.

La concatenación de las palabras **abc** y **abcabc** produce otra palabra de este lenguaje (**abcabcabc**); en cambio, la concatenación de la palabra **abcabc** consigo misma produce la cadena **abcabcabcabc**, que no es una palabra de este lenguaje.
La potencia $(\mathbf{abc})^2$ es una palabra del lenguaje.
La cadena **ab** es un prefijo de todas las palabras no vacías; lo mismo ocurre con el sufijo **bc**.
Finalmente, **b** es una subpalabra de todas las palabras no vacías de este lenguaje.

➔ La concatenación de dos palabras produce una *cadena* que no siempre es una *palabra* del lenguaje, como se observa en el ejemplo anterior y en el ejemplo que sigue.

Ejemplo 35
Sea el lenguaje $L = \{\mathbf{a}^{2n+1} / 0 \leq n \leq 200\}$. Este lenguaje está formado por 201 palabras, cada una de las cuales tiene un número impar de letras a. Por lo tanto, si $n = 0$, la palabra es **a**; si $n = 1$, la palabra es **aaa**; etc.
Si se concatenan dos palabras cualesquiera de L, el resultado será una cadena con una cantidad par de letras a, ya que la suma de dos números impares produce un número par.
En consecuencia, la concatenación de dos palabras cualesquiera de este lenguaje produce una cadena que nunca es una palabra de L.

1.3.3 CARDINALIDAD DE UN LENGUAJE FORMAL

La CARDINALIDAD de un LF es la cantidad de palabras que lo componen.

Ejemplo 36
$L = \{\mathbf{a}, \mathbf{ab}, \mathbf{aab}\}$ es un lenguaje de cardinalidad 3 sobre el alfabeto {a, b}.

Ejemplo 37
El lenguaje $L = \{\varepsilon\}$ es un lenguaje muy especial, pues está formado solo por la palabra vacía. Su cardinalidad es 1, ya que contiene una palabra.

1.3.4 SUBLENGUAJES

Dado que un Lenguaje Formal es un conjunto, un SUBLENGUAJE es un subconjunto de un lenguaje dado.

Ejemplo 38
Sea $L1 = \{\mathbf{a}, \mathbf{ab}, \mathbf{aab}\}$. Entonces, $L2 = \{\mathbf{ab}, \mathbf{aab}\}$ es un sublenguaje de L1, mientras que $L3 = \{\ \}$ es el sublenguaje vacío de L1.

➔ El sublenguaje vacío, habitualmente representado con el símbolo ø, es sublenguaje de cualquier lenguaje.

➔ No se debe confundir el sublenguaje vacío, que tiene cardinalidad 0, con el lenguaje que solo contiene la palabra vacía, que tiene cardinalidad 1.

1.3.5 LENGUAJES FORMALES INFINITOS

Todos los lenguajes ejemplificados hasta el momento han sido LENGUAJES FORMALES FINITOS, es decir: lenguajes con un número finito de palabras.
Sin embargo, los LFs también pueden ser INFINITOS, lo que significa que estos lenguajes pueden tener una cantidad infinita de palabras, pero cada una de longitud finita (no existen las palabras de longitud infinita).

Nota 6: Las propiedades de las palabras que han sido ejemplificadas anteriormente se aplican también a los lenguajes infinitos.

Ejemplo 39
$L = \{a^n / n \geq 1\}$ es un LF infinito ya que no existe un límite superior para el supraíndice n.
Cada palabra de este lenguaje está formada por una secuencia de una o más $a$es. Por ello, la concatenación de dos palabras cualesquiera de este lenguaje producirá siempre otra palabra del lenguaje L.
Por esta propiedad, se dice que este lenguaje L es *cerrado* bajo la concatenación.

Ejemplo 40
El lenguaje $L = \{\mathbf{a}b^n / n \geq 0\}$ es otro lenguaje infinito. Está formado por la palabra **a** y todas aquellas palabras que comienzan con a, seguida de una secuencia de una o más $b$es.
Este lenguaje no es cerrado bajo la concatenación ya que, por caso: si consideramos sus dos palabras de menor longitud –**a** y **ab**– y las concatenamos, obtenemos la cadena **aab**, que no es una palabra de este lenguaje.

