Teoría de Autómatas y Lenguajes Formales

Prácticas
Introducción a JFLAP

Autores:

Araceli Sanchis de Miguel
Agapito Ledezma Espino
Jose A. Iglesias Martínez
Beatriz García Jiménez
Juan Manuel Alonso Weber

* Algunos ejercicios están basados en enunciados del siguiente libro:

- Susan H. Rodger and Thomas W. Finley. *JFLAP: An Interactive Formal Languages and Automata Package*. Jones & Bartlett Publishers, Sudbury, MA (2006).

JFLAP (del inglés, *Java Formal Language and Automata Package*) es un software que permite experimentar de forma gráfica con los conceptos relativos a la teoría de autómatas y lenguajes formales. Permite diseñar, evaluar y realizar distintas transformaciones y comprobaciones sobre autómatas finitos, gramáticas, autómatas a pila, máquinas de Turing, y otros elementos adicionales que no forman parte del contenido de este curso.

Para la realización de estas prácticas se requiere la descargar de la herramienta libre JFLAP de su sitio web (http://www.jflap.org/). Además es necesario consultar el tutorial *on-line* de la herramienta (http://www.jflap.org/tutorial/), o el siguiente libro guía:

> Susan H. Rodger and Thomas W. Finley. *JFLAP: An Interactive Formal Languages and Automata Package*. Jones & Bartlett Publishers, Sudbury, MA (2006).

NOTA IMPORTANTE: Para los ejercicios de limpieza y bien formación de gramáticas, se debe tener en cuenta la siguiente correspondencia entre la nomenclatura y orden de pasos de la explicación teórica y ejercicios, y el proceso utilizado en la herramienta JFLAP, acorde a la siguiente tabla.

Algoritmo	JFLAP	Explicación teórica	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	*λ–production removal*	Eliminación de reglas no generativas	No hay	-----
Reglas de redenominación	*Unit production removal*	Eliminación de Reglas de redenominación	No hay	-----

1. Dada una gramática para generar números naturales:
   G = ({0,1,2,3,4,5,6,7,8,9,0}, {N, C}, N, P)
   P = { N ::= CN | C,
   C ::= 0|1|2|3|4|5|6|7|8|9 }

Comprobad con el derivador múltiple por fuerza bruta (opción *Input → Multiple Brute Force Parse*) qué secuencias son palabras del lenguaje L(G) y determinar el motivo. Anotad las observaciones que realicéis en cada caso, con el derivador por fuerza bruta simple (opción *Input → Brute Force Parse*).

Palabras de prueba	0	00	001	123	1234	12345	9212	43.34

2. Crear una gramática G que genere números reales, con "." para separar la parte entera y la fraccionaria. Comprobad qué secuencias de las siguientes son palabras del lenguaje L(G) y determinar el motivo. Anotad las observaciones que realicéis en cada caso.

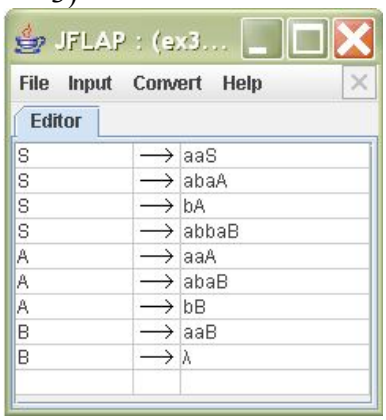
Palabras de prueba	0	00	00.1	3445	1.23	9.21.2	9..1	9,1

3. Obtención de Lenguajes. Escribe las siguientes gramáticas en el editor de JFLAP, y para cada una de ellas haz:
   a) una lista de 5 palabras que son aceptadas,
   b) una lista de 5 palabras que son rechazadas,
   c) da una descripción del lenguaje que representan,
   d) indica el tipo de gramática en la jerarquía de Chomsky.

1)

2)

3)



4. **Gramáticas ambiguas.** Considerar la siguiente gramática ambigua y la palabra *ababab*.

```
S -> abS
S -> Sab
S -> aSb
S -> SS
S -> λ
```

Ejecuta el derivador de fuerza bruta de JFLAP para esta palabra. Proporciona dos árboles de derivación diferentes para esta palabra y anota el orden de producciones empleado.
El algoritmo de JFLAP ejecuta siempre las producciones en el orden en el que están escritas en el editor, por lo que para que el árbol cambie, hay que cambiar el orden de las producciones en el editor.

5. **Eficiencia del Derivador por Fuerza Bruta.** El algoritmo que emplea el Derivador por Fuerza Bruta consiste en generar un árbol de búsqueda en el que cada nodo es una Forma Sentencial (la inicial es el axioma) y cada rama es la aplicación de alguna de las producciones. El método busca generar una sucesión de Formas Sentenciales que conduzcan a la palabra buscada. La búsqueda suele ser exhaustiva y para problemas de tamaño relativamente pequeño generan árboles de tamaño considerable, con el consiguiente gasto en tiempo de procesamiento.

   Para la siguiente gramática y las cadenas de la forma $a^n bb$:

```
G = ({a,b},{S,A,B,C},S,P), donde
     P={  S -> AC
          C -> AB
          B -> CC
          A -> a
          C -> b
          S -> SS
          A -> AA }
```

(a) Ejecuta el brute force parser para n= 2,...,8 y apunta el número de nodos generados. ¿Qué observación puedes hacer acerca del número de nodos generados? ¿Y del tiempo que ha tardado? ¿Sabes el motivo?
(b) Elimina la producción S →SS y ejecuta de nuevo el brute force parser para n=2,...,8 y apunta el número de nodos generados. Compara los resultados con los del apartado (a). ¿Sabes cuál es el motivo de la diferencia?
(c) Añadiéndolo al cambio anterior, cambia la producción A→AA a A→aA. Ejecuta el brute force parser para n=2,...,8 y apunta el número de nodos. Compara los resultados con los de los apartados (a) y (b). ¿Sabes cuál es el motivo de las diferencias?

6. El estudiante Carlos quiere que sus padres le compren un coche. Su padre genera una secuencia en la que transcribe todas las notas que Carlos ha recibido en su vida, una larga cadena hecha con los símbolos de ∑ = {a,b,c,d,f} sin un orden particular. Una *a* es un sobresaliente, y una *f* es la nota más baja. Su padre plantea un criterio simple: si Carlos recibe menos que 4 *d's*, tendrá el coche, y de otra forma, no lo tendrá. Una *f* cuenta igual que una *d.*

   Su padre te pide que diseñes una gramática lineal derecha donde la gramática genere la secuencia de trascripción sí y sólo si Carlos consigue el coche. Nota: Puedes probar si una gramática lineal derecha genera una palabra con el derivador de fuerza bruta, o bien convirtiendo la gramática a un autómata finito y comprobando si el AF acepta la palabra.

   (a) Diseña una gramática lineal derecha con las especificaciones anteriores.

   (b) La reacción inicial de Carlos fue de realizar una súplica, para que su padre adoptase métodos menos draconianos, abogando indirectamente a la noción que la diferencia generacional entre el padre y el hijo pudo haber conducido al padre a ser injustamente duro. El padre decide hacer que una nota *f*, cuente como dos *d's*. Modifica la gramática para que tenga en cuenta este cambio.

   (c) Con un poco más de tacto, Carlos fue capaz de discutir satisfactoriamente con su padre, llegando al acuerdo de que una nota de sobresaliente *a*, debería mitigar un suspenso d. ¿Por qué este lenguaje no puede ser expresado con una gramática lineal derecha?

7. Transformar las siguientes gramáticas regulares lineales derechas a su correspondiente Autómata Finito, utilizando JFLAP.

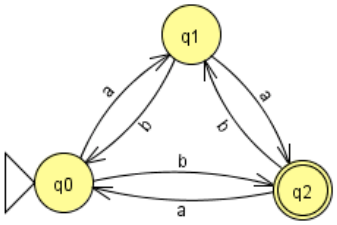
1)
S ::= aA
S ::= bB
B ::= bS
B ::= λ
A ::= aS
A ::= λ

2)
S ::= aS
S ::= aA
A ::= bB
A ::= cC
B ::= bS
B ::= b
C ::= cS
C ::= λ

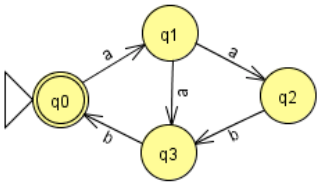
3)
S ::= bS
S ::= aA
S ::= bB
S ::= bC
A ::= aS
B ::= bC
B ::= b
C ::= cS
C ::= c

8. Edita los siguientes autómatas finitos en JFLAP y transfórmalos a sus correspondientes gramáticas, utilizando JFLAP.

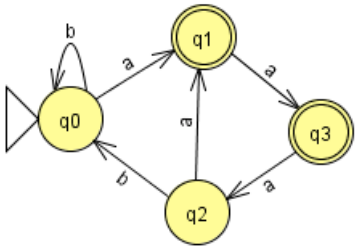
1)



2)



3)



9. Eliminar las reglas no generativas de la gramática G con JFLAP. Anota los pasos de transformación y la gramática obtenida (para cada paso, anota la producción no generativa que vas a eliminar y las producciones nuevas que vas añadiendo).

