

ETSIIT

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



Autómatas

Modelos de Computación

Jose Antonio Padial Molina

josepadial@correo.ugr.es

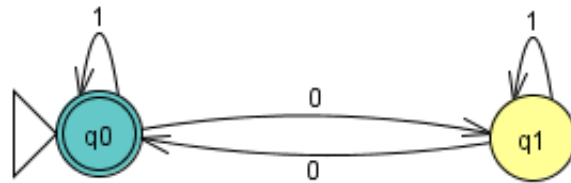
October 10, 2019

Contents

1	Ejercicio 1	2
2	Ejercicio 2	2
3	Ejercicio 3	3
4	Ejercicio 4	4
5	Ejercicio 5	4
6	Ejercicio 6	5
7	Ejercicio 7	6
8	Ejercicio 8	6

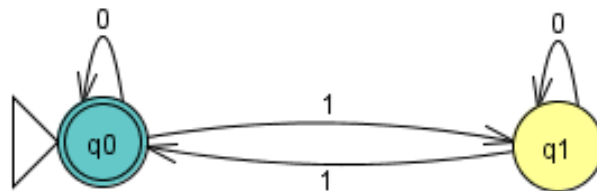
1 Ejercicio 1

Construir el autómata finito determinístico que acepta las palabras de ceros y unos, con un número de ceros que sea par.



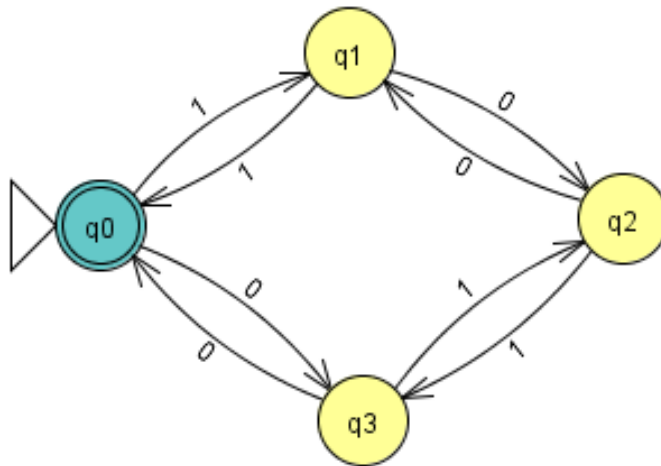
2 Ejercicio 2

Construir el autómata finito determinístico que acepta las palabras de ceros y unos, con un número de unos que sea par.

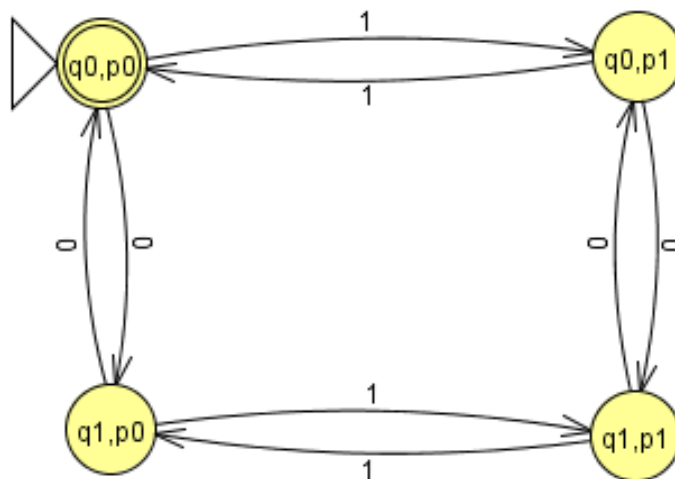


3 Ejercicio 3

Construir el autómata finito determinístico que acepta las palabras con un número de ceros que sea par y con un número de unos que sea par.

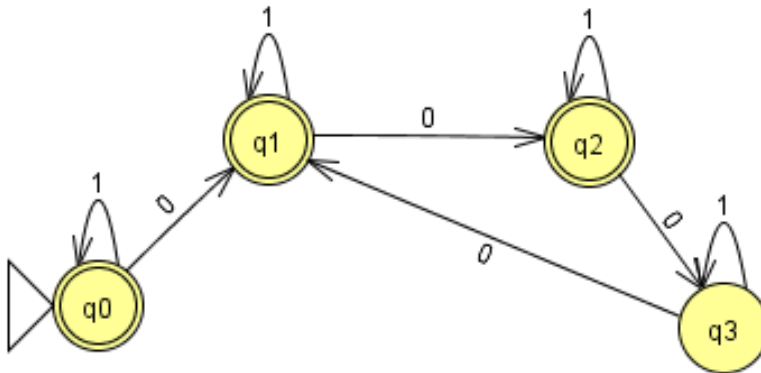


Este ejercicio se puede resolver como la combinación de los dos casos, (numero de 1 par) U (numero de 0 par). Donde el primer autómata sus estados van a ser q_x y el segundo p_x .



