

# MANUAL REDUCIDO

## JFLAP

### (Java Formal Language and Automata Package)

#### 1. Introducción a JFLAP

JFLAP (Java Formal Language and Automata Package) es una herramienta para la enseñanza y la visualización interactiva de lenguajes formales. Permite crear y operar sobre autómatas (finitos, máquinas de Moore y Mealy, Turing...), gramáticas, expresiones regulares y L-systems.

En esta práctica inicial sólo nos vamos a centrar en la parte enfocada a las gramáticas, y según avance el curso, iremos profundizando en los distintos apartados de la aplicación. Aunque algunas secciones no se verán en la asignatura.

#### 2. Conceptos básicos de gramáticas

Antes de comenzar con la explicación de la práctica a realizar en esta sesión, se incluye un breve recordatorio de una serie de definiciones y nociones sobre gramáticas. Todo esto, visto en clases de teoría, será necesario conocerlo a la hora de utilizar el programa.

Una gramática esta formada por:

- Elementos terminales ( $\Sigma_T$ ): Todas las cadenas del lenguaje representado por la G ( $L(G)$ ), están formadas con símbolos de este alfabeto.
- Elementos no terminales ( $\Sigma_N$ ): Es un conjunto de símbolos auxiliares introducidos como elementos auxiliares para la definición de G pero que no figuran en las cadenas de  $L(G)$ .
- Producciones: Conjunto de reglas de producción  $u::=v$  donde  $u \in \Sigma^+$  y  $v \in \Sigma^*$   $u=xAy$  tal que  $x, y \in \Sigma^*$  y  $A \in \Sigma_N$ .
- Axioma: Es un símbolo destacado, no Terminal, a partir del que se comienzan a aplicar las reglas de producción.

#### 3. Iniciar JFLAP

La herramienta se puede descargar gratis de la url: [www.jflap.org](http://www.jflap.org) (rellenando un formulario previamente), o bien os la podéis bajar de aula global.

JFLAP está implementado en Java, por lo que es necesario tener la versión Java SE 1.4 o posterior instalada en vuestro ordenador. En el caso de las aulas informáticas de la clase de prácticas, ya estará instalado.

JFLAP se distribuye como un ejecutable .jar. Para ejecutar este tipo de ficheros en un sistema operativo Windows, basta con hacer doble clic en JFLAP.jar. En el caso de Linux, desde la consola, con el fichero JFLAP.jar en el directorio actual, tendréis que introducir el comando `java -jar JFLAP.jar` (este comando funciona de la misma forma en la consola de Windows).

Una vez abierta la aplicación, aparecerá en pantalla una interfaz como la que se muestra en la Figura 1.



Figura 1: Ventana inicial de JFLAP

En este caso, sólo nos preocuparemos de la parte de JFLAP que nos permite manipular gramáticas, por lo que pulsando en *Grammar* en la interfaz de la Figura 1 pasaréis a la ventana del editor.

### 3.1. Editor de gramáticas

En esta ventana, podremos definir las gramáticas. En la Figura 2 se observa un ejemplo ya introducido.

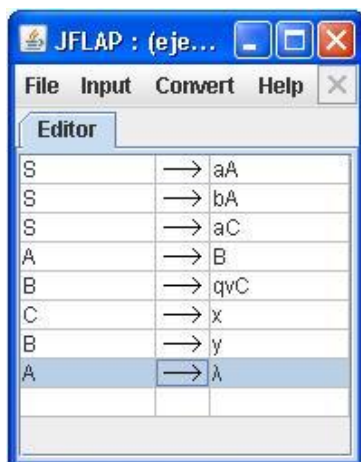


Figura 2: Editor de gramáticas de JFLAP

Figura 3: Vista de una gramática guardada con formato xml

Abriendo la opción *File* del menú, podremos guardar las gramáticas en ficheros (en formato xml, Figura 3), cargarlos más tarde en el programa, imprimirlos, abrir un nuevo editor (puede haber varios abiertos a la vez), cerrarlo, y para el caso en el que haya varias opciones abiertas (editores, brute parse...), *dismiss tab* cerrará la actual.

Antes de empezar a manipularlo, hay unas notas sobre el funcionamiento del editor, que tendremos que tener en cuenta:

- Para escribir en un recuadro específico del editor, posicionarse y hacer clic sobre él, o bien, utilizando la tecla del tabulador, nos podemos mover por las celdas.

- Cada carácter que escribamos en el editor, será un símbolo del alfabeto de la gramática. No se pueden manejar símbolos de más de un carácter.
- Al introducir las producciones hay que tener en cuenta, que JFLAP no nos pedirá que le digamos cuál es el conjunto de los símbolos terminales y el de los no terminales. Esta información la “deducirá” en base a las producciones que escribamos en el editor. JFLAP interpretará como símbolos no terminales todos aquellos escritos en mayúsculas, y como terminales los demás (p.ej.: 4, q, #, a, etc.).
- El axioma será siempre el símbolo que escribamos en la parte izquierda de la primera producción (la primera línea del editor).
- Si al escribir una producción se deja en blanco la parte derecha, el editor entenderá que produce  $\lambda$ .
- JFLAP no acepta notación de Backus.
- El tipo de gramática no se define, ni el editor comprueba que el formato sea correcto, salvo si posteriormente se lleva a cabo alguna operación no permitida, en cuyo caso aparecerá un mensaje de error.

En el ejemplo de la Figura 2, en la que tenemos una gramática, vemos como el axioma es S, el conjunto de no terminales es {S, A, B, C}, y el de los terminales es {a, b, q, v, x y}. La última producción  $A \rightarrow \lambda$ , es la única producción  $\lambda$  que hay.

Ahora, prueba a escribir esta misma gramática en tu ventana del editor.

### 3.2. Brute Force Parser (Derivador por fuerza bruta)

En el menú *Input*, de la ventana del editor de gramáticas, vemos la opción *Brute Force Parser*. Este operador recoge una palabra que hayamos escrito en el recuadro *Input* y, según la gramática introducida en el editor, derivará la sentencia paso a paso, a través de reemplazos de no terminales con producciones, mostrando los pasos intermedios del proceso de derivación. Esos pasos intermedios (llamados formas sentenciales) incluirán elementos terminales y no terminales, hasta que en la última sentencia, cuando la palabra es aceptada por la gramática, la derivación esté formada sólo por elementos terminales.

Este procedimiento se puede visualizar de dos formas cuando la palabra es aceptada: mediante una tabla de derivación (Figura 4), o gráficamente, con un árbol que es la opción seleccionada por defecto por la aplicación (Figura 5)\*.

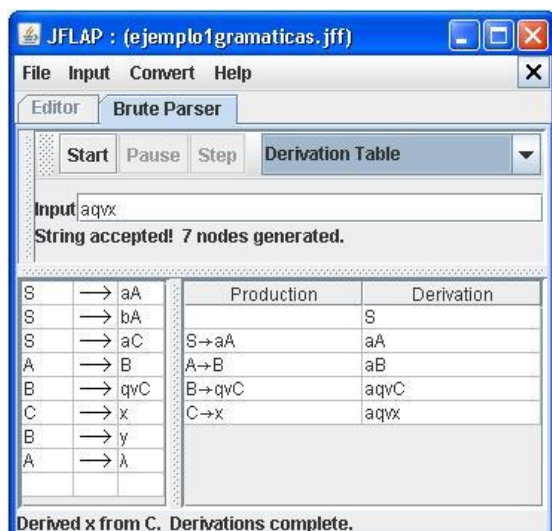


Figura 4: Brute parser con tabla de derivación

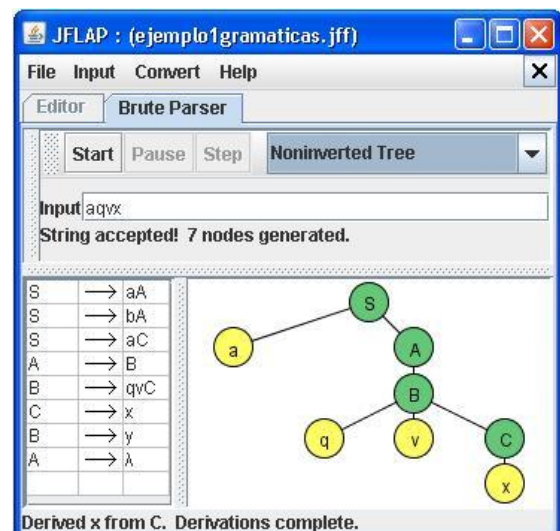


Figura 5: Brute parser con árbol de derivación

\* El árbol de derivación no debe preocuparnos en la primera clase

Para utilizar esta opción de la herramienta, habrá que introducir la palabra, pulsar el botón *Start*, y si la palabra es aceptada (aparecerá un comentario debajo del recuadro de *Input* que lo dirá), podremos pulsar el botón *Step*, para que vayan surgiendo los pasos que ha seguido el programa para comprobar que esa palabra era generada por la gramática.

Una palabra puede ser no aceptada por varios motivos:

- Que la palabra se haya escrito mal, añadiendo un elemento no terminal del alfabeto (las palabras aceptadas por una gramática están formadas sólo por elementos terminales), o que aparezca un símbolo que no pertenezca al alfabeto de la gramática introducida en el editor. En este caso, aparecerá una ventana con el error.
- Que la palabra esté bien escrita, pero que la gramática no la pueda generar (no pertenezca al lenguaje). En ese caso, debajo del recuadro de *Input*, aparecerá: “String rejected”.

### 3.3. Multiple Brute Force Parse (Derivador Múltiple por fuerza bruta)

Esta opción es similar a la anterior. La diferencia radica en que en este caso, se puede comprobar la pertenencia de varias palabras a una gramática de forma simultánea, mientras que antes sólo se comprobaba una (Menú > Input > Multiple Brute Force Parser).

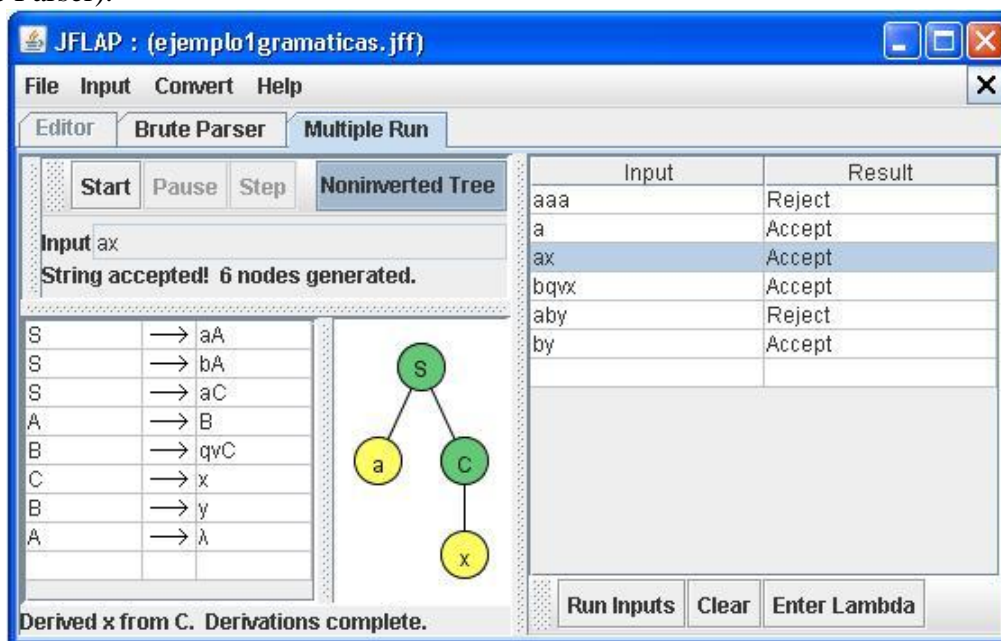


Figura 6: Ejemplo de derivador múltiple por fuerza bruta

En este caso, en la tabla de la derecha (Ver Figura 5), en la columna de *Input*, añadiremos las palabras que queremos comprobar. Una vez agregadas todas, pulsando en el botón *Run Inputs*, la herramienta nos dirá las palabras aceptadas y las rechazadas tal y como muestra la Figura 6. Si queremos ver la tabla de derivaciones o el árbol de alguna palabra aceptada específicamente, habrá que seleccionar la fila correspondiente, y en la parte izquierda de la tabla, pulsar el botón *Start* y *Step*, que funcionan del mismo modo que para el caso explicado en el apartado 3.2.

