Universidad Autónoma de Madrid
Departamento de Ingeniería Informática
3er. Curso 1er. Cuatrimestre
Autómatas y Lenguajes

Enunciados sobre la unidad 1 modelos cómputo Hoja 3

GRAMÁTICAS DE ATRIBUTOS

1.- Se está diseñando un lenguaje de programación en el que se permite definir "familias" de variables añadiendo al identificador que representa el nombre de la familia un subíndice que puede tomar cualquier valor entre un mínimo y un máximo. A continuación se presenta algún ejemplo de la notación que se quiere utilizar:

```
nodo:i|i∈{1,2}
```

Que representa a los siguientes nodos:

```
nodo:1, nodo:2
```

Supondremos que uno de los tipos de datos básicos que soporta el lenguaje de programación es el tipo entero. Será el utilizado como tipo de los nodos en este enunciado. A las variables así definidas se les puede asignar una constante numérica (por lo tanto de tipo entero). A continuación se muestra un ejemplo de este tipo de asignaciones:

```
Nodo:2 = 10
```

La siguiente gramática genera programas que incluyen estas dos características recién expuestas:

- Puede declararse una familia de variables
- Se puede realizar una serie de asignaciones de constante a variables bien definidas

```
P  → N ; S
N → Var : Indice "|" Indice ∈ { Num , Num }
S  → A ; S
S → A ;
A  → Var : Num = Num
```

No se proporcionan reglas para las constantes numéricas y los nombres de variable e índices (representados respectivamente por los símbolos no terminales `Num`, `Var` e `Indice`) . Suponga que las constantes numéricas siguen la notación matemática habitual y que los nombres de variable e índices válidos son cualquier cadena de caracteres alfanuméricos.

Puede suponer que, esos símbolos contienen como información semántica la siguiente

- `Var.nombre`, la cadena de caracteres que es el nombre de la variable
- `Indices.nombre`, idem. al caso anterior
- `Num.valor`, valor entero del número

Se desea ampliar esta gramática con un sistema de atributos que permita controlar los siguientes aspectos:

- El nombre de la familia de variables de la declaración es el mismo que luego aparece en las sentencias de asignación.
- El valor concreto del índice que aparece en cada asignación está dentro del rango indicado en la declaración

Para ello se pide:

Añadir a la gramática anterior un sistema de atributos para conseguir el objetivo descrito. Para ello completa los siguientes puntos (supón que puede utilizarse cualquier tipo de propagación; sepa, también, que las soluciones que utilicen síntesis y herencia serán más valoradas que las que sólo utilicen síntesis y e información global):
1.- Describe explícita y brevemente el significado de cada nuevo atributo que utilices así como el proceso de actualización de cada uno de ellos en las reglas de la gramática.
2.- Define formalmente la gramática utilizando la notación explicada en el temario de esta asignatura.
3.- Muestre como ejemplo el árbol de derivación de la siguiente cadena añadiendo en él los valores de los atributos y alguna indicación gráfica (por ejemplo flechas) que muestren la propagación de sus valores

```
v:i|i∈[1,10];v:1=10;v:3=2;
```

SOLUCIÓN
Se va a utilizar el siguiente conjunto de atributos:

- var, ese una cadena de caracteres que guarda el nombre de la familia de variables declaradas. Lo tienen tanto N como S y A, en el caso de N es sintetizado, en el caso de A y S es heredado
- min y max son enteros que guardan el valor máximo y mínimo del índice declarado para la familia de variables. Lo tienen también N, S y A y se actualiza de la misma manera

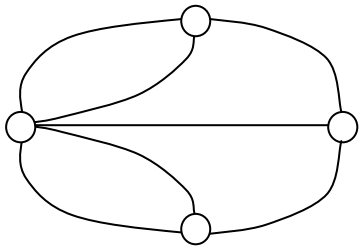
La manera en la que se soluciona el problema es la siguiente: en el subárbol cuya raíz es N, se sintetiza los valores de var, min y max. Luego lo hereda S en la regla del axioma y lo trasmite por herencia a sus hijos S y A para que, cuando A termina mediante la aplicación de la regla que genera una variable concreta, pueda comparar el valor del índice de la declaración con los mínimos y máximos de su rango y que el nombre de la variable coincide con la declarada.
A continuación se muestran las reglas, se utilizará el siguiente convenio de colores: síntesis y herencia.

```
P  → N ; S
1:{
      S.var = N.var;
      S.min = N.min;
      S.max = N.max;
}
N → Var : Indice1 "|" Indice2 ∈ { Num1 , Num2 }
11:{
      SI (Indice1.nombre != Indice2.nombre) O
         (Num1.valor >= Num2.valor)
      {
            ERROR "Inconsistencia en declaración del índice" O
                  "Rango inconsistente";
      }
      OTRO CASO
      {
            N.var = Var.nombre;
            N.min = Num1.valor;
            N.max = Num2.valor;
      }
}
S1  → A ; S2
0:{
      S2.var = S1.var;
      S2.min = S1.min;
      S2.max = S1.max;
      A.var = S1.var;
      A.min = S1.min;
      A.max = S1.max;
}
```

```
S → A ;
0:{
      A.var = S.var;
      A.min = S.min;
      A.max = S.max;
}
A  → Var : Num1 = Num2
5:{
      SI (A.var <> Var.nombre) O (Num1 < A.min) O (Num1 < A.max)
            "Error, id inválido o índice fuera de rango"
}
```

2.- Los Siete Puentes de Königsberg constituyen un problema matemático notable [Wiki] Leonhard Euler proporcionó en 1735 una solución negativa en lo que constituye el origen de la teoría de grafos y un antecedente de la Topología.

La ciudad de Königsberg en Prusia (actualmente Kaliningrado, Rusia) estaba situada a ambos lados del río Pregel, e incluía dos amplias islas conectadas entre sí y con el resto de la ciudad mediante siete puentes de la manera que indica el siguiente grafo.



El problema que estudió Euler consistía en probar si era posible encontrar un camino en este grafo que recorriera todos los arcos exactamente una vez. Ese tipo de caminos se ha terminado llamando "caminos eulerianos".

En su solución (que en el caso de la ciudad de Königsberg mostraba que este camino no existía) utilizó el concepto de grado de un nodo.

El *grado de un nodo* es, informalmente, el número de arcos que lo tocan.

Euler y posteriormente otros investigadores, demostraron que una condición necesaria y suficiente para la existencia de un camino euleriano en un grafo es que éste sea conexo (cada par de nodos está conectado por un camino) y tenga exactamente 0 o 2 nodos de grado impar.

La siguiente gramática independiente del contexto genera grafos con la siguiente notación.

- Los vértices se representan mediante el carácter `n` seguido de un valor numérico que los identifica, como por ejemplo

  `n1 n2 n3 n4`
- Los valores numéricos son representados mediante el símbolo `Num` para el que no se proporcionan reglas.
- Los arcos se representan como pares de vértices, como por ejemplo
  `(n1, n2) (n1, n2) (n2, n4) (n2, n4) (n1 n3) (n2 n3) (n4 n3)`
- Vértices y arcos están separados por el símbolo "`:`"

Así pues, el grafo de los puentes de la ciudad de Könisgberg podría representarse así

```
n1 n2 n3 n4 : (n1,n2) (n1,n2) (n2,n4) (n2,n4) (n1,n3) (n2,n3) (n4,n3)
```

```
G  → Vs : As
Vs → V Vs | V
V  → nNum
As → A As | A
A  → (V,V)
```

Observación, puede omitir, por su comodidad, cualquier aspecto de la gestión de los espacios en blanco

Se quiere ampliar esta gramática con un sistema de atributos para que, tras procesar las cadenas que la gramática genera, se determine si el grafo que representan tiene algún camino euleriano.

Para ello se va a suponer que los diferentes símbolos de la gramática tienen un atributo de nombre `información_nodos` que es una lista de elementos que contienen la siguiente información:

- Nombre (de un vértice del grafo)
- Grado (el grado de ese vértice en el grafo)

Este atributo `información_nodos` se va a sintentizar en el subárbol cuya raíz es `Vs` (el de la regla `G → Vs : As`)
Para ello puede utilizar, por ejemplo, una función para insertar un elemento en la lista con su grado como la siguiente (en el ejemplo se generaría una nueva lista a partir del primer argumento de la función en la que se ha añadido a ella el nuevo elemento representado entre paréntesis en el segundo argumento de la función)

nueva_lista = insertar(l, `(n3,0)`)

Puede representar una lista vacía mediante dos paréntesis: "()"
Observe que cada nodo sólo puede ser generado una vez.
Este atributo se va a propagar por herencia al subárbol cuya raíz sea el símbolo `As` y se va a hacer llegar por herencia a lo largo del subárbol para realizar las siguientes tareas:

- Comprobar que el nodo del arco ha sido generado
    - Para ello puede suponer una función como la siguiente, que devuelve un valor booleano tras comprobar que el primer argumento, que representa un nodo, está incluido en algún elemento de la lista que es el segundo argumento

      pertenece(n3, lista)

    - En el momento en el que un nodo de un arco no pertenezca a la lista, puede terminar el proceso de la cadena que lo contiene indicando esa circunstancia mediante un mensaje

- Actualizar el grado de los nodos que pertenecen al arco. Puede utilizar tantos atributos y variables auxiliares y funciones auxiliares de manipulación de la lista como necesite para hacer esta operación correctamente. Puede expresar en seudocódigo de más alto nivel el proceso que desea realizar.

Al final del proceso de la cadena completa debe mostrar en un mensaje si su grafo contiene algún camino euleriano, es decir, si tiene exactaente 0 o 2 nodos con grado impar.
Para determinar si un número es impar, puede suponer una función que devuelve un valor booleano de la comprobación de esa situación como por ejemplo

impar(valor)

Por su comodidad, puede omitir también las siguientes reglas
`V → nNum`
`A → (V,V)`
Y suponer que el subárbol de raíz `Vs` las hojas terminarán con el símbolo `V` y en el subárbol de raíz `As` con el símbolo `A`.
Así mismo puede suponer

- que los símbolos V tienen como información semántica un atributo "nombre" con el nombre del vértice (por ejemplo V.nombre puede valer n1)
- que los símbolos A tienen como información semántica dos atributos "v1" y "v2" con el nombre respectivamente del primer y el segundo vértice del arco.