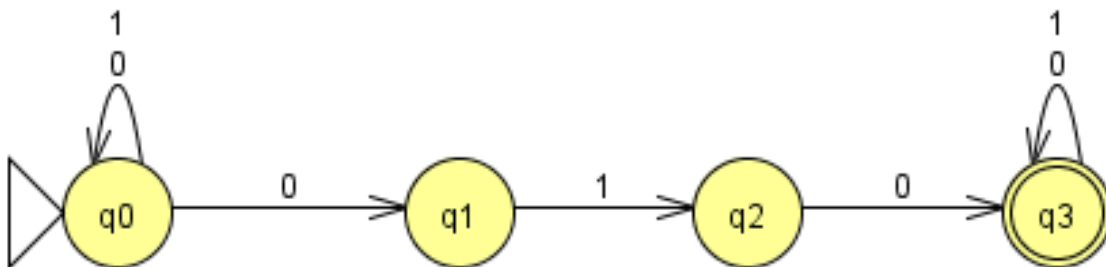
4 Ejercicio 4

Construir el autómata finito determinístico que acepta las palabras de ceros y unos, con un número de ceros que no sea múltiplo de 3.



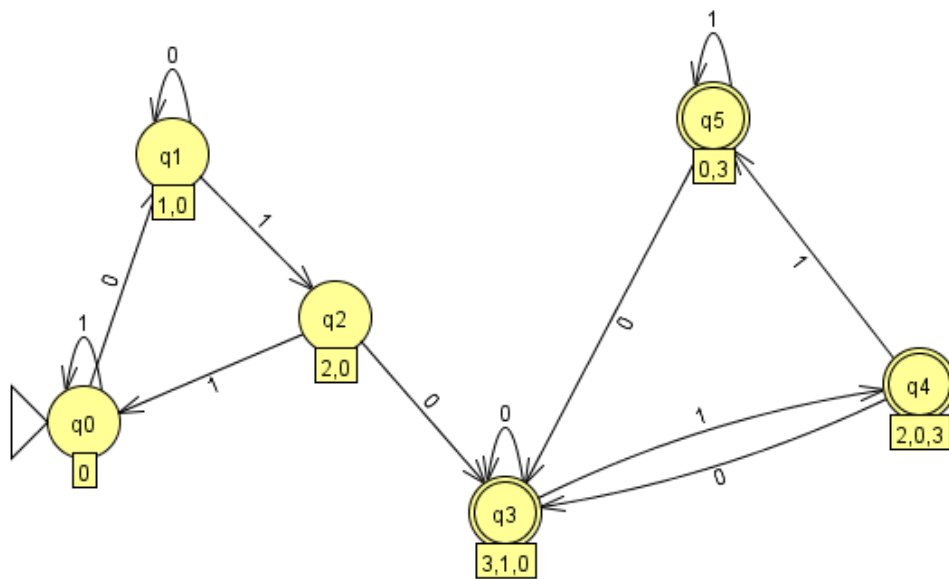
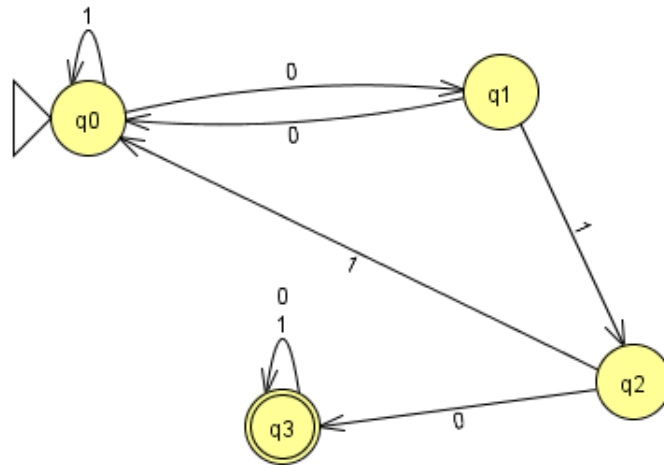
5 Ejercicio 5

Construir el autómata finito no determinístico que acepta las palabras de ceros y unos, que contenga la subcadena 010.



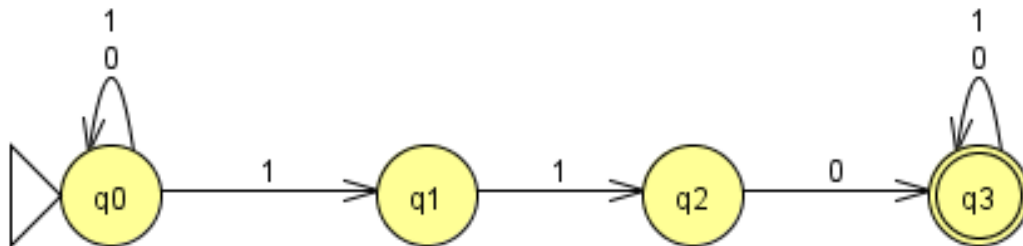
6 Ejercicio 6

Construir el autómata finito determinístico que acepta las palabras de ceros y unos, que contenga la subcadena 010.