G=({a,b,c},{S,A,B},S,P)
P={ S ::= Ac | BaA
A ::= B | a
B ::= bB | λ }

10. Eliminar las reglas superfluas y símbolos inaccesibles usando JFLAP en la gramática G. Anota los pasos de transformación y la gramática obtenida.

G=({a,b,c},{S,A,B},S,P)
P={ S ::= aSB | b
A ::= a | aA
B ::= AaC | bA
C ::= cCc}

SOLUCIONES

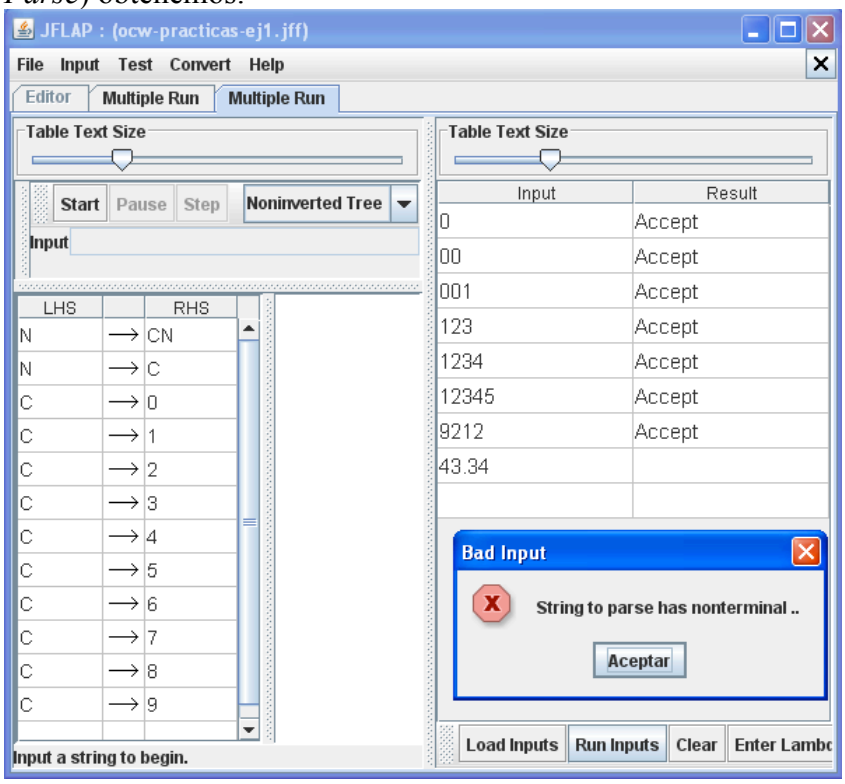
1. **Dada una gramática para generar números naturales:**
   G = ({0,1,2,3,4,5,6,7,8,9,0}, {N, C}, N, P)
   P = { N ::= CN | C,
   C ::= 0|1|2|3|4|5|6|7|8|9 }

Comprobad con el derivador múltiple por fuerza bruta (opción *Input → Multiple Brute Force Parse*) qué secuencias son palabras del lenguaje L(G) y determinar el motivo. Anotad las observaciones que realicéis en cada caso, con el derivador por fuerza bruta simple (opción *Input → Brute Force Parse*).

Palabras de prueba	**0**	**00**	**001**	**123**	**1234**	**12345**	**9212**	**43.34**

Solución:

Aplicando el *derivador múltiple por fuerza bruta* (opción *Input → Multiple Brute Force Parse*) obtenemos:



Las anotaciones de las observaciones para cada palabra, aplicando el *derivador por fuerza bruta simple* (opción *Input → Brute Force Parse*), junto con el resultado de aceptación o no con JFLAP son las siguientes:

Palabra de prueba	0	00	001	123	1234	12345	9212	43.34
Aceptada	Si	Si	Si	Si	Si	Si	Si	*
Nodos generados	3	9	23	25	74	229	65	*

* No se deriva, porque contiene un símbolo que no pertenece al alfabeto (“.”).

2. Crear una gramática G que genere números reales, con "." para separar la parte entera y la fraccionaria. Comprobad qué secuencias de las siguientes son palabras del lenguaje L(G) y determinar el motivo. Anotad las observaciones que realicéis en cada caso.

Palabras de prueba	**0**	**00**	**00.1**	**3445**	**1.23**	**9.21.2**	**9..1**	**9,1**

Solución:

Una posible gramática sencilla que genera números reales sería la siguiente:

$G = (\{0,1,2,3,4,5,6,7,8,9,.\}, \{N, C, D, X\}, N, P)$

$P = \{ N ::= XCX \mid X,$

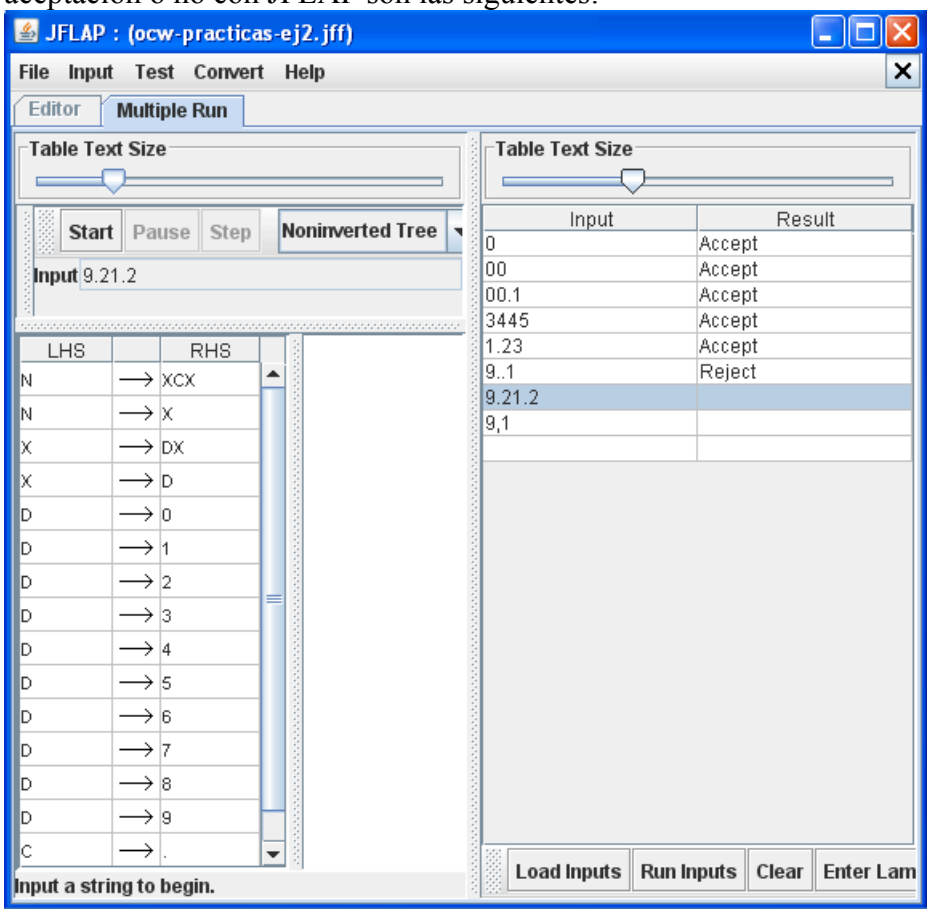
$X ::= DX \mid D$

$D ::= 0|1|2|3|4|5|6|7|8|9$

$C ::= .$

$\}$

Las anotaciones de las observaciones para cada palabra, junto con el resultado de aceptación o no con JFLAP son las siguientes:



Palabras de prueba	0	00	00.1	3445	1.23	9.21.2	9..1	9,1
Aceptada	Si	Si	Si	Si	Si	*	NO	**
Nodos generados	4	10	95	127	92	*	118	**

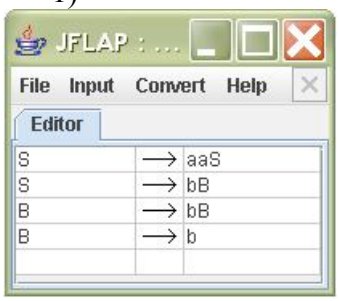
* "Out of Memory". Si se cambia esta palabra por otra más reducida, con sólo un punto decimal, 9.21, se acepta en 92 nodos. Por lo que la existencia de dos puntos decimales influye bastante en la cantidad de memoria necesaria, por el incremento de posibilidades a evaluar.

** Error, porque hay un símbolo que no pertenece al alfabeto (coma en vez de punto).

Cabe destacar que también se pueden diseñar gramáticas más detalladas, que incluyan comprobaciones sobre la existencia de dígitos no significativos a izquierda y derecha del punto decimal. En esos casos, las observaciones sobre el comportamiento de JFLAP ante la lista de palabras del enunciado podría variar. Por ejemplo, para la palabra "00.1", podría no reconocerla en lugar de aceptarla, como hace la gramática sencilla propuesta.

