Universidad de Valladolid
Departamento de Informática
Teoría de autómatas y lenguajes formales . 2º I.T.Informática. Gestión.

Examen de convocatoria ordinaria, 4 de junio de 2007

Apellidos, Nombre..... **Grupo:** .....

Firma:

---

1 **(12 p.)** Dado el reconocedor finito

	a	b
$\rightarrow p$	q	p
(q)	p	q

1. Demostraremos que el lenguaje aceptado es $L = \{w \in (a|b)^* \ / \ |w|_a \text{ es impar } \}$
   Se demostrará por inducción sobre la longitud de w, n, que

   $$f^*(p, w) = q \text{ si y solamente si } w \text{ tiene un número impar de aes}$$

   B *Base: Si $n = 0$, entonces $w = \epsilon$; $f^*(p, w) = p$ y w tiene 0 aes, así que el enunciado es cierto.*
   H *Suponemos que el enunciado es cierto para $n - 1$ con $n \geq 1$*
   P *Sea w de longitud n. Hay dos casos: $w = xa$ o $w = xb$ donde x es el prefijo de w de longitud $n - 1$.*

   - *$w = xa$: Si w tiene un número par de aes, entonces x tiene un número impar, por* [A] *. Luego $f^*(p, x) = q$ por* [H] *. Por lo tanto $f^*(p, w) = p$ por* [C, D] .
     Si w tiene un número impar de aes, entonces x tiene un número par, por [A] *. Luego $f^*(p, x) = p$ por* [H] *. Por lo tanto $f^*(p, w) = q$ por* [C, D]
     Así pues está probado el enunciado para este caso.
   - *$w = xb$*:
     Si w tiene un número par de aes, x también, por [A], luego $f^*(p, x) = p$ por [H]

     Por lo tanto, $f^*(p, w) = p$, por [C, D]

     Si w tiene un número impar de aes, x también, por [A]; luego $f^*(p, x) = q$ por [H].

     Por lo tanto, $f^*(p, w) = q$, por [C, D]

   Se pide completar los huecos de la demostración, justificando por qué cada paso es cierto. Los argumentos deben elegirse entre B (base de la inducción) , H (hipótesis de inducción) y :
   A hechos conocidos sobre enteros o cadenas (por ejemplo: "la suma de un par y un impar es impar", "si $x = yz$ el número de aes de x es la suma del número de aes de y más el de z" , etc.)
   D la definición de f^* (la extensión de la función de transición a $Q \times \Sigma_E^*$)
   C la propiedad $f^*(q, xy) = f^*(f^*(q, x), y))$

2. Renombrando los estados $1 = p$ y $2 = q$ ¿Cuáles son las expresiones regulares siguientes?

   $R_{11}^0 =$ $\epsilon \,|\, b$ $\qquad R_{22}^1 =$ $\epsilon \,|\, b \,|\, ab^*a$ $\qquad R_{12}^2 =$ $b^*a\,(b \,|\, ab^*a)^*$

3. Como la constante del lema de bombeo podría ser 2, toda cadena $z \in L$ de longitud mayor o igual que 2 admite una descomposición uvw con $|uv| \leq 2$ de forma que v es no nula y bombeable. Descríbase quién podría ser v en cada uno de los casos siguientes y justifíquese por qué $uv^iw \in L \ \ \forall i \geq 0$:

   - $z = bbz_1$ $\quad v = bb$; $\quad uv^iw = (bb)^iz_1$, que tiene las mismas aes que z_1, número impar
   - $z = abz_1$ $\quad v = b$; $\quad uv^iw = ab^iz_1$, que tiene el mismo número de aes que abz_1, impar
   - $z = aaz_1$ $\quad v = aa$; $\quad uv^iw = (aa)^iz_1$, que tiene 2^{i-1} aes más que z, impar

2 **(10 p.)** 1. Hallar un RF determinista equivalente (especificando el significado de cada estado) a

	a	b	ϵ
$\rightarrow 1$	3		2
2			5
3		2	
(4)		2	
5	4	5	

	a	b	significado
$\rightarrow A$	B	C	$1, 2, 5$
(B)	D	E	$3, 4$
C	F	C	5
D	D	D	
E	F	C	$2, 5$
(F)	D	E	4

2. Hallar un RF determinista mínimo equivalente al anterior o probar que ya era mínimo:

$$\begin{array}{lcl} \mathcal{P}_0 & = & \{\{A, C, D, E\}\ ,\ \{B, F\}\} \\ \mathcal{P}_1 & = & \{\{A, C, E\}\ ,\ \{D\}\ ,\ \{B, F\}\} \\ \mathcal{P}_2 & = & \{\{A, C, E\}\ ,\ \{D\}\ ,\ \{B, F\}\} \end{array}$$

	a	b
$\rightarrow A$	B	A
(B)	D	A
D	D	D

3 **(12 p.)** Sea G la gramática

$$\begin{array}{lcl} S & \rightarrow & aBB \mid CED \\ A & \rightarrow & BS \\ B & \rightarrow & A \mid b \mid \epsilon \end{array} \qquad \begin{array}{lcl} C & \rightarrow & bEa \mid BB \\ D & \rightarrow & AB \mid \epsilon \\ E & \rightarrow & Eb \mid DE \end{array}$$

1. Los símbolos terminables de G son: $B, D; S, C; A$
2. Los símbolos accesibles de G son: $S; B, C, D, E; A$ (todos)
3. Los símbolos útiles de G son: $S; B; A;$
4. Una gramática equivalente sin símbolos inútiles es por lo tanto:

$$\begin{array}{lcl} S & \rightarrow & aBB \\ A & \rightarrow & BS \\ B & \rightarrow & A \mid b \mid \epsilon \end{array}$$

5. Si al resultado anterior se le eliminan reglas épsilon se obtiene:

$$\begin{array}{lcl} S & \rightarrow & aBB \mid aB \mid a \\ A & \rightarrow & BS \mid S \\ B & \rightarrow & A \mid b \end{array}$$

6. Una gramática equivalente a G sin reglas épsilon, ni reglas simples ni símbolos inútiles es:

$$\begin{array}{lcl} S & \rightarrow & aBB \mid aB \mid a \\ B & \rightarrow & BS \mid aBB \mid aB \mid a \mid b \end{array}$$

(A y sus reglas $A \rightarrow BS \mid aBB \mid aB \mid a$ son ahora inútiles)

4 **(6 p.)** Para diseñar un AP cuyo lenguaje aceptado por vaciado de pila sea

$\{w \in (a|b)^* \ / \$ si x es prefijo de w entonces $|x|_a \leq |x|_b\}$

se comienza poniendo las transiciones (Z es el símbolo inicial de la pila)

$$(r1) \quad f(q_1, b, Z) = \{(q_1, BZ)\} \qquad (r2) \quad f(q_1, b, B) = \{(q_1, BB)\} \qquad (r3) \quad f(q_1, a, B) = \{(q_1, \epsilon)\}$$

1. Explica el significado de estas transiciones

   ($r1$) apila una b (con la pila vacía: al empezar o si ya no hay bes en la pila)

   ($r2$) apila bes sobre bes ya apiladas

   ($r3$) desapila una b con la lectura de una a

2. Completa la definición del AP
   ($r4$) $f(q_1, \epsilon, B) = \{(q_1, \epsilon)\}$ (desapilar las bes en cualquier momento)
   ($r5$) $f(q_1, \epsilon, Z) = \{(q_1, \epsilon)\}$ (desapilar el símbolo inicial)
3. Propón una gramática que genere el lenguaje
   Puede obtenerse directamente del autómata:

$$\begin{array}{lcl} Z & \rightarrow & bBZ \mid \epsilon \\ B & \rightarrow & bBB \mid a \mid \epsilon \end{array}$$

5 **(6 p.)** Se tiene un máquina de Turing M que genera un lenguaje L. Se sabe que L no es recursivo, y que las primeras cadenas generadas son (en ese orden) aa, a, aba, a, ab, $a^{47}b$, b^3ab. Se construye otra máquina de Turing (M'), usando ésta, de la siguiente manera:

```
var long : integer;
begin(* M' *)
     long := 2; writeln ('aa');
     repeat
            hacer que M genere la siguiente x
            if M está parada then exit (parar)
            if length(x) >long then
               begin writeln (x); long := length(x) end
     until (1=0)
end (* M' *)
```

Evidentemente, esta máquina muestra un subconjunto de cadenas de L.

¿Es este lenguaje recursivamente numerable?. (Si/No/Puede serlo o no ... justifíquese).

Si. M' es una máquina de Turing que lo genera.

¿Es este lenguaje recursivo?. (Si/No/Puede serlo o no ... justifíquese).

Si. M' es una máquina de Turing que lo genera de forma que las cadenas aparecen en orden creciente de longitud (estrictamente creciente, de hecho). Las primeras cadenas generadas serán : $aa, aba, a^{47}b$, la siguiente tendrá una longitud mayor que 48, etc. (En este lenguaje existe a lo sumo una cadena de cada longitud). Por lo tanto el algoritmo

```
var x, y : string;
begin(* MR(L') *)
     leer (x); y := 'aa';
     while length(y)<length(x) do
          begin
               que M' genere la siguiente (en y)
               if M' está parada then salir del while
          end
     if x=y then writeln ('SI)
     else writeln ('NO')
end (* MR(L') *)
```

se para siempre, y caracteriza el lenguaje .

6 **(6 p.)** Sean L_1, L_2 y L_3 tres lenguajes

Probar que $L_1(L_2 \cap L_3) \subset (L_1L_2) \cap (L_1L_3)$

Si $w \in L_1(L_2 \cap L_3)$ existen $x \in L_1$ e $y \in L_2 \cap L_3$ tales que $w = xy$.
Por lo tanto $y \in L_2$, de modo que $w \in L_1L_2$.
Análogamente, $y \in L_3$, de modo que $w \in L_1L_3$.
Así pues, $w \in (L_1L_2) \cap (L_1L_3)$

Probar que $L_1(L_2 \cap L_3) \neq (L_1L_2) \cap (L_1L_3)$

Basta encontrar un ejemplo en el que no coincidan. Por ejemplo
$L_1 = a \,|\, ab \qquad L_2 = b \qquad L_3 = \epsilon$
$L_2 \cap L_3 = \emptyset$, luego $L_1(L_2 \cap L_3) = \emptyset$
mientras que
$L_1L_2 = ab \,|\, abb \quad$ y $\quad L_1L_3 = a \,|\, ab$
dando $\quad L_1L_2 \cap L_1L_3 = ab$

7 **(8 p.)** Para la máquina de Turing sobre el alfabeto binario, estado inicial q_1, estado final q_2 y función de transición
$f(q_1, 0) = (q_3, 0, \rightarrow)$; $f(q_1, 1) = (q_2, 1, \leftarrow)$; $f(q_3, 1) = (q_1, 1, \leftarrow)$

el lenguaje de parada es $L_P =$ $\epsilon \mid 0 \mid 00(0|1)^* \mid 1(0|1)^*$

el lenguaje reconocido es $L_R =$ $1(0|1)^*$

una codificación de la máquina es $\overline{10101110101100}\,\overline{10110110110100}\,\overline{1110110101101}$

8 **(15 p.)** Calcular la TASP de la siguiente gramática, especificando los PRIMEROS y SIGUIENTES necesarios:

(1)	S	$\rightarrow$	aSd	(5)	B	$\rightarrow$	ϵ	(9)	F	$\rightarrow$	ABf
(2)		$\mid$	FGa	(6)		$\mid$	b	(10)		$\mid$	$gDfF$
(3)	A	$\rightarrow$	ϵ	(7)	D	$\rightarrow$	AB	(11)	G	$\rightarrow$	$AdBg$
(4)		$\mid$	a	(8)		$\mid$	dSA	(12)		$\mid$	fD

Pr.	S	A	B	D	F	G	a	...
	a	ϵ	ϵ	a	a	a	a	...
	b	a	b	b	b	d		
	f			ϵ	f	f		
	g			d	g			

Sg.	S	A	B	D	F	G
	\$	b	f	f	a	a
	d	f	g	a	d	
	a	d	a		f	
	f	a				

TASP	a	b	d	f	g	\$	
S	1, 2	2		2	2		
A	3, 4	3	3	3			
B	5	6		5	5		
D	7	7	8	7			
F	9	9		9	10		
G	11		11	12			

9 **(5 p.)** En el directorio por defecto se encuentran los siguientes ficheros: (todos de tipo texto)

1	aaa..	3	aaa..wq	5	ab	7	abc.txt	9	p.p	11	uno
2	aaabbb	4	aa.txt	6	abc.p.s	8	1.p	10	a.c	12	1a.pp

- La orden `ls -l | egrep -w "a*.*"` mostrará (*elíjase la opción correcta y complétese*):
  1. las líneas de `ls -l` correspondientes a los ficheros de número:

     1 a 12 (todos)
  2. las líneas de `ls -l` correspondientes a ficheros con determinado contenido:
  3. otra cosa:

- ¿Qué orden (similar a la anterior) mostraría lo mismo pero de los ficheros cuyo nombre sea *carácter punto carácter* (ej. 8, 9, 10)?

