EXAMEN DE TRADUCTORES (2° Ingeniería Informática, final febrero-2012)

FACULTAD DE INFORMÁTICA

Apellidos, nombre: **GRUPO:**
D.N.I.:
Este enunciado y todos los folios usados deben entregarse al salir

Parte I: PREGUNTAS TIPO TEST. 30 %.
Cada respuesta correcta vale 0.2 puntos.
Cada dos respuestas incorrectas anulan una correcta.

1. Elegir, entre los siguientes, el tipo de *máquina abstracta* más adecuado para realizar el **análisis sintáctico** de un lenguaje de programación:
   a) *Autómata Finito*, pues siempre necesitaremos analizar las palabras del lenguaje.
   b) *Autómata de Pila*, a pesar de que la mayoría de los lenguajes de programación no son *libres de contexto* sino *sensibles al contexto*.
   c) *Autómata Linealmente Acotado*, puesto que la mayoría de los lenguajes de programación tienen restricciones contextuales (por ejemplo, la declaración de variables).
2. Elige la frase correcta acerca del **análisis de léxico**:
   a) Suele ser una función a la que llama el *analizador sintáctico* cada vez que necesita un token, aunque podría no realizarse de forma explícita y dejar el reconocimiento de palabras como parte del análisis sintáctico.
   b) Suele generar un fichero explícito de tokens que constituye la entrada del *analizador sintáctico*.
   c) Se realiza mediante la simulación de *autómatas de pila*, que proporcionan la potencia suficiente para las tareas de E/S.
3. Un compilador interpretado consiste en:
   a) Un compilador que da la opción de generar *código máquina* o *código intermedio* dependiendo de las necesidades del usuario.
   b) Un compilador que genera código escrito en *lenguaje de alto nivel* que posteriormente es interpretado por algún intérprete adecuado existente en el sistema.
   c) Un compilador que genera *código intermedio* que posteriormente es interpretado por una **máquina virtual**.
4. Con respecto al código generado por un compilador:
   a) Podemos implementarlo de manera que exista garantía de que siempre genera *código óptimo*, aunque generalmente no merece la pena.
   b) Nunca podremos implementar un compilador que genere código óptimo en ninguna situación, puesto que se trata de un problema *NP-completo*.
   c) Nunca podremos implementar un compilador que garantice la generación de código óptimo en todos los casos.
5. En el proceso de **arranque** debe darse la circunstancia de que:
   a) Coinciden el *lenguaje de implementación* y el *lenguaje destino*.
   b) Coinciden el *lenguaje de implementación* y el *lenguaje fuente*.
   c) Coinciden el *lenguaje de implementación* con el de una *máquina virtual* existente en el sistema.
6. Un **preprocesador** es:
   a) Una herramienta que toma como entrada varios ficheros escritos en lenguaje de alto nivel y permite traducirlos a código máquina.
   b) Un traductor cuyo *lenguaje fuente* está constituido por una serie de macros y su *lenguaje destino* es una forma extendida de algún lenguaje de alto nivel.
   c) Un traductor cuyo *lenguaje fuente* es una forma extendida de un lenguaje de alto nivel y su *lenguaje destino* es la forma estándar del mismo lenguaje.
7. Señalar la razón por la que se considera qua las **técnicas de recuperación de errores** son importantes:
   a) Porque permiten obtener un código objeto sin los errores introducidos por el usuario.
   b) Porque permiten informar al usuario de una forma más precisa.
   c) Porque posibilitan la detección de más de un error.

