Práctica 4

Representación del Conocimiento

Práctica de CLIPS

Autores: Álvaro López García,
Jairo González Gómez,
Nicolas Rodrigo Pérez.

Fecha: 6 de Enero de 2022

Autómatas finitos.

Los autómatas finitos proporcionan un sistema de computación, en el que, a partir de una entrada, devuelven verdadero o falso. En este caso, el estado de inicio es q_0 y, con la entrada 0011 se devuelve verdadero. La computación completa podría ser la siguiente:

$$(q_0, 0011) \Longrightarrow (q_1, 011) \Longrightarrow (q_2, 11) \Longrightarrow (q_2, 1) \Longrightarrow (q2,).$$

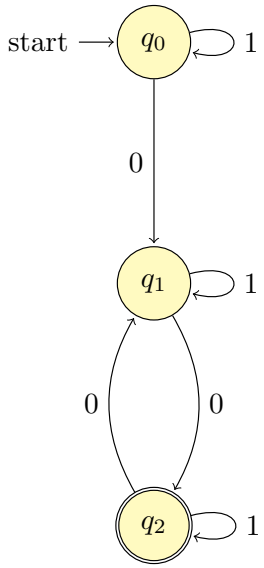


Figura 1: Autómata finito con tres estados.

Primera pregunta.

Programad con CLIPS un conjunto de reglas lógicas, de forma que, dada una entrada, se imprima por pantalla si el autómata dado en el apartado anterior acepta la entrada.
Código empleado para la resolución de este apartado[1]:

```
(deffacts initial
  (estado q0)
  (caracter "0")
  (entrada "0" "1" "1" "lambda")
)

(defrule acepta_palabra
  ?estado_actual <- (estado q2)
  ?cinta <- (entrada)
 =>
  (printout t "Palabra Aceptada" crlf)
)

(defrule transicion_q0_0
  ?estado_actual <- (estado q0)
  ?simbolo_actual <- (caracter "0")
  ?cinta <- (entrada ?siguiente $?otros)
 =>
  (retract ?estado_actual)
  (retract ?simbolo_actual)
  (retract ?cinta)
  (assert (caracter ?siguiente))
  (assert (estado q1))
```

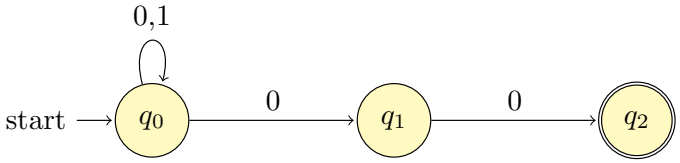
[1] Archivo adjunto `Pregunta1.clp`.

```
    (assert (entrada $?otros))
    (printout t "q0 -> 0 -> q1" crlf)
)

(defrule transicion_q0_1
    ?estado_actual <- (estado q0)
    ?simbolo_actual <- (caracter "1")
    ?cinta <- (entrada ?siguiente $?otros)
 =>
    (retract ?estado_actual)
    (retract ?simbolo_actual)
    (retract ?cinta)
    (assert (caracter ?siguiente))
    (assert (estado q0))
    (assert (entrada $?otros))
    (printout t "q0 -> 1 -> q0" crlf)
)

(defrule transicion_q1_0
    ?estado_actual <- (estado q1)
    ?simbolo_actual <- (caracter "0")
    ?cinta <- (entrada ?siguiente $?otros)
 =>
    (retract ?estado_actual)
    (retract ?simbolo_actual)
    (retract ?cinta)
    (assert (caracter ?siguiente))
    (assert (estado q2))
    (assert (entrada $?otros))
    (printout t "q1 -> 0 -> q2" crlf)
)

(defrule transicion_q1_1
    ?estado_actual <- (estado q1)
    ?simbolo_actual <- (caracter "1")
    ?cinta <- (entrada ?siguiente $?otros)
 =>
    (retract ?estado_actual)
    (retract ?simbolo_actual)
    (retract ?cinta)
    (assert (caracter ?siguiente))
    (assert (estado q1))
    (assert (entrada $?otros))
    (printout t "q1 -> 1 -> q1" crlf)
)

(defrule transicion_q2_0
    ?estado_actual <- (estado q2)
    ?simbolo_actual <- (caracter "0")
    ?cinta <- (entrada ?siguiente $?otros)
 =>
    (retract ?estado_actual)
    (retract ?simbolo_actual)
    (retract ?cinta)
    (assert (caracter ?siguiente))
    (assert (estado q1))
    (assert (entrada $?otros))
    (printout t "q2 -> 0 -> q1" crlf)
)

(defrule transicion_q2_1
    ?estado_actual <- (estado q2)
    ?simbolo_actual <- (caracter "1")
```

```
  ?cinta <- (entrada ?siguiente $?otros)
 =>
  (retract ?estado_actual)
  (retract ?simbolo_actual)
  (retract ?cinta)
  (assert (caracter ?siguiente))
  (assert (estado q2))
  (assert (entrada $?otros))
  (printout t "q2 -> 1 -> q2" crlf)
)
```

Segunda pregunta.