  `ls -l | egrep -w ".[.]."`

10 **(5 p.)** ¿Qué es `y.tab.h`?

Un fichero de cabecera para el programa `.c`, generado por *YACC*. También lo genera *YACC* (si se compila con la opción `-d`), a partir de las especificaciones del fuente (`.y`). Suele contener las definiciones de los enteros (constantes) asociados a los componentes léxicos (`%token`, que se convierten en `define`s), la declaración de la variable para los valores semánticos y el tipo de éstos (`%union`, que se convierte en definición de tipo). Se usa para comunicar estos datos entre el fichero generado por *LEX* y el generado por *YACC* (mediante un `#include "y.tab.h"`)

11 **(15 p.)** *(Se exige una calificación mínima de 7 puntos para considerar la nota de la práctica)*

Elaborar programas fuente en Lex y Yacc para realizar las siguiente tarea: se parte de un programa Pascal correcto (compilado sin errores).

1. partiendo de un programa Pascal correcto (compilado sin errores), devuelve el mismo programa del que se han eliminado los comentarios
2. partiendo de un programa correcto, que no tiene comentarios, ni definiciones de registros (`record` ... `end`), ni sentencias `case`, -pero que puede tener subprogramas–, debe mostrar solamente una línea por cada par `begin` ... `end` del *programa principal* consituída por los números de línea en los que se encuentran estas palabras. En la línea del par que enmarca el programa principal pondrá `PRINCIPAL`.
   `repeat` ... `until` debe considerarse como un par `begin` ... `end`.
   EJEMPLO:

```
(* 1*) program pp
(* 2*) procedure q
(* 3*) procedure r
(* 4*) begin      end
(* 5*) begin
(* 6*) begin      end
(* 7*) begin      end
(* 8*) end
(* 9*)
(*10*) begin                                    SALIDA:
(*11*) begin
(*12*) begin
(*13*) end                                      (12-13)
(*14*) begin                                    (14-15)
(*15*) end                                      (11-16)
(*16*) end                                      (19-19)
(*17*) begin                                    (18-20)
(*18*) repeat                                   (17-21)
(*19*) begin end                                (10-22) PRINCIPAL
(*20*) until ....
(*21*) end
(*22*) end
```

1. Hay varias soluciones para este problema en las páginas del laboratorio
2. **Componentes léxicos:**

BEG	begin, Repeat ... (inicios de pares)
END	eNd, until ... (fines de pares)
CABSP	procedure, Function ... (cabeceras de subprogramas)

   todos ellos en cualquier combinación de mayúsculas y minúsculas.

   Auxiliares de la gramática:

S	programa
B	bloque de subprogramas
P	subprograma
E	parte ejecutable del programa principal
R	parte ejecutable de subprograma

   Fuente Lex:

```
%{
#include "y.tab.h"
%}
%option case-insensitive
id      [a-zA-Z[a-zA-Z0-9]*
 int nl=1;
 extern int yylval;
%%
[ \t]+           ;
repeat           |
begin             {yylval=nl; return BEG;}
until            |
end               {yylval=nl; return END;}
function         |
procedure         {yylval=nl; return CABSP;}
{id}             ;
\n                {nl++;}
.                ;
```

   Fuente Yacc:

```
%{
#include <stdio.h>
yyerror( char * s)
        { fprintf (stderr, "%s\n", s); }
%}
%token BEG END CABSP
%%
S       : B BEG E END { printf ("END (%d-%d)\n", $2, $4); }
        ;
B       : B P
        |
        ;
P       : CABSP B BEG R END
        ;
E       : BEG E END E { printf ("end (%d-%d)\n", $1, $3); }
        |
        ;
R       : BEG R END R
        |
        ;
%%
main(){
        yyparse();
}
```