Kuncheng Feng

CSC 416

Programming Challenge: M&C Interactive Problem Solver

Code:

```
; Original Code ----------------------------------------------
( defun mc ()
	( establish-world )
	( init-move-list )
	( make-moves )
)

( defun make-moves ()
	( display-world )
	( cond
		( ( goalp )
			( write-line "Good work!" )
			nil
		)
		( ( feast-state-p )
			( write-line "Yummy yummy yummy, I got Good in my tummy!!" )
			nil
		)
		( t
			( let ( m )
				( format t ">>> " ) ( setf m ( read ) )
				( if ( applicable-p m )
					( let () ( perform-move m ) ( make-moves ) )
					( let () ( write-line "Move inapplicable" ) nil )
				)
			)
		)
	)
)

( defun perform-move ( move )
	( setf *move-list* ( snoc move *move-list* ) )
	( if ( equal (current-bank) *left-bank* )
		( move-lr move )
		( move-rl move )
	)
)

( defun move-lr ( ml )
	( if ( null ml ) ( return-from move-lr ) )
	( move-lr-1 ( first ml ) )
```

```
        ( move-lr ( rest ml ) )
)

( defun move-rl ( ml )
        ( if ( null ml ) ( return-from move-rl ) )
        ( move-rl-1 ( first ml ) )
        ( move-rl ( rest ml ) )
)

; Added Code ------------------------------------------------
; Functions :-
;       snoc
;       establish-world
;       init-move-list
;       display-world
;       display-solution
;               print-list
;       goalp
;       feast-state-p
;               feast-bank
;       applicable-p
;               check-length
;               check-boat
;               check-content
;               check-exist
;                       check-member
;       move-lr-1
;       move-rl-1
;       current-bank

; Variables :-
;       *move-list*
;       *left-bank*
;       *right-bank*

; Copied from notes
( defun snoc ( o l )
        ( cond
                ( ( null l )
                        ( list o ))
                ( t
                        ( cons ( car l ) ( snoc o ( cdr l ) ) )
                )
        )
)


; Global variables
(setf *left-bank* '())
(setf *right-bank* '())
(setf *move-list* '())

; Functions called at the beginning --------------------------------
(defun establish-world ()
        (setf *left-bank* '(m m m c c c b))
        (setf *right-bank* '())
)

(defun init-move-list ()
        (setf *move-list* '())
)
; ------------------------------------------------------------------
```

```
; Functions that show the state of the world -------------------------
(defun display-world ()
	(format t "*left-bank*     ~A~%" *left-bank*)
	(format t "*right-bank*    ~A~%" *right-bank*)
)

(defun display-solution ()
	(print-list *move-list*)
)

(defun print-list (*list* &aux *first*)
	(setf *first* (car *list*))
	(if (equal *first* nil)
		(return-from print-list)
	)
	(format t "~A~%" *first*)
	(print-list (cdr *list*))
)

(defun current-bank ()
	(if (equal (find 'b *left-bank*) nil)
		(return-from current-bank *right-bank*)
		(return-from current-bank *left-bank*)
	)
)
; ---------------------------------------------------------------------


; Functions that check the state of the world --------------------------
(defun goalp ()
	(equal *left-bank* '())
)

(defun feast-state-p ()
	(or (feast-bank *left-bank*)
		(feast-bank *right-bank*)
	)
)

(defun feast-bank (*bank* &aux m c)
	(setf m (count 'm *bank*))
	(setf c (count 'c *bank*))
	(and (> m 0) (> c m))
)
; ---------------------------------------------------------------------


; Functions that check the validity of the input -----------------------
(defun applicable-p (*move*)
	(and
		(check-length *move*)
		(check-boat *move*)
		(check-content *move*)
		(check-exist *move*)
	)
)

; Min : boat and a person
; Max : boat and two people
; Therefore list length is either 2 or 3.
```

```
(defun check-length (*move*)
	(or
		(equal (length *move*) 2)
		(equal (length *move*) 3)
	)
)

; Boat must be contained in the input
(defun check-boat (*move*)
	(not (equal (find 'b *move*) nil))
)

; Input elements must either be either b, c, or m.
(defun check-content (*move* &aux *first*)
	(setf *first* (car *move*))
	(if (equal *first* nil)
		(return-from check-content t)
		(and
			(or
				(equal 'b *first*)
				(equal 'm *first*)
				(equal 'c *first*)
			)
			(check-content (cdr *move*))
		)
	)
)

; Can't move the things that don't exist
(defun check-exist (*move* &aux *temp-list*)
	(setf *temp-list* (current-bank))
	(check-member *move* *temp-list*)
)
; Extension of (check-exist)
(defun check-member (*things-to-be-moved* *bank* &aux thing)
	(setf thing (car *things-to-be-moved*))
	(if (equal thing nil)
		(return-from check-member t)
	)
	(if (not (equal (find thing *bank*) nil))
		(check-member (cdr *things-to-be-moved*) (remove thing *bank* :count 1))
		(return-from check-member nil)
	)
)
; ---------------------------------------------------------------------


; Functions that move elements between lists ----------------------------
(defun move-lr-1 (element)
	(setf *left-bank* (remove element *left-bank* :count 1))
	(setf *right-bank* (cons element *right-bank*))
)

(defun move-rl-1 (element)
	(setf *right-bank* (remove element *right-bank* :count 1))
	(setf *left-bank* (cons element *left-bank*))
)
; ---------------------------------------------------------------------
```

Demo

Success

```
[1]> (load "assignment_05.l")
;; Loading file assignment_05.l ...
;; Loaded file assignment_05.l
T
[2]> (mc)
*left-bank*     (M M M C C C B)
*right-bank*    NIL
>>> (b c c)
*left-bank*     (M M M C)
*right-bank*    (C C B)
>>> (b c)
*left-bank*     (C B M M M C)
*right-bank*    (C)
>>> (b c c)
*left-bank*     (M M M)
*right-bank*    (C C B C)
>>> (b c)
*left-bank*     (C B M M M)
*right-bank*    (C C)
>>> (b m m)
*left-bank*     (C M)
*right-bank*    (M M B C C)
>>> (b m c)
*left-bank*     (C M B C M)
*right-bank*    (M C)
>>> (b m m)
*left-bank*     (C C)
*right-bank*    (M M B M C)
>>> (b c)
*left-bank*     (C B C C)
*right-bank*    (M M M)
>>> (b c c)
*left-bank*     (C)
*right-bank*    (C C B M M M)
>>> (b c)
*left-bank*     (C B C)
*right-bank*    (C M M M)
>>> (b c c)
*left-bank*     NIL
*right-bank*    (C C B C M M M)
Good work!
NIL
```

Solution

```
[]> (display-solution)
(B C C)
(B C)
(B C C)
(B C)
(B M M)
(B M C)
(B M M)
(B C)
(B C C)
(B C)
(B C C)
NIL
```

Feastivity

```
[1]> (load "assignment_05.l")
;; Loading file assignment_05.l ...
;; Loaded file assignment_05.l
T
[2]> (mc)
*left-bank*     (M M M C C C B)
*right-bank*    NIL
>>> (b m)
*left-bank*     (M M C C C)
*right-bank*    (M B)
Yummy yummy yummy, I got Good in my tummy!!
NIL
```

Boat without operator

```
[]> (mc)
*left-bank*     (M M M C C C B)
*right-bank*    NIL
>>> (c c)
Move inapplicable
NIL
```

Boat with too many people

```
[]> (mc)
*left-bank*     (M M M C C C B)
*right-bank*    NIL
>>> (b c c c)
Move inapplicable
NIL
```

Boat not included

```
[]> (mc)
*left-bank*     (M M M C C C B)
*right-bank*    NIL
>>> (c c)
Move inapplicable
NIL
```

Boat with unknown elements

```
[]> (mc)
*left-bank*     (M M M C C C B)
*right-bank*    NIL
>>> (a b c)
Move inapplicable
NIL
```

Moving people that’s not there

```
[]> (mc)
*left-bank*     (M M M C C C B)
*right-bank*    NIL
>>> (b c c)
*left-bank*     (M M M C)
*right-bank*    (C C B)
>>> (b m m)
Move inapplicable
NIL
```