Modelos de computación (2015-2016)
Grado en Ingeniería Informática
Universidad de Granada

Memoria Prácticas

Francisco Javier Navarro Morales

15 de noviembre de 2015

Índice

1. Practica 1: G = (V,T,P,S), donde V = S,A,B, T = a,b, el símbolo de partida es S y las reglas son:

$$\begin{array}{cccc} S \to aB, & S \to bA, & A \to a, & A \to aS, \\ A \to bAA, & B \to b, & B \to bS, & B \to aBB \end{array}$$

Esta gramática genera el lenguaje: $L(G) = \{u | u \, \epsilon \, \{a, b\}^+ \; y \; Na(u) = Nb(u)\}$, es decir, la gramática genera palabras con el mismo número de 'a' que de 'b'.

Podemos extraer las siguientes interpretaciones de las reglas de producción:

- Interpretación de **'A'** → Genera palabras con un Símbolo Terminal 'a' de más.
- Interpretación de **'B'** → Genera palabras con un Símbolo Terminal 'b' de más.
- Interpretación de **'S'** → Genera una cadena con el mismo numero de 'a' que 'b'.

Hay que demostrar dos cosas:

- Todas las palabras generadas por la gramática tienen el mismo número de a que de b.
- Cualquier palabra con el mismo número de a que de b es generada.

Vamos a ir desarrollando cada posibilidad de forma que para cada paso se apliquen todas las reglas de producción posibles, inicialmente tenemos dos posibilidades:

$$^1S \Rightarrow aB, \qquad ^2S \Rightarrow bA$$

Vamos a iniciar el desarrollo para la primera:

S ⇒ aB ⇒ ab, generamos la palabra **ab**.
S ⇒ aB ⇒ abS ⇒ abaB ⇒ abab, generamos la palabra **abab**.
S ⇒ aB ⇒ aaBB ⇒ aabB ⇒ aabb, generamos la palabra **aabb**.

Continuamos el desarrollo para la segunda posibilidad:

S ⇒ bA ⇒ ba, generamos la palabra **ba**.
S ⇒ bA ⇒ baS ⇒ babA ⇒ baba, generamos la palabra **baba**.
S ⇒ bA ⇒ bbAA ⇒ bbaA ⇒ bbaa, generamos la palabra **bbaa**.

Hemos comprobado que podemos conseguir generar cadenas básicas con $N_a(\mathrm{u}) = N_b(u)$, dónde primero hay símbolos terminales 'a' y luego 'b' ó hay símbolos terminales '$(ab)^+$', y lo mismo cambiando el orden de 'a' y 'b'. Si queremos generar cualquier cadena perteneciente al lenguaje lo podemos conseguir combinando las anteriores palabras generadas. Por ejemplo generemos una palabra usando todas las reglas de producción:

$$S \overset{1}{\Rightarrow} a\underline{B} \overset{8}{\Rightarrow} aa\underline{B}B \overset{7}{\Rightarrow} aab\underline{S}B \overset{1}{\Rightarrow} aaba\underline{B}B \overset{8}{\Rightarrow} aabaa\underline{B}BB \overset{7}{\Rightarrow} aabaab\underline{S}BB \overset{2}{\Rightarrow} aabaabb\underline{A}BB$$
$$\overset{5}{\Rightarrow} aabaabbb\underline{A}ABB \overset{3}{\Rightarrow} aabaabbba\underline{A}BB \overset{4}{\Rightarrow} aabaabbbaa\underline{S}BB \overset{2}{\Rightarrow} aabaabbbaab\underline{A}BB \overset{3}{\Rightarrow}$$
$$aabaabbbaaba\underline{B}B \overset{6}{\Rightarrow} aabaabbbaabab\underline{B} \overset{6}{\Rightarrow} aabaabbbaabab$$

Podemos apreciar que encima de cada fecha indicamos el número de la regla utilizada (las identificamos contando de izquierda a derecha y de arriba hacia abajo) en este ejemplo hemos usado las 8 reglas de producción que nos brinda la gramática, comprobando que la palabra que genera **'aabaabbbaababb'** contiene el mismo número de símbolos terminales 'a' que 'b', en concreto N = 7. Con este ejemplo demostramos también que combinando las reglas de producción podemos generar cualquier palabra con la peculiaridad de que el número de 'a' coincide con el de 'b'.

