Esercitazione CLIPS (I)

IL PROBLEMA DELL'AGRICOLTORE, IL CAVOLO, LA PECORA, IL LUPO

Dati:

- **Situazione Iniziale:**
  Sulla riva di un fiume ci sono un agricoltore, una pecora, un lupo ed un cavolo.
- **Situazione Finale:**
  Portare i protagonisti sull'altra riva del fiume
- **Vincoli:**
  - L'agricoltore può attraversare il fiume portando con se solo una cosa per volta
  - la pecora non può stare da sola ne con il cavolo ne con il lupo

Trovare:

La sequenza di mosse che l'agricoltore deve compiere per raggiungere il suo scopo.

Base di Conoscenza: Fatti

I **fatti** in questo problema rappresenteranno gli *stati legali*, ossia le configurazioni che possono essere accettate in ingresso (come visto, nella descrizione del problema si evince che la pecora non può trovarsi da sola con il cavolo o da sola con il lupo).

Ci sarà bisogno di:

3.Un ***template*** per memorizzare le configurazioni. Esso sarà costituito da 4 slot contenente l'informazione relative ad ogni personaggio:

Nome (A – P – L- C)

Posizione ("riva-vicina" – "riva-lontana")

L'informazione relativa alla posizione risulta necessaria nella scelta della regola da applicare.

… Base di Conoscenza: Fatti

```
(deftemplate configurazione "Stato del gioco"
      (slot agricoltore (type STRING)
                        (default "sulla-riva-vicina"))
      (slot cavolo      (type STRING)
                        (default "sulla-riva-vicina"))
      (slot pecora      (type STRING)
                        (default "sulla-riva-vicina"))
      (slot lupo        (type STRING)
                        (default "sulla-riva-vicina")))
```

… Base di Conoscenza: Fatti

1. Un ***template*** per memorizzare la configurazione iniziale. Esso sarà costituito da un solo slot contenente la configurazione iniziale scelta dall'utente:

```
(deftemplate inizia "Configurazione iniziale"
   (slot config_iniziale  (type SYMBOL)
                          (default ACPL)))
```

… Base di Conoscenza: Fatti

1. Un insieme di possibili configurazioni legali in cui i personaggi possono trovarsi.

```
(deffacts stati_legali  "Stati validi nel gioco"
   (stato  ACPL)  ; Tutti i personaggi sono sulla riva 1
   (stato  ____)  ; Tutti i personaggi sono sulla riva 2
   (stato  A_P_)  ; Agricoltore e pecora sulla riva 1
                  ; Cavolo e Lupo sulla riva 2
   (stato  _C_L)  ; Cavolo e Lupo sulla riva 1
                  ; Agricoltore e pecora sulla riva 2
   (stato  A_PL)  ;...
   (stato  _C__)
   (stato  AC_L)
   (stato  ___L)
   (stato  ACP_)
   (stato  __P_))
```

Base di Conoscenza: Regole

Le **regole** che si andranno a definire agiranno sulla configurazione iniziale inserita dall'utente.

La loro funzione sarà quella di:

- testare la configurazione inserita rispetto alle 10 legali
- eventualmente, eseguire azioni che, cambiando lo stato dei personaggi, portino alla configurazione finale

Segnaliamo e fermiamo il processo nel caso in cui la configurazione data in input non risultasse esatta

Se i quattro personaggi sono tutti sulla riva vicina, **allora** l'agricoltore può andare sulla riva lontana e portare con sé la pecora.

```
(defrule regola1
   ?indice_conf1 <-
   (configurazione  (agricoltore  "sulla-riva-vicina")
                    (cavolo       "sulla-riva-vicina")
                    (pecora       "sulla-riva-vicina")
                    (lupo         "sulla-riva-vicina"))
  =>
    (printout t "Trasborda la pecora" crlf)
    (retract ?indice_conf1)
    (assert(configurazione(agricoltore "sulla-riva-lontana")
                          (cavolo      "sulla-riva-vicina")
                          (pecora      "sulla-riva-lontana")
                          (lupo        "sulla-riva-vicina"))))
```

