Teoría de la Computación

Práctica No. 2: **Procesamiento regular de cadenas**

Objetivo de la práctica

Implementar una máquina de estados y una función de procesamiento regular de cadenas usando técnicas recurrentes.

Fecha límite de entrega

Miércoles 13 de Abril de 2011.

Desarrollo

El alumno deberá implementar una aplicación que tome como entrada una cadena de eventos (*caracteres*) e informe si ésta pertenece o no a un lenguaje regular. El reconocedor de dicho lenguaje deberá estar implementado como un sistema de estados y transiciones (un *autómata finito determinista*) de acuerdo a la especificación que se muestra en la figura 1.

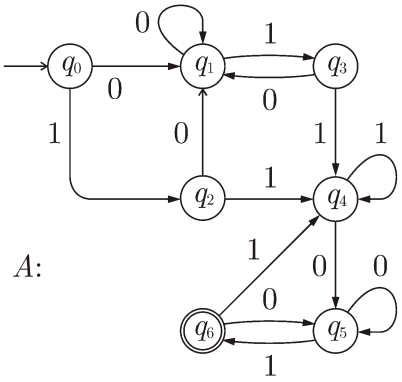

Figura 1. Especificación del lenguaje.

El programa deberá informar lo siguiente en cada paso del procesamiento:

- El estado actual del sistema.
- El próximo símbolo que se va a procesar.
- El estatus actual del procesamiento: *aceptación* ó *rechazo*.

Cuando la cadena haya sido procesada en su totalidad, deberá informar si ésta ha sido reconocida o no por el sistema. Deberán implementarse dos versiones del procesamiento de las cadenas, una versión *inductiva* (*recursiva*) y una versión *iterativa*.

Lineamientos generales para la realización de su solución

1. **Diseño de la implementación**. Deberá implementar las siguientes funciones:

   - Una función de nombre `delta` que calcule el estado sucesor del sistema. Recibe como argumentos un *estado* y un *evento* (un carácter), y retorna como resultado el estado sucesor. Esta función es la implementación de la función δ y será la misma para ambas versiones de su implementación.

     ```
     int delta(int, char);
     ```

   - Una función de nombre `process` que calcule el estado final que es alcanzado después de procesar una cadena de *eventos* (caracteres). Recibe como argumentos un *estado* (un valor entero) y una *secuencia de eventos* (una cadena de caracteres), y retorna como resultado el estado que es alcanzado al final del procesamiento de la cadena recibida. Esta función es la implementación de la función de procesamiento δ'.

     ```
     int process(int, char *);
     ```

- Una función `test` que ejecute el procesamiento de una cadena (función `process`) e informe si el resultado ha sido *positivo* (después del procesamiento se ha alcanzado un estado de aceptación) ó *negativo* (después del procesamiento se ha alcanzado un estado de no aceptación). Esta función recibe como argumento una *secuencia de eventos* (una cadena de caracteres) y devuelve como resultado *falso* ó *verdadero* para los casos negativo ó positivo, respectivamente.

```
bool test(char *);
```

Versión inductiva de la función `process`

Asuma un autómata finito determinista $A = (Q, \Sigma, \delta, q_0, F)$ y una cadena $w \in \Sigma^*$. Se define la función de procesamiento $\delta' : Q \times \Sigma^* \rightarrow Q$ la siguiente manera:

$$\delta'(q, w) = \begin{cases} q & \text{si } w = \lambda \\ \delta'(\delta(q, a), x) & \text{si } w = ax, a \in \Sigma \end{cases} \quad (1)$$

En esta versión de la función `process`, su programa deberá de calcular su resultado mediante la implementación explícita de la función δ', por lo que el cálculo deberá resolverse mediante el uso de la inducción (llamadas *recursivas*) en la función de procesamiento.

Observe que la definición (1) de la *función de procesamiento* difiere ligeramente de la definición estudiada en clase. La versión que aquí se propone presenta ventajas para su implementación, por lo que se sugiere apegarse a ella. Esta forma permite procesar uno a uno los eventos de la cadena (a), comenzando por el primero de ellos y continuando así hacia la derecha, reservando en cada caso el *sufijo* restante x para el paso posterior. El caso base será alcanzado cuando x se haga vacío. Este evento será fácilmente detectado cuando el evento actual sea igual al *final de la cadena* (`'\0'`). El uso de apuntadores para el acceso al evento le facilitará esta implementación.

Versión iterativa de la función `process`

Se define la función de procesamiento $\delta' : Q \times \Sigma^* \rightarrow Q$ de la manera acostumbrada:

$$\delta'(q, w) = \begin{cases} q & \text{si } w = \lambda \\ \delta(\delta'(q, x), a) & \text{si } w = xa, a \in \Sigma \end{cases} \quad (2)$$

Para su versión *iterativa*, en la función `process` deberá implementar un procesamiento cíclico de acuerdo con el diagrama de la figura 2. En el primer paso se calcula la *base* del procesamiento –$\delta'(q_0, \lambda)$– y posteriormente, uno a uno, los eventos de la cadena que se está procesando, tomando cada vez el resultado del paso anterior (la *inducción*) para el cálculo del nuevo estado alcanzado.

Atienda además a los siguientes requerimientos:

- Los *eventos* deberán representados por caracteres (**`char`**), siendo entonces posible representar una *secuencia de eventos* como una cadena de caracteres estándar (**`char *`**).
- Los estados del sistema deberán implementarse como números enteros no negativos (0, 1, 2, etc.), siendo siempre el estado 0 el estado inicial.
- La indefinición de estado (un *estado indefinido* es aquel que es alcanzado cuando no existe alguna transición definida para el evento actual) deberá implementarse como un número negativo. Por ejemplo, –1. Un estado indefinido ocurre cuando sucede un evento para el cual no existe una transición definida. Por ejemplo, para un evento que no pertenece al alfabeto del sistema.

