EXAMEN DE COMPILADORES (2° Grado en Informática, enero-2015)

FACULTAD DE INFORMÁTICA

SOLUCIONES

Parte I: PREGUNTAS TIPO TEST. 30 %.

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

Parte II: PREGUNTAS CORTAS. 10 %.

1. Obtenemos los conjuntos PRIMERO:

PRIMERO$(A) = \{\ a,\ b,\ c\ \}$
PRIMERO$(B) = \{\ b,\ \lambda,\ a,\ c\ \}$
PRIMERO$(C) = \{\ c,\ \lambda,\ a,\ b\ \}$

2. Obtenemos los conjuntos SIGUIENTE:

SIGUIENTE$(A) = \{\ \$,\ b,\ a,\ c\ \}$
SIGUIENTE$(B) = \{\ a,\ b,\ c\ \}$
SIGUIENTE$(C) = \{\ a,\ b,\ c\ \}$

3. Obtenemos los conjuntos PREDICT:

PREDICT$(A \to B\ C\ A) = \{\ a,\ b,\ c\ \}$
PREDICT$(A \to a) = \{\ a\ \}$
PREDICT$(B \to C\ A\ B) = \{\ a,\ b,\ c\ \}$
PREDICT$(B \to b) = \{\ b\ \}$
PREDICT$(B \to \lambda) = \{\ a,\ b,\ c\ \}$
PREDICT$(C \to A\ B\ C) = \{\ a,\ b,\ c\ \}$
PREDICT$(C \to c) = \{\ c\ \}$
PREDICT$(C \to \lambda) = \{\ a,\ b,\ c\ \}$

4. La gramática no puede ser LL(1) porque la intersección de los conjuntos PREDICT de cualquier par de reglas de un mismo no terminal no es vacía:

- Para el no terminal A:
  PREDICT$(A \to B\ C\ A) \cap$ PREDICT$(A \to a) = \{\ a\ \}$
- Para el no terminal B:
  PREDICT$(B \to C\ A\ B) \cap$ PREDICT$(B \to b) = \{\ b\ \}$
  PREDICT$(B \to C\ A\ B) \cap$ PREDICT$(B \to \lambda) = \{\ a,\ b,\ c\ \}$
  PREDICT$(B \to b) \cap$ PREDICT$(B \to \lambda) = \{\ b\ \}$
- Para el no terminal C:
  PREDICT$(C \to A\ B\ C) \cap$ PREDICT$(C \to c) = \{\ c\ \}$
  PREDICT$(C \to A\ B\ C) \cap$ PREDICT$(C \to \lambda) = \{\ a,\ b,\ c\ \}$
  PREDICT$(C \to c) \cap$ PREDICT$(C \to \lambda) = \{\ c\ \}$

Parte III: PROBLEMA. 60 %.

Apartado 1.

Obtenemos los conjuntos PRIMERO y SIGUIENTE que solicita el enunciado:

PRIMERO$(S) = \{$ `if`, `print` $\}$
PRIMERO$(C) = \{$ `id` $\}$
PRIMERO$(E) = \{$ `else` $\}$
SIGUIENTE$(S) = \{$ `else`, `$` $\}$
SIGUIENTE$(E) = \{$ `$`, `else` $\}$
SIGUIENTE$(C) = \{$ `then` $\}$

Para construir la tabla de análisis Ll(1), calculamos los conjuntos PREDICT:

PREDICT$(S \to$ `if` C `then` $S\ E) = \{$ `if` $\}$
PREDICT$(S \to$ `print str` $) = \{$ `print` $\}$
PREDICT$(C \to$ `id == num` $) = \{$ `id` $\}$
PREDICT$(E \to$ `else` $S) = \{$ `else` $\}$

De acuerdo con los conjuntos PREDICT anteriores, la tabla de análisis LL(1) es la siguiente:

NO TERM	**TERMINAL**								
	`if`	`then`	`print`	`str`	`id`	`==`	`num`	`else`	$
S	$S \to$ `if` C *then* S E		$S \to$ `print str`						
C					$C \to$ `id == num`				
E								$E \to$ `else` S	

Como se puede observar, no hay conflictos en la tabla y, por tanto, la gramática es LL(1).

Apartado 2.