2. Practica 2: Determinar si la gramática G = ({S, A, B}, {a, b, c, d}, P, S) genera un lenguaje de tipo 3 donde P es el conjunto de reglas de producción:

$$S \to AB, \qquad A \to Ab, \qquad A \to a, \qquad B \to cB, \qquad B \to d$$

Esta **gramática genera el lenguaje** $L(G) = \{ab^i c^j d : i, j \ \epsilon \ \mathbb{N}\}$

Partiendo de los datos del problema vamos a comprobar que esa gramática genera todas las palabras del lenguaje.

$S \to \underline{A}B \to \underline{A}bB \to \underline{A}bbB \to ab^i\underline{B} \to ab^ic\underline{B} \to ab^icc\underline{B} \to ab^ic^j\underline{B} \to ab^ic^jd$

Vemos que todas las palabras que genera dicha gramática pertenecen al lenguaje y no genera ninguna que no pertenezca al lenguaje. Podemos sacar la siguiente interpretación:

- A ->genera subcadenas que empiezan por 'a' seguido de un numero i de 'b'.
- B ->genera subcadenas que empiezan por un numero j de 'c' y acaban en 'd'.
- Estamos ante una gramática libre del contexto(de tipo 2) puesto que sus producciones son de la forma "terminal-Variable y Variable-terminal" ó "Variable-Variable". Por lo tanto el lenguaje que genera es también de tipo 2.

Hasta ahora no hemos hecho mas que comprobar que los datos del problema son correctos, el siguiente paso es ver si conseguimos modificar la gramática, en concreto sus producciones, para conseguir que esta genere un lenguaje regular. Podemos intentar esto porque no hay relación numérica entre los símbolos terminales del lenguaje.

Según la **Jerarquía de Chomsky** un lenguaje es regular si las reglas de producción son de tipo 3, para esto las producciones de la gramática deben ser del tipo: $A \to uB$ || $A \to u$.

Vamos a usar las siguientes reglas de producción:

$$\text{S} \to \text{aB}, \qquad \text{B} \to \text{bB}, \qquad \text{B} \to \text{C}, \qquad \text{C} \to \text{cC}, \qquad \text{C} \to \text{d}$$

- S → aB: S es el símbolo inicial. Esta prod. genera la primera "a" y da paso a la variable "B".
- B → bB: Generar un numero i de "b".
- B → C: Pasa a la variable "C".

- C → cC: Genera un número j de ”c”.
- C → d: Para finalizar esta producción nos permite colocar el último símbolo terminal ”d” necesario para formar palabras correctas.

En conclusión hemos encontrado unas reglas de producción de tipo 3, que hacen la gramática regular, capaces de generar el mismo lenguaje que generaban las reglas de producción de partida. **Demostramos que la nueva gramática es de tipo 3 por lo tanto el lenguaje también lo es.**

3. Practica 3: Diseñar un autómata finito determinístico que acepte cadenas que contienen las subcadenas '0110' y '1000'. Las subcadenas pueden aparecer juntas o separadas, también pueden contener más símbolos (0+1) delante o detras de ellas.

Para comenzar esta práctica lo más sencillo es **diseñar el autómata no determinístico** que acepta las cadenas que nos pide el ejercicio. Nos ayudamos de los ejemplos de los autómatas que son capaces de reconocer palaras que contienen las subcadenas '0110' ó '1000, de esta forma conseguimos diseñar el autómata que necesitamos de forma muy fácil, para ello **seguimos los siguientes pasos**:

- Q4 deja de ser un nodo final y P0 deja de ser un nodo inicial.
- Unimos los dos autómatas añadiendo una transición nula entre Q4 y P0.