1.3.6 LENGUAJE UNIVERSAL SOBRE UN ALFABETO

Dado un alfabeto **Σ**, el LENGUAJE UNIVERSAL sobre este alfabeto es un lenguaje *infinito* que contiene todas las palabras que se pueden formar con los caracteres del alfabeto **Σ**, más la palabra vacía.
Se lo representa con la notación **Σ***, que se lee "sigma clausura" o "sigma estrella".

➔ Importante: acabamos de introducir un nuevo operador que representamos con el supraíndice *. Este es un operador unario (sobre un solo operando), como lo es la potenciación en matemáticas.
Se lo denomina "clausura de Keene" o "estrella de Kleene", y será muy utilizado en próximos capítulos.

Una propiedad fundamental del Lenguaje Universal es que es cerrado bajo concatenación (por eso la denominación de "sigma clausura").

Ejemplo 41
Si $\Sigma = \{a, b\}$, entonces el Lenguaje Universal para este alfabeto es:

$$\Sigma^* = \{\varepsilon, \mathbf{a}, \mathbf{b}, \mathbf{aa}, \mathbf{ab}, \mathbf{ba}, \mathbf{bb}, \mathbf{aaa}, \mathbf{aab}, \mathbf{aba}, \mathbf{abb}, ..., \mathbf{aabaabbbab}, \ldots\},$$

y la concatenación de palabras de este lenguaje siempre produce una palabra del Lenguaje Universal. ¿Puede encontrar un caso en que esta afirmación sea falsa?

➔ Cualquier lenguaje L sobre el alfabeto **Σ** es un sublenguaje de **Σ***. Por lo tanto, existen infinitos lenguajes sobre un alfabeto dado.

1.4 RESUMEN: TÉRMINOS Y FRASES IMPORTANTES

SÍMBOLO o CARÁCTER
ALFABETO
CADENA (cadena de caracteres)
CONCATENACIÓN DE CARACTERES
LONGITUD DE UNA CADENA
CADENA VACÍA o CADENA NULA
POTENCIACIÓN DE UN SÍMBOLO
OPERADOR SUPRAÍNDICE
CONCATENACIÓN DE CADENAS
CONMUTATIVIDAD DE LA CONCATENACIÓN
POTENCIACIÓN DE UNA CADENA
PREFIJO DE UNA CADENA
SUFIJO DE UNA CADENA
SUBCADENA
LENGUAJES NATURALES vs LENGUAJES FORMALES
PROPIEDADES DE UN LENGUAJE NATURAL
PROPIEDADES DE UN LENGUAJE FORMAL

PALABRA
DIFERENCIA ENTRE CADENA Y PALABRA
DESCRIPCIÓN DE UN LENGUAJE POR ENUMERACIÓN
DESCRIPCIÓN DE UN LENGUAJE POR COMPRENSIÓN
DESCRIPCIÓN DE UN LENGUAJE MEDIANTE UNA FRASE EN LENGUAJE NATURAL
PROPIEDADES DE LAS PALABRAS
CONCATENACIÓN DE PALABRAS
CARDINALIDAD DE UN LENGUAJE
SUBLENGUAJES
LENGUAJES FORMALES
LENGUAJES FORMALES FINITOS
LENGUAJES FORMALES INFINITOS
LENGUAJE UNIVERSAL
OPERADOR "CLAUSURA DE KLEENE" O "ESTRELLA DE KLEENE"

1.5 EJERCICIOS

(1) Dado el alfabeto $\Sigma = \{a, b, c\}$, escriba las palabras del lenguaje $L = \{x / x \in \Sigma\}$.

(2) ¿Cuál es la cardinalidad del lenguaje $L = \{\varepsilon, \mathbf{a}, \mathbf{aa}, \mathbf{aaa}\}$?

(3) Describa, mediante una frase en castellano, el lenguaje del Ejercicio 2.

(4) Escriba todas las palabras del lenguaje $L = \{\mathbf{a}^{2n+1} / 1 \leq n \leq 4\}$.

(5) Describa, mediante una frase en castellano, el lenguaje del Ejercicio (4).

(6) Sea el lenguaje $L = \{\varepsilon, \mathbf{a}, \mathbf{ba}, \mathbf{abc}\}$. ¿Cuál es el mínimo alfabeto Σ sobre el que se puede construir este lenguaje?

(7) Sea el lenguaje $L = \{\mathbf{0}, \mathbf{00}, \mathbf{01}, \mathbf{010}\}$. Escriba una concatenación de dos palabras que produce otra palabra de este lenguaje.