A continuación se muestra el análisis de la entrada `if id num then else print str` con tratamiento de errores en modo pánico.

PILA	ENTRADA	ACCIÓN
\$ S	`if id num then else print str` \$	
\$ E S `then` C `if`	`if id num then else print str` \$	$S \to$ `if` C `then` S E
\$ E S `then` C	`id num then else print str` \$	
\$ E S `then num == id`	`id num then else print str` \$	$C \to$ `id == num`
\$ E S `then num ==`	`num then else print str` \$	**Error**: falta `==` en la entrada. Eliminar de la pila.
\$ E S `then num`	`num then else print str` \$	
\$ E S `then`	`then else print str` \$	
\$ E S	`else print str` \$	**Error:** `else`$\in$ SIG(S). Desapilar S.
\$ E	`else print str` \$	
\$ S `else`	`else print str` \$	$E \to$ `else` S
\$ S	`print str` \$	
\$ `str print`	`print str` \$	$S \to$ `print str`
\$ `str`	`str` \$	
\$	\$	**Fin**: entrada con errores. No aceptar.

Apartado 3.

Para comprobar si la gramática es SLR(1), se construyen los conjuntos de ítems de la colección LR(0):

$I_0 = \{\ S' \to \cdot\ S$
$S \to \cdot$ `if` C `then` S E
$S \to \cdot$ `print str` $\}$

$I_1 = \text{GOTO}(I_0, S) = \{\ S' \to S \cdot\ \}$

$I_2 = \text{GOTO}(I_0, \texttt{if}) = \{\ S \to$ `if` $\cdot\ C$ `then` S E
$C \to \cdot$ `id == num` $\}$

$I_3 = \text{GOTO}(I_0, \texttt{print}) = \{\ S \to$ `print` $\cdot$ `str` $\}$

$I_4 = \text{GOTO}(I_2, C) = \{\ S \to$ `if` $C\ \cdot$ `then` S $E\ \}$

$I_5 = \text{GOTO}(I_2, \texttt{id}) = \{\ C \to$ `id` $\cdot$ `==num` $\}$

$I_6 = \text{GOTO}(I_3, \texttt{str}) = \{\ S \to$ `print str` $\cdot\ \}$

$I_7 = \text{GOTO}(I_4, \texttt{then}) = \{\ S \to$ `if` C `then` $\cdot\ S$ E
$S \to \cdot$ `if` C `then` S E
$S \to \cdot$ `print str` $\}$

$I_8 = \text{GOTO}(I_5, \texttt{==}) = \{\ C \to$ `id ==` $\cdot$ `num` $\}$

$I_9 = \text{GOTO}(I_7, S) = \{\ S \to$ `if` C `then` $S\ \cdot\ E$
$E \to \cdot$ `else` $S\ \}$

$\text{GOTO}(I_7, \texttt{if}) = I_2$

$\text{GOTO}(I_7, \texttt{print}) = I_3$

$I_{10} = \text{GOTO}(I_8, \texttt{num}) = \{\ C \to$ `id == num` $\cdot\ \}$

$I_{11} = \text{GOTO}(I_9, E) = \{\ S \to$ `if` C `then` S $E\ \cdot\ \}$

$I_{12} = \text{GOTO}(I_9, \texttt{else}) = \{\ E \to$ `else` $\cdot\ S$
$S \to \cdot$ `if` C `then` S E
$S \to \cdot$ `print str` $\}$

$I_{13} = \text{GOTO}(I_{12}, S) = \{\ E \to$ `else` $S\ \cdot\ \}$

$\text{GOTO}(I_{12}, \texttt{if}) = I_2$

$\text{GOTO}(I_{12}, \texttt{print}) = I_3$

A partir de la colección anterior, se construye la siguiente tabla de análisis SLR:

Estado	Acción									Ir-a		
	`if`	`then`	`print`	`str`	`id`	`==`	`num`	`else`	$	S	C	E
0	d2		d3							1		
1									acc			
2					d5						4	
3				d6								
4		d7										
5						d8						
6								r2	r2			
7	d2		d3							9		
8							d10					
9								d12				11
10		r3										
11								r1	r1			
12	d2		d3							13		
13								r4	r4			

No hay conflictos en la tabla, de modo que podemos deducir que la gramática es SLR(1).

