

Práctica 1. Simulación de gramáticas y autómatas con JFLAP

El primer lenguaje seleccionado es

$$L = \{a^i b^j a^i b^i \mid i, j \in \mathbb{N}\}$$

Cuya gramática es:

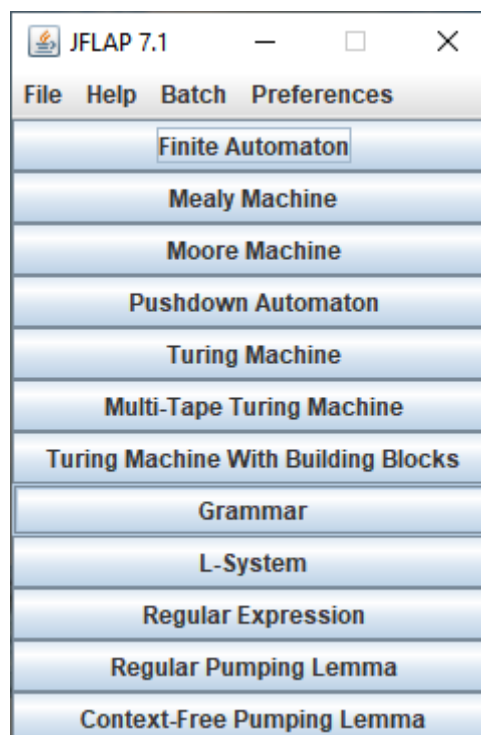
$$S \rightarrow aSb,$$

$$S \rightarrow B,$$

$$B \rightarrow bBa,$$

$$B \rightarrow \epsilon$$

Abrimos JFlap y seleccionamos la opción de 'Grammar'



Introducimos las cadenas necesarias

JFLAP : <untitled2>

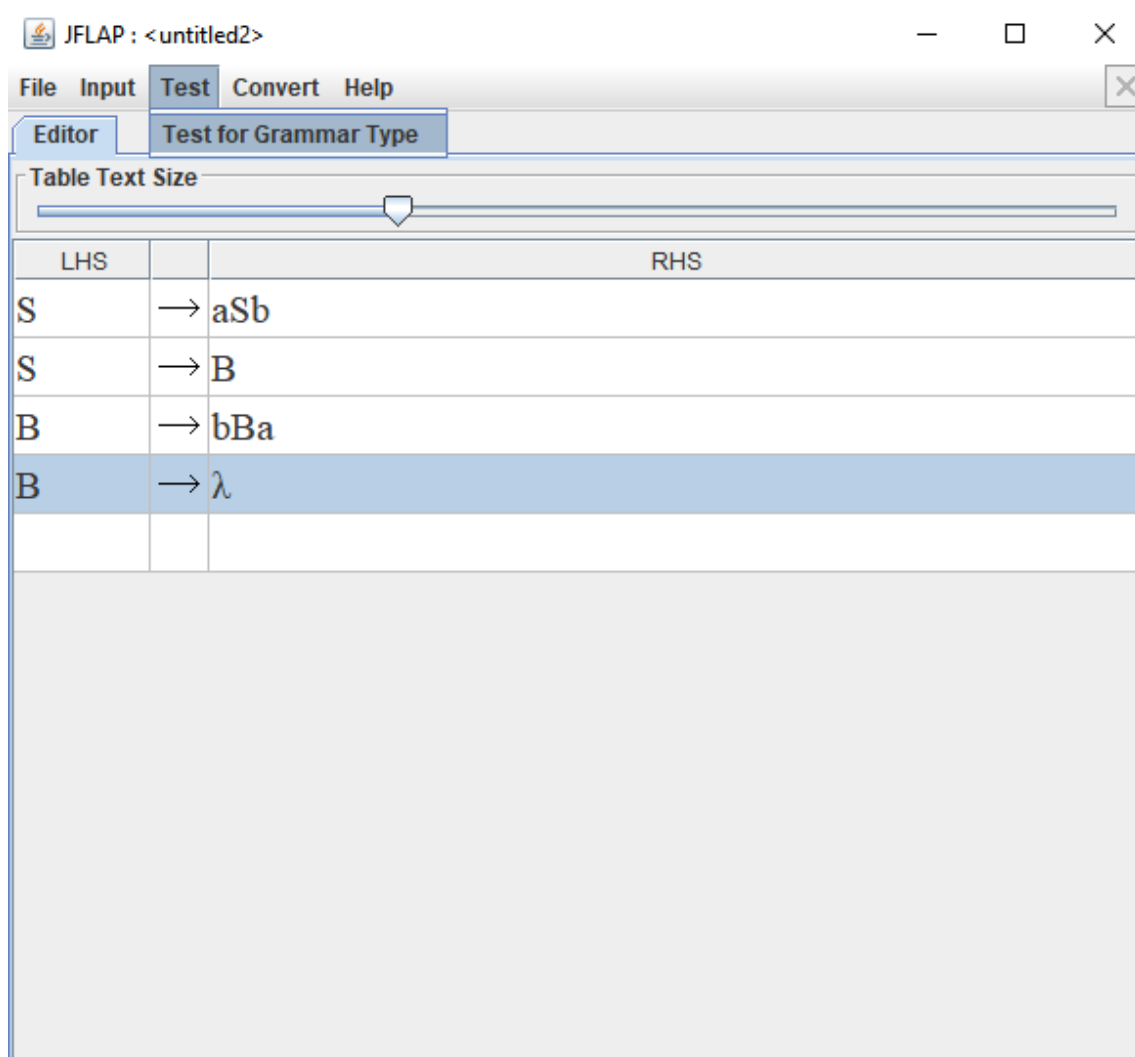
File Input Test Convert Help

Editor

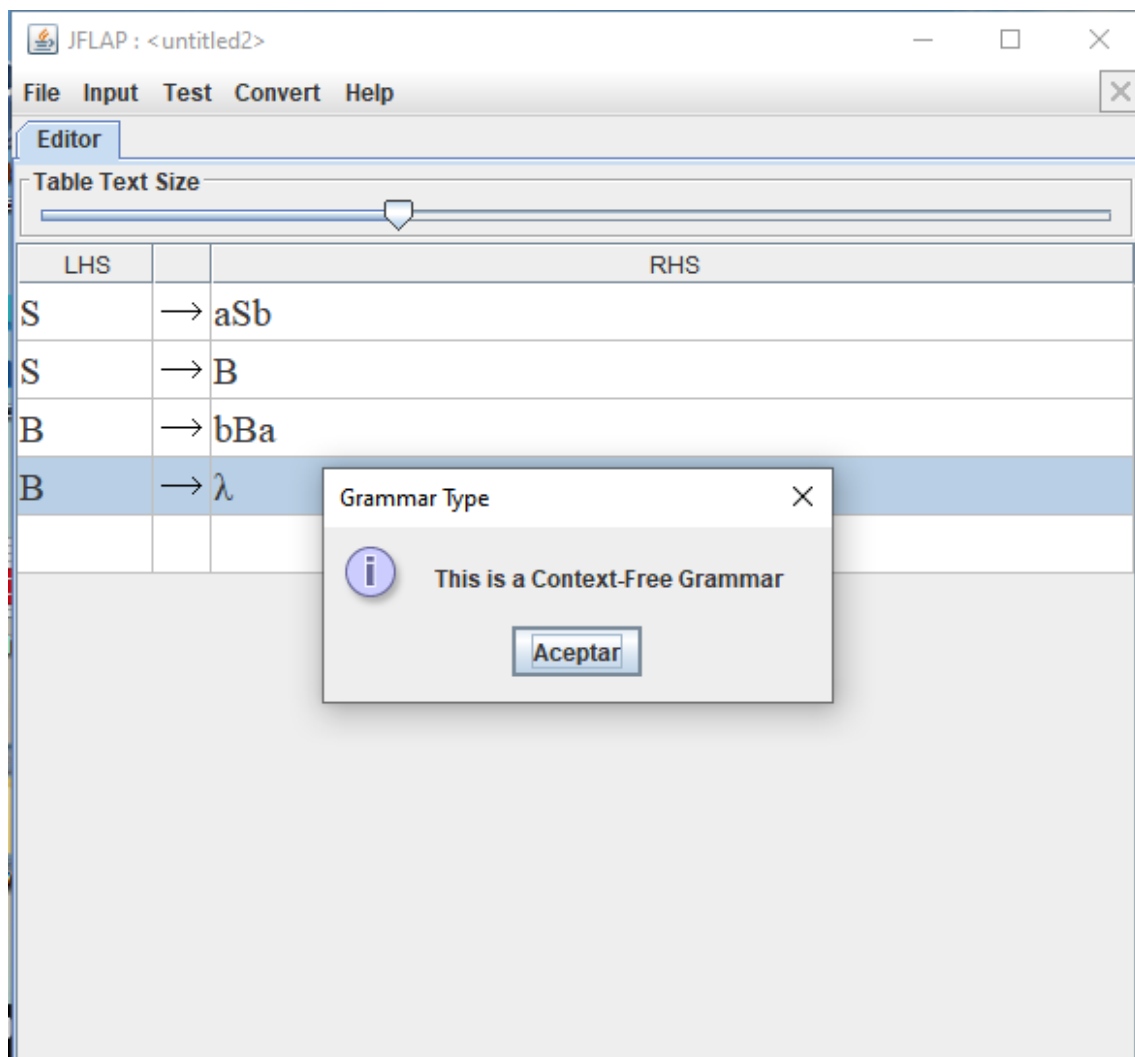
Table Text Size

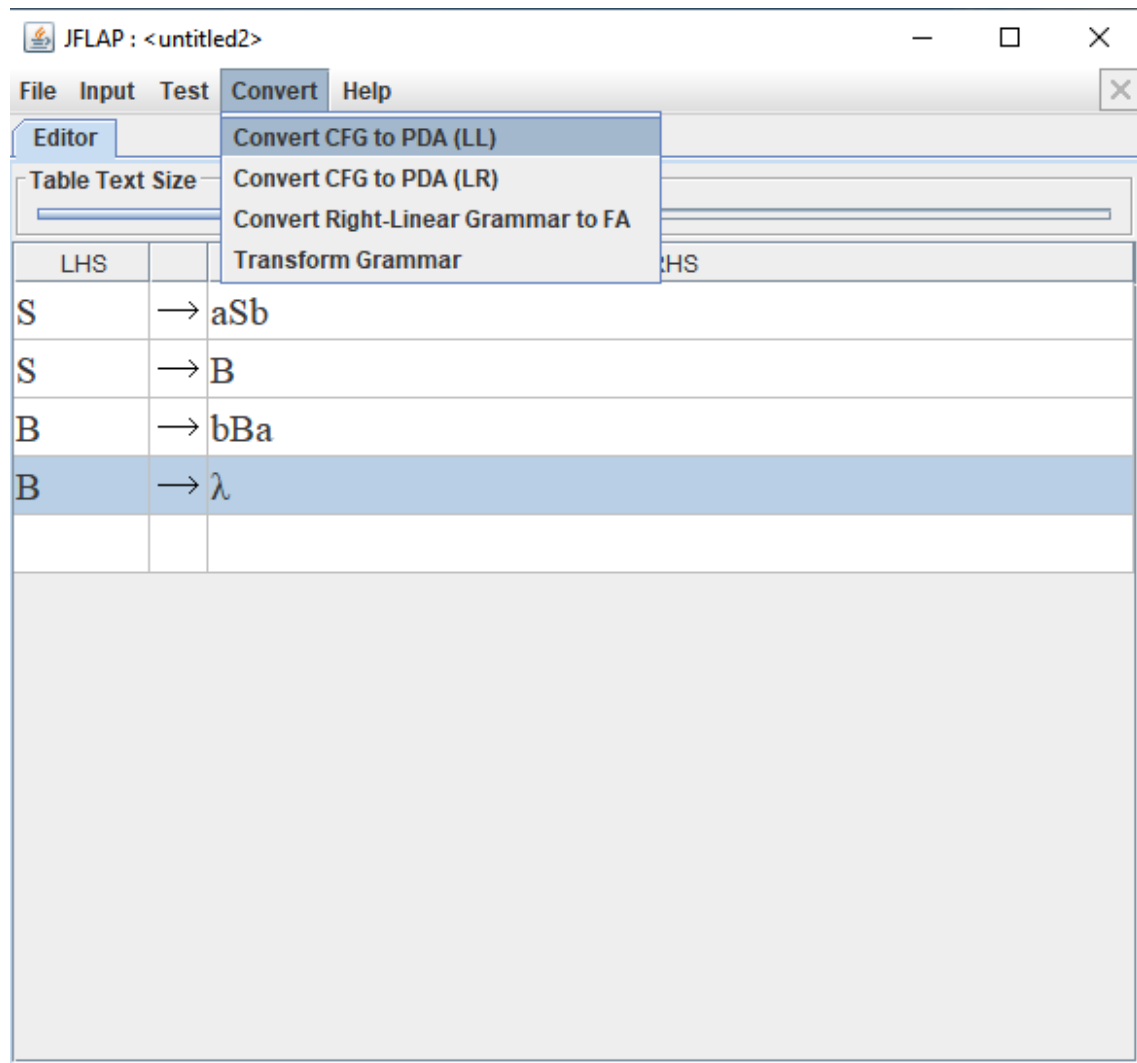
LHS		RHS
S	→	aSb
S	→	B
B	→	bBa
B	→	λ

Seleccionamos probar el tipo de gramática.



Y vemos que estamos ante una gramática libre del contexto.





JFLAP : <untitled2>

File Input Test Convert Help

Editor Convert to PDA (LL)

Table Text Size

Production	Created
$S \rightarrow aSb$	<input checked="" type="checkbox"/>
$S \rightarrow B$	<input checked="" type="checkbox"/>
$B \rightarrow bBa$	<input checked="" type="checkbox"/>
$B \rightarrow \lambda$	<input type="checkbox"/>

q0

$\lambda, B; bBa$

$\lambda, S; B$

$\lambda, S; aSb$

$b, B; \lambda$

$a, a; \lambda$

q1

$\lambda, Z; \lambda$

q2

Automaton Size

JFLAP : <untitled2>

File Input Test Convert Help

Editor Convert to PDA (LL)

