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

Examen de convocatoria ordinaria, 17 de junio de 2008

Solución del examen. Parte teórica

1* **(1 p.)** Se sabe de un reconocedor finito determinista completo que reconoce las cadenas a, ab, bb y aaa y que no reconoce a las cadenas ϵ, b, aa y ba. Conteste justificadamente:

1. ¿cuántos reconocedores de 2 estados pueden hacer esta tarea? , y si es el caso ¿cuáles?
   Si el reconocedor finito tiene dos estados y no es mínimo, será equivalente al que tiene por lenguaje reconocido el vacío o al que tiene por lenguaje reconocido el total; ninguno de los dos es el caso.
   Supongamos pues que es mínimo. Entonces uno de los estados es final y el otro no.
   Como $\epsilon \notin L(R)$, $f(q_1, \epsilon) = q_1$ no es final.
   Como $a \in L(R)$, $f(q_1, a) \in F$, luego es q_2.
   Como $b \notin L(R)$, $f(q_1, b) \notin F$, luego es q_1.
   Como $aa \notin L(R)$, $f(q_2, a) = f(q_1, aa) \notin F$, luego es q_1
   Como $ab \in L(R)$, $f(q_2, b) = f(q_1, ab) \in F$, luego es q_2
   Por lo tanto el reconocedor tiene que ser

	a	b
$\rightarrow 1$	2	1
(2)	1	2

   Pero este no acepta bb, y debería hacerlo; por lo tanto no hay ningún reconocedor de dos estados que cumpla las condiciones pedidas.
2. ¿cuántos reconocedores de 3 estados pueden hacer esta tarea? , y si es el caso ¿cuáles?
   Con 3 estados, podemos suponer que es mínimo, porque de no serlo sería equivalente a otro con menos estados, y ya se ha visto que no es posible.
   Siendo mínimo, los estados se distinguirán en la partición de orden 1 (dos estados serán equivalentes si y sólo si lo son de orden 1), y por lo tanto, cada estado se caracteriza por su respuesta ante las cadenas ϵ, a, b.
   Del comportamiento desde q_1 se sabe:

	ϵ	a	b	aa	ab	ba	bb	aaa	...
C_{q_1}	**0**	**1**	**0**	0	1	0	1	1	...

   Como q_1 no es final, pero $f(q_1, a)$ sí lo es, éste no es q_1. Llamémoslo q_2.
   Ya se tiene

	a	b
$\rightarrow 1$	2	
(2)		

   Como $g(q_2, x) = g(q_1, ax)$, lo que se sabe del comportamiento del reconocedor desde q_2 es

	ϵ	a	b	aa	ab	ba	bb	aaa	...
C_{q_2}	**1**	**0**	**1**	1					...

   Para saber si $f(q_1, b)$ es q_1, q_2 o el tercer estado consideremos el comportamiento del reconocedor desde $f(q_1, b)$: como $g(f(q_1, b), x) = g(q_1, bx)$, se sabe del comportamiento del reconocedor desde $f(q_1, b)$ lo siguiente:

	ϵ	a	b	aa	ab	ba	bb	aaa	...
$C_{f(q_1,b)}$	**0**	**0**	**1**						...

   y por lo tanto, $f(q_1, b)$ no es equivalente ni a q_1 ni a q_2: es q_3. Además no es final.
   Ya se tiene

	a	b
$\rightarrow 1$	2	3
(2)		
3		

   y los tres estados se caracterizan por los comportamientos respectivos para las cadenas de longitud menor que 2: $(0, 1, 0)$, $(1, 0, 1)$ y $(0, 0, 1)$ *respectivamente.*

Para conocer las transiciones que faltan, considérese el comportamiento del reconocedor desde $f(q_2, a)$ y desde $f(q_2, b)$: usando que $g(f(q_2, a), x) = g(q_2, ax)$ y $g(f(q_2, b), x) = g(q_2, bx)$ se deduce:

	ϵ	a	b	aa	ab	ba	bb	aaa	...
$C_{f(q_2,a)}$	**0**	**1**							...
$C_{f(q_2,b)}$	**1**								...

Por lo tanto, $f(q_2, a) = q_1$ y $f(q_2, b) = q_2$.
Y para q_3 se obtiene:

	ϵ	a	b	aa	ab	ba	bb	aaa	...
$C_{f(q_3,a)}$	**0**								...
$C_{f(q_3,b)}$	**1**								...

Por lo tanto $f(q_3, b) = q_2$ y de $f(q_3, b)$ sólo podemos saber que no es q_2.
Por lo tanto hay dos posibilidades:

	a	b
$\to 1$	2	3
(2)	1	2
3	1	2

	a	b
$\to 1$	2	3
(2)	1	2
3	3	2

Ambos autómatas son mínimos: en ambos casos

$$\begin{aligned} \mathcal{P}_0 &= \{\, \{1,3\}, \{2\} \,\} \\ \mathcal{P}_1 &= \{\, \{1\}, \{3\}, \{2\} \,\} \end{aligned}$$

Ambos cumplen las condiciones pedidas, y, dada la forma de construcción, son los únicos que las cumplen.

2 **(1 p.)** Considere el alfabeto $\Sigma = \{a, b, c\}$. Sea L un lenguaje cualquiera sobre Σ tal que $L \subseteq (a \,|\, b)^*$, y sea $L_c = c^+ L \cup (a \,|\, b)^*$

(NOTA: consecuencia interesante de este problema es darse cuenta de que existen lenguajes que cumplen el lema de bombeo sin tener nada de regulares).

1. Probar que L_c cumple el lema de bombeo de los lenguajes regulares, incluso en su "versión fuerte".
   Cumple el *lbr* para $N = 2$.
   En primer lugar, obsérvese que si $z \in L_c$, o bien está formada sólo por aes y bes ($z \in (a|b)^*$) o bien consiste en un prefijo formado por algunas ces (una al menos) y a continuación una cadena de L, formada sólo por aes y bes ($z \in c^+L \subseteq c^+(a|b)^*$).
   Sea z una cadena de L_c con longitud $|z| \geq 2$. Hay pues dos casos:
   - $z \in c^+L$. Entonces $z = c^p z_1$ con $p > 0$ y $z_1 \in L$.
     - Si $p > 1$, se puede bombear la primera c, es decir, tomar $u = \epsilon$, $v = c$ y $w = c^{p-1}z_1$, en cuyo caso $z = uvw$, y $uv^iw = c^{p-1+i}z_1$ que está en c^+L, y por lo tanto en L_c, para todo $i \geq 0$ ($p - 1 + i > 0$), cumpliéndose además que $|uv| = 1 \leq N$
     - Si $p = 1$ ($z = cz_1$), se puede bombear la primera c igualmente, es decir, tomar $u = \epsilon$, $v = c$ y $w = c^{p-1}$; en este caso, uv^iw está en c^+L si $i > 0$, y si $i = 0$, $uv^iw = z_1 \in (a|b)^*$ que también está en L_c
   - $z \in (a|b)^*$. Entonces, se puede tomar $u = \epsilon$, v el primer carácter de z y w el resto; uv^iw siempre estará en $(a|b)^*$ y por lo tanto en L_c
2. Probar que L_c es recursivo si y solamente si L es recursivo.
   - Si L es recursivo, como c^+ y $(a|b)^*$ también lo son, y la concatenación de recursivos lo es, y la unión de recursivos también, L_c es recursivo.
   - Recíprocamente: si L_c es recursivo y Mc es una máquina caracterizadora de L_c, se puede construir una máquina caracterizadora de L mediante el siguiente "algoritmo":

```
leer x (sobre el alfabeto {a, b, c})
si x contiene alguna c, escribir NO
      (mediante un RFD, por ejemplo)
si no
      poner una c delante de x
      aplicar Mc a cx (se parará)
      si MC ha dicho SI, escribir SI
      si no, escribir NO
```

   Este algoritmo (la máquina de Turing construída con él) se para siempre, y escribe SI exactamente para las cadenas de L, ya que $cx \in L_c$ si y solamente si $x \in L$, y NO para las restantes.

3. Probar que L_c es recursivamente numerable si y solamente si L es recursivamente numerable.

   - Si L es recursivamente numerable, existirá una MT generadora de L, que llamaremos Mr. A partir de ella podemos construir una máquina generadora de c^+L mediante la siguiente técnica:
     Generar los pares (i,j) de $\mathbb{N}\times\mathbb{N}$ en el orden siguiente: $(1,1),(1,2),(2,1),(1,3),(2,2),(3,1),(1,4),(2,3)\ldots$, es decir, primero los pares que sumen 2, luego los que sumen 3, los que sumen 4, etc. y, dentro de éstos, en orden creciente de la primera componente (como se hizo en la transparencia 21 del tema de Máquinas de Turing. Cualquier otra numeración de $\mathbb{N}\times\mathbb{N}$ sería también válida).
     Para cada par (i,j), hacer que Mr genere su í-sima cadena, añadirle c^j como prefijo y escribir el resultado.
     De esta manera se generan exactamente las cadenas de c^+L, dado que lo que se genera evidentemente está en c^+L y cualquier cadena z de c^+L será la concatenación de un $c^k, k>0$ y una cadena de z' de L, que saldrá en la generación que hace Mr en alguna posición p (al menos): $(z'=z_p)$. Cuando aparezca en la generación de pares el par (p,k), se mostrará z.
     Por lo tanto c^+L es recursivamente numerable. Por otra parte $(a|b)^*$ es recursivamente numerable, y como la unión de recursivamente numerables lo es, L_c será recursivamente numerable.
   - Recíprocamente, si L_c es recursivamente numerable y Mrc es una máquina generadora de L_c, podemos hacer una máquina generadora de L sin más que poner a trabajar a Mrc, y, capturar la salida; cada vez que Mrc escriba una cadena z, hacer lo siguiente:
     si z no empieza por c, no se escribe;
     si z empieza por c, se omiten todas las ces y se escribe el resto de los caracteres.
     Este algoritmo escribirá exactamente las cadenas de L, y por lo tanto L es recursivamente numerable.

   (NOTA: puede también probarse este apartado trabajando con máquinas reconocedoras, de las que no necesariamente se paran siempre).

3 **(1'5 p.)** Sean $L_1=\{a^ib^j \ / \ 1\le i\le j\}$ y $L_2=\{a^ib^j \ / \ 1\le j\le 2i\}$

1. Hallar un Autómata con Pila Determinista cuyo lenguaje reconocido sea L_1 (especifique si el lenguaje es el reconocido por estado final o por vaciado de pila, y explique brevemente el autómata).

   Se propone el siguiente (naturalmente no es la única posibilidad), que se basa en la siguiente idea:

   - Se reconocerá por estado final: se usará la pila para controlar que el número de bes sea igual al de aes, y entonces se pasará a un estado final que permita admitir las bes restantes. ($L_1=\{\,a^ib^i \ / \ i\ge 1\,\}\,b^*$)
   - Apilar la primera a como A, para distinguir la primera (obligada) de las otras (optativas) [estado 1]
   - Apilar las aes que sigan [primera columna del estado 2]. La aparición de una b despila una a y se pasa a un estado de desapilado (3); si la b desapila la primera a, ya hay tantas aes como bes, y se pasa a un estado de aceptación (4) [segunda columna del estado 2].
   - En el estado 3 se desapilan aes con bes; igualmente el desapilado de la primera a lleva al estado de aceptación.
   - Una vez en el estado 4, ya hay tantas aes como bes, y hay que admitir cualquier cantidad de bes posteriores.
   - El símbolo inicial de la pila es S. R es otro símbolo de la pila.

$\to 1$	a
S	$(2,AR)$

2	a	b
A	$(2,aA)$	$(4,\epsilon)$
a	$(2,aa)$	$(3,\epsilon)$

3	b
A	$(4,\epsilon)$
a	$(3,\epsilon)$

(4)	b
R	$(4,R)$

2. Probar que L_2 es independiente de contexto.

   Puede probarse describiendo un autómata con pila, hallando una gramática que lo genere, ... Hay muchas posibilidades en cualquier caso. Exponemos una de ellas:

   Las reglas

   $$S \ \to \ aSB \mid \epsilon$$

   generarían las fomas sentenciales a^iB^i para $i\ge 0$. Si se sustituye B por bb se obtiene a^ib^{2i}, es decir, a^ib^j con $j=2i$. Para obtener también todas las cadenas en las que $j<2i$ bastaría permitir que B sea sustituído por b o eliminado (sustituído por ϵ). Por lo tanto, la gramática

   $$\begin{array}{lcl} S & \to & aSB \mid \epsilon \\ B & \to & bb \mid b \mid \epsilon \end{array}$$

   generará $\{a^ib^j \ / \ j\le 2i\}$, lo que hace a este lenguaje independiente de contexto. Para conseguir L_2 basta intersecar este lenguaje con a^+b^*. Por lo tanto, L_2 es independiente de contexto (intersección de un independiente de contexto con un regular).

3. Justificar que el lenguaje $L_1 \cap L_2$ es independiente de contexto.
   $L_1 \cap L_2 = \{a^i b^j \ / \ 1 \leq i \leq j \leq 2i\}$
   Por lo tanto el número de bes es cualquiera entre el número de aes y su doble: por cada a hay una o dos bes. Es fácil ver que la gramática

$$\begin{array}{lcl} S & \rightarrow & aSB \mid \epsilon \\ B & \rightarrow & b \mid bb \end{array}$$

   genera $L_3 = \{a^i b^j \ / \ 0 \leq i \leq j \leq 2i\}$ (porque a base de aplicar las dos primeras reglas se obtienen las formas sentenciales $a^n B^n \ / \ n \geq 0$ y luego cada B puede ser sustituída por 1 ó 2 bes, por lo tanto como poco se obtienen n bes, como mucho $2n$, y puede obtenerse cualquier número intermedio).
   Por lo tanto este lenguaje (L_3) es independiente de contexto. Ahora sólo hay que darse cuenta de que

$$L_1 \cap L_2 = L_3 \cap a^+ b^*$$

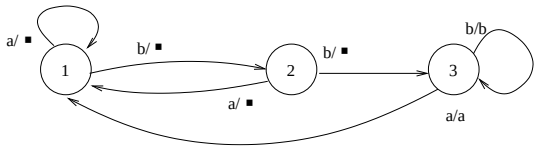
   y se obtiene $L_1 \cap L_2$ como intersección de un independiente de contexto con un regular, que es independiente de contexto.

(NOTA: este es un ejemplo no trivial de intersección de lenguajes independientes de contexto que lo es; como sabemos esto no siempre ocurre).

4 **(1 p.)** Dada máquina de Mealy

	a	b
1	$1/\bullet$	$2/\bullet$
2	$1/\bullet$	$3/\bullet$
3	$1/a$	$3/b$

1. Calcular la salida para la entrada $w = abbababbabbbabbbb$ si se comienza en el estado 1
   El diagram de estados es el siguiente:



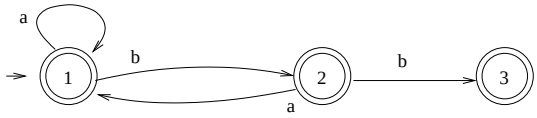
   La salida es

$$\bullet \bullet \bullet a \bullet \bullet \bullet \bullet a \bullet \bullet ba \bullet \bullet bb$$

   (mientras se recorren los estados 112312123123312333 . A continuación se muestra una simulación: en la línea central la entrada, en la inferior el estado y en la superior la salida)

$$_1 a_1^\bullet b_2^\bullet b_3^\bullet a_1^a b_2^\bullet a_1^\bullet b_2^\bullet b_3^\bullet a_1^a b_2^\bullet b_3^\bullet b_3^b a_1^a b_2^\bullet b_3^\bullet b_3^b b_3^b$$

2. Describir de la mejor manera posible el conjunto de cadenas de $(a|b)^*$ para las que esta máquina da como salida una cadena de $\bullet^*$ (es decir, sin aes ni bes).
   El conjunto de cadenas pedido puede describirse quitando de la máquina dada los arcos que producen salida distinta de $\bullet$:
   Se trata pues de las cadenas reconocidas por el siguiente RF:



   El algoritmo de análisis dará como resultado

$$(a \mid ba)^* \, (\epsilon \mid b \mid bb)$$

   (Puesto que se trata de un lenguaje regular, la expresión regular es la "mejor" manera de describirlo)
3. Obtener una máquina de Moore equivalente

	a	b
$1^\bullet/\bullet$	$1^\bullet$	2
$1^a/a$	$1^\bullet$	2
$2/\bullet$	$1^\bullet$	$3^\bullet$
$3^\bullet/\bullet$	1^a	3^b
$3^b/b$	1^a	3^b

5 **(0'5 p.)** Obténgase una gramática sin recursión por la izquierda equivalente a la siguiente:

$$\begin{array}{lcl} A & \to & Ab \mid CCa \\ B & \to & BA \mid CBa \\ C & \to & AAb \mid Ca \mid DB \mid a \\ D & \to & AD \mid a \end{array}$$

B no es terminable; A, C y D lo son. Eliminando no terminables se obtiene:

$$\begin{array}{lcl} A & \to & Ab \mid CCa \\ C & \to & AAb \mid Ca \mid a \\ D & \to & AD \mid a \end{array}$$