(8) Sea el lenguaje anterior. Escriba una concatenación de dos palabras que produce una cadena que no es palabra del lenguaje.

(9) Sea el lenguaje del Ejercicio (7). Escriba tres sublenguajes de L, de distinta cardinalidad.

(10) Sea el lenguaje $L = \{(\mathbf{ab})^{2n} / 0 \leq n \leq 100\}$. Escriba las tres palabras de menor longitud de este lenguaje.

(11) Sea el lenguaje del Ejercicio (10). ¿Es cerrado bajo concatenación?

(12) Describa, mediante una frase en castellano, el lenguaje del Ejercicio (10).

(13) Describa, mediante una frase en castellano, el lenguaje $L = \{\mathbf{a}^n\mathbf{b}^n / 1 \leq n \leq 3000\}$.

(14) Sea $\Sigma = \{a, b\}$ y sea el lenguaje Σ^*. ¿Cuántas palabras de longitud 3 tiene este lenguaje? ¿Y cuántas de longitud 4?

(15) Sea el lenguaje infinito $L = \{\mathbf{ab}^n / n \geq 1\}$. Escriba las tres palabras de menor longitud.

(16) Sea el lenguaje infinito $L = \{(\mathbf{ab})^n / n \geq 1\}$. Escriba las tres palabras de menor longitud.

(17) Sea el alfabeto $\Sigma = \{0, 1\}$. Describa, por comprensión, un lenguaje infinito L sobre Σ (que no sea Σ^*).

(18) Describa, mediante una frase en castellano, el lenguaje definido en el ejercicio anterior.

IMPLEMENTACIÓN EN ANSI C

Para cada una de los ejercicios que siguen, construir en ANSI C la función solicitada y un programa que la pruebe con todos los datos constantes que sean necesarios.

(19) Longitud de una cadena

(20) Determinar si una cadena dada es vacía.

(21) Concatenación de dos cadenas.

(22) Determinar si una subcadena es Prefijo de una cadena dada.

Estándares, Lenguaje y Biblioteca

El estándar que norma al *Lenguaje de Programación C* es el publicado en el documento **ANSI X3.159-1989**, en 1989 y ratificado internacionalmente por el documento **ISO/IEC 9898:1990** en 1990. Este lenguaje es conocido indistintamente como **ANSI C**, **ISO C**, **C89** ó **C90**. Actualmente se encuentra publicada en **ISO/IEC 9898:1990** una segunda versión, el **C99** pero existen pocas implementaciones comerciales del mismo a diferencia del muy extendido C90. El curso, y por lo tanto el presente módulo, giran entorno al C90.

Es importante destacar que el estándar norma dos aspectos importantes y bien diferenciados del Lenguaje de Programación C, el *Lenguaje* en sí mismo y la *Biblioteca*. A manera de ejemplo y en pocas palabras, el Lenguaje se refiere a asuntos como la forma correcta de escribir (*sintaxis*) una *sentencia de selección* y establecer el significado (*semántica*) de ciertas construcciones. Por otro lado, la sección Biblioteca del estándar se encarga, entre otras cosas, de definir la especificación de funciones (e.g. funciones de entrada / salida, de manejo de memoria, etc.) de uso común en todas las plataformas. Estas funciones deben estar disponibles en todas las implementaciones y serán estas las funciones que utilizaremos a lo largo del curso.

1. Lenguaje

1.1. Sobre la Sintaxis

Un Lenguaje de Programación está compuesto por un conjunto de *Lenguajes Regulares* y otro conjunto de *Lenguajes Independientes del Contexto*.

Los *componentes léxicos (tokens)* (identificadores, números enteros, números reales, caracteres constantes, cadenas constantes, operadores y caracteres de puntuación) constituyen diferentes Lenguajes Regulares. Algunos de estos lenguajes son *finitos*, como los operadores y los caracteres de puntuación, y otros son *infinitos*, como sucede con los identificadores o los números reales. En ambos casos, estos lenguajes pueden ser *generados* por *Gramáticas Regulares* y *descriptos* mediante *Expresiones Regulares*.

En cambio, las expresiones y las sentencias de un Lenguaje de Programación son, en general, *Lenguajes Independientes del Contexto*. Como tales, estos lenguajes no pueden ser generados por Gramáticas Regulares sino que requieren *Gramáticas Independientes del Contexto*.