### 3.4. Gramáticas de contexto libre (G2)

Este tipo de gramáticas tienen como característica, que en las producciones hay un solo símbolo no terminal a la izquierda y no tienen restricciones en la parte derecha. Para trabajar con este tipo de gramáticas en JFLAP, se seguirá el mismo procedimiento que lo explicado para gramáticas regulares. En este apartado sólo incluiremos algún ejemplo de trucos para facilitar el reconocimiento de los lenguajes que genera una G2.

Primero pasamos al editor una gramática de tipo 2, como la que muestra la Figura 7, y vemos la tabla de derivación resultado de introducir la palabra 'aaabbbbb' (Figura 8).

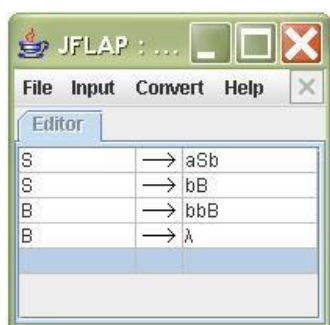


Figura 7: Ejemplo de G2

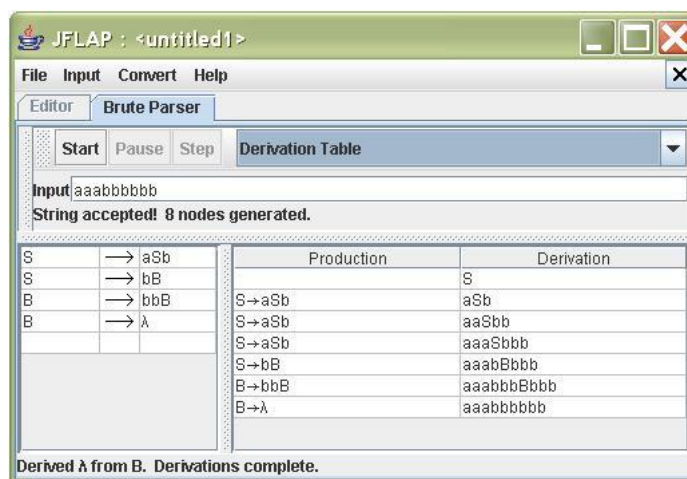


Figura 8: Tabla de derivación que acepta la palabra aaabbbbb

Una forma de pensar en la descripción de todas las palabras que acepta una gramática, es pensar en qué tipo de palabra puede derivar cada elemento no terminal. ¿En qué puede derivar el axioma  $S$ ? Fijándonos en las tres primeras derivaciones de la tabla de la Figura 8,  $S$  puede derivar en  $\{aSb, aaSbb, aaaSbbb, \dots\}$ . Si pensamos en todas las producciones de  $S$ , podemos ver otras formas sentenciales en las que  $S$  puede derivar, como  $\{bB, abBb, aabBbb, \dots\}$ , y podemos sacar la conclusión de que  $S$  puede derivar en  $\{a^n b B b^n \mid n \geq 0\}$ .

Si nos fijamos ahora en el símbolo no terminal  $B$ , ¿en qué puede derivar? Si modificamos lo que tenemos en el editor, colocando como primera producción  $B \rightarrow B$ , como si fuera el axioma, e introducimos unas cuantas palabras de prueba en el derivador múltiple por fuerza bruta, veremos que  $B$  puede derivar en las palabras  $\{\lambda, bb, bbbb, bbbbbb, \dots\}$ , o expresado de otra forma  $(bb)^*$ , un número par de  $b$ 's.

La combinación de en que pueden derivar  $S$  y  $B$  implica que el lenguaje de esta gramática es  $\{a^n b (bb)^* b^n \mid n \geq 0\}$

Por lo tanto, en JFLAP se puede determinar qué palabras genera un símbolo no terminal  $A$ , colocando  $A$  como si fuera el axioma. Para simular esto, introduce la primera producción que hubiera en el editor de tu gramática original al final de la lista de producciones para guardarla, y entonces introduce  $A \rightarrow A$  como la primera producción. Haz pruebas con varias palabras comprobando las que se aceptan y las que no. Según los resultados, podrás definir una regla general para el símbolo no terminal  $A$ . Si unes estos resultados, a la regla que hayas detectado que genera el axioma original, tendrás el lenguaje que genera la gramática inicial.



### 3.5. Otros tipos de gramáticas

Para gramáticas de tipo G0 y G1, el editor de JFLAP se comporta igual que en casos anteriores. Se pueden escribir en la parte derecha y en la parte izquierda de la producción tantos símbolos como tenga la gramática que queremos introducir sin ningún problema. Y a la hora de ver su tabla o árbol de derivación, también se podrá hacer.

En la Figura 9 tenemos un ejemplo de esto. Vemos como tres nodos del árbol producen otros tres nodos que forman una palabra del lenguaje de la gramática definida en el editor.

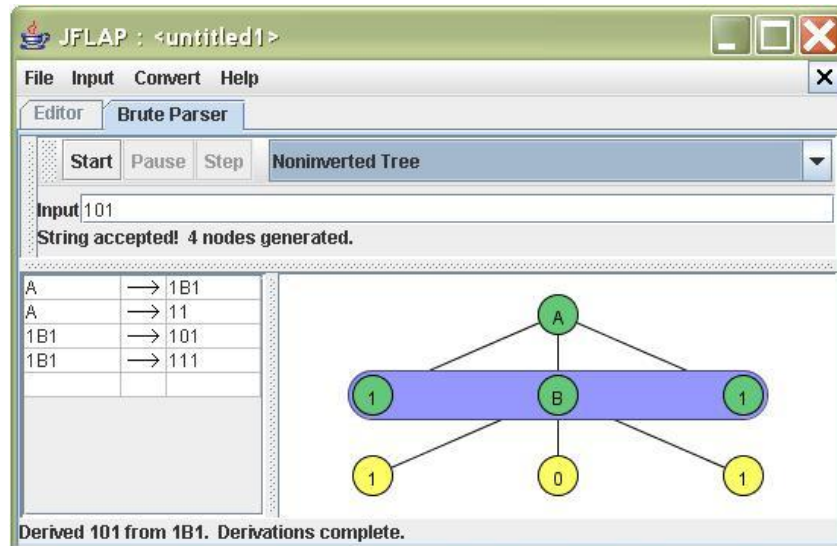


Figura 9: Árbol de derivación de una G1, que acepta la palabra 101

```
1 <?xml version="1.0" encoding="UTF-8"?><!--Created with JFLAP
6.0.--><structure>#13;
2 <type>grammar</type>#13;
3 <!--The list of productions.-->#13;
4 <production>#13;
5 <left>A</left>#13;
6 <right>1B1</right>#13;
7 </production>#13;
8 <production>#13;
9 <left>A</left>#13;
10 <right>11</right>#13;
11 </production>#13;
12 <production>#13;
13 <left>1B1</left>#13;
14 <right>101</right>#13;
15 </production>#13;
16 <production>#13;
17 <left>1B1</left>#13;
18 <right>111</right>#13;
19 </production>#13;
20 </structure>
21
```

Figura 10: Ejemplo de cómo sería el archivo de una gramática 0 en JFLAP

### 3.6 Uso del modo batch

El modo batch es una función que tiene JFLAP para poder probar ficheros con gramáticas y palabras, cargados desde archivo. De esta forma, se puede hacer un test fácilmente de las palabras que acepta o rechaza una gramática. Esta función puede ser muy útil si queremos hacer una comprobación rápida, si tenemos los archivos guardados y sin usar directamente el derivador múltiple por fuerza bruta.

Como ejemplo, usaremos la siguiente gramática con una lista de palabras:

$S \rightarrow aB$	aabbc
$S \rightarrow aA$	abc
$A \rightarrow abB$	aaabbc
$B \rightarrow bC$	abcc
$B \rightarrow aA$	aabaabaabaabbc
$C \rightarrow c$	aabaabbc
	aaabc

Para comenzar, introduce la gramática en el editor de JFLAP y guárdala en el ordenador. Después, crea un archivo \*.txt en una columna con la lista de palabras.

Una vez realizados estos dos pasos, en el menú inicial de JFLAP pincha en **Batch > Batch Test**. Se abrirá una ventana para que indiques donde se encuentra el archivo que has guardado con la gramática, y luego otra para que cargues el fichero con las palabras.

Acto seguido aparecerá una nueva ventana como la de la Figura 16. Pinchando en el botón **Run Inputs**, JFLAP comprobará para todas las palabras del fichero las que son generadas por la gramática y las que no lo son, informando del resultado. Entre paréntesis aparece la solución esperada (por defecto, JFLAP espera que todas las palabras cargadas de fichero sean válidas).

Si seleccionamos con el ratón una de las palabras, y pulsamos en el cuadro de la izquierda en **Start** y **Step**, aparecerá el árbol de derivación, como en el derivador por fuerza bruta.

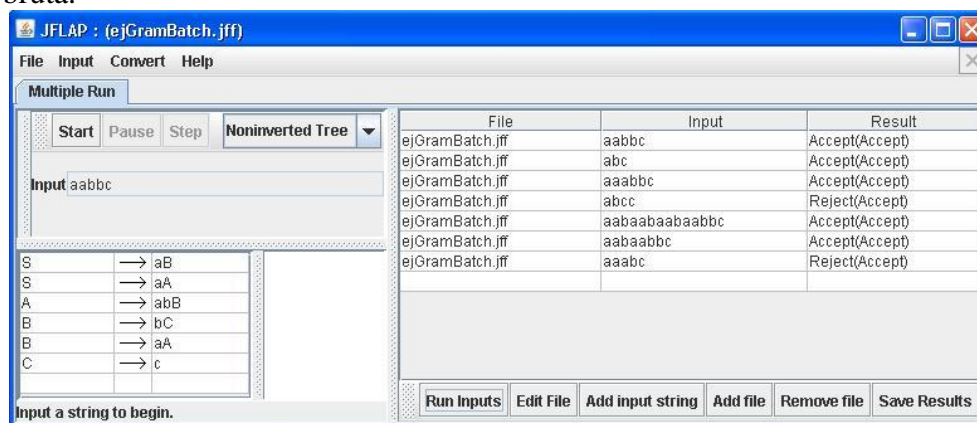


Figura 16: ejemplo de uso del modo batch

Desde esa misma ventana, podremos: editar la gramática (**Edit File**); añadir más palabras, especificando el resultado que esperamos obtener (**Add input string**); cargar otro fichero con otra gramática, y probar la misma lista de palabras para dos gramáticas distintas (**Add file**); guardar los resultados (**Save Results**); y otras operaciones, algunas de las cuales veremos más adelante, como limpiar la gramática, pasar a autómata finito, etc.

## 4. Limpieza de gramáticas

A veces es ventajoso transformar una gramática en otra equivalente con un formato más simple. Sobre todo a la hora de aplicar algún algoritmo sobre la gramática (como el Brute Force Parser), para que la operación sea más rápida y eficiente.

JFLAP tiene implementados cuatro algoritmos de limpieza de gramáticas. Las tres primeras transformaciones eliminan las producciones  $\lambda$  (reglas no generativas), las reglas de redenominación (tipo  $A \rightarrow B$ ) y las reglas superfluas, respectivamente, y la cuarta transforma la gramática resultado de lo anterior a una gramática en Forma Normal de Chomsky (FNC). JFLAP requiere que esas cuatro transformaciones se realicen en este orden, ya que al eliminar las producciones  $\lambda$ , pueden aparecer reglas de redenominación, y eliminar estas puede generar reglas superfluas. JFLAP se saltará alguno de estos pasos en caso de que no sea necesario realizarlo.

Sólo se consideran gramáticas de contexto libre que no deriven en  $\lambda$ , porque la palabra vacía sólo aparece en casos especiales y es fácil eliminarla del lenguaje, y añadirla de nuevo a la gramática resultante después de la transformación.

Dado que jflap realiza la limpieza de las gramáticas con algoritmos con otras denominaciones, se adjunta a continuación una tabla comparativa.

Algoritmo	jflap	Teoría dada en clase	Diferencia	Comentarios
Reglas innecesarias	Unit production removal	Eliminación de Reglas innecesarias	No hay	-----
Símbolos inaccesibles	Useless production removal (2ª parte)	Eliminación de símbolos inaccesibles	No hay	-----
Reglas superfluas	Useless production removal (1ª parte)	Eliminación de reglas superfluas	No hay	-----
Símbolos no generativos	Useless production removal (1ª parte)	Eliminación de símbolos no generativos	En clase se proporciona un algoritmo distinto. Poner el presunto símbolo no generativo como axioma: si se obtiene lenguaje vacío, entonces es no generativo	El símbolo no generativo, si está en la parte derecha de una regla de gramática G2, G3 puede eliminarse tratando esta regla como superflua.
Reglas no generativas	$\lambda$ -production removal	Eliminación de reglas no generativas	No hay	-----
Reglas de redenominación	Unit production removal	Eliminación de Reglas de redenominación	No hay	-----



## 4.1 Eliminando reglas no generativas (Lambda removal)

La idea es identificar las producciones que deriven en  $\lambda$ , eliminarlas, y sustituirlas con producciones equivalentes.

