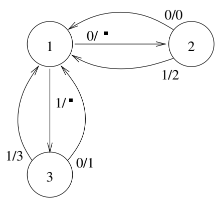
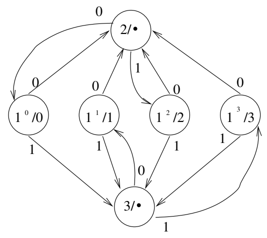
Soluciones del examen ordinario de 16 de junio de 2003.

$\boxed{1}$ La máquina propuesta y la de Moore equivalente (apartado 2) se representan gráficamente:





1. Partiendo del estado q_1 se tiene:
   La salida para 011011 es $\bullet 2 \bullet 1 \bullet 3$, y la máquina queda en el estado q_1
   La salida para 0110111 es $\bullet 2 \bullet 1 \bullet 3 \bullet$, y la máquina queda en el estado q_3 (la misma que para 011011 concatenando después la salida de 1 desde q_1)
   La salida para 01101110 es $\bullet 2 \bullet 1 \bullet 3 \bullet 1$, y la máquina queda en el estado q_1 (la misma que para 0110111 concatenando después la salida de 0 desde q_3)
   La salida para 01101111 es $\bullet 2 \bullet 1 \bullet 3 \bullet 3$, y la máquina queda en el estado q_1 (la misma que para 0110111 concatenando después la salida de 1 desde q_3)
2. Moore equivalente (gráfico anterior); tabla:

	0	1
$q_1^0/0$	$q_2^\bullet$	$q_3^\bullet$
$q_1^1/1$	$q_2^\bullet$	$q_3^\bullet$
$q_1^2/2$	$q_2^\bullet$	$q_3^\bullet$
$q_1^3/3$	$q_2^\bullet$	$q_3^\bullet$
$q_2^\bullet/\bullet$	q_1^0	q_1^2
$q_3^\bullet/\bullet$	q_1^1	q_1^3

   El algoritmo de minimización establece la primera partición (0) como

$$\{q_1^0\}, \{q_1^1\}, \{q_1^2\}, \{q_1^3\}, \{q_2^\bullet, q_3^\bullet\}$$

   y en la segunda se obtiene

$$\{q_1^0\}, \{q_1^1\}, \{q_1^2\}, \{q_1^3\}, \{q_2^\bullet\}, \{q_3^\bullet\}$$

   por lo tanto la máquina es mínima.
3.

*base*2	*base*10	*base*4	inverso de la salida	*estado*	*pendiente*
$\underline{11}\,\underline{01}\,\underline{10}$	54	312	$3 \bullet 1 \bullet 2 \bullet$	1	–
$\underline{1}\,\underline{11}\,\underline{01}\,\underline{10}$	118	1312	$\bullet 3 \bullet 1 \bullet 2 \bullet$	3	1
$\underline{01}\,\underline{11}\,\underline{01}\,\underline{10}$	118	1312	$1 \bullet 3 \bullet 1 \bullet 2 \bullet$	1	–
$\underline{11}\,\underline{11}\,\underline{01}\,\underline{10}$	246	3312	$3 \bullet 3 \bullet 1 \bullet 2 \bullet$	1	–

   El estado almacena la situación cuando se ha leído el último bit. Si el estado es q_1, el resultado es la representación en base 4 del número binario de entrada, eliminando los puntos. Si el estado es q_2, falta añadir un 0; y si es q_3, falta añadir un 1.

   La forma en la que se pasa a base 4 desde base 2 por este autómata, se basa en agrupar los bits de dos en dos. Cada dos bits se traducen en un dígito de base 4 (de la misma manera que tres bits se agrupan para formar un dígito octal, o cuatro para formar un hexadecimal). Cuando el número de bits es impar, el último grupo está incompleto, pero el estado almacena la información parcial: q_2 recuerda un 0 y q_3 un 1. Si la cadena se termina en ese momento, del estado se puede recuperar esta información.

En resumen, un algoritmo que permitiría usar esta máquina para pasar a base 4 sería:

```
leer x, cadena de ceros y unos
proporcionar a la máquina de Mealy los elementos de x en orden inverso
   y escribir la salida en dígitos que va proporcionando, sin escribir los puntos
si el estado en el que ha terminado es q2, escribir 0
   si no, si es q3, escribir 1
interpretar la salida en orden inverso
```

Otra posibilidad es forzar a que la entrada tenga longitud par, añadiendo un 0 si no. En este caso, la conversión sería completa.