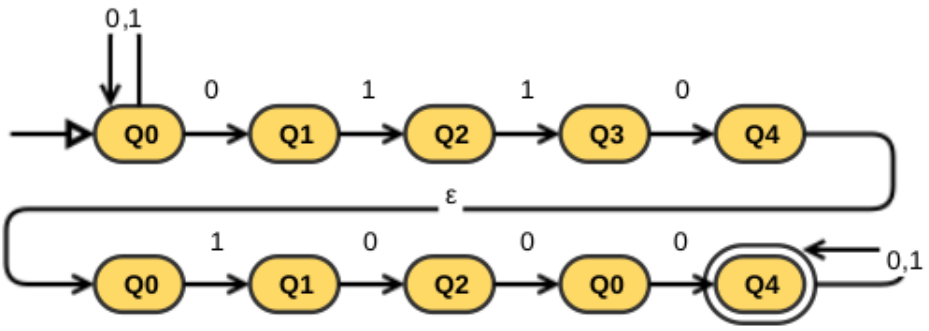

Figura 3.1: NonDeterministic finite automaton(NFA)

Los autómatas no determinísticos son muy ineficientes porque exploran todas las posibilidades. Para resolver el problema que tenemos entre las manos de una forma más eficiente vamos a tranformarlo en un autómata que para cada símbolo leido sepa que acción debe realizar. Es un proceso exponencial a medida en que crece el número de nodos en el 'NFA', para no equivocarnos vamos a realizar una tabla donde indiquemos para cada nodo cual es la acción que realiza al leer cierto símbolo de la cinta de entrada.

Símbolo	**Q0**	**Q1**	**Q2**	**Q3**	**Q4**
0	{Q0,Q1}	{Ø}	{Ø}	{Q4,P0}	{Q4,P0}
1	{Q0}	{Q2}	{Q3}	{Ø}	{Q4,P0,P1}

Tabla 3.1: Tabla nodos Q.

Símbolo	P0	P1	P2	P3	P4
0	{P0}	{P2}	{P3}	{P4}	{P4}
1	{P0,P1}	{Ø}	{Ø}	{Ø}	{P4}

Tabla 3.2: Tabla nodos P.

Para construir nuestro autómata determinístico nos apoyaremos de las tablas que hemos creado. **Comprobamos para cada estado** de los que tenemos entre **{<estados>}** a que **estado(s) pasa** según lea un 0/1, si tenemos un **estado compuesto** por 2 o más estados tenemos que realizar el proceso para cada subestado y el **resultado será la unión** de los resultados de cada subestado. Una vez generado el nuevo estado tenemos **dos posibilidades** que el **nuevo estado se repita**, en este caso trazamos una flecha desde el estado actual hasta el estado obtenido(etiquetando dicha flecha con el símbolo que produce el cambio de estados); por otra parte, si **es un nuevo estado** debemos colocarlo unidos por una flecha etiquetada. **Repetimos** todo el proceso hasta que todos los estados tengan un estado objetivo para cada símbolo posible(en nuestro caso 0/1).

