SSL

Agregados

Jorge D. Muchnik, 2011

20110411

INTRODUCCIÓN

El libro "*Sintaxis y Semántica de los Lenguajes*" está formado por tres volúmenes:

- Vol. 1 – Desde sus Usuarios (programadores y otros)
- Vol. 2 – Desde el Compilador
- Vol. 3 – Algoritmos

A un año de su publicación y luego de ser aplicado en los cursos de la asignatura durante 2010, el autor del libro, el Profesor Jorge Muchnik, produjo nuevo material complementario.

Esta publicación es un complemento al libro (*book companion*) que reúne ese nuevo material: la fe de erratas, aclaraciones sobre los temas *Proceso de Compilación* y *Semántica*, y ejercicios resueltos.

Prof. Ing. José María Sola, Abril, 2011

ÍNDICE

1 VOLUMEN 1

1.1 Fe de Erratas

Página 21 – Ejemplo 8

El alfabeto no es {a, b} sino {a}.

Página 33 – Sección 3.1

Primera frase: en las CATEGORÍAS LÉXICAS o TOKENS agregar **las palabras reservadas** después de **los identificadores**.

Página 41 – Ambas tablas son corregidas así:

PASO Nº	PRODUCCIÓN APLICADA	CADENA DE DERIVACIÓN OBTENIDA
1	(axioma)	E
2	1	E + E
3	2	E * E + E
4	1	E + E * E + E
5	4	N + E * E + E
6	5	1 + E * E + E
7	4	1 + N * E + E
8	5	1 + 2 * E + E
9	3	1 + 2 * (E) + E
10	1	1 + 2 * (E + E) + E
11	4	1 + 2 * (N + E) + E
12	5	1 + 2 * (3 + E) + E
13	4	1 + 2 * (3 + N) + E
14	5	1 + 2 * (3 + 4) + E
15	4	1 + 2 * (3 + 4) + N
16	5	1 + 2 * (3 + 4) + 5

Ejemplo 10

CADENA DE DERIVACIÓN A REDUCIR	PRODUCCIÓN A APLICAR	OPERACIÓN
1 + 2 * (3 + 4) + 5	5	
1 + 2 * (3 + 4) + N.5	4	
1 + 2 * (3 + 4) + E.5	5	
1 + 2 * (3 + N.4) + E.5	4	
1 + 2 * (3 + E.4) + E.5	5	
1 + 2 * (N.3 + E.4) + E.5	4	
1 + 2 * (E.3 + E.4) + E.5	1	3 + 4 = 7
1 + 2 * (E.7) + E.5	3	
1 + E.2 * E.7 + E.5	5	
1 + N.2 * E.7 + E.5	4	
1 + E.2 * E.7 + E.5	5	
N.1 + E.2 * E.7 + E.5	4	
E.1 + E.2 * E.7 + E.5	1	1 + 2 = 3
E.3 * E.7 * E.5	2	3 * 7 = 21
E.21 + E.5	1	21 + 5 = 26
E.26	(axioma)	Resultado Final

Página 44 – Sección 3.5, recuadro

4) un conjunto de PRODUCCIONES o REGLAS

Página 48 – Última frase

Las comillas en **"sintácticamente correcta"** se refieren a la inseguridad de esta frase. Lo que sí puede afirmarse con seguridad es que esa expresión es **derivable** desde la BNF dada.

Página 50 – Ejemplo 22

Debe decir: Un token o categoría léxica se define **de** esta manera, …

Página 52 – Ejemplo 26

Aclaración: main está en *negritas* por ser una palabra clave; no es una palabra reservada.

Página 54 – Ejercicio 32

Léase: `while (a > b) b++;`

Página 60 – Aclaración general

En ANSI C, la evaluación de toda expresión lógica producirá el valor 1 si es verdadera y el valor 0 si es falsa. En general, toda expresión cuya evaluación produce un valor distinto de 0 es verdadera.

Página 61 – Ejercicio 50, linea 26

Debe decir: *expPostfijo*.

Página 61 – Ejemplo 28

En las 3 líneas debe decir: `expPostfijo`.

Página 70 – Sección 4.2.2.4

Debe decir: … delimitada por comillas ("). Como la comilla es delimitador, debe escribírsela de otra manera cuando se la utiliza …

Página 71 – Ejercicio 10

El punto (a) debe leerse: ¿Un operador … de valores? …

Página 79 – Ejercicio 13

Debe decir: Escriba una ER que represente al LR: "Todas las palabras … que comienzan con … 1200 caracteres, o comienzan con …".

Página 82 – Párrafo posterior al Ejercicio 19

Debe decir: La diferencia existente … en los Ejemplos 20 y 22: …

Página 89 – Ejercicio 42

Debe decir: Escriba las 9 palabras de menor …

Página 89 – Ejercicio 43

La 2º ER es (a+b)*b.

Página 101 – Ejercicio 20, Solución 2

Debe decir: `return s[0]=='\0';`

Página 103 – Ejercicio 31

85

Página 103 – Cap. 4, Ejercicio 5

(b) int, long int, int, …

Página 104 – Ejercicio 42

ε, a, c, aa, ab, ac, ca, cb, cc

1.2 Ejercicios Resueltos

(Agradezco la colaboración de las Profesoras Ana Díaz Bott, Adriana Adamoli y Marta Ferrari)

1.2.1 Capítulo 1

* Ejercicio 1 * (pág.8)
Σ = {0, 1, 2, 3, 4, 5, 6, 7, 9, +, –}

* Ejercicio 2 * (pág.8)
012 210

* Ejercicio 3 * (pág.8)
abababcde

* Ejercicio 4 * (pág.9)
$a^{1300}b^{846}a^{257}$

* Ejercicio 5 * (pág.10)
aabbaaabba

* *Ejercicio 6 * (pág.11)
Porque c^0 significaría la ausencia de carácter y eso no tiene sentido.