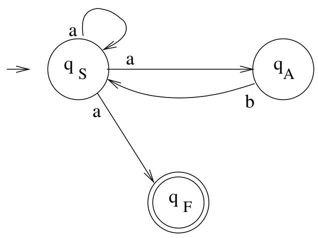
$\boxed{2}$ 1. Sin símbolos inútiles, la gramática resultante es

$$\begin{array}{rcl} S & \to & aS \mid aA \mid a \\ A & \to & bS \end{array}$$

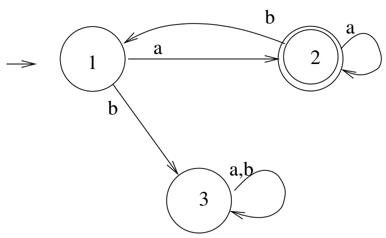
de donde se deducen fácilmente las las cadenas generadas de longitud menor o igual a 4:

$$\{a, aa, aaa, aaba, aba, aaaa, abaa\}$$

2. De la gramática sin símbolos inútiles se obtiene el autómata no determinista:



y a partir de éste, el determinista mínimo



El algoritmo de análisis daría:

$$\begin{array}{rcl}
\alpha_{12}^3 & = & \alpha_{12}^2 | \alpha_{13}^2 {\alpha_{33}^2}^* \alpha_{32}^2 \\
& = & \alpha_{12}^2 | \alpha_{13}^2 {\alpha_{33}^2}^* \emptyset \\
& = & \alpha_{12}^2 \\
& = & \alpha_{12}^1 \mid \alpha_{12}^1 (\alpha_{22}^1)^* \alpha_{22}^1 \\
& = & \alpha_{12}^1 (\alpha_{22}^1)^* \\
\alpha_{12}^1 & & \\
& = & \alpha_{12}^0 \mid \alpha_{11}^0 (\alpha_{11}^0)^* \alpha_{12}^0 \\
& = & a \mid \lambda(\lambda)^* a \\
& = & a \\
\alpha_{22}^1 & & \\
& = & \alpha_{22}^0 \mid \alpha_{21}^0 (\alpha_{11}^0)^* \alpha_{12}^0 \\
& = & (\lambda \mid a) \mid b(\lambda)^* a \\
& = & \lambda | a | ba \\
\alpha_{12}^3 & = & a(\lambda | a | ba)^* \\
& = & a(a|ba)^*
\end{array}$$

Otra expresión equivalente: $a^+(ba^+)^*$

3. Resulta que $(a|ab)^*a = (a(\lambda|b))^*a = a((\lambda|b)a)^* = a(a|ba)^*$ y por lo tanto la diferencia simétrica pedida es $\emptyset$.

   También puede obtenerse la expresión $(a|ab)^*a$ de la observación del autómata no determinista que se obtuvo partiendo de G_1, o comprobar que las dos expresiones son iguales por cualquier método (doble contenido, construyendo los autómatas deterministas mínimos de cada expresión y comparando ...)

4. A partir del reconocedor determinista se puede deducir G_2:

$$\begin{array}{lcl} A_1 & \rightarrow & aA_2 \mid bA_3 \\ A_2 & \rightarrow & aA_2 \mid bA_1 \mid \lambda \\ A_3 & \rightarrow & aA_3 \mid bA_3 \end{array}$$

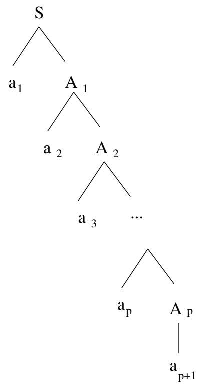
   que, eliminando símbolos inútiles, daría

$$\begin{array}{lcl} A_1 & \rightarrow & aA_2 \\ A_2 & \rightarrow & aA_2 \mid bA_1 \mid \lambda \end{array}$$

   o, si se quiere, sin reglas λ:

$$\begin{array}{lcl} A_1 & \rightarrow & aA_2 \mid a \\ A_2 & \rightarrow & aA_2 \mid bA_1 \mid a \end{array}$$

5. Forma de los árboles:



   Forma de las derivaciones (m.d., m.i es lo mismo):

$$S \Rightarrow a_1A_1 \Rightarrow a_1a_2A_2 \Rightarrow a_1a_2a_3A_3 \Rightarrow \ldots \Rightarrow a_1a_2a_3 \ldots a_pA_p \Rightarrow a_1a_2a_3 \ldots a_pa_{p+1}$$

   Las formas sentenciales son, o bien S, o bien una cadena no nula de terminales seguida de un único auxiliar, o bien una cadena de terminales no nula; es decir, están en el conjunto $\Sigma_T^+\Sigma_A \cup \Sigma_T^+ \cup \{S\}$ Si la forma sentencial es una cadena de terminales, el consecuente del pivote es justamente el último símbolo. Si es una cadena de terminales seguida de un auxiliar, el consecuente del pivote es el sufijo de longitud 2, formado por un terminal y un auxiliar. La regla por la que debe hacerse la reducción dependerá de la gramática concreta.

6. G_1 es $LALR(1)$.

   Dado que se conoce en cada forma sentencial cuál es el consecuente del pivote, y que ninguna forma sentencial utilizará los símbolos B ni C (dado que, al ser inútiles, ninguna forma sentencial que los contenga llegará a convertirse en una cadena de terminales), está perfectamente determinado si ha de realizarse en cada momento un desplazamiento o una reducción. Las reducciónes sólo pueden hacerse por reglas de G_2, y ningún par de ellas tiene el mismo consecuente, por lo que no se producirán conflictos reducción-reducción.

   Se hará un desplazamiento siempre que queden terminales por leer. Cuando ya no haya más (se encuentre el símbolo de fin de cadena en la entrada), se deberá realizar una reducción, puesto que el consecuente del pivote es el último símbolo de la entrada. Si el símbolo de la cima de la pila es entonces a, se realizará la única reducción con consecuente a ($S \rightarrow a$). A partir de ese momento,

   - en la cima de la pila estará aS, en cuyo caso se debe reducir por la regla $S \rightarrow aS$,
   - o en la cima de la pila estará bS, que forzará a la reducción por $A \rightarrow bS$,

- o en la cima de la pila aparece aA, que llevará a la reducción por $S \to aA$,
- o la pila contendrá únicamente S, en cuyo caso se habrá terminado el análisis con éxito.

Todos los demás casos deberán producir un error de análisis.

7. G_1 no es ambigua, puesto que es $LALR(1)$

   G_2 no es ambigua, puesto que el funcionamiento del autómata proporciona la única derivación posible. Planteando un análisis descendente, desde el símbolo inicial S, la decisión entre $S \to a$ y $S \to aA$ viene determinada según la a sea o no la última de la cadena. A partir de ahí, la expansión de A viene determinada por el símbolo siguiente: b obliga a $A \to bS$, y a, nuevamente obliga a una de las dos reglas que empiezan por a según ésta sea o no el último símbolo de la entrada.

   G_3 es ambigua:

$$S \Rightarrow aS \Rightarrow aaA \Rightarrow aabS \Rightarrow aaba$$

$$S \Rightarrow aA \Rightarrow aaA \Rightarrow aabS \Rightarrow aaba$$

8. Ningún lenguaje de tipo 3 es inherentemente ambiguo: siempre podrá encontrarse un autómata finito determinista que reconozca $L - \{\lambda\}$, y la gramática obtenida directamente de él será inambigua, realizando las generaciones de cadenas de la misma forma *determinista*, símbolo a símbolo, que el autómata realiza los reconocimientos.

3 1. L_2 no verifica el lema de bombeo, por ejemplo en su versión fuerte, ya que, dado **cualquier** N que se propusiera, la cadena $z = a^N b^N$, que está en L_2 y tiene longitud $2N \geq N$, no admite ninguna subcadena u bombeable en su primer fragmento de longitud N, ya que debería estar constituída únicamente por *aes*, y, al suprimirla, la cadena resultante tendría menos *aes* que *bes*, y no estaría en L_2. Por lo tanto L_2 no puede ser regular.