D es inaccesible; A, y C son accesibles. Eliminando no accesibles se obtiene:

$$\begin{array}{lcl} A & \to & Ab \mid CCa \\ C & \to & AAb \mid Ca \mid a \end{array}$$

El algoritmo de eliminación de la recursión por la izquierda, numerando A_1 y C_2 transforma la gramática sucesivamente en

$$\left\{\begin{array}{lcl} A & \to & CCaA' \\ A' & \to & bA' \mid \epsilon \\ C & \to & AAb \mid Ca \mid a \end{array}\right.$$

$$\left\{\begin{array}{lcl} A & \to & CCaA' \\ A' & \to & bA' \mid \epsilon \\ C & \to & CCaA'Ab \mid Ca \mid a \end{array}\right.$$

$$\left\{\begin{array}{lcl} A & \to & CCaA' \\ A' & \to & bA' \mid \epsilon \\ C & \to & aC' \\ C' & \to & CaA'AbC' \mid aC' \mid \epsilon \end{array}\right.$$

La numeración C'_{-2}, A'_{-1}, A_1, C_2 justifica que la gramática obtenida no tiene recursión por la izquierda.

La elección de la ordenación C_1 y A_2, por un procedimiento similar, daría la gramática:

$$\left\{\begin{array}{lcl} A & \to & aC'CaA' \\ A' & \to & bA' \mid AbC'CaA' \mid \epsilon \\ C & \to & AAbC' \mid aC' \\ C' & \to & aC' \mid \epsilon \end{array}\right.$$

Contestar en esta misma página:

6 **(1 p.)** 1. Hallar un RF determinista equivalente (especificando el significado de cada estado) a

	a	b	c	ϵ
$\rightarrow 1$		2	3	$2,3$
2	1	3	$1,2$	
(3)				

respuesta:

	a	b	c	
$\rightarrow (A)$	A	B	A	$1,2,3$
(B)	A	C	A	$2,3$
(C)	D	D	D	3
D	D	D	D	

2. Hallar un RF determinista mínimo equivalente al anterior o probar que ya era mínimo:
   El obtenido es mínimo. Esto se prueba, o bien dando cadenas que distingan cada par de estados:

B	ba		
C	a	a	
D	ϵ	ϵ	ϵ
	A	B	C

   o bien aplicando el algoritmo de minimización, que proporciona el mismo reconocedor:

$$\begin{aligned}\mathcal{P}_0 &= \{\,\{A,B,C\},\,\{D\}\,\}\\ \mathcal{P}_1 &= \{\,\{A,B\},\,\{C\},\,\{D\}\,\}\\ \mathcal{P}_1 &= \{\,\{A\},\,\{B\},\,\{C\},\,\{D\}\,\}\end{aligned}$$

7 **(0'5 p.)** Describe brevemente (en esta página) cuáles son los resultados más importantes explicados en la parte teórica de esta asignatura.

Se esperaba, básicamente, que se explicara:

- la clasificación de lenguajes en cuanto a su dificultad de reconocimiento y generación (en los tipos de la jerarquía de Chomsky, 0, 1, 2 y 3, recursivos y recursivamente numerables)
- la relación de cada uno de estos tipos con los algorimos generadores (gramáticas) y reconocedores (máquinas o autómatas)
- la existencia de lenguajes no recursivamente numerables, es decir, imposibles de generar por ningún mecanismo algorítmico (o sea, no generables por máquinas de Turing)
- la existencia de lenguajes recursivamente numerables no recursivos, es decir, generables por algún mecanismo algorítmico (o sea, por máquinas de Turing), pero que no pueden caracterizarse por máquinas de Turing (es decir, por ningún programa)

lo que se puede esquematizar:

lenguajes	**máquinas reconocedoras**	**dispositivos generadores**
generales		
tipo 0 o recursivamente numerables	máquinas de Turing (reconocedoras)	gramáticas tipo 0 o m. de Turing (generadoras)
recursivos	m. Turing reconocedoras que siempre se paran (caracterizadoras)	
tipo 1	autómatas linealmente acotados	gramáticas tipo 1 o gramáticas no contractivas
tipo 2	autómatas con pila no deterministas	gramáticas independientes del contexto
tipo 3	autómatas finitos deterministas o indeterministas	gramáticas regulares por la izquierda o gramáticas regulares por la derecha

estando cada tipo estrictamente contenido en el anterior.