* Ejercicio 7 * (pág.11)
Ambos representan la cadena vacía porque S^0 es la cadena vacía para cualquier cadena S.

* Ejercicio 8 * (pág.11)
$(ab^3)^3 = (abbb)^3 = abbbabbbabbb$ y $((ab)^3)^3 = (ababab)^3 = abababababababababab$

* Ejercicio 10 * (pág.12)
El alfabeto mínimo es = {A, r, g, e, n, t, i, a, H, o, l, d, B, s}

* Ejercicio 11* (pág.13)
Por ejemplo: L = {ccccc, ppppp, cpcpc, ppccp}

* Ejercicio 12 * (pág.14)
$L = \{b^n / 0 \leq n \leq 8\}$

* Ejercicio 13 * (pág.14)
“El lenguaje de todas las palabras sobre el alfabeto {b} que están formadas por la concatenación del carácter b consigo mismo, entre una y ocho veces e incluye a la palabra vacía”.

* Ejercicio 14 * (pág.14)
Por extensión se podría describir aunque sería bastante tedioso. Por comprensión no se puede porque no tenemos los operadores adecuados para hacer esta descripción.

* Ejercicio 15 * (pág.15)
“El lenguaje de todas las primeras 201 palabras sobre el alfabeto {a} formado por la concatenación de la letra a consigo misma y donde cada una de ellas tiene un número impar de letras a”.

* Ejercicio 16 * (pág.15)
$L = \{a^{2n} / 0 \leq n \leq 400\}$

* Ejercicio 17 * (pág.16)
a) aba, abba, abbba.
b) “El lenguaje de todas las palabras sobre el alfabeto {a,b} que comienza con una única a seguida de una o varias bes y terminan con exactamente una a”.
b’) “El lenguaje sobre el alfabeto {a, b} donde todas las palabras tienen exactamente dos aes (una como primer carácter de la palabra y otra como último carácter de la palabra) y en el medio tienen una o más bes.”

* Ejercicio 18 * (pág.17)
NO porque el Lenguaje Universal es cerrado bajo concatenación.

* Ejercicio 19 * (pág.18)
palabras reservadas : L Finito
nombres creados por el programador (Identificadores): L Infinito
constantes enteras y reales: L Infinito
caracteres de puntuación: L Finito
operadores aritméticos: L Finito
operadores lógicos: L Finito
declaraciones: L Infinito (no regular)
expresiones: L Infinito (no regular)
sentencias: L Infinito (no regular)

* Ejercicio 20 * (pág.18)
(a)

```
unsigned int LongitudCadena (char *s) {
  unsigned int i;
  for(i=0; s[i]!='\0'; i++);
    return i;
}
```

(b)
Solución 1:

```
int EsCadenaVacia (char *s) {
   if (s[0] == '\0')
      return 1;
   else
      return 0;
}
```

Solución 2:

```
int EsCadenaVacia (char s[]) {
   return s[0]=='\0';
}
```

(c)

```
void ConcatenaDosCadenas(char* s1, const char* s2) {
  unsigned int i,j;
  for (i=0; s1[i]!='\0'; i++);
  for (j=0; s2[j]!='\0'; i++, j++)
     s1[i] = s2[j];
  s1[i]='\0';
}
```

* Ejercicio 21 * (pág.18)

Sugerencia: utilice cadenas constantes.

(a)

```
#include<stdio.h>
unsigned int LongitudCadena (char*);
int main (void)  {
  char cad1[] = "longitud 11";
  char cad2[] = "";
  char cad3[] = " ";
  printf("La longitud de cad1 es: %u\n", LongitudCadena (cad1));
  printf("La longitud de cad2 es: %u\n", LongitudCadena (cad2));
  printf("La longitud de cad3 es: %u\n", LongitudCadena (cad3));
  return 0;
} /* fin-main */

/* Desarrollo funcion LongitudCadena */
unsigned int LongitudCadena (char *s)  {
   unsigned int i;
   for(i=0; s[i]!='\0'; i++);
   return i;
} /* fin-LongitudCadena */
```

(b)

```
#include<stdio.h>
int EsCadenaVacia (char[]);
int main (void)  {
  char cad1[] = "no vacia";
  char cad2[] = "";
  if (EsCadenaVacia(cad1)) printf("La cadena 1 es vacia\n");
  else printf("La cadena 1 no es vacia %s\n", cad1);
  if (EsCadenaVacia(cad2)) printf("La cadena 2 es vacia\n");
  else printf("La cadena 2 no es vacia %s\n", cad2);
return 0;
} /* fin-main*/

/* Desarrollo funcion EsCadenaVacia */
int EsCadenaVacia(char s[])  {
  return s[0]=='\0';
} /* fin-EsCadenaVacia */
```

(c)

```
#include<stdio.h>
void ConcatenaDosCadenas (char*, const char*);
int main (void)  {
  char cad1[27+1] = "Primera parte ";
  char cad2[] = "Segunda parte";
  ConcatenaDosCadenas (cad1, cad2);
  printf ("La cadena concatenada es: %s\n", cad1);
  return 0;
} /* fin-main */

/* Desarrollo funcion ConcatenaDosCadenas */
void ConcatenaDosCadenas(char* s1, const char* s2) {
  unsigned int i,j;
  for (i=0; s1[i]!='\0'; i++);
  for (j=0; s2[j]!='\0'; i++, j++)
     s1[i] = s2[j];
  s1[i]='\0';
}
```

1.2.2 Capítulo 2

* Ejercicio 1 * (pág.20)
a) S –> aaT, T –> ε => aa
S –> aaT, T –> b => aab
b) No

* Ejercicio 2 * (pág.21)
El lenguaje formal L = {a, aa}
S -> aT, T -> ε => a
S -> aT, T -> a => aa