8. Una **gramática recursiva por la izquierda**:
   a) No puede ser *LL*, ni *LR*.
   b) No puede ser *LL*, aunque sí *SLR*.
   c) No puede ser *SLR*, aunque sí *LR-Canónica*.
9. Las **gramáticas LL** y **LR**:
   a) Pueden tener *λ-reglas* y ser *ambiguas*.
   b) Pueden tener *λ-reglas* pero no pueden ser *ambiguas*.
   c) No pueden tener *λ-reglas* ni ser *ambiguas*.
10. Elegir la opción correcta:
    a) El *método de análisis LL* es ascendente y predictivo.
    b) El *método de análisis LR* es ascendente y no predictivo.
    c) Los *métodos de análisis LL y LR* son ambos predictivos, de manera que el primero funciona obteniendo las derivaciones por la izquierda desde el símbolo inicial de la gramática y el segundo obteniendo las reducciones por la izquierda a partir de la cadena de entrada.
11. El **tratamiento de errores** a nivel de frase en cualquiera de los métodos estudiados:
    a) Es un método sistemático que no varía de una gramática a otra.
    b) Requiere la determinación de posibles errores que puedan producirse en el lenguaje fuente, y para ello podemos hacer una llamada a un procedimiento particular en cada casilla vacía de la tabla de análisis.
    c) Es un método que requiere la determinación previa de todas las posibles frases erróneas que puedan aparecer en un programa y que, además, permite corregirlas.
12. Supongamos que hemos calculado la colección LR(0) y la tabla SLR para la siguiente gramática:

$$E \quad \rightarrow \quad E * E \mid E + E \mid id$$

    de modo que los conjuntos I_5 e I_6 contienen los siguientes items:

$$I_5 = \{E \rightarrow E * E\bullet, E \rightarrow E \bullet * E, E \rightarrow E \bullet + E\}$$
$$I_6 = \{E \rightarrow E \bullet * E, E \rightarrow E + E\bullet, E \rightarrow E \bullet + E\}$$

    produciéndose una tabla SLR con conflictos. Elegir la acción adecuada para la casilla del estado 6 y el símbolo de entrada *:
    a) d3.
    b) r1.
    c) r2.
13. Una **gramática L-Atribuida**:
    a) Puede tener atributos sintetizados y heredados de cualquier otro símbolo de la gramática. De hecho, cualquier gramática *L-Atribuida* es también *S-Atribuida*.
    b) Puede tener atributos sintetizados y heredados sólo de hermanos izquierdos. De hecho, cualquier gramática *L-Atribuida* es también *S-Atribuida*.
    c) Puede tener atributos sintetizados. De hecho, cualquier gramática *S-Atribuida* es también *L-Atribuida*.
14. La **tabla de símbolos**:
    a) Es una estructura de datos útil para analizar la *semántica* de un lenguaje de programación, aunque en ningún caso se usa en la *comprobación de tipos*.
    b) Es una estructura de datos que resulta útil para el almacenamiento de las variables y sus atributos, de manera que suele pasar información de las *declaraciones* a los *usos*.
    c) Es una estructura de datos que sirve para almacenar información semántica, de forma que dicha información se suele añadir a lo largo de la *fase de síntesis* para usarla posteriormente en la *fase de análisis*.
15. Con respecto a la **traducción de expresiones y sentencias** de un lenguaje de programación:
    a) Si son necesarios *atributos heredados* y realizamos un *análisis LR*, sólo podremos llevarlo a cabo accediendo directamente a valores intermedios de la pila de análisis.
    b) Si son necesarios *atributos sintetizados* y realizamos un *análisis LL*, no podremos llevar a cabo la traducción en ningún caso.
    c) Si son necesarios *atributos heredados* y realizamos un *análisis LL*, no podremos llevar a cabo la traducción en ningún caso.

Parte II: PREGUNTAS CORTAS. 10 %.

1. Si una gramática G es LR-Canónica pero no es LALR, en la tabla LALR sólo pueden aparecer conflictos reducción/reducción y no desplazamiento/reducción. Proponer un ejemplo (concreto o genérico) en el que al obtener un estado I_{ij} para una tabla LALR a partir de dos estados sin conflictos I_i e I_j de una colección LR(1), en el primero (el I_{ij}) aparezca un conflicto reducción/reducción.

2. Dada la gramática G con $V_T = \{int, float, ident, , \}$, $V_N = \{DECLARACION, TIPO, LISTA\_VAR\}$, símbolo inicial $DECLARACION$ y el siguiente conjunto de producciones:

$$\begin{array}{lcl} DECLARACION & \rightarrow & TIPO\ LISTA\_VAR \\ TIPO & \rightarrow & int \\ & | & float \\ LISTA\_VAR & \rightarrow & ident\ ,\ LISTA\_VAR \\ & | & ident \end{array}$$

   ¿Es G una gramática **propia**? ¿Por qué? ¿Puede ser una gramática no-propia LR? ¿Y LL? Justificar las respuestas.