Se pide:

Añadir a la gramática anterior un sistema de atributos para conseguir el objetivo descrito. Para ello completa los siguientes puntos (supón que puede utilizarse cualquier tipo de propagación y que no puede utilizar información global):

1.- Describe explícita y brevemente el significado de cada nuevo atributo que utilices así como el proceso de actualización de cada uno de ellos en las reglas de la gramática.
2.- Define formalmente la gramática utilizando la notación explicada en el temario de esta asignatura.
3.- Muestre como ejemplo el árbol de derivación de la siguiente cadena añadiendo en él los valores de los atributos y alguna indicación gráfica (por ejemplo flechas) que muestren la propagación de sus valores

```
n1 n2 n3 : (n1,n2) (n1,n3) (n2,n3)
```

SOLUCIONES:
Es suficiente con el atributo sugerido

Se omite el árbol de la última cuestión.

```
G  → Vs : As
1:{
      As.in = Vs.in;
}
Vs1 → V Vs2
1:{
      Si ( pertenece(Vs2.in, Vs1.in)
            "Error, nodo duplicado"
      ELSE
            Vs1.in = insertar(Vs2.in,(V.nombre,0));
}
Vs → V
1:{
      Vs.in = insertar((),(V.nombre,0));
}
V  → nNum
As1 → A As2
1:{ lista aux_lista, int i1, i2;

      Si ( pertenece(A.v1, As2.in) Y pertenece(A.v2, As2.in)  )
      {
            i1 = grado(A.v1, As1.in);
            i2 = grado(A.v2, As1.in);
            aux_lista = actualizar(As1.in, (A.v1,i1+1));
            As2.in = actualizar(aux_lista, (A.v2,i2+1));
      }
      ELSE
            "Error, algún nodo no está definido"
}
As → A
1:{ lista aux_lista, int i1, i2;

      Si ( pertenece(A.v1, As.in) Y pertenece(A.v2, As.in)  )
      {
            i1 = grado(A.v1, As.in);
            i2 = grado(A.v2, As.in);
            aux_lista = actualizar(As.in, (A.v1,i1+1));
            aux_lista = actualizar(aux_lista, (A.v2,i2+1));
            "Recorrer aux_lista calculando el número de elementos con
grado impar";
            Si es 0 o 2, mostrar "Sí existe camino euleriano" en caso
contrario mostrar "No existe camino euleriano";
      }
      ELSE
            "Error, algún nodo no está definido"

}
A  → (V,V)
```

3.- La siguente gramática representa números romanos menores que 10

```
Unidades → UnidadesMenores | IV | V UnidadesMenores | IX
UnidadesMenores → λ | UnidadesMenores I
```

Y la siguiente añade a un número romano menor que 10 una cadena con la estructura (suponga que los espacios en blanco se han omitido para su comodidad y que no hay que tenerlos en cuenta en ningún caso)

```
=I...I
```

```
S → Unidades = Is
```

```
Unidades → UnidadesMenores | IV | V UnidadesMenores | IX
UnidadesMenores → λ | UnidadesMenores I
```

```
Is → I Is  | I
```

Se quiere modificar esta gramática para construir otra de atributos en la que se comprueben o añadan las siguientes restricciones

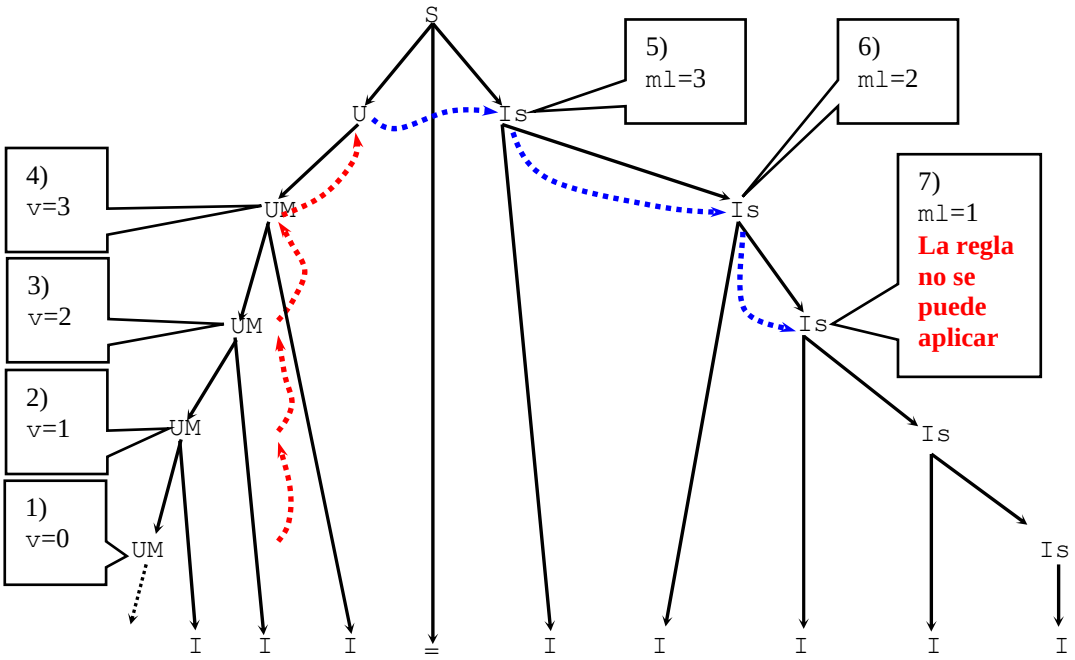
- El número romano es correcto, es decir, el número de simbolos `I` generado por `UnidadesMenores` es menor o igual que 3.
- El número de símbolos I de la segunda parte de la cadena es igual al valor decimal del número romano contenido en la primera parte de la cadena. La gramática debe informar mediante un mensaje en cuanto comprueba que no coincide con el valor de la primera. Tenga en cuenta que si utiliza herencia en alguno de los atributos involucrados en la segunda parte de la cadena para conseguir este objetivo se considerará que la solución es más adecuada.

Se pide:
Añadir a la gramática anterior un sistema de atributos para conseguir el objetivo descrito. Para ello completa los siguientes puntos (puede suponer que el recorrido del árbol de derivación para propagar atributos será en profundidad por la izquierda con retroseguimiento):
1.- Describe explícita y brevemente el significado de cada atributo que utilices así como el proceso de actualización de cada uno de ellos en las reglas de la gramática mencionando explícitamente si se realiza herencia o síntesis en la propagación de cada atributo.
2.- Define formalmente la gramática utilizando la notación explicada en el temario de esta asignatura. Para ello completa la plantilla que tienes a continuación.
3.- Muestre como ejemplo el árbol de derivación de la siguiente cadena añadiendo en él los valores de los atributos y alguna indicación gráfica (por ejemplo flechas) que muestren la propagación de sus valores

```
III:IIIII
```

1.- 2.-
Se utilizará un atributo numérico, `valor`, para calcular el valor del número romano. Se sintetiza en los símbolos `Unidades` y `UnidadesMenores`.
Así mismo, será sobre el valor de `UnidadesMenores` sobre el que se comprobará que no exceda de 3 en las reglas en las que se genera la secuencia de símbolos `III`.
Este valor lo heredará de su hermano izquierdo el símbolo `Is` en la regla del axioma y lo propagará por herencia de padre a hijo en la regla recursiva de `Is`, restándole uno siempre que quede algo que restar.
Cuidado. Como hay salida para `UnidadesMenores` por lambda y por `I`, hemos de permitir en la regla recursiva que queden al menos 0 puesto que lambda no resta. Hay que comprobar en la regla de salida por `I` que queda al menos un palote disponible.

```
S → Unidades = Is
1:{
      Is.max_lista = Unidades.valor;  /* Herencia hermano */
}

Unidades → UnidadesMenores
1:
      Unidades.valor = UnidadesMenores.valor;  /* Síntesis */
}

Unidades → IV
1:{ /* Suponiendo IV solo un símbolo */
      Unidades.valor = 4;           /* Síntesis */
}

Unidades → V UnidadesMenores
1:{
      Unidades.valor = 5 + UnidadesMenores.valor; /* Síntesis */
}

Unidades → IX
1:{     /* Suponiendo IX como un solo símbolo */
      Unidades.valor = 9;           /* Síntesis */
}

UnidadesMenores → λ
0:{
      /* Aquí no hay que comprobar nada ya que siempre se puede
      salir por lambda, no añade ninguna I a las que hubiera */
}
1:{
      Unidades.valor = 0;           /* Síntesis */
}

UnidadesMenores1 → UnidadesMenores2 I
2:{   Si (UnidadesMenores2.valor + 1 > 3 )
            ERROR ;
      Otro caso /* Síntesis */
            UnidadesMenores1.valor = UnidadesMenores2.valor + 1;
}


Is1 → I Is2
0:{
      Si ( Is1.max_listas <= 1 )
            /* Se sale siempre por I por lo que debe quedar al
            menos 1 */
            ERROR ;
      Otro caso
            /* Herencia del padre */
            Is2.max_listas = Is1.max_listas - 1;
}

Is → I
{  /* Ya se ha comprobado previamente que todo vaya bien */

}
```


S
U
Is
5)
ml=3
6)
ml=2
4)
v=3
UM
Is
7)
ml=1
La regla no se puede aplicar
3)
v=2
UM
Is
2)
v=1
UM
Is
1)
v=0
UM
Is
I
I
I
=
I
I
I
I
I

4.- La gramática independiente del contexto que puede deducirse de las siguientes reglas de producción (el axioma es el símbolo X) genera cadenas de parénesis en las que pueden distinguirse dos partes separadas por el símbolo "**:**" (puede ignorar los espacios en blanco a su conveniencia, es decir, se supondrá que pueden aparecer en los lugares que sean necesarios para facilitar la lectura)

X → D:L

D → UD | U
U → () | (D)

L → ()L | ()

La primera (generada por el símbolo no terminal D) genera una lista de listas que pueden contener a su vez listas en las que siempre se cumple que todos los paréntesis están balanceados. Estas listas podría representar la estructura de directorios de un disco.

Estos son algunos ejemplos de cadenas generadas por el no terminal D.

()
() () ()
((()) ()) () ((()))

La segunda (generada por el símbolo no terminal L) genera una secuencia de listas vacías ()()...() con al menos una lista.

Estos son algunos ejemplos de cadenas generadas por el no terminal L.

()
() ()
() () () () () ()

Se quiere diseñar a partir de esta gramática otra de atributos para generar cadenas cuya segunda parte represente "una especie de aplanamiento" de la primera, es decir, la subcadena que sigue al símbolo : tiene tantas listas vacías como listas hay en la primera parte y se garantice que éste número es inferior o igual a 64

A continuación se muestra algún ejemplo de las cadenas que podría generar la gramática de atributos

() (()()) (()):()()()()()()
():()
()(()):()()()

La gramática debe informar mediante un mensaje en cuanto comprueba que el número de listas de la segunda parte no coincide con el de la primera. Tenga en cuenta que si utiliza herencia en alguno de los atributos involucrados en la segunda parte de la cadena para conseguir este objetivo se considerará que la solución es más adecuada.