Una regla no generativa es  $A ::= \lambda$ , siendo  $A$  un no terminal y  $A \neq S$ . Si  $\lambda \in L(G)$  se añade  $S ::= \lambda$  y  $\forall A \in \Sigma_{NT}$  ( $A ::= \lambda$   $A \neq S$ ) y  $\forall$  regla de la gramática de la forma  $B ::= xAy$ , añadiremos a la lista de las producciones una regla de la forma  $B ::= xy$ , excepto en el caso de que  $z=y=\lambda$ .

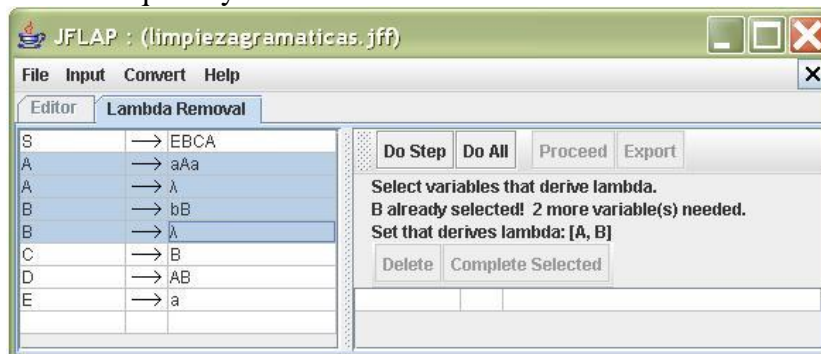


Figura 11: Eliminación de una producción  $\lambda$

### La transformación

Después de introducir una gramática con el editor, selecciona **Convert > Transform Grammar**. Aparecerá una ventana como la de la Figura 11. La transformación se realiza en dos pasos:

- El primero consiste en identificar el conjunto de producciones que derivan en  $\lambda$ . En el ejemplo son las producciones 3ª y 5ª de  $A$  y  $B$ , respectivamente. Pincha en cada una de las producciones de  $A$  ( $A \rightarrow aAa$  y  $A \rightarrow \lambda$ ), y lo mismo con  $B$ , para añadir  $A$  y  $B$  al conjunto de no terminales que derivan en  $\lambda$ , tal y como se muestra en el ejemplo, de la Figura 11.

¿ $C$  puede derivar en  $\lambda$ ? No es algo obvio, como en el caso de  $A$  y  $B$ . Sin embargo, la parte derecha de  $C \rightarrow B$  es un elemento del conjunto de no terminales que derivan en  $\lambda$ , lo que implica que  $C$  también deriva en  $\lambda$ . Pincha en la producción  $C \rightarrow B$  para añadirla al conjunto.

Las producciones que sólo tengan en la parte derecha elementos que pertenezcan al conjunto de no terminales que derivan en  $\lambda$  (como  $D \rightarrow AB$ ), también se incluyen (haz clic sobre  $D$  para añadirlo del mismo modo). En este punto el conjunto se ha completado y JFLAP va al siguiente paso.

- El segundo paso es obtener una gramática equivalente sin producciones  $\lambda$ . La gramática original ahora aparece en la parte derecha de la ventana. Ahora tendremos que borrar las producciones  $\lambda$ , y añadir otras nuevas en esta parte de la ventana. Para cada producción de la gramática original que en la parte derecha tuviera un no terminal del conjunto que hemos obtenido antes (los que derivan en  $\lambda$ ), se tiene que agregar una nueva producción sin ese símbolo, excepto si la nueva producción deriva en  $\lambda$ . En el ejemplo, para  $A \rightarrow aAa$  añadimos  $A \rightarrow aa$  y en el caso de  $D \rightarrow AB$ , añadimos  $D \rightarrow A$  y  $D \rightarrow B$ .

Una forma alternativa de añadir las producciones es seleccionando primero la producción en la gramática original de la izquierda y haciendo clic en **Complete Selected**. Otra forma más rápida es pulsando el botón de **Do Step** o **Do All**.

Cuando se complete el proceso aparecerá un mensaje de aviso en la ventana. A partir de aquí, puedes pasar a la siguiente transformación pulsando en **Proceed**, o puedes exportar primero la gramática modificada a una nueva ventana pulsando **Export**. En cualquier caso, la gramática modificada es equivalente a la original.

## 4.2 Eliminando reglas de red denominación (Unit removal)

Una regla de red denominación es una regla del tipo  $A::=B$ . Estas producciones no añaden terminales en la derivación, y no incrementan la longitud de la forma sentencial. Además, pueden causar problemas si se crea un ciclo ( $A::=B$  y  $B::=A$ ).

Por lo tanto, hay que identificar estas reglas, eliminarlas y añadir otras que las reemplacen en una gramática equivalente. Si hay alguna regla de este tipo, entonces  $\forall A \in \Sigma_{NT} \mid A::=B$  en la gramática y  $\forall (B::=x) \in P$  donde  $x \notin \Sigma_{NT} \Rightarrow P' = P + \{A::=x\}$ .

Usaremos un gráfico de dependencia de variables (VDG, donde los no terminales son nodos) para representar todas las reglas de red denominación que hay y las direcciones de las producciones (si  $A \rightarrow B$ , entonces hay una flecha del nodo  $A$  al  $B$ ).

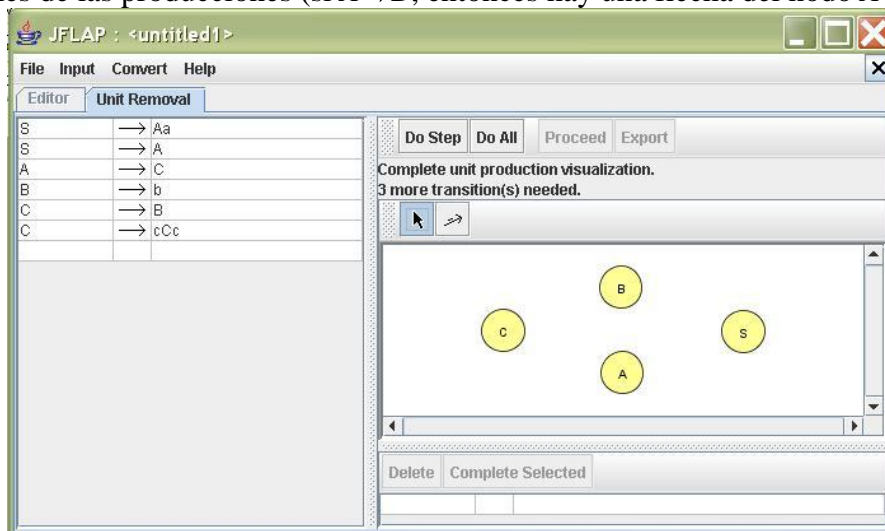




Figura 12: Eliminación de reglas de red denominación

### La transformación

En este caso, probaremos con un ejemplo distinto al anterior. Para llegar a la ventana que se muestra en la Figura 12, se deben seguir los pasos explicados en 4.1. (en este caso, como no hay producciones  $\lambda$ , JFLAP se salta ese paso de la limpieza de gramáticas).

- El primer paso es completar el VDG, que JFLAP crea a partir de la gramática. El gráfico representa las reglas de red denominación como transiciones entre los nodos. Para cada regla (Ej:  $S \rightarrow A$ ), se debe añadir una flecha entre los nodos etiquetados que correspondan (del  $S$  al nodo  $A$ ). Los nodos se pueden mover usando el botón , y los enlaces se ponen con el botón , haciendo clic del nodo que representa la parte izquierda ( $S$ ) y arrastrando hasta el que representa la derecha ( $A$ ) (ver Figura 13).
- Una vez completo el grafo, pasamos al siguiente paso, que es borrar las reglas de red denominación y añadir producciones equivalentes. Para la producción  $A \rightarrow B$  de la gramática original, añadiremos una producción del tipo  $A \rightarrow w$ , por cada regla  $B \rightarrow w$  existente. En el caso de que  $w$  sea un único símbolo no terminal,  $w$  pasa a ser el valor que tomará para las reglas  $X \rightarrow w$ .

Para el caso del ejemplo, primero eliminaremos la producción  $S \rightarrow A$  que es la única con el axioma, seleccionando la regla, y pulsando en Delete. Añade las nuevas producciones según la explicación anterior y repite el proceso para los demás casos.

Otra forma de hacerlo es pulsando en **Complete Selected** o utilizando **Do Step** o **Do All**.

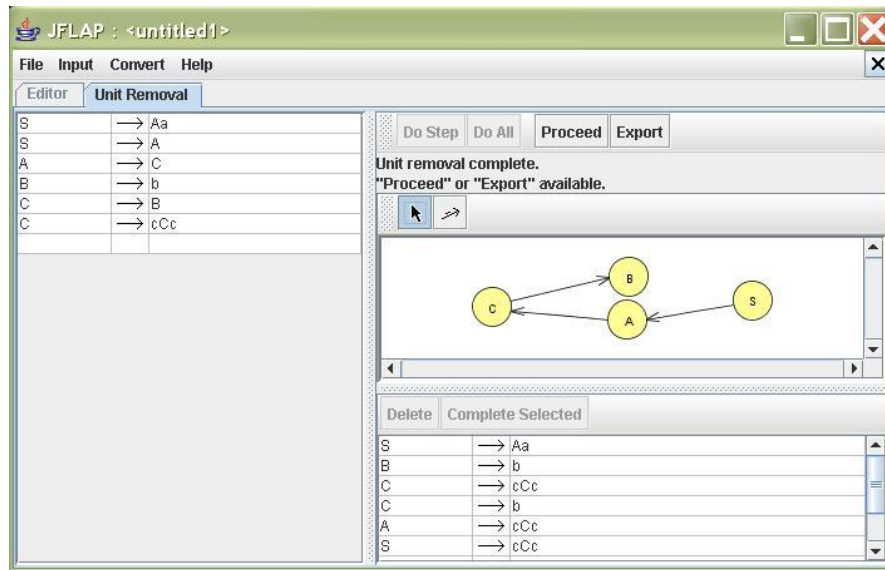


Figura 13: Eliminación completa de reglas de redenominación

Cuando se haya terminado el proceso, podemos seguir con la transformación de la gramática pulsando en **Proceed** o exportar la nueva gramática, equivalente a la anterior, pulsando en **Export**.

### 4.3 Eliminación de reglas superfluas y símbolos inaccesibles (Useless removal)

En este apartado eliminaremos las producciones que no contribuyen a la formación de palabras  $x \in \Sigma_T^*$ .

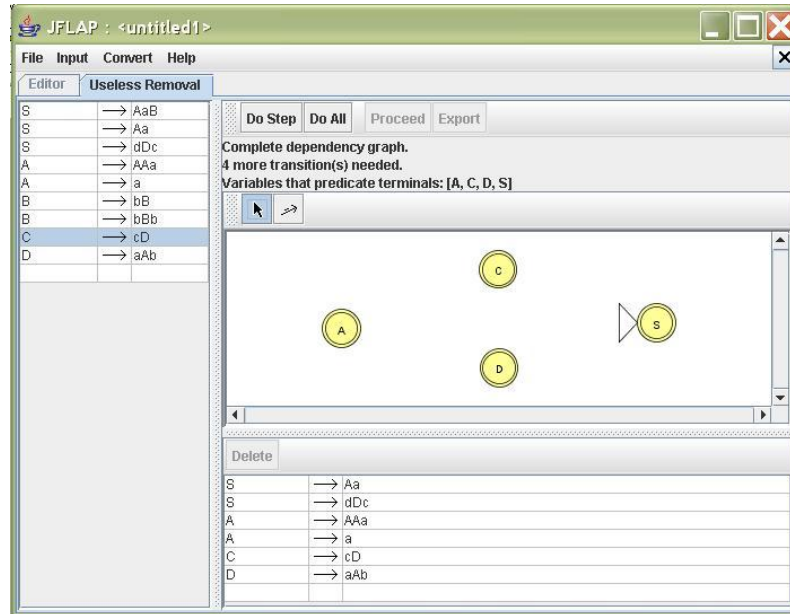


Figura 14: Ejemplo de eliminación de reglas superfluas y símbolos inaccesibles, con el grafo de dependencia completo.

#### La transformación

Si introducimos el ejemplo de la Figura 14, al ir a transformar la gramática, JFLAP se saltará los dos primeros pasos que hemos visto anteriormente porque en este caso no son necesarios.