Parte III: PROBLEMAS. 60 %

La siguiente gramática G, con $V_T = \{(,), :=, \mathbf{id}, \mathbf{num}, ,\}$, $V_N = \{, L, R\}$, símbolo inicial A y el siguiente conjunto P de producciones:

$$\begin{array}{lcl} A & \rightarrow & (\ L\ ) := (\ R\ ) \\ L & \rightarrow & \mathbf{id} \\ & | & L\ ,\ \mathbf{id} \\ R & \rightarrow & \mathbf{num} \\ & | & R\ ,\ \mathbf{num} \end{array}$$

permite definir asignaciones múltiples (es una característica de algunos lenguajes de programación, como Perl). Por ejemplo, la siguiente sentencia:

```
(a, b) := (1, 2)
```

asigna simultáneamente a la variable *a* el valor *1*, y a la variable *b* el valor *2*. Responder a las siguientes cuestiones:

1. *(1 puntos)* Decir, justificando la respuesta, y sin construir ninguna tabla de análisis, si G puede ser LL(1). En caso de que no lo sean, realizar las transformaciones necesarias para llegar a una gramática equivalente que pueda serlo.

2. *(1 punto)* Verificar si G o la gramática equivalente obtenida en el apartado anterior es LL(1), calculando los conjuntos PRIMERO y SIGUIENTE, los conjuntos *predict*, y la tabla de análisis.

3. *(0.5 puntos)* Simular el comportamiento del algoritmo de análisis LL(1), usando la tabla obtenida en el apartado anterior, para la cadena $w \equiv (a, b)(1, 2)$, realizando la recuperación de errores en modo pánico en caso de error.

4. *(1.25 puntos)* Obtener la colección LR(1) para la gramática G inicial y construir la tabla LR-canónica. Indicar si G es una gramática LR-canónica justificando la respuesta.

5. *(0.75 puntos)* Indicar si G es una gramática LALR y/o SLR, justificando la respuesta, y sin calcular ninguna colección de ítems adicional.

6. *(1.5 puntos)* Dar una *definición dirigida por la sintaxis* usando la gramática G para realizar la traducción de las asignaciones múltiples. La definición debe construir dos listas, una con los identificadores de la parte izquierda de la asignación, y otra con los valores de la parte derecha. También debe comprobar que las dos listas tienen la misma longitud, y en caso contrario debe indicar un error semántico. Las funciones que se pueden usar son:

   - $longitud([e_1, e_2, \ldots, e_n]) = n$
   - $iesimo([e_1, e_2, \ldots, e_n], i) = e_i$
   - $añadir([e_1, e_2, \ldots, e_n], e) = [e_1, e_2, \ldots, e_n, e]$
   - $crealista(e) = [e]$
   - $genasig(a, b)$ genera el código de una asignación simple $a := b$.
   - $error(mensaje)$

   Para realizar este apartado es necesario:

   - indicar el número y tipo de atributos asociado a cada símbolo de G.
   - asociar a cada regla de producción de G las acciones semánticas necesarias.
   - decorar el árbol sintáctico correspondiente a la cadena $w \equiv (a, b) := (1, 2)$.
   - indicar si G es S-atribuida y/o L-atribuida, justificando la respuesta.

SOLUCIONES.

Parte I: PREGUNTAS TIPO TEST.

1-b ; 2-a ; 3-c ; 4-c ; 5-b ; 6-c ; 7-c ; 8-b ; 9-b ; 10-c ; 11-b ; 12-a ; 13-c ; 14-b ; 15-a.

Parte II: PREGUNTAS CORTAS.

Ejercicio 1.

Supongamos los siguientes conjuntos de ítems en la colección LR(1):

- $I_i: \{ \ldots, [A \to \alpha\cdot, a], [B \to \beta\cdot, b], \ldots \}$
- $I_j: \{ \ldots, [A \to \alpha\cdot, c], [B \to \beta\cdot, a], \ldots \}$

Suponiendo que la gramática es LR-canónica, estos conjuntos no deben presentar ningún conflicto. Por tanto, $a \neq b$ y $c \neq a$. Al unificar los conjuntos I_i e I_j obtenemos:

$$I_{ij}: \{ \ldots, [A \to \alpha\cdot, a/c], [B \to \beta\cdot, b/a], \ldots \}$$

dando lugar a un conflicto del tipo reduce/reduce, ya que la tabla de análisis tendrá las acciones reduce $A \to \alpha$ y reduce $B \to \beta$ en la entrada $M[ij, a]$. Por tanto, es posible la aparición de conflictos reduce/reduce en la tabla LALR, partiendo de una gramática LR-canónica.