Se pide:
Añadir a la gramática anterior un sistema de atributos para conseguir el objetivo descrito. Para ello completa los siguientes puntos (puede suponer que el recorrido del árbol de derivación para propagar atributos será en profundidad por la izquierda con retroseguimiento):
1.- Describe explícita y brevemente el significado de cada atributo que utilices así como el proceso de actualización de cada uno de ellos en las reglas de la gramática mencionando explícitamente si se realiza herencia o síntesis en la propagación de cada atributo.
2.- Define formalmente la gramática utilizando la notación explicada en el temario de esta asignatura.
3.- Muestre como ejemplo el árbol de derivación de la siguiente cadena añadiendo en él los valores de los atributos y alguna indicación gráfica (por ejemplo flechas) que muestren la propagación de sus valores

(()):()()()

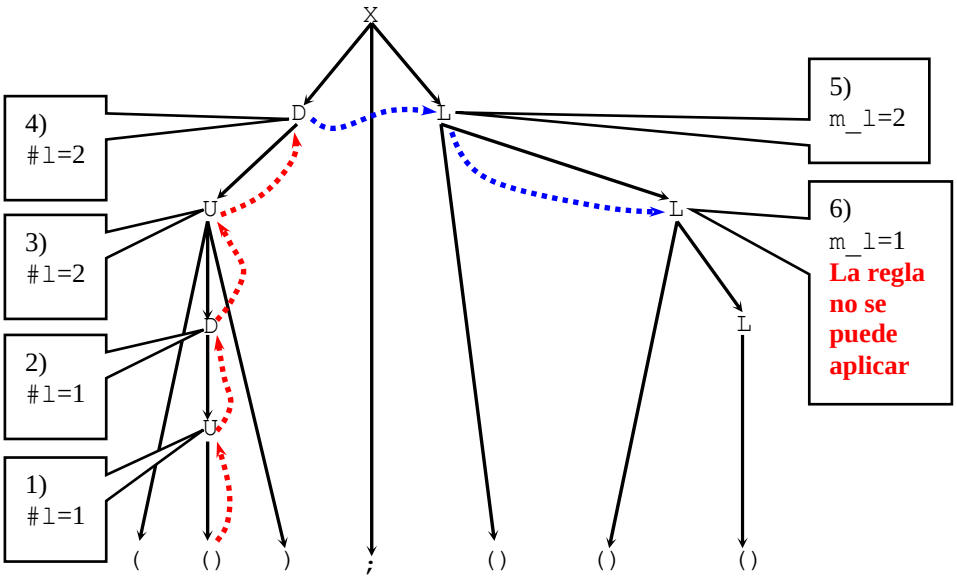
SOLUCIONES

1.-2.-
Es suficiente con tener dos atributos, uno (num_listas) asociado al símbolo no terminal D y que se calcula por síntesis (el número de listas de la parte izquierda de la cadena que "cuelga" del símbolo no terminal D y que tiene que controlarse que sea menor o igual que 64) y otro (max_listas) que se calcula por herencia que es el máximo número de listas y que se propaga desde el hermano izquierdo (D) en la regla del axioma y luego de padre a hijo por la regla recursiva de L. En esta segunda propagación padre-hijo se va quitando uno cada vez que se aplica la regla recursiva de tal manera que, como se termina con la regla L→() hay que tener en cuenta que no se exceda el número máximo controlando que en la última aplicación de la regla recursiva no se disminuya el valor de este atributo por debajo de 1.
En el árbol se ha reducido el número de flechas escribiendo sólo una cuando se deriva la lista vacía.
También se ha utilizado #l para representar num_listas y ml para representar max_listas.

```
X → D:L
2:{
      L.max_listas = D.num_listas   /* Herencia (hermano) */
}

D1 → UD2
2:{
      Si U.num_listas + D2.num_listas > 64
            ERROR;
      Otro caso
            /* Síntesis */
            D1.num_listas = U.num_listas + D2.num_listas;
}

D → U
1:{
      D.num_listas = U.num_listas;  /* Síntesis */
}

U → ()
2:{ /* Suponiendo que cada paréntesis es un símbolo */
      L.num_listas = 1;             /* Síntesis */
}

U → (D)
3:{ /* Suponiendo que cada paréntesis es un símbolo */
      Si D.num_listas >= 64
            ERROR;
      Otro caso
            U.num_listas = D.num_listas + 1 /* Síntesis */
}

L1 → ()L2
0:{
      Si L1.max_listas - 1 < 1
            /* Como la regla de sálida es con (), siempre hay que
            contar con una lista más para acabar correctamente */
            ERROR;
      Otro caso
            L2.max_listas = L1.max_listas - 1;  /* Herencia (padre)*/
}

L → ()
{ /* Como se ha controlado previamente no hay que hacer nada
      necesariamente*/
```

```
}
```

3.-


X
D
L
U
L
D
L
U
(( )) ; ( ) ( ) ( )
4)
#l=2
3)
#l=2
2)
#l=1
1)
#l=1
5)
m_l=2
6)
m_l=1
La regla no se puede aplicar

5.- Suponga la gramática independiente del contexto que puede deducirse de las siguientes reglas de producción (el axioma es el símbolo VIAJE):

```
VIAJE      →   localizador ORIGEN DESTINO EQUIPAJE
ORIGEN     →   MAD | BCN
DESTINO    →    MAD | BCN
EQUIPAJE   →    MALETAS peso
MALETAS    →    etiqueta MALETAS
MALETAS    →    etiqueta
MALETAS    →    λ
```

Se quiere modificar esta gramática para construir una de atributos que compruebe que los datos del viaje sean correctos, esto es que:

1. La ciudad de origen sea distinta a la ciudad destino.
2. El peso sea inferior o igual a 20 Kg.
3. El número de maletas sea inferior o igual a 3.
4. Cada maleta tenga una etiqueta distinta.
5. La información del localizador sea correcta. Por ej, si el localizador es MAD_BCN_3_19 esto significa que el origen tiene que ser MAD, el destino BCN, el número de maletas 3 y el peso total 19 Kg.

Así, por ejemplo, las siguientes cadenas serían correctas:

```
MAD_BCN_3_19 MAD BCN et1 et2 et3 19
  BCN_MAD_2_10 BCN MAD et1 et2 10
```

Y las siguientes cadenas no lo serían:

```
  MAD_MAD_3_19 MAD MAD et1 et2 19
MAD_BCN_3_19 BCN MAD et1 et2 et3 19
```

Se pide:

Añadir a la gramática anterior un sistema de atributos para conseguir el objetivo descrito. Para ello complete los siguientes puntos (sólo puede utilizar síntesis):

- Suponga que se inicializan los atributos necesarios para el localizador al inicio del proceso con los valores adecuados. Esto es, para el localizador MAD_BCN_3_19, es válido suponer que inicialmente se dispone de las informaciones 'MAD' como origen, 'BCN' como destino, 3 como el número de maletas y 19 como el peso. Si decide utilizar atributos para guardar esa información puede suponer que en los nodos del árbol correspondientes al localizador, los atributos se inicializan automáticamente con los valores correctos.
- En el caso de que la expresión sea incorrecta, la gramática debe informar de todos los errores que se encuentre e indicarlos por pantalla.

SOLUCIONES

Una posible gramática de atributos solución al problema planteado sería:

```
VIAJE       →   localizador ORIGEN DESTINO EQUIPAJE {
      si ((localizador.peso>20) || (localizador.num_m>3) ||
         (localizador.origen == localizador.destino)) {
           print "Error: localizador erróneo" ;
      }
     VIAJE.peso = localizador.peso;
      VIAJE.num_m = localizador.num_m;
      VIAJE.origen = localizador.origen;
      VIAJE.destino = localizador.destino;
      si ((ORIGEN.valor  != VIAJE.origen) ||
         (DESTINO.valor != VIAJE.destino) ||
          (EQUIPAJE.peso != VIAJE.peso) ||
          (EQUIPAJE.num_m != VIAJE.num_m)) {
          print "Error: información del viaje incorrecta" ;
      }
}

ORIGEN      →   MAD { ORIGEN.valor = 'MAD' ; }
ORIGEN      →   BCN { ORIGEN.valor = 'BCN' ; }
DESTINO     →    MAD { DESTINO.valor = 'MAD' ; }
DESTINO     →    BCN { DESTINO.valor = 'BCN' ; }

EQUIPAJE    →    MALETAS peso {
                 EQUIPAJE.peso = peso.valor;
                 EQUIPAJE.num_m = MALETAS.num_m;
}

MALETAS1    →    etiqueta MALETAS2 {
                 MALETAS1.num_m = MALETAS2.num_m+1;
                 si (encuentra(etiqueta,MALETAS2.etiquetas)){
                       print "Error: etiqueta repetida";
                 }
                 else MALETAS1.etiqueta =
                       inserta(etiqueta,MALETAS2.etiquetas);
}
MALETAS     →    etiqueta {
                 MALETAS.num_m = 1;
                  MALETAS1.etiqueta = inserta
                             (etiqueta , '');
}
MALETAS     →    λ {
                 MALETAS.num_m = 0;
                 MALETAS.etiquetas = '';
}
```

6.- Suponga la gramática independiente del contexto que puede deducirse de las siguientes reglas de producción (el axioma es el símbolo SUMA)

```
SUMA   →     PRIMERO + CANTIDAD
PRIMERO    →    DINERO
CANTIDAD     →     DINERO + CANTIDAD | DINERO
DINERO       →     SIMBOLO1 cifra | cifra SIMBOLO2
SIMBOLO1     →     $ | £
SIMBOLO2     →     €
```

Suponga que el terminal `cifra`, representa de manera abreviada números naturales con dígitos del conjunto {0,...,9}
Que querría usarse para generar sumas en las que todas las cantidades fueran de la misma moneda.
Las siguientes cadenas serían correctas

```
34€ + 153€
$34 +  $55 + $120 + $25
```

Y la siguiente cadena no

```
$34 + 55€
```

Se quiere modificar esta gramática para construir una de atributos que calcule el valor total de la suma de las cantidades intermedias y compruebe que no se suman valores cuyos tipos no coinciden.