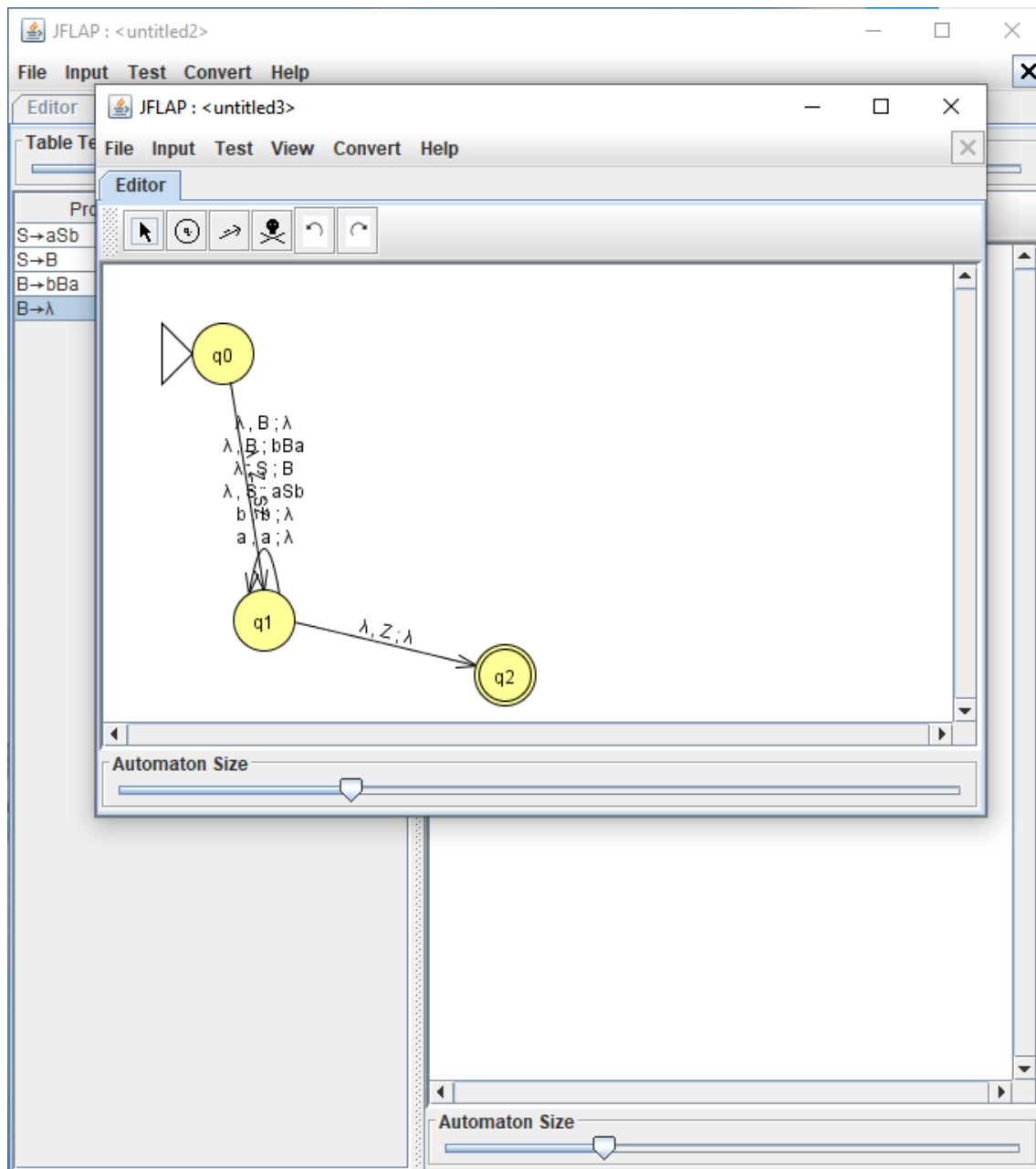
Table Text Size

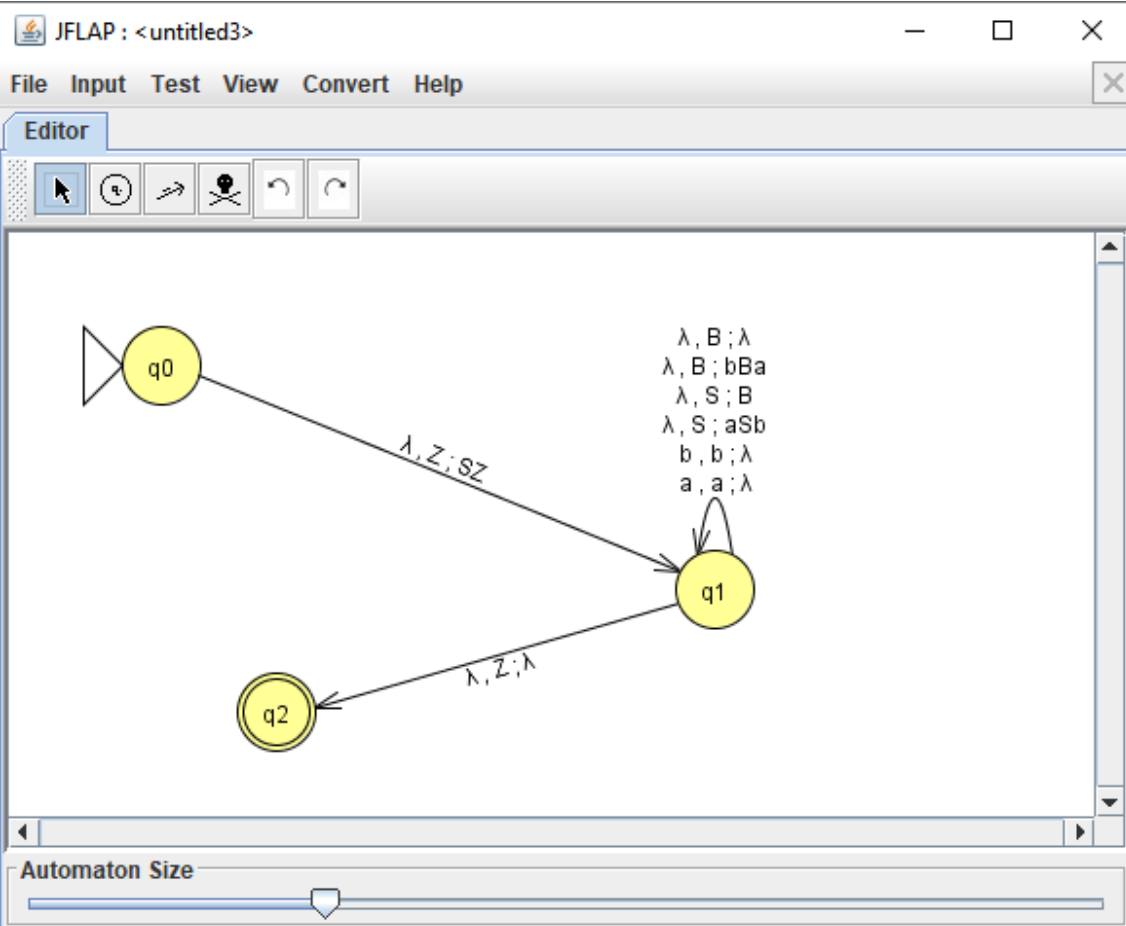
Production	Created
$S \rightarrow aSb$	<input checked="" type="checkbox"/>
$S \rightarrow B$	<input checked="" type="checkbox"/>
$B \rightarrow bBa$	<input checked="" type="checkbox"/>
$B \rightarrow \lambda$	<input checked="" type="checkbox"/>

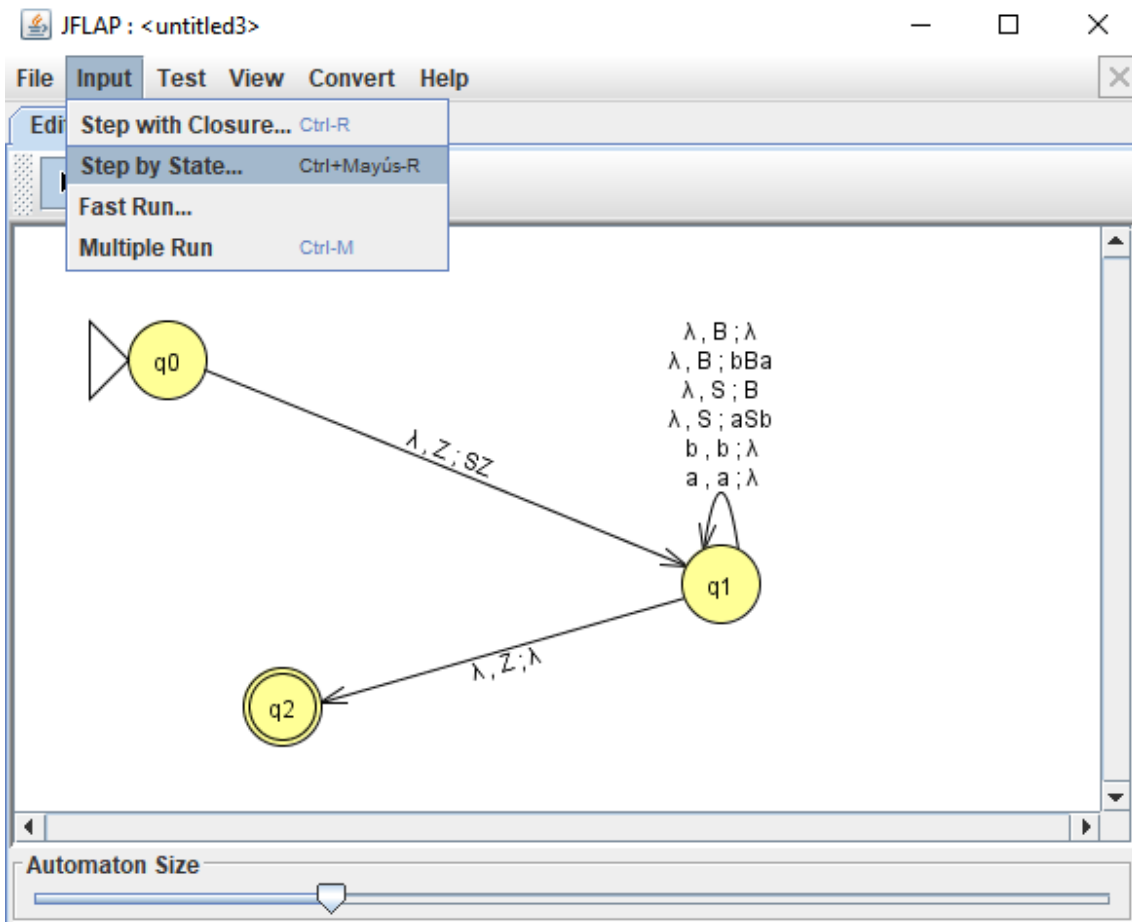
Automaton Diagram:

```
graph TD; Start(( )) --> q0((q0)); q0 -- "λ, B; bBa" --> q1((q1)); q1 -- "λ, B; B" --> q0; q1 -- "λ, a; aSb" --> q1; q1 -- "b, a; λ" --> q1; q1 -- "λ, Z; λ" --> q2(((q2))); style Start fill:none,stroke:none; style q0 fill:#ffff00,stroke:#000,stroke-width:1px; style q1 fill:#ffff00,stroke:#000,stroke-width:1px; style q2 fill:#ffff00,stroke:#000,stroke-width:1px,stroke-dasharray: 5 5;
```

Automaton Size







Input

Click to Open Input File

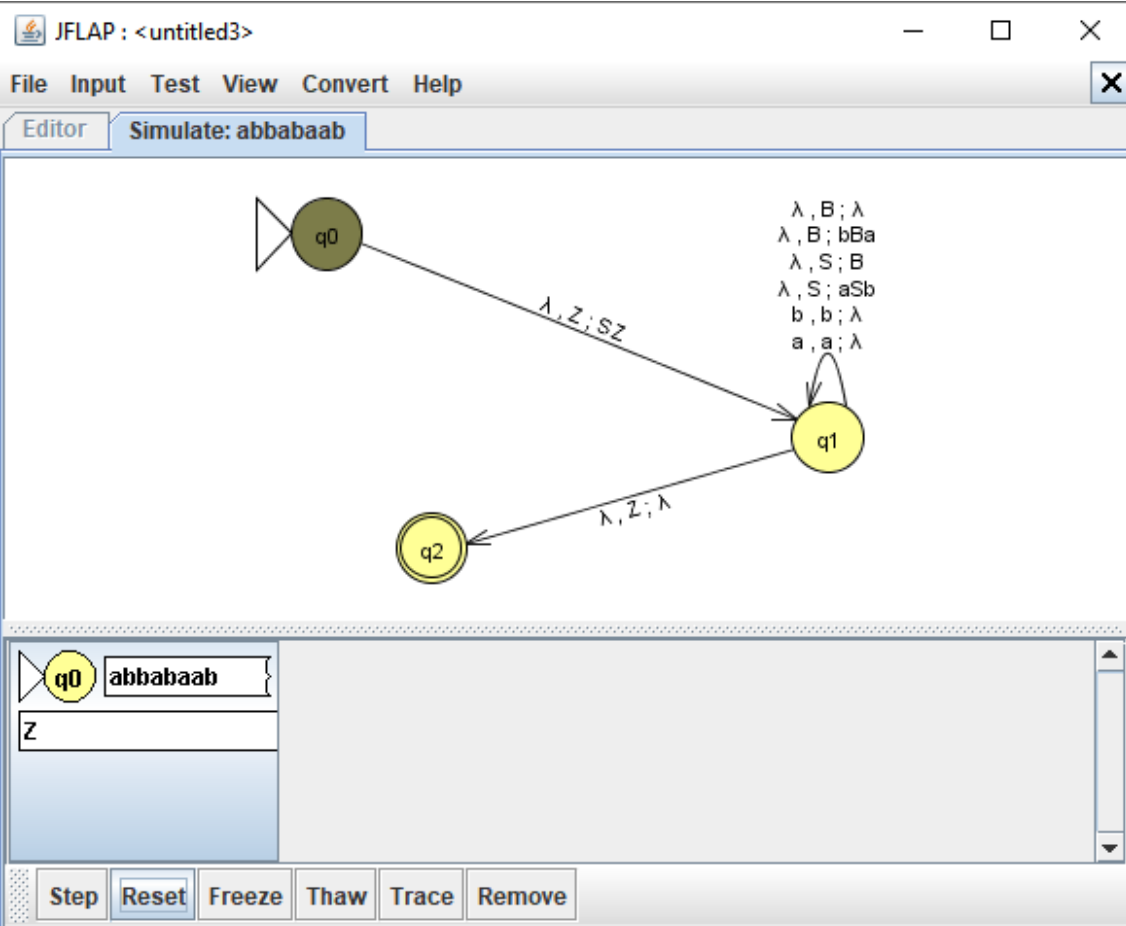
Aceptar Cancelar

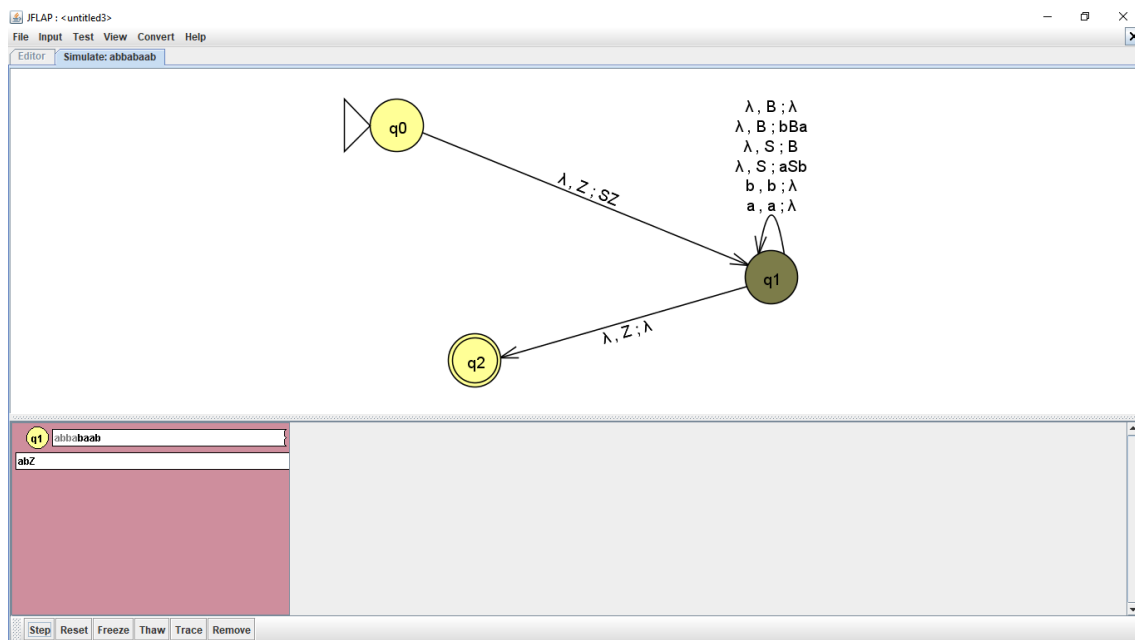
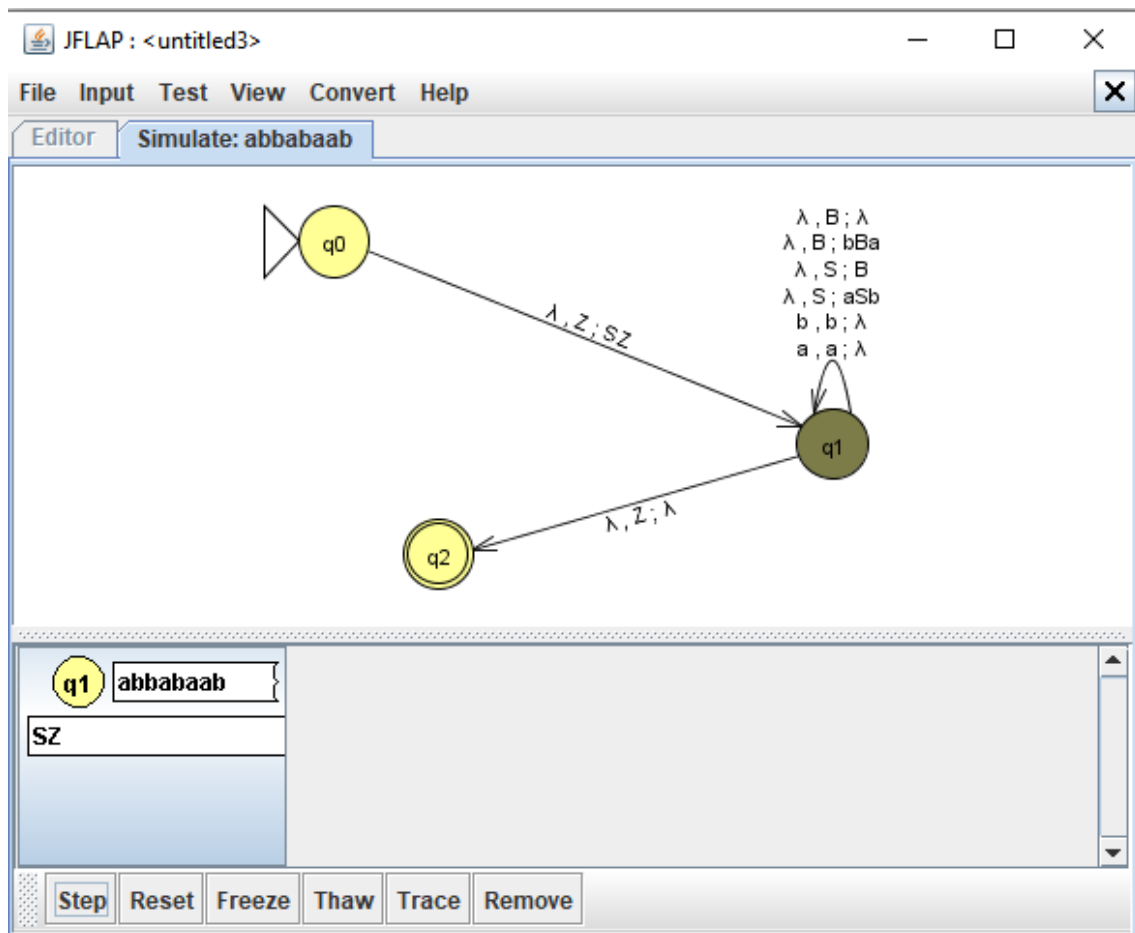
Input