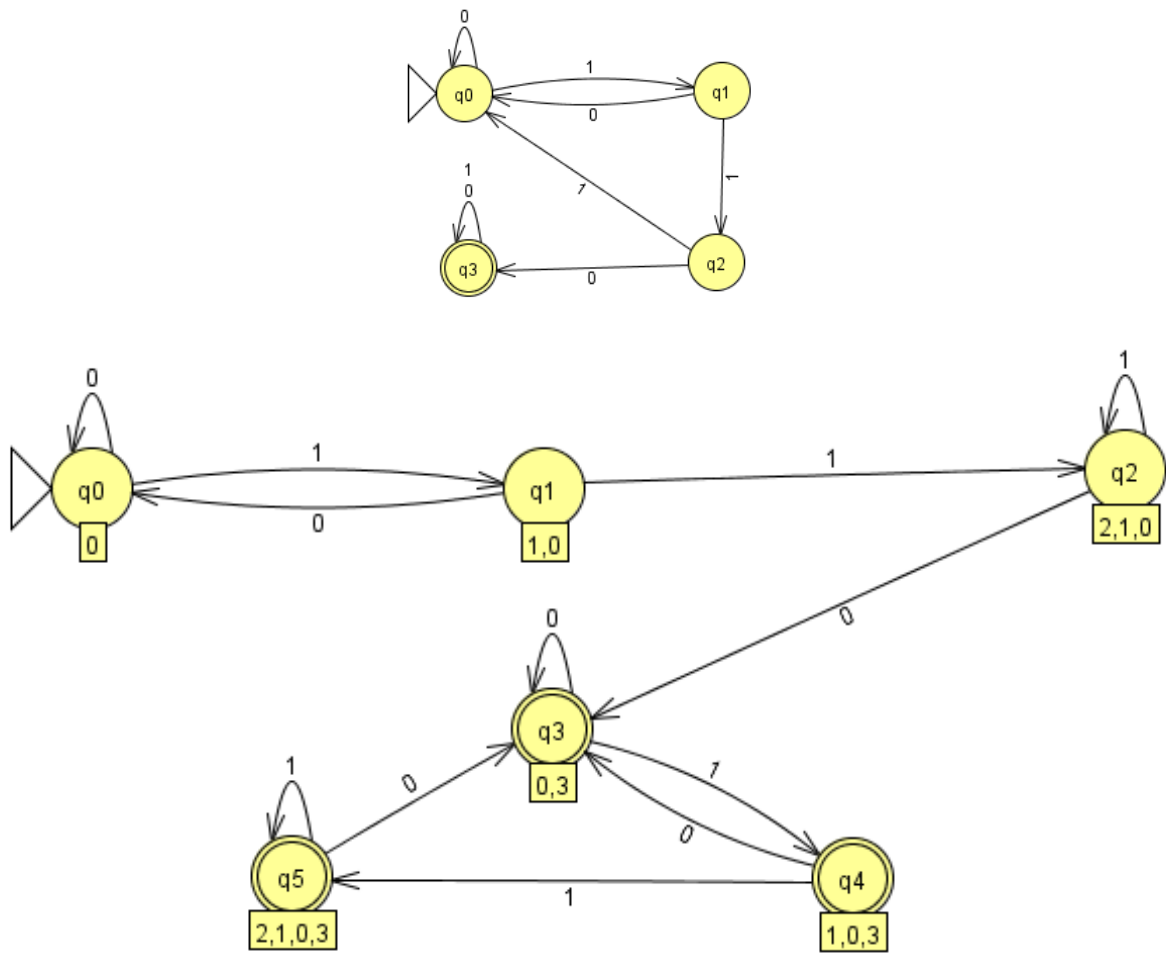
7 Ejercicio 7

Construir el autómata finito no determinístico que acepta las palabras de ceros y unos, que contenga la subcadena 110.



8 Ejercicio 8

Construir el autómata finito determinístico que acepta las palabras de ceros y unos, que contenga la subcadena 110.