El proceso tiene dos partes:

- La primera es encontrar los no terminales que no derivan en una cadena de terminales y eliminar las producciones con esos símbolos (reglas superfluas). Para ello, se realiza el proceso inverso, identificando las producciones tipo  $A \rightarrow a$ , que derivan sólo en elementos terminales. Haz clic en esas producciones, para que se agreguen al conjunto de no terminales que derivan en terminales.

Una vez hecho esto, hay que marcar los no terminales para los que existe una regla  $U ::= x$  donde  $x \in \Sigma^*$  ( $x$  es una cadena de terminales, o contiene no terminales que ya se han marcado anteriormente). En el ejemplo, cuando hayamos marcado todos, veremos algo similar a la Figura 14. En la gramática del recuadro inferior derecha de la ventana, vemos como las producciones que contenían el no terminal 'B' han sido eliminadas, por ser superfluas.

- La segunda parte de la transformación es encontrar los símbolos inaccesibles desde el axioma y eliminar las producciones con esos elementos. En el ejemplo, si añadimos las flechas de las producciones al gráfico, vemos como no se puede acceder al elemento 'C' desde el axioma. Por lo que esa regla se puede eliminar (seleccionala en el editor y pulsa en **Delete**). La gramática queda tal y como se muestra en la Figura 15.

La gramática que tenemos ahora es equivalente a la anterior, más pequeña, una vez que se han eliminado las producciones que no eran necesarias. Llegado este punto, está preparada para pasarla a FNC.

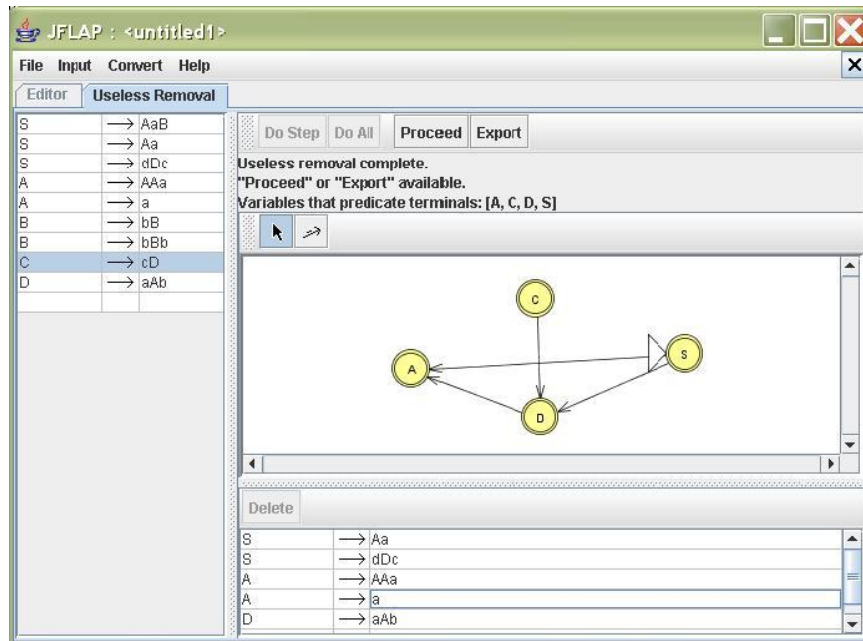


Figura 15: Ejemplo con la transformación completa 4.3



#### 4.4 Convertir a Forma Normal de Chomsky.

La FNC es una forma restringida de una gramática de contexto libre. En la FNC, la parte derecha de una producción es o un elemento terminal, o dos elementos no terminales. De esta forma, el proceso de derivación es más rápido.

La idea de esta conversión es reemplazar cada conjunto de terminales en la parte derecha de longitud superior a 1, por un no terminal. Entonces reemplazar cada par de no terminales adyacentes por otro no terminal, hasta que todas las partes derechas de las producciones con no terminales tengan exactamente dos no terminales. Pero primero todas las producciones  $\lambda$ , reglas de red denominación y superfluas, que hubiera tienen que ser eliminadas, según lo que hemos visto anteriormente.

##### La transformación

Escribe en el editor de gramáticas de JFLAP el ejemplo que se muestra en la Figura 16. Selecciona **Convert > Transform Grammar**, y aparecerá una ventana como la de la figura.

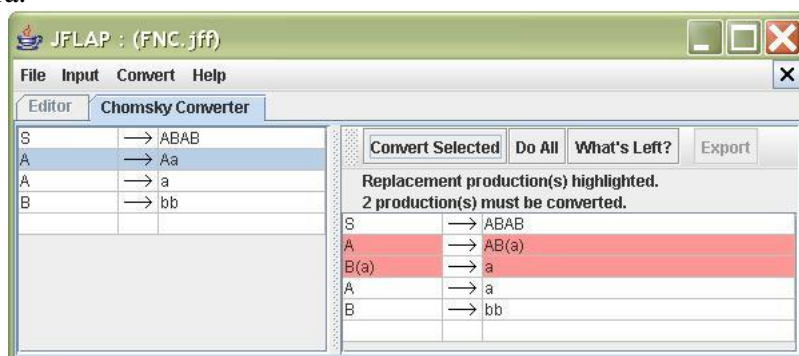


Figura 16: Ejemplo de transformación a FNC

Primero convertiremos las producciones más simples. Selecciona  $A \rightarrow Aa$  (la 2ª producción) de la tabla de la parte derecha. Esta producción necesita ser reemplazada por dos producciones equivalentes que estén en FNC. Haz clic en **Convert Selected**, para modificarla. En este caso,  $B(a)$  es un nuevo no terminal representada por el terminal a, como una nueva producción  $B(a) \rightarrow a$ .

Ahora selecciona  $B \rightarrow bb$  y haz clic en **Convert Selected**. En este caso sólo aparece una nueva variable  $B(b)$ , con  $B \rightarrow B(b)B(b)$ .

Sólo queda una producción por convertir. Selecciona  $S \rightarrow ABAB$ . La parte derecha es muy larga, por lo que se necesitan pares de no terminales adyacentes para sustituir por variables nuevas, hasta que todas las producciones nuevas tengan la longitud correcta. En JFLAP esos no terminales nuevos serán  $D(n)$  donde  $n$  es un número que comienza en 1. Haz clic en **Convert Selected** para dividir esta producción en dos:  $S \rightarrow AD(1)$  y  $D(1) \rightarrow BAB$ . Esta última regla necesita ser fragmentada en dos. Selecciónala y pulsa el botón anterior para dividirla en  $D(1) \rightarrow BD(2)$  y  $D(2) \rightarrow AB$ . Ahora todas las reglas tienen la forma adecuada, y la conversión se ha completado, como se puede ver en la Figura 17.

En este punto, los no terminales de la forma  $B(x)$  y  $D(n)$  son temporalmente no terminales utilizados solo en el algoritmo de conversión. Cuando la gramática sea exportada, serán reemplazados por letras en mayúsculas del abecedario que no se estuvieran usando ya. Si la gramática reformada tiene más de 26 no terminales, JFLAP no podrá exportarla porque será más grande que el número de letras del abecedario inglés.

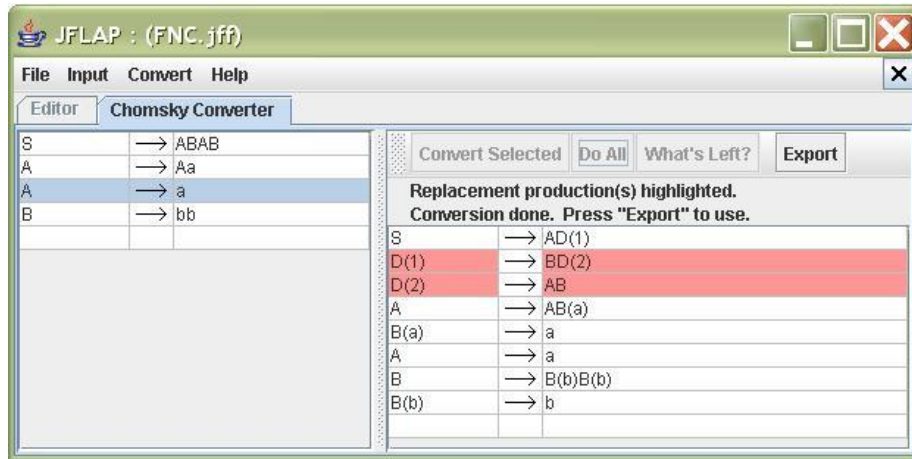


Figura 17: Ejemplo completo de transformación a FNC

Ahora compara la gramática en FNC con la original, ejecutando el brute-force parser con la palabra *aaabbaabb* para los dos casos. ¿Cuál de las dos ejecuciones es más lenta? ¿Qué árbol de derivación será más pequeño?

## 5. Convertir una gramática lineal derecha a un autómata finito

En esta sección describiremos la conversión que puede hacer JFLAP de una gramática lineal derecha a su equivalente autómata finito (AF). La gramática y el autómata reconocerán el mismo lenguaje regular.

La idea detrás de esta transformación es que los no terminales en una gramática lineal derecha pueden funcionar de forma similar a los estados de un AF. Como ejemplo, considera la gramática de la Figura 2 de este manual, y observa la derivación con la palabra *aqvx* que aparece en la Figura 4 y 5. Empezamos la derivación con el axioma, lo que se asemeja con el estado inicial del autómata. Cuando vamos a la producción  $S \rightarrow aA$ , es similar a leer una 'a' y movernos al estado A. El siguiente paso de la derivación sería  $A \rightarrow B$ , lo que es parecido a transitar del estado A al B, sin leer nada, o dicho de otra forma, como si leyéramos  $\lambda$ . Después, la producción  $B \rightarrow qvC$  es como si del estado B fuéramos al C, una vez leído *qv*. Finalmente, la última producción  $C \rightarrow x$ , es análoga a leer *x*, y pasar al estado final desde C, ya que no hay más derivaciones.

De forma genérica, con  $\Sigma_{NT}=\{A,B\}$  y  $\Sigma_T=\{\alpha,\beta\}$  podemos decir que una producción  $A \rightarrow \alpha B$  es como una transición en el AF desde el estado A al B, leyendo  $\alpha$ . Y una producción  $A \rightarrow \beta$  es como una transición del estado A, al estado final del autómata, leyendo  $\beta$ .

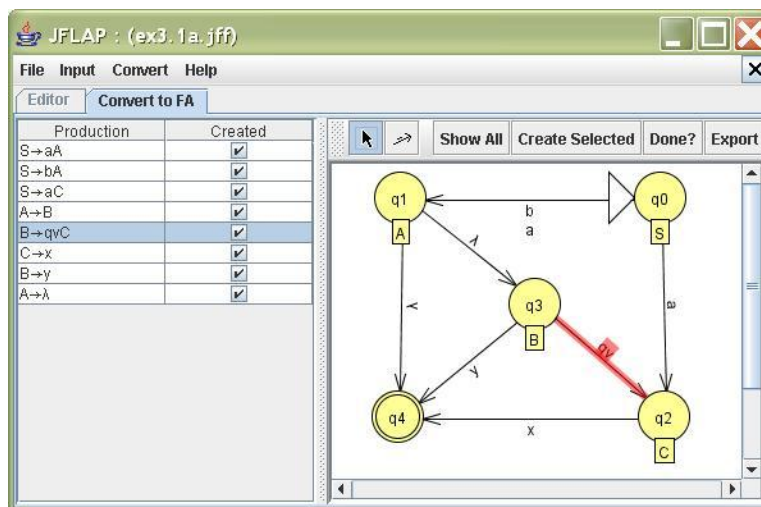


Figura 18: Conversión finalizada a un AF.

La Figura 18 muestra la conversión finalizada de la gramática que hemos usado de ejemplo a un AF. Para ir al conversor, una vez cargada la gramática a transformar, selecciona en el menú **Convert > Convert Right-Linear Grammar to FA**. La ventana que aparecerá será similar a la de la figura, con los estados del autómata marcados con los símbolos no terminales equivalentes, exceptuando el estado final, que no tiene.

La finalidad del conversor es crear las transiciones del AF según cada una de las producciones de la gramática a transformar. La primera producción que convertiremos será  $S \rightarrow aA$ . Crea una transición del estado S al A con el terminal *a*, y cuando lo hagas verás como se marca el cuadrado de la columna **Created**, para indicar que ya se ha marcado la transición. Realiza este mismo proceso con las demás producciones de la gramática.

JFLAP gestiona algunos errores que pueden aparecer en el conversor. Como prueba de ello, intenta introducir una transición errónea, como mover del estado  $C$  al  $A$ , leyendo  $xyz$ . No existe ninguna producción  $A \rightarrow xyzS$ , por lo que esta transición no pertenece al AF. Cuando termines de editar la transición, una ventana de aviso aparecerá para decirte que esa transición no está en la gramática que estamos convirtiendo.