El mismo problema si el autómata deja de ser determinista.



Código empleado para la resolución de este apartado[2]:

```
(deffacts initial
  (automata q0 "0" "0" "1" "1" "$")
)

(defrule acepta_palabra
  (declare (salience 0))
  (automata q2 "$")
 =>
  (printout t "Palabra Aceptada" crlf)
)

(defrule rechaza_palabra
  (declare (salience 0))
  (not (automata q2 "$"))
 =>
  (printout t "Palabra Rechazada" crlf)
)

(defrule error
  (declare (salience 1))
  ?automata <- (automata ?qaux $siguiente $?otros)
 =>
  (retract ?automata)
)

(defrule transicion_q0_1
  (declare (salience 2))
  ?automata <- (automata q0 "1" $?otros)
 =>
  (retract ?automata)
  (assert (automata q0 $?otros))
  (printout t "q0 -> 1 -> q0" crlf)
)

(defrule transicion_q0_0
  (declare (salience 2))
```

[2]Archivo adjunto `Pregunta2_1.clp`.

```
  ?automata <- (automata q0 "0" $?otros)
 =>
  (retract ?automata)
  (assert (automata q0 $?otros))
  (assert (automata q1 $?otros))
  (printout t "q0 -> 0 -> q0" crlf)
  (printout t "q0 -> 0 -> q1" crlf)
)

(defrule transicion_q1_0
  (declare (salience 3))
  ?automata <- (automata q1 "0" $?otros)
 =>
  (retract ?automata)
  (assert (automata q2 $?otros))
  (printout t "q1 -> 0 -> q2" crlf)
)
```

Además, como tarea adicional, se puede mejorar el código anterior, En vez de definir una regla para cada transición, definid una regla que sirva para todas las transiciones. Para ello es necesario definir las transciones como hechos, como se puede ver en el siguiente ejemplo:

```
(assert (transicion q0 "1" q0))
(assert (transicion q0 "0" q1))
(assert (transicion q0 "0" q1))
(assert (transicion q1 "0" q2))
```

Y escribir una única regla que utilice estos hechos como transiciones. Para ello, se menciona que cuan- do una variable esta definida en la parte izquierda de la regla, queda ligada. Esto es, que el valor que toma ya esta asignado. Se puede consultar el libro de la bibliografía «CLIPS User's Guide», página 44.

Código empleado para la resolución de este apartado[3]:

```
(deffacts initial
  (automata q0 "1" "1" "0" "0" "$")
  (transicion q1 "0" q2)
  (transicion q0 "0" q1)
  (transicion q0 "1" q0)
  (transicion q0 "0" q0)
)

(defrule acepta_palabra
  (declare (salience 0))
  (automata q2 "$")
 =>
  (printout t "Palabra Aceptada" crlf)
)

(defrule rechaza_palabra
  (declare (salience 0))
  (not (automata q2 "$"))
 =>
  (printout t "Palabra Rechazada" crlf)
)

(defrule error
  (declare (salience 1))
  ?automata <- (automata ?qaux $siguiente $?otros)
```

[3] Archivo adjunto `Pregunta2_2.clp`.

```
 =>
   (retract ?automata)
)


(defrule transicciona
   (declare (salience 2))
   ?transicion <- (transicion ?qstart ?char ?qend)
   ?automata <- (automata ?qstart ?char $?otros)
 =>
   (retract ?automata)
   (assert (automata ?qend ?otros))
   (printout t ?qstart " -> " ?char " -> " ?qend crlf)
)
```

Tercera pregunta.

La tercera pregunta trata de encontrar una palabra que sea aceptado por un autómata. Se pide que se escriba un programa que dada una representación de un autómata, devuelva una palabra que se acepte por el autómata en caso de existir. El programa no puede entrar en bucle infinito para ninguna entrada.

Código empleado para la resolución de este apartado[4]:

```
(deffacts initial
   (estado q0)
   (transicion q0 "0" q1)
   (transicion q0 "1" q0)
   (transicion q1 "0" q2)
   (estado_final q2)
)

(defrule palabra_aceptada
   (not (transicion ?qini ?char ?qfin))
   ?estado <- (estado ?q $?chars)
   ?estado_final <- (estado_final ?q)
 =>
   (printout t $?chars crlf)
)

(defrule recorre
   ?transicion <- (transicion ?qini ?char ?qfin)
   ?estado <- (estado ?qini $?caracteres)
 =>
   (retract ?estado)
   (retract ?transicion)
   (assert (estado ?qfin $?caracteres ?char))
)
```

[4] Archivo adjunto `pregunta3.clp`.