Modelos de Computación

grado en ingeniería informática

Prácticas resueltas

Autor
Carlos Sánchez Páez

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2019-2020

Índice

1. Práctica 1

Construir un autómata finito determinístico para aceptar cadenas de ceros y unos que tengan un número de ceros que no sea múltiplo de 3. Usar JFLAP para simular el autómata.

Solución

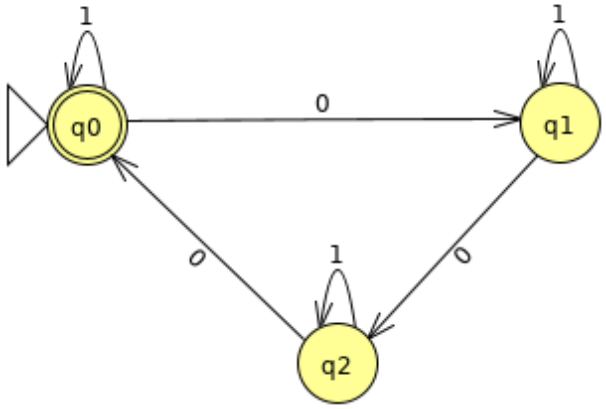
Para construir el siguiente autómata emplearemos la metodología inversa: elaboraremos uno que acepte un número de ceros múltiplo de 3 y cambiaremos sus estados finales por normales y viceversa, ya que así es más sencillo.
Necesitaremos definir tres estados:

- q_0: hasta ahora el autómata ha leído un número de ceros múltiplo de 3. Es estado final y también inicial, ya que ϵ, la cadena vacía, también debe ser aceptada (0 es múltiplo de 3).
- q_1: hemos leído un cero extra.
- q_2: hemos leído dos ceros extra.

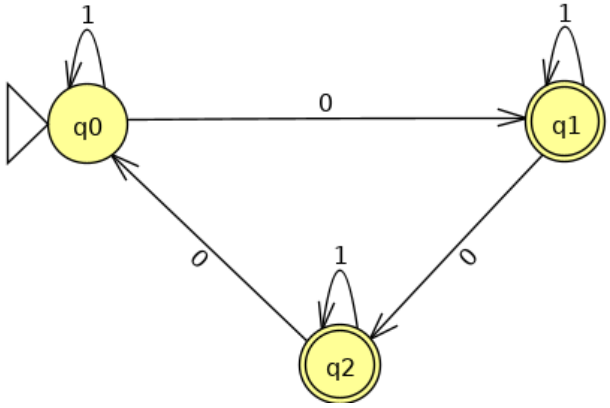
Las transiciones serán las siguientes:

- Si leemos un uno, nos quedamos en el mismo estado: $q_0 \overset{1}{\Rightarrow} q_0$, $q_1 \overset{1}{\Rightarrow} q_1$ y $q_2 \overset{1}{\Rightarrow} q_2$.
- Si leemos un cero, avanzamos al siguiente estado: $q_0 \overset{0}{\Rightarrow} q_1$, $q_1 \overset{0}{\Rightarrow} q_2$ y $q_2 \overset{0}{\Rightarrow} q_0$.

Por tanto, nuestro autómata sería el siguiente:



Ahora cambiamos los estados normales por finales y viceversa, obteniendo la solución al problema:



2. Práctica 2

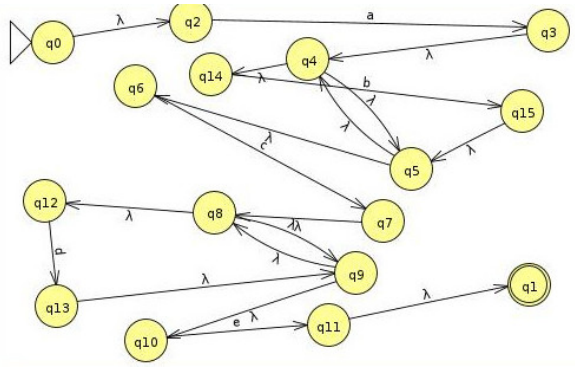
Plantear un lenguaje regular con un número infinito de cadenas. Obtener la expresión regular del lenguaje y convertirlo a AFD minimal y AFND con transiciones nulas mediante JFLAP.