Suponiendo, de nuevo, que la gramática es LR-canónica, para que exista un conflicto desplaza/reduce en la tabla LALR, debería obtenerse en I_{ij} un par de ítems del tipo:

$$I_{ij}: \{ \ldots, [A \to \alpha \cdot a\beta, x/y],\ [B \to \beta\cdot, a/z] \ldots \}$$

Pero para que esto sea posible, debe existir un par de ítems $[A \to \alpha \cdot a\beta, x]$ y $[B \to \beta\cdot, a]$ previamente en I_i o I_j, de modo que la gramática tampoco sería LR-canónica, ya que el conflicto desplaza/reduce existiría en uno de los dos conjuntos, contradiciendo el axioma de partida. Por tanto, no pueden aparecer conflictos desplaza/reduce en la tabla LALR si la gramática de partida es LR-canónica.

Ejercicio 2.

La gramática del enunciado es propia, puesto que es λ-libre, libre de ciclos y no tiene símbolos inútiles.

Por otra parte, una gramática no-propia puede ser LR y LL:

- Para que sea LR, la gramática debe ser no ambigua. Si, por ejemplo, la gramática tiene una λ-transición, no es propia, pero sin embargo puede ser no ambigua, y en consecuencia, LR.
- Para que sea LL, la gramática debe ser no ambigua, no recursiva por la izquierda y debe estar factorizada. Durante la factorización se pueden producir λ-transiciones, que harán que la gramática no sea propia, aunque sea LL.

Parte III: PROBLEMAS.

Apartado 1. La gramática no es LL(1) porque es recursiva por la izquierda para los terminales L y R. Se puede aplicar el algoritmo de eliminación de la recursividad izquierda de toda la gramática, ya que G es propia.

El orden de los no terminales es: $V_N = \{A, L, R\}$.

Con la eliminación de la recursividad inmediata de L se obtienen las reglas:

$$L \to \textbf{id} \mid \textbf{id}\ L'$$
$$L' \to , \textbf{id} \mid , \textbf{id}\ L'$$

Con la eliminación de la recursividad inmediata de R se obtienen las reglas:

$$R \to \textbf{num} \mid \textbf{num}\ R'$$
$$R' \to , \textbf{num} \mid , \textbf{num}\ R'$$

La gramática obtenida tras la eliminación de la recursividad es:

$$
\begin{aligned}
A &\rightarrow (\ L\ ) := (\ R\ ) \\
L &\rightarrow \textbf{id} \mid \textbf{id}\ L' \\
L' &\rightarrow , \textbf{id} \mid , \textbf{id}\ L' \\
R &\rightarrow \textbf{num} \mid \textbf{num}\ R' \\
R' &\rightarrow , \textbf{num} \mid , \textbf{num}\ R'
\end{aligned}
$$

Antes de aplicar el algoritmo LL(1) es necesario factorizar la gramática:

$$
\begin{aligned}
A &\rightarrow (\ L\ ) := (\ R\ ) \\
L &\rightarrow \textbf{id}\ L'' \\
L' &\rightarrow , \textbf{id}\ L'' \\
L'' &\rightarrow \lambda \mid L' \\
R &\rightarrow \textbf{num}\ R'' \\
R' &\rightarrow , \textbf{num}\ R'' \\
R'' &\rightarrow \lambda \mid R'
\end{aligned}
$$

Se puede simplificar la gramática anterior eliminando los no terminales L' y R' y renombrando L'' y R'':

$$
\begin{aligned}
&(1) \quad A \rightarrow (\ L\ ) := (\ R\ ) \\
&(2) \quad L \rightarrow \textbf{id}\ L' \\
&(3) \quad L' \rightarrow \lambda \\
&(4) \quad L' \rightarrow , \textbf{id}\ L' \\
&(5) \quad R \rightarrow \textbf{num}\ R' \\
&(6) \quad R' \rightarrow \lambda \\
&(7) \quad R' \rightarrow , \textbf{num}\ R'
\end{aligned}
$$

