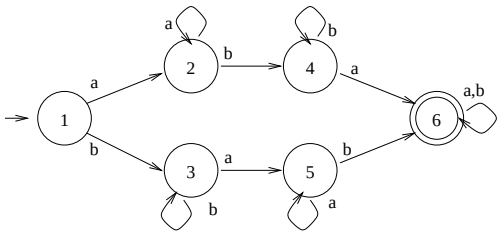
Soluciones

1

1. [F] Si L_1 cumple el lema de bombeo de los lenguajes regulares entonces es regular.
2. [V] Si L_1 no cumple el lema de bombeo de los lenguajes regulares entonces no es regular.
3. [V] Si L_1 y L_2 son independientes de contexto, entonces L_1L_2 también lo es.
4. [F] Si L_1 y L_2 son independientes de contexto, entonces $L_1 \cap L_2$ también lo es.
5. [F] Si L_1 es recursivamente numerable, entonces es recursivo.
6. [F] Si L_1 es recursivamente numerable, entonces su complementario también lo es.
7. [V] Si L_1 y L_2 son recursivamente numerables, entonces su unión también lo es.
8. [V] Si L_1 y L_2 son recursivos, su intersección también lo es.
9. [V] Si L_1 es independiente de contexto, entonces su complementario es recursivo.

2



	a	b	significado del estado
$\rightarrow 1$	2	3	
2	2	4	leído $\in a^+$
3	5	3	leído $\in b^+$
4	6	4	leído $\in a^+b^+$, subcadena ab
5	5	6	leído $\in b^+a^+$, subcadena ba
(6)	6	6	condición cumplida

El autómata es mínimo, lo que puede comprobarse aplicando el algoritmo de minimización, o considerando una tabla como la siguiente, en la que en cada casilla aparece una cadena que distingue a los estados que encabezan la fila y columna correspondientes.

	1	2	3	4	5
2	ba				
3	ab	ba			
4	a	a	a		
5	b	b	b	b	
6	ϵ	ϵ	ϵ	ϵ	ϵ

Puede obtenerse directamente, o a partir de la observación de que el lenguaje pedido es $L_1 \cap L_2$ siendo L_1 el formado por las cadenas que admiten ab como subcadena y L_2 el de las que admiten ba. Como fácilmente se ve, el RFD mínimo para L_1 es:

R_1 :

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

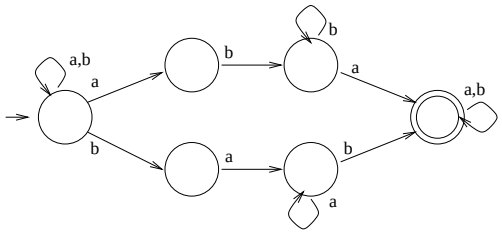
y similar para L_2 con lo que habrá que

- complementar R_1
- complementar R_2
- "unir" los obtenidos (arco ϵ a cada uno de los estados iniciales de R_1 y R_2 desde el nuevo estado inicial, RF no determinista)

- complementar este último.

El proceso obtiene exactamente el propuesto en primer lugar.

Otra posibilidad es partir de una expresión para el lenguaje, por ejemplo $(a|b)^*(abb^*a|baa^*b)(a|b)^*$i, para obtener un RFN y aplicar el algoritmo de determinación



Expresión regular (obtenida del reconocedor determinista): $(aa^*bb^*a|bb^*aa^*b)(a|b)^*$

Gramáticas de tipo 3:

$$\begin{array}{lcl} S & \to & aB \mid bC \\ B & \to & aB \mid bD \\ C & \to & aE \mid bC \\ D & \to & aF \mid bD \mid a \\ E & \to & aE \mid bF \mid b \\ F & \to & aF \mid bF \mid a \mid b \end{array} \qquad \begin{array}{lcl} S & \to & Sa \mid Sb \mid Da \mid Eb \\ D & \to & Db \mid Bb \\ E & \to & Ea \mid Ca \\ B & \to & Ba \mid a \\ C & \to & Cb \mid b \end{array}$$

3

1. G_1 y G_2 son de tipo 2 y G_3 de tipo $0'$.
2. $L(G_1) = \{a^{4k+1}/k \geq 0\}$
   $L(G_2) = \{a^{2k}ca^{2k}/k \geq 0\}$
   $L(G_3) = \{a^{6k+1}/k \geq 0\}$
3. $L(G_1) = (aaaa)^*a$ es regular.
   $L(G_2)$ es independiente de contexto (basta comprobar que no verifica el lema de bombeo de los regulares).
   $L(G_3) = (aaaaaa)^*a$ es regular.
4. G_1 no es $LL(1)$ (no está factorizada por la izquierda).
   G_2 es $LL(1)$:

Primeros	S	A	B	C
	a	a	a	a
	c			c

$TASP$	a	c	\$
S	$S \to aA$	$S \to c$	
A	$A \to Ba$		
B	$B \to aC$		
C	$C \to Sa$	$C \to Sa$	

5. De la derivación más a la derecha $S \Rightarrow \underline{aA} \Rightarrow a\underline{Ba} \Rightarrow a\underline{aC}a \Rightarrow aa\underline{Sa}a \Rightarrow aa\underline{c}aa$
   se deduce el análisis

PILA	ENTRADA	ACCIÓN
\$	$aacaa$\$	desplazar
\$$a$	$acaa$\$	desplazar
\$$aa$	caa\$	desplazar
\$$aa\underline{c}$	aa\$	reducir $S \to c$
\$$aaS$	aa\$	desplazar
\$$aa\underline{Sa}$	a\$	reducir $C \to Sa$
\$$a\underline{aC}$	a\$	reducir $B \to aC$
\$$aB$	a\$	desplazar
\$$a\underline{Ba}$	\$	reducir $A \to Ba$
\$$\underline{aA}$	\$	reducir $S \to aA$
\$$S$	\$	

Será $LALR(1)$, porque las formas sentenciales sólo pueden ser de 5 formas, y en cada una de ellas el pivote puede determinarse sin depender de ningún símbolo posterior:

a^pAa^q	$p = q+1$ y q par	el pivote está en aA con la única regla posible
a^pBa^q	$p = q$ impar	el pivote está en Ba con la única regla posible
a^pCa^q	$p = q+1$ y q impar	el pivote está en aC con la única regla posible
a^pSa^q	$p = q$ y q par	el pivote está en aSa con la única regla posible

Por lo tanto, el analizador deberá desplazar mientras lea aes, distinguiendo si el número de desplazamientos ha sido par o impar. Si el número de aes desplazadas es par, y se lee una c también se desplazará (en otro caso, error). Una c en la cima de la pila con una a en la entrada forzará a una reducción por $S \to c$ (una única vez). A partir de este momento, se realizarán los desplazamientos adecuados para conseguir los únicos posibles consecuentes de pivotes aA, Ba, aC y Sa en la cima de la pila, que serán reducidos en cuanto aparezcan según la única regla posible.

3 1. (Se usa $\diamond$ para representar el blanco)

$$\begin{aligned}(q_1, \underline{a}ababbb) &\vdash (q_1, a\underline{a}babbb) \vdash (q_1, aa\underline{b}abbb)\\ &\vdash (q_2, a\underline{a}babbb)\\ &\vdash (q_3, \underline{a}ababbb)\\ &\vdash (q_2, \underline{\diamond}aababbb)\\ &\vdash (q_6, \underline{a}ababbb)\\ &\quad Parada(q_6)\end{aligned}$$

$$\begin{aligned}(q_1, \underline{a}aababbb) &\vdash^* (q_1, aaa\underline{b}abbb)\\ &\vdash (q_2, aa\underline{a}babbb)\\ &\vdash (q_3, a\underline{a}ababbb)\\ &\vdash (q_2, \underline{a}aababbb)\\ &\vdash (q_3, \underline{\diamond}aaababbb)\\ &\vdash (q_4, \underline{\diamond}\diamond aaababbb)\\ &\vdash (q_4, \underline{\diamond}\diamond\diamond aaababbb)\\ &\vdash \ldots(\infty)\end{aligned}$$

$$(q_1, \underline{a}aa) \vdash^* (q_1, aaa\underline{\diamond}) \vdash (q_5, aa\underline{a}) \quad Parada(q_5)$$

2. $L_r = \{x \in (a|b)^* \ / \ q_1 x \vdash^* \alpha q_5 \beta\} = a^*$
3. La existencia de esta máquina de Turing, que no siempre se para, demuestra que L_r es recursivamente numerable (aunque de hecho sea regular).
4. $a^* \mid (aa)^+ b(a|b)^*$

   El primer término de la unión describe las cadenas para las que la máquina se para en q_5, y el segundo para las que se para en q_6. La máquina no se puede parar en ningún otro estado.
5. L_p es (al menos) recursivamente numerable, puesto que basta construir una nueva máquina que devuelva "SI" (pase a un estado de aceptación y parada) cada vez que la primera se pare (aceptando o no).

```
%{
#include "y.tab.h"
%}
if      [Ii][Ff]
then    [Tt][Hh][Ee][Nn]
letra   [a-zA-Z]
dig_    [0-9_]
id      {letra}({letra}|{dig_})*
%%
[ \\t\\n]+     ;
{if}           return IF;
{then}         return THEN;
{id}           return ID;
":="           return ASIGN;
"<"            |
"<="           |
"<>"           |
">"            |
">="           |
"="            return OPREL;
";"            return yytext[0];
.              printf ("Error léxico: %s\\n", yytext);
```

Fuente Yacc:

```
%{
#include <stdio.h>
#include "lex.yy.c"
yyerror( char * s)
       { fprintf (stderr, "%s\n", s); }

%}
%token IF THEN ID ASIGN OPREL
%%
L       : L ';' S
        | S
        ;
S       : IF C THEN A
        ;
A       : ID ASIGN ID
        ;
C       : ID OPREL ID
        ;
%%
main(){
       yyparse();
}
```

Proceso:

```
yacc -d AS.y
lex AL.l
cc y.tab.c -ll
```