Una gramática formal *no solo genera* un lenguaje formal, sino que también *se puede utilizar para describir* la sintaxis del lenguaje generado. Como veremos en la descripción sintáctica del ANSI C, aún muchos componentes léxicos son descriptos mediante Gramáticas Independientes del Contexto, aunque, en este caso, la estructura de las producciones es muy similar a la de las Gramáticas Regulares, como se observará en el siguiente análisis.

> *Nota*: Los no-terminales se representan mediante nombres o frases encerrados entre corchetes angulares (< y >), como, por ejemplo, <identificador>; por otro lado, los terminales se resaltan en "**negritas**".

En las descripciones gramaticales que siguen, hay varios no-terminales con producciones del tipo v -> v | vw, en las que tanto v como w son no-terminales, pero donde w actúa como "simplificación" de un grupo de terminales. Un ejemplo de esta situación es la producción:

```
<identificador> -> <identificador> <dígito>
```

Por lo tanto, ese tipo de producción es "quasi-lineal a izquierda" y, por ser recursiva, es la producción base en la generación del lenguaje regular w+.

1.2. Sobre las Gramáticas

Las gramáticas formales utilizadas para definir la sintaxis de los Lenguajes de Programación pertenecen a una categoría llamada *Gramáticas Independientes del Contexto (GICs).*

La característica fundamental de una GIC es que todas sus producciones son de la forma: variable -> secuencia de variables y/o terminales, es decir:

```
variable -> (variable + terminal)*
```

Ejemplos:

```
S -> a    corresponde a   variable -> terminal
S -> Ra   corresponde a   variable -> variable terminal
R -> aa   corresponde a   variable -> terminal terminal
T -> QPZ  corresponde a   variable -> variable variable variable
```

Los Lenguajes Formales más importantes son los lenguajes infinitos. Consecuentemente, las GICs más importantes son aquellas que generan lenguajes infinitos.

En la descripción sintáctica del ANSI C encontraremos algunas GICs que describen lenguajes finitos y otras GICs que describen lenguajes infinitos.

1.3. Sobre la Sintaxis del ANSI C

Analizaré -con todos ustedes- aspectos importantes de la sintaxis del ANSI C, ampliando lo que escribí en las dos secciones anteriores, Sobre la Sintaxis y Sobre la Gramática.

La sintaxis de un Lenguaje de Programación debe describirse con precisión, utilizando una notación sin ambigüedades. La notación que utilizamos en este módulo, comúnmente llamada **BNF extendida**, deriva de la usada en la representación de las reglas o producciones de una GIC.

Ejemplo 1

Describamos una estructura sintáctica de dos maneras:

- Mediante un lenguaje natural, normalmente ambiguo;
- Mediante una notación especial no-ambigua.

Si queremos especificar (o definir) informalmente la sintaxis de la **sentencia WHILE** en **Pascal**, utilizando el lenguaje castellano, podemos decir:

```
Una sentencia WHILE comienza con la palabra reservada WHILE,
     que es seguida de una expresión Booleana,
     que es seguida de la palabra reservada DO,
     que es seguida de una sentencia.
```

Si, en cambio, utilizamos una notación BNF como la que es utilizada en este módulo, la sintaxis de la **sentencia WHILE** en Pascal la especificamos de esta manera:

```
<sentencia-WHILE> -> WHILE <expresión Booleana> DO <sentencia>
```

1. ¿Comprende cabalmente esta última descripción?

2. ¿Cuál especificación le parece mejor y por qué?

Ejemplo 2

Como escribe David Watt en su libro "*Programming Language Syntax and Semantics*" (1991, Prentice Hall): La siguiente es la especificación informal de la sintaxis de **identificadores**, parafraseada de un manual de un viejo lenguaje de programación:

```
Un identificador es una secuencia de letras,
     posiblemente con la inclusión de subrayados (guiones bajos) en
     medio.
```

A partir de esta especificación, es evidente que `PI`, `CANTIDAD` y `CANTIDAD_TOTAL` son identificadores correctos. También es muy claro que `PI34`, `78` y `CANTIDAD_` no son identificadores válidos.

3. ¿Está de acuerdo?

Pero, si nos atenemos a esa definición: 1) ¿Es una única letra, como X, un identificador válido?; 2) ¿la frase “subrayados en medio” significa que puede haber varios consecutivos o que deben estar separados o que ...?

4. ¿Puede responder con seguridad a estas preguntas?

5. ¿Se le ocurre alguna otra *ambigüedad* en la especificación informal que estamos analizando?

Esas imprecisiones o ambigüedades son casi inevitables en una especificación informal, como la que se realiza utilizando un lenguaje natural.

Ahora, supongamos que usamos una notación BNF para describir estos identificadores, y lo hacemos de la siguiente manera (asumiendo que ya hemos definido <letra>):

```
<identificador> ->
     <letra> | <identificador> <letra> | <identificador> _ <letra>
```

Aclarando que el símbolo ‘|’ significa ‘o’, las líneas anteriores involucran tres reglas que actúan en la definición de nuestro ya famoso “identificador”.

Observe que la primera regla nos dice que un identificador puede ser una letra (por ejemplo, X), mientras que las otras dos reglas son *recursivas* (porque `<identificador>` aparece en ambos lados de la regla). Estas reglas recursivas permiten generar, muy simplemente, un número infinito de identificadores mediante la aplicación sucesiva de esas reglas.

Ejemplo 3

```
<identificador> ->
     <identificador> <letra>
     <identificador> A
     <identificador> <letra> A
     <identificador> BA
     <identificador> _ <letra> BA
```

<identificador> _XBA
<letra> _XBA
R_XBA

6. ¿Comprende cómo generar cualquier identificador?

7. De acuerdo a la descripción formal en BNF, ¿puede un identificador comenzar con un subrayado?

8. ¿Es A__B un identificador válido? ¿Lo puede generar?

9. ¿Puede generar el identificador A_B_C?

10. ¿Podría describir en castellano, sin ambigüedades, el conjunto de identificadores válidos en este antiguo Lenguaje de Programación?

Luego de leídas las dos secciones previas, pasamos a iniciar el análisis de la "Sintaxis del ANSI C", cuya especificación en BNF comienza con la siguiente sección, "Gramática Léxica".

En la tercera subsección, “Identificadores”, se especifica cómo son los identificadores en ANSI C. La notación utilizada ya no debe presentar dificultades. Sin embargo, aparece algo nuevo: la frase “*uno de*”.

Esta la primera incorporación novedosa que hacemos a la notación BNF que usamos hasta ahora. La empleamos para mejorar la legibilidad cuando existen varios ‘o’, es decir: por ejemplo, sin el uso de “*uno de*”, <dígito> debería especificarse así:

<dígito> -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

y esta enumeración no es tan legible como la que ocurre con el uso de “*uno de*”.

11. De acuerdo a la especificación de la página 4, ¿es ___123 un identificador válido en ANSI C?

En la sección *Identificador* se especifica cómo son los identificadores en ANSI C. En la notación utilizada, llamada *BNF Extendida*, se utilizan algunos símbolos especiales que colaboran en la definición o especificación de la correspondiente construcción.

Por ejemplo, para definir la sintaxis de <identificador> se utilizan ciertos símbolos especiales:

<, >, ->, |, *uno de*

Cada uno de ellos tiene una aplicación especial y se denomina *metasímbolo*.

Seguimos adelante; encontramos la sección *Constante Real*. En esta sección se define, obviamente, la sintaxis correcta de cualquier constante real correcta en ANSI C.

Comienza diciendo que una <constante real> adopta dos formas posibles. En esta especificación aparece un nuevo metasímbolo: el representado por el "**?**". Este metasímbolo significa que el elemento que lo precede puede figurar o no.

Por ejemplo, “<parte exponente>**?**” significa que <parte exponente> puede figurar o no en la construcción de una constante real.

Entonces, la primera línea de esta especificación:

```
<constante real> -> <constante fracción> <parte exponente>? <sufijo
     real>?
```

se lee de esta manera: “una *constante real* en ANSI C es una *constante fracción*, seguida *eventualmente* de una *parte exponente*, seguida *eventualmente* de un *sufijo real*.

Por lo tanto, esta notación compacta permite definir cuatro tipos diferentes de constantes reales:

```
<constante real> -> <constante fracción> <parte exponente> <sufijo
     real>
<constante real> -> <constante fracción> <sufijo real>
<constante real> -> <constante fracción> <parte exponente>
<constante real> -> <constante fracción>
```

Para completar la definición de una constante real, debemos analizar la definición de <constante fracción>, la de <parte exponente> y la de <sufijo real>.

12. Termine el análisis de la especificación sintáctica de la <constante real> en ANSI C, escriba 2 ejemplos de cada caso posible y escriba 5 “constantes reales” erróneas.