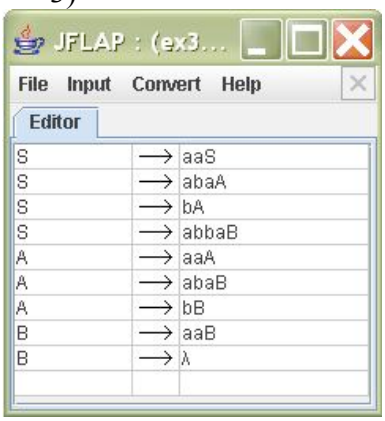
3. Obtención de Lenguajes. Escribe las siguientes gramáticas en el editor de JFLAP, y para cada una de ellas haz:
a) una lista de 5 palabras que son aceptadas,
b) una lista de 5 palabras que son rechazadas,
c) da una descripción del lenguaje que representan,
d) indica el tipo de gramática en la jerarquía de Chomsky.

1)



2)

3)



Solución:

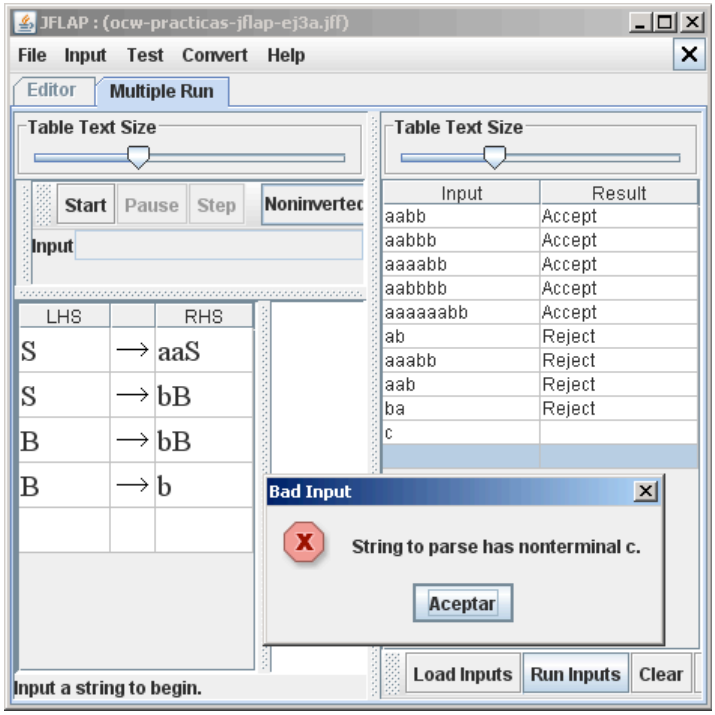
3.1:

a) Palabras aceptadas:
aabb, aabbb, aaaabb, aabbbb, aaaaaabb

b) Palabras rechazadas:
ab, aaabb, aab, ba, c
("c" no es un terminal)

c) L = $\{(aa)^n b^m \mid n>0, m>1\}$

d) Tipo gramática: 2



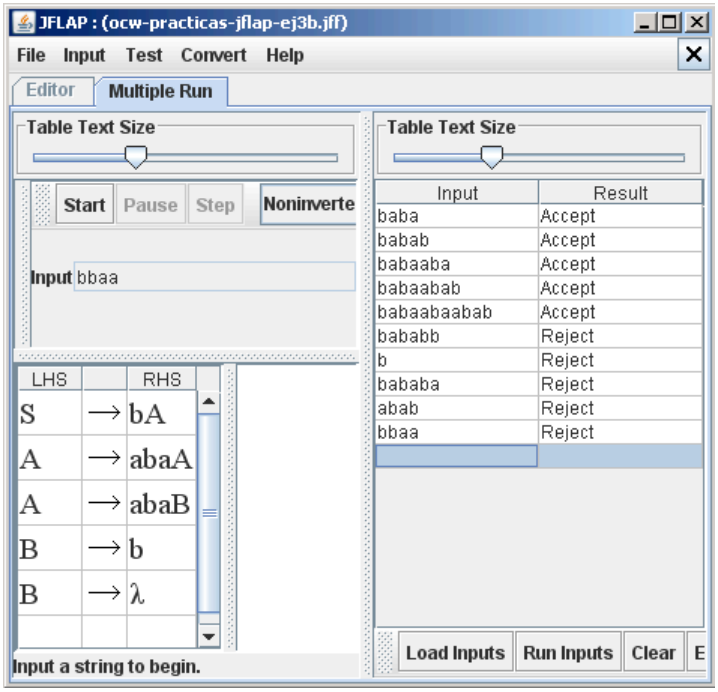
3.2:

a) Palabras aceptadas:
baba, babab, babaaba, babaabab, babaabaabab

b) Palabras rechazadas:
bababb, b, bababa, abab, bbaa

c) L =
$\{b(aba)^n b^m \mid n>0,\ m \in [0,1]\}$

d) Tipo gramática: 0
(regla compresora $B \rightarrow \lambda$)



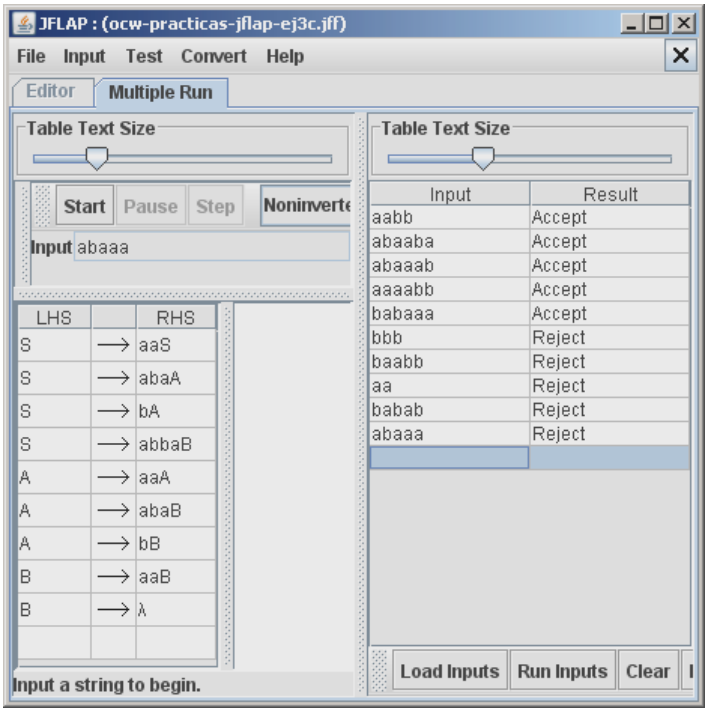
3.3:

a) Palabras aceptadas:
aabb, abaaba, abaaab, aaaabb, babaaa

b) Palabras rechazadas:
bbb, baabb, aa, babab, abaaa

c) $L = \{(aa)^n \ [a^m b a^m \ (aa)^p \ a^q b a^q \ | \ abba \ ] \ (aa)^r \ | \ | \ n,p,r \geq 0; \ m,q \in [0,1] \ \}$
O
$L = \{(aa)^n \ (a^m b a^m \ (aa)^p \ a^q b a^q \ )^s \ (abba)^t \ (aa)^r \ | \ | \ n,p,r \geq 0; \ m,q,s,t \in [0,1]; \ s+t = 1\}$
(ver desarrollo a continuación)

d) Tipo gramática: 0
(regla compresora $B \rightarrow \lambda$)



c) Desarrollo:

$S \rightarrow (aa)^* S \ | \rightarrow (aa)^* aba A \ | \rightarrow (aa)^* b A \ | \rightarrow (aa)^* abba B$

$A \rightarrow (aa)^* A \ | \rightarrow aba B \ | \rightarrow b B$

$B \rightarrow (aa)^* B \ | \rightarrow \lambda$

Sustituyendo A en S:

$S \rightarrow (aa)^* aba A \rightarrow (aa)^* aba \mathbf{(aa)^* aba B}$
$\rightarrow (aa)^* aba \mathbf{(aa)^* b B}$
$\rightarrow (aa)^* b A \rightarrow (aa)^* b \mathbf{(aa)^* aba B}$
$\rightarrow (aa)^* b \mathbf{(aa)^* b B}$
$\rightarrow (aa)^* abba B$

Sustituyendo B, en S y en A:

$S \rightarrow (aa)^* aba A \rightarrow (aa)^* aba (aa)^* aba B \rightarrow (aa)^* aba (aa)^* aba \mathbf{(aa)^*}$
$\rightarrow (aa)^* aba (aa)^* b B \rightarrow (aa)^* aba (aa)^* b \mathbf{(aa)^*}$
$\rightarrow (aa)^* b A \rightarrow (aa)^* b (aa)^* aba B \rightarrow (aa)^* b (aa)^* aba \mathbf{(aa)^*}$
$\rightarrow (aa)^* b (aa)^* b B \rightarrow (aa)^* b (aa)^* b \mathbf{(aa)^*}$
$\rightarrow (aa)^* abba B \rightarrow (aa)^* abba \mathbf{(aa)^*}$

Se puede observar que las 4 primeras posibles derivaciones siguen la estructura (aa)* x (aa)* y (aa)*, pudiendo tomar x e y los valores "b" o "aba", lo que resulta en las 4 combinaciones dadas: aba-aba, aba-b, b-aba y b-b.