Se pide:

Añadir a la gramática anterior un sistema de atributos para conseguir el objetivo descrito. Para ello complete los siguientes puntos (no puede utilizar información global):

- Considere que el primer sumando (desde la izquierda), que tiene asociado el no terminal `PRIMERO`, es el que determina el tipo que deberían tener los demás.
- En el caso de que la expresión sea incorrecta, la gramática debe mostrar la primera cantidad (desde la izquierda) que no es del tipo adecuado mostrando su valor, el tipo que debería tener (el del primer sumando) y el que realmente tiene.
- Describa explícita y brevemente el significado de cada atributo que utilice así como el proceso de actualización de cada uno de ellos en las reglas de la gramática.
- Defina formalmente la gramática.
- Puede suponer que el valor de cada `cifra` se inicializa automáticamente en el árbol, si lo necesita.

SOLUCIONES

Puesto que el objetivo es calcular el valor total, uno de los atributos es `valor`, y además como hay que comprobar los tipos de las cifras intermedias, otro atributo que se necesita es el del `tipo`. El primer operando da el tipo global, se puede utilizar el mismo atributo `tipo` para ello. Respecto a la comprobación, la estructura de las reglas hace que las cantidades concretas (asociadas con el no terminal `DINERO`) se vayan generando por la izquierda. Así que la regla recursiva de `CANTIDAD` (`CANTIDAD::=DINERO+CANTIDAD`) podría controlar si su tipo (que es heredado desde la parte izquierda y por tanto representa el tipo que debería ser el global) coincide con el tipo de `DINERO` (sintetizado desde la cadena de terminales). De esta manera, ya que el orden de recorrido del árbol terminará primero con las cantidades concretas desde la izquierda, hay que hacer la comprobación en esta regla. También hay que hacerla en la regla de salida de `CANTIDAD` (`CANTIDAD::=DINERO`) ya que el último sumando podría ser también incorrecto. Por lo tanto los atributos necesarios para cada símbolo de la gramática proporcionada son:

1. Para `SIMBOLO1` y `SIMBOLO2`: `tipo`.
2. Para `cifra`: `valor` (inicializado automáticamente en el árbol).
3. Para `DINERO` y `CANTIDAD`: `tipo`, y `valor`. En el caso de `DINERO`, el tipo será sintetizado y es el del sumando concreto. En el caso de `CANTIDAD`, el tipo será el heredado del primero sumando y por tanto representa el tipo global que debería tener toda la suma.
4. Para `PRIMERO` y `SUMA` necesitan `tipo` y `valor`. El tipo de `PRIMERO` es el de su sumando que coincide con el que debería tener la expresión completa porque es él el que lo determina y el

de SUMA es sintetizado y (sólo en el caso de que la expresión sea correcta) acumulará en valor el de la suma total.

Una posible gramática de atributos es:

SUMA → PRIMERO + CANTIDAD{
CANTIDAD.tipo = PRIMERO.tipo;
SUMA.tipo = PRIMERO.tipo;
SUMA.valor = PRIMERO.valor + CANTIDAD.valor;
}
PRIMERO → DINERO{
PRIMERO.tipo = DINERO.tipo;
PRIMERO.valor = DINERO.valor;
}

$CANTIDAD_1$ → DINERO + $CANTIDAD_2$ {
$CANTIDAD_2$.tipo = $CANTIDAD_1$.tipo;
SI (DINERO.tipo <> CANTIDAD.tipo) ERROR;
$CANTIDAD_1$.valor = DINERO.valor + $CANTIDAD_2$.valor;
}
CANTIDAD → DINERO {
SI (DINERO.tipo <> CANTIDAD.tipo) ERROR;
CANTIDAD.valor = DINERO.valor;
}
DINERO → SIMBOLO1 cifra {
DINERO.tipo = SIMBOLO1.tipo;
DINERO.valor = cifra.valor;}
}
DINERO → cifra SIMBOLO2 {
DINERO.tipo = SIMBOLO2.tipo;
DINERO.valor = cifra.valor;}
}
SIMBOLO1 → $ { SIMBOLO1.tipo = '$'; }
SIMBOLO1 → £ { SIMBOLO1.tipo = '£'; }
SIMBOLO2 → € { SIMBOLO2.tipo = '€'; }

Y utilizando la notación extendida:

SUMA → PRIMERO + CANTIDAD
1:{ CANTIDAD.tipo = PRIMERO.tipo;
SUMA.tipo = PRIMERO.tipo; }
3:{ SUMA.valor = PRIMERO.valor + CANTIDAD.valor; }
PRIMERO → DINERO
1:{
PRIMERO.tipo = DINERO.tipo;
PRIMERO.valor = DINERO.valor;
}

$CANTIDAD_1$ → DINERO + $CANTIDAD_2$
0:{ $CANTIDAD_2$.tipo = $CANTIDAD_1$.tipo; }
1:{ **SI (DINERO.tipo <> CANTIDAD.tipo) ERROR;** }
3:{ $CANTIDAD_1$.valor = DINERO.valor + $CANTIDAD_2$.valor; }
CANTIDAD → DINERO
1:{
SI (DINERO.tipo <> CANTIDAD.tipo) ERROR;
CANTIDAD.valor = DINERO.valor;
}
DINERO → SIMBOLO1 cifra
2:{
DINERO.tipo = SIMBOLO1.tipo;
DINERO.valor = cifra.valor;}
}
DINERO → cifra SIMBOLO2
2:{
DINERO.tipo = SIMBOLO2.tipo;
DINERO.valor = cifra.valor;}
}
SIMBOLO1 → $ 1:{ SIMBOLO1.tipo = '$'; }
SIMBOLO1 → £ 1:{ SIMBOLO1.tipo = '£'; }
SIMBOLO2 → € 1:{ SIMBOLO2.tipo = '€'; }

7**.-** Suponga la gramática independiente del contexto que puede deducirse de las siguientes reglas de producción (el axioma es el símbolo SENT)

```
SENT →      INST SENT
SENT →      INST
INST   →    '+' NUM
INST   →    '>' cifra
NUM    →    cifra ':' cifra
```

Suponga el símbolo terminal `cifra` representa cualquier natural de dígitos decimales.

Que genera una secuencia de instrucciones del siguiente tipo:

- Tipo '+' como por ejemplo

```
+ 153:9
```

- Tipo '>' como por ejemplo

```
> 153
```

Se quiere añadir algunas restricciones semánticas a esta gramática.
Estas restricciones tienen como objetivo manipular una base de datos de códigos numéricos (números naturales) mediante las siguientes operaciones:

1. '+' Representa la inserción en la base de datos del número
2. '>' Representa la búsqueda en la base de datos del número

Para asegurar la corrección en los números introducidos, en la operación de inserción se añade una información de control que sigue al código. Ambas partes están separadas por el símbolo ':'. Esta información de control es un número natural cuyo valor tiene que coincidir con la suma de los dígitos del código.
Los siguientes números serían correctos:

```
153:9 2:2 121:4
```

Pero estos otros no:

```
153:153 2:12 121:1
```

Cuando se inserta un número debe comprobarse que no está ya insertado en la base de datos. En otro caso debe producirse una indicación de error y terminar el proceso.
Cuando se busca un código, debe haber sido previamente introducido en la base de datos. En caso contrario debe producirse una indicación de error y terminar el proceso.

Las siguientes secuencias de operaciones serían correctas

```
+34:7 >34 +15:6 >34
+34:7 +15:6 >34 >15 >34
```

Las siguientes secuencias de operaciones no serían correctas ya que alguno de sus códigos son incorrectos

```
+34:12 >34 +15:6 >34
+34:7 +15:56 >34 >15 >34
```

Las siguientes secuencias de operaciones tampoco lo serían

```
>34 +34:7  +15:6 >34
+34:7 +34:7 >34 >15 >34
```

Ya que en la primera se busca el 34 antes de insertarlo y en la segunda la segunda inserción del 34 se encuentra el valor en la base de datos.

También se quiere que el axioma disponga de una información que acumule y muestre el número de operaciones realizadas en total.

Se pide:

Añadir a la gramática anterior un sistema de atributos para conseguir el objetivo descrito. Para ello complete los siguientes puntos (la única restricción en su diseño es que la única información global que puede utilizar es la base de datos descrita en el enunciado):

- Considere que la base de datos se manipula mediante las operaciones (que realizan la función descrita en su nombre terminando la ejecución del sistema completo tras informar del error producido en caso de que no pueda realizarse la misma):
  - `insertar(bd, cifra)`
  - `buscar(bd, cifra)`
- Defina formalmente la gramática.
- Describa explícita y brevemente el significado de cada atributo que utilice así como el proceso de actualización de cada uno de ellos en las reglas de la gramática.
- Puede suponer, si lo necesita, que antes de comenzar el proceso, en las hojas del árbol se inicializa automáticamente el valor de los atributos del símbolo terminal `cifra` tanto con su valor como con el de la suma de sus dígitos.

SOLUCIONES

Puesto que el objetivo es controlar que los códigos insertados sean correctos es suficiente con disponer de atributos para el valor y la suma de dígitos de cada cifra. La gestión de la base de datos no requiere de ningún atributo adicional. Contar las operaciones realizadas puede realizarse con un atributo sintetizado que llamaremos total.

- Para cifra: valor y suma (inicializados en el árbol automáticamente en el momento inicial)
- NUM: guardará en valor el valor de su código ya que esto es lo que hay que insertar.
- SENT total.
- El resto de los símbolos no necesitan ningún atributo adicional
- Se utiliza como información global la base de datos bd

  Para distinguir el tipo de propagación se utiliza el siguiente convenio de colores:

  Síntesis

  Información global

  Comprobaciones

```
SENT → INST SENT1
{
    SENT.total = SENT1.total+1;
}
SENT → INST
{
    SENT.total = 1;
}
INST → '+' NUM
{
    insertar(bd,NUM.valor);
}

INST → '>' cifra
{
    buscar(bd,cifra.valor);
}

NUM → cifra1 ':' cifra2
{
    SI cifra1.suma != cifra2.valor ENTONCES SALIR_ERROR;
    OTRO CASO NUM.valor = cifra1.valor;
}
```

8.- La siguiente gramática independiente del contexto representa una versión simplificada de las oraciones del idioma castellano

```
<oracion> ::= <sujeto> <predicado>
            |   <predicado> <sujeto>
<sujeto> ::= <grupo_nombre>
<grupo_nombre> ::= <articulo> <nombre>
                 |   <nombre>
<predicado> ::= <verbo> <complemento>
<complemento> ::= <preposicion> <grupo_nombre>
                |   <grupo_nombre>
<articulo> ::= el | la | los | las
<nombre> ::= yo | tú | chico | chica | chicos | chicas
         | manzanas | carne | casa
<verbo> ::= como | comes | come | comen
<preposicion> ::= en | de
```

Observe que se han realizado muchísimas simplificaciones, una de las cuales es considerar los pronombres personales como nombres.

Se quiere diseñar una gramática de atributos que añada a las oraciones que pueden derivarse de la anterior gramática las siguientes restricciones:

- Dentro del grupo de un nombre el artículo (si hay) y el nombre deben concordar en género (femenino o masculino) y en número (singular o plural).
- Dentro de la oración el sujeto y el verbo del predicado deben concordar en número y persona (primera, segunda o tercera).

Estas comprobaciones no aseguran que las oraciones sean totalmente correctas en castellano pero a continuación se muestran algunas que sí lo son

```
el chico come en la casa
    yo como manzanas
```

Y otras que no lo son, la primera de ellas porque en el grupo del nombre “el chicos” no hay concordancia en número y en la segunda porque entre el sujeto (yo) y el predicado (comen) no hay concordancia ni en número ni en persona.

```
el chicos come en la casa
    yo comen manzanas
```

Se pide:

1) Añadir a la gramática anterior un sistema de atributos para conseguir el objetivo descrito. Para ello complete los siguientes puntos (para ello sólo puede utilizar síntesis de atributos):

- Describa explícita y brevemente el significado de cada atributo que utilice así como el proceso de actualización de cada uno de ellos en las reglas de la gramática.
- Defina formalmente la gramática
- Puede suponer que inicialmente, en las hojas del árbol correspondientes, se inicializa automáticamente para cada artículo, nombre, verbo y preposición su información semántica en los correspondientes atributos de los no terminales `<articulo>`, `<verbo>`, `<nombre>` y `<preposicion>`. Esto significa que, por ejemplo, si hay que analizar `comen` puede suponer que se representa mediante el símbolo terminal `<verbo>` que es de la tercera persona del plural.

2) Muestre su funcionamiento mediante los árboles de derivación de las siguientes cadenas (para ello añada en cada nodo del árbol un contador secuencial que indique el orden de recorrido del árbol para la evaluación de los atributos así como el valor de cada uno de ellos en cada instante en el que se visite el nodo): Indique explícitamente si las cadenas son correctas o no.

```
el chicos come carne en la casa
    la carne come chicos
```

Soluciones:

Los no terminales tendrán, como mucho, tres informaciones semánticas (todas ellas sintetizadas):

- género
- número
- persona

Cuyo valor permite distinguir entre femenino y masculino, singular y plural y entre primera, segunda y tercera persona.
<nombre>, necesita las tres
<articulo>, no necesita persona
<verbo>, no necesita género
<grupo_nombre>, no necesita género
<sujeto>, no necesita género
<predicado>, no necesita género.
Los no terminales no mencionados no necesitan atributos.

Formalmente (sólo se describen las reglas de producción).
Reglas de producción.

```
<oracion> ::= <sujeto> <predicado>
            |   <predicado> <sujeto>
            {
                  Si (<sujeto>.número != <predicado>.número ) O
                     (<sujeto>.persona != <predicado>.persona )
                  Error de concordancia entre sujeto y predicado
            }
<sujeto> ::= <grupo_nombre>
            {
                  <sujeto>.número = <grupo_nombre>.número;
                  <sujeto>.persona = <grupo_nombre>.persona;
            }
<grupo_nombre> ::= <articulo> <nombre>
            {
                  Si (<articulo>.número != <nombre>.número ) O
                     (<articulo>.género != <nombre>.género )
                        Error de concordancia entre artículo y nombre
                  OTRO CASO
                  {
                        <grupo_nombre>.número = <articulo>.número;
                        <grupo_nombre>.persona = <nombre>.persona;
                  }
            }
                  |   <nombre>
            {
                  <grupo_nombre>.número = <nombre>.número;
                  <grupo_nombre>.persona = <nombre>.persona;
            }
<predicado> ::= <verbo> <complemento>
            {
                  <predicado>.número = <verbo>.número;
                  <predicado>.persona = <verbo>.persona;
            }
<complemento> ::= <preposicion> <grupo_nombre> {}
                |   <grupo_nombre> {}
```

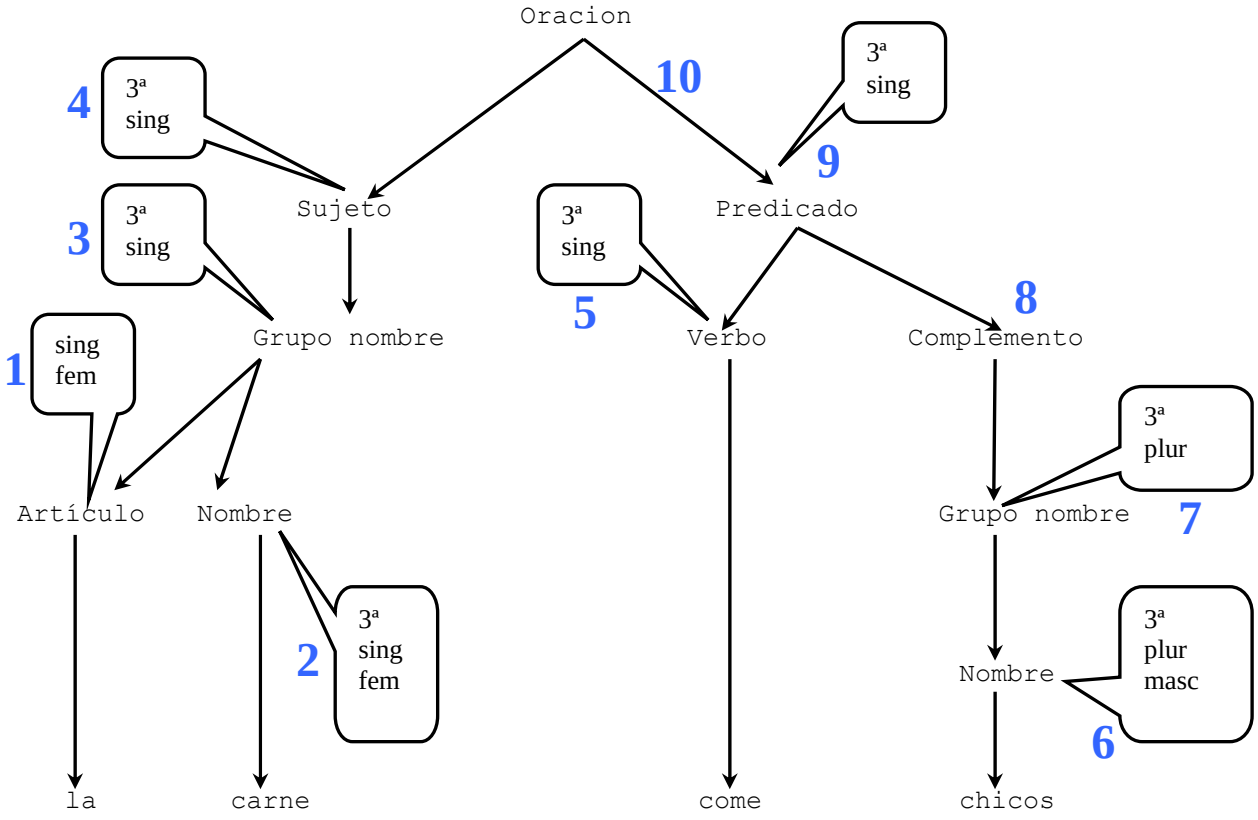
Sobre las dos cadenas.

```
el chicos come carne en la casa
la carne come chicos
```

La primera es incorrecta por dos razones:

1. El complemento, según la última regla, sólo puede ser uno por lo que la oración no puede ser generada por la gramática.
2. Hay un error en la concordancia entre el número del artículo y el del nombre del grupo de nombre del sujeto.

La segunda es correcta.



9.- Suponga la gramática independiente del contexto que puede deducirse de las siguientes reglas de producción (el axioma es el símbolo `S`)

```
S::=P:C:F
P::=0P|...|9P|0|...|9
C::=0C|...|9C|0|...|9
F::=0F|...|9F|0|...|9
```

A continuación tiene como ejemplo algunas de las cadenas que puede generar

`234:1:3234`
`893:39029:0231`

Se quiere modificar esta gramática para construir una de atributos que compruebe que en las cadenas generadas se cumplen las siguientes condiciones:

- La suma de los dígitos de la cadena inicial es igual a la suma de los dígitos de la cadena final.
- El número de dígitos de la cadena central coincide con la suma mencionada anteriormente.

A continuación se muestran algunas de las cadenas que puede generar

`211:9201:4`
`32:12019:1112`

Se pide:

1) Añadir a la gramática anterior un sistema de atributos para conseguir el objetivo descrito. Para ello complete los siguientes puntos (sólo puede utilizar síntesis e información global):

- Describa explícita y brevemente el significado de cada atributo o información global que utilice así como el proceso de actualización de cada uno de ellos en las reglas de la gramática.
- Defina formalmente la gramática
- Puede suponer que, inicialmente, los nodos del árbol que contienen los terminales correspondientes a los dígitos son inicializados con la información semántica que necesiten. Por ejemplo, para acceder al valor de cada dígito puede utilizar la función `valor(digito)` (por ejemplo `valor('9')` es el entero `9`).

2) Muestre su funcionamiento mediante los árboles de derivación de las siguientes cadenas (para ello añada en cada nodo del árbol un contador secuencial que indique el orden de recorrido del árbol para la evaluación de los atributos así como el valor de cada uno de ellos -junto con la información global- en cada instante en el que se visite el nodo): Indique explícitamente si las cadenas son correctas o no.

`2:01:11`
`2:010:11`

Soluciones:

Los símbolos y sus atributos son:

`P` tendrá un atributo entero, `suma`, para calcular la suma de sus dígitos.
`C` tendrá un atributo entero, `tamaño`, para calcular su longitud
`F` tendrá un atributo entero, `suma`, para calcular la suma de sus dígitos.

Los tres se calculan por síntesis en las reglas que definen su bloque.
En la regla que define la estructura de las cadenas, se comprueba que la suma del primer bloque sea igual que el tamaño del segundo y la suma del tercero.

