Modelos de la Computación (2014-2015) Grupo: B3
Grado en Ingeniería Informática
Universidad de Granada

Cuaderno de Practicas
6 problemas entregados

Carlos de la Torre Fanin

7 de febrero de 2015

Índice

1. Practica 1 ..... Corregida **1**
1.1. Objetivos de la Practica . . . 1
1.2. Ejercicio a resolver, Gramática . . . 1
1.3. Resolución del Ejercicio, Lenguaje . . . 1
1.3.1. Primer ejemplo . . . 2
1.3.2. Segundo ejemplo . . . 2
1.4. Conclusión del Lenguaje . . . 2

2. Practica 2 ..... Corregida **3**
2.1. Objetivos de la Practica . . . 3
2.2. Ejercicio a resolver, Gramática . . . 3
2.3. Resolución del Ejercicio, Lenguaje . . . 3
2.4. Conclusión del Lenguaje . . . 4

3. Practica 3 ..... Corregida **4**
3.1. Objetivos de la Practica . . . 4
3.1.1. Autómata Finito No Determinista - AFND[2] . . . 4
3.1.2. Autómata Finito Determinista - AFD[1] . . . 4
3.2. Ejercicio a resolver, Autómatas . . . 4
3.3. Resolución del Ejercicio, Conversión . . . 5
3.3.1. Primer ejemplo . . . 5
3.3.2. Segundo ejemplo . . . 5
3.4. Conclusión . . . 6

4. Practica 4 ..... Corregida **6**
4.1. Objetivos de la Practica . . . 6
4.2. Ejercicio a resolver . . . 6
4.3. Resolución del Ejercicio . . . 6
4.3.1. Primer ejemplo . . . 6
4.3.2. Segundo ejemplo . . . 7
4.4. Conclusión del Lenguaje . . . 7

5. Practica 5 ..... Corregida **7**
5.1. Objetivos de la Practica . . . 7
5.2. Ejercicio a resolver . . . 7
5.3. Resolución del Ejercicio . . . 8
5.3.1. Primer ejemplo . . . 8
5.3.2. Segundo ejemplo . . . 9
5.4. Conclusión . . . 9

6. Practica 6 ..... Corregida **9**
6.1. Objetivos de la Practica . . . 9
6.2. Ejercicio a resolver . . . 9
6.3. Resolución del Ejercicio . . . 9
6.3.1. Primer ejemplo . . . 10
6.3.2. Segundo ejemplo . . . 12
6.4. Conclusión . . . 12

Índice de figuras

3.1. Autómatas Finitos No Deterministas . . . 4
3.2. Autómatas Finitos Deterministas . . . 5
5.1. Intersección de Autómatas . . . 8
5.2. . . . 8
6.1. Pasos a seguir en la tabla de minimización . . . 11
6.2. Pasos a seguir en la tabla de minimización . . . 12
6.3. Tabla y autómata final . . . 12

6.4. Ejercicio explicado en clase 14
6.5. Ejercicio explicado en clase $L1L2 \cup L2L1$ 14

1. Practica 1 ..... Corregida

1.1. Objetivos de la Practica

En esta primera practica veremos algunos conceptos principales como gramática, lenguaje, autómatas... etc, para ello lo que se realizará en esta practica es extraer desde una gramática en concreto, su lenguaje derivado, pero.., ¿ que es una gramática ?, una gramática la definimos como una cuádrupla de diferentes componentes entre los que se encuentran:

- Alfabeto de variables: que no es mas que un conjunto de variables, que se denotan por la letra V.
- Alfabeto de símbolos terminales: que no es mas que un conjunto de símbolos u objetos que podemos usar para la resolución del problema, y se denota por la letra T.
- Reglas de producción: que son las parejas de símbolos y variables, que podemos encontrar en el conjunto $(V \cup T)^*$, se suelen representar $\alpha \to \beta$
- Símbolo de partida: este ultimo componente tiene que ser necesariamente un elemento de V, se suele representar con la letra S.

Bien ya sabemos lo que es una gramática ahora ¿ Que es el lenguaje ?.
Técnicamente un lenguaje es el conjunto de cadenas formadas por símbolos terminales que existen en una gramática y que se pueden construir a partir de una serie de reglas de producción previamente fijadas.

1.2. Ejercicio a resolver, Gramática

De a cuerdo ya tenemos las definiciones principales, para poder describir cual es nuestro problema, en esta practica se pretende descubrir cual es el lenguaje resultante, a partir de una gramática dada.
En este caso la gramática será: (Variables, Símbolos, Inicio, Reglas) donde las variables pueden ser cualquiera que pertenezca al conjunto $\{S, A, B\}$, los símbolos pueden ser cualquiera del conjunto $\{a, b\}$, el inicio será S y las reglas de producción serán cualquiera que pertenezca al conjunto:

$$\{S \to aB, S \to bA, A \to a, B \to b, A \to aS, B \to bS, A \to bAA, B \to aBB\}$$

1.3. Resolución del Ejercicio, Lenguaje

Para poder extraer el lenguaje resultante desde esta gramática, no hace falta ser un genio, la verdad es que es algo bastante trivial, el gran problema llega cuando, nos damos cuenta de las diferentes ambigüedades que surgen de la propia naturaleza de los lenguajes.

Y es que, primero nos damos cuenta de la gran cantidad de posibilidades que surgen, cuando el conjunto de símbolos empieza a crecer, ya que el conjunto con 2 elementos tiene 2^2 posibilidades, 4 en total, pero es que el conjunto con 5 símbolos, o para que todo el mundo lo entienda, los faros, las ruedas, los asientos, las puertas y el volante de un coche, el numero de posibilidades se dispara a 5^5, que son un total de 3125 posibilidades.

Pero claro, tal y como podemos darnos cuenta por el ejemplo anterior no todas las combinaciones son posibles, ya que un asiento no puede ir en el lugar de una rueda, para que esto no ocurra existen las reglas de producción, como su propio nombre indica son reglas, y sirven para tener un orden a la hora de generar nuestro lenguaje.

Según los símbolos terminales que tenemos en el conjunto que hemos visto anteriormente en el lenguaje resultante solo podremos obtener cadenas compuestas por a, b ahora bien, ¿ Podemos obtener cualquier cadena que contenga estos símbolos ?, y ¿ en cualquier numero ?, para eso tenemos que fijarnos en las reglas de producción.

A continuación pasaremos a explicar mediante ejemplos la manera en la cual se extrae el lenguaje de esta gramática:

1.3.1. Primer ejemplo

Como ya hemos explicado antes existen unas variables de un tipo en concreto llamadas *Símbolo Inicial* que es desde donde comienza a construirse una posible cadena que será aceptada por nuestro lenguaje, quiero que quede claro que este procedimiento que vamos a explicar a continuación, no es la forma en la que realmente se tendría que utilizarse nuestro lenguaje, ni la gramática que tenemos, ya que el lenguaje esta concebido para reconocer cadenas correctas, según la gramática que tenemos, osea que nosotros tendríamos una serie de cadenas ya construidas y utilizaríamos la gramática dada para comprobar que dichas cadenas pertenecen al lenguaje que tenemos como correcto. Pero para conocer el proceso de lectura de cadenas utilizamos la gramática para construir cadenas que serán aceptadas por nuestro lenguaje de manera inmediata.