Apartado 4.

Por definición, toda gramática SLR(1) es también LALR(1) y LR(1).

Apartado 5.

La gramática no es ambigua porque:

1. No hay operadores binarios cuya precedencia y asociatividad no esté definida por las estructuras de la gramática.
2. Todas las sentencias de la gramática, derivadas de S, contienen el no terminal `else`, de modo que tampoco se produce la ambigüedad por la coincidencia de sentencias `if` y sentencias `if-else`.

Por tanto, aunque la gramática permite anidamiento de sentencias, no hay confusión sobre qué `if` corresponde a cada `else`: es siempre el inmediatamente anterior.

Apartado 6.

a) La definición dirigida por la sintaxis puede emplear los siguientes atributos:

Símbolo	**Atributo**	**Tipo**	**Comentario**
S	n_1	`int`	Nivel de anidamiento del propio nodo S.
S	n_2	`int`	Nivel de anidamiento del nodo S (puede ser un descendiente) en el que está el `print` que se imprime porque su condición es la que se cumple.
S	l	`int`	Identificador del `str` que se imprime, almacenado en una tabla de cadenas.
C	b	`boolean`	Valor lógico de la condición del propio nodo C.
E	n_1	`int`	Nivel de anidamiento del propio nodo E.
E	n_2	`int`	Nivel de anidamiento del nodo S (descendiente) en el que está el `print` que se imprime porque su condición es la que se cumple.
E	l	`int`	Identificador del `str` que se imprime, almacenado en una tabla de cadenas.
`id`	lex	`char*`	Cadena de caracteres del lexema de `id`.
`num`	v	`int`	Valor numérico del lexema de `num`.
`str`	l	`int`	Identificador del `str` que se imprime, almacenado en una tabla de cadenas.

Consideramos que el analizador léxico inserta en una tabla de cadenas la información relativa a los `str` que va encontrando. También consideramos que existe una función `getVal()` a la que se le pasa el lexema de un identificador y devuelve el valor de dicho identificador.

En cuanto a las reglas semánticas que evalúan los atributos, podemos plantear las siguientes:

Regla de producción	**Regla semántica**
$S' \rightarrow S$	$S.n_1$ = 0;
$S \rightarrow$ if C then S_1 E	$S_1.n_1$ = $S.n_1$; $E.n_1$ = $S.n_1$; if ($C.b$) { $S.l$ = $S_1.l$; $S.n_2$ = $S_1.n_2$; } else { $S.l$ = $E.l$; $S.n_2$ = $E.n_2$; }
$S \rightarrow$ print str	$S.l$ = str.l; $S.n_2$ = $S.n_1$;
$C \rightarrow$ id == num	if (getVal(id.lex) == num.v) $C.b$ = true; else $C.b$ = false;
$E \rightarrow$ else S	$S.n_1$ = $E.n_1 + 1$; $E.n_2$ = $S.n_2$; $E.l$ = $S.l$;

b) El árbol anotado para la entrada propuesta se muestra en la página siguiente. Para su comprensión, se considera que la tabla de símbolos tiene los siguientes valores:

Tabla de símbolos

id	valor
a	1
b	2

En cuanto a la tabla de cadenas, contiene las siguientes entradas al final del análisis:

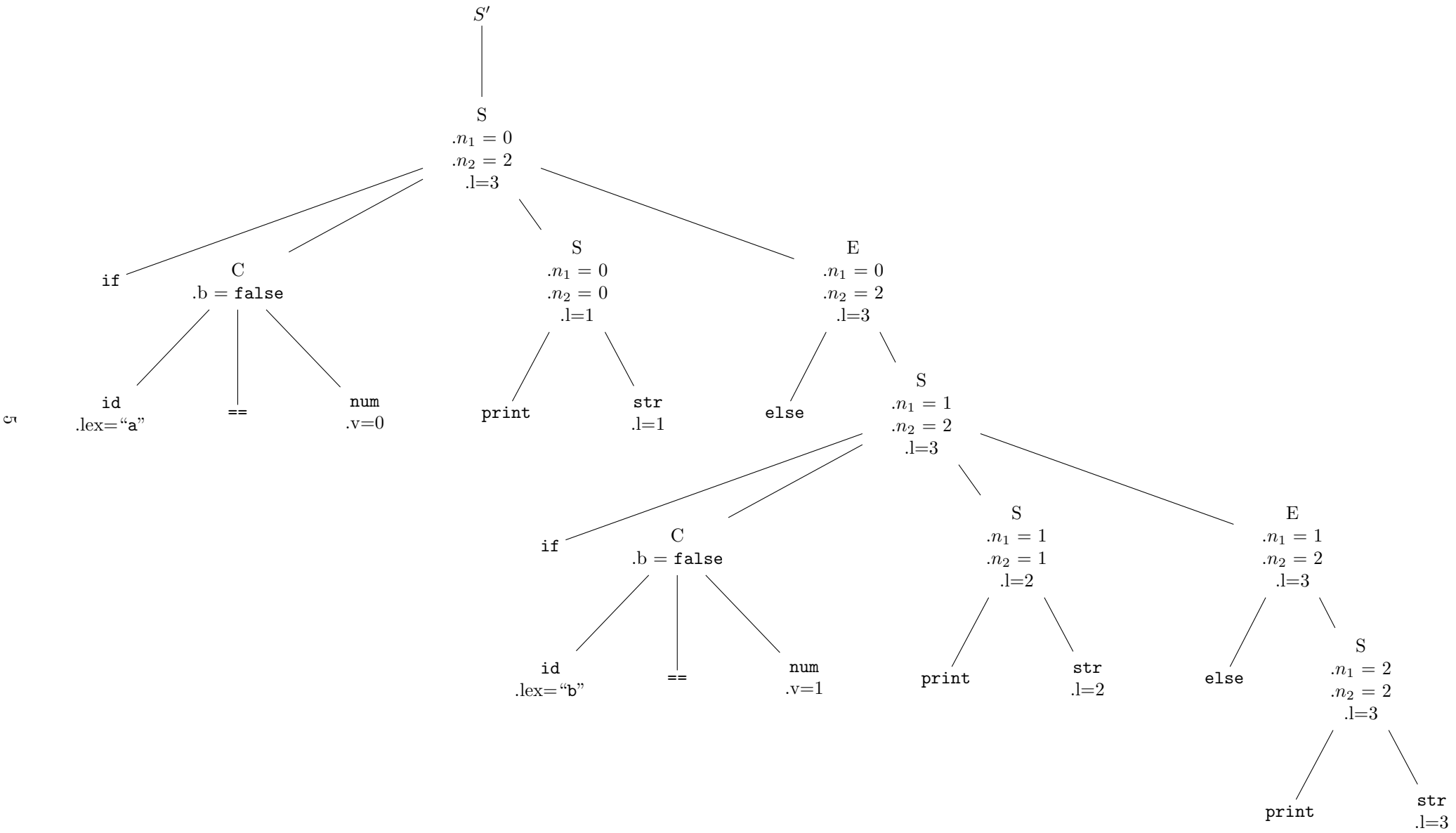
Tabla de cadenas

label	str
1	"a es 0"
2	"a no es 0; b es 1"
3	"a no es 0; b no es 1"

Para facilitar la comprensión de la DDS, se indica el nivel de anidamiento correspondiente a cada sentencia `print` de la entrada en el comentario asociado a cada línea:

```
if a==0 then print "a es 0"                          // Nivel 0
else if b==1 then print "a no es 0; be es 1"         // Nivel 1
     else print "a no es 0; b no es 1"               // Nivel 2
```

c) La DDS anterior define una gramática L-atribuida, porque el atributo n_1 es heredado del padre.


S'
S
$.n_1 = 0$
$.n_2 = 2$
.l=3
if
C
.b = false
id
.lex="a"
==
num
.v=0
S
$.n_1 = 0$
$.n_2 = 0$
.l=1
print
str
.l=1
E
$.n_1 = 0$
$.n_2 = 2$
.l=3
else
S
$.n_1 = 1$
$.n_2 = 2$
.l=3
if
C
.b = false
id
.lex="b"
==
num
.v=1
S
$.n_1 = 1$
$.n_2 = 1$
.l=2
print
str
.l=2
E
$.n_1 = 1$
$.n_2 = 2$
.l=3
else
S
$.n_1 = 2$
$.n_2 = 2$
.l=3
print
str
.l=3