Tarea 2
CC4101 - Lenguajes de Programación

Éric Tanter
Auxiliar: Richard Ibarra

Fecha de Entrega. Viernes 30 de Abril *

Programar con máquinas de estado

1. **(2 pts)** Una máquina de estado es un proceso que consume una lista de simbolos en entrada y retorna `true` si acepta el input, `false` si no. A medida que consume simbolos, va cambiando de estado de acuerdo a ciertas transiciones entre estados. En esta tarea, para simplificar, consideramos que cada estado es un estado de aceptación: o sea, una máquina acepta cualquier input que pueda consumir enteramente. Además, en vez de trabajar con simbolos, trabajaremos con *strings*.

   Por ejemplo, el siguiente código Scheme define una máquina que reconoce la expresión regular $(a^+b)^*$ (por ej. `"ab"`, `"abaaabab"`). Tiene dos estados, `waita` (el estado inicial) y `waitab`.

   ```
   (define m '(waita
               (waita ("a" waitab))
               (waitab ("a" waitab)
                       ("b" waita))))
   ```

   (a) **(0.3pt)** Como pueden apreciar, representamos una máquina de estado con una lista. Más precisamente, como estan representados los estados de la máquina? las transiciones? Como se especifica el estado inicial?

   (b) **(0.3pt)** Describa en BNF la sintáxis de definición de máquinas de estados.

   (c) **(0.4pt)** Define usando esa sintáxis una máquina de estado que acepta las strings de input correspondientes a los identificadores `car`, `cdr`, `cadr`, `caddr`, `cddar`, etc.

   (d) **(1pt)** Define una función Scheme llamada `run-sm` que, dado una máquina y un string de input, retorna `#t` si la máquina acepta el input, `#f` sino.

   Hints:

   - conviertan la string de input en una lista de strings de largo 1 usando la función siguiente:

     ```
     (define (as-list s) (map string (string->list s)))
     > (as-list "aaaab")
     ("a" "a" "a" "a" "b")
     ```

   - una lista de listas en Scheme se llama "lista de asociaciones" y es posible hacer busqueda en ella usando `assoc`, quien retorna la lista cuyo primer elemento es `equal?` al especificado:

*Ver las reglas de entrega y evaluación en `http://pleiad.cl/teaching/cc4101/reglas`.

```
> (assoc 'waita '((waita  ("a" waitab))
                  (waitab ("a" waitab) ("b" waita))))
(waita ("a" waitab))
> (assoc "a" '(("a" waitab) ("b" waita)))
("a" waitab)
```

2. **(3 pts)** Ahora, queremos extender el lenguaje WAE visto en clases para que sea posible programar con máquinas de estado directamente.

   (a) **(0.7pt)** Primero, para hacer el lenguaje más interesante, agreguele valores booleanos, y strings. También agregue una expresión condicional `if` (similar al `if` de Scheme), así como al menos una funcion para hacer comparaciones numéricas. Ejemplo:

   ```
   {with {x 1}
     {with {y 3}
       {if {< {+ x y} 0}
          "well done!"
          {- y 1}}}}
   ```

   (b) **(1.5pt)** Agregue la posibilidad de definir máquinas de estado, usando la misma sintáxis vista anteriormente:

   ```
   {with {m {sm (waita (waita ("a" waitab))
                       (waitab ("a" waitab)
                               ("b" waita))))}}
     {if {m "aaab"}
       "matched!"
       "not matched"}}

   --> "matched!"
   ```

   Notese la sintáxis para crear máquinas (`sm`) y para ejecutarlas (igual a una aplicación de función).

   (c) **(0.3pt)** Para permitir interacción con el usuario, extienda el lenguaje para soportar el `read` de Scheme. Incluya un ejemplo.

   (d) **(0.5pt)** En lo anterior, la definición de una máquina de estado contiene strings directamente en sus transiciones. Extienda su lenguaje para permitir el uso de cualquier expresión (que tiene que evaluarse a una string en el momento de la definición de la máquina). Ilustre.