Se l'agricoltore e la pecora sono sulla riva lontana **allora** l'agricoltore può tornare sulla riva vicina (*sarebbe inutile la regola che, in questa situazione portasse con sé la pecora*).

```
(defrule regola2
   ?indice_conf2 <-
   (configurazione  (agricoltore  "sulla-riva- lontana")
                    (cavolo       "sulla-riva-vicina")
                    (pecora       "sulla-riva-lontana")
                    (lupo         "sulla-riva-vicina"))
  =>
    (printout t "Ritorna alla riva vicina" crlf)
    (retract ?indice_conf2)
    (assert(configurazione(agricoltore "sulla-riva-vicina")
                          (cavolo      "sulla-riva- vicina ")
                          (pecora      "sulla-riva-lontana")
                          (lupo        "sulla-riva-vicina"))))
```

Se l'agricoltore è sulla riva vicina con il lupo e il cavolo, **allora** l'agricoltore può andare sulla riva lontana portando con sé il lupo.

```
(defrule regola3
   ?indice_conf3 <-
   (configurazione  (agricoltore  "sulla-riva-vicina")
                    (cavolo       "sulla-riva-vicina")
                    (pecora       "sulla-riva-lontana")
                    (lupo         "sulla-riva-vicina"))
  =>
    (printout t "Trasborda il lupo" crlf)
    (retract ?indice_conf3)
    (assert(configurazione(agricoltore "sulla-riva-lontana")
                          (cavolo      "sulla-riva-vicina")
                          (pecora      "sulla-riva-lontana")
                          (lupo        "sulla-riva-lontana"))))
```

ma anche

```
(defrule regola4
   ?indice_conf4 <-
   (configurazione  (agricoltore  "sulla-riva-vicina")
                    (cavolo       "sulla-riva-vicina")
                    (pecora       "sulla-riva-lontana")
                    (lupo         "sulla-riva-vicina"))
  =>
    (printout t "Trasborda il cavolo" crlf)
    (retract ?indice_conf4)
    (assert(configurazione(agricoltore "sulla-riva-lontana")
                          (cavolo      "sulla-riva-lontana")
                          (pecora      "sulla-riva-lontana")
                          (lupo        "sulla-riva-vicina"))))
```

Se l'agricoltore la pecora e il lupo (o il cavolo) sono sulla riva lontana **allora** l'agricoltore può andare sulla riva vicina portando con sé la pecora.

```
(defrule regola5
   ?indice_conf5 <-
   (configurazione  (agricoltore  "sulla-riva-lontana")
                    (cavolo       "sulla-riva-vicina")
                    (pecora       "sulla-riva-lontana")
                    (lupo         "sulla-riva-lontana"))
  =>
    (printout t "Riporta indietro la pecora" crlf)
    (retract ?indice_conf5)
    (assert(configurazione(agricoltore "sulla-riva-vicina")
                          (cavolo      "sulla-riva-vicina")
                          (pecora      "sulla-riva-vicina")
                          (lupo        "sulla-riva-lontana"))))
```

```
(defrule regola6
   ?indice_conf6 <-
   (configurazione  (agricoltore  "sulla-riva-lontana")
                    (cavolo       "sulla-riva-lontana")
                    (pecora       "sulla-riva-lontana")
                    (lupo         "sulla-riva-vicina"))
  =>
    (printout t "Riporta indietro la pecora" crlf)
    (retract ?indice_conf6)
    (assert(configurazione(agricoltore "sulla-riva-vicina")
                          (cavolo      "sulla-riva-lontana")
                          (pecora      "sulla-riva-vicina")
                          (lupo        "sulla-riva-vicina"))))
```

Se solo il lupo è sulla riva lontana **allora** l'agricoltore può andare sulla riva lontana portando con sé il cavolo.

```
(defrule regola7
   ?indice_conf7 <-
   (configurazione  (agricoltore  "sulla-riva-vicina")
                    (cavolo       "sulla-riva-vicina")
                    (pecora       "sulla-riva-vicina")
                    (lupo         "sulla-riva-lontana"))
  =>
    (printout t "Trasborda il cavolo" crlf)
    (retract ?indice_conf7)
    (assert(configurazione(agricoltore "sulla-riva-lontana")
                          (cavolo      "sulla-riva-lontana")
                          (pecora      "sulla-riva-vicina")
                          (lupo        "sulla-riva-lontana"))))
```