Al fijarnos en las reglas de producción solo tenemos dos reglas que contengan el símbolo inicial S así que cogemos cualquiera de ellas en este caso $S \to aB$ que contiene el símbolo terminal a y la variable B, al ver estas primeras reglas de producción podemos darnos cuenta rápidamente, que nuestro lenguaje solo aceptara las cadenas que contenga como mínimo dos símbolos terminales, ya que no hay ninguna regla de producción con una variable S (símbolo inicial) a la izquierda de la regla y solo un símbolo terminal a la derecha de la regla.

$$Regla \mapsto Resultado \mapsto Regla = ResultadoFinal$$
$$[S \to aB] \mapsto [aB] \mapsto [B \to b] = [ab]$$

1.3.2. Segundo ejemplo

En esta primera aproximación podemos ver que tenemos dos símbolos terminales y que hay el mismo numero de a como de b, pero esta claro que esta cadena es muy sencilla, vamos a pasar al siguiente nivel y escogeremos unas reglas de producción que nos de como resultado una cadena de al menos 6 símbolos, esta claro que después de ver el apartado anterior las cadenas resultantes contendrán un numero par de símbolos terminales.

$$\left.\begin{array}{c} S \to aB \mapsto aB \\ B \to aBB \mapsto aaBB \\ B \to aBB \mapsto aaaBBB \\ B \to b \mapsto aaabBB \\ B \to b \mapsto aaabbB \\ B \to b \mapsto aaabbb \end{array}\right\} \text{Como podemos ver aplicando las diferentes reglas de producción podemos llegar a las diferentes cadenas que serán aceptadas por nuestro lenguaje}$$

Quiero prestar especial atención a este ejemplo puesto que nos da una pista muy interesante del lenguaje que resulta de la gramática que estamos estudiando, y es que en la segunda y tercera linea, podemos ver que cada vez que aplicamos la regla de producción pertinente, lo que estamos haciendo es añadir un símbolo terminal a, pero también añadimos dos variables B lo cual implica que sin mas remedio que el lenguaje tendrá que aceptar cadenas que tengan el mismo numero de símbolos terminales a que b.

Por ultimo las dos ultimas reglas de producción que nos faltan por probar son aquellas que nos permiten intercalar los símbolos terminales, así pues aun manteniendo las restricciones propias de las anteriores reglas de producción podríamos conseguir cadenas de símbolos mezclados, tal y como mostramos en el siguiente ejemplo:

$$\left.\begin{array}{c} S \to aB \mapsto aB \\ B \to aBB \mapsto aaBB \\ B \to bS \mapsto aabSB \\ S \to aB \mapsto aabaBB \\ B \to b \mapsto aababB \\ B \to b \mapsto aababb \end{array}\right\} \text{En este ejemplo podemos ver que aunque el orden en el que están colocados los símbolos terminales no es igual que el anterior, si es verdad que se sigue manteniendo la premisa de que hay el mismo numero de } a \text{ que de } b$$

1.4. Conclusión del Lenguaje

Después de haber visto los ejemplos anteriores podemos deducir que el lenguaje resultante podría ser algo tal que así:

$$L(G) = \{u \mid u \in \{a,b\}^+ \ y \ N_a(u) = N_b(u)\}$$

Esto se lee: El lenguaje L resultante de la gramática G es igual al conjunto formado por las palabras u tal que u pertenezca al conjunto formado por las parejas de símbolos a,b siempre en cantidades positivas y siendo el número de símbolos terminales a contenidos en u igual al número de símbolos terminales b contenidos en u
En esta simple formula tendríamos acotadas todas las palabras resultantes de la gramática estudiada.

2. Practica 2 ..... Corregida

2.1. Objetivos de la Practica

En esta practica se pretende, analizar las diferentes gramáticas que se nos presentan para poder etiquetarlas según la jerarquía de Chomsky[3], Noam Chomsky es un lingüista que determino que había 4 clases de gramáticas, las del tipo 0 que cubrían cualquier tipo de lenguaje, las del tipo 1 que son aquellas dependientes del contexto, las del tipo 2 aquellas que son libres del contexto, y por ultimo las del tipo 3 que son los conjuntos regulares.
Entendiendo estos conceptos tenemos que dada una gramática saber encontrar al tipo de lenguaje al que pertenece.

2.2. Ejercicio a resolver, Gramática

El ejercicio propuesto para esta practica es dada la gramática que se puede encontrar en la pagina 89 de las diapositivas del tema 1, saber a que tipo de lenguaje pertenece dicha gramática.

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

$$P = \{S \to AB, A \to Ab, A \to a, B \to cB, B \to d\}$$

2.3. Resolución del Ejercicio, Lenguaje

Para resolver este ejercicio tenemos que tener claros algunos conceptos de la jerarquía de Chomsky, por ejemplo para poder detectar el tipo de lenguaje de una gramática nos tenemos que fijar en sus reglas de producción.
La manera mas fácil de detectar a que tipo pertenece es descartando, osea, que nos vamos al tipo de lenguaje 3 según la jerarquía de Chomsky que son los lenguajes regulares y por lo tanto los mas fáciles de implementar, y miramos si sus reglas de producción coinciden con todas las reglas de producción de nuestra gramática, ya que si coincidieran todas menos una regla, nuestra gramática seria del tipo 2, libre del contexto, por lo tanto en principio deberíamos tener una memoria para poder producir las cadenas de nuestra gramática.
Para mostrar cual es el lenguaje que genera esta gramática realizaremos esta sencilla aplicación de las reglas de producción:

$$S \to AB \Longrightarrow AB \quad A \to Ab \Longrightarrow AbB \quad A \to Ab \Longrightarrow AbbB \quad A \to Ab \Longrightarrow AbbbB \quad A \to a \Longrightarrow abbbB$$
$$B \to cB \Longrightarrow abbbcB \quad B \to cB \Longrightarrow abbbccB \quad B \to cB \Longrightarrow abbbcccB \quad B \to d \Longrightarrow abbbcccd$$

Pero si nos fijamos un poco mas a fondo, y intentamos realizar una optimización de las reglas podemos darnos cuenta de que la gramática actual aunque pertenece al tipo 2, también podemos construir una gramática del tipo 3 y que resuelva el mismo lenguaje por ejemplo:

$$P = \{S \to aB, B \to bB, B \to C, C \to cC, C \to d\}$$

Si bien es cierto que la regla de producción $B \to C$ en un principio pudiera parecer que no pertenece a una gramática de tipo 3, hay que tener en cuenta de que antes de la variable C podemos tener un símbolo terminal como este ε que claramente es el vacío, por lo tanto esa regla seria realmente $B \to \varepsilon C$, pero claro esta por motivos obvios es un símbolo que no se muestra.
Entonces al final tendremos algo parecido ha esto para construir la misma cadena:

$$S \to aB \Longrightarrow aB \quad B \to bB \Longrightarrow abB \quad B \to bB \Longrightarrow abbB \quad B \to bB \Longrightarrow abbbB \quad B \to C \Longrightarrow abbbC$$
$$C \to cC \Longrightarrow abbbcC \quad C \to cC \Longrightarrow abbbccC \quad C \to cC \Longrightarrow abbbcccC \quad C \to d \Longrightarrow abbbcccd$$

