Universidad de Valladolid

Departamento de Informática

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

Examen de segunda convocatoria, 5 de septiembre de 2007

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

Firma:

---

1 **(14 p.)** La gramática

$$\begin{array}{lll} S & \rightarrow & AR \\ R & \rightarrow & aBR \mid bAR \mid \epsilon \end{array} \qquad \begin{array}{ll} A & \rightarrow a \mid bAA \\ B & \rightarrow b \mid aBB \end{array}$$

genera el lenguaje $L = \{w \in (a|b)^* \ / \ |w|_a = |w|_b + 1\}$.

1. Constrúyase un Autómata con Pila, con un solo estado, cuyo lenguaje aceptado por vaciado de pila sea L

   Se puede obtener una gramática en FNG equivalente y de ahí el AP:

$$\begin{array}{lll} S & \rightarrow & a \,|\, aR \,|\, bAA \,|\, bAA \,|\, bAAR \\ R & \rightarrow & aBR \,|aB \,|\,|\, bAR \,|\, bA \\ A & \rightarrow & a \,|\, bAA \\ B & \rightarrow & b \,|\, aBB \end{array}$$

q	a	b
S	$(q,R),(q,\epsilon)$	$(q,R),(q,AAR)$
R	$(q,BR),(q,B)$	$(q,AR),(q,A)$
A	(q,ϵ)	(q,AA)
B	(q,BB)	(q,ϵ)

2. Veremos que L no cumple el lema de bombeo de los regulares:

   sea N una pretendida constante para el lema. ¿Qué cadena elegirías para z? $a^{N+1}b^N$

   Si se descompusiera $z = uvw$ de forma que $|uv| \leq N$ y $|v| > 0$, ¿quiénes serían u, v y w?

   $u =$ a^r $v =$ a^p , $p > 0$ $w =$ $a^{N+1-p-r}b^N$

   ¿qué valor de i habría que elegir para que $uv^iw \notin L$ $i = 0$

   ¿Por qué razón $uv^iw \notin L$? $uv^0w = a^{N+1-p}b^N$ con $N+1-p \neq N+1$

   Suponiendo que el apartado 1 y el razonamiento anterior está completado ¿qué se puede concluir sobre el tipo de L en la jerarquía de Chomsky? L es independiente de contexto no regular

2 **(12 p.)** 1. Enunciar un algoritmo para eliminar la recursión **directa** por la derecha de una gramática i.c. sin reglas inútiles, ni simples, ni reglas "épsilon":

Sustituir

$$\begin{array}{ll} A \rightarrow & \alpha_1 A \mid \ldots \mid \alpha_p A \\ & \mid \beta_1 \mid \ldots \mid \beta_q \\ \text{donde} & \text{las } \beta_i \text{ no acaban por } A \end{array}$$

por

$$\begin{array}{ll} A \rightarrow & A'\beta_1 \mid \ldots \mid A'\beta_q \\ A' \rightarrow & A'\alpha_1 \mid \ldots \mid A'\alpha_p \mid \epsilon \\ & \\ \text{siendo} & A' \text{ un nuevo auxiliar} \end{array}$$

2. Eliminar la recursión por la derecha de la gramática

$$\begin{array}{llll} S & \rightarrow & aBB \mid a & \leftarrow \text{ rec. directa en } S \\ B & \rightarrow & SB \mid BS \mid b & \leftarrow \text{ rec. directa e indirecta} \end{array}$$

   Aplicando la misma técnica que para eliminar la recursión por la izquierda, y numerando S_1 y B_2:

$$\begin{array}{lll} S & \rightarrow & aBB \mid a \\ B & \rightarrow & B'Ba \mid B'b \\ B' & \rightarrow & B'S \mid B'BaB \mid \epsilon \end{array}$$

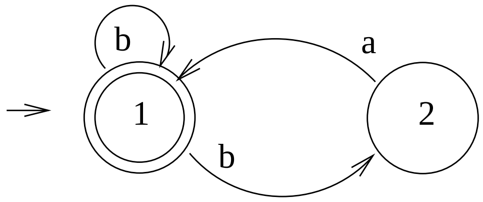
$\boxed{3}$ **(14 p.)** Considérense los tres lenguajes siguientes, sobre el alfabeto $\{a, b\}$:

- $L_1 = (b \mid ba)^*$
- $L_2 = (bb^*a)^*b^*$
- $L_3 = (bb^*a)^*a(a \mid b)^*$

Mostrar, justificadamente, la relación que hay entre ellos.