De la última posible cadena de derivaciones se obtiene una estructura diferente (abba, sin (aa)* intermedio).

4. Gramáticas ambiguas. Considerar la siguiente gramática ambigua y la palabra *ababab*.

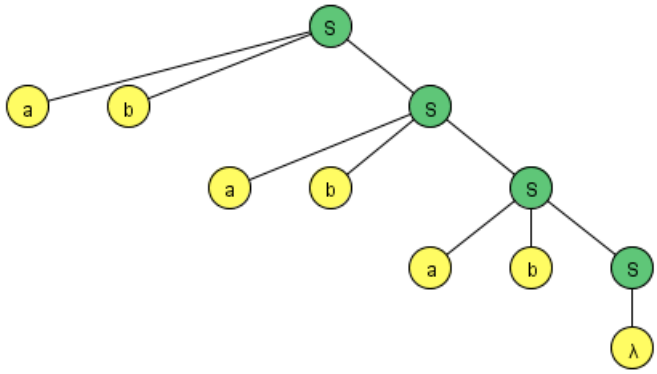
```
S -> abS
S -> Sab
S -> aSb
S -> SS
S -> λ
```

Ejecuta el derivador de fuerza bruta de JFLAP para esta palabra. Proporciona dos árboles de derivación diferentes para esta palabra y anota el orden de producciones empleado.
El algoritmo de JFLAP ejecuta siempre las producciones en el orden en el que están escritas en el editor, por lo que para que el árbol cambie, hay que cambiar el orden de las producciones en el editor.

Solución:

a) Con el orden de producciones indicado el árbol de derivaciones que se obtiene es:

S	→	ab S
S	→	Sa b
S	→	aS b
S	→	SS
S	→	λ

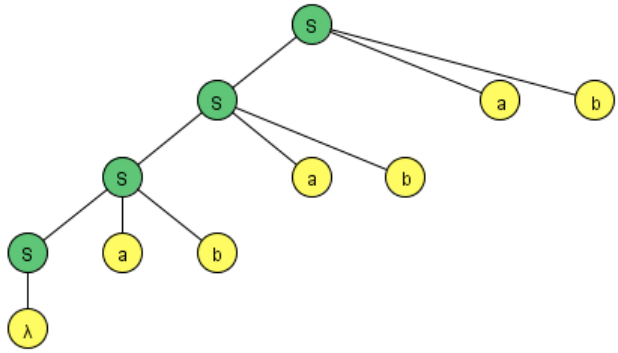


Production	Derivation
	S
S→abS	abS
S→abS	ababS
S→abS	abababS
S→λ	ababab

Orden de producciones: 1ª, 1ª, 1ª, 5ª

b) Cambiando el orden de las dos primeras producciones, el árbol de derivación obtenido es simétrico al anterior. Con dos árboles de derivación diferentes para una misma palabra es suficiente para demostrar la ambigüedad de la gramática. En ambos casos se emplean 657 nodos.

S	→	Sab
S	→	abS
S	→	aSb
S	→	SS
S	→	λ

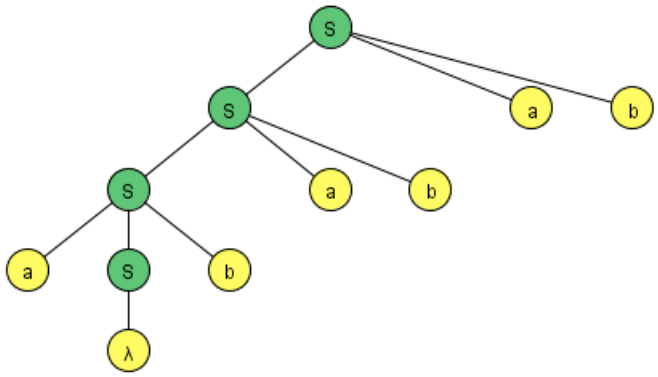


Production	Derivation
	S
S→Sab	Sab
S→Sab	Sabab
S→Sab	Sababab
S→λ	ababab

Orden de producciones: 1ª, 1ª, 1ª, 5ª

c) Otro opción sería colocar $S \rightarrow aSb$ como la primera producción, obteniendo un tercer árbol de derivación diferente para la misma palabra. En este caso, se emplean 760 nodos.

S	→	aSb
S	→	Sab
S	→	abS
S	→	SS
S	→	λ

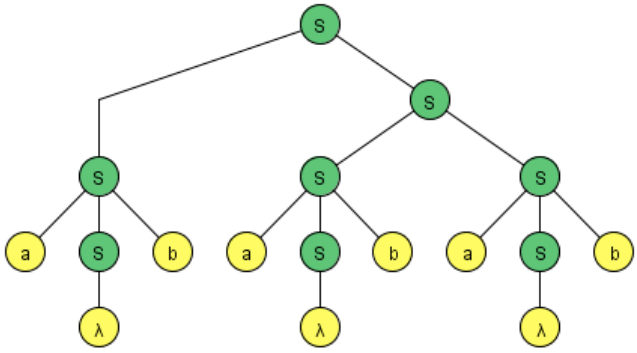


Production	Derivation
	S
S→Sab	Sab
S→Sab	Sabab
S→aSb	aSbabab
S→λ	ababab

Orden de producciones: 2ª, 2ª, 1ª, 5ª

d) Una opción más sería poner $S \rightarrow SS$ como la primera producción, y con las demás en cualquier orden, saldrán distintos árboles para la misma palabra. Por ejemplo, se emplean 4613 nodos con la siguiente gramática, árbol de derivación y orden de producciones:

S	→	SS
S	→	aSb
S	→	abS
S	→	Sab
S	→	λ



Production	Derivation
	S
S→SS	SS
S→SS	SSS
S→aSb	aSbSS
S→aSb	aSbaSbS
S→aSb	aSbaSbaSb
S→λ	abaSbaSb
S→λ	ababaSb
S→λ	abababab

Orden de producciones: 1ª, 1ª, 2ª, 2ª, 2ª, 5ª, 5ª, 5ª

5. Eficiencia del Derivador por Fuerza Bruta. El algoritmo que emplea el Derivador por Fuerza Bruta consiste en generar un árbol de búsqueda en el que cada nodo es una Forma Sentencial (la inicial es el axioma) y cada rama es la aplicación de alguna de las producciones. El método busca generar una sucesión de Formas Sentenciales que conduzcan a la palabra buscada. La búsqueda suele ser exhaustiva y para problemas de tamaño relativamente pequeño generan árboles de tamaño considerable, con el consiguiente gasto en tiempo de procesamiento.

Para la siguiente gramática y las cadenas de la forma $a^n bb$:

```
G = ({a,b},{S,A,B,C},S,P), donde
     P={  S -> AC
          C -> AB
          B -> CC
          A -> a
          C -> b
          S -> SS
          A -> AA }
```

(a) Ejecuta el brute force parser para n= 2,...,8 y apunta el número de nodos generados. ¿Qué observación puedes hacer acerca del número de nodos generados? ¿Y del tiempo que ha tardado? ¿Sabes el motivo?

(b) Elimina la producción S →SS y ejecuta de nuevo el brute force parser para n=2,...,8 y apunta el número de nodos generados. Compara los resultados con los del apartado (a). ¿Sabes cuál es el motivo de la diferencia?

(c) Añadiéndolo al cambio anterior, cambia la producción A→AA a A→aA. Ejecuta el brute force parser para n=2,...,8 y apunta el número de nodos. Compara los resultados con los de los apartados (a) y (b). ¿Sabes cuál es el motivo de las diferencias?

Solución:

Palabras de la forma: $a^n bb$

a)

Tam. Palabra	Num. Nodos	Tiempo (aprox.) CPU antigua	Tiempo (aprox.) CPU más reciente
n=2	75	0 s	<1 s
n=3	182	0 s	<1 s
n=4	382	1 s	<1 s
n=5	782	1.5 s	<1 s
n=6	1328	2.5 s	<1 s
n=7	16855	9 s	6 s
n=8	33019	11 s	8 s
n=9	59733	-	35 s

El tiempo y el número de nodos aumentan según se incrementa el valor de "n". Esto es debido a que jflap tiene que aplicar cada vez más producciones para generar las palabras más largas.
También se puede observar que dependiendo de la potencia de cómputo del ordenador, los tiempos pueden variar, disminuyendo al incrementarse la potencia, pasando incluso de no poder verificar la generación de una palabra a sí hacerlo (como sucede para n=9).

b) Sin S → SS

Tam. Palabra	Num. Nodos	Tiempo (aprox.) CPU antigua	Tiempo (aprox.) CPU más reciente
n=2	43	0 s	0 s
n=3	98	0 s	0 s
n=4	222	0 s	0 s
n=5	556	1 s	<1 s
n=6	1077	1.5 s	<1 s
n=7	3632	3 s	2.5 s
n=8	7282	6 s	4 s
n=9	17017	-	28 s

El tiempo sigue aumentando con la longitud de la palabra, pero ahora es menor que en el caso anterior, en todas las palabras.
La producción eliminada S → SS es una producción que no es necesaria para derivar las palabras del tipo $a^n bb$, se puede prescindir de ella obteniendo dichas palabras. No obstante, al realizar la búsqueda, se intenta aplicar esa producción por lo que el número de nodos de búsqueda generados es mayor que si se prescinde de ella.

c) Sin S → SS y cambiar A→AA por A→aA.

