EXAMEN DE COMPILADORES (2° **Grado en Informática, final septiembre-2011**)

FACULTAD DE INFORMÁTICA

SOLUCIONES.

Parte I: PREGUNTAS TIPO TEST.

1. a)
2. b)
3. c)
4. b)
5. a)
6. b)
7. a)
8. c)
9. b)
10. a)
11. a) Explicación: $S \Rightarrow A \Rightarrow id = L; \Rightarrow id = L = L; \Rightarrow id = L = L = L; \Rightarrow id = L = L = \underline{id};$
12. a)
13. b)
14. c)
15. b)

Parte II: PREGUNTAS CORTAS.

1. Dada la gramática G con el siguiente conjunto de producciones:

$$\begin{array}{lcl} T & \rightarrow & B\ C \\ B & \rightarrow & int \mid float \\ C & \rightarrow & [\ num\ ]\ C \ \mid \lambda \end{array}$$

que genera el lenguaje para declarar arrays en C, enumerar los tokens de G, indicando cuáles tendrían atributo asociado en caso de que usáramos la gramática para realizar un compilador. Finalmente, dada la entrada

```
int[45][2]
```

indicar qué información proporcionaría el analizador léxico al sintáctico.

Solución:

La gramática G tiene los cinco tokens siguientes:

TOKEN	**ATRIBUTO**
INT	–
$FLOAT$	–
NUM	$valor$
$CORI$	–
$CORD$	–

donde $CORI$ representa al terminal '[' y $CORD$ al terminal ']'. El atributo $valor$ del token NUM contiene el lexema del número reconocido. Dada la entrada $w \equiv$ `int[45][2]`, el analizador léxico devuelve la siguiente secuencia de tokens:

INT , $CORI$, NUM_1 , $CORD$, $CORI$, NUM_2 , $CORD$

donde NUM_1 y NUM_2 son dos instancias distintas de token de tipo NUM. Los atributos asociados a estas instancias son $NUM_1.valor = 45$ y $NUM_2.valor = 2$.

2. Supongamos que hemos calculado la colección LR(0) y la tabla SLR para la siguiente gramática:

$$E \ \rightarrow \ E \wedge E \mid E \vee E \mid id$$

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

$$I_5 = \{E \rightarrow E \wedge E\bullet, E \rightarrow E \bullet \wedge E, E \rightarrow E \bullet \vee E\}$$
$$I_6 = \{E \rightarrow E \bullet \wedge E, E \rightarrow E \vee E\bullet, E \rightarrow E \bullet \vee E\}$$

produciéndose en la tabla SLR los siguientes conflictos:

ESTADO	**accion**		...
	$\wedge$	$\vee$	...
...	...	...	
5	$r1/d3$	$r1/d4$	...
6	$r2/d3$	$r2/d4$	...

Resolverlos dando mayor prioridad al operador $\wedge$ que al $\vee$ y considerando que ambos son asociativos por la izquierda.

Solución:

El problema se puede dividir en dos partes:

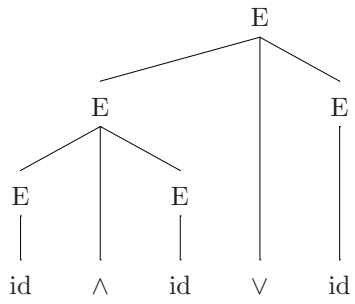
- Resolver los conflictos de acuerdo con la **prioridad** de los operadores.
- Resolver los conflictos de acuerdo con la **asociatividad** de los operadores.

Para estudiar la **prioridad** de los operadores, debemos seleccionar las acciones que permitan una interpretación adecuada de expresiones que combinen ambos tipos de operadores, es decir:

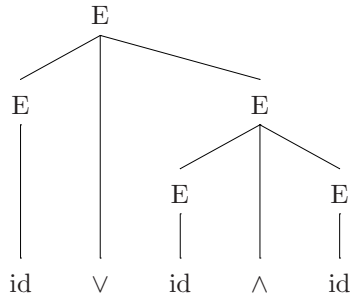
- $w_1 \equiv id \wedge id \vee id$

- $w_2 \equiv id \vee id \wedge id$

Conforme al enunciado del problema, la interpretación correcta de la primera expresión, usando una notación con paréntesis, es: $w_1 \equiv (id \wedge id) \vee id$. Para forzar esta interpretación hay que observar que en el estado I_5 se encuentra el ítem $E \rightarrow E \wedge E\bullet$. Este ítem representa la situación en la que se puede reducir una expresión con el operador $\wedge$ (regla de producción 1). Cuando el analizador haya reconocido la subexpresión de w_1 indicada entre paréntesis, se encontrará en el estado 5 y el siguiente token será $\vee$. En la tabla nos encontramos las opciones $r1$ y $d4$. Lo correcto es optar por $r1$, ya que implica dar prioridad a $\wedge$ frente a $\vee$, generando el árbol sintáctico:



