Cannibals and Missionaries Problem

Another classic AI problem. The point is to get three cannibals and three missionaries across a stream with a boat that can only hold two people. If the cannibals outnumber the missionaries on either side of the stream, then the cannibals will eat the missionaries.

defmodule

```
(defmodule MAIN
  (export deftemplate status)
  (export defglobal initial-missionaries initial-cannibals))
```

The state or Frame

```
;;;*************
;;;* TEMPLATES *
;;;*************

;;; The status facts hold the state
;;; information of the search tree.

(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)))
```

Initial Conditions

```
;;;*****************
;;;* INITIAL STATE *
;;;*****************

(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))
```

Constraints: Number of Cannibals less than Missionaries

```
(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))
```

Same node exists of lower depth

```
(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))
```

Creating the next level

```
(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))))))
```

Recognizing that we are done

```
(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)))
```