ETSIIT

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



Máquinas de estados

Modelos de Computación

Jose Antonio Padial Molina

josepadial@correo.ugr.es

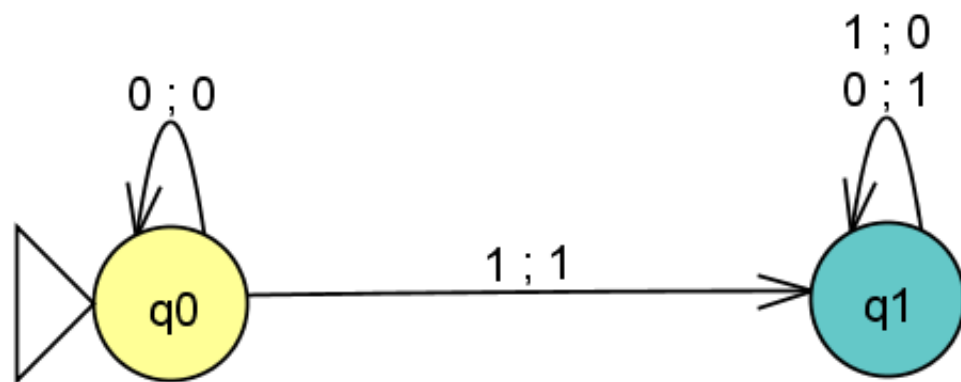
December 3, 2019

Contents

1	Ejercicio 1	2
2	Ejercicio 2	2
3	Ejercicio 3	3
4	Ejercicio 4	4
5	Ejercicio 5	5

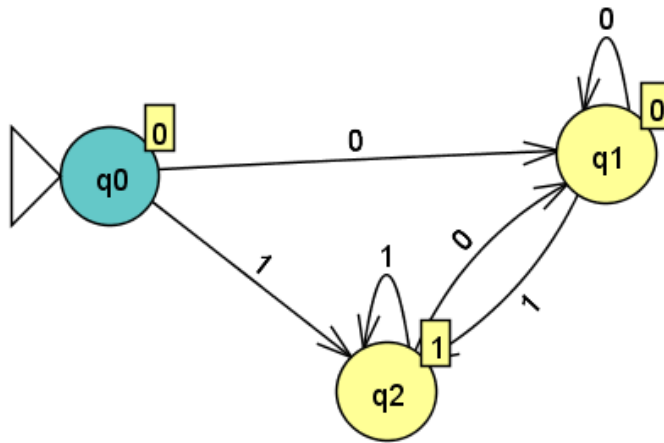
1 Ejercicio 1

Crear una máquina de Mealy que, teniendo como entrada un número binario, proporcione como salida su complemento a dos. El complemento a 2 de un número binario se calcula creando un nuevo número binario resultado de intercambiar todos los 0's por 1's y todos los 1's por 0's, y sumándole 1. Por ejemplo, ante la entrada 10101, la máquina deberá devolver 01011. En otro ejemplo, ante la entrada 0110, la máquina deberá devolver 1010.



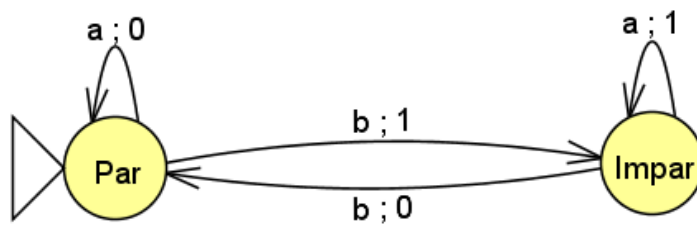
2 Ejercicio 2

Crear una máquina de Moore que tenga como entrada una secuencia de símbolos del alfabeto 0,1. Tras recibir el primer símbolo, la máquina deberá devolver 0 independientemente de que este sea 0 ó 1. Para los símbolos siguientes, la máquina devolverá el símbolo anterior de la entrada. Así, si Entrada(n) es el n-ésimo símbolo de entrada a la máquina y Salida(n) el n-ésimo símbolo de salida. Por ejemplo, para la entrada 10110, la máquina devolverá 01011.



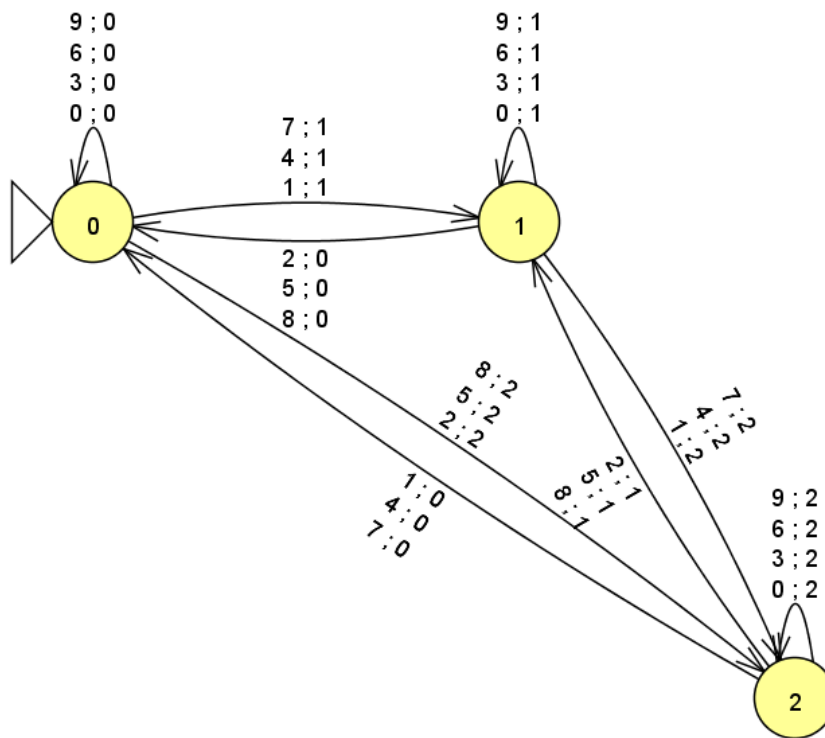
3 Ejercicio 3

Construir una maquina de Mealy que codifica palabras del alfabeto a, b en palabras del alfabeto $0, 1$ de acuerdo con las siguientes reglas: – Si la cantidad de símbolos b leído hasta el momento es par, entonces una a se transforma en un 0 , y una b en un 1 . – Si la cantidad de símbolos b leído hasta el momento es impar, entonces una a se transforma en un 1 y una b en un 0 .