2.4. Conclusión del Lenguaje

En definitiva, ambas gramáticas pueden representar el mismo lenguaje, y dependiendo de la pericia que tenga el desarrollador de la gramática conseguiremos un problema resuelto de una manera mas fácil y económica, o bien un problema resuelto de forma mas difícil y caro.
Como punto final exponemos aquí cual es el lenguaje que sale a partir de cualquiera de las dos gramáticas:

$$L = \{ab^i c^j d : i, j \in N\}$$

3. Practica 3 ..... Corregida

3.1. Objetivos de la Practica

El objetivo de esta practica es entender la forma en la que un autómata **no** determinista se puede pasar a un autómata determinista con sus claras diferencias. Para tener claras estas diferencias vamos a explicar brevemente que es un autómata no determinista un uno determinista:

3.1.1. Autómata Finito No Determinista - AFND[2]

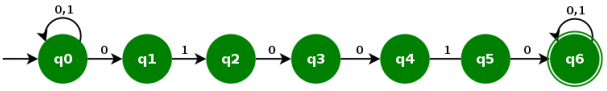
La característica mas relevante de un autómata no determinista es que, estando en un estado cualquiera de su ejecución, tiene la posibilidad de leyendo un único símbolo del alfabeto, puede utilizar diferentes transiciones hacia diferentes estados y todas esas transiciones serian validas, ya que desde un mismo estado se puede ir a varios estados diferentes, sin que tenga una lógica clara, osea, que la elección de la transición depende de elementos arbitrarios del propio autómata, de ahí que se no determinista.
Por supuesto hay otras características que hacen compatible este tipo de autómatas con uno determinista, osea que, puede darse perfectamente la situación en la que un autómata no determinista lea el mismo lenguaje que un autómata determinista.

3.1.2. Autómata Finito Determinista - AFD[1]

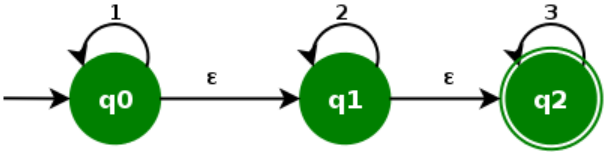
Al contrario que AFND la característica mas relevante de AFD es que no depende de situaciones arbitrarias para la transición entre los diferentes estados, osea, que podemos ver desde el comienzo de la lectura de los símbolos del alfabeto cual sera el camino que seguirá nuestro autómata, ya que tiene un camino muy bien definido según los símbolos que se leen desde la cadena de caracteres y las transiciones que existen entre los diferentes estados del autómata.

3.2. Ejercicio a resolver, Autómatas

Para entender correctamente el funcionamiento, de los autómatas y su conversión entre uno no determinista y uno determinista utilizaremos los ejemplos que están en las paginas 31-54 y 68-77 del tema 2 de teoría, en ellas se muestra paso a paso como es posible pasar dos autómatas no determinista a determinista tal y como se muestra en las siguientes figuras 3.1 y 3.2.
Hay que tener en cuenta que en las imágenes lo que se muestran son los mismos ejemplos que en las paginas del tema 2 pero se les ha cambiado los nombres a los diferentes estados, para que el autómata de ïnicioz el del "final"tengan el mayor numero de similitudes, respetando siempre, por supuesto que son dos tipos de autómatas diferentes.

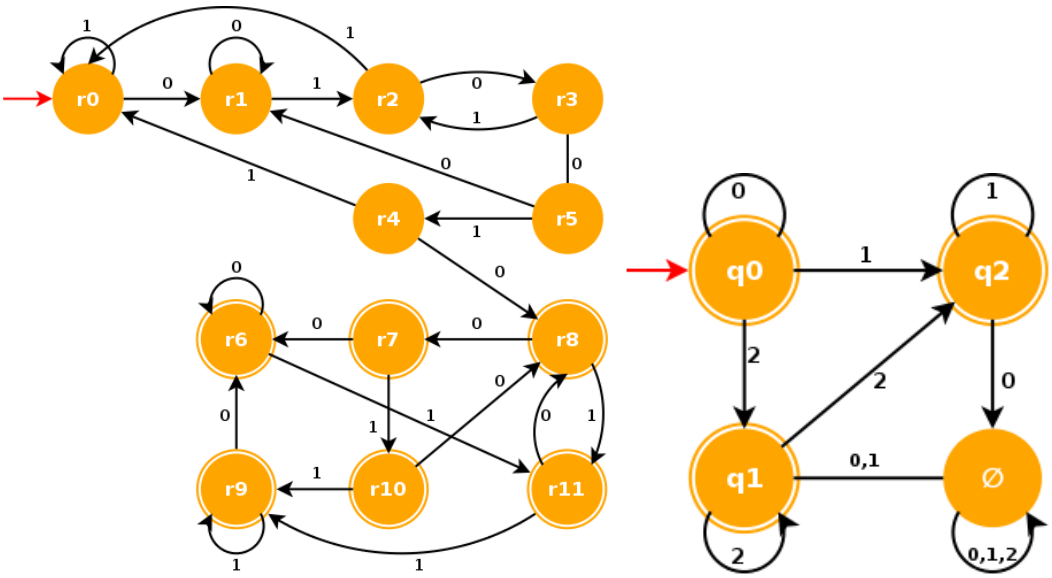


(a) Ejemplo de la página 31 Tema 2



(b) Ejemplo de la página 68 Tema 2

Figura 3.1: Autómatas Finitos No Deterministas



(a) Ejemplo de la página 54 Tema 2 (b) Ejemplo de la página 77 Tema 2

Figura 3.2: Autómatas Finitos Deterministas

3.3. Resolución del Ejercicio, Conversión

Lo que vamos ha hacer es explicar paso a paso como pasar desde un AFND a un AFD.

3.3.1. Primer ejemplo

Tenemos un autónoma finito no determinista sin transiciones nulas el cual acepta cadenas con un numero indeterminado de 0 o de 1 y a continuación viene la cadena 010010 y un símbolo mas que puede ser o un 0 o un 1, como ya hemos dicho con anterioridad el problema de los AFND es que pueden darse situaciones que no sepamos bien en que estado se encuentra y eso es lo que le pasa al autómata de nuestro ejemplo. En este ejemplo lo que ocurre si lo analizamos es lo siguiente:
En el estado de inicio $q0$ el autómata puede leer un símbolo 0, y leyendo este símbolo tiene dos caminos diferentes para seguir, de ahí que sea no determinista, entonces en el autómata finito determinista generaremos un nuevo estado el cual sera la unión entre los dos estados a los que se puede ir desde $q0$ aplicando la transición 0 osea el nuevo estado seria la unión del resultado de aplicar la transición 0 al estado $q0$ osea $r1 = q0 \cup q1$ (figura 3.2a), pero claro esta en el estado $q0$ también nos podemos encontrar con la transición 1 que lo que hace en este caso es solo quedarse en el estado en donde nos encontramos por lo tanto agregamos esa transición al AFD (figura 3.2a), ya lo que nos quedaría seria seguir este mismo proceso para todos los estados nuevos que fuéramos generando, osea tendríamos que aplicar ambas transiciones 0 y 1 al estado $r1$ que hemos generado nuevo y ver hacia donde nos conduce.

3.3.2. Segundo ejemplo

En este ejemplo tenemos un autómata finito no determinista con transiciones nulas lo primero que tenemos que explicar es ¿que es? una transición nula o lo que es lo mismo que es lo que significa el símbolo ε. Básicamente lo que nos permite este símbolo es poder pasar de un estado a otro del autómata sin necesidad de leer ningún símbolo del la cadena de caracteres. La utilidad de este tipo de transiciones es muy clara ya que nos permiten construir un álgebra para autómatas, osea que nos permiten sumar diferentes autómatas sin que sea muy complicado.
Otra cosas que tenemos que tener clara es que en el ejemplo que nos ocupa (figura 3.1b) al tener las transiciones nulas entre los estados $q0$ y $q1$ y también entre los estados $q1$ y $q2$, cualquiera de estos tres estados puede considerarse un estado inicial del autómata ya que al poder moverse entre los estados sin leer ningún símbolo de la cadena de caracteres serian iniciales, ahora bien, no es lo mismo que empiece por el estado $q0$ que por el estado $q2$, ya que, empezando por el estado $q0$ esta cadena de símbolos seria aceptada (0011222) mientras que empezando por el el estado $q2$ esta cadena no seria aceptada (2220011) puesto que el autómata no tiene la posibilidad de retroceder entre estados.

3.4. Conclusión

Hay que tener claro que se puede realizar conversiones entre autómatas, y que estas son muy útiles para cuando los estamos programando, ya que es bastante mas fácil empezar desarrollando el AFND para luego con diferentes pasos, AFND-ε , AFD y por ultimo AFD minimal y así poder llegar a nuestro objetivo sin tener que *esforzarnos* en exceso.

4. Practica 4 ..... Corregida

4.1. Objetivos de la Practica

Implementar un pequeño programa de LEX que nos permita reconocer una serie de cadenas y nos de como resultado si son variables, enteros, o números reales, básicamente lo que se pretende es aprender minimamente a utilizar la sintaxis y el funcionamiento de un programa LEX.

4.2. Ejercicio a resolver

El ejercicio a resolver esta extraído de las diapositivas del profesor, mas concretamente de la pagina 216 del tema 2, ahí es donde se encuentra el código que se muestra a continuación:

```
1  car [a-zA-Z]
2  digito [0-9]
3  signo (\-|\+)
4  suc ({digito}+)
5  enter ({signo}?{suc})
6  real1 ({enter}\.{digito}*)
7  real2 ({signo}?\.{suc})
8    int ent=0, real=0, ident=0, sumaent=0;
9  %%
10   int i;
11
12 {enter}{
13   ent++;
14   sscanf(yytext," %d",&i);
15   sumaent += i;
16   printf("Numero entero %s\n",yytext);
17 }
18 ({real1}|{real2}){
19   real++;
20   printf("Num. real %s\n",yytext);
21 }
22 {car}({car}|{digito})*{
23   ident++;
24   printf("Var. ident. %s\n",yytext);
25 }
26 . {;}
27 \n {yywrap(); return 0;}
28 %%
29 yywrap(){
30   printf("Numero de Enteros %d, reales %d, ident %d, Suma de Enteros %d\n",ent,real,ident
       ,sumaent);
31   return 1;
32 }
```

4.3. Resolución del Ejercicio

4.3.1. Primer ejemplo

En este ejemplo es bastante fácil resolver el ejercicio puesto que lo único que hay que hacer es entender la filosofía de trabajo de LEX que se puede considerar otro lenguaje mas de programación, de echo es un lenguaje de programación de autómatas.
Si nos fijamos al igual que cualquier programa C podemos encontrarnos en la parte superior la definición de las variables (lineas desde 1 hasta 10) del programa y a continuación viene el desarrollo del programa (lineas 12 hasta 28) y por ultimo nos encontramos una salida por consola que nos dice cual es el numero total de las variables que se han detectado y del tipo que se han detectado.
Para tener claro como se crea un fichero lex vamos a describir todas las lineas del código para entender

como es posible crear este analizador sintáctico. La primera linea que nos encontramos es una variable que se iguala a una expresión regular la cual aceptaría todas las letras del abecedario tanto en mayúsculas como en minúsculas, después nos encontramos otra variable que se encarga de los numeros comprendidos entre el 0 y el 9, esto no quiere decir que solo acepte esos 10 números si no que cualquier cadena constituida por los caracteres que se encuentran dentro de este rango seria aceptada.
La siguiente variable indica que la variable signo solo puede estar compuesta por solo dos símbolos que son - y + ya que estos llevan el carácter de escape ˙
La linea 4 nos indica que la variable suc puede ser una sucesión de números en cualquier orden ya que el + que se utiliza después de la variable dígito indica que se puede repetir tantas veces como sea necesario los caracteres aceptados en la variable dígito, por lo tanto esto sera verdadero hasta que el símbolo que se lea no sea aceptada por esa expresión regular de dígito.
La linea 5 tiene la variable enter que significa entero y esta a su vez contiene a las subcadenas signo y suc separadas por un ? que lo que indica que el signo solo se puede repetirse o 1 o ninguna vez antes de que aparezca cualquier número, ya que la variable suc es precisamente eso lo que indica.
Las dos lineas siguientes contienen las variables real1 y real2 estas variables hacen referencia a las expresiones regulares usadas para leer números reales con y sin signo.
Después de estas lineas tenemos la definición de variables globales que nos servirán para controlar la ejecución de nuestro programa, en este caso la cantidad de, numeros enteros (ent), reales (real), variables (ident) y las sumas que se realizan en la cadena leída.
La linea 9 es como un separador de las variables globales del programa con la propia estructura del programa lex osea que a partir de aquí tenemos el bloque de ejecución de lex (lineas 12-28) y las variables locales del programa (linea 10).
A continuación lo que hacemos es como pequeñas funciones donde realizamos una serie de tareas según la variable que leamos por ejemplo *enter* leeria cualquier numero entero y sumaria 1 unidad a la variable global ent luego leeríamos la cadena leída que se encuentra en la variable por defecto yytext la almacenaríamos en la variable local i y después se la sumaríamos a la variable global sumaent para saber el resultado total de la suma.
El resto de lineas se tratarían de la misma manera solo hace falta leer el código de la misma manera que lo hemos hecho hasta ahora.

4.3.2. Segundo ejemplo

Si te da tiempo crear un ejemplo propio para presentar en esta practica.

4.4. Conclusión del Lenguaje

La conclusión clara que encontramos con este LEX es que con pocas lineas de código podemos generar un autómata fácilmente ya que al abrir el fichero .c que genera el el programa LEX nos damos cuenta que es unas 100 veces mas grande que el fichero lex original, lo que nos da idea de cuanto nos puede ayudar es te programa.

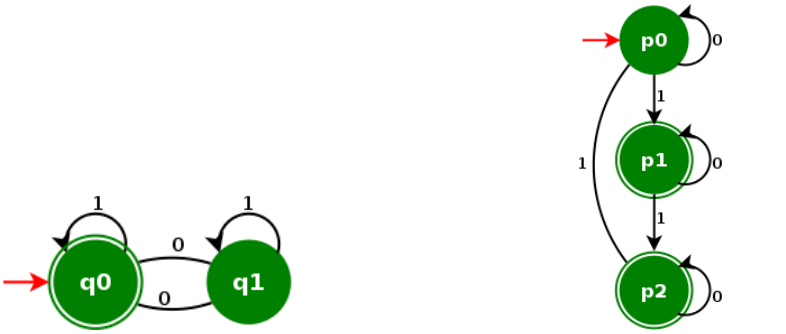
5. Practica 5 ..... Corregida

5.1. Objetivos de la Practica

En esta practica se pretende realizar la unión de varios autómatas muy concretos que se pueden encontrar en las paginas 37 a la 51 del tema 3. Lo que se pretende estudiar en esta practica es la forma en la que podemos utilizar los diferentes operadores de la aritmética de autómatas para poder reutilizar autómatas y sacar otros nuevos a partir de estos.