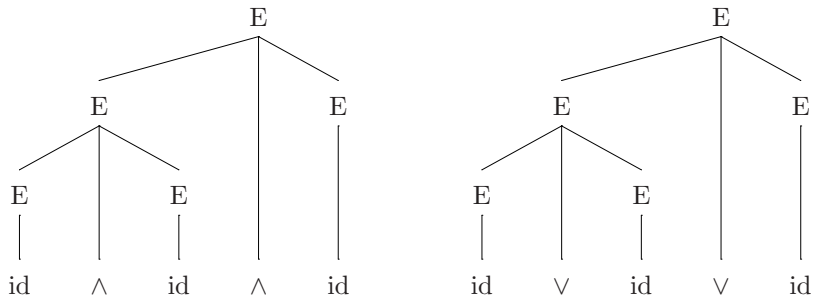
En el segundo caso, la interpretación correcta es $w_2 \equiv id \vee (id \wedge id)$. Debemos estudiar cuál es el comportamiento adecuado cuando el analizador ha reconocido una expresión de la forma $E \vee E$ y seguidamente se encuentra con el token $\wedge$ en la entrada. El estado I_6 contiene el ítem $E \rightarrow E \vee E\bullet$ (regla de producción 2), y en la columna $\wedge$ encontramos las acciones $r2$ y $d3$. Si optásemos por $r2$, estaríamos dando más prioridad a $\vee$ que a $\wedge$, y nuestra intención es justo la contraria. Elegimos $d3$ en este caso. Se puede observar que esta acción está asociada al ítem $E \rightarrow E \bullet \wedge E$, que representa la situación en la que se ha reconocido el primer operando de la subexpresión entre paréntesis de w_2 y se continúa pasando a la pila el operador $\wedge$. Se obtiene el árbol sintáctico:



La **asociatividad** de los operadores nos permite resolver los otros dos conflictos pendientes. En este caso tenemos que estudiar cuál debe ser el comportamiento con expresiones que usen un único operador:

- $w_3 \equiv id \wedge id \wedge id$
- $w_4 \equiv id \vee id \vee id$

El reconocimiento correcto de estas expresiones, teniendo en cuenta que la asociatividad es por la izquierda, es: $w_3 \equiv (id \wedge id) \wedge id$ y $w_4 \equiv (id \vee id) \vee id$. Siguiendo un razonamiento parecido al del primero caso, en el estado I_5, cuando la entrada es $\wedge$ hay que seleccionar la acción $r1$, mientras que en el estado I_6, cuando la entrada es $\vee$ hay que seleccionar la acción $r2$. De esta forma obtendríamos árboles sintácticos como los siguientes:



Por tanto, la tabla de análisis sin conflictos quedaría de la siguiente forma:

ESTADO	**accion**		...
	$\wedge$	$\vee$	...
...	...	...	
5	$r1$	$r1$	...
6	$d3$	$r2$	...

Parte III: PROBLEMAS.

Dada la gramática G, con $V_T = \{id, =, ;\}$, $V_N = \{S, A, L\}$, símbolo inicial S y el siguiente conjunto P de producciones:

$$\begin{array}{lcl} S & \rightarrow & S\ A \\ & | & A \\ A & \rightarrow & id\ =\ L\ ; \\ L & \rightarrow & id \\ & | & L\ =\ L \end{array}$$

responder a las siguientes cuestiones:

1. Decir, justificando las respuestas, y sin construir ninguna tabla de análisis, si G es:

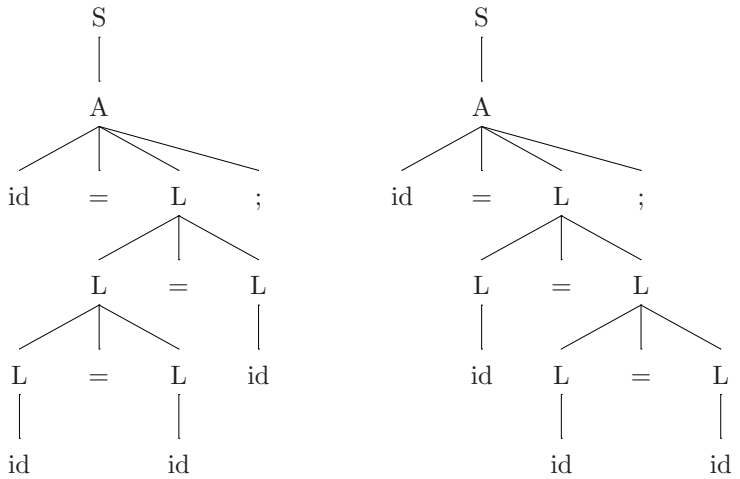
   a) Propia.
   b) Ambigua.
   c) LR-Canónica(1).
   d) LALR(1).
   e) SLR(1).
   f) LL(1).

   En caso de que G no cumpla alguna de estas propiedades, dar todas las razones que se conozcan para justificarlo.

   Solución:

   Una gramática es propia si no tiene símbolos inútiles, es λ-libre y libre de ciclos. La gramática G cumple todas esas condiciones, y por tanto **es propia**.

   La gramática **es ambigua**. Por ejemplo, la entrada $w \equiv$ `id = id = id = id;` tiene estos dos posibles árboles de derivación:



   La ambigüedad de la gramática se debe a que no refleja una asociatividad concreta para el operador =. Dicha ambigüedad da lugar a conflictos en las tablas de análisis. Por tanto, la gramática **no es LR-Canónica(1), LALR(1), SLR(1), ni LL(1)**. Por otra parte, la gramática es **recursiva por la izquierda** en las producciones de S y L, siendo esta otra razón por la que no es LL(1).

2. Construir los conjuntos *PRIMERO* y *SIGUIENTE* para cada no terminal, y *predict* para cada regla. Construir la tabla LL.

   Solución:

   $PRIMERO(S) = \{id\}$ $\quad$ $SIGUIENTE(S) = \{id, \$\}$
   $PRIMERO(A) = \{id\}$ $\quad$ $SIGUIENTE(A) = \{id, \$\}$
   $PRIMERO(L) = \{id\}$ $\quad$ $SIGUIENTE(L) = \{;, =\}$
   $predict(1) = \{id\}$ $\quad$ $predict(4) = \{id\}$
   $predict(2) = \{id\}$ $\quad$ $predict(5) = \{id\}$
   $predict(3) = \{id\}$

NO TERM	TERMINAL			
	id	=	;	$
S	1/2			
A	3			
L	4/5			

La gramática **no es LL(1)**, puesto que aparecen conflictos en la tabla, corroborando lo que se ha indicado en el apartado anterior.

3. Construir la colección LR(0) y la tabla de análisis SLR. Modificar esta tabla para conseguir que el operador de asignación (=) sea asociativo por la derecha. Si usáramos *Bison* para generar la tabla, ¿cómo podría conseguirse esa asociatividad por la derecha?

Solución:

Aumentamos la gramática con el nuevo símbolo inicial S' y la regla $S' \rightarrow S$.

$I_0 \equiv \{S' \rightarrow \bullet S\ ,\ S \rightarrow \bullet SA\ ,\ \ S \rightarrow \bullet A\ ,\ A \rightarrow \bullet id = L;\}$

$GOTO(I_0, S) \equiv I_1 \equiv \{S' \rightarrow S\bullet\ ,\ S \rightarrow S \bullet A\ ,\ A \rightarrow \bullet id = L;\}$

$GOTO(I_0, A) \equiv I_2 \equiv \{\ S \rightarrow A\bullet\}$

$GOTO(I_0, id) \equiv I_3 \equiv \{A \rightarrow id\bullet = L;\}$

$GOTO(I_1, A) \equiv I_4 \equiv \{S \rightarrow SA\bullet\}$

$GOTO(I_1, id) \equiv I_3$

$GOTO(I_3, =) \equiv I_5 \equiv \{A \rightarrow id = \bullet L;\ ,\ L \rightarrow \bullet id\ ,\ L \rightarrow \bullet L = L\}$

$GOTO(I_5, L) \equiv I_6 \equiv \{A \rightarrow id = L\bullet;\ ,\ L \rightarrow L\bullet = L\}$