Si pinchas en el botón **Done**, JFLAP te avisará en caso de que no se haya completado la conversión, especificando el número de transiciones que faltan por añadir, y resaltando en rojo las producciones concretas de la gramática que falten por aplicar al AF.

Una forma más rápida de crear las producciones, es seleccionar la fila de la gramática con la producción y pinchar en el botón **Create Selected**. JFLAP creará la transición correspondiente automáticamente. También se puede pulsar en **Show All** para completar toda la conversión.

Cuando hayas terminado todo el proceso, pulsa el botón **Export**, y tu autómeta finito determinista aparecerá en una nueva ventana. Después de exportarlo, puedes modificar el AF.

## 6. Convertir un AF en una gramática lineal derecha

Ahora veremos el proceso contrario a lo visto en el punto 5: pasar de AF a gramática. La idea es la misma que antes, pero a la inversa: si en la conversión anterior una producción  $A \rightarrow xyzB$ , era una transición del estado  $A$  al  $B$ , con  $xyz$ ; ahora, una transición del estado  $A$  al  $B$ , leyendo  $xyz$ , se traduce en una producción  $A \rightarrow xyzB$ .

Al principio, la herramienta asignará a cada estado un no terminal, fijando el estado inicial como el axioma  $S$ . El caso de las producciones ya lo conocemos. Por ello, la parte donde hay que tener cuidado es en la conversión del estado final.

En el punto 5, el estado final no correspondía a ningún elemento de la gramática. Era el destino de las transiciones resultantes de las producciones que no tenían no terminales en la parte derecha, lo que implicaba el final de la palabra. Sin embargo, ahora es diferente: puede haber transiciones desde el estado final a otro estado cualquiera del autómata. Esto significa que en la conversión a la gramática, un estado final deber corresponder a un no terminal mientras la expansión de la cadena sea posible. Por lo tanto, para cada estado final  $F$ , se deberá incluir la producción  $F \rightarrow \lambda$ .

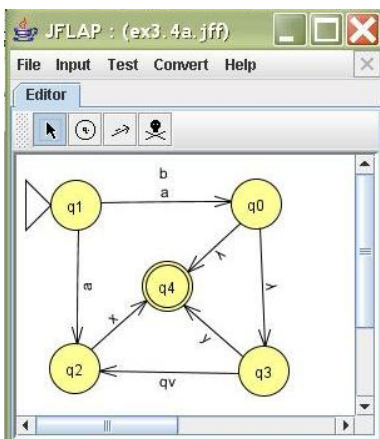


Figura 19: Autómata finito inicial.

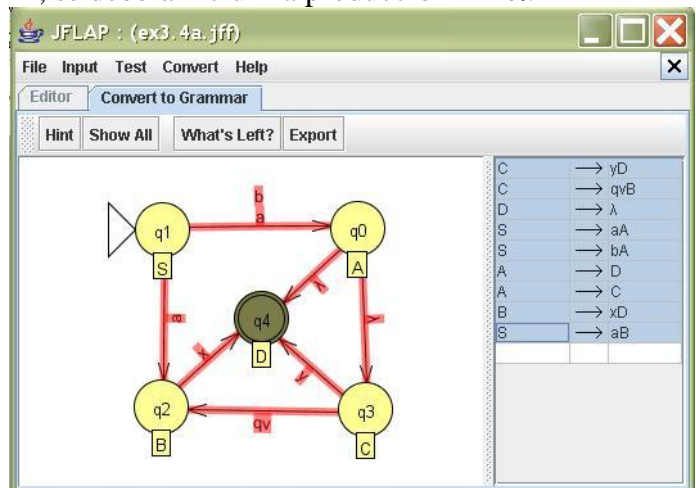


Figura 20: Ejemplo de conversión de un AF.

Para la conversión de un AF a una gramática, desde la ventana de autómatas finitos, seleccionar **Convert > Convert to Grammar**.

El método es muy sencillo. Cada transición y cada estado final corresponden a una producción en la gramática. Cuando selecciones una transición o estado final, su producción aparecerá en la parte derecha de la ventana (ver Figuras 19 y 20).

Pincha en cada transición del esquema del autómata, y también en el estado final  $q4$ , lo que producirá  $D \rightarrow \lambda$ . Al final, deberá quedar algo similar a lo que aparece en la Figura 20.

El botón **Hint**, mostrará una producción para un elemento que no ha sido añadido aún a la tabla de la derecha. Si seleccionas este botón, una producción se añadirá, y la parte correspondiente del esquema del AF se sombreadrá en color rojo.

El botón **What's Left**, remarcará los objetos que queden que aún no se han convertido. Una forma de hacer la conversión de una vez para todo el autómata es pulsando en el botón **Show All**. Al final pulsa en el botón **Export** para obtener en una ventana a parte la nueva gramática obtenida a partir del autómata.


## 7. Autómatas Finitos

En este capítulo construiremos un autómata determinista en JFLAP, ilustrando algunos métodos de simulación, etc... En las secciones 7.1-7.4, mostraremos la definición estándar de AFD, y en la 7.5 veremos como JFLAP maneja una definición más general de un AFD con múltiples transiciones de caracteres.

### 7.1 Un autómata finito simple


Para empezar a construir un AF, pincha en el menú de inicio de JFLAP en el botón **Finite Automaton**. Aparecerá una ventana, con un menú, una pestaña que es el Editor, una barra de herramientas y un área en blanco que ocupará casi toda la ventana.

#### 7.1.1 Crear estados

Un autómata está formado por un conjunto de estados. Antes de crear uno, primero debes activar el botón , en la barra de herramientas. Mientras esté seleccionado, el botón estará sombreado.


La zona grande en blanco, llamada *canvas*, es donde construiremos el autómata. Ahora que la herramienta para crear estados está activada, pincha en el canvas para crear un estado. Aparecerá en la zona donde hayas pinchado. JFLAP los identificará en el orden en el que se creen como  $q_0$ ,  $q_1$ ,  $q_2$ ...

#### 7.1.2 Definir el estado inicial y el final

Todos los autómatas requieren un estado inicial y un conjunto de estados finales. Para marcar  $q_0$  como el estado inicial, selecciona el botón  y después, con el botón derecho del ratón pincha en el estado  $q_0$ . Aparecerá un menú sobre el estado del autómata (el resto de las opciones del menú se explicarán en la sección 7.1.5), en el que escogeremos **Initial**. Ahora el estado  $q_0$  queda marcado con una flecha que indica que es el inicial. Igualmente, realiza el mismo procedimiento con  $q_3$  pero seleccionando **Final** en el menú. El estado  $q_3$  quedará con un doble círculo.

Para modificar los estados iniciales/finales, se realiza el mismo procedimiento, volviendo a escoger **Initial/Final** en el menú, para eliminar la marca de check que habíamos puesto antes.

#### 7.1.3 Creando transiciones

En el ejemplo, hay 3 transiciones. Para crearlas, pincha en el botón . Una vez seleccionado, pincha en el estado  $q_0$ , y sin soltar el botón, arrastra el cursor del ratón hacia el estado  $q_1$  y después, suéltalo. Un campo de texto aparecerá entre los dos estados. Escribe una 'b' y dale a intro. Una nueva transición b del estado  $q_0$  a  $q_1$  apareció. De la misma forma, crea el resto de transiciones para que el autómata quede igual que el de la Figura 21. Para transiciones  $\lambda$ , deja el campo de texto en blanco.

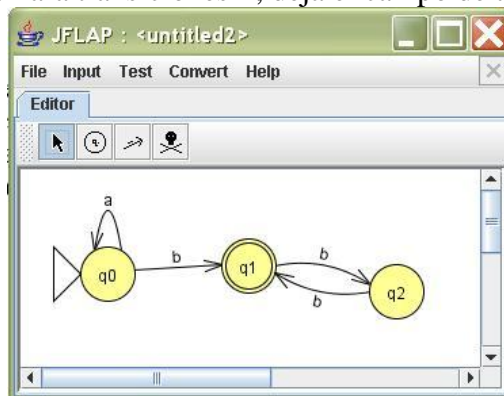


Figura 21: Ejemplo del editor de autómatas finitos.





Para los loops, simplemente pincha sobre el estado en el que quieras hacer el loop y suelta el ratón. Aparecerá la ventana donde deberás escribir el texto.

Si quieres cancelar en algún momento una transición, o una edición, pulsa escape.


#### 7.1.4 Borrando estados y transiciones

Si te confundes y creas estados o transiciones de más, borrarlos es muy sencillo.

Selecciona el botón , y haz doble clic sobre el elemento que quieras eliminar. En el caso de las transiciones puedes pinchar en la misma flecha o en la etiqueta. En los estados, cuando se borra uno, se eliminarán automáticamente todas las transiciones que entren o salgan del estado.

Para no eliminar algo por error, selecciona el botón  de nuevo, y así ya no estará activada la herramienta de borrado.

#### 7.1.5 Herramienta de editor

En el punto 7.1.2 ya se ha explicado alguna función para el botón , pero tiene otras muchas funciones relacionadas a la modificación de los atributos de los estados existentes y de las transiciones:

- Marcar estados como inicial/final: esto ya lo hemos visto en la sección 7.1.2
- Mover estados y transiciones: pincha sobre el elemento que desees mover y arrástralo a la nueva localización. Si mueves una transición, moverás también los estados implicados.
- Editar transiciones existentes: haz clic sobre la transición que quieras modificar, y cambia lo que había antes en el campo de texto.
- Etiquetas: Si pinchas con el botón derecho del ratón sobre un estado, veras en el menú **Change Label**. Al seleccionar esa opción, aparecerá una nueva ventana donde podrás escribir la nueva etiqueta. Estas etiquetas ayudan a identificar el significado del estado.

En el ejemplo de la Figura 21, cuando estemos en q2, será cuando llegue al autómata un número par de b's. En q1 tendremos un número impar. Por ello, podríamos etiquetar q2 con “nº par de b's” y q1 con “nº impar de b's”.

Para borrar una etiqueta existente, haz clic en **Clear Label**, y para borrar todas las que haya, pincha en **Clear All Labels**.

Si pinchas con el botón derecho, en una zona en blanco, aparecerá un menú diferente con la opción **Display State Labels**. Esta opción está activada por defecto. Si la desactivas, las etiquetas serán invisibles. Las podrás ver, posicionando el cursor del ratón durante un par de segundos sobre un estado en el que hubieras añadido previamente una etiqueta.

- Disposición automática: pulsa el botón derecho en una zona en blanco. Observa que hay un elemento en el menú, que pone **Layout Graph**. Cuando está seleccionado, JFLAp aplicará un algoritmo de ordenación gráfico sobre el autómata. Esta opción es útil sobre todo, cuando hay un gran número de estados y transiciones, ya que JFLAP los colocará para que el autómata se vea de la forma más sencilla posible, ahorrándonos el proceso, a veces tedioso, de ir moviendo cada elemento por separado.

Nota: Existen comandos para ejecutar algunas de estas opciones más rápido. Por ejemplo, manteniendo el cursor del ratón sobre el icono de la herramienta de creación de estados, aparecerá **(S)tate Creator**. El paréntesis que encierra a la **S** indica que esa es la tecla rápida para crear estados. Pulsando la letra correspondiente (en minúsculas), activaremos la opción deseada sin necesidad de usar el ratón.

## 7.2 Simulación de una entrada en el autómeta

En esta sección estudiaremos tres de los métodos de JFLAP para simular el resultado de una entrada en un autómeta: simulación por pasos, simulación rápida y simulación múltiple. El cuarto tipo “stepping by state” se verá brevemente en 7.3.

### 7.2.1 Simulación por pasos (*Stepping simulation or Stepping with closure*)

La simulación por pasos te permite ver cada configuración generada por un autómeta al procesar una entrada. La siguiente figura muestra un ejemplo de una simulación por pasos de la entrada *aabbb* en el autómeta de la Figura 21. Para seleccionar esta herramienta, ve a **Input > Stepping with closure**. Te aparecerá una ventana, donde tendrás que introducir la entrada del autómeta.

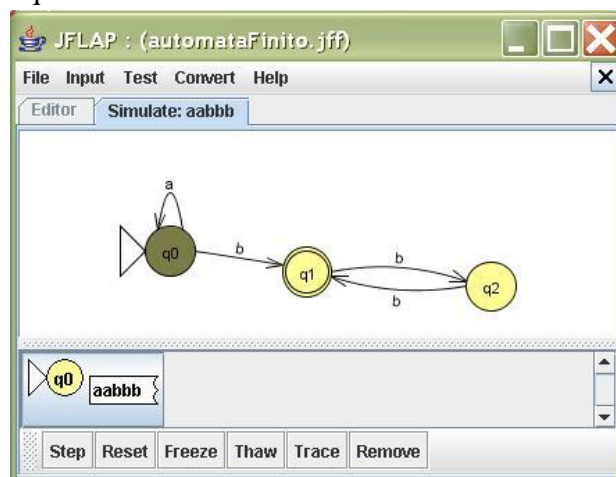


Figura 22: Inicio de la simulación por pasos con la entrada *aabbb*.

En la Figura 22 vemos como el estado activo del AF está sombreado. La zona por debajo del dibujo del AF muestra la configuración actual. Comenzamos en el estado inicial *q0*, y al pulsar **Step** las letras de la palabra de entrada se irán sombreado, a la vez que cambiará el estado actual, según sea la entrada recibida en cada caso. Si la palabra es aceptada, el dibujo con la configuración actual tendrá un fondo verde. Si no es aceptada por el AF, el fondo será rojo.

Algunas de las operaciones de la barra de herramientas que se encuentra en la parte baja de la ventana (Figura 22) solo funcionan cuando hay una configuración seleccionada. Para seleccionarla, pincha en ella. Puedes ver como al hacerlo, el color se oscurece un poco. Si pinchas ahora en **Remove**, eliminarás la configuración actual. Para empezar de nuevo con la simulación, pincha en **Reset**.

Para ver todos los pasos anteriores que se han dado para llegar a una configuración, selecciónala y después pincha en **Trace**. Se abrirá una nueva ventana con todo el proceso desde la configuración inicial, arriba del todo, a la actual.

### 7.2.2 Simulación rápida (*Fast Simulation*)

Esta simulación permite ver rápidamente si el autómeta acepta o no una entrada. Si la acepta, aparecerá la lista con los pasos dados hasta llegar al final (similar a obtener la traza en el punto anterior). En caso contrario, aparecerá un mensaje para informar.

Selecciona **Input > Fast Run**. Escribe *aabbb* como entrada al autómeta y el resultado que te dará JFLAP será como el de la Figura 23.

Observa los dos botones de la ventana. **I'm Done** cerrará la ventana. **Keep Looking** será útil para autómetas no deterministas y eso lo veremos en la sección 7.3.2.

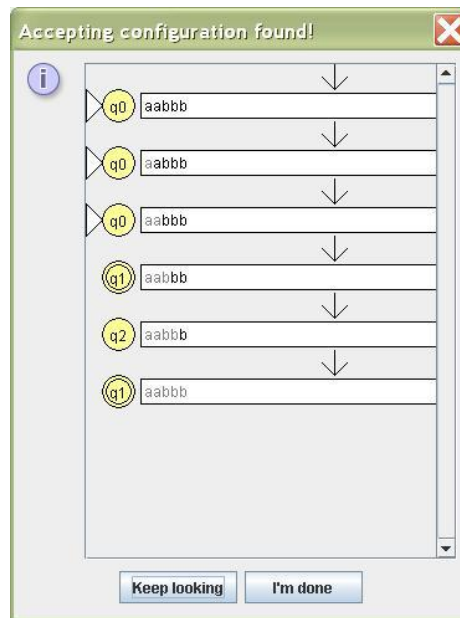


Figura 23: Resultado de una simulación rápida.

### 7.2.3 Simulación múltiple

Este método permite realizar funcionamientos múltiples de una vez, rápidamente. Selecciona **Input > Multiple Run**. Aparecerá una ventana similar a lo que se muestra en la Figura 24.

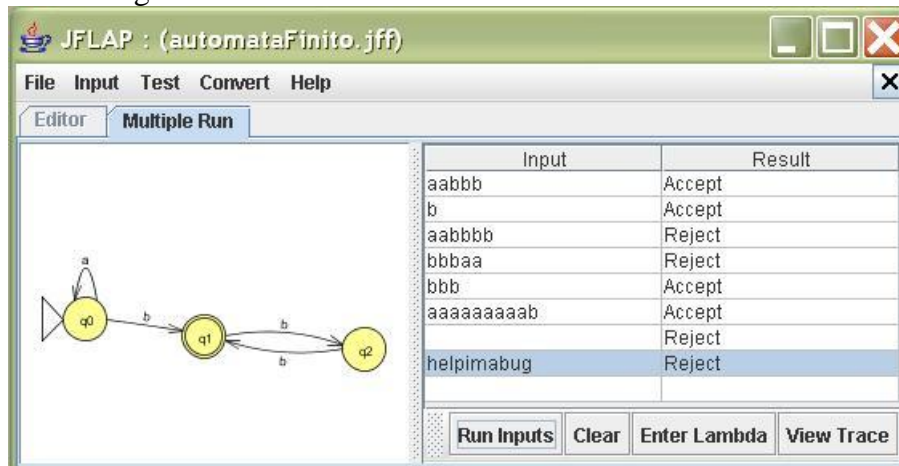


Figura 24: Ejemplo de simulación múltiple. En la fila 7 de inputs se introdujo lambda.

El procedimiento es similar al del derivador múltiple por fuerza bruta que utilizamos en gramáticas. Escribe las entradas que quieras probar en la columna de inputs, como se muestra en el ejemplo, y pulsa en **Run Inputs**. La columna de la derecha mostrará los resultados obtenidos.

La lista de inputs que introduzcas, JFLAP los recordará la próxima vez que abras la simulación múltiple, mientras no reinicies el programa.

### 7.3 No determinismo

En esta sección trataremos los autómatas finitos no deterministas (AFND) en JFLAP, utilizando el autómata que se muestra en la Figura 25 como ejemplo.

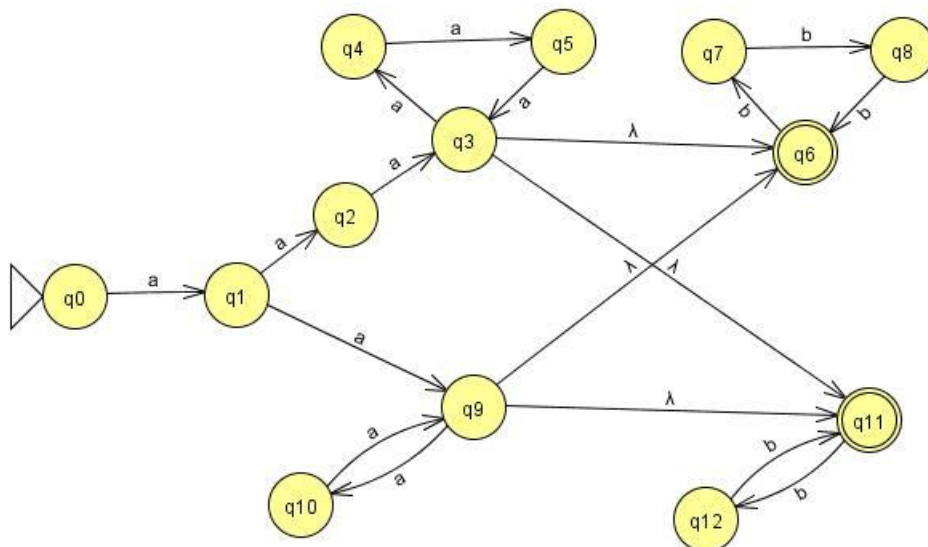


Figura 25: AFND que acepta el lenguaje  $a^n b^m$ , con  $n > 0$ ,  $m \geq 0$ , y  $n, m$  divisibles por dos y por tres.

Hay dos condiciones que implican que un AF sea no determinista. La primera condición es que el AF tenga dos transiciones del mismo estado que tengan el mismo símbolo de entrada (en la figura q1 tiene dos transiciones que leen 'a'). La segunda es que alguna transición lea  $\lambda$ .

#### 7.3.1 Crear un autómata no determinista

El proceso es básicamente el mismo que para un autómata determinista. La mayor diferencia es la existencia de transiciones  $\lambda$ . Para crear una de ellas, solo tienes que crear una transición normal, pero dejando el campo de texto en blanco.

Ahora fíjate en la Figura 25, y en un editor nuevo de autómatas haz el mismo esquema. Recuerda que los estados se numeran según los vayas creando, así que ten cuidado en que estén colocados en el mismo orden.

#### 7.3.2 Simulación

Durante la simulación, una entrada en una máquina determinista produce un solo camino de configuraciones, mientras que en una máquina no determinista, los caminos son múltiples. Los simuladores de JFLAP diferencian ambos casos:

- Simulación por pasos, (Step with Closure): Selecciona **Input > Step with Closure** e introduce la palabra 'aaaabb'. Después de esto, verás el familiar simulador por pasos, con una configuración inicial en q0, con toda la palabra esperando por ser procesada. Pulsa **Step** una vez para mover a q1. Sin embargo, si pulsas una segunda vez, verás como la forma del simulador cambia, como se muestra la Figura 26.

Hay 4 configuraciones en el simulador porque es un AFND: la última configuración estaba en q1, y faltaba por leer aaabb, y q1 tiene transiciones a al estado q2 y q9. Pero al estar conectados esos estados con transiciones  $\lambda$ , la simulación por pasos no solo conecta con q1, sino también con aquellos que leen la palabra vacía desde q1. El conjunto de estados alcanzables desde q1 con transiciones  $\lambda$ , es llamado "closure" (encierro) de q1.

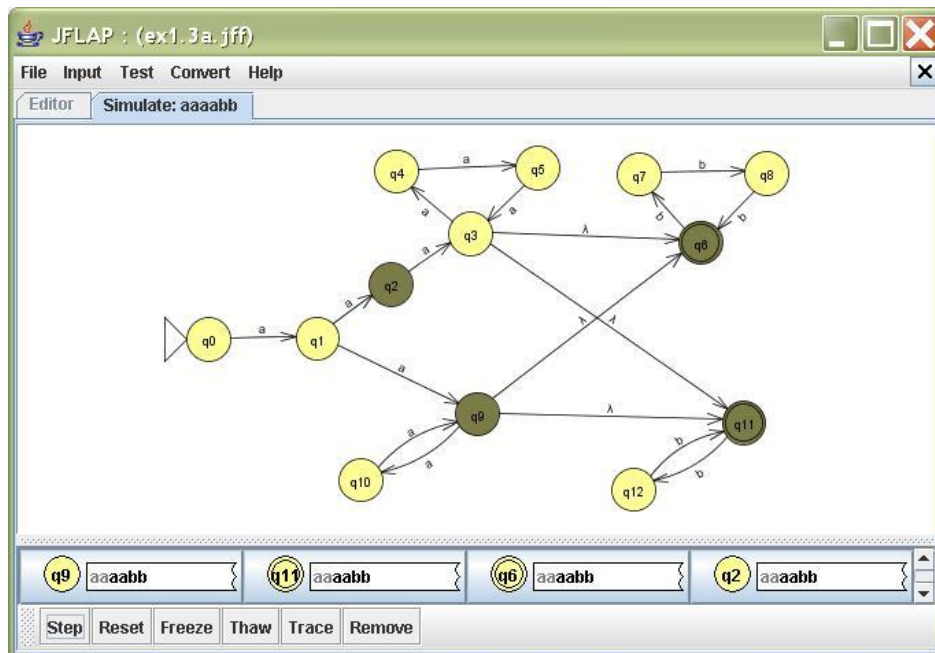


Figura 26: Simulación de *aaaabb* después de dos pasos en el AFND.

De estos caminos de configuraciones, el de  $q_9$  es el que puedes ver que nos llevará a la aceptación de la palabra. Selecciona esa configuración, y pincha en Freeze. El recuadro se coloreará de un azul más oscuro. Ahora pulsa en Step de nuevo: mientras que las demás configuraciones se mueven (y son rechazadas), esta configuración no progresa porque la hemos “congelado”. Ahora pulsa Thaw, con esa configuración seleccionada. Esto la “descongelará”, y al pulsar en Step, podremos continuar con ese camino hasta ver como es aceptada.

Pincha en la configuración aceptada y pulsa en Trace. Observa como hay un camino directo de  $q_{10}$  a  $q_{11}$ , aunque no hay transición de  $q_{10}$  a  $q_{11}$ . No se ha generado ninguna transición para  $q_9$  por la transición  $\lambda$  que hay.

- Simulación por pasos (Step by State): Selecciona en el menú **Input > Step by State**, e introduce la palabra “*aaaabb*”. En esta simulación la herramienta pasa explícitamente por las transiciones  $\lambda$ . Si pulsas en **Step** dos veces, tendrás las configuraciones en  $q_2$  y  $q_9$ , pero no las configuraciones en  $q_6$  y  $q_{11}$ , que vimos antes. Si pulsas de nuevo, la configuración en  $q_9$  se dividirá en tres más, dos de las cuales son  $q_6$  y  $q_{11}$ . Si continuas avanzando en la simulación hasta la aceptación de la palabra, y muestras la traza, la configuración después de  $q_{10}$  es  $q_9$ , lo cual deriva en  $q_{11}$  con la transición  $\lambda$ . Aunque esta simulación es menos confusa que la anterior, suele usarse menos esta, porque no garantiza que en cada paso se lea un símbolo de entrada.