2. Se sabe que L_2 puede generarse mediante la gramática independiente de contexto $S \to aSb \mid \lambda$, y por lo tanto **es** independiente de contexto, y por lo tanto **debe** verificar el lema de bombeo de los lenbguajes independientes de contexto.

   (En concreto, para $N = 3$, si $z \in L_2$ y $|z| \geq 3$, siempre z tendrá una a y una b para hacer el papel de y y v, o v e y en el bombeo xy^iuv^iw)

3. L_1 verifica el lema de bombeo de los lenguajes regulares, por ejemplo para la constante $N = 2$. En cuanto una cadena de L_1 tenga longitud mayor o igual que 2, contendrá alguna subcadena de la forma ab o de la forma ba, que puede bombearse tranquilamente sin salirse del lenguaje, ya que siempre se sigue manteniendo que el número de aes coincide con el de bes.

4. L_1 no es regular, ya que, si lo fuera, también sería regular $L_1 \cap a^*b^*$, y éste es L_2, que no es regular, dado que. como se ha visto en 1, no verifica el lema de bombeo.

   También se puede comprobar que L_1 no verifica la versión fuerte del lema de bombeo, puesto que, para cualquier N, la cadena $a^N b^N$, que pertenece a L_1 y tiene longitud mayor que N, no admite ninguna subcadena bombeable que esté en el primer fragmento de N caracteres, formado sólo por aes.

4 1. La respuesta adecuada es (d). Por ser el lenguaje recursivamente numerable, cualquier programa que lo reconozca debe responder afirmativamente ante cadenas de L, y quizá negativamente ante las de $\overline{L}$, pero, al no ser recursivo, es **seguro** que esto ocurre alguna vez (de lo contrario, ya temndríamos un programa que realiza el reconocimiento "total").

2. P_L podría existir, pero, tomándolo como entrada de *superP*, éste determinaría si se para o no ante una cadena de entrada dada, con lo que obtendríamos un programa que reconoce completamente a L y a $\overline{L}$, y L sería recursivo, con lo que **el lenguaje** L es el que no podría existir. (NOTA: como sabemos que existe, el que desde luego no existe es *superP*, así que la empresa no ha podido construirlo). Por lo tanto, *superP*, de existir, **no** podría dar respuesta ante P_L con algunas cadenas, de forma que debería dar algún tipo de error: la respuesta más adecuada es (d).

5 1. `grep -n "^$"` *fichero*

2. `sed '/^$/d'` *fichero*

3. `awk '!/^$/{}'` *fichero*

6 ANALISIS LÉXICO: LEX

```
%{ #include "y.tab.h"
int cont;
%}
num (+|-)? [0-9]+ ("." [0-9]+)?
%S V
%%
"["         { c=0; BEGIN (V);}
<V>{num}    { c++;
              yylval.num[c] = atof(yytext);}
<V>"]"      { yylval.dim = cont;
              RETURN (VECT);
              BEGIN (INITIAL);}
"+"           RETURN ("+");
"\n"          RETURN ("\n");
.           { printf ('Error Léxico');
              exit(0);}
```

ANALISIS SINTÁCTICO: YACC

```
%{ typedef struct aux{
                int dim;
                float elem [10];
}
#define aux YYSTYPE;
int i;
%}
%token VECT '+' '\n' '(' ')'
%left "+"
%%
S:        exp '\n'     { printf('['); for (i=0; i++; i<=$1.dim){
                                          printf (%s, $1.elem[i];}
                         printf (']');
                       }
          ;
exp:      exp '+' exp  { if ($1.dim == $3.dim){
                            for (i=0; i++; i<=$1.dim){
                               $$.elem[i] = $1.elem[i] + $3.elem[i];}
                            $$.dim = $1.dim;}
                         else {
                            printf ('Error Semántico');
                         exit (0);}
                       }
          | '(' exp ')' { $$.dim = S2.dim;
                          for (i=0; i++; i<=$2.dim){
                               $$.elem[i] = $2.elem[i];}
                        }
          | VECT        { $$.dim = $1.dim;
                          for (i=0; i++; i<=$1.dim){
                               $$.elem[i] = $1.elem[i];}
                        }
          ;
%%
main () {
         yyparse();
}
```