$GOTO(I_5, id) \equiv I_7 \equiv \{L \rightarrow id\bullet\}$

$GOTO(I_6, ;) \equiv I_8 \equiv \{A \rightarrow id = L;\bullet\}$

$GOTO(I_6, =) \equiv I_9 \equiv \{L \rightarrow L = \bullet L\ ,\ L \rightarrow \bullet id\ ,\ L \rightarrow \bullet L = L\}$

$GOTO(I_9, id) \equiv I_7$

$GOTO(I_9, L) \equiv I_{10} \equiv \{\ L \rightarrow L = L\bullet\ ,\ L \rightarrow L\bullet = L\}$

$GOTO(I_{10}, =) \equiv I_9$

La tabla de análisis que se obtiene es la siguiente:

ESTADO	accion				ir_a		
	id	=	;	$	**S**	**A**	**L**
0	$d3$				1	2	
1	$d3$			*aceptar*		4	
2	$r2$			$r2$			
3		$d5$					
4	$r1$			$r1$			
5	$d7$						6
6		$d9$	$d8$				
7		$r4$	$r4$				
8	$r3$			$r3$			
9	$d7$						10
10		$d9/r5$	$r5$				

El conflicto del estado 10 corrobora que la gramática no es SLR. Para hacer que el operador `=` sea asociativo por la derecha hay que resolver el conflicto eligiendo la acción d9. En la pregunta corta número 2 se solicitaba que la asociatividad fuese por la izquierda. En este caso se pide por la derecha, razón por la cual se realiza el desplazamiento. En `Bison` podríamos resolver el conflicto indicando la directiva de asociatividad para el token que representa el símbolo `=`. Supongamos que dicho token es `EQ`. La directiva se emplearía de la siguiente forma:

```
%right EQ
```

4. Simular el algoritmo de análisis ascendente predictivo no recursivo para las cadena de entrada $w_1 \equiv a = b = c = d;$ y $w_2 \equiv a == b;$, realizando recuperación en modo pánico en caso de error. Comenzar la simulación en ambos casos sustituyendo cada lexema por el código de token correspondiente.

Solución:

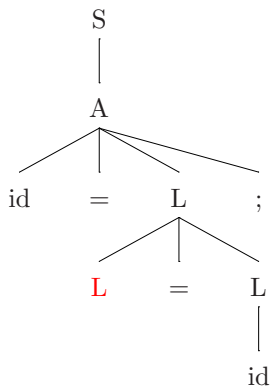
Se sustituyen en w_1 los lexemas por los tokens correspondientes, obteniendo $w_1 \equiv id = id = id = id;$. La simulación de w_1 se indica en la siguiente tabla:

Pila	**Entrada**	**Acción**
0	$id = id = id = id; \$$	$d3$
0 id 3	$= id = id = id; \$$	$d5$
0 id 3 $=$ 5	$id = id = id; \$$	$d7$
0 id 3 $=$ 5 id 7	$= id = id; \$$	$r4\ L \rightarrow id$
0 id 3 $=$ 5 L 6	$= id = id; \$$	$d9$
0 id 3 $=$ 5 L 6 $=$ 9	$id = id; \$$	$d7$
0 id 3 $=$ 5 L 6 $=$ 9 id 7	$= id; \$$	$r4\ L \rightarrow id$
0 id 3 $=$ 5 L 6 $=$ 9 L 10	$= id; \$$	$d9$
0 id 3 $=$ 5 L 6 $=$ 9 L 10 $=$ 9	$id; \$$	$d7$
0 id 3 $=$ 5 L 6 $=$ 9 L 10 $=$ 9 id 7	$; \$$	$r4\ L \rightarrow id$
0 id 3 $=$ 5 L 6 $=$ 9 L 10 $=$ 9 L 10	$; \$$	$r5\ L \rightarrow L = L$
0 id 3 $=$ 5 L 6 $=$ 9 L 10	$; \$$	$r5\ L \rightarrow L = L$
0 id 3 $=$ 5 L 6	$; \$$	$d8$
0 id 3 $=$ 5 L 6 ;8	$\$$	$r3\ A \rightarrow id = L;$
0 A 2	$\$$	$r2\ S \rightarrow A$
0 S 1	$\$$	$aceptar$

La simulación de $w_2 \equiv id == id;$, con tratamiento de errores en modo pánico, es la siguiente:

Pila	**Entrada**	**Acción**
0	$id == id; \$$	$d3$
0 id 3	$== id; \$$	$d5$
0 id 3 $=$ 5	$= id; \$$	$error!\ ir\_a[5, L] \neq NULL \wedge\ = \in SIG(L)$
0 id 3 $=$ 5 L 6	$= id; \$$	$d9$
0 id 3 $=$ 5 L 6 $=$ 9	$id; \$$	$d7$
0 id 3 $=$ 5 L 6 $=$ 9 id 7	$; \$$	$r4\ L \rightarrow id$
0 id 3 $=$ 5 L 6 $=$ 9 L 10	$; \$$	$r5\ L \rightarrow L = L$
0 id 3 $=$ 5 L 6	$; \$$	$d8$
0 id 3 $=$ 5 L 6 ; 8	$\$$	$r3\ A \rightarrow id = L;$
0 A 2	$\$$	$r2\ S \rightarrow A$
0 S 1	$\$$	$aceptar$

En el tratamiento de errores en modo pánico se quitan estados de la cima de la pila hasta encontrar uno que tenga definida la función ir_a para algún no terminal. En este caso $ir\_a[5, L] \neq NULL$ y además, $= \in SIG(L)$, de modo que puede continuar inmediatamente el análisis. Se obtiene el árbol sintáctico siguiente:



El nodo en rojo corresponde al no terminal L insertado artificialmente durante el tratamiento de errores.

5. Dar la definición dirigida por la sintaxis para, suponiendo que disponemos de una tabla de símbolos con los identificadores ya declarados, cada uno con su tipo correspondiente, comprobar que la asignación se hace entre tipos compatibles. Para ello, suponemos que cada identificador dispone de un atributo entrada que apunta a su posición en dicha tabla de símbolos, y que podemos hacer uso de una función `buscaTipo(apTS)` que devuelve el tipo del identificador apuntado por `apTS`. Indicar los atributos necesarios para cada símbolo, el tipo de cada uno de ellos (sintetizado o heredado) y las acciones semánticas para cada regla. Finalmente, evaluar el árbol de

derivación asociado a la cadena de entrada w_1 de la pregunta 4, suponiendo que todos los identificadores son de tipo entero. ¿Es esta gramática S-atribuida? ¿Y L-Atribuida? Justificar las respuestas.

Solución:

Podemos suponer que disponemos de una función:

`tipo compatible (tipo l, tipo r)`

que devuelve un código de tipo para una operación de asignación, siendo l el tipo de la parte izquierda y r el de la parte derecha de la asignación.

Suponemos que la función verifica la compatibilidad entre los tipos usando un criterio de ampliación basado en el árbol mostrado a la izquierda. El tipo del argumento `l` de la función `compatible` debe ser mayor o igual que el de `r` en la relación de orden establecida por este árbol. Si no es así, el tipo devuelto es `TERROR`. Si alguno de los argumentos toma el valor `TERROR`, el resultado es también `TERROR`. En caso contrario, devuelve el tipo del argumento `l`.

Se definen los siguientes atributos para los símbolos de la gramática:

Símbolo	Atributo	Tipo	Comentario
S	res	`boolean`	`true` si todas las asignaciones son compatibles.
A	res	`boolean`	`true` si la asignación múltiple es compatible.
L	$tipo$	Código de tipo	`TERROR` en caso de error de compatibilidad. En otro caso, `DOUBLE`, `FLOAT`, `LONG`, `INT` o `SHORT`
id	$apTS$	Índice de la tabla de símbolos.	

La definición dirigida por la sintaxis (DDS) para la comprobación de tipos se indica a continuación:

Regla de producción	Acción
$S \rightarrow S_1 A$	$S.res$ `=` $S_1.res$ `&&` $A.res$`;`
$S \rightarrow A$	$S.res$ `=` $A.res$`;`

$A \rightarrow id = L;$	`if (compatible(buscaTipo(`$id.apTS$`),`$L.tipo$`) != TERROR)`
	  $A.res$ `= true;`
	`else {`
	  $A.res$ `= false; error("Tipos incompatibles");`
	`}`

$L \rightarrow id$	$L.tipo$ `= buscaTipo(`$id.apTS$`);`
$L \rightarrow L_1 = L_2$	$L.tipo$ `= compatible(`$L_1.tipo$`,`$L_2.tipo$`);`

Con esta DDS, la gramática es S-atribuida y, por tanto, también es L-atribuida, ya que todos los atributos se calculan en función únicamente de los valores de los atributos de los nodos hijo.

El árbol decorado para la entrada w_1 del apartado 4 es el siguiente: