Intelligent Systems: Reasoning and Recognition

James L. Crowley

ENSIMAG 2 / MoSIG M1 Second Semester 2014/2015

Lesson 8 - Exercise 3 6 March 2015

Exercise : Family Relations

The goal of this exercise is to program a set of classes and message handlers that can respond to questions about the relations within a family. Family relations, such as father, mother, brother and sister are represented by slots. Answers are determined by "handlers"

a) Define an abstract class for person with slots name, father, mother, brother and sister. The slots for brother and sister must be multi-slots so that they can contain a list.

Define a concrete class for MAN as a subclass of person, with the slots "wife" and "gender" having fixed values of "male".

```
(defclass PERSON (is-a USER)
     (role abstract)
     (slot ID (create-accessor read-write))
     (slot father (create-accessor read-write) (default unknown))
     (slot mother (create-accessor read-write) (default unknown))
     (multislot brothers (create-accessor read-write))
     (multislot sisters (create-accessor read-write))
)
```

Define a concrete class for WOMAN as a subclass of person, with the slots "husband" and "gender" having fixed values of "female".

b) Create a rule to build the family structure by asking for the wife for a man, and the husband for a wife, and the father and mother for each person.

```
(defclass MAN (is-a PERSON)
     (role concrete)(pattern-match reactive)
     (slot wife (create-accessor  read-write) (default unknown))
     (slot gender (storage shared)
               (default male) (create-accessor read))
)

(defclass WOMAN (is-a PERSON)
     (role concrete)(pattern-match reactive)
     (slot husband (create-accessor read-write) (default unknown))
     (slot gender (storage shared)(access read-only)
               (default female) (create-accessor read))
)
```

c) Define the message handlers for the class PERSON that can determine the objects that represent the paternal Grandmother and Grandfather.

```
(defrule ask-wife
     ?M <- (object (is-a MAN) (ID ?n) (wife  unknown))
=>
     (printout t "Who is the wife of " ?n "? ")
     (bind ?ID (read))
     (send ?M put-wife ?ID)
     (if (neq ?ID nil) then
          (make-instance ?ID of WOMAN (ID ?ID) (husband ?n)))
)

(make-instance [Jean] of MAN (ID Jean))
(make-instance [Paul] of MAN (ID Paul))

(run 1)
(send [Paul] get-wife)

(defrule ask-father
     ?M <- (object (is-a MAN) (ID ?n) (father  unknown))
=>
     (printout t "Who is the father of " ?n "? ")
 (bind ?ID (read))
     (send ?M put-father ?ID)
 (make-instance ?ID of MAN (ID ?ID))
)
```

d) Define the message handlers that return the Names of the paternal grandfather and grandmother.

```
(defmessage-handler PERSON paternal-grandfather ()
     (bind ?g-father (send ?self:father get-father))
 (send ?g-father get-ID)
)

(defmessage-handler PERSON paternal-grandmother ()
     (bind ?g-father (send ?self:father get-mother))
 (send ?g-father get-ID)
)
```

e) Define a message handler to determine the pointers to the uncles of a person. (brother of father and brothers of mother). Hint: a list can be created with the function create$.
Ex : (a b c) <- (create$ a b c)

```
(defmessage-handler PERSONNE uncles ()
     (create$ (send ?self:father get-brothers)
              (send ?self:mother get-brothers)
 )
)
```

f) Define a message handler to determine the names of the uncles.

```
(defmessage-handler PERSON name-the-uncles ()
     (bind $?uncles
    (create$ (send ?self:father get-brothers)
             (send ?self:mother get-brothers)
    )
 )
 (progn$ (?uncle $uncles)
     (printout t "the names of " ?oncle " is ")
     (printout t (send ?Uncle get-ID) crlf)
  )
)

(defrule ask-brother
     ?M <- (object (is-a MAN) (ID ?ID) (brothers $?brothers))
     (test (eq (nth 1  $?brothers) unknown))
=>
     (printout t "Who is the brother of " ?ID "? ")
     (bind ?b (read))
     (if  (eq ?b nil) then (bind $?brothers (delete$ $?brothers 1
1))
          else (replace$ $?brothers 1 1 ?b))
     (send ?M put-brothers $?brothers))
)
```

g) Define the message handler to determine the list of names for all of the grandparents.

```
(defmessage-handler PERSON grand-parents ()
   (create$
      (send (send ?self:father get-father) get-ID)
      (send (send ?self:father get-mother) get-ID)
      (send (send ?self:mother get-father) get-ID)
      (send (send ?self:mother get-mother) get-ID)
   )
)
```