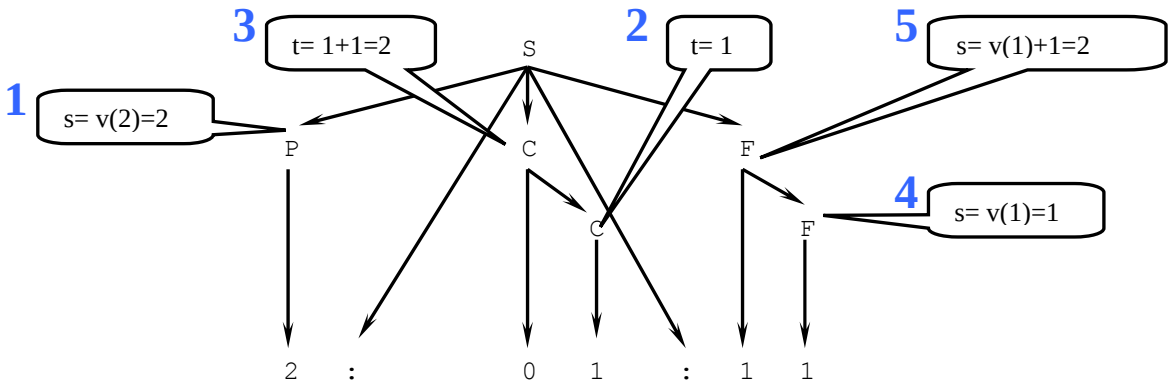
Formalmente.
Símbolos terminales.
No cambian respecto a la gramática independiente del contexto. Se supone que la inicialización inicial de los nodos del árbol que los contiene asigna el valor necesario, se utilizará la notación `valor(x)` para acceder al valor numérico del dígito representado por el símbolo `x`.
Símbolos no terminales.
`{P(int suma), C(int tamaño), P(int suma)}`
Reglas de producción.

Se utilizará una abreviatura para representar todas las reglas de comportamiento similar

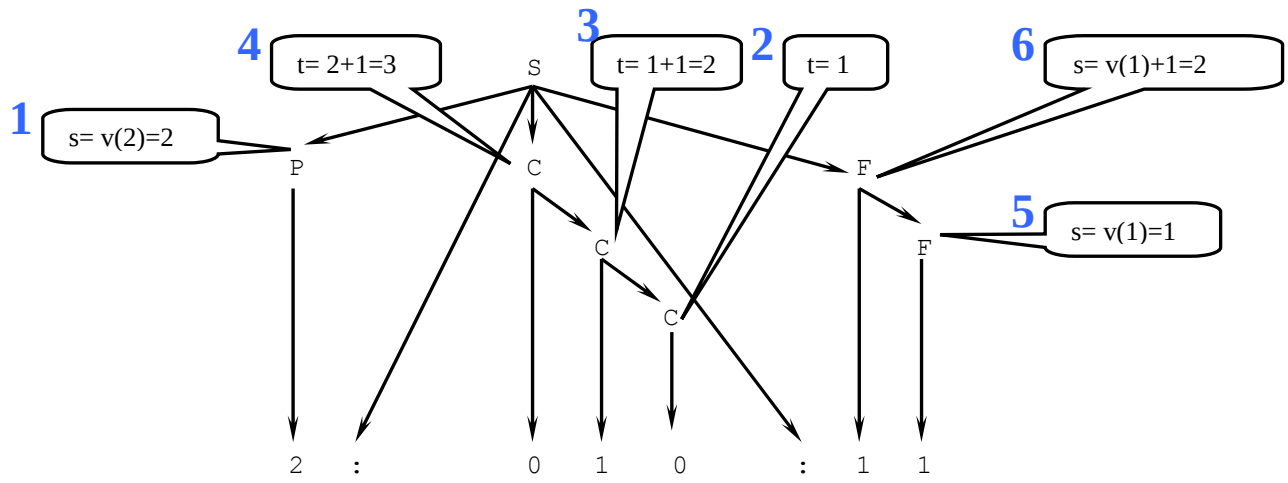
```
S→P:C:F{Si ( NO (P.suma == F.suma) O NO(P.suma == C.tamaño))
      ERROR SEMÁNTICO }
∀ x∈{0,1,2,...,9}
P1→xP2 { P1.suma = valor(x)+ P2.suma;}
P→x { P.suma = valor(x);}
F1→xF2 { F1.suma = valor(x)+ F2.suma;}
F→x { F.suma = valor(x);}
C1→xC2 { C1.tamaño = 1+ C2.tamaño;}
C→x { C.tamaño = 1;}
```

Información global.
No se utiliza.

Para la palabra `2:01:11`:(que es correcta)



Para la palabra `2:010:11`:(que es incorrecta, ya que en la raíz puede comprobarse que las sumas de los extremos son correctas (2) pero el tamaño central (3) no lo es)



7.I.- Suponga que la siguiente gramática independiente del contexto genera cadenas de caracteres que representan árboles según la siguiente notación.

Los árboles aparecen delimitados por paréntesis `( ··· )`

El árbol más sencillo posible es precisamente una lista vacía `()`.

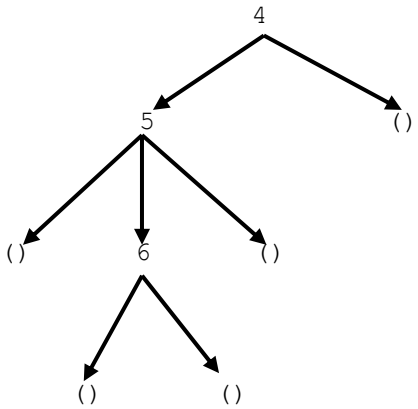
Si el árbol no está vacío,

- La raíz del árbol es un número natural y es el primer elemento de la lista.
- Los hijos del árbol son los siguientes elementos de la lista. Cada hijo es a su vez un árbol cualquiera.

A continuación se muestra algún ejemplo de árboles

```
()

(4   (5   ()   (6 () () )   ()   )   ()   )
```

Este último podría representarse gráficamente de la siguiente manera




```
axioma       →     arbol
arbol →      ( NUMERO lista_arboles )
             |     ()
lista_arboles      →     arbol
                   |     arbol lista_arboles
```

Observe que el no terminal `axioma` tiene una regla para garantizar que hay una regla explícita asociada con el principio de la derivación.

Considere que esta gramática se extiende asignando a todos sus símbolos un atributo de nombre `profundidad` y tipo `entero` que tiene como objetivo calcular la profundidad del árbol.

1. Diseñe una gramática de atributos a partir de esta gramática en la que sólo se tenga el atributo mencionado y gestiónelo para calcular por síntesis la profundidad del árbol.
2. Muestre como ejemplo el árbol de derivación correspondiente al segundo ejemplo del enunciado en el que añada en cada nodo el valor del atributo e indique explícitamente (por ejemplo mediante una secuencia de números en cada nodo) el orden en el que se recorren los nodos para calcular el valor de los atributos.

7.II.- Suponga que la gramática se modifica para sustituir la primera regla por la siguiente:

```
axioma      →     NUMERO arbol
```

Y que todos los símbolos no terminales de la gramática tienen un atributo de tipo `entero` y nombre `profundidad_maxima` y que se utilizará para producir un error semántico en árboles de profundidad mayor que la especificada.
La gestión de ese atributo puede resumirse de la siguiente manera:

1. El valor del número generado por la primera regla y que precede al árbol es precisamente el valor de `profundidad_maxima`
2. Ese valor debe ser propagado (por síntesis) hasta un no terminal a partir del cual pueda heredarse para poder hacer la comprobación semántica pedida (informar de un error cuando se supere la profundidad máxima) en el nivel más profundo del árbol posible.

Se pide:

1. Añadir a las reglas de la gramática resultante de la primera parte las acciones necesarias para la gestión del nuevo atributo.
2. Mostrar como ejemplo el árbol de derivación de las siguientes cadenas (mostrando, como en el apartado anterior, en cada nodo el valor de todos los atributos y un número que muestre la secuencia mediante la que se debe recorrer el árbol para obtener los valores):

```
6 (4   (5  ()  (6 () () )  ()  )   ()   )
    6 ( 1 ( 2 () ) () ( 3 ( 4 () ) ) )
```

3. Indique explícitamente si las cadenas anteriores representan árboles correctos o incorrectos.

SOLUCIONES (I y II)

A continuación se muestra una gramática de atributos que hace más gestión de la pedida para los problemas I y II. Se trata de que el alumno observe cómo se puede obtener la gramática particular del problema I y la particular del II.
La siguiente gramática de atributos genera cadenas con las siguientes características:

- En lugar de sólo un valor previo al árbol (como se especifica en el problema II) se generan 3 con el objetivo de especificar con uno de ellos la profundidad máxima (como se pide en el problema II) y además, la suma máxima de los números de las raíces de los árboles y del número máximo de listas (paréntesis de apertura) que aparezcan en todo el árbol. Obsérvese que el problema 4 sólo se refiere a la gestión de la profundidad máxima. El resto se deja como ejemplo de otras gestiones posibles mediante gramáticas de atributos.
- La profundidad del árbol se gestiona por síntesis. Esta gestión es la única pedida en el problema I. El problema II añade el control en el nivel más profundo del árbol de que no se supere la condición de profundidad máxima.
- Para realizar el control en el nivel más profundo del árbol (el más cercano a las raíces del mismo) tiene que tenerse en cuenta que la síntesis de la profundidad sólo ***modifica el valor de la misma en la regla en la que aparecen los paréntesis nuevos***, es decir en la regla `arbol → ( NUMERO lista_arboles )`. Por tanto será en ese punto en donde deberíamos controlar que no se supera la profundidad máxima.
- También se calcula el número de listas y se acumula la suma de todas las raíces. La gestión también es por síntesis y es análoga a la de la profundidad. Esta gestión no se pide ni en el problema I ni en el problema II.
- La gramática completa realmente sólo generaría cadenas cuyo árbol no tuviera una profundidad mayor ni un número de listas mayor ni una suma de raíces mayor que el valor correspondiente de los tres generados delante del árbol. El problema I sólo requiere que se calcule por síntesis la profundidad sin ningún control adicional (su regla inicial no genera ningún número delante del

árbol). El problema II sólo controla profundidad. El control del número máximo de listas y de la suma máxima de raíces se deja como ejemplo.

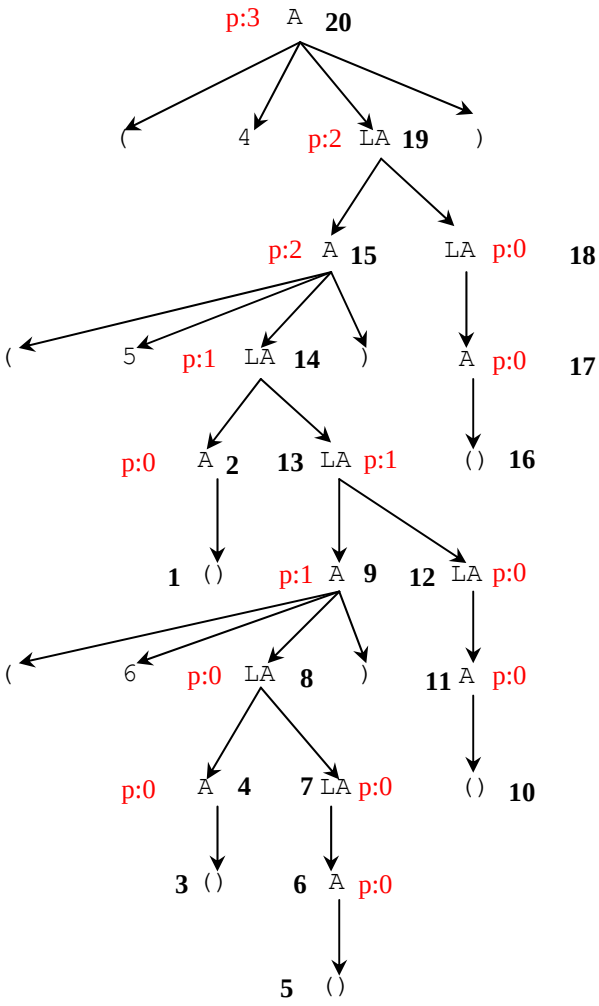
Dado que sólo se aporta una gramática para los dos problemas (I y II) y que además se añade más gestión, se resalta en negrita lo necesario para el problema II y se subraya, además, lo necesario para sólo el problema I.

```
Información global: no se utiliza.
Atributos:
axioma(entero profundidad, p_max, numero_arboles, num_max, suma,
suma_max)
arbol(entero profundidad, p_max, numero_arboles, num_max, suma,
suma_max)
TOK_NUM (entero valor). Suponemos que este valor se inicializa al
principio del proceso.
axioma:     TOK_NUM1 TOK_NUM2 TOK_NUM3
            arbol
      {
            arbol.p_max = TOK_NUM1.valor;
            arbol.num_max = TOK_NUM2.valor;
            arbol.suma_max = TOK_NUM3.valor;

      }
arbol:      '(' TOK_NUM lista_arboles ')'
      {
            arbol.profundidad = 1+lista_arboles.profundidad;
            arbol.numero_arboles = 1+ lista_arboles.numero_arboles;
            arbol.suma = TOK_NUM.suma +  lista_arboles.suma;
            lista_arboles.p_max = arbol.p_max;
            lista_arboles.num_max = arbol.num_max;
            lista_arboles.suma_max = arbol.suma_max;
            if (lista_arboles.p_max < lista_arboles.profundidad) ERROR;
            if (lista_arboles.num_max < lista_arboles.numero_arboles)
                                                      ERROR;
            if (lista_arboles.suma_max < lista_arboles.suma) ERROR;

      }
      |
      '(' ')'
      {
            arbol.profundidad = 0;
            arbol.numero_arboles = 1;
            arbol.suma = 0;
      }

lista_arboles:    arbol
      {
            lista_arboles.profundidad = arbol.profundidad;
            lista_arboles.numero_arboles = arbol.numero_arboles;
            lista_arboles.suma = arbol.suma;
            arbol.p_max = lista_arboles.p_max;
            arbol.num_max = lista_arboles.num_max;
            arbol.suma_max = lista_arboles.suma_max;

      }
            |
            arbol lista_arboles2
      {
            (*)if (arbol.profundidad > lista_arboles2.profundidad )
                  lista_arboles.profundidad = arbol.profundidad;
            else
                  lista_arboles.profundidad =
```

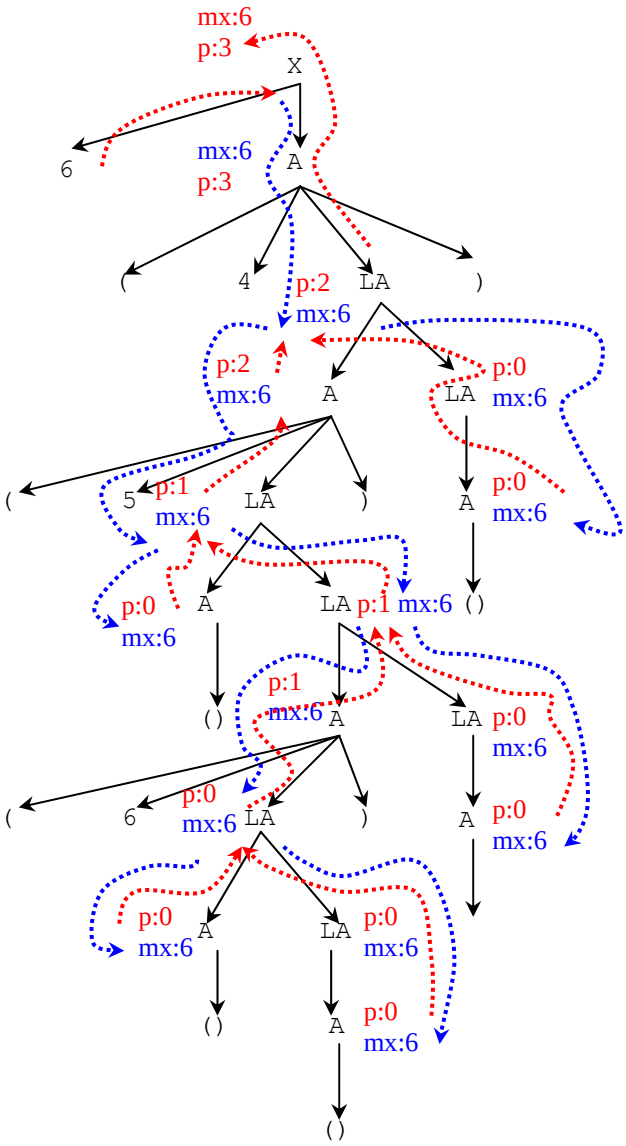
```
                                                  lista_arboles2.profundidad;
            lista_arboles.numero_arboles =
                  arbol.numero_arboles +
                  lista_arboles2.numero_arboles;
            lista_arboles.suma = arbol.suma + lista_arboles2.suma;
            arbol.p_max = lista_arboles.p_max;
            arbol.num_max = lista_arboles.num_max;
            arbol.suma_max = lista_arboles.suma_max;
            lista_arboles2.p_max = lista_arboles.p_max;
            lista_arboles2.num_max = lista_arboles.num_max;
            lista_arboles2.suma_max = lista_arboles.suma_max;
            (*)if(lista_arboles.p_max<lista_arboles.profundidad) ERROR;
            if (lista_arboles.num_max < lista_arboles.numero_arboles)
                                                            ERROR;
            if (lista_arboles.suma_max < lista_arboles.suma) ERROR;

      }
```

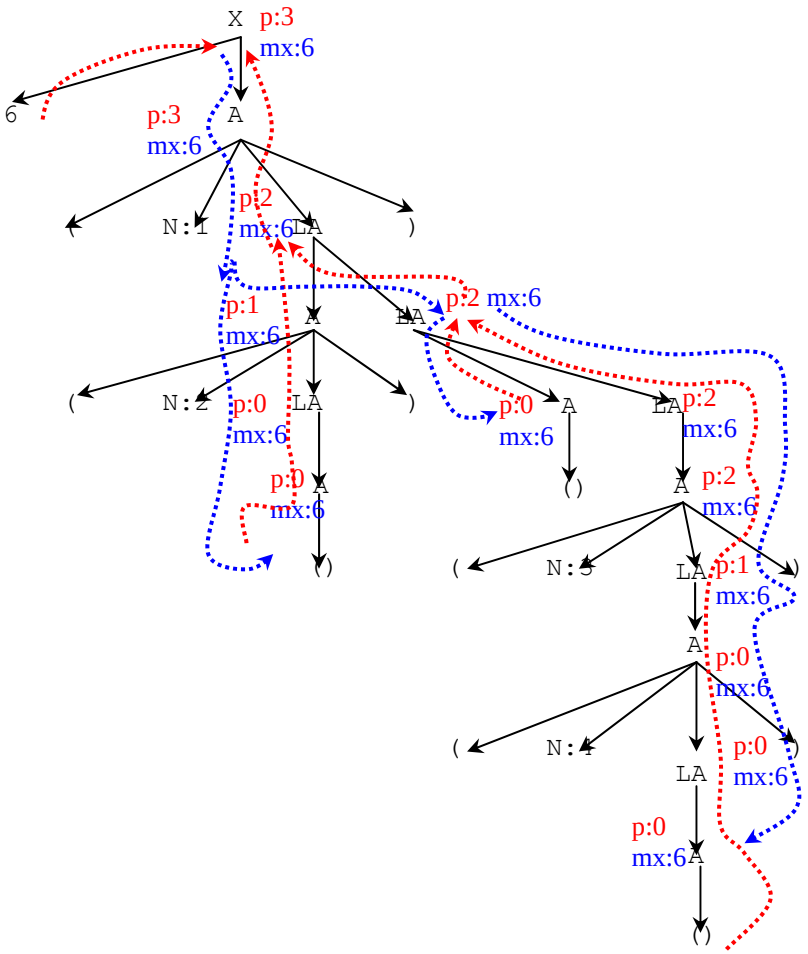
(*) Obsérvese que esta comprobación no es incorrecta pero es innecesaria ya que, si se controla que la profundidad sea correcta desde los niveles más bajos, de producirse esa circunstancia, se habría detectado en el punto más cercano a la raíz en el que se hubiera producido (en alguna regla de incremento de profundidad por presencia de un nuevo paréntesis)

El árbol pedido en el problema I es el siguiente.



A continuación se muestran los árboles pedidos en el problema II. Para simplificar la cuestión del orden del recorrido, se ha dibujado explícitamente las flechas de dependencia entre atributos..




X p:3
mx:6
6
p:3
mx:6
A
p:2
mx:6
N:1
LA
)
p:1
mx:6
A
LA
p:2 mx:6
N:2
p:0
mx:6
LA
)
p:0
mx:6
A
LA
p:2
mx:6
p:0
mx:6
A
()
A p:2
mx:6
(
N:3
LA
p:1
mx:6
)
A
p:0
mx:6
(
N:4
LA
p:0
mx:6
)
p:0
mx:6
A
()

10.- Se está diseñando un protocolo de comunicaciones entre ordenadores. Para incrementar la seguridad en la comunicación se ha decidido que la estructura de las cadenas que se transmiten sea la de una secuencia de "sub"cadenas concatenadas que tengan la misma longitud.
A continuación se muestra una gramática de atributos incompleta para esta opción. Las subcadenas están separadas mediante el símbolo `*`. No hay ningún separador ni al principio ni al final de la cadena completa.
La variable de tipo entero global `total` lleva cuenta del tamaño que deben tener las subcadenas. Este tamaño se calcula en las derivaciones del símbolo no terminal `I` (de bloque inicial). Los demás bloques están representados por el símbolo no terminal `B` (de bloque) que tiene un atributo de tipo entero, (de nombre `t` - de tamaño -) y que representa el tamaño del bloque.

```
{
        Σ_T={0,1,*},
        Σ_N={   X(),
                I(),
                B(entero t)
                },
        X,
        P={
                X→ I  {}
                X→ X*B  {}
                B_1→ 0B_2  {   B_1.t = B_2.t++; }
                B_1→ 1B_2  {   B_1.t = B_2.t++; }
                B→ 0  { B.t = 1; }
                B→ 1  { B.t = 1; }
                I→ 0I  { total++; }
                I→ 1I  { total++; }
                I→ 0  { total++; }
                I→ 1  { total++; }
        }
        Información_global = {entero total = 0}
}
```

Se pide:

1. Completar razonadamente la gramática para que, cuando se compruebe que una subcadena no tiene el mismo tamaño que la primera subcadena, se genere un error semántico. No puede añadir ningún atributo para este apartado, sólo puede modificar las acciones semánticas que se muestran.
2. Completar razonadamente la gramática con los atributos y acciones que considere oportuno para que una nueva variable global de tipo `entero` y nombre `num_sub` contenga, tras la derivación, el número de subcadenas de la cadena generada.
3. Reemplazar la información global total por atributos sintetizados y/o heredados y las acciones semánticas correspondientes para realizar la misma tarea. Describa brevemente las modificaciones realizadas y muestre el árbol de derivación anotado con los valores de los atributos para la cadena `101*111*000`. En las anotaciones de cada nodo del árbol debe añadir la siguiente información:
   - Los nombres y valores de los atributos del símbolo no terminal del nodo.
   - Los valores de las informaciones globales después de evaluar las reglas semánticas del nodo.
   - El orden en el que el nodo es visitado en el proceso de anotación y cálculo de los valores semánticos.

Soluciones:

1

```
{       Σ_T={0,1,*},
        Σ_N={   X(),
                I(),
                B(entero t)
                },
```

```
      X,
      P={   X→ I {}
            X→ X*B {SI ( B.t != total ) error_semantico();}
            B1→ 0B2 {   B1.t = B2.t++; }
            B1→ 1B2 {   B1.t = B2.t++; }
            B→ 0 {      B.t = 1;  }
            B→ 1 {      B.t = 1;}
            I→ 0I { total++; }
            I→ 1I { total++; }
            I→ 0 { total++; }
            I→ 1 { total++; }}
      Información_global = {entero total = 0}}
```

Se resalta la modificación. En este punto ya se sabe el valor del atributo sintetizado `t` de `B` y se puede comprobar si coincide con el `total` calculado en `I`.

2

```
{
      ΣT={0,1,*},
      ΣN={  X(),
            I(),
            B(entero t)
            },
      X,
      P={
            X→ I {}
            X→ X*B
            {
                  SI ( B.t != total ) error_semantico();
                  num_sub++;
            }
            B1→ 0B2 {   B1.t = B2.t++; }
            B1→ 1B2 {   B1.t = B2.t++; }
            B→ 0 {      B.t = 1;  }
            B→ 1 {      B.t = 1;}
            I→ 0I { total++; }
            I→ 1I { total++; }
            I→ 0 { total++; }
            I→ 1 { total++; }
      }
      Información_global = {entero total = 0, num_sub = 1}
}
```

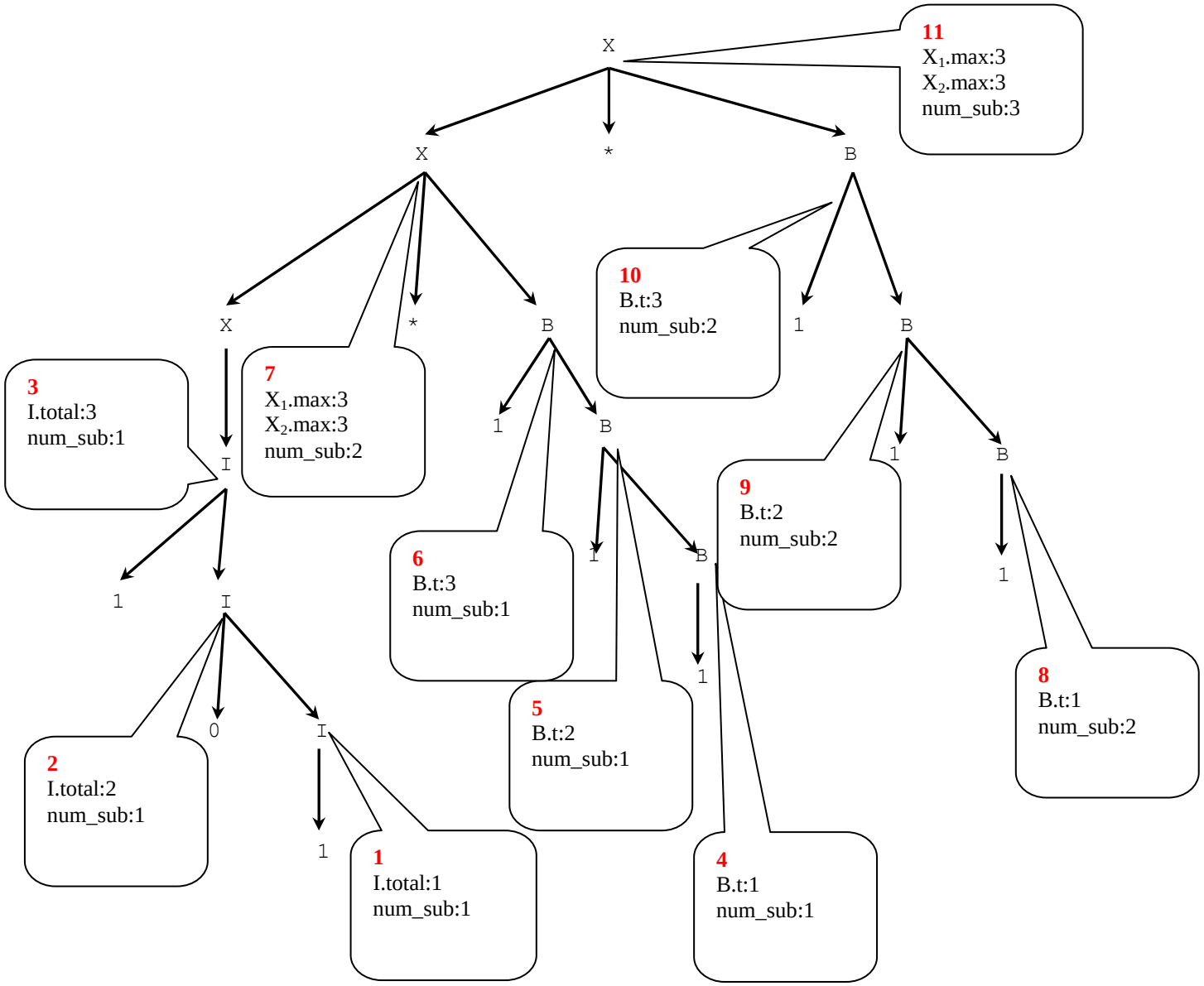
Se resalta también la nueva modificación. Es suficiente con inicializar la variable global a 1 e incrementarla cada vez que se genera un separador.

3

```
{
      ΣT={0,1,*},
      ΣN={  X(entero max),
            I(entero total),
            B(entero t)
            },
      X,
      P={
            X→ I {X.max = I.total;}
```

```
            X1→ X2*B
            {
                  X1.max = X2.max;
                  SI ( B.t != X2.max ) error_semantico();
                  num_sub++;
            }
            B1→ 0B2 {   B1.t = B2.t++; }
            B1→ 1B2 {   B1.t = B2.t++; }
            B→ 0 {      B.t = 1;  }
            B→ 1 {      B.t = 1;}
            I1→ 0I2 { I1.total= I2.total++; }
            I1→ 1I2 { I1.total= I2.total++; }
            I→ 0 { I.total=1; }
            I→ 1 { I.total=1; }
      }
      Información_global = {num_sub = 1}
}
```

El atributo `total` de `I` hace lo mismo que el atributo `t` de `B` pero en el bloque inicial, es decir, calcula el tamaño del bloque. El valor inicial se lo da la última regla, la no recursiva y que termina el bloque, se incrementa el valor en 1 en las reglas recursivas.
El no terminal `X` tiene un nuevo atributo (`max`) que propaga por síntesis hacia las `X` padre y que permite, en la regla en la que se deriva un nuevo bloque (`X::=X*B`), la comparación entre el tamaño del bloque inicial (que está en el atributo `max` de las `X`) con el tamaño del nuevo bloque, sintetizado en el subárbol con raíz ésta `B`.



A continuación se muestra otra posible solución, es más complicada porque utiliza atributos heredados y sintetizados y requiere un recorrido del árbol de derivación más complejo (se omite el árbol de derivación anotado pedido para la cadena)

```
{
      Σ_T={0,1,*},
      Σ_N={   X(entero max),
              I(entero max_i, max_f),
              B(entero t)
              },
      X,
      P={
              X→ I {      I.max_i = 1;
                          X.max = I.max_f;
                    }
              X1→ X2*B
              {     X1.max = X2.max;
                    num_sub++;
                    SI ( B.t != X2.max ) error_semantico();
              }
              B1→ 0B2 {   B1.t = B2.t++; }
              B1→ 1B2 {   B1.t = B2.t++; }
              B→ 0 {      B.t = 1;  }
              B→ 1 {      B.t = 1;}
              I1→ 0I2 {
                          I2.max_i = I1.max_i++;
                          I1.max_f = I2.max_f;
                    }
              I1→ 1I2 {
                          I2.max_i = I1.max_i++;
                          I1.max_f = I2.max_f;
                    }
              I→ 0 { I.max_f = I.max_i; }
              I→ 1 { I.max_f = I.max_i; }
      }
      Información_global = {entero num_sub = 1}
}
```