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

Soluciones al examen de primera convocatoria. 16 de junio de 2006

---

1 **(2 p.)** (Por abreviar, no se hace el diagrama de transiciones)

	a	b	
$\rightarrow 1$	2	1	no hay nada leído, o lo último fué b
2	3	1	el último símbolo era una a
(3)	5	4	subcadena aa leída
(4)	3	4	después de la subcadena, último símbolo leído: b
5	5	5	leída una a después de leída aa

Mínimo: se puede aplicar el algoritmo de minimización o hallar cadenas que distingan cada par de estados:

2	a			
3	ϵ	ϵ		
4	ϵ	ϵ	a	
5	aa	a	ϵ	ϵ
	1	2	3	4

Una constante válida para el lema de bombeo es 5 (número de estados del reconocedor). Como uno de los estados es sumidero, también es válida la constante 4 (para las cadenas $bbaa$, $baab$, $aabb$ puede bombearse b; para la cadena $abaa$ puede bombearse ab, y para $aaba$ puede bombearse ba; éstas son todas las cadenas del lenguaje de longitud 4).

Ahora bien, las cadenas del lenguaje de longitud 3 son baa y aab, y en ambos casos puede bombearse b, luego la constante puede rebajarse a 3.

Más no puede rebajarse, porque aa no admite ninguna subcadena bombeable sin salirse del lenguaje.

2 **(2 p.)** 1. Es claro que el símbolo A de ambas gramáticas genera exactamente a^*, y que el símolo B de la primera genera b^*. Por lo tanto, la primera gramática, dada la regla $S \rightarrow ABA$, genera $a^*b^*a^*$

Para la segunda, también es claro que el símbolo B genera única y exclusivamente cadenas de $a^*b\gamma$, donde γ debe ser generado por C, y que C genera $b^*\alpha$, donde α debe ser generado por A. Por lo tanto, de C se obtiene b^*a^* y de B, $a^*bb^*a^*$. Finalmente, como S genera la unión de la parte generada por A y por B, se tiene que

$L(G_2) = a^*|a^*b^+a^* = a^*a^*|a^*b^+a^* = a^*(\epsilon|b^+)a^* = a^*b^*a^*$

2.

primeros	S	A	B
	a	a	b
	b	ϵ	ϵ
	ϵ		

siguientes	S	A	B
	\$	b	a
		a	\$
		\$	

TASP	a	b	\$
S	ABA	ABA	ABA
A	aA/ϵ	ϵ	ϵ
B	ϵ	bB	ϵ

primeros	S	A	B	C
	a	a	a	b
	b	ϵ	b	a
	ϵ			ϵ

siguientes	S	A	B	C
	\$	\$	\$	\$

TASP	a	b	\$
S	A/B	B	A
A	aA		ϵ
B	aB	bC	
C	A	bC	A

Ninguna es $LL(1)$, como se ve.

La primera es ambigua, como muestran por ejemplo las dos derivaciones más a la izquierda siguientes

$$S \Rightarrow ABA \Rightarrow BA \Rightarrow A \Rightarrow aA \Rightarrow a$$

$$S \Rightarrow ABA \Rightarrow aABA \Rightarrow aBA \Rightarrow aA \Rightarrow a$$

La segunda no es ambigua: se ve que las generaciones desde los símbolos A, B y C son no ambiguas (si se prescinde del símbolo S, el resto es $LL(1)$), y las reglas $S \to A \mid B$ muestran una unión disjunta: si la cadena tiene una b, obligatoriamente debe proceder de B, y si no la contiene, de A.

3. Puesto que el lenguaje es regular, la máquina más adecuada es un reconocedor finito determinista mínimo, que puede obtenerse muy fácilmente siguiendo la pista que sugiere la segunda gramática: $a^*b^*a^* = a^*|a^*b^+a^* = a^*|a^*b^+|a^*b^+a^+$
   o determinando el autómata

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

   De cualquier forma, se obtiene el RFD mínimo siguiente:

	a	b
$\to (1)$	1	2
(2)	3	2
(3)	3	4
4	4	4

4. Del reconocedor anterior se puede obtener la gramática

$$\begin{cases} S & \to & aS \mid bB \mid \epsilon \\ B & \to & aC \mid bB \mid \epsilon \\ C & \to & aC \mid \epsilon \end{cases}$$

   cuya TASP es:

primeros	S	B	C
	a	a	a
	b	b	ϵ
	ϵ	ϵ	

siguientes	S	B	C
	\$	\$	\$

TASP	a	b	\$
S	aS	bB	ϵ
B	aC	bB	ϵ
C	aC		ϵ

3 **(1 p.)** Siguiendo los mismos mecanismos que para las expresiones aritméticas, se construye:

$$G_R : \begin{cases} R & \to & R + T \mid T \\ T & \to & T \bullet F \mid F \\ F & \to & F * \mid U \\ U & \to & (R) \mid \mathbf{1} \mid \mathbf{0} \mid \mathbf{a} \end{cases}$$

El lenguaje formado por las expresiones regulares **no** es un lenguaje regular. Considérese para justificarlo su intersección, por ejemplo, con el lenguaje (regular) $(^*\mathbf{0})^*$, que resulta $\{(^n\mathbf{0})^n \ / \ n \geq 0\}$, que no es regular (no cumple el lema de bombeo).

4 **(2 p.)** 1. $\emptyset$ (A nunca sale de la pila).

2. Las cadenas deben ser de la forma $1^{n+1}01^{m+1}0$, con $m \geq 0$ y $n \geq 0$ y además ser consumidas completamente por el autómata, acabando éste en el estado q. Desde el estado inicial (q), una vez apilados todos los unos sobre la A, el primer 0 lo llevará al estado p, desde donde, para llegar al estado q de nuevo, habrá que desapilar los unos para encontrar el símbolo A en la pila con la lectura del último 0, que lleva al estado final q. Por lo tanto el lenguaje pedido es $\{1^n01^n0 \ / \ n > 0\}$
3. L no es regular. Si lo fuera, $L \cap 1^+01^+0$, calculado en el apartado anterior, también lo sería; pero evidentemente no lo es, lo que puede probarse comprobando que no cumple el lema de bombeo de los lenguajes regulares.
4. Se añaden al dado los estados p_1 y r, y un nuevo símbolo de la pila, B. p_1 es el nuevo estado inicial y B el nuevo símbolo de inicio de pila, y las tablas de transición como sigue:

$\to p_1$	ϵ
B	(q, AB)

$\not\to(q)$	0	1	ϵ
A		$(q, 1A)$	(r, ϵ)
1	$(p, 1)$	$(q, 11)$	(r, ϵ)
B			(r, ϵ)

p	0	1
A	(q, A)	
1		(p, ϵ)

r	ϵ
A	(r, ϵ)
1	(r, ϵ)
B	(r, ϵ)

5 **(0'7 p.)** 1. Con las condiciones del enunciado, resulta que $L_2 = (L_1 \cup L_2) - L_1 = (L_1 \cup L_2) \cap \overline{L_1}$ Como, dado un lenguaje recursivo, su complementario lo es, y la intersección de recursivos es recursiva, L_2 debe ser recursivo.

Otra forma de verlo es construir el algoritmo:

```
leer x
si MR(L_1 U L_2)(x) = SI entonces (está en la unión)
     si MR(L_1) = SI entonces (está en la unión y en L1, luego no en L2)
               escribir NO
     si no (está en la unión y no en L1, luego en L2)
               escribir SI
     fin si
si no  (no está en la unión, luego tampoco en L2)
     escribir NO
fin si
```

siendo `MR(L)` una máquina reconocedora para `L` en cada caso. Como la unión de L_1 y $L-2$ es recursiva y L_1 también, existen máquinas reconocedoras que siempre se paran. El algoritmo anterior, por lo tanto, para siempre, y caracteriza el lenguaje L_2

2. Si la intersección no es vacía, el razonamiento anterior ya no es válido, y es posible que L_2 no sea recursivo.

   Por ejemplo, tómese un lenguaje L_2 no recursivo cualquiera, y como L_1 el lenguaje universal. Evidentemente, L_1 es recursivo (hasta es regular) y $L_1 \cup L_2$ también (es L_1), pero L_2 no lo es.

   (Por otra parte, tómese $L_1 = a^*$ y $L_2 = b^*$ (no son disjuntos). En este caso la unión es recursiva y L_1 y L_2 también.

   Por lo tanto, si la intersección no es vacía, puede pasar cualquier cosa).

6 **(1'5 p.)** Usando las herramientas Lex y Yacc, construir un analizador sintáctico para las expresiones de sumas y productos de polinomios de hasta grado 10. Cada polinomio de la expresión de entrada estará encerrado entre corchetes. Los términos del polinomio estarán constituidos por tres elementos: un número entero (el coeficiente del término) seguido de la variable y por último un número entero menor o igual que 10 (el grado del término). El programa resultante deberá devolver el resultado de realizar las operaciones algebraicas indicadas en la expresión de la entrada sólo en el caso de que ésta sea correcta.

Por ejemplo: Entrada: `([3x3 - 5x1+6x0] + [1x2+7x1]) * [1x1+1x0]`

Salida: `[3x4+4x3+3x2+8x1+6x0]`

7 **(0'8 p.)** Preguntas para contestar en esta misma página. (Se valorará la correcta expresión de la respuesta):

1. Para ambos existe una máquina de Turing reconocedora; pero para los recursivos, existe una que se para siempre (los caracteriza), mientras que para los recursivamente numerables no puede garantizarse que exista tal. Más concretamente, para los recursivamente numerables no recursivos (que los hay), seguro que no existe una máquina que siempre se pare y los reconozca.
2. Todo lenguaje de tipo 1 es recursivo. No todo lenguaje recursivo es de tipo 1.

   Todo lenguaje recursivo se recursivamente numerable. No todo lenguaje recursivamente numerable es recursivo.

   Los lenguajes recursivamente numerables son exactamente los de tipo 0.
3. A una máquina de Turing (MTU) que tiene como entradas otra máquina de Turing cualquiera (M) y la entrada para esta última (w), y cuya tarea es simular el proceso de la máquina leída sobre la cadena leída (M(w)).
4. Es una gramática en la que todas sus reglas tienen el consecuente de longitud mayor o igual que la del antecedente.

   A partir de una gramática no contractiva se puede construir un algoritmo de reconocimiento del lenguaje generado, que caracterizará dicho lenguaje (parándose siempre); es decir, la existencia de una gramática no contractiva garantiza la recursividad del lenguaje generado.