Teoría de Lenguajes

Práctica 10
(Gramáticas de atributos y TDS)

1. Determinar el conjunto de cadenas generadas por la siguiente gramática de atributos:

   $$G = \langle \{S, X, Y, Z\}, \{\mathtt{x}, \mathtt{y}, \mathtt{z}\}, P, S \rangle$$

   size es un atributo sintetizado de X y un atributo heredado de Y y Z, y P es:

   $$\begin{array}{lcll} S & \longrightarrow & XYZ & \{Y.size = X.size \ ; \ Z.size = X.size\} \\ X & \longrightarrow & \mathtt{x} & \{X.size = 1\} \\ & | & X_2\mathtt{x} & \{X.size = X_2.size + 1\} \\ Y & \longrightarrow & \mathtt{y} & \{\text{Condition: } Y.size = 1\} \\ & | & Y_2\mathtt{y} & \{Y_2.size = Y.size - 1\} \\ Z & \longrightarrow & \mathtt{z} & \{\text{Condition: } Z.size = 1\} \\ & | & Z_2\mathtt{z} & \{Z_2.size = Z.size - 1\} \end{array}$$

2. Sin cambiar el lenguaje generado, modificar la gramática del ejercicio 1 para que *size* se utilice sólo como atributo sintetizado.

3. Dar una gramática de atributos que genere el lenguaje $L = \{a^n(bc^n)^m \mid n \geq 2\}$. Construir un árbol decorado para la cadena *aabccbcc*.

4. Dar una gramática de atributos que genere el lenguaje $L = \{\alpha \mid \alpha \in (a|b|c)^*d^* \wedge |\alpha|_a = |\alpha|_b = |\alpha|_c = |\alpha|_d\}$. Construir un árbol decorado para las cadenas *aabcbcdd* y *bbccdd*.

5. Dada la siguiente gramática que genera expresiones aritméticas de suma y producto, definir un atributo *exp* de tipo string que sintetice la expresión generada pero sin paréntesis redundantes.

   $G = \langle \{E, T, F\}, \{\mathtt{+}, \mathtt{*}, \mathtt{id}, \mathtt{(}, \mathtt{)}\}, P, E\} \rangle$, con P:

   $$\begin{array}{lcl} E & \longrightarrow & E \ \mathtt{+} \ T \mid T \\ T & \longrightarrow & T \ \mathtt{*} \ F \mid F \\ F & \longrightarrow & \mathtt{id} \mid \mathtt{(} \ E \ \mathtt{)} \end{array}$$

6. Dada la gramática $G = \langle \{E\}, \{\mathtt{+}, \mathtt{*}, \mathtt{var}, \mathtt{const}, \mathtt{(}, \mathtt{)}\}, P, E \rangle$, con P:

   $E \longrightarrow E \ \mathtt{+} \ E \mid E \ \mathtt{*} \ E \mid \mathtt{var} \mid \mathtt{const} \mid \mathtt{(} \ E \ \mathtt{)}$

- Definir una gramática de atributos que sintetice la expresión original, pero reemplazando las subexpresiones en las que sólo aparezcan constantes por su resultado. Por ejemplo:

Si la expresión original es:	La expresión sintetizada debe ser:
$a + 3 * 4$	$a + 12$
$(3 + 2) * 4 + a + 2$	$20 + a + 2$
$(3 + a) * 2 + 2 * a$	$(3 + a) * 2 + 2 * a$

  El terminal `var` tiene un atributo *text* de tipo string. El terminal `const` tiene un atributo *val* de tipo entero. Se cuenta con la función toString(*entero*): string que convierte un entero en su representación decimal.

- Definir una gramática de atributos que sintetice la expresión original, pero aplicando las siguientes reglas de simplificación:

Expresión original	Expresión simplificada
$0 + E$	E
$E + 0$	E
$1 * E$	E
$E * 1$	E
$0 * E$	0
$E * 0$	0

  Los terminales `var` y `const` tienen un atributo *text* de tipo string.

Nota: no tener en cuenta el hecho de que la gramática es ambigua. Asumir que el árbol de derivación se armará según las precedencias usuales.

7. Sea *val* un atributo sintetizado que da el valor binario generado por S en la siguiente gramática. Por ejemplo, si la entrada es 101.101, $S.val = 5{,}625$.

   $G = \langle \{S, L, B\}, \{\texttt{0}, \texttt{1}, \texttt{.}\}, P, S \rangle$, con P:

$$\begin{array}{lcl} S & \longrightarrow & L\,\texttt{.}\,L \\ L & \longrightarrow & LB \mid B \\ B & \longrightarrow & \texttt{0} \mid \texttt{1} \end{array}$$

   a) Utilizar atributos sintetizados para calcular $S.val$.

   b) Calcular $S.val$ con una gramática de atributos en la cual el único atributo sintetizado de B sea *cont*, que da la contribución del bit generado por B al valor final. Por ejemplo, la contribución del primer bit en 101.101 es 4, la del último es 0,125.

8. Escribir una gramática de atributos que acepte cadenas sobre el alfabeto:

$$\{(,), +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$$

   cuyo formato sea como el del siguiente ejemplo:

$$(20, 5) + (F, 16) - (110, 2)$$

   Las cadenas se interpretan de la siguiente manera: cada par ordenado representa un número natural. El segundo elemento (escrito en base 10)

indica la base y el primer elemento es la representación del número en esa base. En el ejemplo, la cadena debe ser interpretada como $10+15-6 = 19$.

La gramática debe sintetizar el valor de la expresión en un atributo del símbolo inicial. Además, se deben rechazar las cadenas en las que algún par no represente un número válido. Por ejemplo, la cadena $(124, 3)$ debe ser rechazada porque "4" no puede aparecer en un número escrito en base 3.

9. La gramática $G = \langle \{E, T\}, \{\texttt{num}, ., \texttt{+}\}, P, E \rangle$, donde P es:

$$\begin{array}{lcl} E & \longrightarrow & E \texttt{ + } T \mid T \\ T & \longrightarrow & \texttt{num.num} \mid \texttt{num} \end{array}$$

genera expresiones formadas por la aplicación del operador + a constantes enteras y reales. Cuando se suman dos enteros, el resultado es entero, y en cualquier otro caso es real.

   a) Escribir una traducción dirigida por sintaxis que determine el tipo de las expresiones generadas por G.

   b) Extender la traducción de (a) para que además traduzca las expresiones a notación postfija. Los dos operandos del + deben ser del mismo tipo, y para esto se debe usar cuando sea necesario el operador unario `a_real` que convierte un valor entero a un valor real equivalente.

10. Sea $G = \langle \{D, E, T\}, \{\texttt{d}, \texttt{var}, \texttt{:}, \texttt{+}, \texttt{*}, \uparrow, \texttt{const}\}, P, D \rangle$, con P:

$$\begin{array}{lcl} D & \longrightarrow & \texttt{d var:} E \\ E & \longrightarrow & E \texttt{ + } T \mid T \\ T & \longrightarrow & \texttt{const * var} \uparrow \texttt{const} \end{array}$$

Se tienen los atributos $\texttt{const}.\mathit{val} : \mathit{int}$, con el valor numérico de la constante, y $\texttt{var}.\mathit{nombre} : \mathit{string}$, con el texto identificador de la variable.

Escribir una traducción dirigida por sintaxis que imprima una cadena con la derivada respecto de la variable que se encuentra luego del `d`, del polinomio que aparece a continuación de los dos puntos. Ejemplo:

Cadena de entrada: $dx : 2 * x \uparrow 3 + 3 * y \uparrow 2 + 5 * x \uparrow 5$
Se debe imprimir: $6 * x \uparrow 2 + 0 + 25 * x \uparrow 4$

Se dispone de la función $\mathit{toString}(\mathit{int}) : \mathit{String}$, que dado un entero devuelve una cadena con su representación decimal.

11. Consideraremos una versión simplificada de arreglos en PHP. Un arreglo en este lenguaje es una estructura que asocia valores a claves. Las claves pueden ser enteros o cadenas. Los valores pueden ser enteros, cadenas, u otros arreglos. Por ejemplo, el siguiente es un arreglo válido:

```
["abc"=>123, 45=>867, 90=>[7=>[], "d"=>"fg"]]
```

Además, indicar las claves es opcional. Si una clave no se especifica, toma el valor siguiente a la mayor clave entera encontrada hasta el momento (o 0 si no hay ninguna clave entera anterior). O sea que:

```
["a"=>1, 2, 3, 9=>4, 6=>["b"], 5]
```

es equivalente a:

```
["a"=>1, 0=>2, 1=>3, 9=>4, 6=>[0=>"b"], 10=>5]
```

La siguiente gramática genera arreglos y expresiones atómicas según la sintaxis descripta: $G =< \{E, L, V, A\}, \{\texttt{[}, \texttt{]}, \texttt{,}, \texttt{=>}, int, string\}, E, P >$, con P:

$$
\begin{array}{l}
E \longrightarrow A \mid \texttt{[}\ L\ \texttt{]} \mid \texttt{[]} \\
L \longrightarrow L\ \texttt{,}\ V \mid V \\
V \longrightarrow E \mid A\ \texttt{=>}\ E \\
A \longrightarrow int \mid string
\end{array}
$$

El terminal *int* representa un valor entero (`45`, `123`, etc.). El terminal *string* representa una cadena de caracteres: (`."bc"`, `"d"`, etc). Ambos tienen un atributo *valor* del tipo correspondiente.

Se pide convertir G en una *Traducción dirigida por sintaxis* que imprima una expresión equivalente a la de la entrada pero con todas las claves explícitas.

12. Dada la gramática G, en la que se definen algunas operaciones con listas, se desea realizar una traducción dirigida por sintaxis que imprima la evaluación de la expresión dada.

    $G = \langle \{S, L, L', L'', C, C', P, V\}, \{\texttt{function}, \texttt{(}, \texttt{)}, \texttt{[}, \texttt{]}, \texttt{\{}, \texttt{\}}, \texttt{,}, \texttt{var}, \texttt{num}\}, S, P \rangle$, con P:

    $$
    \begin{array}{lcl}
    S & \longrightarrow & \texttt{function (}\ L\ \texttt{)} \mid L \\
    L & \longrightarrow & C\ \texttt{[}\ L'\ \texttt{]} \\
    L' & \longrightarrow & L'' \mid \lambda \\
    L'' & \longrightarrow & V\ \texttt{,}\ L'' \mid V \\
    C & \longrightarrow & \texttt{\{}\ C'\texttt{\}} \mid \lambda \\
    C' & \longrightarrow & P\ \texttt{,}\ C' \mid P \\
    P & \longrightarrow & \texttt{(}\ \texttt{var}\ \texttt{,}\ \texttt{num}\ \texttt{)} \\
    V & \longrightarrow & \texttt{num} \mid \texttt{var} \mid \texttt{(}\ \texttt{num}\ \texttt{,}\ \texttt{num}\ \texttt{)}
    \end{array}
    $$

    El token `function` tiene un atributo *name*, que indica la función a evaluar:

    - compress: que, dada una lista, obtiene la versión comprimida de la misma.
      Por ejemplo, `compress([1,3,3,5,5,5,3])` $\rightsquigarrow$ `[1,(3,2),(5,3),3]`,
    - expand: que, dada una lista, obtiene la versión expandida de la misma.
      Por ejemplo, `expand([(2,8),3,10])` $\rightsquigarrow$ `[2,2,2,2,2,2,2,2,3,10]`,
    - max: que obtiene el máximo valor de una lista, comprimida o no. Si la lista está vacía, el máximo es `null`.
      Por ejemplo, `max([9,10,-1])` $\rightsquigarrow$ `10`, `max([9, 13, (14,10)])` $\rightsquigarrow$ `14`

    Las listas pueden contener variables, por lo que hay un contexto de variables, opcional, donde constan variables y su valor. Todas las variables deben tener un valor asignado para que se pueda evaluar una expresión.
    Por ejemplo:
    `{(x,3),(y,1)}[3,x,34,x,5]` $\rightsquigarrow$ `[3,3,34,3,5]`, `{(z,3)}[]` $\rightsquigarrow$ `[]`,
    `compress({(y,8),(x,8),(z,3)}[x,y,z])` $\rightsquigarrow$ `[(8,2),3]`,
    `{(x,3)}[x,x,y]` $\rightsquigarrow$ `ERROR ('y' sin definir)`,
    `max({(x,1)}[8,z])` $\rightsquigarrow$ `ERROR ('z' sin definir)`.