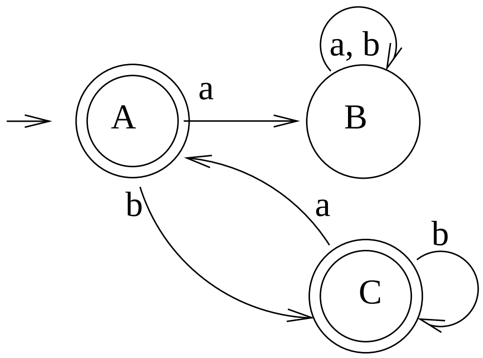
$$L_1 = L_2 = \overline{L_3}$$

Justificación : un RFN para L_1 es



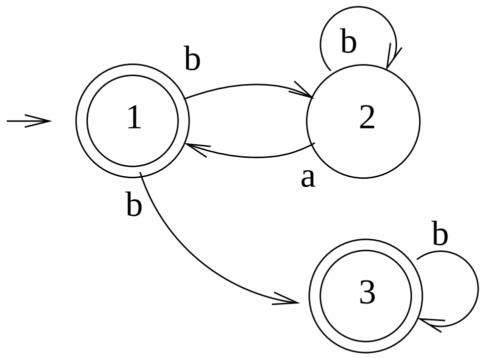
	a	b
(1)		1, 2
2	1	

de donde se obtiene el RFD mínimo:



	a	b	
(A)	B	C	1
B	B	B	
(C)	A	C	1, 2

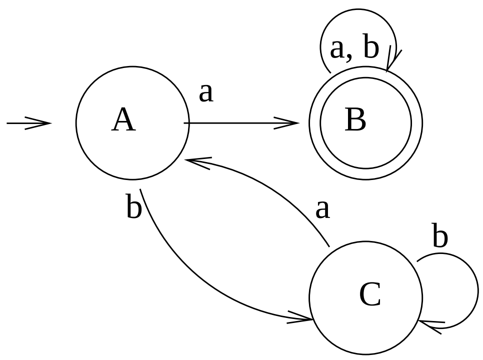
Un RFN para L_2 es



	a	b
(1)		2, 3
2	1	
(3)		3

de donde se obtiene el mismo RFD anterior. Por lo tanto, $L_1 = L_2$

Y un RFN (que resulta ser determinista) para L_3 es:



	a	b
A	B	C
(B)	B	B
C	A	C

que es el complementario del RFD para L_1 (y L_2).

$\boxed{4}$ **(6 p.)** Supongamos que un lenguaje verifica que $L^2 \subseteq L$. Demuestra que, entonces, para todo $n > 1$ se tiene que $L^n \subseteq L$.

Inducción sobre n: (B) (base de la inducción): $n = 2$: $L^2 \subseteq L$ (hipótesis)
(H) (hipótesis de inducción): Si $n >= 2$, entonces $L^n \subseteq L$
(P) (paso de inducción): $n+1$: $L^{n+1} = L^n L \subseteq LL$ (por H) $= L^2 \subseteq L$ (por B)

Un ejemplo en el que $L^2 \subseteq L$: $L = a^*$ En este caso $L^2 = a^* a^* = a^* = L \subseteq L$

Un ejemplo en el que $L^2 \not\subseteq L$: $L = a$ En este caso $L^2 = aa \not\subseteq L$

$\boxed{5}$ **(14 p.)** 1. Para la máquina de Turing sobre el alfabeto binario, estado inicial q_1, estados $\{q_1, q_2, q_3\}$, estado final q_2 y función de transición:

$f(q_1, 0) = (q_2, 0, \rightarrow)$; $f(q_1, 1) = (q_3, 1, \leftarrow)$; $f(q_3, \hbar) = (q_1, \hbar, \rightarrow)$

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

El lenguaje reconocido es $L_R =$ $0(0 \mid 1)^*$

Poniendo $X_1 = 0,\ X_2 = 1,\ X_3 = \hbar,\ D_1 = \leftarrow,\ D_2 = \rightarrow$

una codificación de la máquina es 10101101011 00 1011011101101 00 1110111010111011

2. Considérese el lenguaje formado por todas las codificaciones de máquinas de Turing cuyo lenguaje reconocido es no vacío:

$$L_{nv} = \{< M > \in (0|1)^* \,/\, L_R(M) \neq \emptyset\}$$

Supondremos que toda cadena de $(0|1)^*$ es una máquina de Turing: si no corresponde a una codificación, se entiende que se trata de una (la) máquina sin transiciones. Por ejemplo, 10101101011 y la codificación de la máquina del apartado anterior están en L_{nv}, pero ni 1010101011 ni 10 están en L_{nv}.

Describir un algoritmo reconocedor que justifique que L_{nv} es recursivamente numerable (se dispone de la Máquina de Turing Universal).

```
leer <M>
x := cadena vacía
repetir
     aplicar M a x (MTU sobre el par <M> <x>)
     si M ha aceptado (se ha parado dejando 11 en la cinta de estados)
               escribir SI (M ha reconocido algo) y PARAR
     x := siguiente (x) (en el orden natural)
hasta que 1=0
```

3. Dada un máquina de Turing M y una cadena w de entrada para M, se construye otra máquina de Turing M' con la siguiente "lógica":

```
var x: string
begin
    leer x (en la cinta de entrada)
    borrar x
    simular la operación de M sobre w
           (haciendo uso de la M.T. Universal)
    escribir SI (si M se ha parado sobre w)
end
```

¿Cuál es el lenguaje reconocido por M'?

Depende.

Si M se para sobre w entonces $L_R(M') = \Sigma^*$, el lenguaje universal;
es decir, M' reconoce a cualquier cadena.

Si M no se para sobre w entonces $L_R(M') = \emptyset$, el lenguaje vacío;
es decir, M' no reconoce a ninguna cadena.

6 **(15 p.)** Calcular la TASP de la siguiente gramática, especificando los PRIMEROS y SIGUIENTES necesarios:

(1)	S	$\rightarrow$	aSc	(5)	B	$\rightarrow$	ϵ	(9)	D	$\rightarrow$	ABd
(2)		$\mid$	DEb	(6)		$\mid$	b	(10)		$\mid$	$eCdD$
(3)	A	$\rightarrow$	ϵ	(7)	C	$\rightarrow$	BA	(11)	E	$\rightarrow$	$AcBe$
(4)		$\mid$	a	(8)		$\mid$	cSB	(12)		$\mid$	dC

Pr.	S	A	B	C	D	E	a	...
	a	ϵ	ϵ	b	a	a	a	
	b	a	b	a	b	c		
	d			ϵ	d	d		
	e			c	e			

Sg.	S	A	B	C	D	E
	$\$$	b	a	d	a	b
	c	d	d	b	c	
	b	c	e		d	
	d		b			

TASP	a	b	c	d	e	$	
S	1,2	2		2	2		
A	4	3	3	3			
B	5	5, 6		5	5		
C	7	7	8	7			
D	9	9		9	10		
E	11		11	12			

7 **(5 p.)** Escribir órdenes grep que calculen cuántas líneas de la entrada:

1. contienen un punto `grep -c [.]` *fichero*
2. contienen dos puntos consecutivos `grep -c [.][.]` *fichero*
3. contienen exactamente dos puntos no necesariamente consecutivos `grep -c  ^[^.]*[.][^.]*[.][^.]*$` *fichero*

(Por ejemplo, para la entrada siguiente las salidas deben ser, respectivamente, 7, 4 y 3)

```
-rw-r--r-- 1 pepito users   0 may 30 19:34 aaa...       1  2  -
-rw-r--r-- 1 pepito users   0 may 30 19:35 aaa..        1  2  3
-rw-r--r-- 1 pepito users   0 may 30 19:36 aaabbb       -  -  -
-rw-r--r-- 1 pepito users   0 may 30 19:37 aaa..wq      1  2  3
-rw-r--r-- 1 pepito users   0 may 30 19:38 aa.txt       1  -  -
-rw-r--r-- 1 pepito users   0 may 30 19:39 ab..t.a      1  2  -
-rw-r--r-- 1 pepito users   0 may 30 19:40 abc.p.s      1  -  3
-rw-r--r-- 1 pepito users   0 may 30 19:41 ab.txt       1  -  -
                                              total:    7  4  3
```

8 **(5 p.)** ¿Qué es `y.output`?

Un archivo generado por Yacc con la opción `-v` que contiene las tablas de análisis sintáctico por desplazamiento-reducción que se usarán en el analizador sintáctico (`y.tab.c`).

9 **(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 siguientes tareas (un programa o una combinación de fuente Lex y fuente Yacc para cada apartado):

1. partiendo de un programa Pascal correcto (compilable sin errores), devuelve solamente los comentarios
2. partiendo de un programa correcto, que no tiene comentarios, ni definiciones de registros (`record` ... `end`), ni sentencias `case`, debe mostrar solamente una línea por cada subprograma, constituída por el número de línea en que comienza la cabecera del mismo y los números de línea en que se encuentran las palabras reservadas BEGIN y END que lo enmarcan, más una línea con la misma información referida al programa principal.

EJEMPLO:

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

```
SALIDA:


3: (4-4) FUNCION
2: (5-8) PROCEDIMIENTO
1: (10-22) PRINCIPAL
```

1.

2. **Componentes léxicos:**

BEG	begin, Repeat ... (inicios de pares)
END	eNd, until ... (fines de pares)
PROGRAM, PROCEDURE; FUNCTION	program, procedure, Function (respectivamente)

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 y de los subprogramas)

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;}
program         {yylval=nl; return PROGRAM;}
procedure       {yylval=nl; return PROCEDURE;}
function        {yylval=nl; return FUNCTION;}
{id}            ;
\n              {nl++;}
.               ;
```

Fuente Yacc:

```
%{
#include <stdio.h>
yyerror( char * s)
        { fprintf (stderr, "%s\n", s); }
%}
%token BEG END PROCEDURE FUNCTION PROGRAM
%%
S     : PROGRAM B BEG E END
          { printf ("%d: (%d-%d) PRINCIPAL\n", $1, $3, $5); }
      ;
B     : B P
      |
      ;
P     : PROCEDURE B BEG E END
          {printf ("%d: (%d-%d) PROCEDIMIENTO \n", $1, $3, $5);}
      | FUNCTION  B BEG E END
          {printf ("%d: (%d-%d) FUNCION \n", $1, $3, $5);}
      ;
E     : BEG E END E
      |
      ;

%%
main(){
     yyparse();
}
```