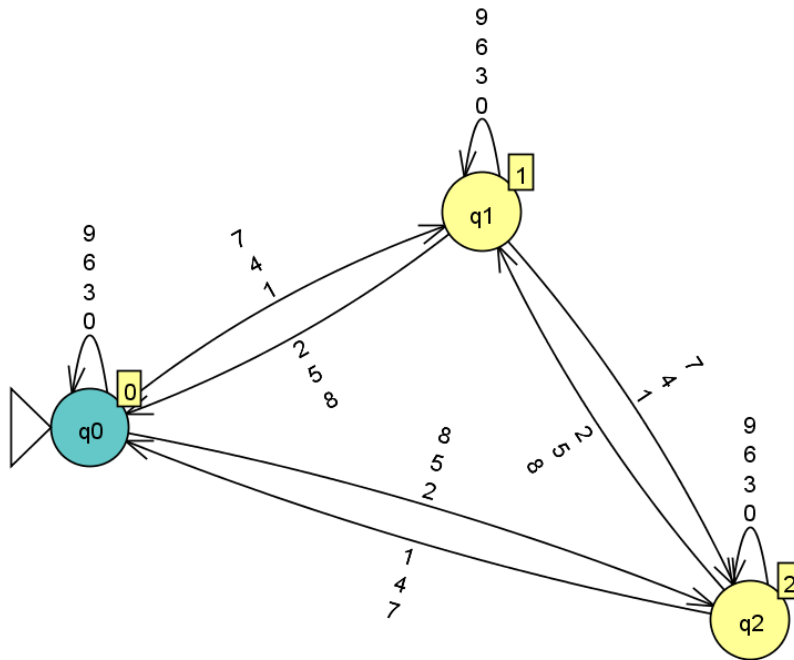
4 Ejercicio 4

Construir una Maquina de Mealy capaz de ir calculando la suma modulo 3 de los numeros que vaya recibiendo del alfabeto 0, 1, 2, 3, 4, 5, 6, 7, 8, 9



5 Ejercicio 5

Construir una Maquina de Moore capaz de ir calculando la suma modulo 3 de los numeros que vaya recibiendo del alfabeto 0, 1, 2, 3, 4, 5, 6, 7, 8, 9



ETSIIT

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



Práctica 2

Modelos de Computación

Jose Antonio Padial Molina

josepadial@correo.ugr.es

December 3, 2019

Contents

1	Ejercicio 1
---	-------------

2

I Ejercicio 1

Elegir un lenguaje regular, cualquiera. Obtener la expresión regular para las cadenas del lenguaje escogido. En JFlap pasar a NFA -> DFA -> Minimal -> Multiple Run -> Expression regular

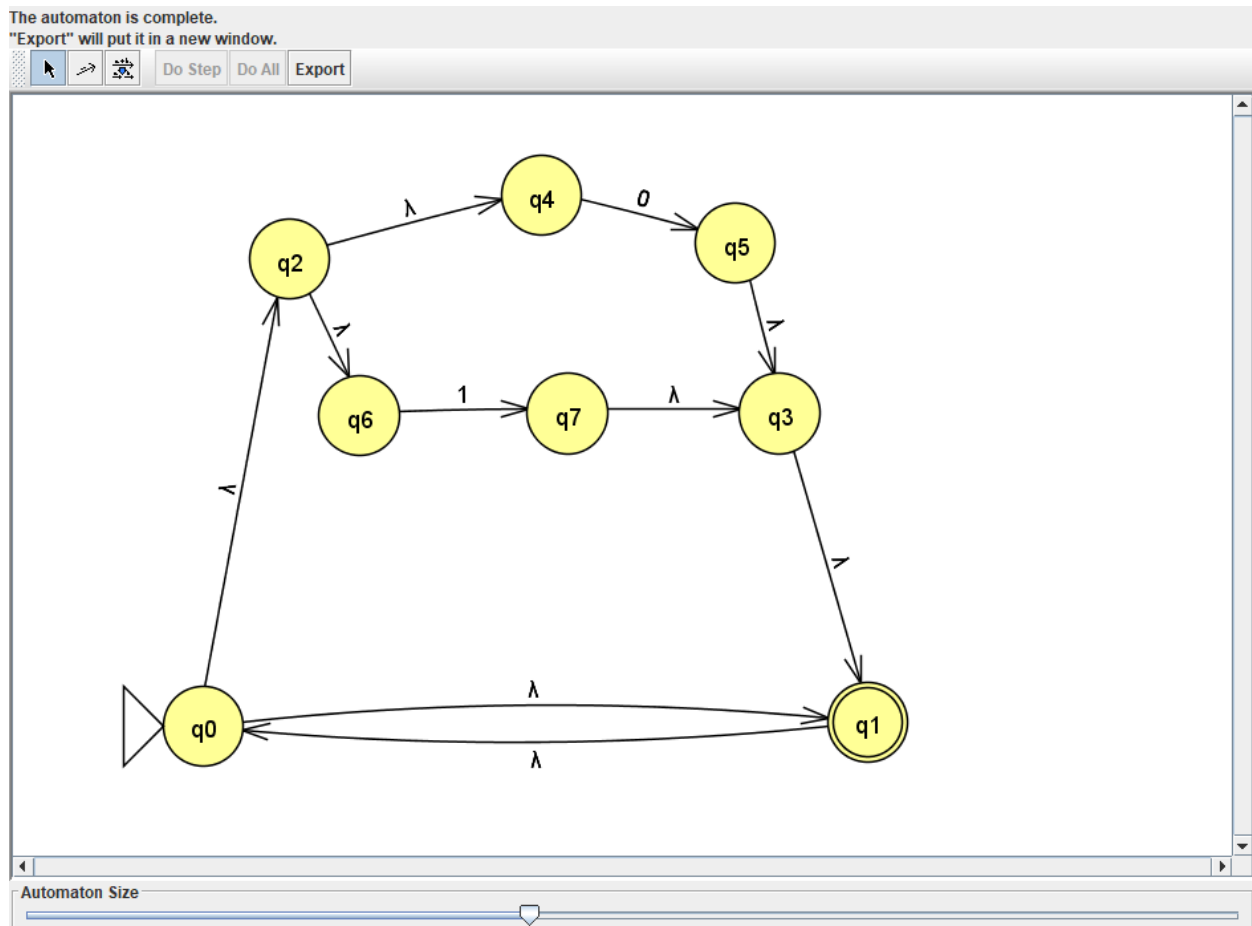
El lenguaje escogido es:

$$L = \{0, 1\}^*$$

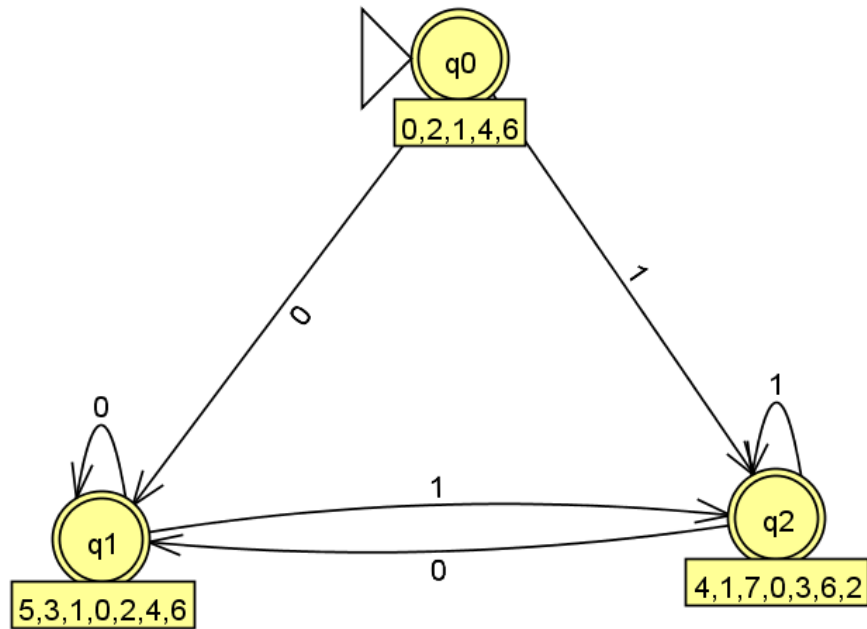
Y la expresión regular es:

$$(0 + 1)^*$$

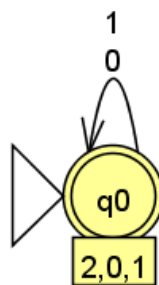
1. Pasar de expresion regular a NFA



2. Pasar de NFA a DFA



3. Pasar de DFA a minimal



4. Multiple run

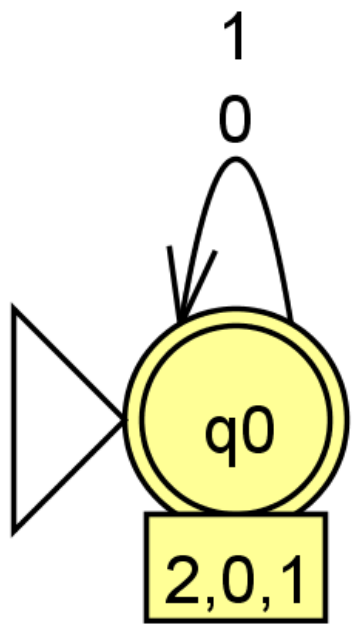


Diagram of a finite state automaton (FSA) with a single state q_0 . The state is represented by a yellow circle. A triangle on the left points to q_0 , indicating it is the start state. A self-loop on q_0 is labeled with '1' above and '0' below. Below the state circle is a box containing the tuple $2,0,1$.

Table Text Size

Input	Result
01001010101	Accept
01001001010	Accept
11111111111	Accept
00000000000	Accept
10101011101	Accept

Load Inputs Run Inputs Clear Enter Lambda View Trace

5. Pasar de minimal a gramatica

Hint Show All What's Left? Export

LHS		RHS
S	→	0S
S	→	1S
S	→	λ

Table Text Size

6. Pasar de DFA a expresion regular

Generalized Transition Graph Finished!

File Convert Help

Editor

Edit the regular expression below:

$(1+0)^*$

Input Field Text Size (For optimization, adjust the size of this window after resi

ETSIIT

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



LEX

Modelos de Computación

Jose Antonio Padial Molina

josepadial@correo.ugr.es

November 14, 2019

Contents

1	Calculadora de escritorio	2
2	calc.lex	2
3	calc.yacc	2
4	Compilar el programa	5
5	Capturas de pantalla	6

1 Calculadora de escritorio

Con lex y yacc se va a desarrollar una calculadora simple de escritorio. La cual realiza operaciones de suma, resta, multiplicación y división. Este programa de calculadora también le permite asignar valores a las variables (cada una designada con una letra minúscula) y luego usar las variables en los cálculos.