- Simulación rápida: El simulador rápido tiene algunas características adicionales específicas para el no determinismo. Selecciona **Input > Fast Run**, e introduce la palabra “*aaaaaabb*”. Una vez hecho esto JFLAP mostrará una de las trazas de aceptación de la palabra. El botón **Keep Looking** es útil para máquinas no deterministas, ya que sirve para visualizar la traza del resto de caminos distintos que hay para llegar a aceptar la palabra. Si pulsas ese botón una vez, comprobarás como JFLAP muestra otro caso distinto. Y si pulsas de nuevo, aparecerá un mensaje que te dirá que ha habido dos configuraciones (dos trazas) aceptadas, y que no hay más configuraciones posibles.

- Simulación múltiple: La simulación multiple solo presenta una traza por ejecución. Específicamente, cuando se acepta una palabra, JFLAP mostrará la traza de la primera configuración aceptada, y cuando se rechaza, se mostrará la traza de la última configuración rechazada. Por lo que si quieres depurar un AFND es mejor utilizar alguna de las otras simulaciones.

## 7.4 Operadores simples del análisis

Además de las simulaciones, JFLAP ofrece algunas herramientas para determinar algunas propiedades del autómata.

### 7.4.1 Comparar equivalencias

Este operador compara si dos autómatas aceptan el mismo lenguaje. Para ilustrar el funcionamiento, de esta herramienta, compararemos el autómata de la Figura 20, y el de la 27.

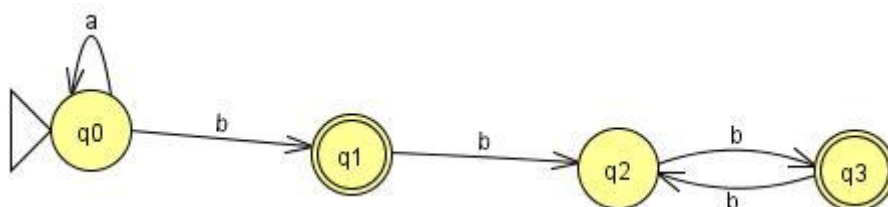


Figura 27: ejemplo de un autómata finito que reconoce el mismo lenguaje que el autómata de la figura 20.

Para comparar dos autómatas, debes tener una ventana de editor abierta para cada uno de los autómatas que vayas a comparar. Después, en una de esas ventanas, en la barra de herramientas ve a **Test > Compare Equivalente**. Aparecerá una ventana emergente con una lista donde podrás elegir el nombre del autómata con el que quieres comparar el autómata del editor actual. Una vez seleccionado, pincha en OK, y aparecerá una ventana informando de que son equivalentes. En el caso de que los autómatas no lo sean (prueba a cambiar algo en uno de los dos autómatas y a compararlos de nuevo), recibiremos el aviso de que no aceptan el mismo lenguaje.

### 7.4.2 Marcar no determinismo

Este operador mostrará al usuario que estados en el autómata son no deterministas. En el ejemplo de la Figura 25, el estado q1 es obviamente no determinista y además de ese tipo de estados con dos salidas diferentes para una misma entrada, JFLAP considera a todos los estados con transiciones  $\lambda$  salientes como no deterministas, por lo que q3 y q9 también lo son. Selecciona **Test > Highlight Nondeterminism**, para marcar esos estados.

### 7.4.3 Marcar transiciones $\lambda$

Este operador mostrará de rojo todas las transiciones  $\lambda$ . Selecciona **Test > Highlight  $\lambda$ .Transitions** para marcarlas.



## 7.5 Transiciones alternativas de múltiples caracteres.

JFLAP proporciona una definición más general de un AF, permitiendo múltiples caracteres en una transición. Esto puede resultar en AF's simples. En la Figura 28 hay un AFND de 5 estados que acepta el mismo lenguaje que el AFND de la Figura 25. Observa que en la Figura 28 las transiciones que no tienen múltiples símbolos son transiciones  $\lambda$ . Una configuración puede ir a una transición de  $n$  caracteres  $s_1s_2\dots s_n$ , si los próximos símbolos de entrada son  $s_1, s_2$ , hasta  $s_n$ .

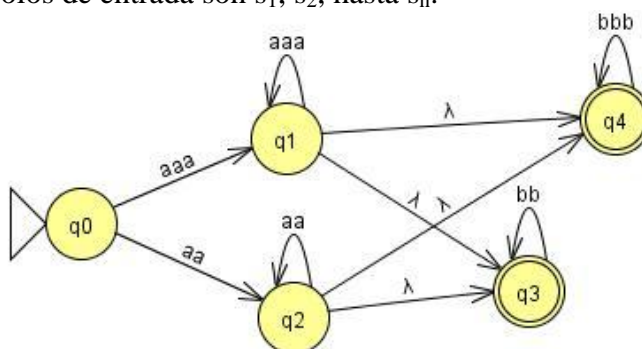


Figura 28: Ejemplo de AFND con múltiples caracteres en las transiciones

Ahora ejecutaremos una simulación de este AFND. Selecciona **Step With Closure**, e introduce “aaaabb”. Aparecerá la ventana del simulador con la configuración iniciada en q0. Pincha una vez en **Step**, y verás seis configuraciones. Hay dos por q3 y q4, una por q1 y otra por q2. Observa que esas dos configuraciones tienen una cantidad diferente de elementos que aún no se han leído. Da dos pasos más y aparecerá un estado de aceptación en q3 (figura 29)

Al permitir múltiples caracteres en las transiciones, la primera condición para un AFND de la sección 7.3 cambia. La primera condición esa ahora la que sigue: si un AF tiene dos transiciones desde el mismo estado que leen la palabra A y B, donde A es un prefijo de B, el AF es considerado AFND. Por ejemplo, en la Figura 28, q0 es un estado no determinista: tiene dos transiciones, una con *aaa* y otra con *aa*, por lo que es un AFND.

## 7.6 Definición de AF en JFLAP

JFLAP define un AF  $M$  como la quintupla  $M = (Q, \Sigma, \delta, q_s, F)$  donde:

- $Q$  es un conjunto finito de estados  $\{q_i \mid i \in \mathbb{Z}\}$
- $\Sigma$  es el alfabeto finito de entrada.
- $\delta$  es la función de transición,  $\delta: D \rightarrow 2^Q$  donde  $D$  es un subconjunto finito de  $Q \times \Sigma^*$ .
- $q_s \in Q$  es el estado inicial.
- $F \subseteq Q$  es el conjunto de los estados finales.

Hay unas variaciones respecto a la función de transición: si es un AFD  $\delta: Q \times \Sigma \rightarrow Q$ , y si es un AFND  $\delta$  es la función de transición  $\delta: Q \cup \{\lambda\} \rightarrow 2^Q$ .

Para el caso de las transiciones con múltiples caracteres, la definición del dominio de  $\delta$  cambia: el conjunto  $Q \times \Sigma^*$  es de infinita cardinalidad, aunque la función de transición requiere un dominio finito.  $\Sigma^*$  simboliza una palabra de 0 o más símbolos del alfabeto de entrada.

## 8. AFND a AFD y a AFD mínimo

Este capítulo muestra como un AFND puede ser convertido en su equivalente AFD y como cada AFD puede ser reducido a un AFD con el número mínimo de estados. Aunque un AFND puede ser más fácil de construir que un AFD, es menos eficiente en la ejecución.

### 8.1 AFND a AFD

La idea es crear estados en el AFD que representen múltiples estados de un AFND. El estado inicial en el AFD representa el estado inicial en el AFND y cualquier estado accesible desde el con  $\lambda$ . Para cada nuevo estado en el AFD y para cada letra del alfabeto, uno determina todos los estados accesibles desde el correspondiente estado no determinista y los combina en un nuevo estado para el AFD. Este estado en el AFD tendrá una etiqueta que contendrá los números de los estados del AFND que podrán ser accedidos tomando el mismo camino.

#### 8.1.1 Idea para la conversión

Primero vamos a examinar las opciones que hay cuando se procesa una palabra en un AFND, con el ejemplo que se muestra a continuación:

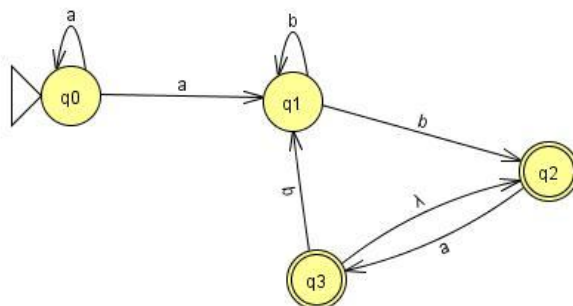



Figura 29: Ejemplo de AFND

Selecciona **Input > Step with Closure** e introduce “aabbbaa”. Si vas pinchando en **Step**, verás como de inicio podemos ir a q0 y q1, y que hay siempre 3 configuraciones (una de ellas, de rechazo).

Los estados en el AFD construido representarán la combinación de estados del AFND. Por ejemplo, procesando una *a* resulta el estado q0 o el q1. El AFD podrá tener un estado que represente a ambos. Procesando *aabbbaa* llegamos al estado q2 y q3. El AFD podrá tener un estado que represente a esos dos.

#### 8.1.2 Ejemplo de conversión

Ahora vamos a convertir el AFND de la figura 29 a un AFD (pincha en **Convert > Convert to DFA**). El estado inicial en el AFD es q0 y tiene la etiqueta 0, lo que representa el estado q0 del AFND.

Después de esto podemos añadir el estado que se alcanza desde q0 leyendo *a*. Selecciona el Expand Group, de la herramienta . Pincha y mantiene pulsado el botón del ratón en el estado q0, y arrastra el cursor hacia donde quieras colocar el siguiente estado y suéltalo. Aparecerá una ventana preguntando el terminal con el que vamos a expandir, que será el símbolo *a*. Al introducir esto, JFLAP nos preguntará por el grupo de estados del AFND que son accesibles desde q0 al introducir una *a*, que serán q0 y q1 (estos estados están representados por los números 0 y 1, que es lo que habrá que

introducir en la ventana). Una vez hecho todo esto, un nuevo estado q1 aparecerá, tal y como se muestra en la figura 30.

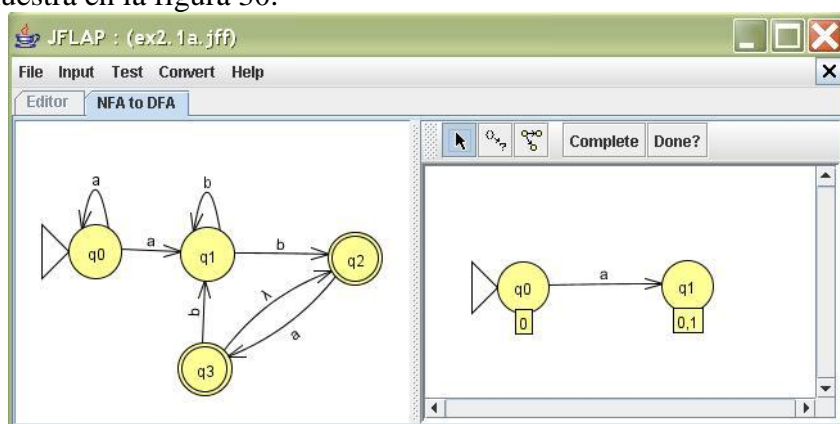


Figura 30: Ejemplo de expansión del estado q0 con la entrada 'a'.

Intenta expandir el estado q0 del AFD, con el terminal  $b$ . Como no hay ningún camino desde el AFND con esas características, un mensaje de aviso aparecerá.

Ahora expande el estado q1 del AFD con el terminal  $a$ . Date cuenta que ese estado representa los estados q0 y q1 del AFND. En el AFND, el estado q0 al leer  $a$ , transita a q0 y a q1, y el estado q1 no transita a ningún otro estado. La unión de esos resultados (0,1) será el q1 del AFD. Para ello, expande el estado q1 añadiendo un loop. Después de eso, expandelo de nuevo en el estado q1 con  $b$ . El resultado de esas expansiones se muestra en la figura 31. El estado q2 del AFD estará marcado como un estado final, porque los estados equivalentes en el AFND son finales también.