Tam. Palabra	Num. Nodos	Tiempo (aprox.) CPU antigua	Tiempo (aprox.) CPU más reciente
n=2	34	0 s	0 s
n=3	78	0 s	0 s
n=4	136	0 s	0 s
n=5	205	0 s	0 s
n=6	749	1.5 s	<1 s
n=7	1148	2 s	1 s
n=8	8888	22 s	17 s
n=9	15121	-	40 s

La derivación es instantánea en casi todos los casos, menos en el último, donde el tiempo invertido es bastante más elevado, con respecto a los anteriores.
La única forma de convertir una A en un terminal es sustituirla por una a, utilizando las reglas de derivación disponibles. Sustituyendo A→ AA por A → aA (dado que existe la producción A → a) obtenemos una gramática equivalente, pero prescindimos de una recursividad, con lo que los procesos de búsqueda son más rápidos.

En general, para los tres casos, con cada transformación, el proceso de derivación es mucho más rápido.

6. El estudiante Carlos quiere que sus padres le compren un coche. Su padre genera una secuencia en la que transcribe todas las notas que Carlos ha recibido en su vida, una larga cadena hecha con los símbolos de ∑ = {a,b,c,d,f} sin un orden particular. Una *a* es un sobresaliente, y una *f* es la nota más baja. Su padre plantea un criterio simple: si Carlos recibe menos que 4 *d's*, tendrá el coche, y de otra forma, no lo tendrá. Una *f* cuenta igual que una *d.*

Su padre te pide que diseñes una gramática lineal derecha donde la gramática genere la secuencia de trascripción sí y sólo si Carlos consigue el coche. Nota: Puedes probar si una gramática lineal derecha genera una palabra con el derivador de fuerza bruta, o bien convirtiendo la gramática a un autómata finito y comprobando si el AF acepta la palabra.

(a) Diseña una gramática lineal derecha con las especificaciones anteriores.

(b) La reacción inicial de Carlos fue de realizar una súplica, para que su padre adoptase métodos menos draconianos, abogando indirectamente a la noción que la diferencia generacional entre el padre y el hijo pudo haber conducido al padre a ser injustamente duro. El padre decide hacer que una nota *f*, cuente como dos *d's*. Modifica la gramática para que tenga en cuenta este cambio.

(c) Con un poco más de tacto, Carlos fue capaz de discutir satisfactoriamente con su padre, llegando al acuerdo de que una nota de sobresaliente *a*, debería mitigar un suspenso d. ¿Por qué este lenguaje no puede ser expresado con una gramática lineal derecha?

Solución:

Con el mismo alfabeto ∑ = {a,b,c,d,f} para los diferentes apartados del ejercicio. Cada uno de los símbolos del alfabeto representa una de las calificaciones que puede obtener Carlos.

a)

G_a = ({a,b,c,d,f}, {S,X,Y,Z}, S, P_a)

P_a={

S ::= aS | bS | cS | dX | fX | a | b | c | d | f

X ::= aX | bX | cX | dY | fY | a | b | c | d | f

Y ::= aY | bY | cY | dZ | fZ | a | b | c | d | f

Z ::= aZ | bZ | cZ | a | b | c

}

S genera sentencias sin ningún suspenso (d ó f).

X genera sentencias con 1 suspenso (d ó f).

Y genera sentencias con 2 suspensos (d ó f).

Z genera sentencias con 3 suspensos (d ó f).

b)

G_b = ({a,b,c,d,f}, {S,X,Y,Z}, S, P_b)

P_b={

S ::= aS | bS | cS | dX | f**Y**| a | b | c

X ::= aX | bX | cX | dY | f**Z** | λ

Y ::= aY | bY | cY | dZ | λ
Z ::= aZ | bZ | cZ | λ
}

otro conjunto de reglas equivalente, sin reglas de producción con λ sería:
P'_b={
S ::= aS | bS | cS | dX | f**Y**| a | b | c | d | f
X ::= aX | bX | cX | dY | f**Z** | a | b | c| d | f
Y ::= aY | bY | cY | dZ | a | b | c | d
Z ::= aZ | bZ | cZ | a | b | c
}

Ahora cuando aparece una f, se salta de S a Y (0 a 2 suspensos), y de X a Z (1 a 3 suspensos), y cuando se está en Y (2 suspensos) ya no se permiten f's (2 + **2** > 3 suspensos permitidos). Estos cambios afectan a las siguientes producciones:

- cambiar S::= fX por S::= fY
- cambiar X::= fY por S::= fZ
- eliminar Y ::= fZ

c)
Porque el número de sobresalientes no está limitado, y no se pueden establecer infinitas reglas para almacenar todos los casos posibles (1 a y 0 d's, 1 a y 1 d, 2 a's y 0 d's, 2 a's y 1 d, etc.). Ya que en las gramáticas regulares se necesita un símbolo No Terminal (con su conjunto de reglas asociado), para almacenar cada situación que requiera almacenar un valor concreto "en la memoria" del proceso de derivación.

7. Transformar las siguientes gramáticas regulares lineales derechas a su correspondiente Autómata Finito, utilizando JFLAP.

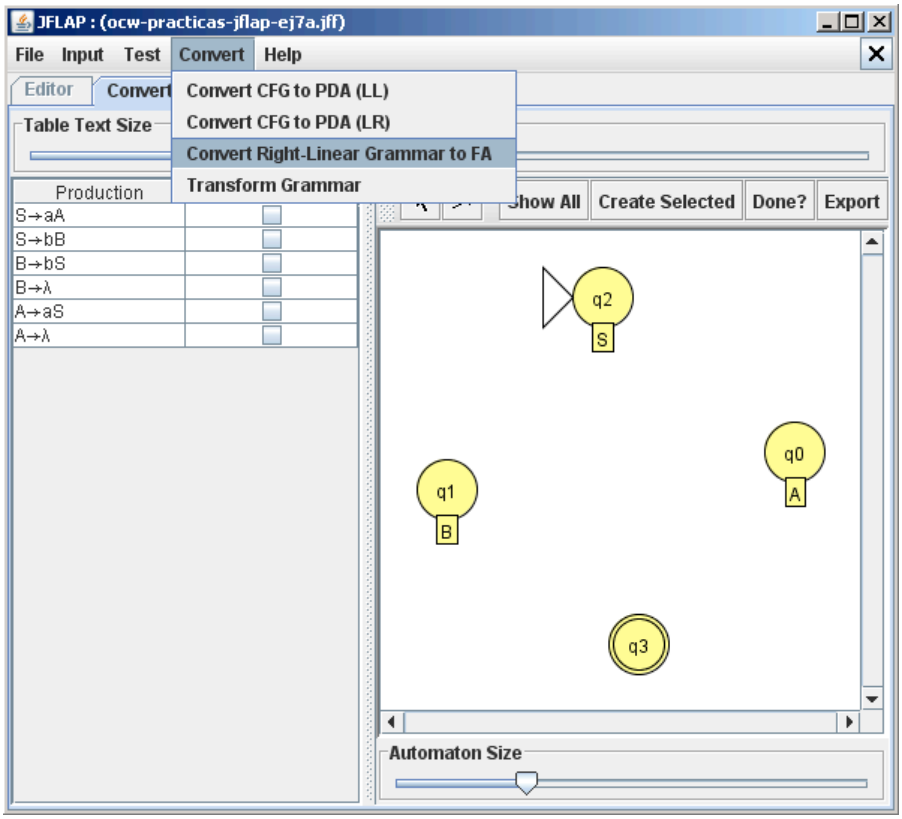
1)
S ::= aA
S ::= bB
B ::= bS
B ::= λ
A ::= aS
A ::= λ