5.2. Ejercicio a resolver

Los autómatas con los que vamos a tener para realizar la intersección se muestran en las figuras 5.1a y 5.1b, de lo que se trata es usar ambos autómatas para realizar la operación de intersección y conseguir que teniendo dos autómatas que a priori no tendrían por que cumplir con un lenguaje dado al realizar la intersección entre ellos conseguimos que las cadenas que nosotros queremos sean aceptadas por un lenguaje dado.



(a) página 37 parte de 0 (b) página 37 parte de 1

Figura 5.1: Intersección de Autómatas

El ejercicio reza así: Construir el autómata que acepta las palabras con un número de ceros que es múltiplo de 2 y un número de unos que no sea múltiplo de 3.

5.3. Resolución del Ejercicio

Aunque primero vamos a poner el resultado final, que es el que se muestra en la figura 5.2, lo hacemos de esta manera para sobre este resultado explicar cual es la metodología que se ha aplicado para llegar al resultado y por que.

5.3.1. Primer ejemplo

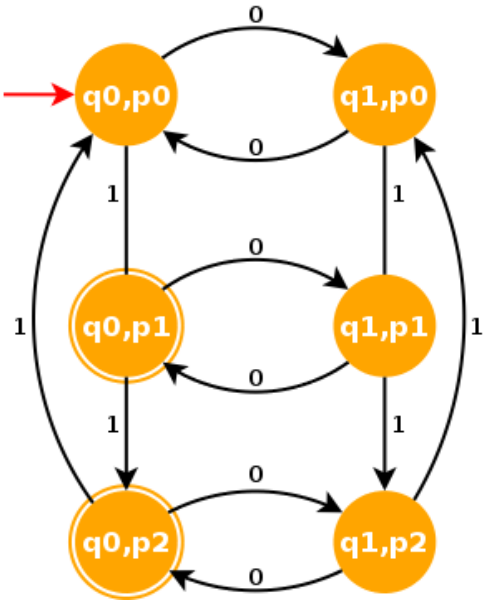
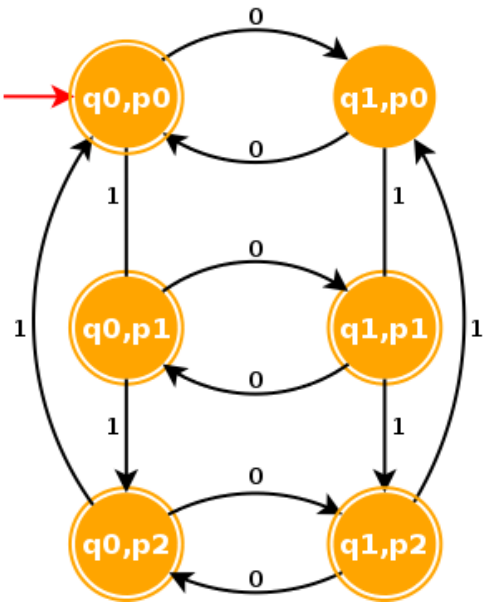


Figura 5.2

En este primer ejemplo vamos a resolver la intersección de los dos autómatas que se muestran en las figuras 5.1a y 5.1b $L1 \cap L2$, el resultado final sería el que se muestra en la figura. Para llegar ha este resultado final, lo que hemos tenido que hacer es comprobar estado ha estado cuales son los diferentes caminos que podemos seguir con los diferentes símbolos que estos autómatas son capaces de leer, así pues, el primer estado q_0, p_0 es la intersección de los estados que se indican en la figura puesto que son los estados de inicio de los 2 autómatas (L1, L2) a partir de aquí tenemos que analizar que sucede en los autómatas finitos no deterministas cuando leemos uno de los símbolos que tenemos que admitir en el AFD, osea empezamos con el 0, en el $L1$ pasamos del estado q_0 a q_1 por lo tanto en el siguiente estado de nuestro AFD al leer un 0 tenemos que pasar a un estado que sea compatible con q_1, ya tenemos la primera parte de nuestro nuevo estado para el AFD, ahora analizamos el segundo AFND $L2$, en este caso al leer un 0 pasamos de p_0 a p_0, osea nos quedamos en el mismo estado, con esto ya tenemos la segunda parte de nuestro nuevo estado, entonces, al final nuestro nuevo estado se quedaría q_1, p_0, este sería el estado que hay a la derecha del estado inicial en la figura, pero claro esta aun no hemos terminado de analizar este estado inicial pues nos queda analizar el otro símbolo que debe aceptar el AFD, osea 1, que en este caso nos genera otro nuevo estado, concretamente el que hay debajo del estado inicial como se muestra en la figura. Ya lo único que resta es analizar en el autómata $L1$ cuando se lee un 1 en el estado inicial del AFND, que parece ser que se queda en el mismo estado en donde esta, y en el segundo AFND $L2$, que se cambia al estado p_1, en esta ocasión ambos estados tanto el de $L1$ como el de $L2$, son estados finales por lo tanto el estado resultante sera estado final, al estar analizando la intercesión.
este mismo procedimiento tendríamos que realizarlo con todos los nuevos estados que se fueran generando hasta que llegara el momento que no se generan mas estados.

5.3.2. Segundo ejemplo



En este ejemplo haremos la misma descripción que en le anterior pero esta vez con la unión de los autómatas$L1 \cup L2$, así pues, la metodología a seguir es muy parecida por no decir exactamente igual a excepción de que cuando nos encontramos con los estados finales en cualquiera de los autómatas origen $L1, L2$ tenemos que poner el estado resultante como estado final también, asi que si nos fijamos en la imagen nos daremos cuenta de que tiene mas estados finales que en el anterior ejemplo, como ya hemos dicho esto es debido a la unión.

5.4. Conclusión

Bueno esta claro que la conclusión de esta practica es que, realizar un proceso de este tipo a mano si no se tiene puede ser una autentica locura, hay que hacer notar que en el proceso seguido en ambas operaciones solo teníamos un par de símbolos terminales, con esto quiero que el lector entienda que los problemas normalmente no tienen un par de variables, si no que tienen bastante mas, con el consiguiente aumento de operaciones a realizar para descubrir los diferentes estados del AFD final, y por supuesto el aumento exponencial de la cantidad de estados que tendría el AFD. Claro esta que nos quedaría realizar la minimización del autómata que nos diera como resultado, precisamente este tema es el que se intentara resolver en la siguiente practica.

6. Practica 6 ..... Corregida

6.1. Objetivos de la Practica

En esta practica se pretende aprender a realizar la minimización de autómatas tal y como se muestra en las paginas 83 hasta 117 del tema 3 de las transparencias del profesor.

6.2. Ejercicio a resolver

Lo que tenemos que realizar es intentar explicar cual es el proceso que hay que realizar para poder minimizar un autómata finito determinista a su estado minimal, el proceso se explicara siguiendo el hilo de las transparencias pero se intentara realizar con un ejemplo diferente.

6.3. Resolución del Ejercicio

Para resolver este ejercicio utilizaremos el algoritmo que se explica en la pagina 35 del tema 3 de las transparencias del profesor **Jose Antonio Garcia Soria** y que pongo a continuación para proceder al estudio del mismo y para que me sirva de guía en el análisis de la minimización del autómata.

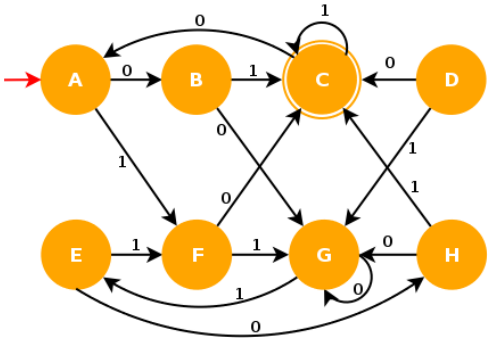
1. Eliminar estados inaccesibles.
2. Para cada pareja de estados accesibles $\{qi, qj\}$
   2.1. Si uno de ellos es final y el otro no, hacer la variable booleana asociada igual a true.
3. Para cada pareja de estados accesibles $\{qi, qj\}$
   3.1. Para cada símbolo $\boldsymbol{a}$ del alfabeto de entrada.
      3.1.1. Calcular los estados qk y ql a los que evoluciona el autómata desde qi y qj leyendo $\boldsymbol{a}$ (un símbolo).

3.1.2. Si qk = ql entonces

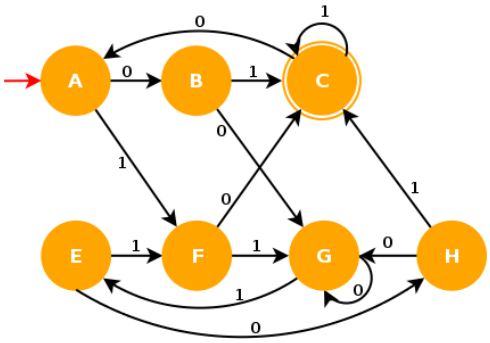
2.1. Si la $\{qk, ql\}$ está marcada entonces se marca la pareja $\{qi, qj\}$ y recursivamente todas las parejas en la lista asociada.

3.1.3. Si la pareja $\{qk, ql\}$ no está marcada, se añade la pareja $\{qi, qj\}$ a la lista asociada a la pareja $\{qk, ql\}$.

6.3.1. Primer ejemplo



Como se puede ver en la figura de la derecha es una autómata de 8 estados uno inicial **A**, otro final **C** y los estados intermedios, ahora lo que vamos a hacer es aplicar el algoritmo que hemos visto en el apartado anterior así pues el primer paso indica que tenemos que eliminar los estados inaccesibles, a primera vista parece que todos los estados son accesibles pero por eso en este paso hay que tener mucho cuidado y fijarse bien en todos los estados, una de las técnicas que yo sigo es buscar los estados a los cuales no les llega ninguna flecha de entrada (regla de producción), así pues si recorremos todos los estados nos damos cuenta que el estado **D** no recibe ninguna flecha de entrada, por lo tanto es inaccesible, una ves descubierto esto también tenemos que fijarnos en los estados a los que se llega desde este estado **D** puesto que si estos no tuvieran ningún camino mas, aparte del que llega desde **D**, estos últimos también se marcarían como inaccesibles puesto que no podríamos llegar desde ningún otro estado aparte del **D**.

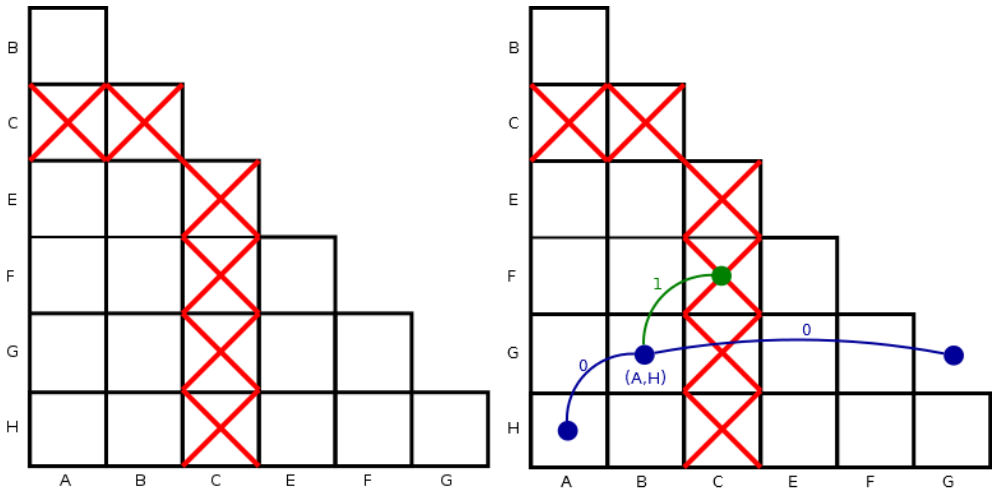


Ya tenemos eliminado los estados inaccesibles tal y como se especifica en el algoritmo, vamos con el segundo paso del algoritmo, en este caso entra en juego el estado final **C** puesto que la variable booleana se debe de poner a verdadero, cuando un estado final entra en juego, así pues todas las parejas que en las que aparezca el estado **C** se marcaran tal y como se muestra en la imagen de la tabla de minimización (figura 6.1a). Si nos fijamos detalladamente en la imagen de la tabla podemos darnos cuenta que al compararla con el autómata, lo que se hace es relacionar todos los estados finales, en este caso solo uno **C** con todos los demás estados.

Ahora lo que vamos ha hacer es seguir los pasos 3 hasta 3.1.1, estos pasos se pueden ver claramente en la figura 6.1b y también en la tabla 6.1, si nos fijamos en la linea azul de la figura podemos darnos cuenta que llegamos al final del camino en el estado ***G*** puesto que si siguiéramos aplicando el símbolo 0 nos quedaríamos en el mismo estado, y como no esta marcado pues no podríamos marcar el la pareja (A,H) que es el principio del camino, pero si nos fijamos en el camino verde de la figura podemos darnos cuenta que, después de realizar el primer paso, (B,G) como se muestra en la tabla, podemos escoger el otro camino o símbolo que es el 1, y nos lleva directamente a una casilla que si esta marcada, (C,F) en este caso, marcaríamos la casilla de origen (A,H), y dejaríamos en el estado (B,G) la lista de estados a marcar cuando este se marque.

Origen	Símbolo 0	Símbolo 0	Origen	Símbolo 0	Símbolo 1
A	B	G	A	B	C
H	G	G	H	G	F

Tabla 6.1: Tabla de estados figura 6.1b



(a) Pasos 2, 2.1 del algoritmo (b) Primera vuelta de minimización

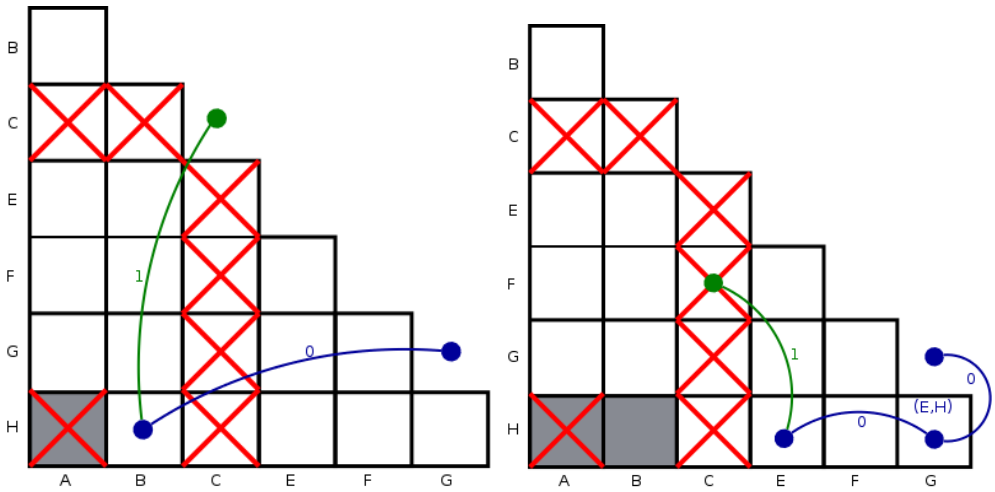
Figura 6.1: Pasos a seguir en la tabla de minimización

Después del paso anterior seguimos con el siguiente estado que en este caso seria (B,H) y ponemos el color de fondo gris en la pareja (A,H) para indicar que ya esta visitada y la cruz roja para indicar que la variable booleana la ponemos a true, ahora escogemos el símbolo 0 (camino azul) como se muestra en la figura 6.2a, y que son los pasos que se indican en la primera parte de la tabla 6.2, nos damos cuenta de que llega a una posición imposible por que no hay ninguna posición que sea la pareja (G,G), por lo tanto por este camino no podemos marcar la variable, ahora escogemos el símbolo 1 (camino verde), y nos volvemos a dar cuenta de que llega a una posición que no se contempla (C,C) por lo tanto por este camino tampoco se puede marcar la variable, entonces en este paso lo único que hacemos es marcar la posición como visitada poniendo el fondo en gris.

Origen	Símbolo 0	Símbolo 0	Origen	Símbolo 1
B	G		B	C
H	G		H	C
E	G	G	B	C
H	H	G	H	F

Tabla 6.2: Tabla de estados figura 6.2b

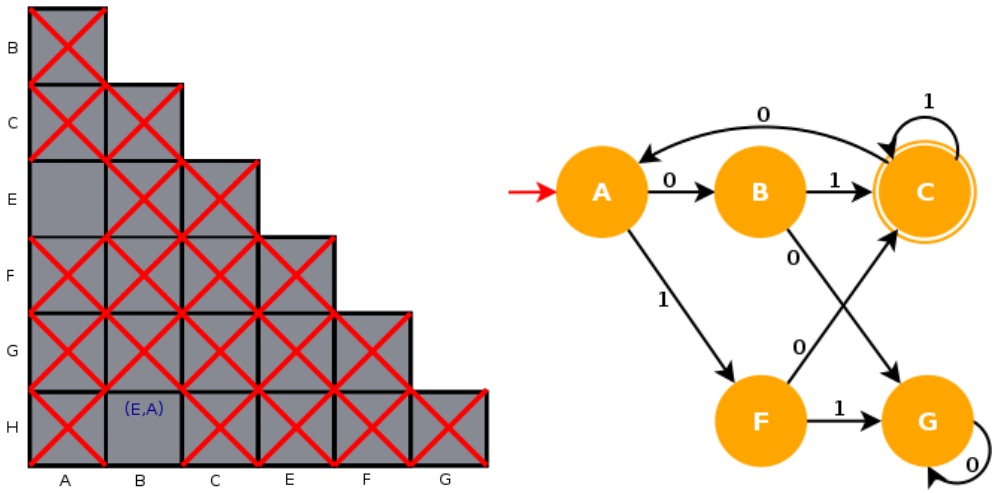
Pasamos al siguiente grupo para analizar, en este caso (E,H), y tal como muestra el camino azul de la figura 6.2b, este acaba en una posición imposible en la tabla de minimización (G,G), por lo tanto al igual que en el paso anterior por este camino no podemos marcar la variable, ahora probamos con el otro símbolo 1, marcado con el camino verde, y en este caso la posición (C,F) que si esta marcada, por lo tanto podemos marcar el grupo que estamos analizando (E,H).
Lo único que nos quedaría es seguir con la misma metodología que hasta ahora, marcamos con la cruz roja el conjunto (E,H) y también le ponemos el fondo gris en señal de que ya ha sido visitada, y seguimos con el siguiente conjunto (G,H).
Para que no se haga demasiado pesada la explicación, en la figura 6.3a, se muestra el resultado final de como quedaría la tabla de minimización.



(a) Segunda vuelta de minimización (b) Tercera vuelta de minimización

Figura 6.2: Pasos a seguir en la tabla de minimización

Como se puede apreciar en esta ultima figura 6.3a podemos ver que el algoritmo ha pasado por todas las casillas puesto que todas ellas tienen el color de fondo gris, y también podemos ver cuales has sido marcadas como validas por la cruz roja, por lo tanto lo que nos dice el algoritmo es que las casillas que no tienen la cruz son indistinguibles, y por lo tanto podemos quitar uno de los dos estados, osea que como tenemos dos parejas, (A,E) y (B,H), quitaríamos los estados E y H y al final nos quedaría un autómata finito determinista sin transiciones nulas y minimizado como el que se muestra en la figura 6.3b, hay que tener claro que al quitar los estados tenemos que colocar correctamente las transiciones existentes en estos estado, esto nos sirve de comprobación puesto que si al realizar un seguimiento de las transiciones sale alguna mas de las que hay es que nos hemos equivocado puesto que al quitar estados indistinguibles las transiciones tienen que ser exactamente iguales.



(a) Ultimo estado de la tabla de minimización (b) Versión minimizada del Autómata

Figura 6.3: Tabla y autómata final

6.3.2. Segundo ejemplo

Si tienes tiempo intenta realizarlo con un autómata que tu hayas creado

6.4. Conclusión

Este sería siempre el ultimo paso cuando construimos autómatas finitos deterministas, y tiene toda la lógica de hacerlo por que se supone que siempre que construimos un autómata queremos ahorrar el máximo espacio posible para poder, también ahorrar tanto recursos como dinero, por lo tanto lo mas lógico

es siempre realizar este paso en ultima instancia.

Referencias

[1] Wikipedia. Autómata finito determinista, Consultado el 7 de febrero de 2015. http://es.wikipedia.org/wiki/Aut%C3%B3mata_finito.

[2] Wikipedia. Autómata finito no determinista, Consultado el 7 de febrero de 2015. http://es.wikipedia.org/wiki/Aut%C3%B3mata_finito_no_determinista.

[3] Wikipedia. Jerarquía de chomsky, Consultado el 7 de febrero de 2015. http://es.wikipedia.org/wiki/Jerarqu%C3%ADa_de_Chomsky.

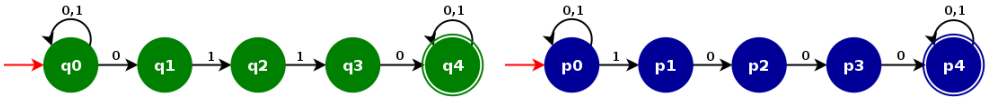
Entrega para el Examen

Objetivos del ejercicio

En este ejercicio se pretende realizar una unión de varios autómatas finitos no deterministas, de tal forma que partiendo de los autómatas L1 y L2 que se muestran en la figura 6.4a y 6.4b tenemos que conseguir realizar $L1L2 \cup L2L1, L1 \cup L2, L1*$, estos ejercicios se explicaron en clase del día 07/11/14.

Ejercicio a resolver

Lo que se pretende es resolver los tres ejercicios que se han puesto en la pizarra, para tener claro lo que hay que entregar nos ayudaremos de las siguientes figuras para saber cuales son los autómatas de inicio.



(a) Autómata L1 (b) Autómata L2

Figura 6.4: Ejercicio explicado en clase

Los lenguajes que aceptan estos autómatas son los siguientes:

$$L_1(G) = \{u\ 0110\ v \mid u,\ v \in \{0,1\}\}$$
$$L_2(G) = \{u\ 1000\ v \mid u,\ v \in \{0,1\}\}$$

Resolución del Ejercicio, Lenguaje

Básicamente lo que se trata es de realizar las operaciones aritméticas que se muestran en lo siguientes ejercicios pero siempre teniendo en cuenta las restricciones de los autómatas que se dan como origen.

Primer ejemplo

En este primer ejercicio se trata de realizar la unión de los autómatas $L_1 y L_2$ pero de una manera un tanto especial $L1L2 \cup L2L1$, entonces para saber exactamente que tipo de lenguaje tiene que reconocer el autómata resultante vamos a sacar el lenguaje usando los lenguajes que se han mostrado al principio del ejercicio:

$$L_1L_2 \cup L_2L_1(G) = \{u\ 0110\ v\ 1000\ w \mid u,\ v,\ w \in \{0,1\}\}$$

Una vez que sabemos que es lo que queremos reconocer con nuestro autómata podemos empezar con la construcción, para ello vamos a juntar los dos autómatas $L_1 y L_2$ uno detrás del otro y quitaremos el estado final del L_1 y también quitaremos las reglas de producción 0,1 del estado p_0 del autómata L_2, con estos pasos conseguiremos un autómata igual que el que se muestra en la parte superior de la figura 6.5, con este paso tendríamos la mitad del autómata que necesitamos como resultado, para conseguir la otra parte de la unión (∪) , necesitamos hacer el mismo paso pero esta vez invirtiendo los autómatas, el resultado seria el que se muestra en la parte de abajo de la 6.5, bien, hasta el momento tenemos esto $L_1L_2 y L_2L_1$, ahora solo resta realizar la unión de ambos autómatas y eso lo hacemos incluyendo un nuevo estado y dos transiciones nulas mas aparte de las que ya hemos incluido con anterioridad para *juntar* los dos autómatas.
Con este ultimo paso damos por finalizado la construcción del primer autómata.

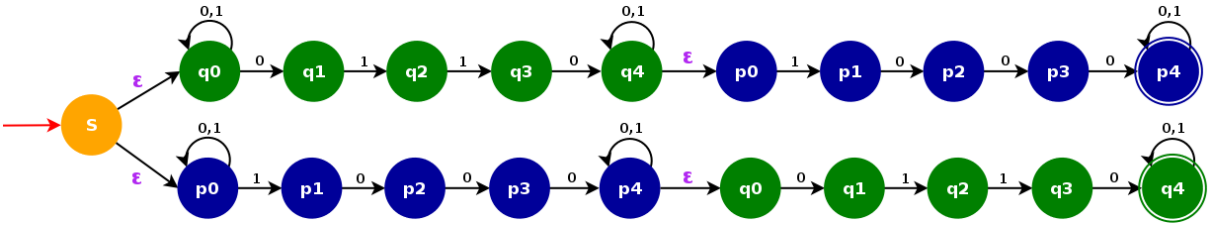
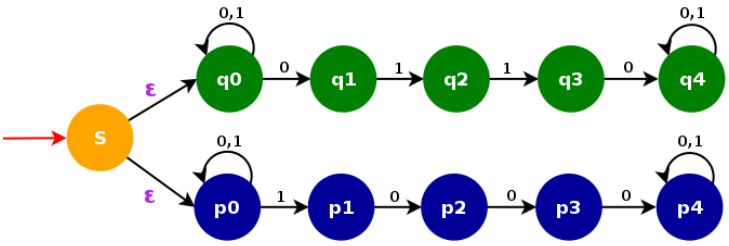


Figura 6.5: Ejercicio explicado en clase $L1L2 \cup L2L1$

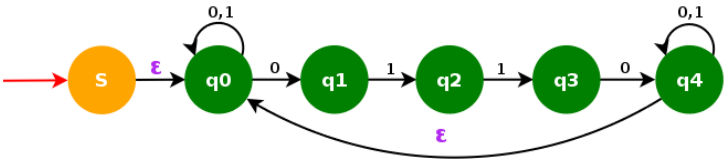
Segundo ejemplo



Este segundo ejemplo es muy parecido al anterior, lo único que cambia es que no tenemos que *juntar* previamente los dos autómatas para posteriormente realizar la unión, sino que en este caso realizamos la unión $L1 \cup L2$ directamente, tal y como ya hacíamos en el anterior ejemplo, en este caso también quitaremos los estados finales de ambos autómatas, lo cual plantea la siguiente pregunta: *¿ Cuando para de leer el Autómata ?*, pues la respuesta es obvia, cuando no queden mas símbolos terminales que leer en la cinta de lectura.
Como lo que tenemos que realizar sería la unión de ambos autómatas simplemente hacemos lo mismo que en el ultimo paso del ejemplo anterior, añadimos un estado mas, que será el inicial del autómata resultante y ponemos dos transiciones nulas hacia los autómatas $L_1 y L_2$.
Por ultimo mostraremos cual seria el lenguaje que reconocería este autómata:

$$L_1 \cup L_2(G) = \{u\ 0110\ v, u\ 1000\ v \mid u,\ v \in \{0,1\}\}$$

Tercer ejemplo



Por ultimo este autómata acepta las cadenas que contengan 0110 cualquier cantidad de veces ($L1^*$), en este caso tambien quitaremos el estado final puesto que, como nos sabemos cuantas veces puede estar contenida la cadena 0110 dentro de lo que leemos tenemos que llegar hasta el final de la cinta de lectura, para dar por valida la cadena.
El lenguaje que aceptaría este autómata es:

$$L^*(G) = \{(u\ 0110\ v) \times n \mid u,\ v, \in \{0,1\}\}$$

Y para construirlo necesitamos poner dos transiciones nulas, la primera será la que hay desde el nuevo estado de partida hasta el primer estado del autómata, esto se hace así para poder poner la segunda transición nula que es la que nos permite repetir tantas veces como sea necesario la cadena aceptada, ya que al poner esta transición desde el ultimo estado del autómata hasta el ahora segundo estado nos permite empezar a reconocer una nueva cadena con los valores 0110. De esta forma tenemos la posibilidad de retroceder en los estados del autómata.