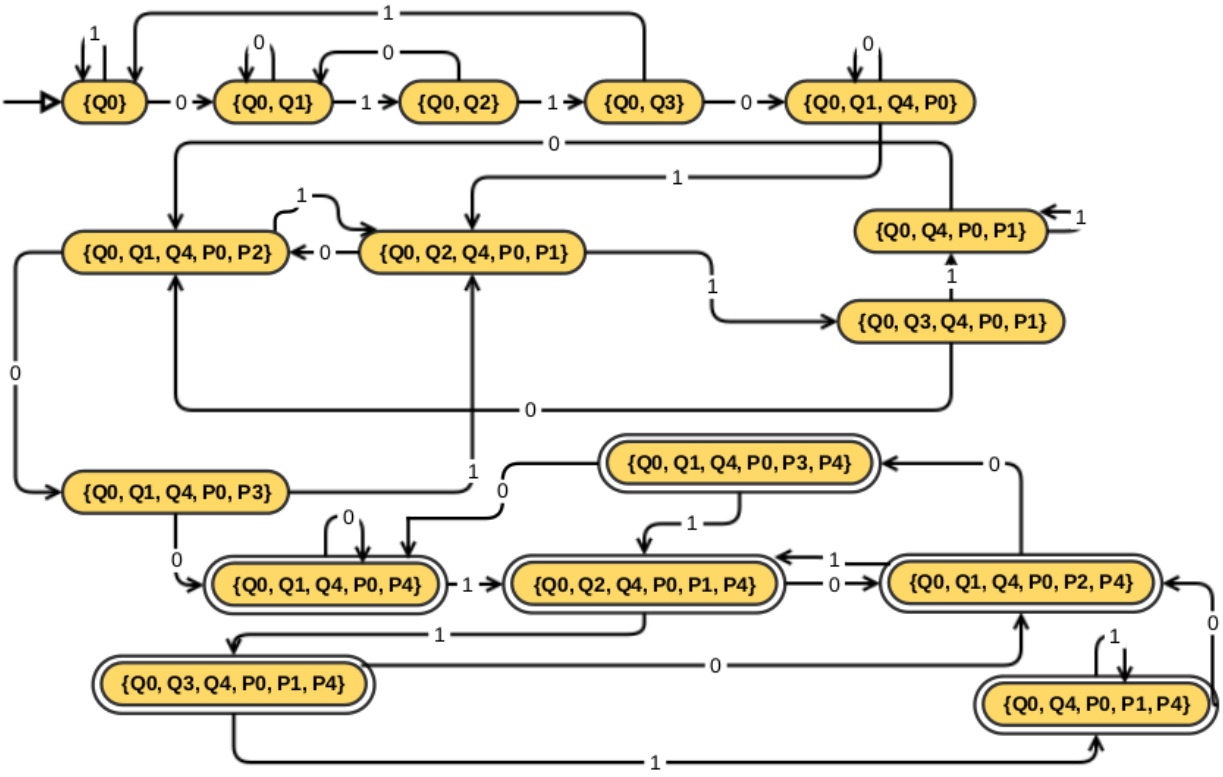

Figura 3.2: Deterministic finite automaton(DFA)

4. Practica 4: Realizar programas usando la herramienta de reconocimiento de expresiones regulares LEX

Este programa de ejemplo es capaz de reconocer números(tanto reales como enteros) e identificadores formados por al menos un carácter seguido de más caracteres o dígitos. En el caso de los enteros también lleva la suma de estos que al final muestra junto al numero de reales, enteros e identificadores reconocidos.

Escribe el siguiente código en un fichero llamado `ejemplo.c`:

```
/*Definir Variables Globales*/
%{
int ent=0, real=0, ident=0, sumaent=0, i=0;
%}

/*Definir Expresiones Regulares*/
car     [a-zA-Z]
digito  [0-9]
signo   (\-|\+)
suc     ({digito}+)
enter   ({signo}?{suc})
real1   ({enter}\.{digito}*)
real2   ({signo}?\.{suc})

/*Definir Acciones*/
%%
{enter}                 {ent++; sscanf(yytext,"%d",&i); sumaent +=
    i; printf("Numero entero: %s\n\n",yytext);}

({real1}|{real2})       {real++; printf("Num. real: %s\n\n",yytext
   );}

{car}({car}|{digito})*  {ident++; printf("Var. ident: %s\n\n",
   yytext);}

.|\n                    {;}

%%
yywrap()
        {printf("Numero de Enteros: %d, reales: %d, ident: %d,
        Suma de Enteros: %d",ent,real,ident,sumaent); return 1;}
```

Ahora compila usando `lex` y `gcc`:

```
$ lex ejemplo
$ gcc lex.yy.c -o prog -ll
```

Por último compilamos de la siguiente forma:

```
$ ./prog < 'fichero_entrada' > 'fichero_salida'
```