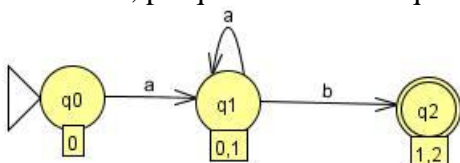


Figura 31: Expansión de  $a$ ,  $b$  desde q1

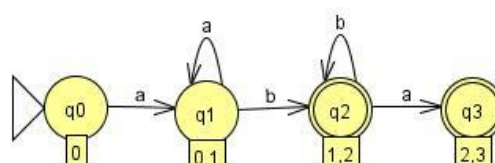



Figura 32: Expansión de  $a$ ,  $b$  desde q2.

Expande ahora el estado q2 del AFD con  $a$ . Este estado está representado por los estados q1 y q2 del AFND, en los que q1 no tiene transiciones  $a$ , y q2 si la tiene.

Expande el estado q2 del AFD con  $b$ , pinchando en el estado q2. El AFD resultante es el de la figura 32.

Hay otra forma de expandir un estado, y es utilizando la herramienta . Cuando se selecciona esta herramienta y se pincha en un estado, y todos los arcos que tuvieran que salir de ese estado aparecen automáticamente. Prueba a realizar esto en q3. ¿Está el autómata completo? Pulsando en el botón **Done?** JFLAP nos avisará sobre los elementos que aun falten, o nos dirá si ya lo hemos completado. En nuestro caso, aun falta una transición, del estado q4 nuevo que ha aparecido. Utiliza la herramienta anterior para completarlo, y el autómata resultante deberá ser como el de la figura 33.

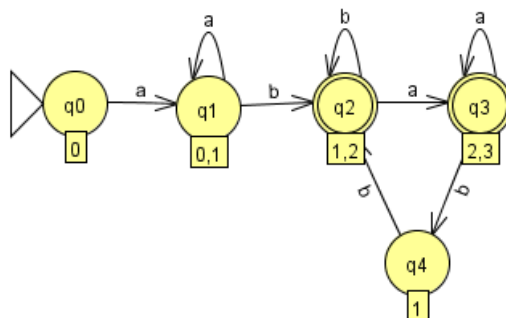


Figura 33: El autómata finito determinista completo.

Ahora vuelve a pulsar el botón Done?. El AFD completo es exportado a una nueva ventana. De forma alternativa, el botón Complete puede ser seleccionado en cualquier momento del proceso de construcción y el AFD se completará automáticamente.

El nuevo AFD debe ser equivalente al AFND. Para probar esto, selecciona en la barra de herramientas **Test > Compare Equivalence**.

## 8.2 Conversión al AFD mínimo

En esta sección mostraremos como convertir un AFD a un autómata equivalente con un número mínimo de estados. Esto viene de considerar dos estados  $p$  y  $q$  de un AFD, que procesan un símbolo desde su estado. Si hay al menos una palabra  $w$  para la cual los estados  $p$  y  $q$  la procesan, y un estado acepta  $w$  y otro rechaza  $w$ , entonces esos estados son distinguibles y no pueden ser combinados. Por otra partes, si los estados  $p$  y  $q$  actúan del mismo modo, significa que son indistinguibles, y pueden ser combinados.

### 8.2.1 La idea de la conversión

Utilizaremos el AFD de la siguiente figura de ejemplo, para explicar el proceso.

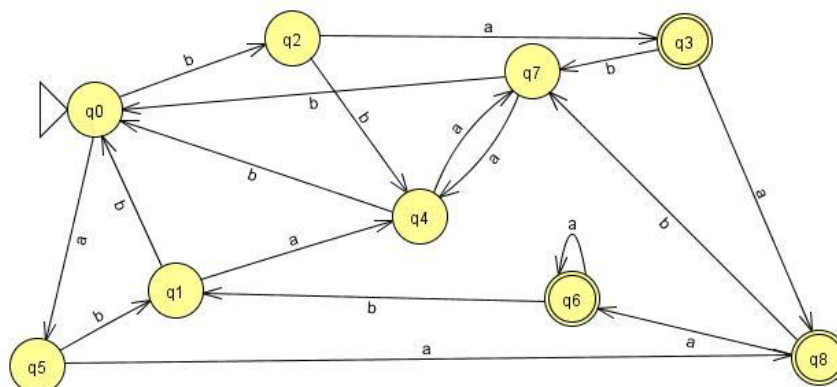


Figura 34: AFD de ejemplo para el punto 8.2.

Una forma de ir viendo los estados distinguibles y los que no, es abriendo dos ventanas distintas de JFLAP, con el mismo ejemplo. Para ello, créalo en el editor, guárdalo dos veces con distintos nombres, y podrás verlos en dos ventanas.

Ahora pasaremos a examinar los estados  $q0$  y  $q1$ , para ver si son distinguibles. En una de las dos ventanas, cambia el estado inicial por  $q1$ . Examina los dos AFD. ¿Hay alguna palabra que un AFD acepte y el otro rechace?

Vamos a explorar varias entradas para ver si hay alguna diferencia. En las dos ventanas, selecciona **Input > Multiple Run**, e introduce estas entradas y otras que se te ocurran: “a”, “aab”, “aaaab”, “baa”, “baaa” y “bba”. Pulsa **Run Inputs** y examina los resultados. ¿Son los mismos resultados? Hay por lo menos una palabra en la que el resultado es **Accept** para un AFD y **Reject** en el otro. Por tanto, esos dos estados son distinguibles y no se pueden combinar.

Ahora vamos a examinar los estados  $q2$  y  $q5$ . En una de las ventanas del editor de los dos AFD cambia el estado inicial por  $q2$ , y en la otra por  $q5$ . Selecciona **Input > Multiple Run** de nuevo. Las entradas de la última ejecución aún aparecen en la ventana, así que pulsa en **Run Inputs** para probarlas. Añade otras palabras y pruébalas también. ¿Son distinguibles? Para determinar que dos estados son indistinguibles, se deben probar todas las posibles entradas, pero como esto no es posible, un conjunto razonable de test que nos creemos servirá.



### 8.2.2 Ejemplo de conversión.

Abre el archivo donde tuvieras guardado el autómata de la Figura 34 y selecciona **Convert > Minimize DFA**. La ventana se separará en dos mostrando el AFD a la izquierda y un árbol de estados en la derecha.

Se asume al principio que todos los estados son indistinguibles. La raíz del árbol contiene todos los estados. Cada vez que determinemos una distinción entre estados, añadiremos un nodo en el árbol para mostrarla. Se continuará dividiendo nodos hasta que no haya más divisiones posibles. Cada hoja del árbol final representará un grupo de estados indistinguibles.

El primer paso que JFLAP mostrará al abrirse la ventana, será para distinguir los estados finales y los no finales. Por ello, el árbol se divide en el conjunto de estados finales y en los no finales, tal y como muestra la Figura 35.

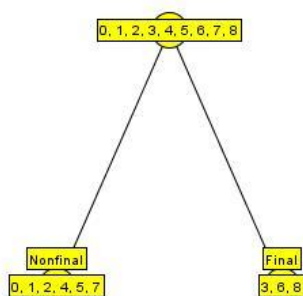


Figura 35: División inicial de estados del conversor.

Para futuras divisiones, se seleccionará un terminal que distinga los estados en el nodo. Si alguno de los estados en un nodo hoja, para ese terminal, va a algún estado de otro nodo hoja, y otros estados con el mismo terminal van a estados que están en otro nodo hoja, entonces el nodo deberá ser dividido en dos grupos de estados.

Vamos a explorar primero el nodo hoja con los estados no finales. ¿Qué ocurre para cada uno de esos estados si procesamos una *b*? El estado *q0* transita al *q2*, el *q1* va al *q0*, y así sucesivamente. Cada uno de esos estados van a un mismo estado de ese nodo. Por lo tanto, la entrada *b* no los distingue. Si tratas de introducir *b*, en **Set Terminal**, JFLAP mostrará un mensaje informando sobre lo que ya habíamos averiguado (para poder seleccionar esa herramienta, debes pulsar primero en el nodo del árbol correspondiente).

Selecciona de nuevo **Set Terminal** e introduce el terminal *a*. En este caso, si que distingue los estados, por lo que el nodo se divide. El conjunto de estados que van en cada división debe ser introducido, en grupos que son indistinguibles. Un número de estado puede ser introducido seleccionando primero el nodo hoja donde estaba asignado, y entonces pinchando en el correspondiente estado en el AFD. Pincha en la hoja izquierda del nodo y entonces pincha en el estado *q0* del AFD. El estado número 0 deberá aparecer en el nodo hoja, tal y como se muestra en la Figura 36.

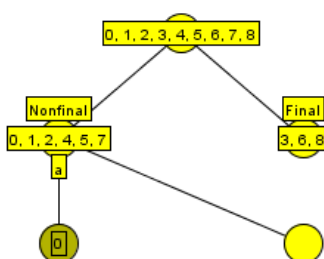


Figura 36: División del nodo con *a*.

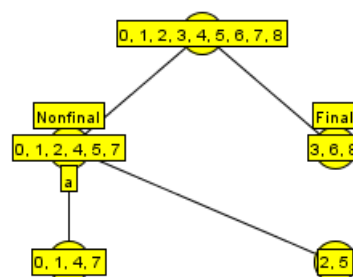


Figura 37: División completa del nodo(0,1,2,4,5,7).

Añade el resto de estados, a los nodos hojas, igual que has hecho con el estado q0, hasta llegar a la imagen mostrada en la Figura 37. Para ver si esta bien hecho, pincha en **Check Node**.

Ahora debemos continuar con el resto. Probaremos con el nodo hoja con estados 0, 1, 4 y 7, con *a*. Selecciona **Set Terminal** e introduce *a*. El estado q0 transita a q5 con *a*, el cual está en otro nodo hoja, y los estados q1, q4 y q7 realizan transiciones entre ellos, por lo que se mantienen en el mismo nodo. Así que ya tenemos los grupos a dividir. Para ello, selecciona **Auto partition** y los estados se insertarán de forma automática como aparece en la Figura 38.

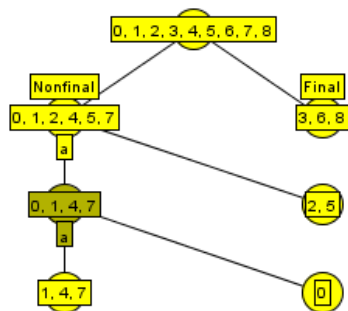


Figura 38: Árbol completo de estados distinguibles.

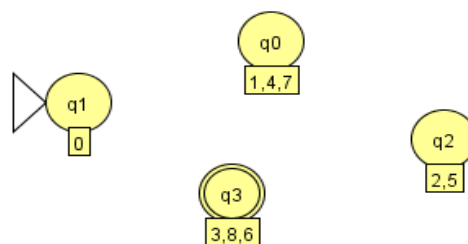


Figura 39: Estados para el AFD mínimo.

Cuando el árbol se haya completado, la única opción visible será Finish. Selecciona ese botón, y la parte derecha de la ventana será reemplazada por los nuevos estados del AFD mínimo. Hay un estado para cada nodo hoja del árbol (observa como las etiquetas de esos estados son las mismas que las del árbol son las mismas etiquetas).

Ahora añade los arcos que faltan en el nuevo AFD utilizando la herramienta **Transition Creator**. En el AFD original hay una *a* desde el estado q0 al q5, por lo que en el AFD nuevo, deberá existir una transición que corresponda a esa, desde el estado q1 (que representa al antiguo estado q0), al q2 (representando al antiguo q5). Seleccionando **Hint**, JFLAP añadirá una transición por ti, y seleccionando **Complete**, se completará automáticamente el autómata. Selecciona **Done?** para exportar el nuevo AFD a una ventana del editor.

El AFD de estados mínimos debe de ser equivalente al AFD original. Pruébalo utilizando **Test > Compare Equivalence**.

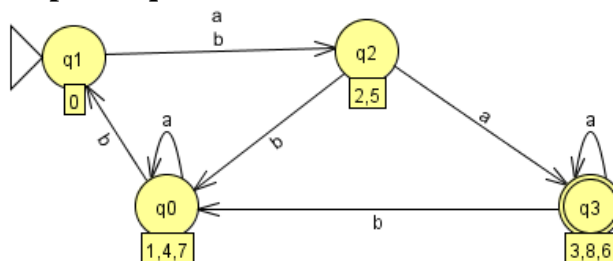


Figura 40: El AFD mínimo.

*Nota:* cuando vas a dividir un nodo con la herramienta **Set Terminal**, aparecen dos hijos, pero es posible que el nodo a dividir pueda necesitar más hijos. Si es así, es que puede haber 3 o más grupos distinguibles a dividir con un terminal. En ese caso, tu puedes añadir los nodos hojas adicionales seleccionando el botón **Add Child** para cada hijo adicional deseado.