2)
S ::= aS
S ::= aA
A ::= bB
A ::= cC
B ::= bS
B ::= b
C ::= cS
C ::= λ

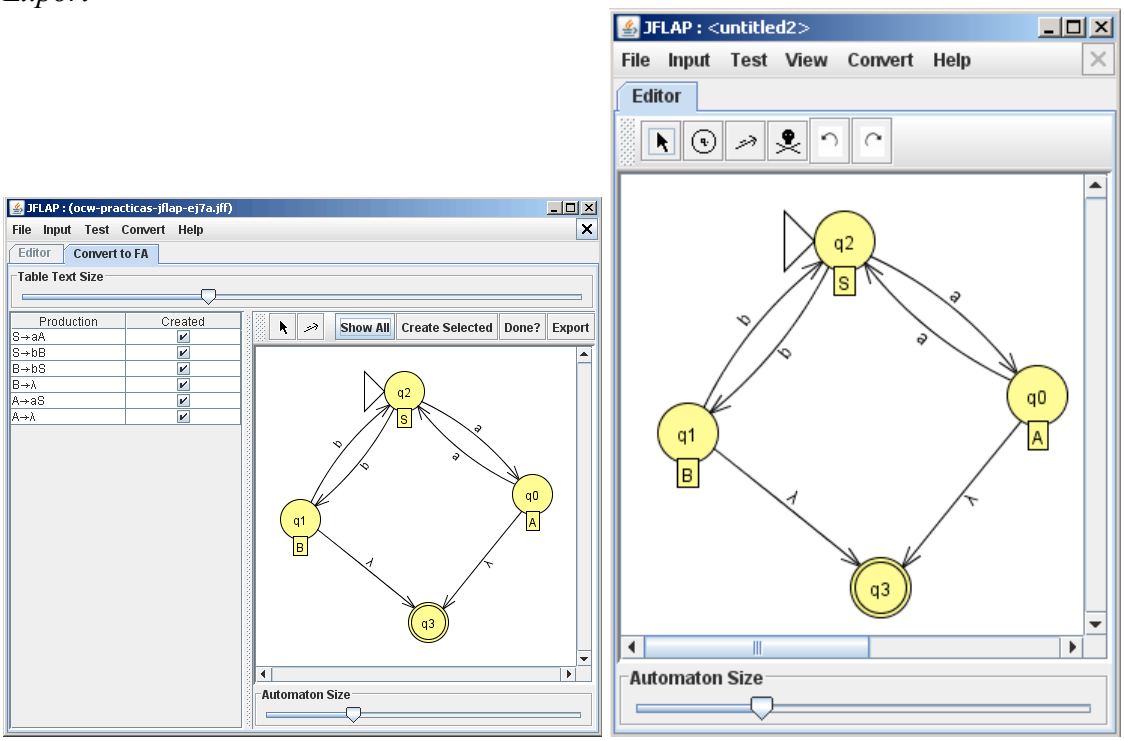
3)
S ::= bS
S ::= aA
S ::= bB
S ::= bC
A ::= aS
B ::= bC
B ::= b
C ::= cS
C ::= c

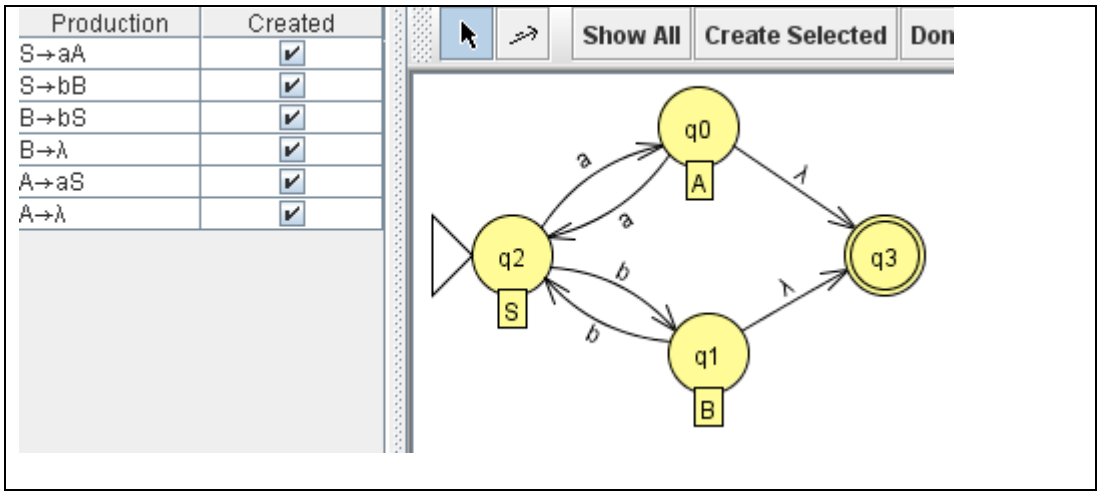
Solución:

1) Por pasos: *Convert → Convert Right-Linear Grammar to FA*

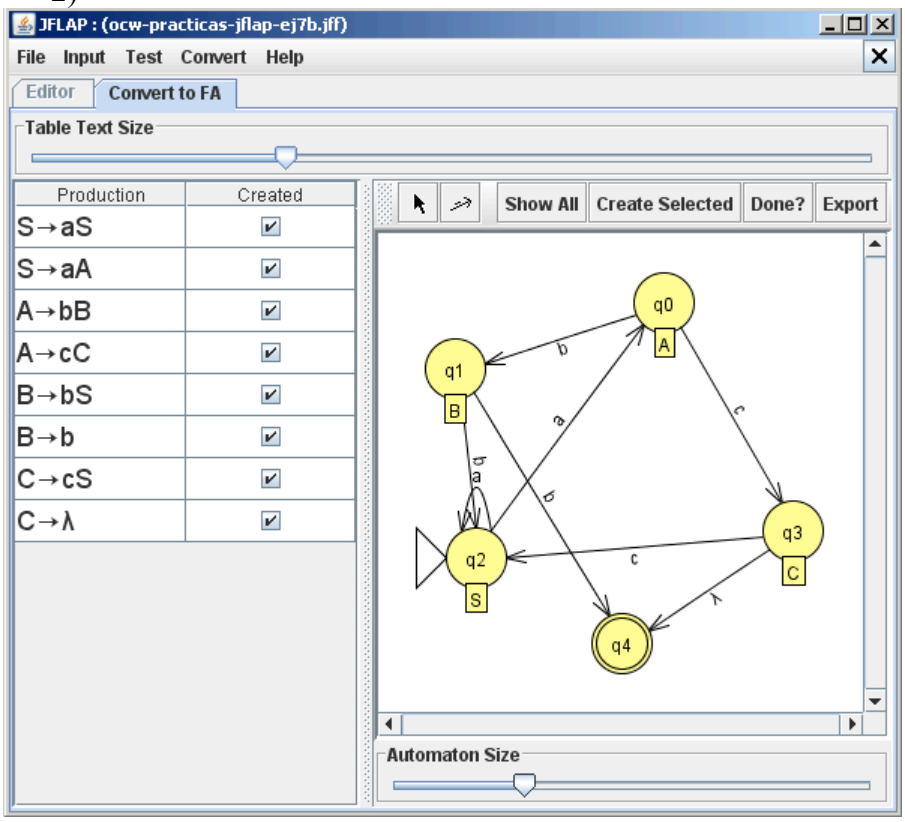


Pulsar sucesivamente:
Show all
Done?
Export

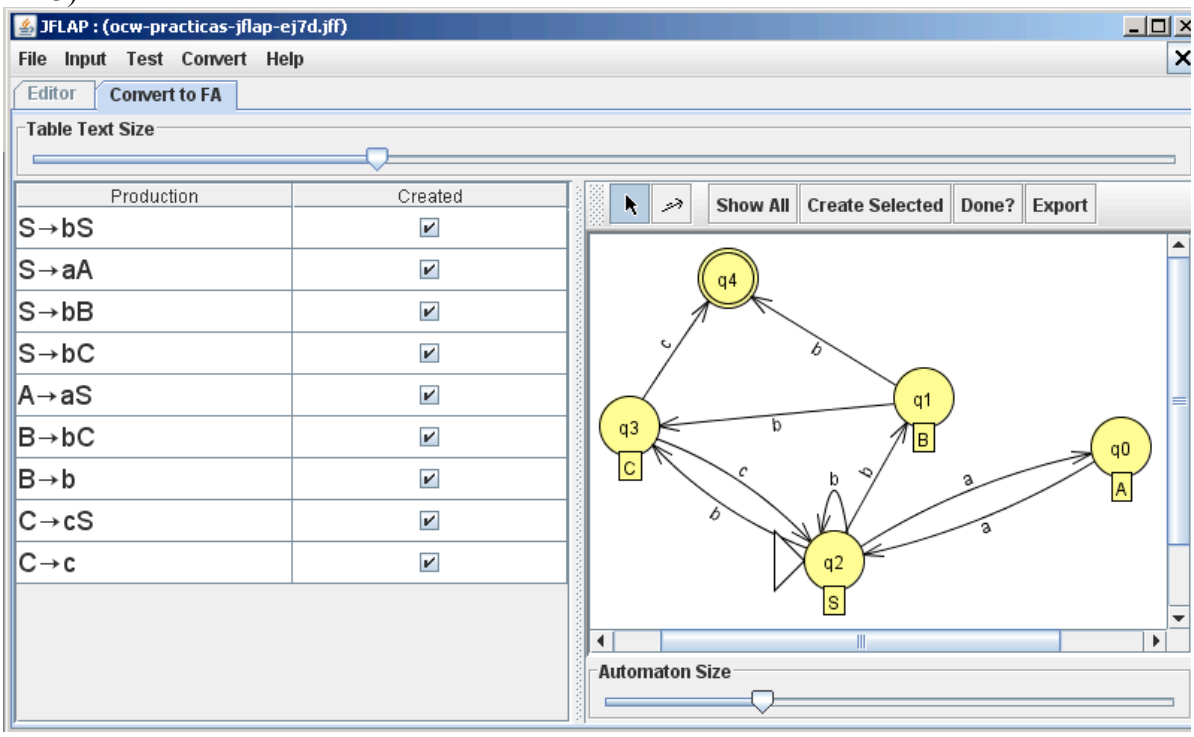




2)