2 calc.lex

Especifica el archivo de especificación del comando lex que define las reglas de análisis léxico.

```
%{

#include <stdio.h>
#include "y.tab.h"
int c;
}%
%%
" "      ;
[a-z]    {
    c = yytext[0];
    yylval.a = c - 'a';
    return(LETTER);
}
[0-9]    {
    c = yytext[0];
    yylval.a = c - '0';
    return(DIGIT);
}
[~a-zA-Z0-9\b] {
    c = yytext[0];
    return(c);
}
%%
```

3 calc.yacc

Especifica el archivo de gramática del comando yacc que define las reglas de análisis y llama a la subrutina yylex creada por el comando lex para proporcionar información.

El código contiene las siguientes secciones:

- Declaraciones.
- Reglas: inicio, %union, %type, %toksn.
- Programas: Principals, errores.

```
%{
#include <stdio.h>
```



```

int regs[26];
int base;

%}

%start list

%union { int a; }

%token DIGIT LETTER

%left '|'
%left '&'
%left '+' '-'
%left '*' '/' '%'
%left UMINUS /*supplies precedence for unary minus */

%%
/* beginning of rules section */

list: /*empty */
    |
    list stat '\n'
    |
    list error '\n'
    {
        yyerrok;
    }
    ;
stat:
    expr
    {
        printf("%d\n", $1);
    }
    |
    LETTER '=' expr
    {
        regs[$1.a] = $3.a;
    }
    ;

expr:
    '(' expr ')'
    {
        $$ = $2;
    }
    |

```

```

expr '*' expr
{
    $$$.a = $1.a * $3.a;
}
|
expr '/' expr
{
    $$$.a = $1.a / $3.a;
}
|
expr '%' expr
{
    $$$.a = $1.a % $3.a;
}
|
expr '+' expr
{
    $$$.a = $1.a + $3.a;
}
|
expr '-' expr
{
    $$$.a = $1.a - $3.a;
}
|
expr '&' expr
{
    $$$.a = $1.a & $3.a;
}
|
expr '|' expr
{
    $$$.a = $1.a | $3.a;
}
|
'-' expr %prec UMINUS
{
    $$$.a = -$2.a;
}
|
LETTER
{
    $$$.a = regs[$1.a];
}

```

```

    |
    number
    ;

number: DIGIT
{
    $$ = $1;
    base = ($1.a==0) ? 8 : 10;
}
|
number DIGIT
{
    $$ .a = base * $1.a + $2.a;
}
;

%%

main()
{
    return(yyparse());
}

yyerror(s)
char *s;
{
    fprintf(stderr, "%s\n",s);
}

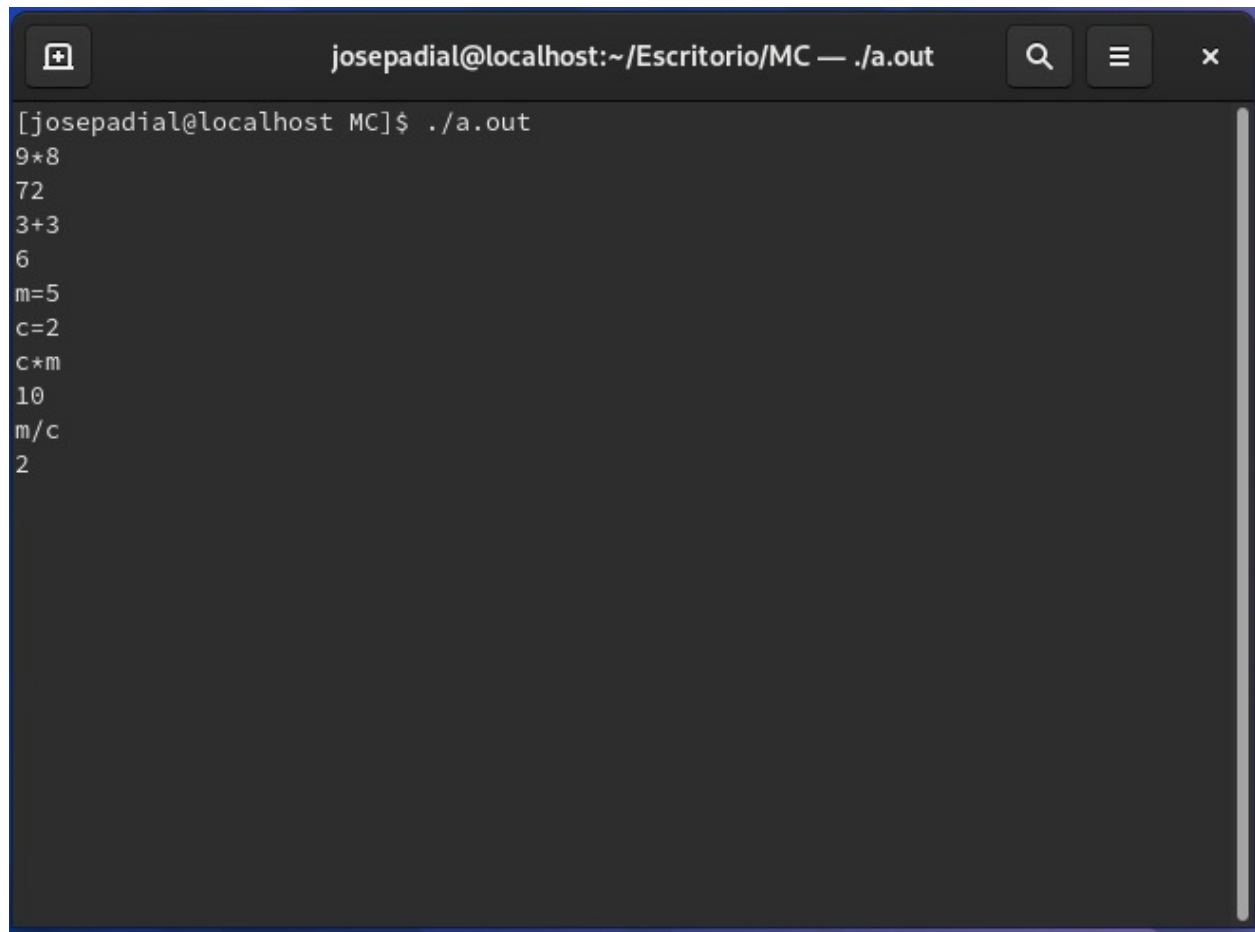
yywrap()
{
    return(1);
}