Apartado 2. Los conjuntos PRIMERO y SIGUIENTE de la última gramática obtenida en el apartado anterior son:

PRIMERO(A) = { (}
PRIMERO(L) = { **id** }
PRIMERO(L') = { λ , }
PRIMERO(R) = { **num** }
PRIMERO(R') = { λ , }

SIGUIENTE(A) = { $ }
SIGUIENTE(L) = {) }
SIGUIENTE(L') = {) }
SIGUIENTE(R) = {) }
SIGUIENTE(R') = {) }

Los conjuntos Predict de las reglas de producción son:

Predict(1) = { (}
Predict(2) = { **id** }
Predict(3) = {) }
Predict(4) = { , }
Predict(5) = { **num** }
Predict(6) = {) }
Predict(7) = { , }

Y la tabla de análisis LL(1) es:

No terminal	Símbolo de Entrada						
	id	**num**	(	)	:=	,	$
A			1				
L	2						
L'				3		4	
R		5					
R'				6		7	

Apartado 3. La simulación de la cadena errónea $w = (a, b)(1, 2)$ se muestra a continuación:

Pila	Entrada	Acción
$\$A$	$(\mathbf{id}, \mathbf{id})(\mathbf{num}, \mathbf{num})\$$	$A \to (L) := (R)$
$\$)R(:=)L($	$(\mathbf{id}, \mathbf{id})(\mathbf{num}, \mathbf{num})\$$	
$\$)R(:=)L$	$\mathbf{id}, \mathbf{id})(\mathbf{num}, \mathbf{num})\$$	$L \to \mathbf{id}L'$
$\$)R(:=)L'\mathbf{id}$	$\mathbf{id}, \mathbf{id})(\mathbf{num}, \mathbf{num})\$$	
$\$)R(:=)L'$	$, \mathbf{id})(\mathbf{num}, \mathbf{num})\$$	$L' \to , \mathbf{id}L'$
$\$)R(:=)L'\mathbf{id},$	$, \mathbf{id})(\mathbf{num}, \mathbf{num})\$$	
$\$)R(:=)L'\mathbf{id}$	$\mathbf{id})(\mathbf{num}, \mathbf{num})\$$	
$\$)R(:=)L'$	$)(\mathbf{num}, \mathbf{num})\$$	$L \to \lambda$
$\$)R(:=)$	$)(\mathbf{num}, \mathbf{num})\$$	
$\$)R(:=$	$(\mathbf{num}, \mathbf{num})\$$	**Error**: desapilar := e informar de que falta el operador de asignación.
$\$)R($	$(\mathbf{num}, \mathbf{num})\$$	
$\$)R$	$\mathbf{num}, \mathbf{num})\$$	$R \to \mathbf{num}R'$
$\$)R'\mathbf{num}$	$\mathbf{num}, \mathbf{num})\$$	
$\$)R'$	$, \mathbf{num})\$$	$R' \to , \mathbf{num}R'$
$\$)R'\mathbf{num},$	$, \mathbf{num})\$$	
$\$)R'\mathbf{num}$	$\mathbf{num})\$$	
$\$)R'$	$)\$$	$R' \to \lambda$
$\$)$	$)\$$	
$\$$	$\$$	

Apartado 4. La colección de ítems LR(1) de la gramática inicial es la siguiente:

$I_0 = \{ [\ A' \to \cdot A\ ,\ \$\ ]$
$\quad [\ A \to \cdot(L) := (R)\ ,\ \$\ ]\ \}$

$I_1 = \textsc{Goto}(I_0, A) = \{ [\ A' \to A\cdot\ ,\ \$\ ]\ \}$

$I_2 = \textsc{Goto}(I_0, () = \{ [\ A \to (\cdot L) := (R)\ ,\ \$\ ]$
$\quad [\ L \to \cdot\mathbf{id}\ ,\ )/,\ ]$
$\quad [\ L \to \cdot L, \mathbf{id}\ ,\ )/,\ ]\ \}$