Procure atender al principio de señalar e identificar el final de una cadena mediante el carácter nulo (`'\0'`), como es habitual en el lenguaje C. El uso de esta marca facilitará la implementación de algunas de sus operaciones.

2. **Lenguaje de implementación**. Se exhorta al uso del lenguaje ANSI C/C++ como lenguaje de desarrollo. El uso de interfaces gráficas y/o entornos de desarrollo será opcional, y su uso se considerará responsabilidad del alumno. Cualquier inconveniente ocurrido durante la revisión que sea originado por el uso de estos será causa de su anulación.

3. **Asesoría para su elaboración**. El alumno podrá asesorarse en todo momento con el profesor durante el desarrollo de su proyecto, empleando para ello preferentemente los horarios destinados a este fin.

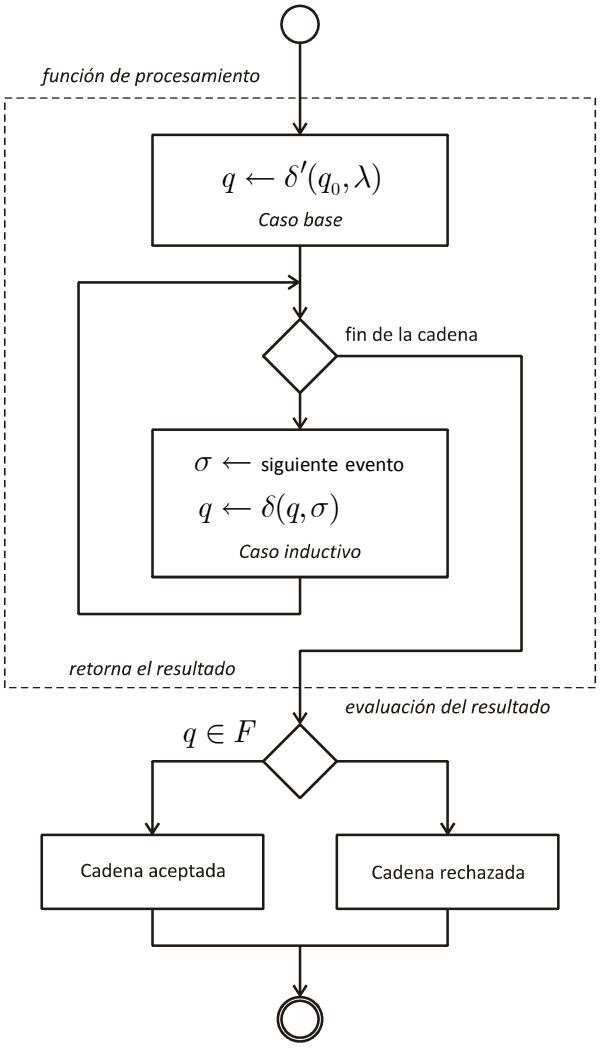

Figura 2. Procesamiento iterativo de cadenas.

Requerimientos para la evaluación de la práctica

Los siguientes requerimientos serán solicitados para la evaluación de esta práctica:

1. **Verificación de la aplicación**. Se hará una corrida de prueba, durante la cual se proporcionará una cadena arbitraria de ceros y unos al programa. Se hará la verificación de los resultados de acuerdo a lo expuesto en la sección **Desarrollo**. Además se verificará que la correcta implementación de sus funciones, de acuerdo a los requerimientos presentados arriba.

   Aunque para esta práctica no es requerido, se considerará una apreciación favorable el que se informe en cada paso del análisis cual es el estado actual del sistema, cual es el próximo símbolo a procesar y cuál es el *estatus* actual del procesamiento (*aceptación* ó *rechazo*).

2. **Comprensión del problema**. El alumno deberá tener una clara comprensión técnica y teórica del problema, la cual podrá ejercitar durante el desarrollo de esta práctica. Este punto será evaluado mediante una indagación rápida que el profesor hará durante la verificación de la aplicación así como la revisión del código fuente.

3. **Documentación del proyecto**. Deberá elaborarse un reporte de práctica de acuerdo a lo que se expone en el *Manual para la elaboración de los Reportes de Práctica*. Haga énfasis en los algoritmos y técnicas empleadas para la implementación de sus funciones. La entrega de la documentación impresa será un requisito necesario para dar inicio a la revisión de su proyecto.

> Procure hacer una caracterización del lenguaje del autómata A e inclúyala en su reporte. De esta manera le será más fácil proporcionar cadenas de prueba a su programa además de tener un mejor entendimiento del sistema.

Ejemplo

Suponga que se proporciona la siguiente entrada: `101100101`. Entonces su programa debería generar una salida similar a la siguiente:

```
[estado,evento] estatus
-----------------------
[0,1] Rechazo
[2,0] Rechazo
[1,1] Rechazo
[3,1] Rechazo
[4,0] Rechazo
[5,0] Rechazo
[5,1] Rechazo
[6,0] Aceptacion
[5,1] Rechazo
[6,-] Aceptacion

Fin del procesamiento
Resultado: La cadena es aceptada
```

Por otro lado, una cadena como `00101110111` debería ser rechazada, ya que el último estado alcanzado es el estado 4 (q_4).