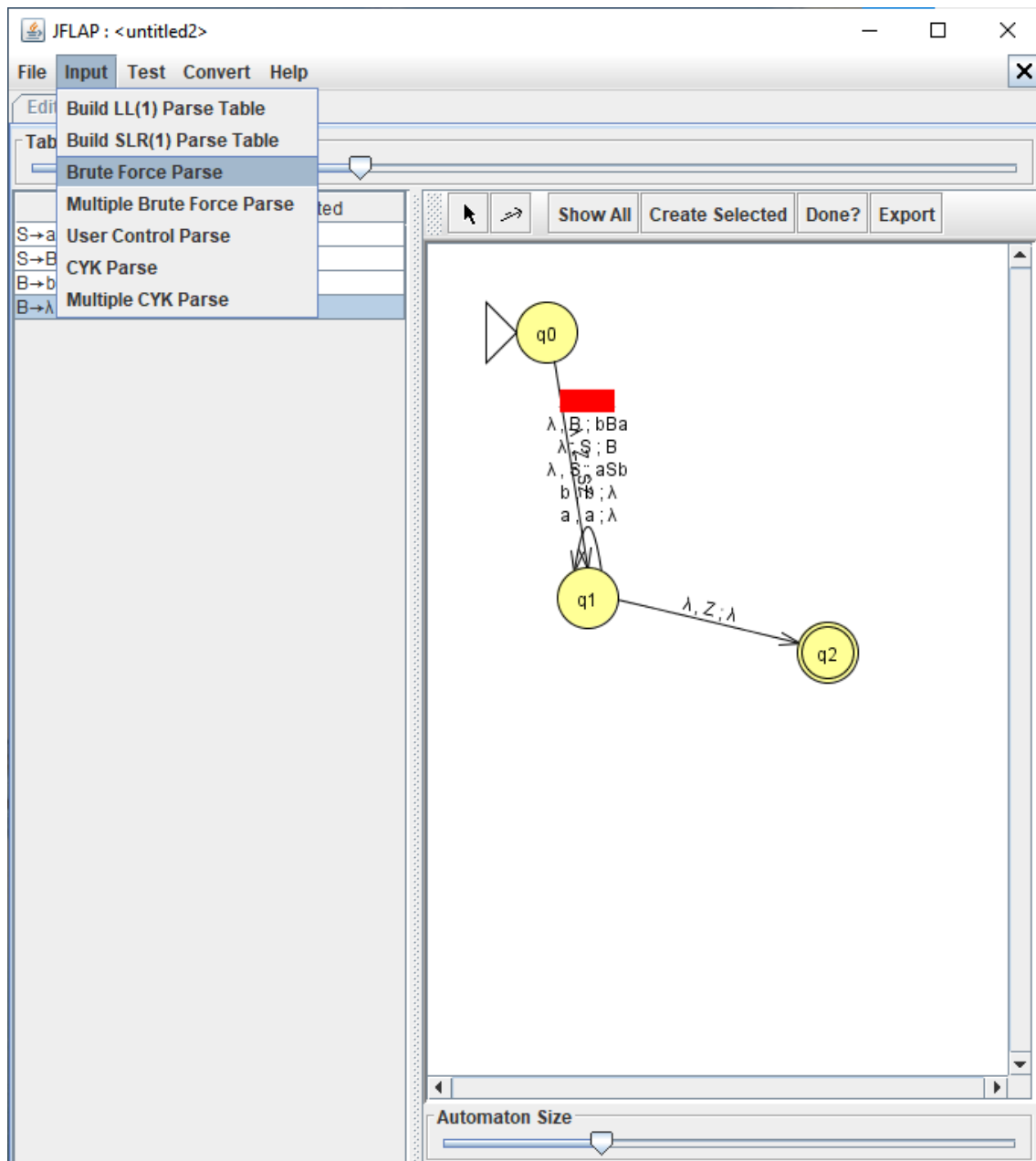
Accept by

Final State

Aceptar Cancelar







JFLAP : <untitled2>

FileInputTestConvertHelp

EditorConvert to PDA (LL)Brute Parser

StartPauseStepNoninverted Tree

Inputaaaabbbbaaabb

Input Field Text Size (For optimization, move one of the window size adjusters around this window after resizing t...

Table Text Size

LHS		RHS
S	→	aSb
S	→	B
B	→	bBa
B	→	λ

Input a string to begin.

JFLAP : <untitled2>

File Input Test Convert Help

Editor Convert to PDA (LL) Brute Parser

Start Pause Step Noninverted Tree

Input: aaaabbbbaaabb
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	→	aSb
S	→	B
B	→	bBa
B	→	λ

Derived A from B. Derivations complete.

JFLAP : <untitled2>

File Input Test Convert Help

Editor Convert to PDA (LL) Brute Parser

Start Pause Step Derivation Table

Input: aaaabbbbaaabb
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	→	aSb
S	→	B
B	→	bBa
B	→	λ

Table Text Size	
S → aSb	S
S → aSb	aSb
S → aSb	aaSb
S → aSb	aaaSbbb
S → aSb	aaaaSbbbb
S → B	aaaaBbbbb
B → bBa	aaaabBabbbb
B → bBa	aaaabbBaabbbb
B → bBa	aaaabbbBaabbbb
B → λ	aaaabbbbaaabb

Derived A from B. Derivations complete.

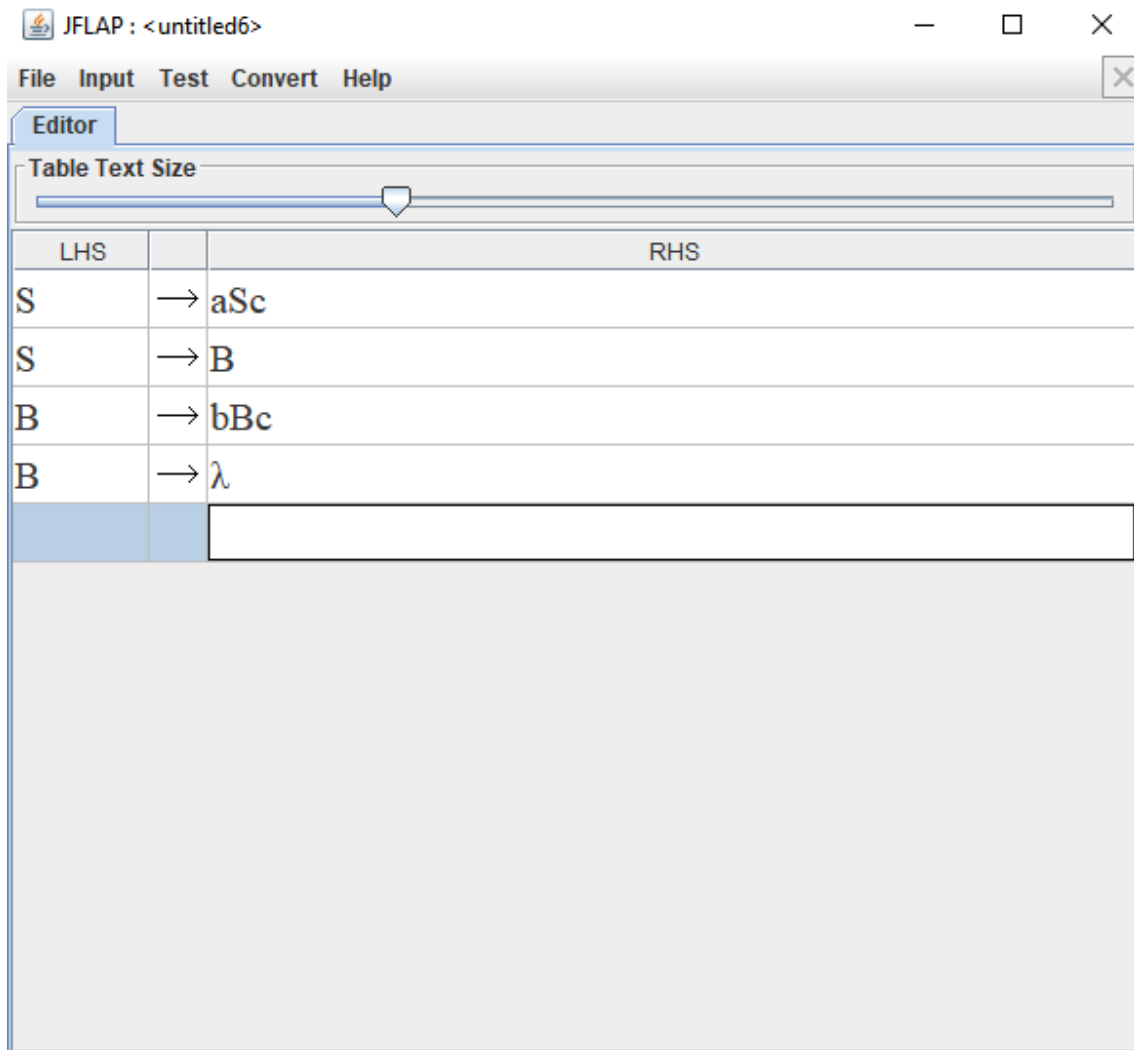
Para la segunda prueba el lenguaje seleccionado es el siguiente:

$$L = \{a^i b^j c^{i+j} \mid i, j \in \mathbb{N}\}$$

Con su gramática:

$$S \rightarrow aSc, \quad S \rightarrow B, \quad B \rightarrow bBc, \quad B \rightarrow \epsilon$$

Y el procedimiento para probarlo es similar al primero de los tres ya visto.



JFLAP : <untitled6>


File Input Test Convert Help

Editor

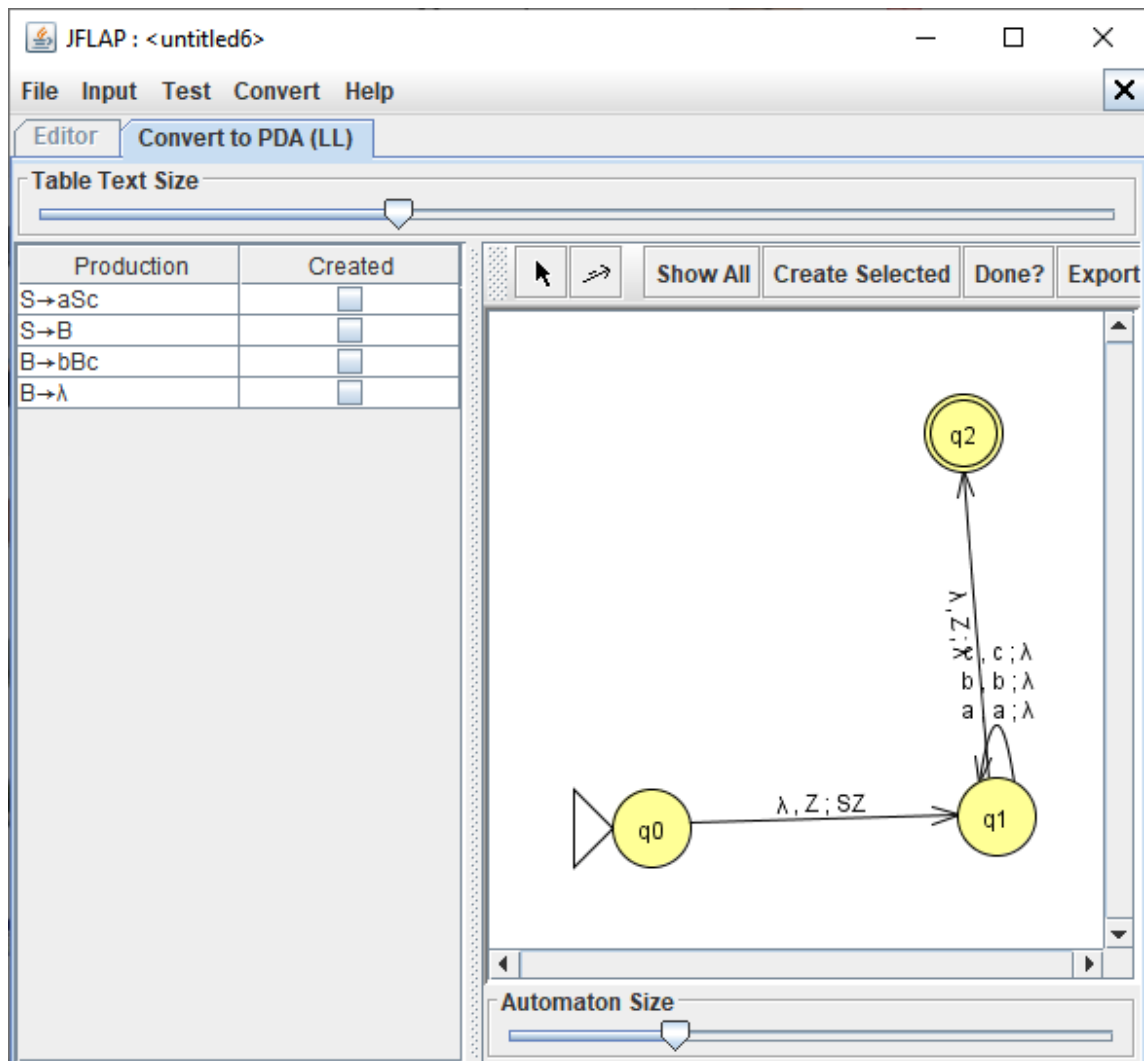
Table Text Size

LHS		RHS
S	→	aSc
S	→	B
B	→	bBc
B	→	λ
	→	

Grammar Type

 This is a Context-Free Grammar

Acceptar



JFLAP : <untitled6>

File Input Test Convert Help

Editor

Convert to PDA (LL)

Table Text Size

Production	Created
$S \rightarrow aSc$	<input checked="" type="checkbox"/>
$S \rightarrow B$	<input checked="" type="checkbox"/>
$B \rightarrow bBc$	<input checked="" type="checkbox"/>
$B \rightarrow \lambda$	<input checked="" type="checkbox"/>

Show All

Create Selected

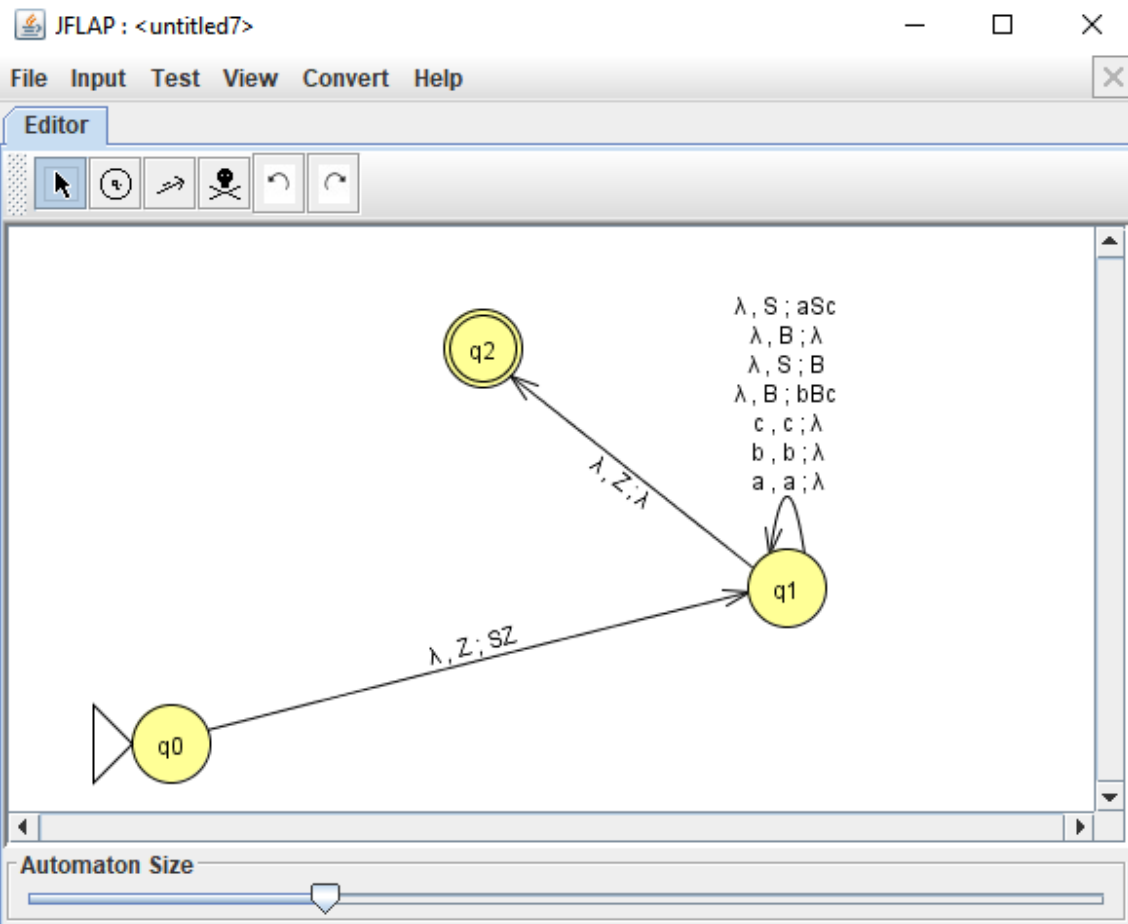
Done?

Export

```

graph LR
    q0((q0)) -- "λ, Z; SZ" --> q1((q1))
    q1 -- "λ, S; aSc" --> q2(((q2)))
    q1 -- "λ, B; λ" --> q2
    q1 -- "λ, S; B" --> q1
    q1 -- "λ, B; bBc" --> q1
    q1 -- "c, c; λ" --> q1
    q1 -- "b, b; λ" --> q1
    q1 -- "a, a; λ" --> q1
  
```

Automaton Size



Input

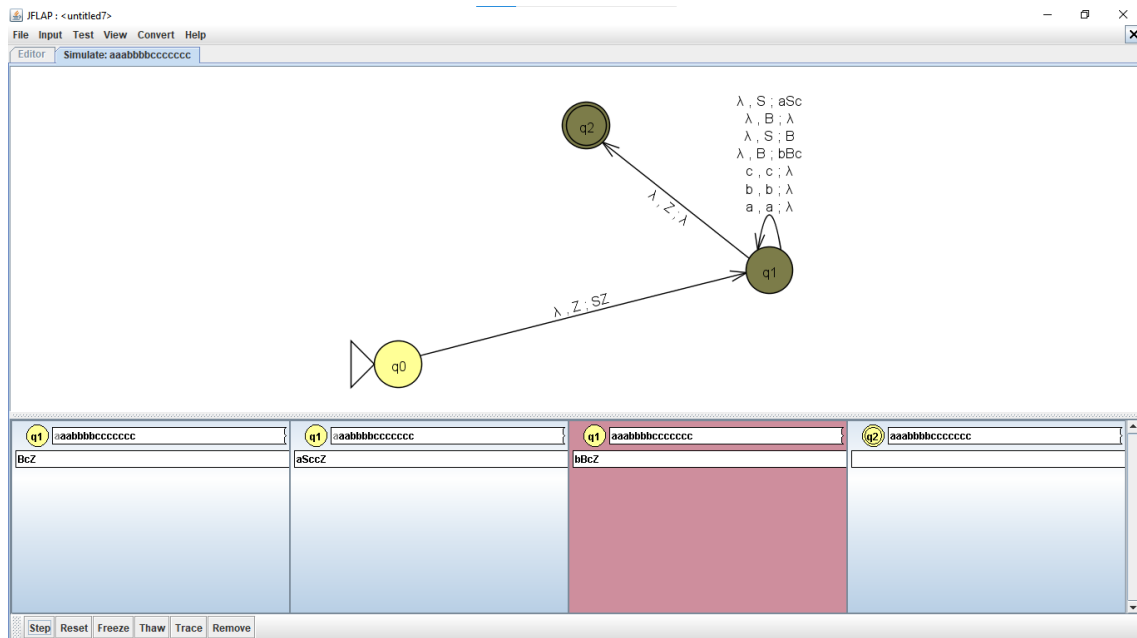
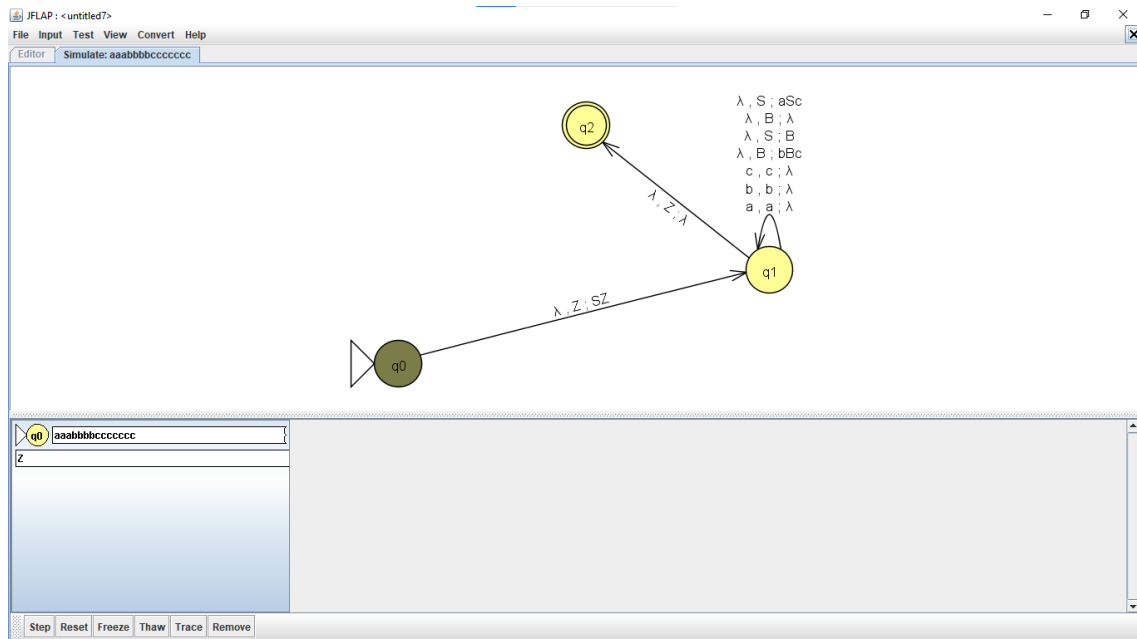
?

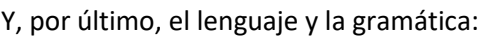
Input

aaabbbbccccccq

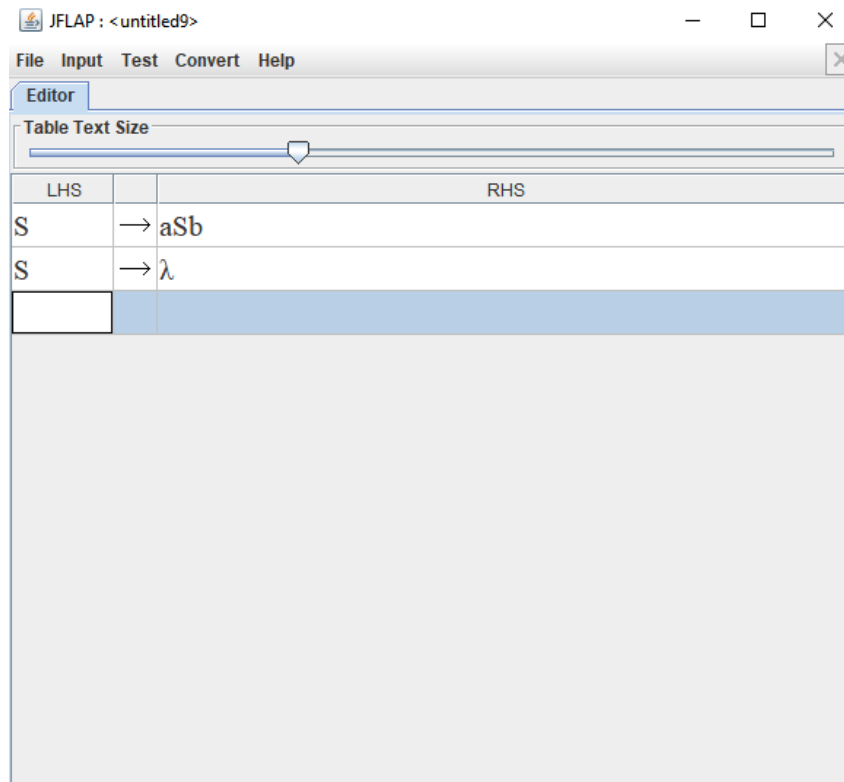
Click to Open Input File

Aceptar Cancelar





$$S \rightarrow aSb, \quad S \rightarrow \epsilon$$



JFLAP : <untitled9>


File Input Test Convert Help

Editor

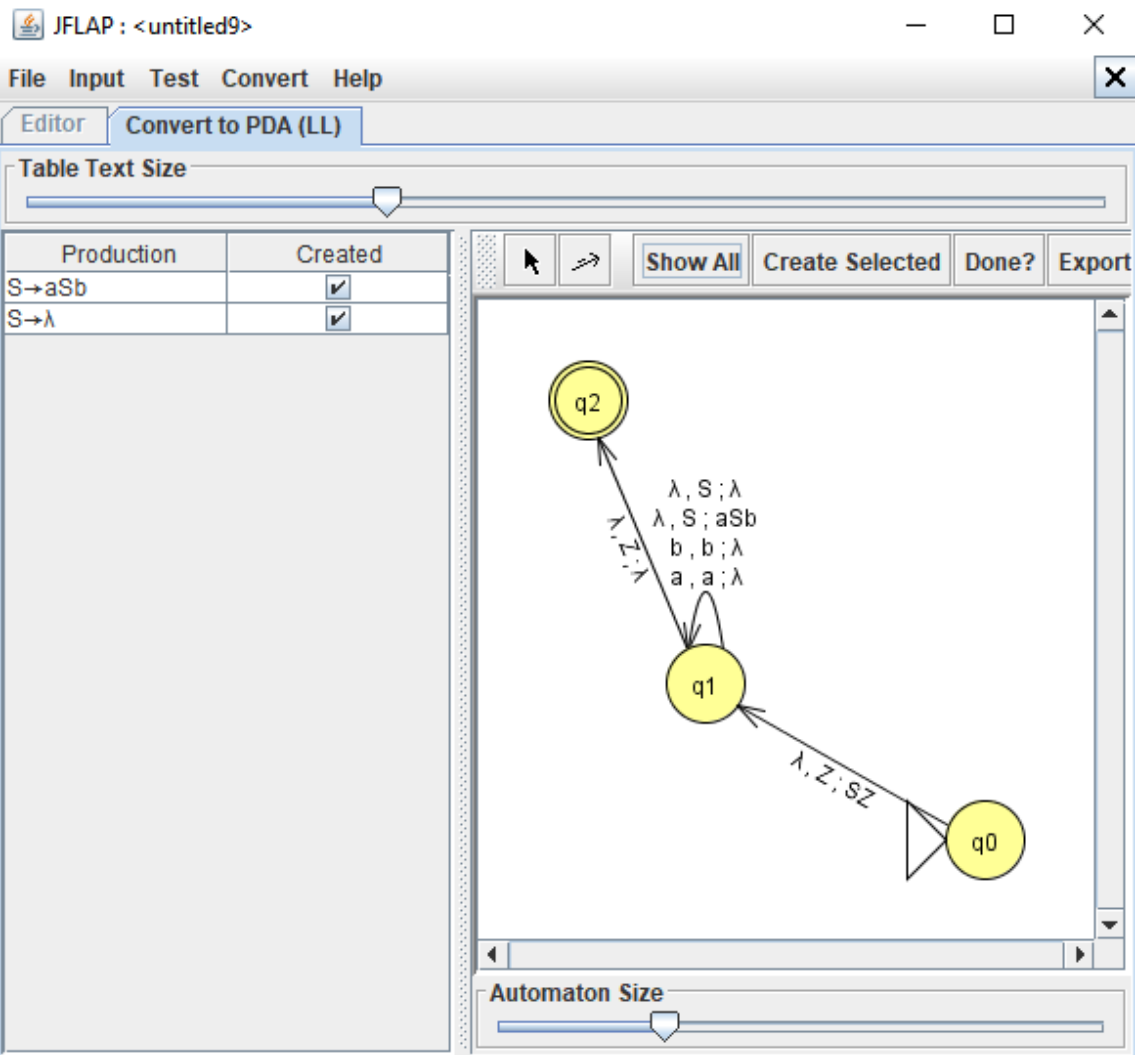
Table Text Size

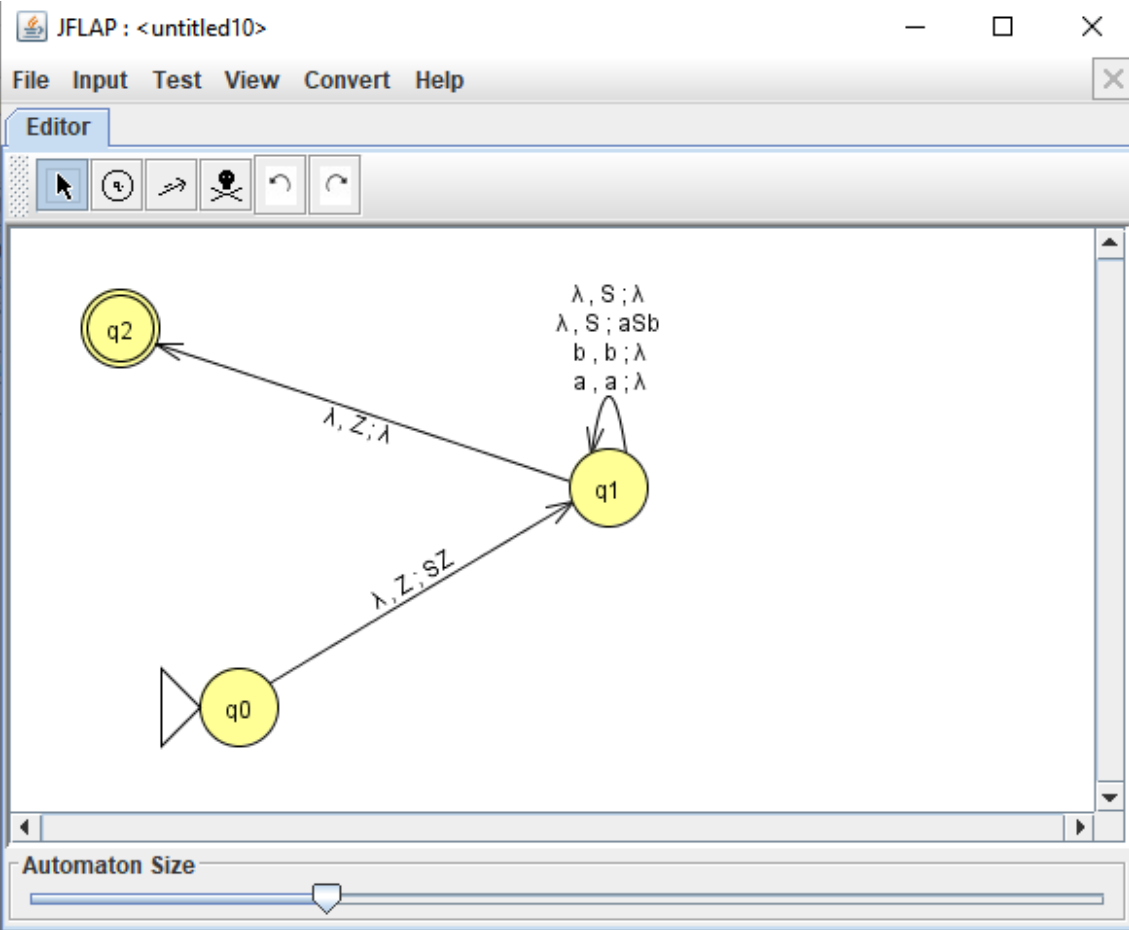
LHS		RHS
S	→	aSb
S	→	λ
	→	

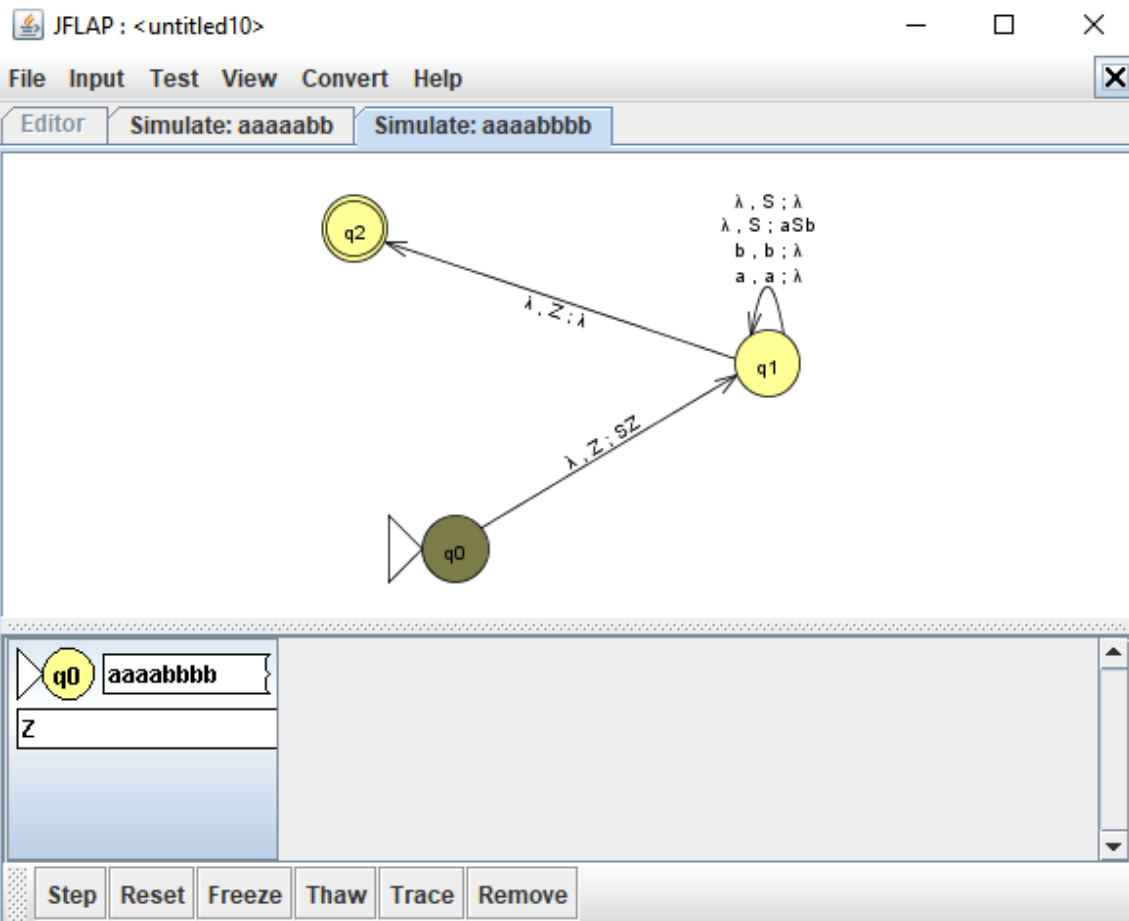
Grammar Type

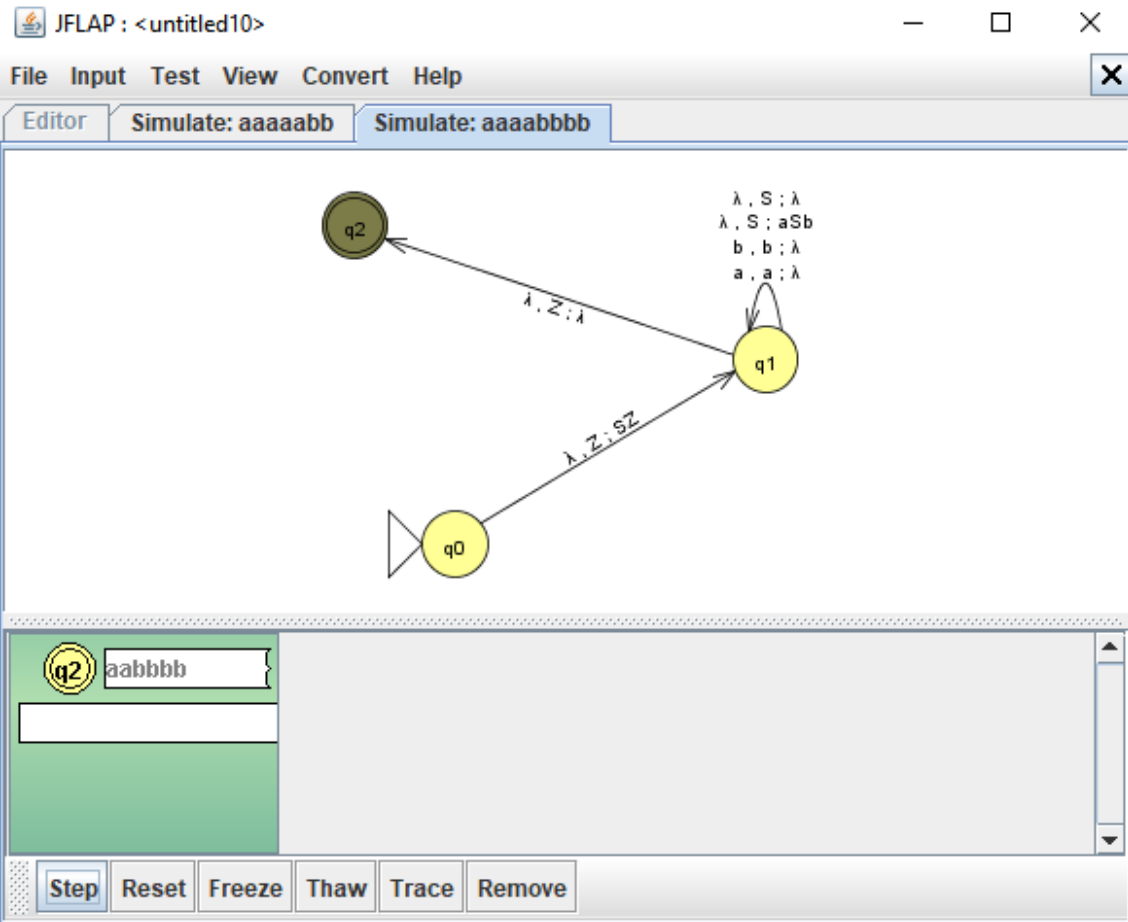
 This is a Context-Free Grammar

Acceptar









JFLAP : <untitled9>

File Input Test Convert Help

Editor Convert to PDA (LL) Brute Parser

Start Pause Step Noninverted Tree

Input(aabbbb)
String rejected. 6 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	→ aSb
S	→ λ

Try another string.

Práctica 2. Analizador léxico (FLEX)

El primer ejemplo que comentaré será uno dado en clase. Comentaré qué significa cada definición, cómo ejecutarlo y compilarlo y por último un ejemplo de su funcionamiento.

```
1  letra      [a-zA-Z]
2  num        [0-9]
3  letra_num  [a-zA-Z.0-9\_ ]
4  signo      (\+)
5  telefono_m ({signo}?{num}){11}
6  telefono_d ({num}){9}
7  dominio    (\@({letra}+\.){1,2})
8  correo     ({letra_num}+{dominio}){letra}{2,}
9  mejores_series ("Black Mirror"|"Mr Robot"|"La casa de papel"|"Lucifer")
10 mejor_asig ("MC"|"ED")
11
12 %%
13 {telefono_m}      {printf("\nTelefono valido %s\n", yytext);}
14 {telefono_d}      {printf("\nTelefono valido %s\n", yytext);}
15 {correo}           {printf("\nCorreo valido %s\n", yytext);}
16 {mejores_series}  {printf("\nMejor serie valida %s\n", yytext);}
17 {mejor_asig}       {printf("\nMejor asignatura valida %s\n", yytext);}
18 .+                {printf("\nIncorrecto %s\n", yytext);}
19 %%
20
21 int yywrap(){
22     return 1;
23 }
24
25 int main()
26 {
27     yyin = fopen("texto2.txt","r");
28     yylex();
29     return 1;
30 }
```

Aquí el texto como tal por si quiere ser copiado y pegado en un archivo .l para su prueba.

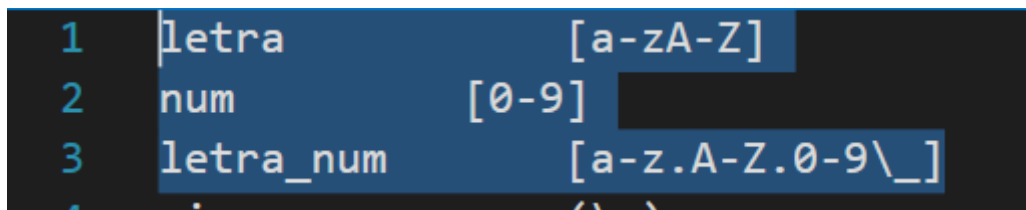
```
letra      [a-zA-Z]
num        [0-9]
letra_num  [a-zA-Z.0-9\_ ]
signo      (\+)
telefono_m ({signo}?{num}{11})
telefono_d ({num}{9})
dominio     (\@({letra}+\.){1,2})
correo      ({letra_num}+{dominio}{letra}{2,})
mejores_series ("Black Mirror"|"Mr Robot"|"La casa de papel"|"Lucifer")
mejor_asig   ("MC"|"ED")

%%
{telefono_m}      {printf("\nTelefono valido %s\n", yytext);}
{telefono_d}      {printf("\nTelefono valido %s\n", yytext);}
{correo}           {printf("\nCorreo valido %s\n", yytext);}
{mejores_series}  {printf("\nMejor serie valida %s\n", yytext);}
{mejor_asig}       {printf("\nMejor asignatura valida %s\n", yytext);}
.+
%%

int yywrap(){
return 1;
}

int main()
{
yyin = fopen("texto2.txt","r");
yylex();
return 1;
}
```

En la primera sección se definen las expresiones regulares. La primera es “letra” que se refiere a cualquier carácter de la ‘a’ a la ‘z’ en mayúscula o minúscula, la segunda es “num” que es cualquier dígito del 0 al 9 y el tercero es “letra_num” que combina tener cualquier letra en mayúscula o minúscula, cualquier dígito del 0 al 9 y el guion bajo (‘_’).



```
1 letra      [a-zA-Z]
2 num        [0-9]
3 letra_num  [a-zA-Z.0-9\_ ]
```

“signo” es la expresión regular que contiene el símbolo ‘+’ solamente. Nuestro objetivo es conocer si los números de teléfonos que metamos son correctos o no según la definición de expresión regular que nosotros demos. En este caso, definimos la estructura léxica correcta en “telefono_m” (teléfono móvil) y “telefono_d”. El primero de ellos contiene un signo (invocamos “signo” que contiene el símbolo ‘+’) que puede estar o no, debido al signo de interrogación que tiene detrás, y luego un número que está repetido 11 veces (por ejemplo: +34 658412541). Por otra parte, “telefono_d” es igual pero sin prefijo, por lo que tiene un número repetido 9 veces.


```

3 letra_num      [a-z.A-Z.0-9\_ ]
4 signo          (\+)
5 telefono_m      ({signo}?{num}{11})
6 telefono_d      ({num}{9})
7 dominio         (\@({letra}+\.){1,2})

```

Definimos el correo electrónico de una forma elemental. Primero ponemos “dominio” para que sea invocado desde “correo” y lo que especificamos es un literal arroba y después, concatenado, “letra” cualquier número de veces (mayor que uno, por ello el símbolo más) y un punto. Esto se puede repetir 1 o 2 veces para que puedas poner en el dominio varias secciones entre puntos (tanto “@hotmail.” como “@correo.ugr.” serían válidos). Y tras esto, definimos el “correo” como letra o número cualquier número de veces mayor que 1, seguido de un “dominio” previamente definido y tras el punto “letra” 2 veces o más (“es”, “com”, “org”, etc.).

```

7 dominio         (\@({letra}+\.){1,2})
8 correo          ({letra_num}+{dominio}{letra}{2,})
9 mejores_series  ("Black Mirror"|"Mr Robot"|"La casa de papel"|"Lucifer")

```

Estas dos expresiones regulares se refieren a la mejor serie y la mejor asignatura. El contenido va entrecomillado porque son palabras o expresiones literales, separadas por una “|” que significa unión.

```

mejores_series    ("Black Mirror"|"Mr Robot"|"La casa de papel"|"Lucifer")
mejor_asig        ("MC"|"ED")

```

```

11
12 %%

```

Este símbolo significa el cambio de sección y pasamos a la segunda de ellas.

En esta sección ponemos la expresión regular (por ejemplo “{teléfono_m}”) y a la derecha la acción a ejecutar cuando encontremos una ocurrencia de esa expresión. Así, si por ejemplo encuentra un teléfono móvil válido mostrará un mensaje que lo confirme seguido del número en sí, almacenado en “yytext” que es una cadena de caracteres.

```

3 {telefono_m}      {printf("\nTelefono valido %s\n", yytext);}
4 {telefono_d}      {printf("\nTelefono valido %s\n", yytext);}
5 {correo}          {printf("\nCorreo valido %s\n", yytext);}
6 {mejores_series}  {printf("\nMejor serie valida %s\n", yytext);}
7 {mejor_asig}      {printf("\nMejor asignatura valida %s\n", yytext);}

```

Tras estas líneas ponemos “.+” para el caso en el que encuentre algo incorrecto. Si la cadena que lee no se corresponde a ninguna expresión regular de las especificadas en la segunda sección, saldrá ese mensaje de error seguido de la misma cadena que almacena.

```
{telefono_m}      {printf("\nTelefono valido %s\n", yytext);}
{telefono_d}      {printf("\nTelefono valido %s\n", yytext);}
{correo}          {printf("\nCorreo valido %s\n", yytext);}
{mejores_series}  {printf("\nMejor serie valida %s\n", yytext);}
{mejor_asig}      {printf("\nMejor asignatura valida %s\n", yytext);}
.+               {printf("\nIncorrecto %s\n", yytext);}
%%
```

En la tercera sección está el main donde decimos que llame a yylex() que es la función que busca los patrones. La función yywrap() devuelve 1 cuando encuentra un fin en el fichero o una interrupción forzada, y con eso termina simplemente. De escribir un 0, lo que haría sería seguir leyendo de entrada.

```
%%
/o/o
/o/o

int yywrap(){
    return 1;
}

int main()
{
    yyin = fopen("texto2.txt", "r");
    yylex();
    return 1;
}
```

Se puede usar desde la terminal sin pasarle ningún fichero, pero, como se ve en la imagen anterior, tengo que abrir el archivo de texto “texto2.txt”. A continuación, haré una prueba que muestre el correcto funcionamiento. El contenido del fichero que lee es:

```
1 +34648124025
2 ED
3 javierramirezp@correo.ugr.es
4 632015048
5 mal
6 nada
7 vacio
8 .
```

De las cuales deberían ser erróneas las 4 últimas. En la siguiente imagen se ve el procedimiento de compilación y ejecución, además del resultado sobre el caso de ejemplo ya especificado.

```
PS C:\Users\xaviv\Desktop\JAVIER\UNIVERSIDAD\CURSO 20.21\CURSO\1º CUATRI\MODELOS DE COMPUTACION\PRACTICAS\PRACTICA2\C++> flex .\ejemplo2.1
PS C:\Users\xaviv\Desktop\JAVIER\UNIVERSIDAD\CURSO 20.21\CURSO\1º CUATRI\MODELOS DE COMPUTACION\PRACTICAS\PRACTICA2\C++> gcc lex.yy.c -lfl
PS C:\Users\xaviv\Desktop\JAVIER\UNIVERSIDAD\CURSO 20.21\CURSO\1º CUATRI\MODELOS DE COMPUTACION\PRACTICAS\PRACTICA2\C++> .\a.exe
```

Telefono valido +34648124025

Mejor asignatura valida ED

Correo valido javierramirezp@correo.ugr.es

Telefono valido 632015048

Incorrecto mal

Incorrecto nada

Incorrecto vacio

Incorrecto .

```
PS C:\Users\xaviv\Desktop\JAVIER\UNIVERSIDAD\CURSO 20.21\CURSO\1º CUATRI\MODELOS DE COMPUTACION\PRACTICAS\PRACTICA2\C++>
```

El siguiente ejemplo lo que realizará será un conteo de la cantidad de líneas, caracteres y dígitos que tenga una entrada que hagamos. En mi caso, volveré a utilizar un archivo de entrada, pero antes, vamos a ver qué hay sección por sección. Esta es una imagen general del código:

```
1  %{
2  int numLineas=0, numChar=0, numDig=0;
3  %}
4  %%
5  \n          {++numLineas;}
6  [ \t]       {;}
7  [a-zA-Z]    {++numChar;}
8  [0-9]       {++numDig;}
9
10 %%
11
12 int yywrap(){
13     return 1;
14 }
15
16 int main()
17 {
18     yyin = fopen("texto1.txt","r");
19     yylex();
20     printf("Numero de Caracteres = %d, Numero de Lineas = %d, Numero de Digitos = %d\n", numChar, numLineas, numDig);
21     return 1;
22 }
```

Ahora un cuadro de texto con él para facilitar la copia y prueba del mismo:

```
%{
int numLineas=0, numChar=0, numDig=0;
%}
%%
\n          {++numLineas;}
[ \t]       {;}
[a-zA-Z]    {++numChar;}
[0-9]       {++numDig;}

%%

int yywrap(){
    return 1;
}

int main()
{
    yyin = fopen("texto1.txt","r");
    yylex();
    printf("Numero de Caracteres = %d, Numero de Lineas = %d, Numero de Digitos = %d\n", numChar,
numLineas, numDig);
    return 1;
}
```

Y ahora vayamos por partes. En la primera de ella nos defino expresiones regulares, sino que declaramos 3 variables globales de tipo entero e inicializadas a 0. Estas serán las que me digan la cantidad de caracteres, el numero de líneas y de dígitos.

```
%{
int numLineas=0, numChar=0, numDig=0;
}%
%%
```

En la siguiente sección están definidas las acciones a realizar cuando vemos una de estas expresiones regulares. De esta forma, si lee un salto de línea (“\n”) cuenta como una línea más, si hay un espacio o tabulador (“\t”) que no haga nada (su acción asociado es “{;}”), si lee un carácter en mayúsculas o minúsculas aumenta en uno la cantidad de letras y de ver un dígito cualquiera entre 0 y 9 aumenta la cantidad de números.

```
%%
\n      {++numLineas;}
[ \t]   {;}
[a-zA-Z] {++numChar;}
[0-9]   {++numDig;}

%%
```

Y por último tenemos el main que abre el archivo de entrada “texto2.txt” que es donde estará escrito el contenido de la prueba que vamos a hacer.

```
%%
int yywrap(){
    return 1;
}

int main()
{
    yyin = fopen("texto1.txt","r");
    yylex();
    printf("Numero de Caracteres = %d, Numero de Lineas = %d, Numero de Digitos = %d\n", numChar, numLineas, numDig);
    return 1;
}
```

El archivo de prueba:

```
1 Este es un mensaje de prueba que nos hara ver si funciona todo bien
2 no se que pasa si pongo tildes por el tema de los caracteres asi que no me la jugare
3 pero esto sin tildes ni comas ni puntos queda un poco raro
4 me nos mal que esta parte no se la lee nadie despues
```

Y la ejecución:

```
** Visual Studio 2019 Developer PowerShell v16.7.4
** Copyright (c) 2020 Microsoft Corporation
*****
PS C:\Users\xaviv\Desktop\JAVIER\UNIVERSIDAD\CURSO 20.21\CURSO\1º CUATRI\MODELOS DE COMPUTACION\PRACTICAS\PRACTICA2\C++> flex .\ejemplo1.1
PS C:\Users\xaviv\Desktop\JAVIER\UNIVERSIDAD\CURSO 20.21\CURSO\1º CUATRI\MODELOS DE COMPUTACION\PRACTICAS\PRACTICA2\C++> gcc lex.yy.c -lfl
PS C:\Users\xaviv\Desktop\JAVIER\UNIVERSIDAD\CURSO 20.21\CURSO\1º CUATRI\MODELOS DE COMPUTACION\PRACTICAS\PRACTICA2\C++> .\a.exe
Numero de Caracteres = 208, Numero de Líneas = 3, Numero de Dígitos = 0
PS C:\Users\xaviv\Desktop\JAVIER\UNIVERSIDAD\CURSO 20.21\CURSO\1º CUATRI\MODELOS DE COMPUTACION\PRACTICAS\PRACTICA2\C++>
```

Dice que el número de líneas es 3 porque solo cuenta los saltos de línea. Una forma de arreglar esto sería que empiece inicializada a 1 o que al final, a la hora de mostrar, muestre una más de las que contó siempre y cuando cuente alguna.

Y ya sin tantas explicaciones, yendo más al grano y simplemente a modo de ejemplo, pongo un tercer caso.

```
car      [a-zA-Z]
digito   [0-9]
signo    (\-|\+ )
suc      ({digito}+)
enter    ({signo}?{suc})
real1    ({enter}\.{digito}*)
real2    ({signo}?\.{suc})
int      ent=0, real=0, ident=0, sumaent=0;

%%
    int i;
{enter}          {ent++; sscanf(yytext, "%d", &i); sumaent += i; printf("Numero entero %s\n", yytext);}
({real1}|{real2}) {real++; printf("Num. real %s\n", yytext);}
{car}({car}|{digito})* {ident++; printf("Var. ident. %s\n", yytext);}
. | \n           {;}
%%
int main(){
    yyin = fopen("texto3.txt", "r");
    yylex();
    printf("Numero de Enteros %d, reales %d, ident %d, Suma de Enteros %d", ent, real, ident, sumaent);
}
```

```
car      [a-zA-Z]
digito   [0-9]
signo    (\-|\+ )
suc      ({digito}+)
enter    ({signo}?{suc})
real1    ({enter}\.{digito}*)
real2    ({signo}?\.{suc})
int      ent=0, real=0, ident=0, sumaent=0;

%%
    int i;
{enter}          {ent++; sscanf(yytext, "%d",
&i); sumaent += i; printf("Numero entero %s\n", yytext);}
({real1}|{real2}) {real++; printf("Num. real %s\n", yytext);}
{car}({car}|{digito})* {ident++; printf("Var. ident. %s\n",
yytext);}
. | \n           {;}
%%
int main(){
    yyin = fopen("texto3.txt", "r");
    yylex();
    printf("Numero de Enteros %d, reales %d, ident %d, Suma de Enteros %d",
ent, real, ident, sumaent);
}
```

Se declaran al principio las expresiones regulares para caracteres, dígitos, signo (en este caso pueden ser “-” o “+”), secuencia de dígitos (“suc” formada por cualquier cadena de dígitos y por tanto invocamos a esta expresión regular), enteros y reales. El contador de cada tipo se pone a 0.

En la siguiente sección declaramos una variable local “i” y definimos la acción que se realizará según qué leamos del archivo de entrada. Por lo general muestra un texto informando sobre el tipo que ha encontrado y enseñando ese signo o carácter en sí. De encontrar un punto o un salto de línea, no hace nada.

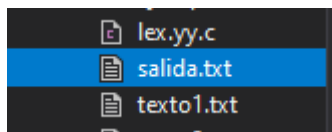
El texto de prueba en este caso será:

```
A finales del pasado año, la localidad de 16.615 habitantes tenía ya la mitad del paro industrial que alcanzó en 2008, cuando alcanzó el cenit debido a una crisis provocada por la fuga del lujo hacia destinos de producción más económicos en 2004. Escarmentadas por la caída en la calidad, muchas de las firmas huidas comenzaron a regresar a partir de entonces. A la vuelta, Ubrique les esperó sin rencores y alejada de vicios pasados, como la economía sumergida o la competencia desleal. A finales de 2019, apenas quedaban 872 desempleados en esta actividad, frente a los 1.634 que se alcanzaron hace 12 años; según datos ya aportados por la Delegación Territorial de Conocimiento y Empleo de la Junta de Andalucía.
```

La compilación y ejecución para que vuelque la salida en un archivo de texto:

```
PS C:\Users\xaviv\Desktop\JAVIER\UNIVERSIDAD\CURSO 20.21\CURSO\1º CUATRI\MODELOS DE COMPUTACION\PRACTICAS\PRACTICA2\C++> flex .\ejemploflex.l
PS C:\Users\xaviv\Desktop\JAVIER\UNIVERSIDAD\CURSO 20.21\CURSO\1º CUATRI\MODELOS DE COMPUTACION\PRACTICAS\PRACTICA2\C++> gcc lex.yy.c -lf1
PS C:\Users\xaviv\Desktop\JAVIER\UNIVERSIDAD\CURSO 20.21\CURSO\1º CUATRI\MODELOS DE COMPUTACION\PRACTICAS\PRACTICA2\C++> .\a.exe > salida.txt
PS C:\Users\xaviv\Desktop\JAVIER\UNIVERSIDAD\CURSO 20.21\CURSO\1º CUATRI\MODELOS DE COMPUTACION\PRACTICAS\PRACTICA2\C++>
```

Que se crea automáticamente:



Y contiene:

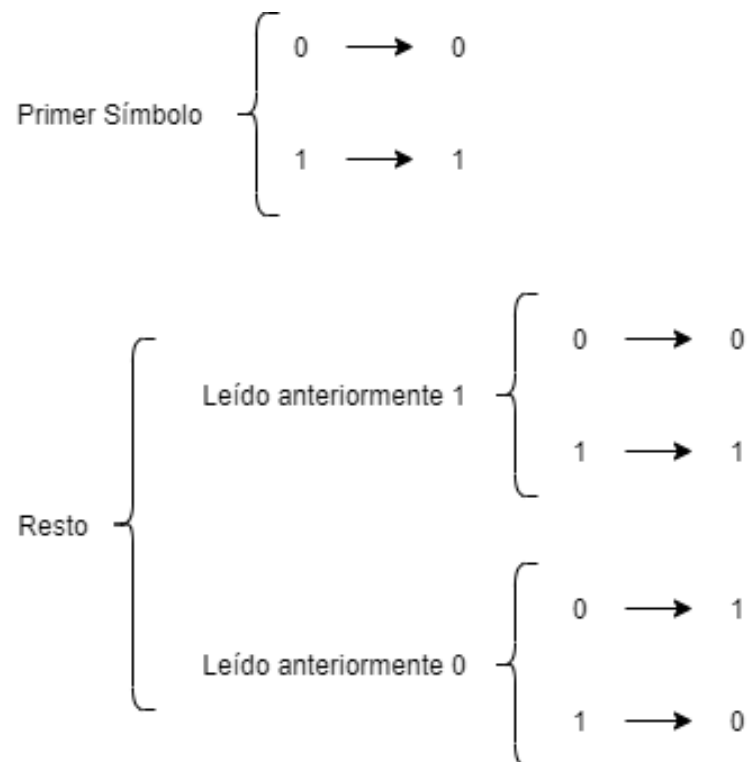

```
Var. ident. A
Var. ident. finales
Var. ident. del
Var. ident. pasado
Var. ident. a
Var. ident. o
Var. ident. la
Var. ident. localidad
Var. ident. de
Num. real 16.615
Var. ident. habitantes
Var. ident. ten
Var. ident. a
Var. ident. ya
Var. ident. la
Var. ident. mitad
Var. ident. del
Var. ident. paro
Var. ident. industrial
Var. ident. que
Var. ident. alcanz
Var. ident. en
Numero entero 2008
Var. ident. cuando
Var. ident. alcanz
Var. ident. el
Var. ident. cenit
```

Práctica 3. Implementar un codificador y un decodificador con una máquina de estados.

La forma de codificar consistirá en leer el primer símbolo de la cadena y codificarlo como un 1 si es un 1 o como un 0 si es un 0. Este siempre se deja así para que al decodificar, el resultado sea unívoco y no de lugar a errores.

A raíz de esto, la codificación pasará por tener en cuenta si lo leído anteriormente es un 1, manteniendo la misma cadena (1 -> 1 y 0 -> 0), o si es un 0, codificando el 0 como 1 y viceversa.

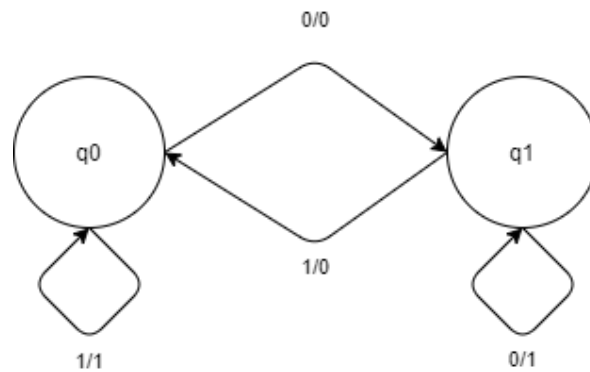
De una forma más visual, estas son las transformaciones que tendríamos:



Para el que un ejemplo sería:

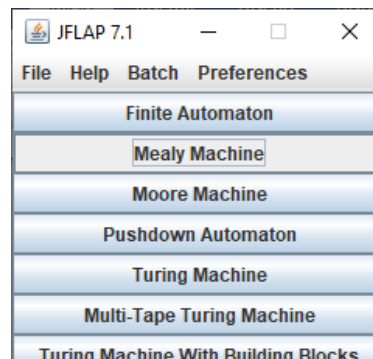
1	1	0	0	1	0	1	1	0
↓	↓	↓	↓	↓	↓	↓	↓	↓
1	1	0	1	0	0	0	1	0

Quedando la máquina de estados asociada de la siguiente manera:

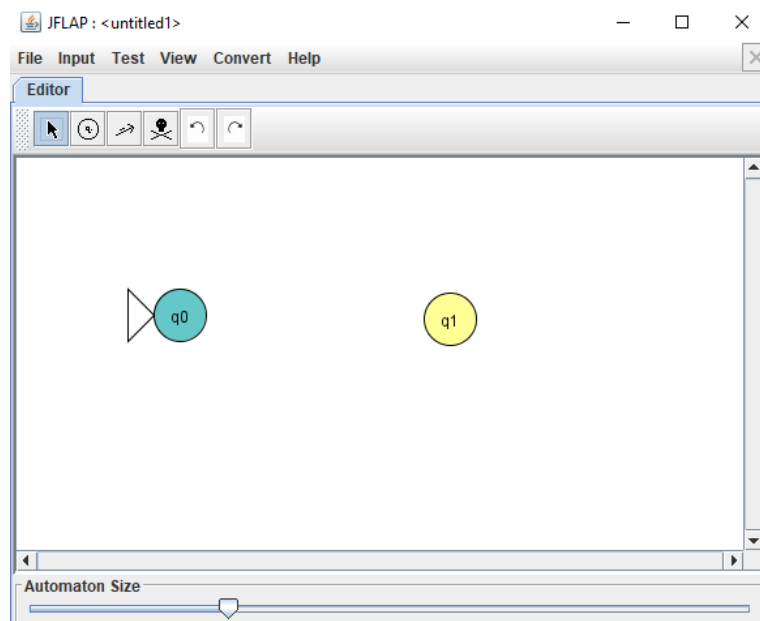


JFLAP

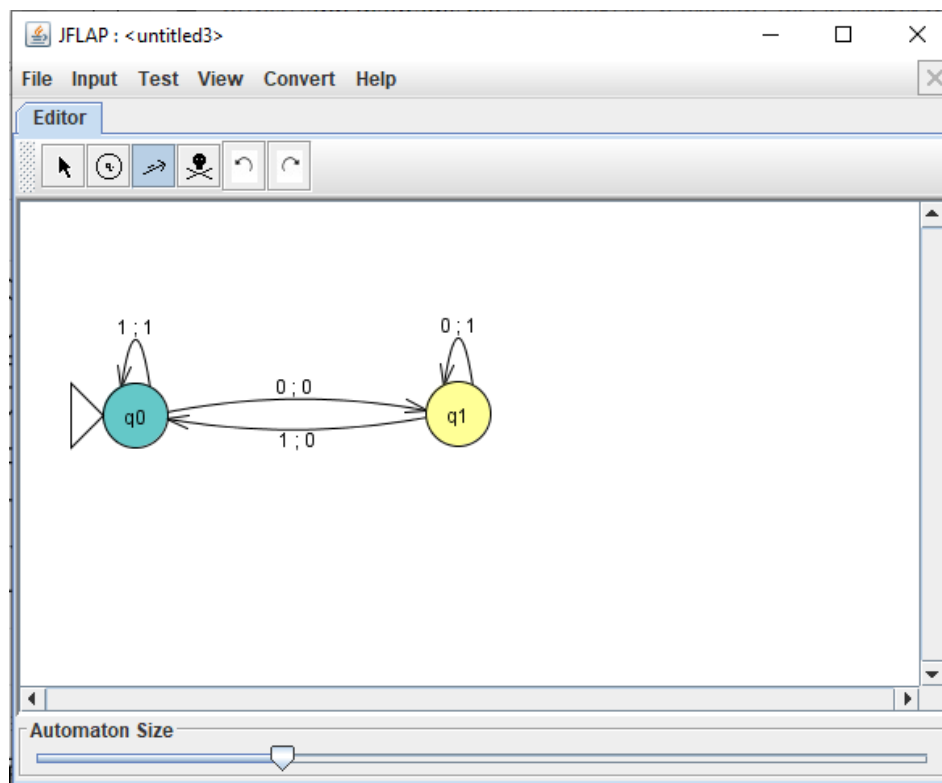
Ahora vamos a implementarlo en JFlap y vamos a probar varias cadenas. Para ello, en JFlap, abrimos una Máquina de Mealy.



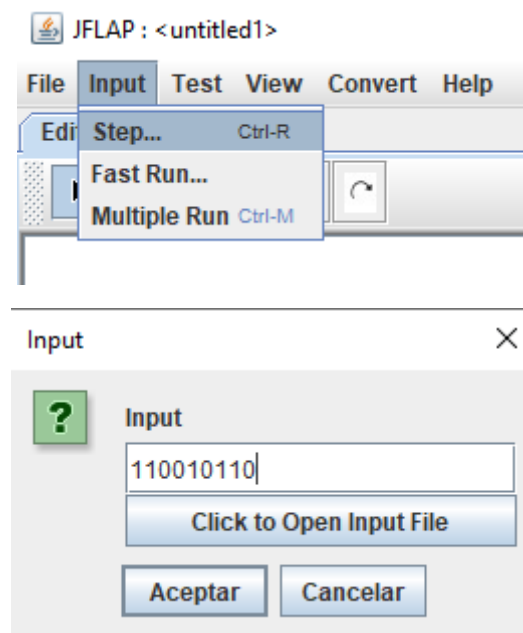
Metemos 2 estados y marcamos q_1 como el inicial:

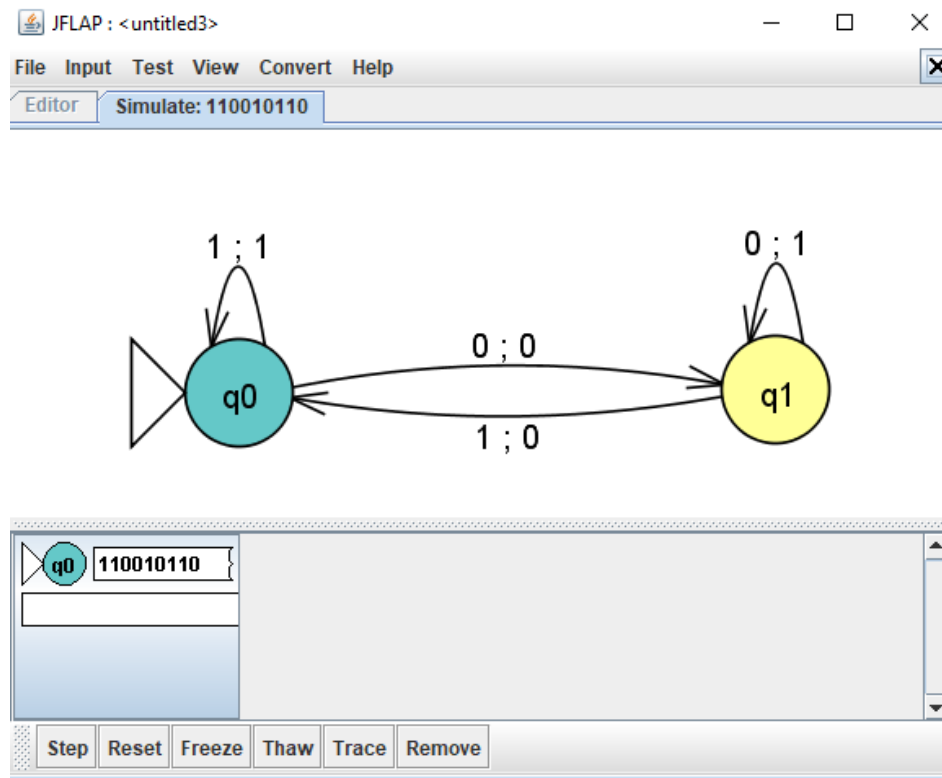


Y ponemos las transiciones que se realizarán según lo leído anteriormente:

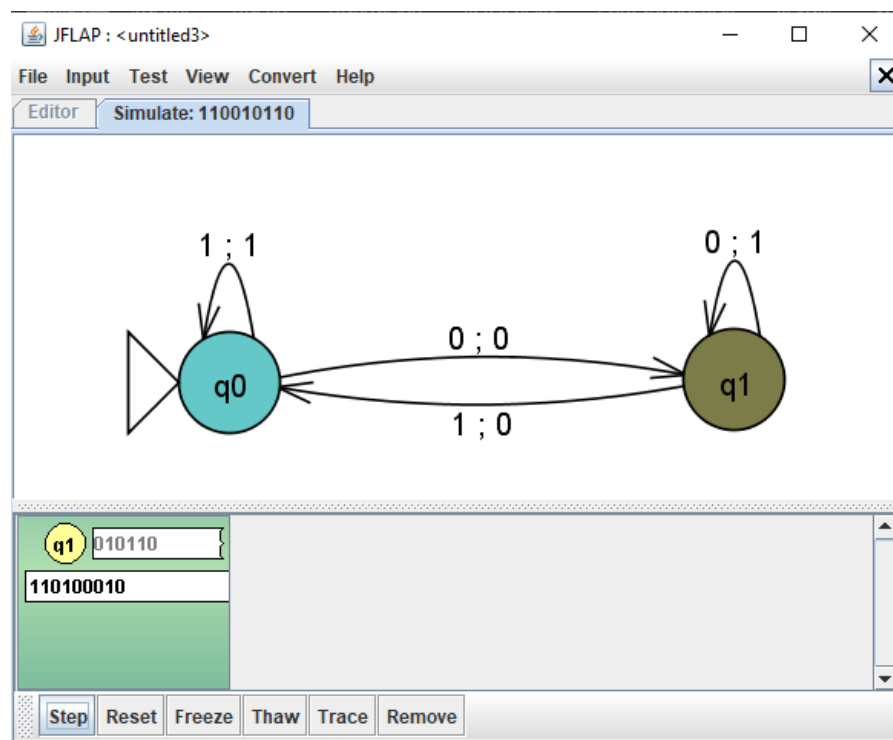


Y de entrada como primera prueba metemos la cadena que hemos probado anteriormente de forma manual:





Y al realizar todos los pasos, termina construyendo la siguiente cadena:

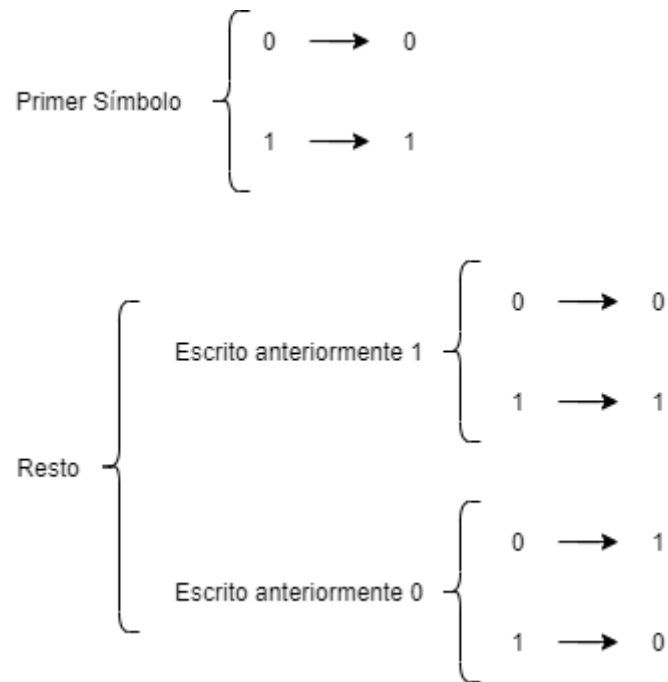


Que como vemos, coincide con la obtenida a mano:

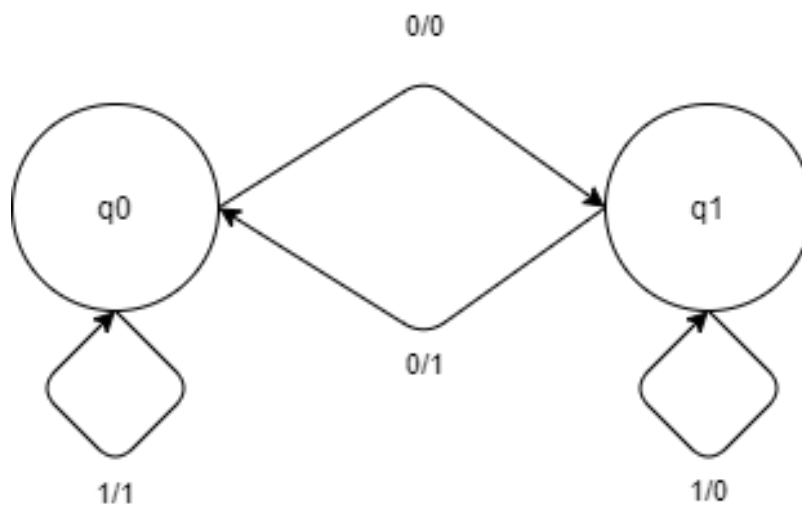
1 1 0 1 0 0 0 1 0

Y ahora pasamos al decodificador.

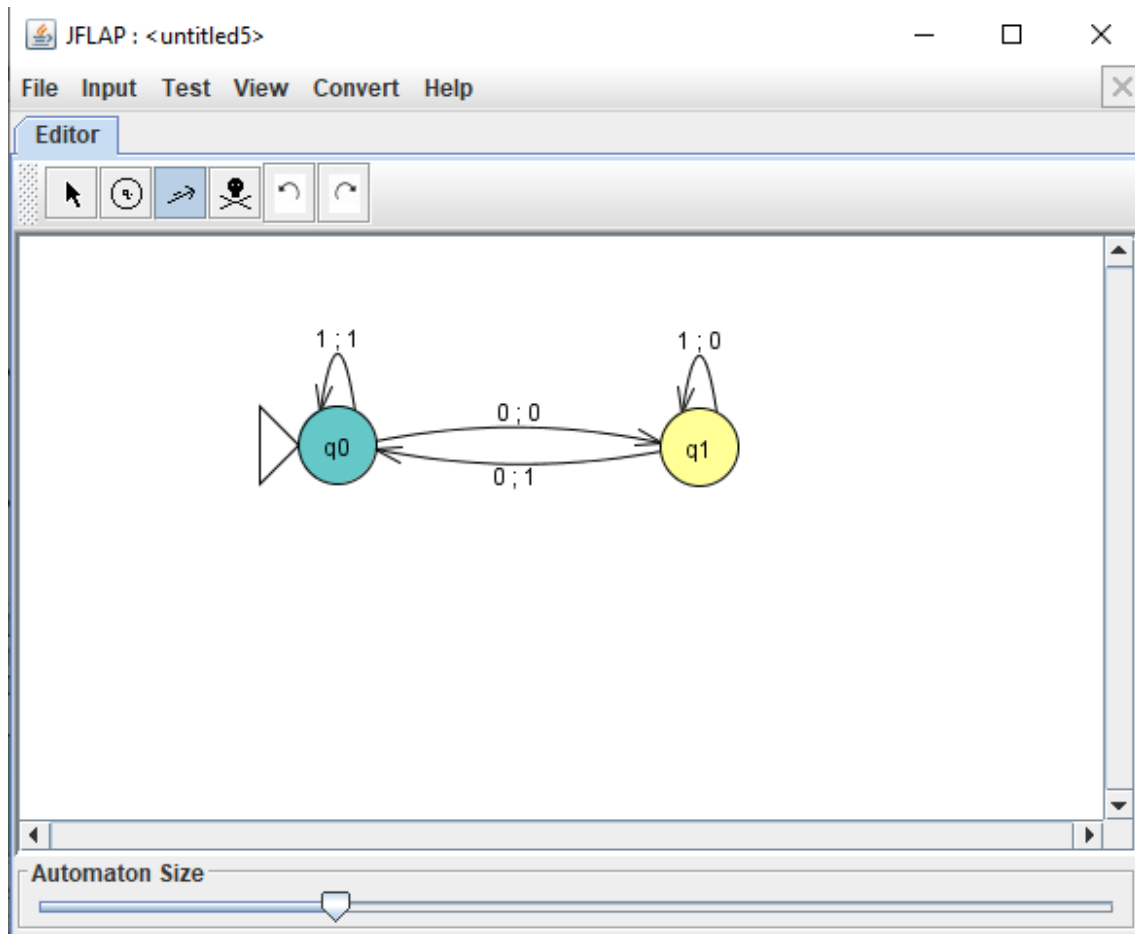
Puede parecer que simplemente hay que seguir los pasos a la inversa, pero la clave está en que no nos fijamos en qué se leyó previamente, sino en qué se escribió. Para ello, tenemos en cuenta los símbolos ya recuperados de la cadena. Finalmente, obtenemos las siguientes reglas:



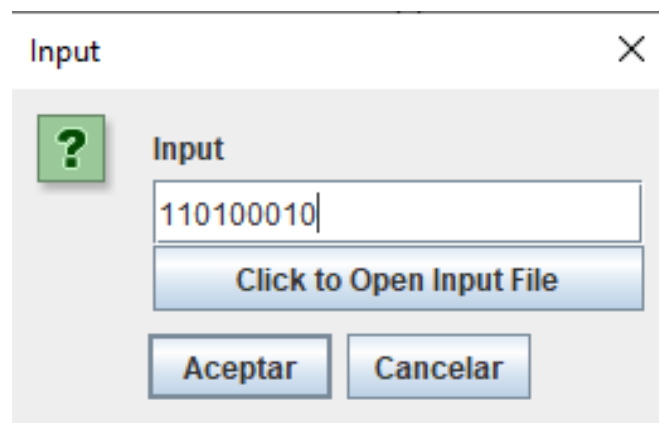
Resultando en la siguiente máquina de estados:



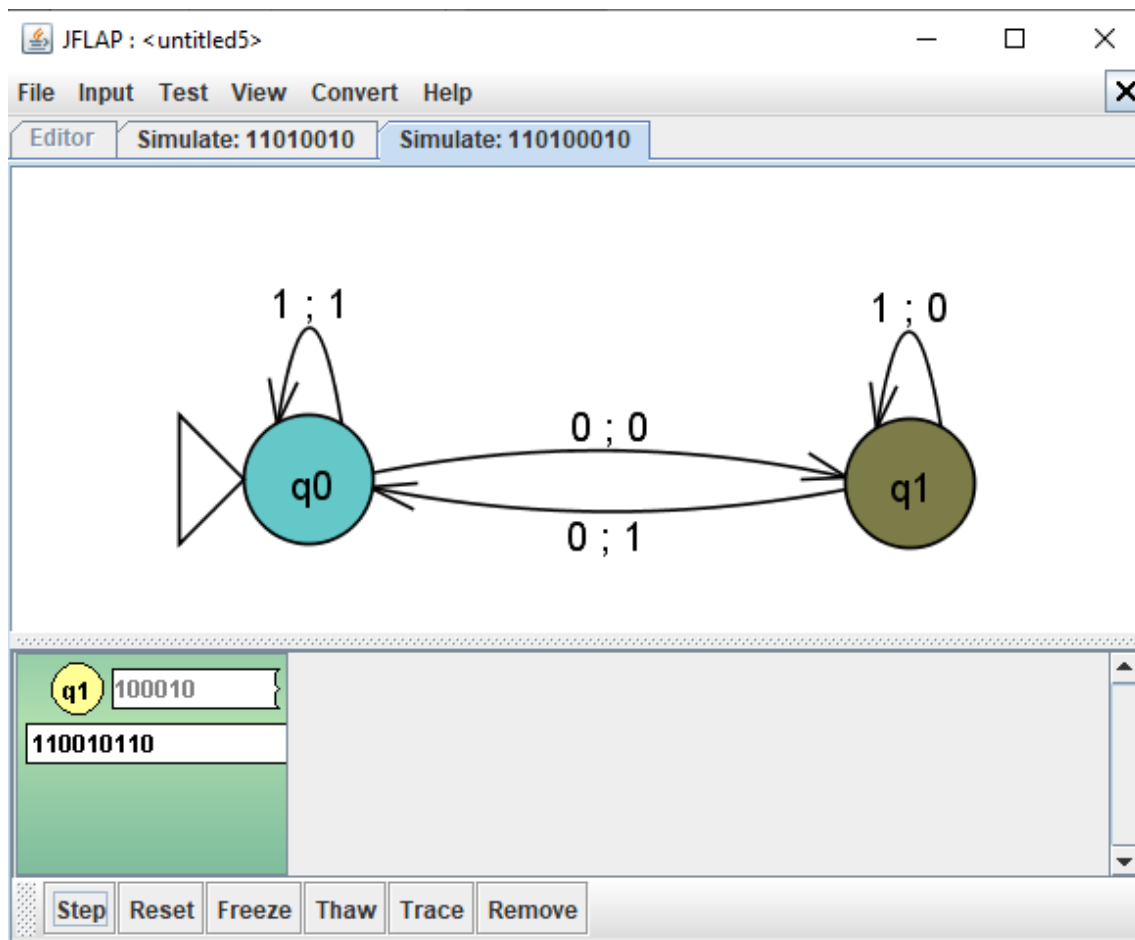
Que vamos a llevar a JFlap como hicimos con el codificador:



Y le meto lo que sería la cadena anterior tras la codificación:



Y tras todos los pasos, esta es la cadena que obtiene:

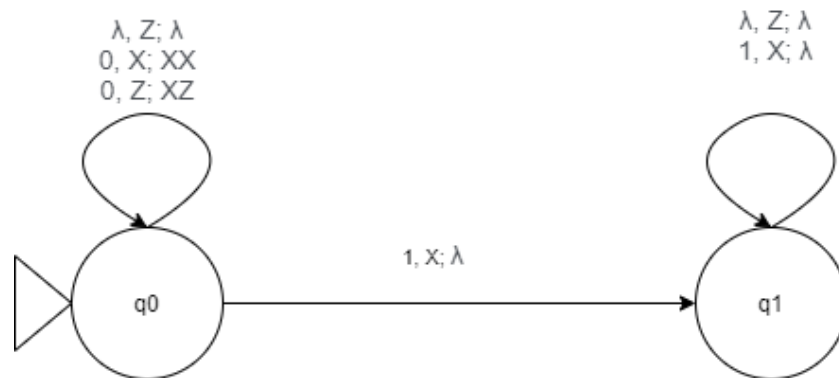


Que, como podemos ver, es igual que la cadena que introdujimos originalmente:

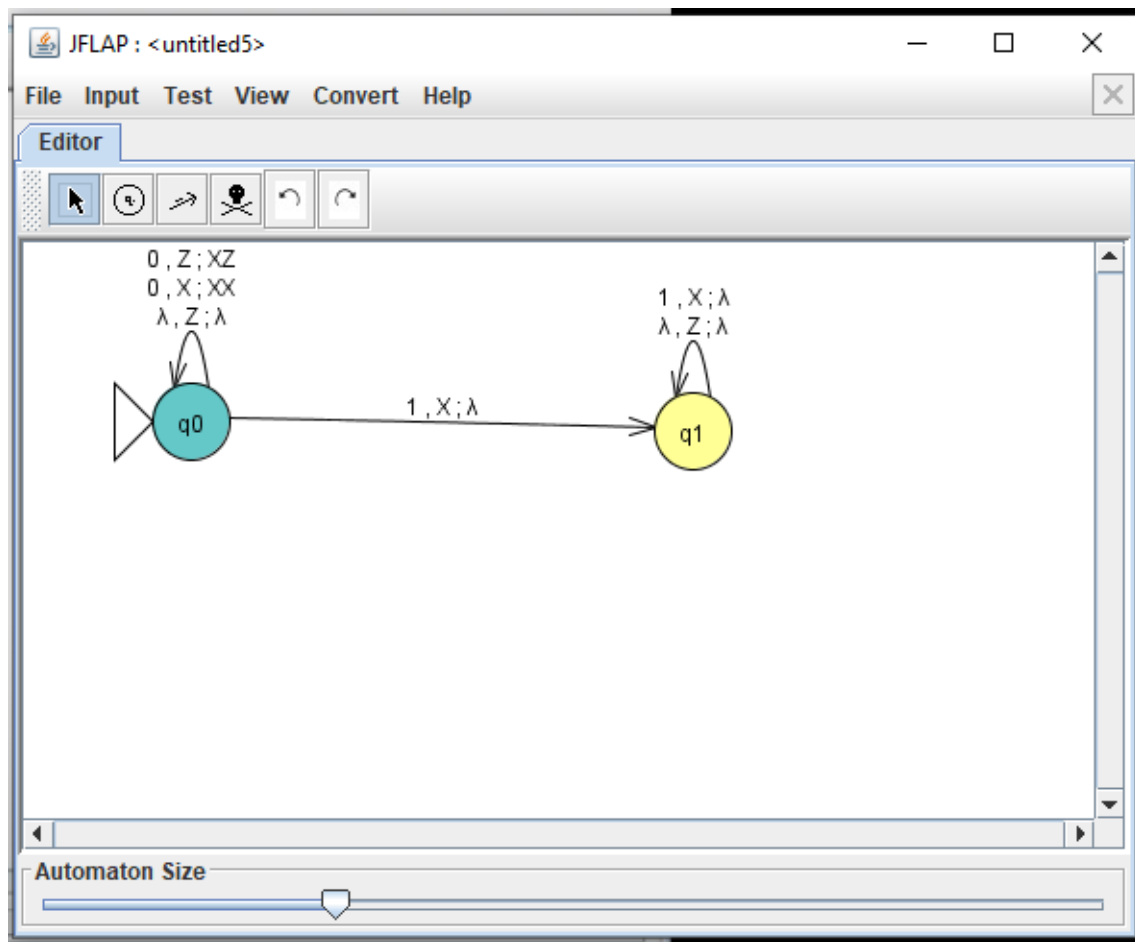
1 1 0 0 1 0 1 1 0

Práctica 4. Autómata con pila que acepta cadenas que tienen el mismo número de 0 que de 1 de acuerdo al criterio de estado final.