Solución

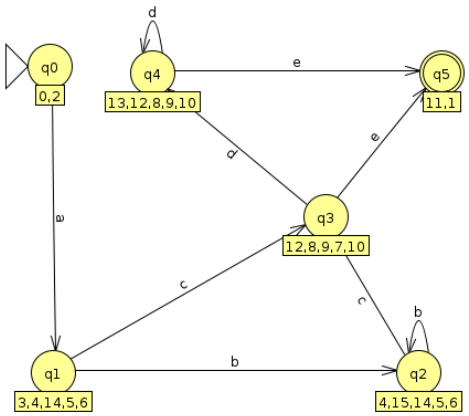
Tomemos el siguiente lenguaje:

$$L = \{ab^i cd^j e : i, j \in \mathbb{N}\}$$

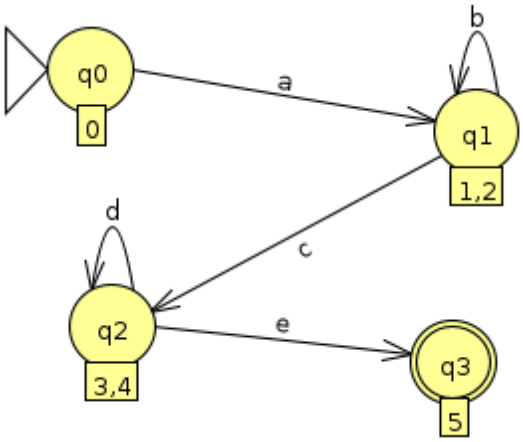
La expresión regular resultante es ab^*cd^*e. El AFND con transiciones nulas obtenido es el siguiente:



Lo convertimos a AFD y obtenemos:



Finalmente, lo minimizamos:


q0
0
a
q1
1,2
b
c
d
q2
3,4
e
q3
5

3. Práctica 3

Crear un fichero LEX, ejecutar el programa sobre él, compilar la salida y ejecutarla sobre un conjunto de datos de entrada.

Solución

Mi fichero LEX es el siguiente:

```
car [a-zA-Z]
digito [0-9]
signo (\-|\+)
suc ({digito}+)
enter ({signo}?{suc})
real1 ({enter}\.{digito}*)
real2 ({signo}?\.{suc})
dni ({digito}{8}\-{car}{1})
nie ({car}{1}\-{digito}{7}\-{car}{1})
matricula ({digito}{4}\-{car}{3})
matricula_old ({car}{2}\-{digito}{4}\-{car}{2})
matricula_ciclomotor ({car}{1}{digito}{1}\-{digito}{3}\-{car}{3})
ss ({digito}{2}[\-|\/]{digito}{10})
receta ("papicas"|"huevicos"|"cebollica")
  int ent=0,real=0,sumaent=0,numdni=0,numnie=0,nummat=0,
  nummatold=0,nummatc=0,numss=0,numings=0;
%%
  int i;
{enter} {ent++;sscanf(yytext,"%d",&i);sumaent += i; printf("Entero -> %s\
    ↪ n",yytext);}
({real1}|{real2}) {real++;printf("Real -> %s\n",yytext);}
{dni} {numdni++;printf("DNI -> %s\n",yytext);}
{nie} {numnie++;printf("NIE -> %s\n",yytext);}
{matricula} {nummat++;printf("Matricula -> %s\n",yytext);}
{matricula_old} {nummatold++;printf("Matricula provincial -> %s\n",yytext
    ↪ );}
{matricula_ciclomotor} {nummatc++;printf("Matricula de ciclomotor -> %s\n
    ↪ ",yytext);}
{ss} {numss++;printf("Numero de Seguridad Social -> %s\n",yytext);}
{receta} {numings++;printf("Ingrediente para la tortilla de patatas -> %s
    ↪ .\n",yytext);}
.|\n {;}
%%
yywrap()
  {printf("*****************************************\n \
          Numeros enteros -> %d\n \
          Numeros reales -> %d\n \
          Suma de enteros -> %d\n \
          DNI -> %d\n \
          NIE -> %d\n \
          Matriculas -> %d\n \
          Matriculas provinciales -> %d\n \
```

```
        Matriculas de ciclomotor -> %d\n \
        Numeros de Seguridad Social -> %d\n \
        Ingredientes para la tortilla de patatas -> %d", \
        ent,real,sumaent,numdni,numnie,nummat,nummatold,nummatc,numss,
            ↪ numings);\
        return 1;}
```

Básicamente clasifica las entradas en varias categorías:

- Números enteros.
- Números reales.
- DNI.
- NIE.
- Matrículas.
- Matrículas provinciales.
- Matrículas de ciclomotor.
- Números de Seguridad Social.
- Ingredientes para una tortilla de patatas.

Con el siguiente fichero de entrada:

```
25478521-C
20
cebollica
X-2551923-G
2578-BJP
MA-4857-BN
huevicos
C6-852-AFR
18/8520147853
58-2369854785
2874-HDV
papicas
23874587-H
W-8756523-L
53349482-Q
25874102-W
2.6363665
-2.2
-9994758585985.2
231231323
5874-LRW
GR-2331-KW
```

se obtiene el siguiente resultado:

```
DNI -> 25478521-C
Entero -> 20
Ingrediente para la tortilla de patatas -> cebollica.
NIE -> X-2551923-G
Matricula -> 2578-BJP
Matricula provincial -> MA-4857-BN
Ingrediente para la tortilla de patatas -> huevicos.
Matricula de ciclomotor -> C6-852-AFR
Numero de Seguridad Social -> 18/8520147853
Numero de Seguridad Social -> 58-2369854785
Matricula -> 2874-HDV
Ingrediente para la tortilla de patatas -> papicas.
DNI -> 23874587-H
NIE -> W-8756523-L
DNI -> 53349482-Q
DNI -> 25874102-W
Real -> 2.6363665
Real -> -2.2
Real -> -9994758585985.2
Entero -> 231231323
Matricula -> 5874-LRW
Matricula provincial -> GR-2331-KW
****************************************
          Numeros enteros -> 2
          Numeros reales -> 3
          Suma de enteros -> 231231343
          DNI -> 4
          NIE -> 2
          Matriculas -> 3
          Matriculas provinciales -> 2
          Matriculas de ciclomotor -> 1
          Numeros de Seguridad Social -> 2
          Ingredientes para la tortilla de patatas -> 3
```

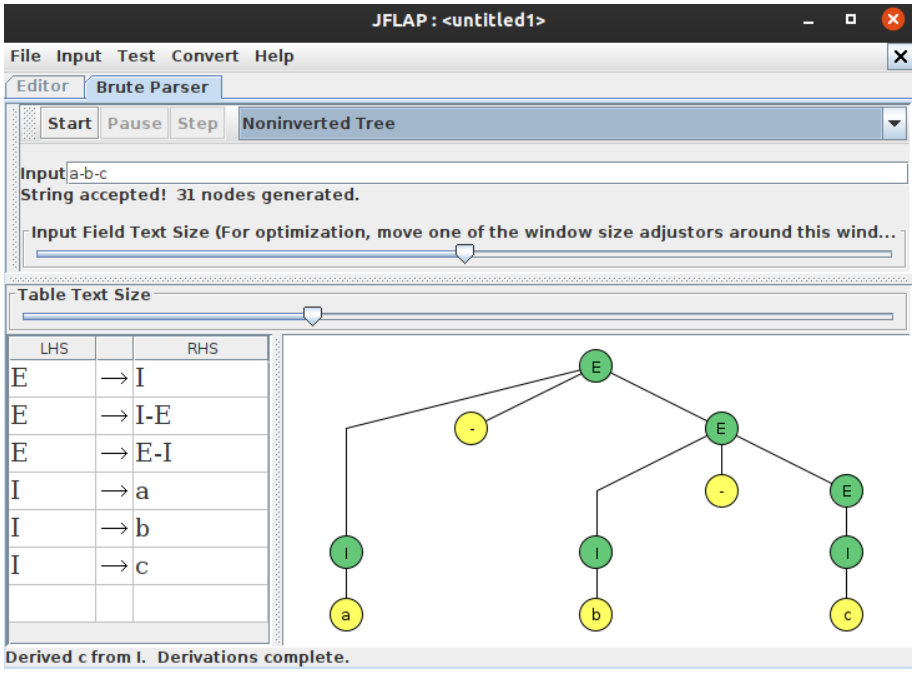
4. Práctica 4

Demostrar mediante JFLAP que la siguiente gramática es ambigua:

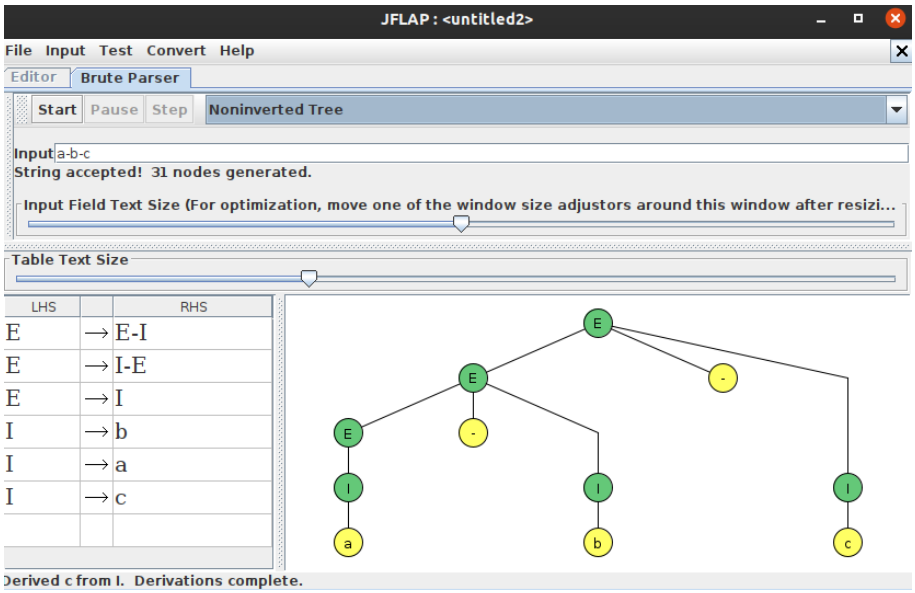
$$\begin{aligned} E &\Longrightarrow I \\ E &\Longrightarrow I - E \\ E &\Longrightarrow E - I \\ I &\Longrightarrow a|b|c|d \end{aligned}$$

Solución
Tomaremos la palabra $a - b - c$ para demostrar la ambigüedad:

1. Comenzamos introduciendo la gramática en JFLAP-Grammar.
2. Accedemos a la opción *Brute Force Parse*.
3. Introducimos nuestra palabra y obtenemos su árbol de derivación:



4. Ahora abrimos una nueva ventana e introducimos la gramática pero cambiando el orden de las reglas.
5. Generamos el árbol de derivación y obtenemos lo siguiente:



6. Vemos que los árboles son distintos para una misma palabra, por lo que concluimos que la gramática es ambigua.