Se, invece, è solo il cavolo ad essere sulla riva lontana **allora** l'agricoltore può andare sulla riva lontana portando con sé il lupo.

```
(defrule regola8
   ?indice_conf8 <-
   (configurazione  (agricoltore  "sulla-riva-vicina")
                    (cavolo       "sulla-riva-lontana")
                    (pecora       "sulla-riva-vicina")
                    (lupo         "sulla-riva-vicina"))
  =>
    (printout t "Trasborda il lupo" crlf)
    (retract ?indice_conf8)
    (assert(configurazione(agricoltore "sulla-riva-lontana")
                          (cavolo      "sulla-riva-lontana")
                          (pecora      "sulla-riva-vicina")
                          (lupo        "sulla-riva-lontana"))))
```

Se c'è solo la pecora sulla riva vicina **allora** l'agricoltore può andare sulla riva vicina.

```
(defrule regola9
   ?indice_conf9 <-
   (configurazione  (agricoltore  "sulla-riva-lontana")
                    (cavolo       "sulla-riva-lontana")
                    (pecora       "sulla-riva-vicina")
                    (lupo         "sulla-riva-lontana"))
  =>
    (printout t "Ritorna alla riva vicina" crlf)
    (retract ?indice_conf9)
    (assert(configurazione(agricoltore "sulla-riva-vicina")
                          (cavolo      "sulla-riva-lontana")
                          (pecora      "sulla-riva-vicina")
                          (lupo        "sulla-riva-lontana"))))
```

Se l'agricoltore è sulla riva vicina con la sola pecora **allora** l'agricoltore può andare sulla riva lontana portando con sé la pecora.

```
(defrule regola10
   ?indice_conf10 <-
   (configurazione  (agricoltore  "sulla-riva-vicina")
                    (cavolo       "sulla-riva-lontana")
                    (pecora       "sulla-riva-vicina")
                    (lupo         "sulla-riva-lontana"))
  =>
    (printout t "Trasborda la pecora" crlf)
    (retract ?indice_conf10)
    (assert(configurazione(agricoltore "sulla-riva-lontana")
                          (cavolo      "sulla-riva-lontana")
                          (pecora      "sulla-riva-lontana")
                          (lupo        "sulla-riva-lontana"))))
```

Definiamo la regola per iniziare a giocare

```
(defrule chiedi_configurazione_iniziale
      (declare (salience 10000))
 =>
(printout t crlf"****************************" crlf crlf
          "Inserisci la configurazione iniziale:" crlf crlf)
(printout t "L'agricoltore è sulla riva vicina? (S/N): ")
(bind ?agric (read))
(if (eq ?agric S) then (bind ?temp_a A) else (bind ?temp_a _))
(printout t "Il cavolo è sulla riva vicina?     (S/N): ")
(bind ?cav (read))
(if (eq ?cav S) then (bind ?temp_c C) else (bind ?temp_c _))
(printout t "La pecora è sulla riva vicina?     (S/N): ")
(bind ?pec (read))
(if (eq ?pec S) then (bind ?temp_p P) else (bind ?temp_p _))
(printout t "Il lupo è sulla riva vicina?       (S/N): ")
(bind ?lup (read))
(if (eq ?lup S) then (bind ?temp_l L) else (bind ?temp_l _))
(bind ?config (sym-cat ?temp_a ?temp_c ?temp_p ?temp_l))
(assert (inizia (config_iniziale ?config)))
(assert (config inesatta))
(printout t crlf crlf))
```

Definiamo la regola per controllare la configurazione di input

```
(defrule controlla_input
  (declare (salience 9000))
  ?indice <- (inizia (config_iniziale ?conf_iniz))
  (stato ?conf_iniz)
  ?ind <- (config inesatta)
 =>
  (retract ?indice ?ind)
  (assert (config esatta))
  (if (eq ?conf_iniz ACPL)
    then (assert (configurazione))
    else (if (eq ?conf_iniz ____) then
    (assert (configurazione (agricoltore "sulla-riva-lontana")
                            (cavolo "sulla-riva-lontana")
                            (pecora "sulla-riva-lontana")
                            (lupo "sulla-riva-lontana")))
       else ...

  (printout t "Le mosse dell'agricoltore sono: " crlf crlf))
```

Definiamo la regola per segnalare che la configurazione iniziale è inesatta

```
(defrule ritenta "Se la configurazione data in input è errata"
  (declare (salience 8500))
  ?indice <- (config inesatta)
 =>
  (retract ?indice)
  (printout t crlf "La configurazione iniziale data in input
                          risulta illegale!!!" crlf crlf crlf)
  (printout t "Vuoi riprovare? (S/N): ")
  (bind ?risposta (read))
  (if (eq ?risposta S) then (reset)
      else (printout t crlf crlf"***************" crlf crlf)))
```

Definiamo la regola per continuare a giocare dopo il termine di una sessione di gioco

```
(defrule continua_gioco "L'utente vuole giocare ancora?“
  (declare (salience -10000))
  ?indice <- (config esatta)
 =>
  (retract ?indice)
  (printout t crlf crlf crlf "Vuoi giocare ancora? (S/N): ")
  (bind ?risposta(read))
  (if (eq ?risposta S) then (reset)
      else (printout t crlf "********************" crlf crlf)))
```

Esercizio 1

Si vuole modellare un incrocio stradale con un semaforo ed un segnale per i pedoni. Le decisioni da prendere sono se: ***1. Camminare***, ***2. Non camminare*** e ***3. Essere cauti***

Il risultato dovrebbe essere fornito in termini di una asserzione **walk-status** e stampe a video.

Le luci del semaforo e del segnale per i pedoni hanno le seguenti condizioni:

Segnale luminoso del semaforo

Verde - Giallo - Rosso - Guasto - Lampeggiante

Segnale per i pedoni

Avanti - Stop - Guasto

Esercizio 1 (linee guida)

Utilizzare un ***template*** intermedio che ha le condizioni di entrambi i segnali:

- stato-segnale-semaforo
- stato-segnale-pedoni

Un insieme di regole dovrebbe gestire il semaforo (***stato-segnale-semaforo***) ed un altro insieme di regole dovrebbe gestire il segnale per i pedoni (***stato-segnale-pedoni***).

Esercizio 2

Creare uno stimatore di durata della vita in base ai seguenti principi:

- La vita media di un uomo è di 70 anni
- La vita media di una donna è di 75 anni
- Il fumo riduce la durata della vita di 10 anni
- L'obesità riduce la durata della vita di 5 anni
- L'alcool riduce la durata della vita di 5 anni
- Non fare nessuna attività fisica riduce la durata della vita di 5 anni

Il problema dei missionari e dei cannibali

Tre missionari e tre cannibali si trovano sulla riva di un fiume. Tutti sono d'accordo sul fatto che vorrebbero andare dall'altra parte, ma i missionari temono di essere assaliti e mangiati dai cannibali. Quindi, i missionari vogliono realizzare l'attraversamento del fiume in modo tale che il numero di missionari su ognuna delle rive del fiume non sia mai minore del numero dei cannibali sulla stessa riva. L'unica barca a disposizione porta solo due persone alla volta.

Come si può fare perché tutti attraversino il fiume senza che i missionari rischino di venire mangiati?

Dati:

- **Situazione Iniziale:**
  Sulla riva di un fiume ci sono 3 missionari, 3 cannibali e una barca.
- **Situazione Finale:**
  Portare i protagonisti e la barca sull'altra riva del fiume.
- **Vincoli:**
  - Su ogni riva il numero di cannibali non deve superare il numero di missionari.
  - La barca può trasportare al massimo 2 passeggeri

Trovare:

La sequenza di mosse che devono essere eseguite per raggiungere lo stato finale.

Mosse Legali

sposta 1 cannibale (c)

C **-> <-** C *% un cannibale può essere trasportato sulla riva 1 o sulla riva 2*

sposta 2 c

CC **-> <-** CC *% due cannibali possono essere trasportati sulla riva 1 o sulla riva 2*

sposta 1 missionario (m)

M **-> <-** M *% un missionario può essere trasportato sulla riva 1 o sulla riva 2*

sposta 2 m

MM **-> <-** MM *% due missionari possono essere trasportati sulla riva 1 o sulla riva 2*

sposta 1 m e 1 c

MC **-> <-** MC *% un missionario e un cannibale può essere trasportato sulla riva 1 o sulla riva 2*

Una possibile soluzione

- *sposta 1 missionario e 1 cannibale sulla riva 2*
  MMMCCC 0 → MMCC MC
- *sposta 1 missionario sulla riva 1*
  MMCC MC → MMMCC C
- *sposta 2 cannibali sulla riva 2*
  MMMCC C → MMM CCC
- *sposta 1 cannibale sulla riva 1*
  MMM CCC → MMMC CC
- *sposta 2 missionari sulla riva 2*
  MMMC CC → MC MMCC
- *sposta 1 missionario e 1 cannibale sulla riva 1*
  MC MMCC → MMCC MC
- *sposta 2 missionari sulla riva 2*
  MMCC MC → CC MMMC
- *Sposta 1 cannibale sulla riva 1*
  CC MMMC → CCC MMM
- *Sposta 2 cannibali sulla riva 2*
  CCC MMM → C MMMCC
- *Sposta 1 missionario sulla riva 1*
  C MMMCC → CM MMCC
- *sposta 1 missionario e 1 cannibale sulla riva 1*
  CM MMCC → 0 MMMCCC

- La soluzione del problema, ovvero il percorso scelto, deve essere fornito come output
- Ci sono più soluzioni possibili per risolvere il problema
- Si corre il rischio di cadere in un loop infinito in alcuni casi

abbiamo bisogno di tenere traccia del percorso che stiamo seguendo onde evitare mosse inutili nella soluzione attuale o trovare la stessa soluzione che avevamo trovato in un primo momento.

Template per la memorizzazione degli stati correnti

```
(deftemplate MAIN::status
   (slot search-depth  (type INTEGER)
                       (range 1 ?VARIABLE))
   (slot parent (type FACT-ADDRESS SYMBOL)
                (allowed-symbols no-parent))
   (slot shore-1-missionaries (type INTEGER)
                              (range 0 ?VARIABLE))
   (slot shore-1-cannibals (type INTEGER)
                           (range 0 ?VARIABLE))
   (slot shore-2-missionaries (type INTEGER)
                              (range 0 ?VARIABLE))
   (slot shore-2-cannibals (type INTEGER)
                           (range 0 ?VARIABLE))
   (slot boat-location (type SYMBOL)
                       (allowed-values shore-1 shore-2))
   (slot last-move (type STRING)))
```

Conoscenza Iniziale

```
(defglobal MAIN ?*initial-missionaries* = 3
                ?*initial-cannibals* = 3)

(deffacts MAIN::initial-positions
  (status (search-depth 1)
          (parent no-parent)
          (shore-1-missionaries ?*initial-missionaries*)
          (shore-2-missionaries 0)
          (shore-1-cannibals ?*initial-cannibals*)
          (shore-2-cannibals 0)
          (boat-location shore-1)
          (last-move "No move.")))

(deffacts MAIN::boat-information
  (boat-can-hold 2))
```

Vincoli: Numero di cannibali su una riva minore di quello dei missionari

```
(defrule CONSTRAINTS::cannibals-eat-missionaries
   (declare (auto-focus TRUE))
    ?node <- (status (shore-1-missionaries ?s1m)
                     (shore-1-cannibals ?s1c)
                     (shore-2-missionaries ?s2m)
                     (shore-2-cannibals ?s2c))
   (test (or (and (> ?s2c ?s2m) (<> ?s2m 0))
             (and (> ?s1c ?s1m) (<> ?s1m 0))))
  =>
   (retract ?node))
```

Controllo sul nodo generato: esiste già un tale nodo nei livelli più bassi?

```
(defrule CONSTRAINTS::circular-path
  (declare (auto-focus TRUE))
  (status (search-depth ?sd1)
          (boat-location ?bl)
          (shore-1-missionaries ?s1m)
          (shore-1-cannibals ?s1c)
          (shore-2-missionaries ?s2m)
          (shore-2-cannibals ?s2c))
  ?node <- (status (search-depth ?sd2&:(< ?sd1 ?sd2))
                   (boat-location ?bl)
                   (shore-1-missionaries ?s1m)
                   (shore-1-cannibals ?s1c)
                   (shore-2-missionaries ?s2m)
                   (shore-2-cannibals ?s2c))
  =>
  (retract ?node))
```

Cambiamento di stato: creazione del livello successivo

```
(defrule MAIN::shore-1-move
  ?node <- (status (search-depth ?num)
                   (boat-location shore-1)
                   (shore-1-missionaries ?s1m)
                   (shore-1-cannibals ?s1c)
                   (shore-2-missionaries ?s2m)
                   (shore-2-cannibals ?s2c))
  (boat-can-hold ?limit)
  =>
  (bind ?max-missionaries (min ?s1m ?limit))
  (loop-for-count (?missionaries 0 ?max-missionaries)
    (bind ?min-cannibals (max 0 (- 1 ?missionaries)))
    (bind ?max-cannibals (min ?s1c (- ?limit ?missionaries)))
    (loop-for-count (?cannibals ?min-cannibals ?max-cannibals)
      (duplicate ?node (search-depth =(+ 1 ?num))
                       (parent ?node)
                       (shore-1-missionaries (- ?s1m ?missionaries))
                       (shore-1-cannibals (- ?s1c ?cannibals))
                       (shore-2-missionaries (+ ?s2m ?missionaries))
                       (shore-2-cannibals (+ ?s2c ?cannibals))
                       (boat-location shore-2)
                       (last-move (move-string ?missionaries
                                               ?cannibals shore-2))))))
```

```
(deffunction MAIN::move-string (?missionaries ?cannibals ?shore)
  (switch ?missionaries
     (case 0 then
        (if (eq ?cannibals 1)
            then (format nil "Move 1 cannibal to %s.%n" ?shore)
            else (format nil "Move %d cannibals to %s.%n" ?cannibals
                                                          ?shore)))
     (case 1 then
        (switch ?cannibals
           (case 0 then
              (format nil "Move 1 missionary to %s.%n" ?shore))
           (case 1 then
              (format nil "Move 1 missionary and 1 cannibal to
                                                    %s.%n" ?shore))
           (default then
              (format nil "Move 1 missionary and %d cannibals to
                           %s.%n" ?cannibals ?shore))))
     (default
        (switch ?cannibals
           (case 0 then
              (format nil "Move %d missionaries to %s.%n"
                                       ?missionaries ?shore))
           (case 1 then
              (format nil "Move %d missionaries and 1 cannibal to
                           %s.%n" ?missionaries ?shore))
           (default then
              (format nil "Move %d missionary and %d cannibals to
                       %s.%n" ?missionaries ?cannibals ?shore)))))))
```

Riconoscere se si è in uno stato finale

```
(deftemplate SOLUTION::moves
  (slot id (type FACT-ADDRESS SYMBOL)
                                  (allowed-symbols no-parent))
   (multislot moves-list (type STRING)))

(defrule SOLUTION::recognize-solution
  (declare (auto-focus TRUE))
  ?node <- (status (parent ?parent)
  (shore-2-missionaries ?m&:(= ?m ?*initial-missionaries*))
  (shore-2-cannibals ?c&:(= ?c ?*initial-cannibals*))
  (last-move ?move))
  =>
  (retract ?node)
  (assert (moves (id ?parent) (moves-list ?move))))

(defrule SOLUTION::print-solution
  ?mv <- (moves (id no-parent) (moves-list "No move." $?m))
  =>
  (retract ?mv)
  (printout t t  "Solution found: " t t)
  (progn$ (?move ?m) (printout t ?move)))
```