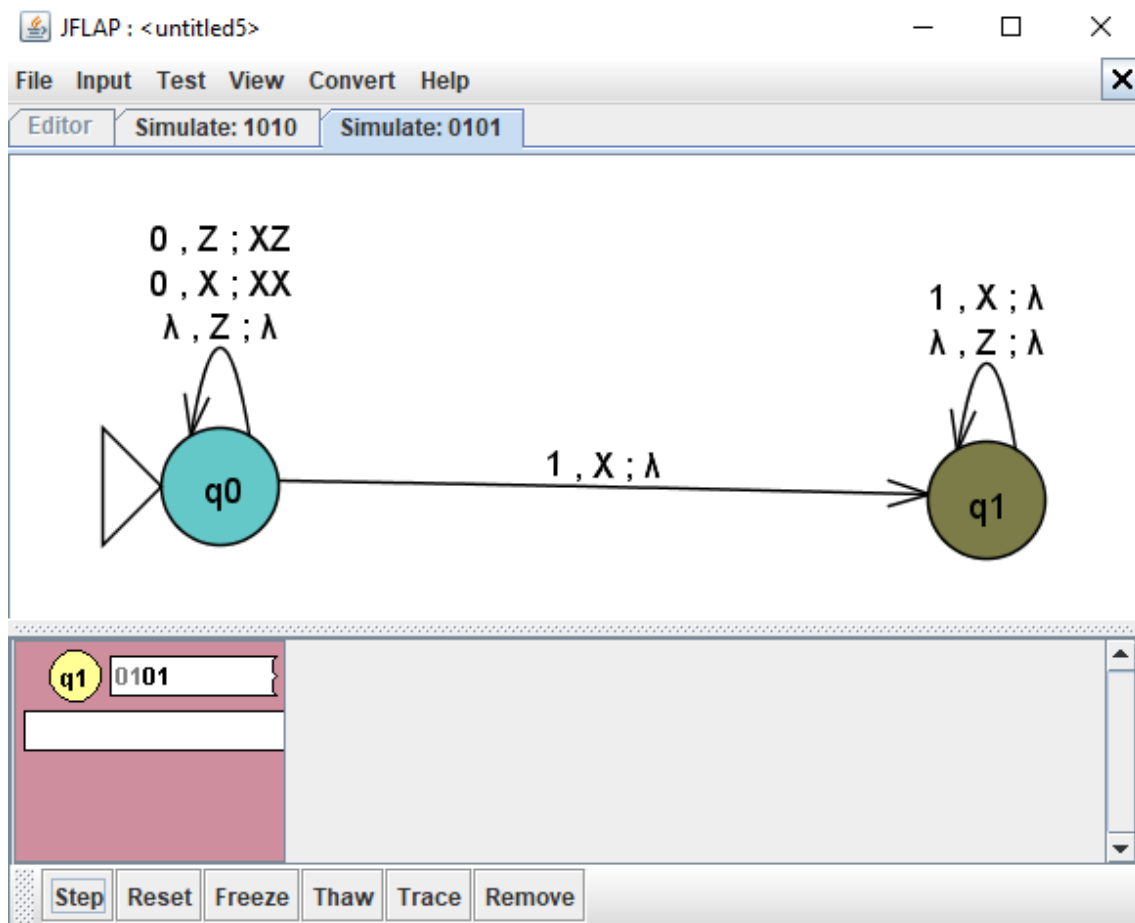
Para esta práctica partiremos de un autómata con pila que sigue el criterio de pila vacía. Este autómata permite la entrada de cadenas formadas por 0 y 1 que contienen la misma cantidad de ambos caracteres. Su máquina de estados sería la siguiente:



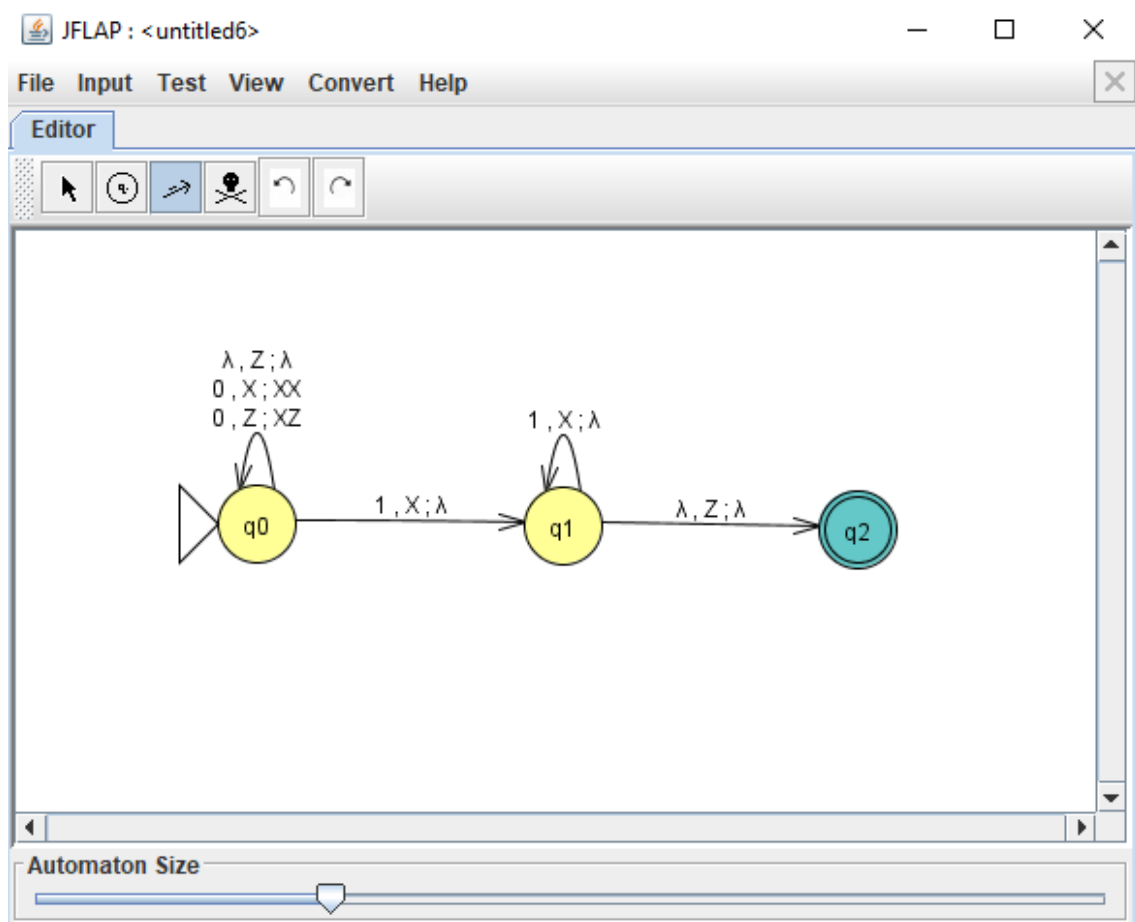
Al probar este elemento en JFlap, veremos que tiene una errata. Antes de verlo en un ejemplo práctico, entendemos que estando en q_0 , al leer un 1 sacará una X de la pila, no escribirá nada y se pasará a q_1 . Una vez aquí, no tenemos especificado qué hacer si se lee un 0 en algún momento, por lo que en ese caso se abortará la cadena y no será aceptada.



Y vemos que al probar la cadena 0101, que a priori debería ser aceptada por tener el mismo número de 0 que de 1, se produce el error previamente comentado.



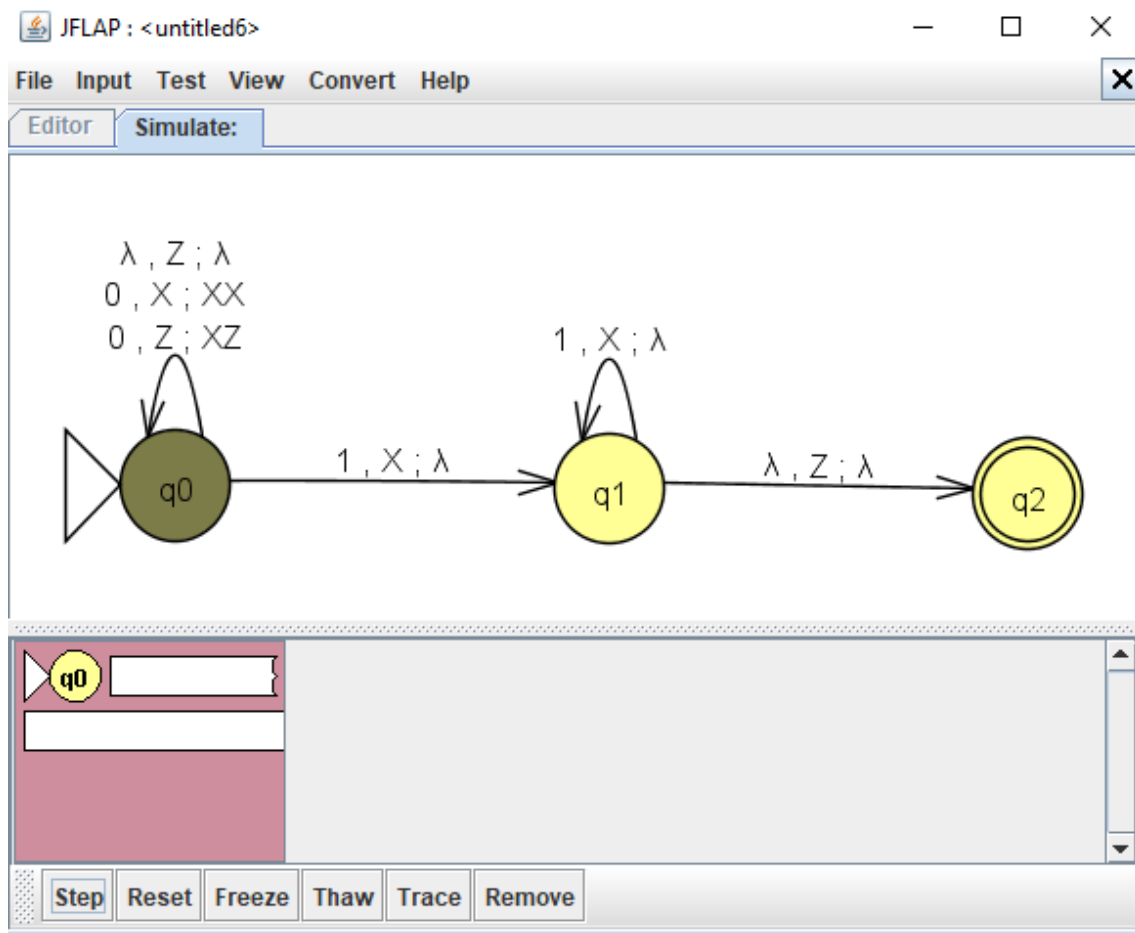
Entonces lo que vamos a hacer son un par de modificaciones para, partiendo de ese modelo, conseguir que siga el criterio de estado final. Para ello, obviamente debemos añadir un nuevo estado (q2) que será el final y al que se irá al no leer nada, borrando así Z de la pila (es el símbolo de pila vacía). La forma del autómata con criterio de estado final sería:



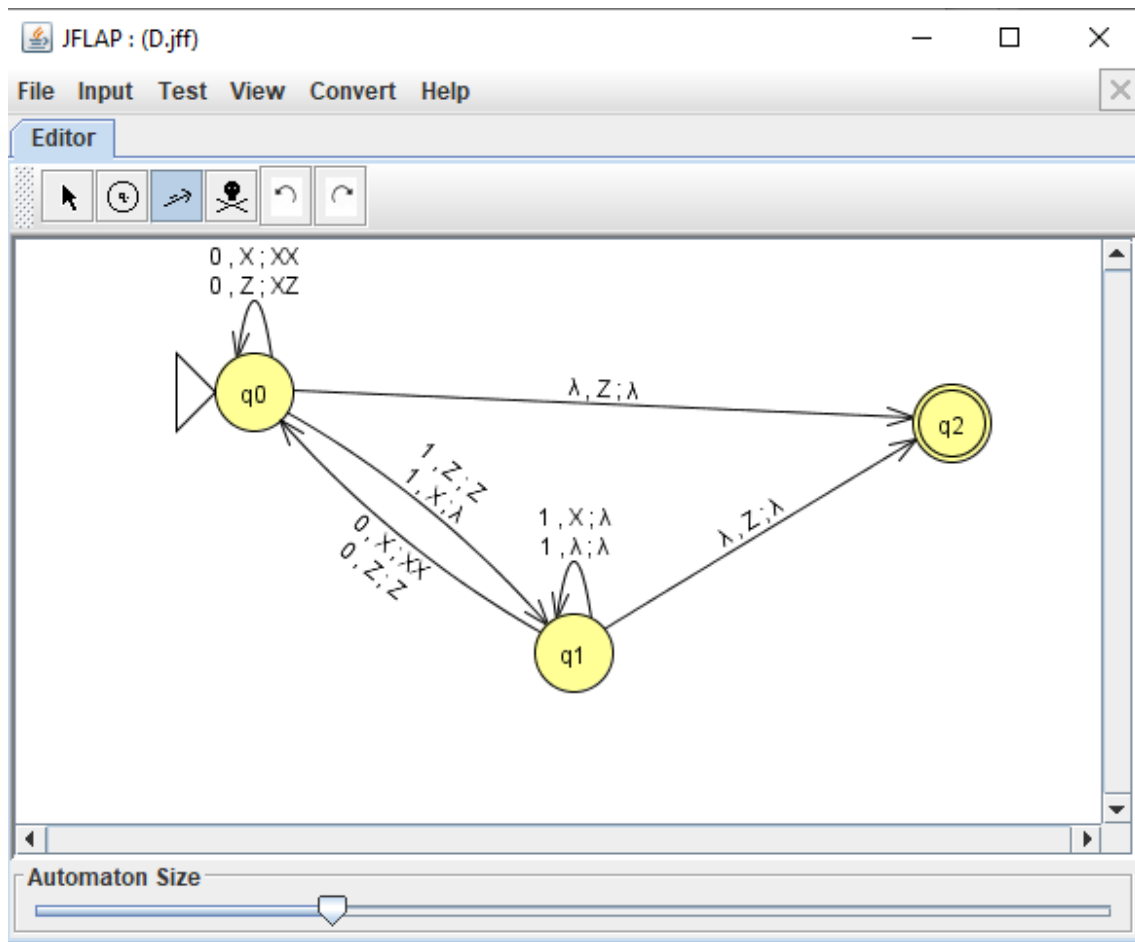
Pero aquí ya podemos ver el error que nos pedían corregir. Al meter una cadena vacía, el número de 0 y de 1 es el mismo (0 de cada uno) por lo que debería tomarlo como una cadena válida. Partiendo de esa base, imaginemos cómo se comporta ante una cadena vacía:

- Estamos en q_0 por ser el inicial
- No lee nada de entrada (caso de $\lambda, Z; \lambda$)
- Saca la Z de la pila
- Termina estando en q_0 que no es el estado final y rechaza la cadena.

Aunque ya hemos razonado el error, aquí va una prueba gráfica de que no es una cadena aceptada:



Para arreglar el error solo debemos modificar una de las transiciones de q_0 y conectarla con el estado final. Más concretamente, aquella sobre no leer nada de entrada. También habrá que añadir la acción asociada a leer un 0 encontrándonos en q_1 , que sería volver a q_0 e introducir una X en la pila. El modelo final quedaría tal que:



Y vamos a probar tanto la cadena rechaza anteriormente por el criterio de pila vacía, como la cadena sin contenido.