* Ejercicio 3 * (pág.22)
a) No, porque no hay una producción que permita obtener la última b. S –> bQ, Q –> a, ?
b) {aa, ab, ba, b}

* Ejercicio 4 * (pág.23)
a) S –> aT | aQ
   Q–> aT | aR
   T –> b
   R –> aT
b) G = ({S, Q, T, R}, {a, b}, { S –> aT | aQ, Q–> aT | aR, T –> b, R -> aT}, S)

* Ejercicio 5 * (pág.23)
a) S –> aR | aQ | ε
   Q–> aT
   T –> bR
   R –> b
b) G = ({S, Q, T, R}, {a, b}, {S-> aR, S -> aQ, S -> ε, Q -> aT, T -> bR, R -> b}, S)

* Ejercicio 6 * (pág.24)
ab S -> aT, T -> b

* Ejercicio 7 * (pág.24)
1º) S -> aS, 2º) S -> aS, 3º) S -> aT y 4º) T -> b y se genera la palabra: **aaab**

* Ejercicio 8 * (pág.25)
a) GR = ({S}, {0,1,2,3,4,5,6,7,8,9}, {S -> 0S, S ->1S, S -> 2S, S -> 3S, S -> 4S, S -> 5S,
S -> 6S, S -> 7S, S -> 8S, S -> 9S, S -> 0, S ->1, S -> 2, S -> 3, S -> 4, S -> 5,
S -> 6, S -> 7, S -> 8, S -> 9}, S)
b) GQR = ({S,N}, {0,1,2,3,4,5,6,7,8,9}, {S -> N, S -> NS, N -> 0, N -> 1, N -> 2, N -> 3, N -> 4,
N -> 5, N -> 6, N -> 7, N -> 8, N -> 9}, S)
c) GR 20 producciones y la GQR 12 producciones

* Ejercicio 9 * (pág.25)
S -> AN
A -> BN
B -> N | BN
N -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

* Ejercicio 10 * (pág.25)
S -> 0A, S-> 1A, S -> 2A, S -> 3A, S -> 4A, S -> 5A, S -> 6A, S -> 7A,
A -> 0B, A-> 1B, A -> 2B, A -> 3B, A -> 4B, A -> 5B, A -> 6B, A -> 7B,
B -> 0B, B-> 1B, B -> 2B, B -> 3B, B -> 4B, B -> 5B, B -> 6B, B -> 7B,
B -> 0, B-> 1, B -> 2, B -> 3, B -> 4, B -> 5, B -> 6, B -> 7

* Ejercicio 11 * (pág.26)
a) Sí, porque una GR siempre es un caso especial de una GIC.
b) No, porque las producciones de una GR son un subconjunto de las producciones de una GIC.
Por ejemplo: S -> abc es una producción válida para una GIC pero no lo es para una GR.

* Ejercicio 12 * (pág.26)
a) **a** aplicando la producción S -> a entonces se genera la palabra **a**
b) **aab** aplicando las producciones: 1º) S -> aSb, 2º) S -> a entonces se genera la palabra **aab**

* Ejercicio 13 * (pág.26)
$L = \{a^{n+1}b^n / n \geq 0\}$

* Ejercicio 14 * (pág.26)
S -> aSb | b

* Ejercicio 15 * (pág.26)
S -> aaTbQ
T -> aaTb | b
Q -> aQ | ε

* Ejercicio 16 * (pág.26)
Si toda GQR puede ser re-escrita mediante una GR y, a su vez, toda GR es un subconjunto de las producciones de una GIC entonces, una GQR es un caso particular de una GIC.

* Ejercicio 17 * (pág.27)
S
ACaB
AaaCB (aplicada Ca -> aaC)
AaaDB (" CB -> DB)
AaDaB (" aD -> Da)
ADaaB (" aD -> Da)
ACaaB (" AD -> AC)
AaaCaB (" Ca -> aaC)
AaaaaCB (" Ca -> aaC)
AaaaaE (" CB -> E)
AaaaEa (" aE -> Ea)
AaaEaa (" aE -> Ea)
AaEaaa (" aE -> Ea)
AEaaaa(" aE -> Ea)
εaaaa (" AE -> ε)
aaaa

* Ejercicio 18 * (pág.28)
S
aSb
aaSbb
aaaSbbb
aaaabbbb No es una palabra del LIC
No hay manera de producir una b más sin una a

* Ejercicio 19 * (pág.30)
G= ({S, T}, {a, b, c}, {S -> Tb, T -> aTc, T -> abc}, S) o
G= ({S, T}, {a, b, c}, {S -> Tb, T -> aTc, T -> b}, S)

* Ejercicio 20 * (pág.30)
a) aaabcccb
S
Tb
aTcb
aaTccb
aaaTcccb
aaabcccb Es palabra del lenguaje
b) aabbccb
S
Tb

aTcb
aaTccb
¿? No hay manera de producir dos bes entre la a y la c, por lo tanto, no es palabra del lenguaje
c) aaabcccbb
S
Tb
aTcb
aaTccb
aaaTcccb
¿? No hay manera de producir una segunda b después de la última c, por lo tanto, no es una palabra del lenguaje
d) aaccb
S
Tb
aTcb
aaTccb
¿? No hay manera de no producir la b entre la a y la c, por lo tanto, no es palabra del lenguaje

* Ejercicio 21 * (pág.30)
GR = ({S, T}, {a,b,c,d,2,3,4,5,6}, {S –> a | b | c | d | aT | bT | cT | dT,
T –> aT | bT | cT | dT | 2T | 3T | 4T | 5T | 6T | 2 | 3 | 4 | 5 | 6}, S)
La GQR es más sencilla para ser leída