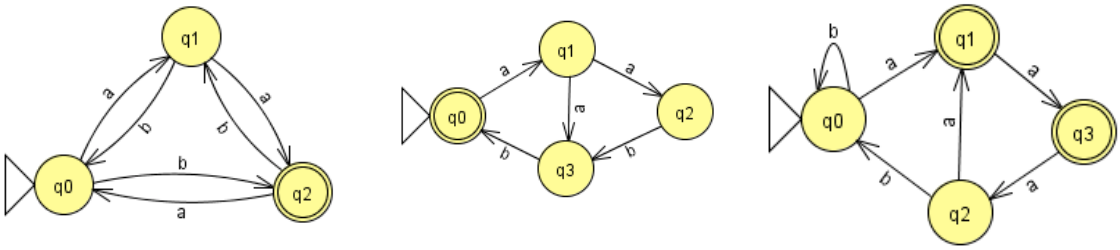

JFLAP : (ocw-practicas-jflap-ej7b.jff)
File Input Test Convert Help
Editor Convert to FA
Table Text Size
Production Created
S→aS
S→aA
A→bB
A→cC
B→bS
B→b
C→cS
C→λ
Show All Create Selected Done? Export
q0 A
q1 B
q2 S
q3 C
q4
Automaton Size

3)


JFLAP : (ocw-practicas-jflap-ej7d.jff)
File Input Test Convert Help
Editor Convert to FA
Table Text Size
Production Created
S→bS
S→aA
S→bB
S→bC
A→aS
B→bC
B→b
C→cS
C→c
Show All Create Selected Done? Export
q4
q1 B
q3 C
q0 A
q2 S
Automaton Size

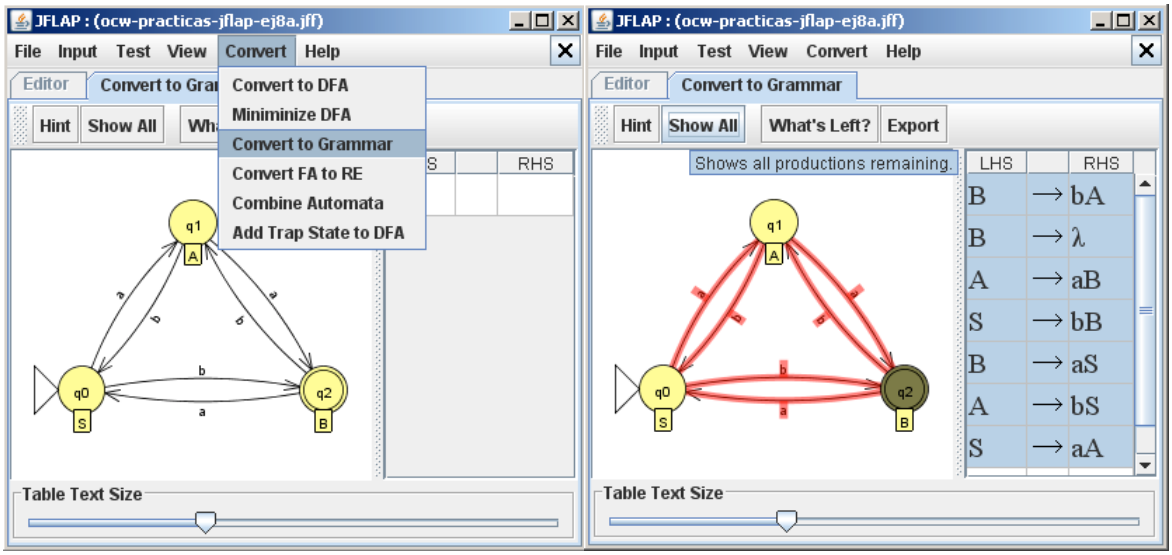
8. Edita los siguientes autómatas finitos en JFLAP y transfórmalos a sus correspondientes gramáticas, utilizando JFLAP.

1) 2) 3)

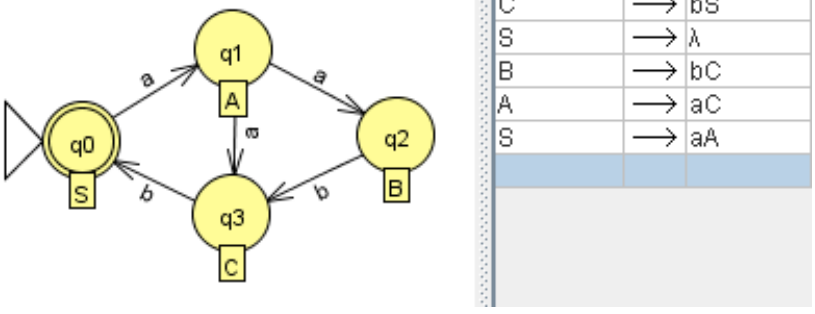


Solución:

1) Por pasos: *Convert* → *Convert to Grammar*, *Show All*, *What's Left*, *Export*.

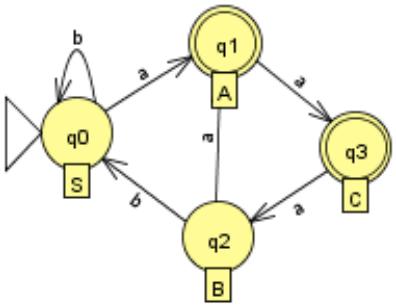


2)



A	→	aB
C	→	bS
S	→	λ
B	→	bC
A	→	aC
S	→	aA

3)



C	⟶	λ
S	⟶	bS
B	⟶	aA
C	⟶	aB
A	⟶	aC
A	⟶	λ
S	⟶	aA
B	⟶	bS

9. Eliminar las reglas no generativas de la gramática G con JFLAP. Anota los pasos de transformación y la gramática obtenida (para cada paso, anota la producción no generativa que vas a eliminar y las producciones nuevas que vas añadiendo).

G=({a,b,c},{S,A,B},S,P)

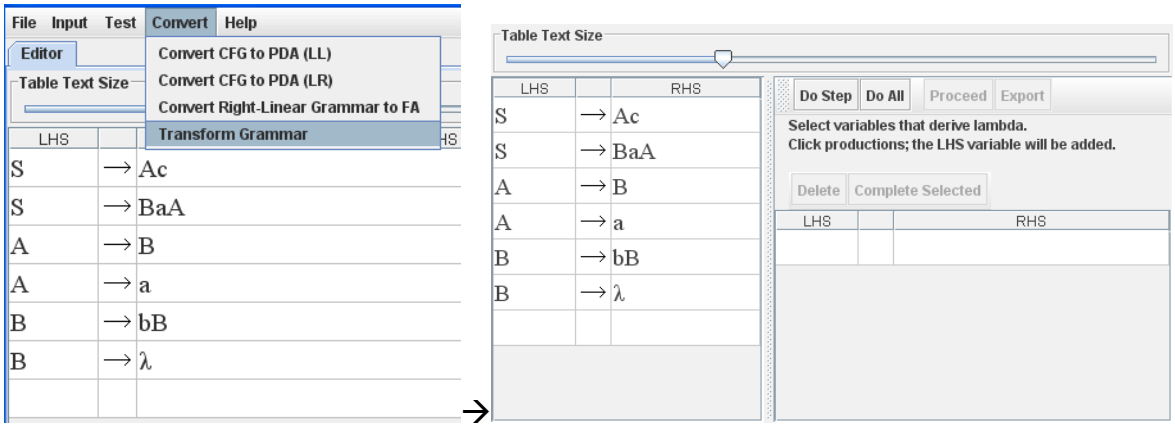
P={ S ::= Ac | BaA

A ::= B | a

B ::= bB | λ }

Solución:

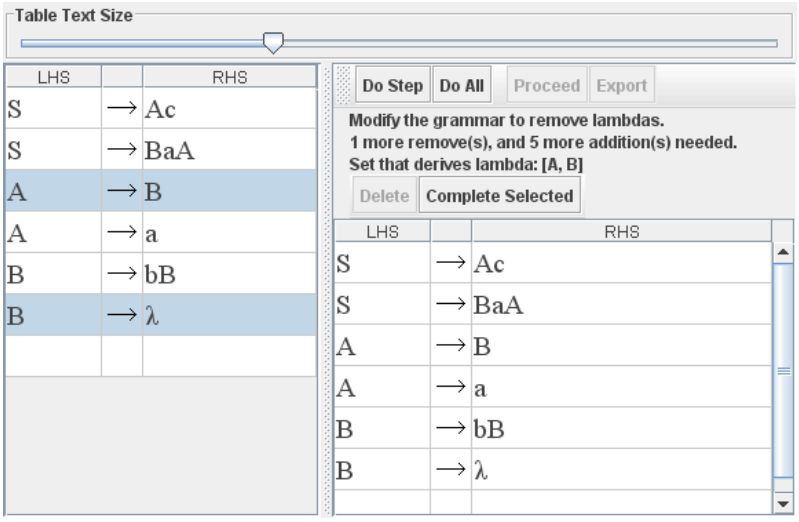
En JFLAP, opción *Convert → Transform Grammar*:



a) Seleccionar (en la pantalla de la izquierda) los no terminales que generan lambda: [B,A].

B por B::= λ

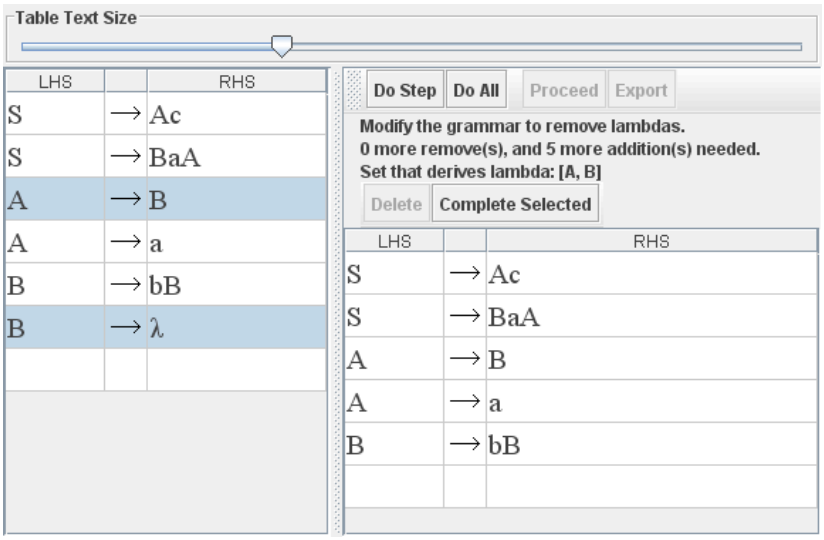
A por A ::= B y B::= λ



b) Eliminar y añadir producciones:

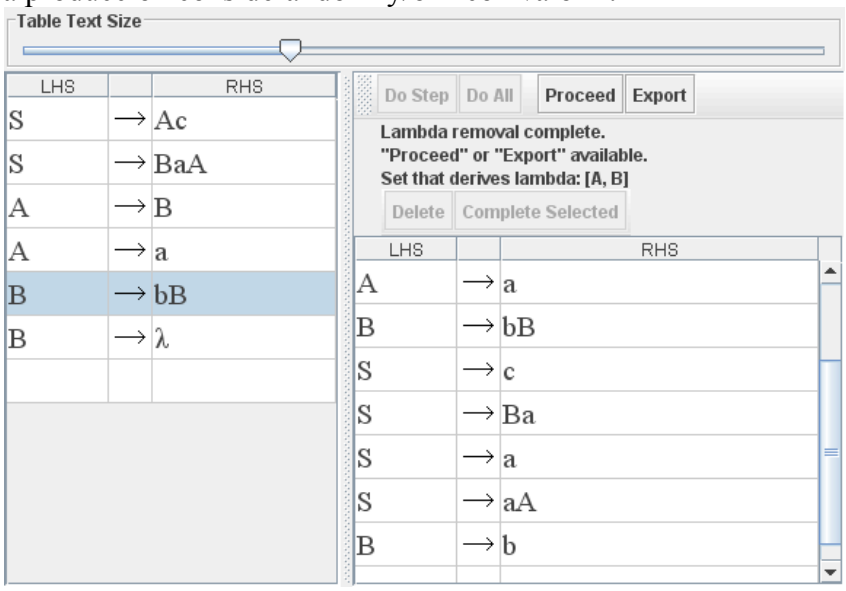
Eliminar: B ::= λ.

En JFLAP, seleccionar en la pantalla de la derecha y pulsar *Delete*, y desaparece de la gramática resultado.

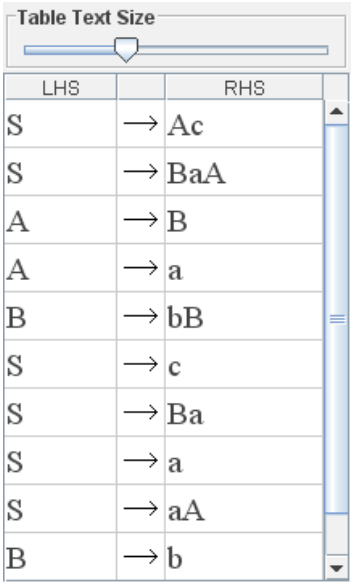


Añadir:
S::=c (por S::=Ac, con A= λ)
S::=a (por S::=BaA, con A= λ y B= λ)
S::=Ba (por S::=BaA, con A= λ)
S::=aA (por S::=BaA, con B= λ)
B::=b (por B::=bB, con B= λ)

En JFLAP, ir seleccionando la producción origen y pulsar *Complete Selected*, para generar la nueva producción considerando A y/o B con valor λ:



A continuación, pulsar *Export*, para obtener la nueva gramática sin producciones no generativas en una pantalla nueva, desde la que poder realizar nuevas operaciones (transformaciones, guardar, verificar la generación de palabras, etc.):



10. Eliminar las reglas superfluas y símbolos inaccesibles usando JFLAP en la gramática G. Anota los pasos de transformación y la gramática obtenida.

G=({a,b,c},{S,A,B},S,P)

P={ S ::= aSB | b

A ::= a | aA

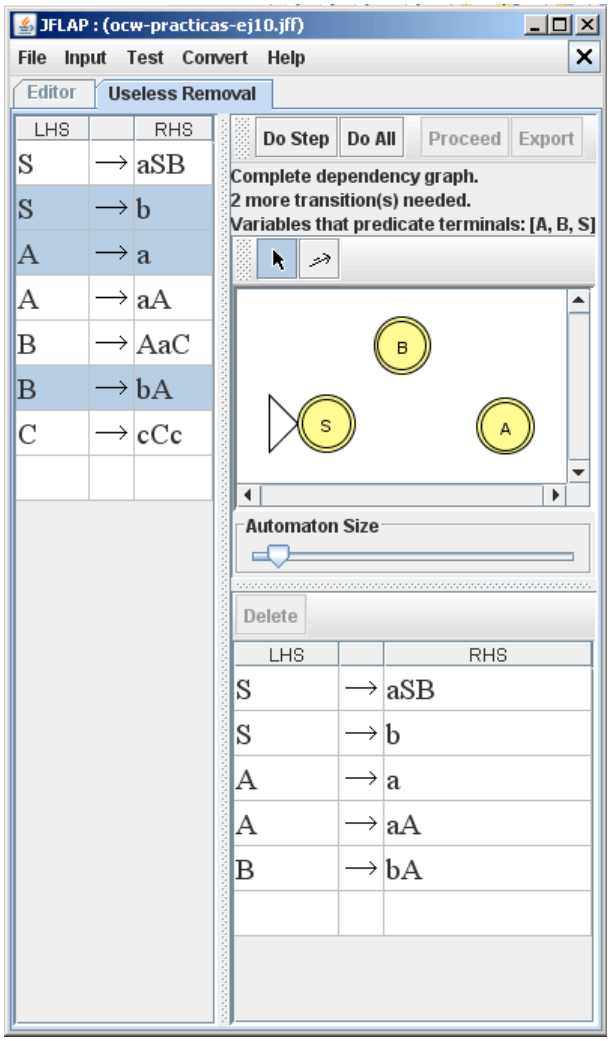
B ::= AaC | bA

C ::= cCc}

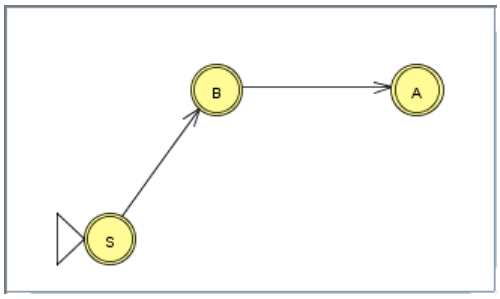
Solución:

En JFLAP, opción *Convert → Transform Grammar*:

a) Seleccionar no terminales que derivan en terminales (producciones marcadas en azul), que son los que sí pueden generar palabras:



b) Se construye el grafo de dependencias entre los nodos (no terminales), incluyendo las dependencias (o transiciones) a mano, o pulsando en *Do Step*.



c) En este grafo se determina que tanto el no terminal 'C', como el terminal 'c' no son accesibles, por lo que se puede prescindir de ellos. Las producciones en las que intervienen ambos son las reglas superfluas, porque también son prescindibles. Pulsando *Export* finalizamos el proceso: