Autómatas y Lenguajes Formales 2017-1
Facultad de Ciencias UNAM
Nota de Clase 8

Favio E. Miranda Perea   A. Liliana Reyes Cabello   Lourdes González Huesca

26 de septiembre de 2016

1. Gramáticas Regulares

Recordemos que una gramática regular es una gramática lineal por la derecha o lineal por la izquierda y que no se permite mezclar ambos tipos de producciones:

$$\begin{array}{ll} \text{derecha} & A \to aB \quad A \to a \quad A \to \varepsilon \\ \text{izquierda} & A \to Ba \quad A \to a \quad A \to \varepsilon \end{array}$$

con $A, B \in V,\ a \in T$, obsérvese que los símbolos no-terminales están a la derecha o a la izquierda respectivamente.

Definición 1 *Decimos que un lenguaje L es regular si existe una gramática regular G que lo genere, es decir, si $L = L(G)$.*

También recordemos que un lenguaje tiene un tipo i si L es generado por una gramática de ese tipo el cual debe asegurarse que i es máximo.

Ejemplo: Sea el lenguaje $L = 0^\star 1 0^\star 1 0^\star$ y es generado por:

$$\begin{array}{lcl} S & \to & A1A1A \\ A & \to & 0A \mid \varepsilon \end{array}$$

Esta gramática no es regular, pero el lenguaje si lo es al existir una gramática regular equivalente:

$$\begin{array}{lcl} S & \to & 0S \mid 1A \\ A & \to & 0A \mid 1B \\ B & \to & 0B \mid \varepsilon \end{array}$$

Ejemplo: El lenguaje $L = (a+b)^\star b$ es generado por la siguiente gramática:

$$\begin{array}{lcl} S & \to & aS \mid bC \\ C & \to & bC \mid aS \mid \varepsilon \end{array}$$

2. Lenguajes y gramáticas regulares

Hay una correspondencia entre los lenguajes regulares y las gramáticas regulares, veamos las dos partes de esta equivalencia que involucran a los autómatas finitos:

1. Autómatas Finitos $\Rightarrow$ Gramáticas regulares
   Dado una máquina finita $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ existe una gramática regular $G = \langle V, T, S, P\rangle$ tal que $L(M) = L(G)$. Es decir, todo lenguaje regular es generado por una gramática regular.

   La construcción de una gramática regular G dado un autómata M se obtiene como sigue:

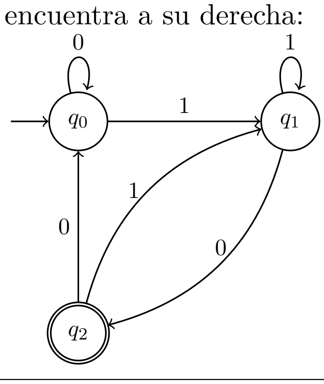
   - Suponemos sin pérdida de generalidad que no hay $\boldsymbol{\varepsilon}$-transiciones en M, es decir puede ser que M sea no-determinista.
   - Se definen las partes de G, $V = Q \qquad T = \Sigma \qquad S = q_0$.
     Es decir que los estados serán los símbolos no-terminales y los terminales serán exactamente los símbolos del alfabeto.
   - Las reglas de producción P se definen como sigue:
     - Si $p \in \delta(q, a)$ entonces agregamos $q \to ap$ a P.
     - Además, si $q_f \in \delta(q, a)$ con $q_f \in F$ entonces agregamos $q \to a$. [1]

2. Gramática regular $\Rightarrow$ Autómata Finito
   Dada una gramática regular $G = \langle V, T, S, P\rangle$ existe un autómata finito $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ tal que $L(M) = L(G)$. Es decir todo lenguaje generado por una gramática regular es regular.

   Definimos al autómata M en base a las reglas de producción como sigue:

   - Suponemos sin pérdida de generalidad que G es lineal derecha.
   - El alfabeto es exactamente el conjunto de símbolos terminales $\Sigma = T$ y los estados serán $Q = V \cup \{q_F\} \qquad q_0 = S \qquad F = \{q_F\}$
   - La función de transición δ se define como sigue:
     - Si $A \to aB \in P$ entonces $B \in \delta(A, a)$.
     - Si $A \to a \in P$ entonces $q_F \in \delta(A, a)$.
     - Si $A \to \boldsymbol{\varepsilon} \in P$ entonces $q_F \in \delta(A, \boldsymbol{\varepsilon})$.

Ejemplo: Considere el siguiente autómata, la gramática correspondiente obtenida con el método anterior se encuentra a su derecha:



$$
\begin{aligned}
q_0 &\to 0q_0 \mid 1q_1\\
q_1 &\to 1q_1 \mid 0q_2 \mid 0\\
q_2 &\to 0q_0 \mid 1q_1
\end{aligned}
$$

[1] No hay necesidad de agregar transiciones del tipo $q \to \boldsymbol{\varepsilon}$ ya que se asegura con las opciones anteriores que los estados finales están representados por $q \to a$ si $\delta(q, a) = q_f$.

2.1. Equivalencia entre gramáticas lineales

Se puede probar que toda gramática lineal por la izquierda es equivalente a una gramática lineal por la derecha. Veamos cómo convertir gramáticas lineales por la izquierda a lineales por la derecha y viceversa, para ello utilizamos las transformaciones a autómatas finitos.

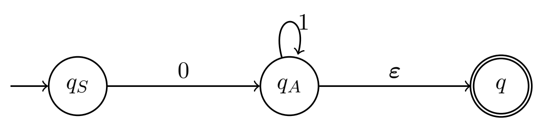
1. Dada una gramática lineal por la derecha se debe construir un autómata finito que reconozca al mismo lenguaje.
2. Para construir la gramática lineal por la izquierda se debe obtener un autómata *dual*:
   - intercambiar los estados inicial y final
   - invertir las transiciones
3. Obtener una nueva gramática del autómata nuevo y después reordenar las reglas de producción $A \to aB$ en $A \to Ba$.

El método anterior se puede usar cuando el autómata tiene un sólo estado final.
Para la transformación de lineal izquierda a lineal derecha se siguen los mismos pasos que antes.

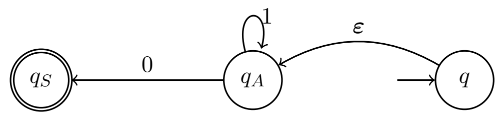
Ejemplo: Convertir la siguiente gramática lineal por la derecha en una equivalente lineal por la izquierda:

$$S \to 0A \qquad\qquad A \to 1A \mid \varepsilon$$

Se obtiene primero el autómata:



Después el autómata inverso o dual:



Las transiciones del autómata anterior son:

$$q \to \varepsilon q_A \qquad\qquad q_A \to 1q_A \mid 0$$

Y por último se invierten los símbolos y se elimina ε:

$$q \to q_A \qquad\qquad q_A \to q_A1 \mid 0$$

3. Aplicaciones prácticas de los lenguajes regulares

- Expresiones regulares para filtrar cadenas en archivos.
  Hay muchos usos de las expresiones regulares en Unix, por ejemplo en la búsqueda de patrones o las expresiones usadas por los comandos `awk` o `grep` [2]. Veamos algunos ejemplos:
  - `$ ls -ld [[:upper:]]*`
    esta línea obtiene todos los archivos que inician con mayúscula en cierto directorio.

[2] `http://www.gnu.org/software/grep/manual/grep.html#Regular-Expressions`

- `$ grep '\<c...h\>' /usr/share/dict/words`
  este comando obtiene las palabras del diccionario que empiezan en `c`, terminan en `h` y tienen exactamente 5 caracteres (cada punto indica que hay un caracter).
- `$ grep -c "go" demo_text`
  obtiene el número de líneas en el archivo `demo_text` que contienen la cadena `go`.

- Análisis de programas booleanos
  Por ejemplo programas que modelan sistemas de estados finitos como el comportamiento de un elevador o una máquina despachadora.

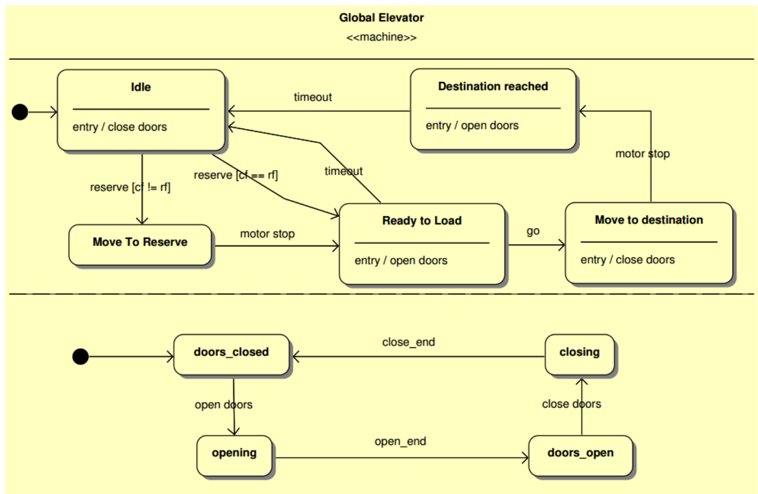


Figura 1: Eventos de un elevador y puertas

- Análisis léxico
  Parte inicial de un compilador que se encarga de separar un programa en pequeños grupos de caracteres llamados *tokens*, es decir en nombres de variables, números, *keywords*, etc. Se describe el analizador léxico a través de una o varias expresiones regulares para distinguir las clases de tokens. Por ejemplo, las siguientes expresiones identifican los espacios en blanco, nombres y números:

  $\alpha 1$ `= <hspace> + <newline>,`
  $\alpha 2$ `= [letter] ([letter] + [digit])*,`
  $\alpha 3$ `= [digit] [digit]*(% + E [digit] [digit]*),`

  Se debe procesar un archivo y el analizador verifica subcadenas del programa que son precisamente los tokens, usualmente se considera al prefijo más largo que sea un token como separador.
  La forma más sencilla de realizar un análisis léxico o *lexer* es mediante autómatas no determinísticos, debe haber un autómata por cada definición de token y el autómata principal es la unión de los anteriores.