* Ejercicio 22 * (pág.30)
a) ab23 (Derivación a izquierda)
S
SD
SDD
SLDD
LLDD
aLDD
abDD
ab2D
ab23 es una palabra válida
a') 2a3b (Derivación a izquierda)
S
SD
SDD
SLDD
?? no se puede seguir derivando porque S no produce D, por lo tanto, no es válida
b) ab23 (Derivación a derecha)
S
SD
S3
SD3
S23
SL
Sb23
Lb23
ab23 es una palabra válida
b') 2a3b (Derivación a derecha)
S
SL

Sb
SDb
S3b
SL3b
Sa3b
?? no se puede seguir derivando porque S no produce D, por lo tanto, no es válida

* Ejercicio 23 * (pág.31)
a) S
E;
T;
6; Correcta
b) S
E;
E + T; ¿? No hay manera de producir el terminal + en el extremo derecho.
Por lo tanto, no es correcta

c) S
E;
¿? No hay manera de producir el terminal + en el extremo derecho.
Por lo tanto, no es correcta
d) S
E;
E + T;
E + T + T;
E + T + T + T;
T + T + T + T;
6 + T + T + T;
6 + 6 + T + T;
6 + 6 + 6 + T;
6 + 6 + 6 + 2; Correcta

1.2.3 Capítulo 3

* Ejercicio 2 * (pág.34)
No se puede por la ambigüedad de la descripción de los Identificadores mediante una frase en Lenguaje Natural.

* Ejercicio 3 * (pág.34)
Hay otras. Investigue.

* Ejercicio 4 * (pág.35)
(Abreviamos los nombres de los noterminales: Id, Let y GB)
Id
Id GB Let
Id GB Let GB Let
Let GB Let GB Let
R GB Let GB Let
R_ Let GB Let
R_X GB Let
R_X_ Let
R_X_A

* Ejercicio 5 * (pág.35)
(Abreviamos los nombres de los noterminales: Id, Let y GB)
Id
Id GB Let
¿? No hay manera de producir dos guiones bajos consecutivos porque no hay una producción que sea: Identificador -> Identificador GuiónBajo

* Ejercicio 6 * (pág.36)
(Abreviamos los nombres de los noterminales: Id, Let, Res y GB)
Id -> Let | Id Let | Id Res
Res -> GB Let
GB -> _
Let -> 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
Tiene una producción más

* Ejercicio 7 * (pág.36)
En base al Ejemplo 3 de la página 36
(Abreviamos los nombres de los noterminales: Id, Let, Res y GB)

Derivación Vertical a Derecha	**Comentario: Se aplica la producción**
Id	Id −> Let Res
Let Res	Res −> GB Let Res
Let GB Let Res	Res −> GB Let Res
Let GB Let GB Let Res	Res −> ε
Let GB Let GB Let	Let −> A
Let GB Let GB A	GB −> _
Let GB Let _A	Let −> X
Let GB X_A	GB −> _
Let _X_A	Let −> R
R_X_A	

* Ejercicio 8 * (pág.36)
(Abreviamos los nombres de los noterminales: Id, Let, Res y GB)
Id
Let Res
A Res
A GB Let Res
A_ Let Res
¿? No hay manera de producir dos guiones seguidos.

* Ejercicio 9 * (pág.37)
Expresión -> Término |
Expresión + Término
Término -> Factor |
Término * Factor
Factor -> Número |
(Expresión)
Número -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

* Ejercicio 10 * (pág.38)
Por ejemplo:
Expresión

Término
Factor
Número
4

* Ejercicio 11 * (pág.38)
(Abreviamos los nombres de los noterminales: Exp, Tér, Fac y Num)

PRODUCCIÓN APLICADA	CADENA DE DERIVACIÓN OBTENIDA
(axioma)	Exp
1	Tér
3	Fac
6	(Exp)
1	(Tér)
3	(Fac)
6	((Exp))
1	((Tér))
3	((Fac))
5	((Num))
7	((2))

* Ejercicio 12 * (pág.38)
Expresión
Expresión + Término
Expresión + Término + Término
¿? No hay manera de producir dos ++ consecutivos

* Ejercicio 13 * (pág.40)

CADENA DE DERIVACIÓN A REDUCIR	PRODUCCIÓN A APLICAR	OPERACIÓN
(1 + 2) * (3 + 4)	7	
(1 + 2) * (3 + Número4)	5	
(1 + 2) * (3 + Factor4)	3	
(1 + 2) * (3 + Término4)	7	
(1 + 2) * (Número3 + Término4)	5	
(1 + 2) * (Factor3 + Término4)	3	
(1 + 2) * (Término3 + Término4)	1	
(1 + 2) * (Expresión3 + Término4)	2	3 + 4 = 7
(1 + 2) * (Expresión7)	6	
(1 + 2) * Factor7	7	
(1 + Número2) * Factor7	5	
(1 + Factor2) * Factor7	3	
(1 + Término2) * factor7	7	
(Número1 + Número2) * Factor7	5	
(Factor1 + Término2) * Factor7	3	
(Término1 + Término2) * Factor7	1	
(Expresión1 + Término2) * Factor7	2	1 + 2 = 3
(Expresión3) * Factor7	6	
Factor3 * Factor7	3	
Término3 * Factor7	4	3 * 7 = 21
Término21	1	
Expresión	(axioma)	Resultado Final

* Ejercicio 14 * (pág.40)
G = ({E, N}, {+, *, (,), 1, 2, 3, 4, 5}, {E −> E + E, E −> E * E, E −> (E), E −> N, N −> 1|2|3|4|5}, E)

ACLARACIÓN: los dos ejemplos que se presentan a continuación son la FE DE ERRATAS de los ejemplos de las páginas 40-41, del Vol.1)

* Ejemplo 9 *
Corrección a la Tabla de Evaluación:

PRODUCCIÓN APLICADA	CADENA DE DERIVACIÓN OBTENIDA
(axioma)	E
1	E + E
2	E * E + E
1	E + E * E + E
4	N + E * E + E
5	1 + E * E + E
4	1 + N * E + E
5	1 + 2 * E + E
3	1 + 2 * (E) + E
1	1 + 2 * (E + E) + E
4	1 + 2 * (N + E) + E
5	1 + 2 * (3 + E) + E
4	1 + 2 * (3 + N) + E
5	1 + 2 * (3 + 4) + E
4	1 + 2 * (3 + 4) + N
5	1 + 2 * (3 + 4) + 5

* Ejemplo 10 * (pág.41)
Corrección a la Tabla de Evaluación (derivación a izquierda):

CADENA DE DERIVACIÓN A REDUCIR	PRODUCCIÓN A APLICAR	OPERACIÓN
1 + 2 * (3 + 4) + 5	5	
1 + 2 * (3 + 4) + N.5	4	
1 + 2 * (3 + 4) + E.5	5	
1 + 2 * (3 + N.4) + E.5	4	
1 + 2 * (3 + E.4) + E.5	5	
1 + 2 * (N.3 + E.4) + E.5	4	
1 + 2 * (**E.3** + **E.4**) + E.5	1	3 + 4 = 7
1 + 2 * (E.7) + E.5	3	
1 + 2 * E.7 + E.5	5	
1 + N.2 * E.7 + E.5	4	
1 + E.2 * E.7 + E.5	5	
N.1 + E.2 * E.7 + E.5	4	
E.1 + **E.2** * E.7 **+** E.5	1	1 + 2 = 3
E.3 * E.7 + E.5	2	3 * 7 = 21
E.21 + **E.5**	1	21 + 5 = 26
E.26	(axioma)	Resultado Final

* Ejercicio 16 * (pág.42)
SÍ.

* Ejercicio 17 * (pág.43)
Significa la producción vacía, es decir, que no produce nada.

* Ejercicio 18 * (pág.44)
(Abreviamos los nombres de los noterminales: <número entero> <NE>, <entero sin signo> <ESS>, <dígito> <D>)
<NE>
- <ESS>
- <ESS> <D>
- <ESS> <D> <D>
- <ESS> <D> <D> <D>
- <ESS> <D> <D> <D> <D>
- <ESS> <D> <D> <D> <D> <D>
- <D> <D> <D> <D> <D> <D>
- 1 <D> <D> <D> <D> <D>
-12 <D> <D> <D> <D>
-123 <D> <D> <D>
-1234 <D> <D>
-12345 <D>
-123456

* Ejercicio 19 * (pág.45)
(a) <identificador>: “Un identificador debe comenzar obligatoriamente con una letra y puede o no estar seguida de una secuencia de una o más letras o dígitos (en cualquier orden y cantidad).”

(b) <identificador> ::= <letra> |
<identificador> <letra> |
<identificador> <dígito>
<letra> y <dígito> quedan iguales

* Ejercicio 20 * (pág.46)
(a) {**.**, **E**, **+**, **-**, **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**}
(b) {**<>**, **::=**, **|**, **{}** }
(c) 70.70, 70.70E7, 70.70E+7, 70.70E-7, 7000E7, 7000E+7, 7000E-7

* Ejercicio 21 * (págs.46-47)
(a) subrayadas
(b) 14
{**.**, program, **(**, **)**, **:=**, **;**, **,**, const, **=**, var, **:**, begin, end}
(c) Significa sentencia vacía
(d) 5
(e) SÍ.
<sentencia compuesta>
begin <sentencia> end
begin <sentencia simple> end
begin <sentencia vacía> end
begin <vacío> end
begin end
(f) Infinitas

* Ejercicio 22 * (pág.47)
(1) begin
a := 3
end;
(2) begin
readln (a);

```
      readln (b);
      writeln (a+b)
    end;
```

* Ejercicio 23 * (pág.47)
+ – or
* / and

* Ejercicio 24 * (pág.48)
No se puede.

* Ejercicio 25 * (pág.49)
No. Son derivables pero no son sintácticamente correctas.

* Ejercicio 26 * (pág.50)
No se puede.

* Ejercicio 27 * (pág.51)
LRs: *identificador*, *constantes numéricas* y *literalCadena* son infinitos; los restantes son finitos.

* Ejercicio 29 * (pág.52)
programaC líneas 1 a 17
noC líneas 1 y 2
prototipo línea 3
main líneas 4 a 11
función líneas 12 a 17
etc.

* Ejercicio 31 * (pág.54)
85

* Ejercicio 32 * (pág.54)

LEXEMA	TOKEN
double	palabraReservada
XX	identificador
(	carácterPuntuación
double	palabraReservada
a	identificador
,	carácterPuntuación
Int	palabraReservada
b	identificador
)	carácterPuntuación
{	carácterPuntuación
while	palabraReservada
(	carácterPuntuación
a	identificador
>	operador
b	identificador
)	carácterPuntuación
b	identificador
++	operador
;	carácterPuntuación
return	palabraReservada
b	identificador
;	carácterPuntuación
}	carácterPuntuación

* Ejercicio 36 (pág.56)
b) NO. ¿por qué?

* Ejercicio 37 * (pág.56)
(a) un dígito no cero (como cadena) o 1 (como valor)
(b) se representa con cero (si bien el cero es octal, coincide con la constante decimal cero)

* Ejercicio 38 * (pág.56)
(a) octal, decimal, hexadecimal, octal, decimal, octal
(b) (Abreviamos los nombres de los noterminales: constanteHexadecimal cH, dígitoHexadecimal dH)
cH
cH dH
cH dH dH
cH dH dH dH
0X dH dH dH dH
0Xa dH dH dH
0Xa4 dH dH
0Xa4b dH
0Xa4b8

* Ejercicio 39 * (pág.57)
(b) SÍ porque es derivable
(Abreviando los noterminales: *constanteReal* CR, *constanteFraccionaria* CF, *secuenciaDígitos* SD, *dígito* d)
CR

CF

SD.

SD d.

SD d d.

d d d.

4 d d.

42 d.

425.

* Ejercicio 40 * (pág.57)

(a) **0. .0 0E0 0e0 0E+0 0e+0**

(b) (Abreviando los noterminales: SD secuenciaDígitos, OE operadorE, S signo)

FORMATO REAL	EJEMPLO
SD.	13.
SD.f	13.f
SD.F	13.F
SD.l	13.l
SD.L	13.L
.SD	.42
.SDf	.42f
.SDF	.42F
.SDl	.42l
.SDL	.42L
SD.OE SD	23.e2
SD.OE SD	23.E2
SD.OE SDf	23.e2f
SD.OE SDF	23.E2F
SD.OE SDl	23.e2l
SD.OE SDL	23.E2L
SD.OE S SD	50.e+6
SD.OE S SD	50.E+6
SD.OE S SD	50.e-3
SD.OE S SD	50.E-3
SD.OE S SDf	43.e+2f
SD.OE S SDf	43.E-2f
SD.OE S SDF	43.E+2F
SD.OE S SDF	43.e-2F
SD.OE S SDl	31.e+3l
SD.OE S SDL	31.E+3L
SD.OE S SDl	31.e-3l
SD.OE S SDL	31.e-3L
.SD OE SD	.120E2
.SD OE SD	.120e2
.SD OE SDf	.79E2f
. SD OE SDF	.79e2F
SD OE SD	205E8
SD OE SD	205e8
SD OE SDf	205e8f
SD OE SDL	205E8F
SD OE S SD	1000e+4
SD OE S SD	1000E-4
SD OE S SDf	36E+5f
SD OE S SDL	36E-5L
Faltan combinaciones	. . . . .

* Ejercicio 41 * (pág.57)
(Abreviando los noterminales: CR *constanteReal*, CF *constanteFraccionaria*, PE *parteExponenete*, SD *secuenciaDígitos*, S *signo*, OE *operadorE*)
CR
CF PE
SD.SD PE
D.SD PE
4.SD PE
4.D PE
4.6 PE
4.6 OE S SD
4.6E S SD
4.6E- SD
4.6E- SD D
4.6E- D D
4.6E-2 D
4.6E-23

* Ejercicio 44 * (pág. 57)
(e) literalCadena: “dígito dígito secuenciaDígitos” |
“”

secuenciaDígitos: dígito |
secuenciaDígitos dígito

* Ejercicio 51 * (pág.61)
expPostfijo
expPostfijo [expresión]
expPostfijo [expresión] [expresión]
expPrimaria [expresión] [expresión]
identificador [expresión] [expresión]
identificador dígito [expresión] [expresión]
identificador dígito dígito [expresión] [expresión]
identificador noDígito dígito dígito [expresión] [expresión]
identificador noDígito noDígito dígito dígito [expresión] [expresión]
identificador noDígito noDígito noDígito dígito dígito [expresión] [expresión]
identificador noDígito noDígito noDígito noDígito dígito dígito [expresión] [expresión]
identificador nodígito noDígito noDígito noDígito noDígito dígito dígito [expresión] [expresión]
noDígito nodígito noDígito noDígito noDígito noDígito dígito dígito [expresión] [expresión]
m nodígito noDígito noDígito noDígito noDígito dígito dígito [expresión] [expresión]
ma noDígito noDígito noDígito noDígito dígito dígito [expresión] [expresión]
mat noDígito noDígito noDígito dígito dígito [expresión] [expresión]
matr noDígito noDígito dígito dígito [expresión] [expresión]
matri noDígito dígito dígito [expresión] [expresión]
matriz dígito dígito [expresión] [expresión]
matri1 dígito [expresión] [expresión]
matriz12 [expresión] [expresión]
matriz12 [expAsignación] [expresión]
matriz12 [expCondicional] [expresión]
matriz12 [expOr] [expresión]
matriz12 [expAnd] [expresión]
matriz12 [expIgualdad] [expresión]
matriz12 [expRelacional] [expresión]
matriz12 [expAditiva] [expresión]

matriz12 [expAditiva +expMultiplicativa] [expresión]
matriz12 [expMultiplicativa + expMultiplicativa] [expresión]
matriz12 [expUnaria + expMultiplicativa] [expresión]
matriz12 [expPostfijo + expMultiplicativa] [expresión]
matriz12 [expPrimaria expMultiplicativa] [expresión]
matriz12 [constante + expMultiplicativa] [expresión]
matriz12 [2 + expMultiplicativa] [expresión]
matri12 [2+ expUnaria] [expresión]
matriz12 [2+ expPostfijo] [expresión]
matriz12 [2+ expPrimaria] [expresión]
matriz12 [2+ constante] [expresión]
matriz12 [2+3] [expresión]
matriz12 [2+3] [expAsignación]
matriz12 [2+3] [expCondicional]
matriz12 [2+3] [expOr]
matriz12 [2+3] [expAnd]
matriz12 [2+3] [expIgualdad]
matriz12 [2+3] [expRelacional]
matriz12 [2+3] [expAditiva]
matriz12 [2+3] [expMultiplicativa]
matriz12 [2+3] [expMultiplicativa * expUnaria]
matriz12 [2+3] [expUnaria * expUnaria]
matriz12 [2+3] [expPostfijo * expUnaria]
matriz12 [2+3] [expPrimaria * expUnaria]
matriz12 [2+3] [constante * expUnaria]
matriz12 [2+3] [4* expUnaria]
matriz12 [2+3] [4* expPostfijo]
matriz12 [2+3] [4* expPrimaria]
matriz12 [2+3] [4* constante]
matriz12 [2+3] [4*6]

* Ejercicio 52 * (pág. 61)
Infinitas

* Ejercicio 54 * (pág.62)
Expresión
expAsignación
expUnaria operAsignación expAsignación
expPostfijo operasignación expAsignación
expPrimaria operAsignación expAsignación
constante operAsignación expAsignación
1 operAsignación expAsignación
1 = expAsignación
1 = expCondicional
1 = expOr
1 = expAnd
1 = expIgualdad
1 = expRelacional
1 = expAditiva
1 = expMultiplicativa
1 = expUnaria
1 = expPostfijo
1 = expPrimaria

1 = constante
1 = 2

* Ejercicio 56 * (pág.62)
NO es sintácticamente correcta, es DERIVABLE. Desde el punto de vista del programador no tiene sentido.

* Ejercicio 57 * (pàg.62)
NO es sintácticamente correcta, es DERIVABLE. Desde el punto de vista del programador no tiene sentido.

* Ejercicio 58 * (pág.62)
El lenguaje Pascal tiene alrededor de 20 operadores y ANSI C tiene 45 operadores.
En Pascal, el and tiene la misma prioridad que la multiplicación; mientras que en ANSI C, el **&&** tiene menor prioridad que la multiplicación; etc.

* Ejercicio 60 * (pág.64)
declaVarSimples
tipoDato listaVarSimples **;**
int listaVarSimples **;**
int listaVarSimples **,** unaVarSimple **;**
int listaVarSimples **,** unaVarSimple **,** unaVarSimple **;**
int listaVarSimples **,** unaVarSimple **,** unaVarSimple **,** unaVarSimple **;**
int unaVarSimple **,** unaVarSimple **,** unaVarSimple **,** unaVarSimple **;**
int variable **,** unaVarSimple **,** unaVarSimple **,** unaVarSimple **;**
int identificador **,** unaVarSimple **,** unaVarSimple **,** unaVarSimple **;**
int a, unaVarSimple **,** unaVarSimple **,** unaVarSimple **;**
int a, variable inicial **,** unaVarSimple **,** unaVarSimple **;**
int a, b inicial **,** unaVarSimple **,** unaVarSimple **;**
int a, b = constante **,** unaVarSimple **,** unaVarSimple **;**
int a, b = 10, unaVarSimple **,** unaVarSimple **;**
int a, b = 10, variable **,** unaVarSimple **;**
int a, b = 10, identificador **,** unaVarSimple **;**
int a, b = 10, c, unaVarSimple **;**
int a, b = 10, c, variable inicial **;**
int a, b = 10, c, identificador inicial**;**
int a, b = 10, c, d inicial **;**
int a, b = 10, c, d = constante **;**
int a, b = 10, c, d = 4;

* Ejercicio 61 * (pág.64)
declaraTipo: *tipo estructura* {*listaCampos*} *nombreTipo* **;**
tipo: **typedef**
estructura: **struct**
listaCampos: *unCampo*
listaCampos **;** *unCampo*
unCampo: *tipoDato listaVarSimples* **;**
tipoDato: *uno de* **int char double**
listaVarSimples: *unaVarSimple*
listaVarSimples **,** *unaVarSimple*
unaVarSimple: *variable*
variable: *identificador*
nombreTipo: *identificador*

identificador: *noDígito*
identificador noDígito
identificador dígito
noDígito: *uno de* _ **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: *uno de* **0 1 2 3 4 5 6 7 8 9**

* Ejercicio 62 * (pág.64)
(b) Derivable.
(c) Sintácticamente correcto.

* Ejercicio 63 * (pág.65)
(a) SÍ porque en la BNF:
sentCompuesta: **{***listaDeclaraciones*$_{op}$ *listaSentencias*$_{op}$**}** esto implica que puede escribirse **{}**
(b) Es una sentencia compuesta vacía porque lo encerrado entre las llaves es un comentario y cada comentario es reemplazado por un blanco por el Preprocesador.

* Ejercicio 66 * (pág.65)
(a) Sí ; (b) Sí ; (c) dos: **14;** y **;** (sentencia vacía)

* Ejercicio 69 * (pág.65)
En Pascal la expresión tiene que ser booleana y los paréntesis son opcionales. En ANSI C puede ser cualquier expresión y los paréntesis son obligatorios. Además, en Pascal es if-then-else y en ANSI C es if-else.
En Pascal:
<sentencia if> ::= if <expresión> then <sentencia> |
if <expresión> then <sentencia> else <sentencia>
En ANSI C:
sentSelección: **if (** *expresión* **)** *sentencia*
if (*expresión* **)** *sentencia* **else** *sentencia*

* Ejercicio 73 * (pág.65)
No se ejecutará ninguna sentencia. Es sintácticamente correcto.

* Ejercicio 74 * (pág.66)
(a) SÍ.
sentIteración
for (; *expresión* **;)** *sentencia*
for (; *expAsignación* **;)** *sentencia*
for (; *expCondicional* **;)** *sentencia*
for (; *expOr* **;)** *sentencia*
for (; *expAnd* **;)** *sentencia*
for (; *expIgualdad* **;)** *sentencia*
for (; *expRelacional* **;)** *sentencia*
for (; *expAditiva* **;)** *sentencia*
for (; *expMultiplicativa* **;)** *sentencia*
for (; *expUnaria* **;)** *sentencia*
for (; operUnario *expUnaria* **;)** *sentencia*
for (; - *expUnaria* **;)** *sentencia*
for (; - *expPostfijo* **;)** *sentencia*
for (; - *expPrimaria* **;)** *sentencia*
for (; - *expUnaria* **;)** *sentencia*
for (; - *constante* **;)** *sentencia*

. . . . .
for (; -*4* **;)** *sentencia*
for (; -*4* **;)** *sentExpresión*
for (; -*4* **;) ;**

(b) SÍ.
(c) Es un ciclo infinito porque su segunda expresión siempre será verdadera.

* Ejercicio 75 * (pág.66)
Es un ciclo infinito que no hace nada.

* Ejercicio 76 * (pág.66)
(a) SÍ.
sentSalto
return *expresión* **;**
return *expresión* **;**
return *expUnaria operAsignación expasignación* **;**
return *expPostfijo operAsignación expAsignación* **;**
return *expPrimaria operAssignación expAsignación* **;**
return *identificador operAsignación expAsignación* **;**
. . . .
return a *operAsignación expAsignación* **;**
return a = *expAsignación* **;**
return a = *expCondicional* **;**
return a = *expOr* **;**
return a = *expAnd* **;**
return a = *expIgualdad* **;**
return a = *expRelacional* **;**
return a = *expAditiva* **;**
return a = *expMultiplicativa* **;**
return a = *expUnaria* **;**
return a = *expPostfijo* **;**
return a = *expPrimaria* **;**
return a = *constante* **;**
. . . . .
r**eturn a = 8;**

(b) Sí, es derivable y sintácticamente correcta.
(c) Le asigna el valor 8 a la variable a y retorna ese valor.

* Ejercicio 77 * (pág.66)
(a) Archivo: Secuencia | Archivo Secuencia
Secuencia: Dígitos FIN
Dígitos: Dígito | Dígitos Dígito
FIN: #
Dígito: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
(b) Archivo: Secuencia | Archivo Secuencia
Secuencia: Dígitos FIN | FIN
Dígitos: Dígito | Dígitos Dígito
FIN: #
Dígito: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

* Ejercicio 78 * (pág.66)
(a) Suponemos que secuenciaCompuesta y expresiónBooleana son terminales para no tener que definirlas.

(c) sentenciaSelección: **SI** Constructos **FIN**
Constructos: Constructo1 Constructo2
Constructo1: expresiónBooleana **: {** secuenciaCompuesta **}**
Constructo1 expresiónBooleana **: {** secuenciaCompuesta **}**
Constructo2: **OTRO : {** secuenciaCompuesta **}**

1.3 Semántica (Pratt)

Terrence Pratt: “Programming Languages – Design and Implementation, 2nd edition”, Prentice Hall

1.3.1 Pratt p.26

La SINTAXIS de un LP es la forma en la cual los programas son escritos.

La SEMÁNTICA de un LP es el significado otorgado a los diferentes constructos sintácticos. Este significado tiene que ver con lo que sucede en tiempo de ejecución.

Ejemplo 1:
Sea en Pascal la declaración **`V: array [1..10] of real;`**

Esta declaración brinda la sintaxis de un vector de 10 elementos reales llamado V.

La semántica de esta declaración (el significado de la misma) es, por ejemplo: esta declaración se coloca al comienzo de un subprograma para crear, en cada invocación a ese subprograma, el vector mencionado, cuyo espacio será liberado al terminar de ejecutarse el subprograma.

Ejemplo 2:
Analicemos una situación en ANSI C. Sea la sentencia **`while`**.

Su sintaxis es: **`while (`** *`expresión`* **`)`** *`sentencia`*

Su semántica, según el MROC, dice: la evaluación de la expresión de control se realiza antes de cada ejecución del cuerpo del ciclo. Entonces, el cuerpo del ciclo es ejecutado repetidamente hasta que el expresión sea nula (0, 0.0, etc).

Ejemplo 3:
En ANSI C, la semántica de **`while (2) 3;`** representa un ciclo infinito.

1.3.2 Pratt p.345

Problemas en Semántica

Problema 1: Definición de Semántica

El problema práctico. Un manual de un LP debe definir el significado de cada construcción del lenguaje, tanto en forma aislada como en conjunción con otras construcciones del lenguaje. Un lenguaje provee una variedad de diferentes constructos, y tanto el usuario del lenguaje como el implementador requieren una definición precisa de la semántica de cada constructo. El programador necesita la definición para escribir programas correctos y para ser capaz de predecir el efecto de la ejecución de cualquier sentencia del programa. El implementador necesita la definición para poder construir una implementación correcta del LP.

En la mayoría de los manuales de los LPs la definición de la semántica está dada en lenguaje natural. Comúnmente, una producción (o producciones) de una BNF son dadas para definir la

sintaxis de un constructo, y luego unos párrafos en lenguaje natural y algunos ejemplos son dados para definir la semántica.

1.3.3 Pratt p.317

Análisis Semántico

Completa el Análisis Sintáctico y, además, puede producir el código objeto (generalmente para una MV).

El Análisis Semántico se lleva a cabo a través de un conjunto de rutinas, cada una de las cuales se ocupa de una actividad o de un constructo.

Ejemplo 4:
Las declaraciones de los arreglos pueden ser manejadas por una rutina semántica; las expresiones aritméticas por otra, etc. Una rutina semántica apropiada es llamada por el Parser cada vez que éste reconoce un constructo que debe ser procesado.

Una rutina semántica puede actuar conjuntamente con otras rutinas semánticas para llevar a cabo una tarea determinada.

1.3.4 Pratt p.318

Mantenimiento de la TS

Una TS es una estructura central en todo compilador. Una TS típica contiene una entrada para cada identificador que existe en el programa fuente. El Scanner coloca el identificador detectado, pero el Análisis Semántico (AS) tiene la responsabilidad fundamental después de ello.

En general, la TS no solo contiene a los identificadores sino que también contiene los atributos de cada uno de ellos: su clase (variable simple, arreglo, función, parámetro, etc.), tipo de los valores (entero, real, etc.), alcance, otra información que proveen las declaraciones (y definiciones), etc.

El AS, a través de sus diferentes rutinas, ingresa esta info en la TS a medida que procesa declaraciones, encabezamientos de funciones y sentencias del programa.

Otras rutinas del AS utilizan esta info para construir, luego, códigos ejecutables eficientes.

//