```

4 Compilar el programa

1. yacc -d calc.yacc
2. lex calc.lex
3. cc y.tab.c lex.yy.c
4. ./a.out

5 Capturas de pantalla



A screenshot of a terminal window. The title bar at the top reads "josepadial@localhost:~/Escritorio/MC — ./a.out". The terminal content shows the command `./a.out` being executed, followed by several lines of output: `9*8`, `72`, `3+3`, `6`, `m=5`, `c=2`, `c*m`, `10`, `m/c`, and `2`. The terminal has a dark background and a light-colored scrollbar on the right side.

```
josepadial@localhost:~/Escritorio/MC — ./a.out
[josepadial@localhost MC]$ ./a.out
9*8
72
3+3
6
m=5
c=2
c*m
10
m/c
2
```

ETSIIT

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



Práctica 4

Modelos de Computación

Jose Antonio Padial Molina

josepadial@correo.ugr.es

December 4, 2019

Contents

1	Ejercicio 1
---	-------------

2

I Ejercicio I

Usar JFlap para demostrar que una gramática libre de contexto es ambigua

1. Gramatica

Table Text Size		
LHS		RHS
S	→	1S1
S	→	0S0
S	→	1
S	→	0
S	→	λ

2. Derivation table y arbol de entrada 1

Start Pause Step Derivation Table

Input: 1001100110011001
String accepted! 15 nodes generated.

Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing the text field)

Table Text Size

LHS	RHS
S	→ 1S1
S	→ 0S0
S	→ 1
S	→ 0
S	→ λ

Table Text Size

S	S
1S1	1S1
10S01	10S01
100S001	100S001
1001S1001	1001S1001
10011S11001	10011S11001
100110S011001	100110S011001
1001100S0011001	1001100S0011001
10011001S10011001	10011001S10011001
1001100110011001	1001100110011001

Derived λ from S. Derivations complete.

Start Pause Step Noninverted Tree

Input: 1001100110011001
String accepted! 15 nodes generated.

Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing the text field)

Table Text Size

LHS	RHS
S	→ 1S1
S	→ 0S0
S	→ 1
S	→ 0
S	→ λ

Derived λ from S. Derivations complete.

3. Derivation table y arbol de entrada 1

Start Pause Step Derivation Table

Input 1001100110011001
String accepted! 15 nodes generated.

Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing the text field)

Table Text Size

LHS		RHS
S	→	1S1
S	→	0
S	→	1
S	→	0S0
S	→	λ

Table Text Size

S
1S1
10S01
100S001
1001S1001
10011S11001
100110S011001
1001100S0011001
10011001S10011001
1001100110011001

Derived λ from S. Derivations complete.

Start Pause Step Noninverted Tree

Input 1001100110011001
String accepted! 15 nodes generated.

Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing the text field)

Table Text Size

LHS		RHS
S	→	1S1
S	→	0
S	→	1
S	→	0S0
S	→	λ

Derived λ from S. Derivations complete.

4. SLR

Do Selected Do Step Do All Next Parse

Table Text Size

S' → S
S → 1S1
S → 0S0
S → 1
S → 0
S → λ

Parse table complete. Press "parse" to use it.

	FIRST	FOLLOW
S	{0, 1, λ}	{0, 1, \$}

	0	1	\$	S
0	r5	r5	r5	3
1	r4	r4	r4	4
2	r3	r3	r3	5
3				
4	s5		acc	
5		s7		
6	r2	r2	r2	
7	r1	r1	r1	

Table Text Size

5. Chomsky

Table Text Size

LHS RHS

S → 1S1
S → 0S0
S → 1
S → 0
S → 11
S → 00

Convert Selected Do All What's Left? Export

All productions completed.

Table Text Size

Conversion done. Press "Export" to use.

LHS RHS

S → B(1)D(2)
D(2) → SB(1)
S → B(0)D(1)
D(1) → SB(0)
S → 1
S → 0
S → B(1)B(1)
B(1) → 1
S → B(0)B(0)
B(0) → 0