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.

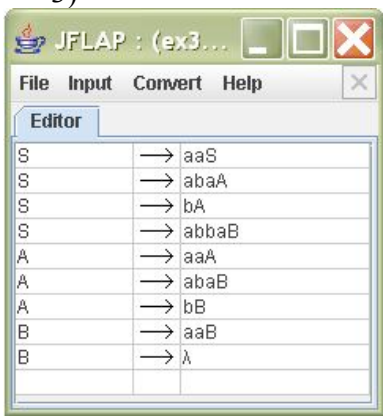
Palabras de prueba	0	00	00.1	3445	1.23	9.21.2	9..1	9,1

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.

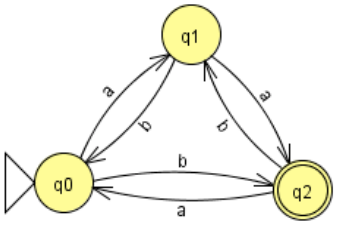
1)
S ::= aA
S ::= bB
B ::= bS
B ::= λ
A ::= aS
A ::= λ

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

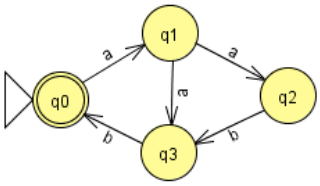
3)
S ::= bS
S ::= aA
S ::= bB
S ::= bC
A ::= aS
B ::= bC
B ::= b
C ::= cS
C ::= c

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

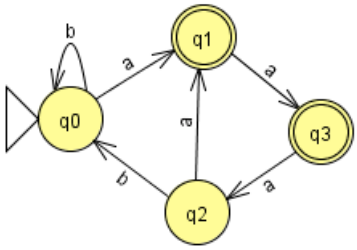
1)



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}