Teoría de Lenguajes:
Práctica 10 – Gramáticas de atributos y TDS

1er. cuatrimestre 2017

1. Determinar el conjunto de cadenas generadas por la siguiente gramática de atributos $G_1 = \langle V_N, V_T, P, S \rangle$

   Donde:

   $V_N = \{S, X, Y, Z\}$
   $V_T = \{\texttt{x}, \texttt{y}, \texttt{z}\}$

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

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

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 $L = \{a^n(bc^n)^* | n \geq 2\}$. Construir un 'arbol decorado para la cadena $aabccbcc$.

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

5. Dada la siguiente gramática G_2 que genera expresiones aritméticas de suma y producto, definir un atributo exp que sintetice la expresión generada pero sin paréntesis redundantes.

   $G_2 = \langle \{E, T, F\}, \{\texttt{+}, \texttt{*}, \texttt{id}, \texttt{(}, \texttt{)}\}, P_2, E \} \rangle$, con P_2:

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

6. Dada la gramática $G_3 = \langle \{E\}, \{\texttt{+}, \texttt{*}, \texttt{var}, \texttt{const}\}, P_3, E \rangle$, con P_3:

   $E \rightarrow E\texttt{+}E \mid E\texttt{*}E \mid \texttt{var} \mid \texttt{const} \mid \texttt{(}E\texttt{)}$

   - 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 'arbol 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 el input es 101.101, $S.val = 5{,}625$.
$G_4 = \langle\{S, L, B\}, \{\texttt{0}, \texttt{1}, \texttt{.}\}, P_4, S\rangle$, con P_4:

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

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

   b) Calcular $S.val$ con una gramática de atributos en la cual el 'unico 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.

8. La gramática $G_5 = \langle\{E, T\}, \{\texttt{num}, \texttt{.}, \texttt{+}\}, P_5, E\rangle$, donde P_5 es:

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

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_5.

   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 el operador unario `a_real` que convierte un valor entero a un valor real equivalente.

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

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

10. Sea $G_6 = \langle\{D, E, T\}, \{\texttt{d}, \texttt{var}, \texttt{:}, \texttt{+}, \texttt{*}, \uparrow, \texttt{const}\}, P_6, D\rangle$, con P_6:

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

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

    Se debe 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 $toString(int) : String$, que dado un entero devuelve una cadena con su representación decimal.

11. Hacer una gramática de atributos para G_7, en la que se definen expresiones en un lenguaje de programación con subtipado[1], donde se desea obtener el tipo de una expresión en el atributo '.tipoó detectar errores lo antes posible, cuando los hubiera.

$G_7 = \langle\{S, F, V, L, E\}, \{\texttt{+}, \texttt{and}, \texttt{or}, \texttt{flatten}, \texttt{(}, \texttt{)}\, \texttt{[}, \texttt{]}, \texttt{,}, \texttt{float}, \texttt{int}, \texttt{nat}, \texttt{bool}, \texttt{string}, \texttt{char}, P_3, S\}\rangle$, con P_7:

$$\begin{array}{lcl} S & \rightarrow & V \mid F \\ F & \rightarrow & \texttt{+(}\, V \,\texttt{)} \mid \texttt{and(}\, V \,\texttt{)} \mid \texttt{or(}\, V \,\texttt{)} \mid \texttt{flatten(}\, V \,\texttt{)} \\ V & \rightarrow & E \mid \texttt{[}\, L \,\texttt{]} \\ L & \rightarrow & V \,\texttt{,}\, L \mid V \\ E & \rightarrow & \texttt{float} \mid \texttt{int} \mid \texttt{nat} \mid \texttt{bool} \mid \texttt{string} \mid \texttt{char} \end{array}$$

[1] Las relaciones de subtipado (`subtipo <: tipo`) son reflexivas y transitivas, e implican que donde se espera una expresión del tipo determinado, se puede utilizar una del subtipo, sin generar errores:
`bool <: nat, nat <: int, int <: float, char <: string`
Lo mismo aplica para listas (e.g., `[bool] <: [nat]`, etc.)

Por ejemplo,
$w_1 =$ `flatten([[float, num], [bool], [int, bool, int]])` $\in L(G_7)$
y su tipo es `[float]`,
$w_2 =$ `and([nat,char])`, $w_3 =$ `+([string, int])`, $w_4 =$ `flatten([nat,[int]])`
$\notin L(G_7)$, y
$w_5 =$ `+(nat)`, $w_6 =$ `+([string, string])`, $w_7 =$ `+([nat, int, nat ])`
$\in L(G_7)$ y su tipo es `nat`, `string` e `int`, respectivamente.

12. Sea $G_8 = \langle \{E, C, S, T\}, \{\texttt{+}, \texttt{\&\&}, \texttt{==}, \texttt{!}, \texttt{true}, \texttt{false}, \texttt{num}, \texttt{string}\}, P_3, E \rangle$, con P_3:

$$\begin{array}{lcl} E & \rightarrow & E \texttt{ \&\& } C \mid C \\ C & \rightarrow & C \texttt{ == } S \mid S \\ S & \rightarrow & S \texttt{ + } T \mid T \\ T & \rightarrow & \texttt{true} \mid \texttt{false} \\ & \mid & \texttt{num} \mid \texttt{string} \\ & \mid & \texttt{(} E \texttt{)} \\ & \mid & \texttt{!} T \end{array}$$

Convertir a G_8 en una *traducción dirigida por sintaxis*, agregando atributos y reglas semánticas de manera que se imprima la evaluación de la expresión, o se indique que hubo un error de tipado y cuál fue este (e.g., "ERROR: Se quiso sumar un número con un booleano"), además, se deberá tener registro del tipo de cada subexpresión en todo momento.
En cuanto a los operadores, el de igualdad (`==`) requerirá que ambos operandos sean del mismo tipo, el de conjunción (`&&`) y negación (`!`) esperan booleanos, mientras que la suma (`+`) espera que ambos lados sean numéricos, o, si alguno de ellos es string, el otro puede ser de cualquier tipo y resultado de la operación será de tipo cadena.