$I_3 = \textsc{Goto}(I_2, L) = \{ [\ A \to (L\cdot) := (R)\ ,\ \$\ ]$
$\quad [\ L \to L\cdot, \mathbf{id}\ ,\ )/,\ ]\ \}$

$I_4 = \textsc{Goto}(I_2, \mathbf{id}) = \{ [\ L \to \mathbf{id}\cdot\ ,\ )/,\ ]\ \}$

$I_5 = \textsc{Goto}(I_3,)) = \{ [\ A \to (L)\cdot := (R)\ ,\ \$\ ]\ \}$

$I_6 = \textsc{Goto}(I_3, ,) = \{ [\ L \to L, \cdot\mathbf{id}\ ,\ )/,\ ]\ \}$

$I_7 = \textsc{Goto}(I_5, :=) = \{ [\ A \to (L) := \cdot(R)\ ,\ \$\ ]\ \}$

$I_8 = \textsc{Goto}(I_6, \mathbf{id}) = \{ [\ L \to L, \mathbf{id}\cdot\ ,\ )/,\ ]\ \}$

$I_9 = \textsc{Goto}(I_7, () = \{ [\ A \to (L) := (\cdot R)\ ,\ \$\ ]$
$\quad [\ R \to \cdot\mathbf{num}\ ,\ )/,\ ]$
$\quad [\ R \to \cdot R, \mathbf{num}\ ,\ )/,\ ]\ \}$

$I_{10} = \textsc{Goto}(I_9, R) = \{ [\ A \to (L) := (R\cdot)\ ,\ \$\ ]$
$\quad [\ R \to R\cdot, \mathbf{num}\ ,\ )/,\ ]\ \}$

$I_{11} = \textsc{Goto}(I_9, \mathbf{num}) = \{ [\ R \to \mathbf{num}\cdot\ ,\ )/,\ ]\ \}$

$I_{12} = \textsc{Goto}(I_{10},)) = \{ [\ A \to (L) := (R)\cdot\ ,\ \$\ ]\ \}$

$I_{13} = \textsc{Goto}(I_{10}, ,) = \{ [\ R \to R, \cdot\mathbf{num}\ ,\ )/,\ ]\ \}$

$I_{14} = \textsc{Goto}(I_{13}, \mathbf{num}) = \{ [\ R \to R, \mathbf{num}\cdot\ ,\ )/,\ ]\ \}$

Se numeran las reglas de producción de la siguiente forma:

(1) $A \to (\ L\ ) := (\ R\ )$
(2) $L \to \mathbf{id}$
(3) $L \to L\ ,\ \mathbf{id}$
(4) $R \to \mathbf{num}$
(5) $R \to R\ ,\ \mathbf{num}$

La tabla de análisis LR-canónica es la siguiente:

Estado	Acción							Ir-a		
	(	)	,	:=	**id**	**num**	$	A	L	R
0	d2							1		
1							acc			
2					d4				3	
3		d5	d6							
4		r2	r2							
5				d7						
6					d8					
7	d9									
8		r3	r3							
9						d11				10
10		d12	d13							
11		r4	r4							
12							r1			
13						d14				
14		r5	r5							

Al no presentar conflictos la tabla, se deduce que la gramática G es LR-canónica.

Apartado 5. Para comprobar si la gramática es LALR debemos intentar, en primer lugar, unificar estados de la colección LR(1). Como podemos observar, no es posible unificar estados, ya que todos son distintos en los ítems punteados. Por tanto, la colección LALR es exactamente igual a la LR(1), con lo que la gramática también es LALR.

Para comprobar si es la gramática es SLR, debemos analizar los ítems de la colección LALR (igual que la LR(1)), para comprobar si los símbolos de anticipación coinciden con los símbolos de los conjuntos SIGUIENTE de los no terminales a la izquierda de los ítems. Los conjuntos PRIMERO y SIGUIENTE de la gramática inicial son:

PRIMERO(A) = { (}
PRIMERO(L) = { **id** }
PRIMERO(R) = { **num** }

SIGUIENTE(A) = { $ }
SIGUIENTE(L) = {) , }
SIGUIENTE(R) = {) , }

Y se puede observar que los conjuntos de símbolos coinciden:

- [$L \to \mathbf{id}\cdot$,)/,] en I_4.
- [$L \to L, \mathbf{id}\cdot$,)/,] en I_8.
- [$R \to \mathbf{num}\cdot$,)/,] en I_{11}.
- [$A \to (L) := (R)\cdot$, $] en I_{12}.
- [$R \to R, \mathbf{num}\cdot$,)/,] en I_{14}.

Por tanto, la gramática también es SLR, y tiene la misma tabla de análisis.

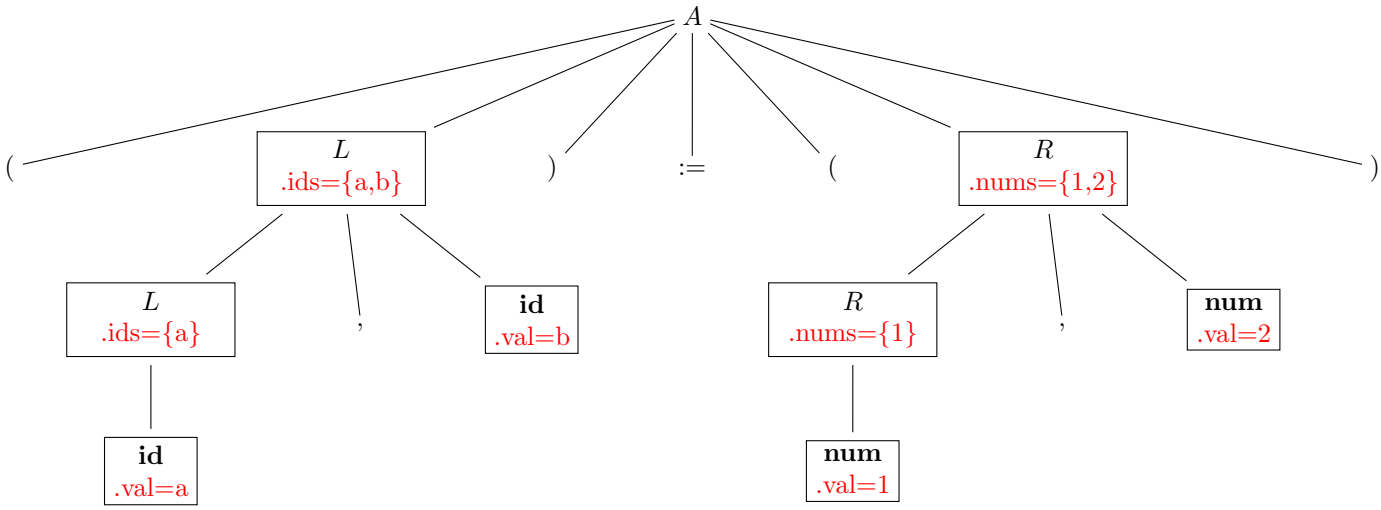
Apartado 6. Los atributos de la definición dirigida por la sintaxis son los siguientes:

Símbolo	**Atributo**	**Tipo**	**Comentario**
L	ids	`Lista`	Lista de identificadores.
R	nums	`Lista`	Lista de números.
id	val	`string`	Lexema del identificador, en formato string.
num	val	`int`	Lexema del número, en formato entero.

La definición dirigida por la sintaxis para la generación del código de la asignación múltiple es:

Regla de producción	**Acción**
$A \to (L) := (R)$	`if (longitud(`L`.ids) != longitud(`R`.nums)) error("Longitud distinta");` `else for i=1 to longitud(`L`.ids) genasig(iesimo(`L`.ids,i),iesimo(`R`.nums,i));`
$L \to \mathbf{id}$	L`.ids = crealista(`**id**`.val);`
$L \to L_1, \mathbf{id}$	L`.ids = añadir(`L_1`.ids, `**id**`.val);`
$R \to \mathbf{num}$	R`.nums = crealista(`**num**`.val);`
$R \to R_1, \mathbf{num}$	R`.nums = añadir(`R_1`.nums, `**num**`.val);`

La asignación del enunciado genera el siguiente árbol de análisis decorado:



Todos los atributos son **sintetizados**, ya que se obtienen a partir de los valores de los atributos de los nodos hijo. Por tanto, la gramática es **S-atribuida**, y en consecuencia, también es **L-atribuida**.