Assignment 4, Module Interface Specification

Jason Nagy

April 14, 2018

The following is a series of MISes to control the game board state for a game of Freecell.

Card ADT Module

Template Module

CardT

Uses

N/A

Syntax

Exported Types

SuitT={NAS, SPADES, CLUBS, HEARTS, DIAMONDS}
RankT={NAR, ACE, TWO, THREE, FOUR, FIVE, SIX, SEVEN, EIGHT, NINE, TEN, JACK, QUEEN, KING}
ColourT={NAC,RED,BLACK}
CardT=?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
CardT	SuitT, RankT	CardT	
getSuit		SuitT	
getRank		RankT	
isValid		$\mathbb{B}$	
getColour		ColourT	

Semantics

State Variables

s: SuitT
r: RankT

State Invariant

None

Assumptions

- The constructor CardT is called for each object instance before any other access routine is called for that object. The constuctor cannot be called on an existing object.

Access Routine Semantics

CardT(S, R):

- transition: $s, r := S, R$
- output: $out := self$
- exception: None

getSuit():

- output: $out := s$
- exception: None

getRank():

- output: $out := r$
- exception: None

isValid():

- output: $out := suit = \text{NAS} \wedge rank = \text{NAR}$
- exception: None

getColour():

- output: $out := r = (\text{SPADES} \vee \text{CLUBS}) \Rightarrow \text{BLACK} | r = (\text{HEARTS} \vee \text{DIAMONDS}) \Rightarrow \text{RED} | r = (\text{NAS}) \Rightarrow \text{NAC}$
- exception: None

Deck ADT Module

Template Module

DeckT

Uses

CardADT for CardT, SuitT, RankT

Syntax

Exported Types

DeckT=?

Exported Constants

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
DeckT		DeckT	
remCard			stack_empty
draw		CardT	stack_empty
shuffle			
size		$\mathbb{N}$	

Semantics

State Variables

d: sequence of CardT

State Invariant

- The deck will never have duplicate cards.
- The max amount of cards a DeckT may have is 52 cards where there is 4 sets of 13 cards (Ace to King for all four suits).

Assumptions

- The constructor DeckT is called for each object instance before any other access routine is called for that object. The constuctor cannot be called on an existing object.

Access Routine Semantics

DeckT():

- transition: $d := \forall(s : \text{SuitT} | s \in \text{SuitT} : \forall(r : \text{RankT} | r \in \text{RankT} : \text{append}(d, \text{CardT}(s, r)))$
- output: $out := self$
- exception: None

draw():

- output : $out := \exists(c : \text{CardT} | c \in d : c)$
- exception: $(|s| = 0 \Rightarrow \text{is\_empty})$

remCard():

- transition: $s := s \backslash draw()$
- exception: $(|s| = 0 \Rightarrow \text{is\_empty})$

shuffle():

- transition: $d := \forall(s : \text{SuitT} | s \in \text{SuitT} : \forall(r : \text{RankT} | r \in \text{RankT} : \text{append}(d, \text{CardT}(s, r)))$
- exception: None

size():

- output $out := |d|$
- exception: None

Local Functions

append: seq of CardT × CardT ⇒ seq of CardT
transition: $S := S || C$

Stack ADT Module

Template Module

Stack

Uses

CardADT for CardT

Syntax

Exported Types

StackT=?

Exported Constants

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
StackT			
addCard	CardT		
remCard		CardT	is_empty
peek		CardT	is_empty
size		$\mathbb{N}$	

Semantics

State Variables

c: seq of CardT

State Invariant

None

Assumptions

- The constructor StackT is called for each object instance before any other access routine is called for that object. The constuctor cannot be called on an existing object.
- StackT can be considered empty when it is of length 0 or the StackT is of length 1 and peek() returns a CardT with getSuit()=NAS and getRank()=NAR.

Access Routine Semantics

StackT():

- transition: $c := \{\}$
- output: $out := self$
- exception: None

addCard(C):

- transition: $c := c || C$
- exception: None

remCard():

- transition: $c := c[1 : |c| - 1]$
- exception: $(|c| = 0 \Rightarrow \text{is\_empty})$

peek():

- output: $out := c[0]$
- exception: $(|c| = 0 \Rightarrow \text{is\_empty})$

size():

- output: $out := |c|$
- exception: None

Board ADT Module

Template Module

Board

Uses

CardADT for CardT, SuitT, RankT
DeckADT for DeckT
StackADT for StackT

Syntax

Exported Types

BoardT=?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
BoardT		BoardT	
hasWon		$\mathbb{B}$	
getStack	$\mathbb{N}$	StackT	invalid_index
getFree	$\mathbb{N}$	CardT	invalid_index
getWin	$\mathbb{N}$	CardT	invalid_index
setStack	$\mathbb{N}$, StackT		invalid_index
setFree	$\mathbb{N}$, CardT		invalid_index
setWin	$\mathbb{N}$, CardT		invalid_index
moveColToCol	$\mathbb{N}, \mathbb{N}$		invalid_index, stack_empty, not_alternating_colour, not_decending_rank
moveColToFree	$\mathbb{N}, \mathbb{N}$		invalid_index, stack_empty, occupied_cell
moveFreeToCol	$\mathbb{N}, \mathbb{N}$		invalid_index, is_empty, unoccupied_cell, not_alternating_colour, not_decending_rank
moveColToWin	$\mathbb{N}, \mathbb{N}$		invalid_index, is_empty, not_same_suit, not_ascending_rank
moveFreeToWin	$\mathbb{N}, \mathbb{N}$		invalid_index, unoccupied_cell, not_same_suit, not_ascending_rank
isValidMoves		$\mathbb{B}$	

Semantics

State Variables

col: sequence of StackT
fre: sequence of CardT
fou: sequence of CardT
dek: DeckT

State Invariant

- All StackTs within *col* must have a CardT with getSuit()=NAS and getRank()=NAR at the bottom (first added on).

Assumptions

- The constructor BoardT is called for each object instance before any other access routine is called for that object. The constuctor cannot be called on an existing object.

- Unallocated fre locations are to be filled with a CardT with getSuit()=NAS and getRank()=NAR.

Access Routine Semantics

BoardT():

- transition: $col := \forall(c : \text{CardT} | c \in dek : col || c)$

  $fre :=$ seq of CardT

  $fou :=$ seq of CardT

  $dek :=$ DeckT()
- output: $out := self$
- exception: None

hasWon():

- output: $out := \text{BoardEmpty}(col) \wedge \forall(c : \text{CardT} | c \in fre : \text{FreeCellEmpty}(c)) \wedge forall(C : \text{CardT} | C \in fou : \text{FoundationComplete}(C))$
- exception: None

getStack(i):

- output: $out := col[i]$
- exception: $(\neg(0 <= i < 8) \Rightarrow \text{invalid\_index})$

getFree(i):

- output: $out := fre[i]$
- exception: $(\neg(0 <= i < 4) \Rightarrow \text{invalid\_index})$

getWin(i):

- output: $out := fou[i]$
- exception: $(\neg(0 <= i < 4) \Rightarrow \text{invalid\_index})$

setStack(i,S):

- transition: $col[i] = S$
- exception: $(\neg(0 <= i < 8) \Rightarrow$ invalid_index)

getFree(i,C):

- transition: $fre[i] = C$
- exception: $(\neg(0 <= i < 4) \Rightarrow$ invalid_index)

getWin(i,C):

- transition: $fou[i] = C$
- exception: $(\neg(0 <= i < 4) \Rightarrow$ invalid_index)

moveColToCol(a,b):

- transition: $col[a], col[b] :=$ $col[a]$.remCard(), $col[b]$.addCard($col[a]$.peek())
- exception: ((¬ValidIndex($8, 8, a, b$) $\Rightarrow$ invalid_index)$\vee$(StackEmpty($col[a]$) $\Rightarrow$ stack_empty)$\vee$ (¬AlternatingColour($col[a]$.peek(), $col[b]$.peek()) $\Rightarrow$ not_alternating_colour)$\vee$(¬DecreasingRank($col[a$ not_decreasing_rank))

moveColToFree(a,b):

- transition: $col[a], fre[b] := col[a].remCard(), fre[b] = col[a].peek()$
- exception: ((¬ValidIndex($8, 4, a, b$) $\Rightarrow$ invalid_index)$\vee$(StackEmpty($col[a]$) $\Rightarrow$ stack_empty)$\vee$ (¬CellFree(b) $\Rightarrow$ occupied_cell))

moveFreeToCol(a,b):

- transition: $fre[a], col[b] := fre[a] =$ CardT(NAS,NAR), $col[a]$.addCard($fre[a]$)
- exception: ((¬ValidIndex($4, 8, a, b$) $\Rightarrow$ invalid_index)$\vee$(StackEmpty($col[b]$) $\Rightarrow$ stack_empty)$\vee$ (CellFree(a) $\Rightarrow$ occupied_cell))$\vee$(¬AlternatingColour($fre[a], col[b]$.peek()) $\Rightarrow$ not_alternating_colour) (¬DecreasingRank($fre[a], col[b]$.peek()) $\Rightarrow$ not_decreasing_rank))

moveColToWin(a,b):

- transition: $col[a], fou[b] :=$ $col[a]$.remCard(), $fou[b] = col[a]$.peek()
- exception: ((¬ValidIndex($8, 4, a, b$) $\Rightarrow$ invalid_index)$\vee$(StackEmpty($col[a]$) $\Rightarrow$ stack_empty)$\vee$ (¬SameSuit($col[a]$.peek(), $fou[b]$) $\Rightarrow$ not_same_suit)$\vee$(¬IncreasingRank($fou[b], col[a]$.peek()) $\Rightarrow$ not_ascending_rank)

moveFreeToWin(a,b):

- transition: $fre[a], fou[b] := fre[a] = \text{CardT(NAS,NAR)}, fou[b] = col[a].\text{peek()}$
- exception: $((\neg\text{ValidIndex}(4, 4, a, b) \Rightarrow \text{invalid\_index}) \vee (\text{CellFree}(a) \Rightarrow \text{occupied\_cell})) \vee (\neg\text{SameSuit}(fre[a], fou[b]) \Rightarrow \text{not\_same\_suit}) \vee (\neg\text{IncreasingRank}(fou[b], fre[a] \Rightarrow \text{not\_ascending\_rank})$

isValidMoves():

- output $out := \exists(s : \text{StackT}|s \in col : \exists(c : \text{CardT}|c \in fou : \text{isIncreasingRank}(c, s.\text{peek}()) \wedge \text{SameSuit}(c, s.\text{peek}()))) \vee \exists(c_1 : \text{CardT}|c_1 \in fre : \exists(c_2 : \text{CardT}|c_2 \in fou : \text{isIncreasingRank}(c_2, c_1) \wedge \text{SameSuit}(c_1, c_2)))$
  $\vee\exists(s_1 : \text{StackT}|s_1 \in col : \exists(s_2 : \text{StackT}|s_2 \in col : s_1 \neq s_2 \wedge (isIncreasingRank(s_1.peek(), s_2.peek()) \vee isDecreasingRank(s_1.peek(), s_2.peek())) \wedge iAlternatingRank(S_1.peek(), s_2.peek()) \wedge \neg isStackEmpty(s_1) \wedge \neg isStackEmpty(s_2)))$
  $\vee\exists(c_1 : \text{CardT}|c_1 \in fre : \exists(s_1 : \text{StackT}|s_1 \in col : (\text{AlternatingColour}(c_1, s_1.\text{peek}) \wedge (\text{IncreasingRank}(c_1, s_1.\text{peek}) \vee \text{DecreasingRank}(c_1, s_1.\text{peek})) \wedge c_1.\text{isValid}()) \vee (\neg c_1.\text{isValid}) \wedge \neg\text{isStackEmpty}(s_1)))$

- exception: None

Local Functions

ValidIndex: $\mathbb{N}_1 \times \mathbb{N}_2 \times \mathbb{N}_3 \times \mathbb{N}_4 \rightarrow \mathbb{B}$
output: $out := (0 <= \mathbb{N}_3 < \mathbb{N}_1) \wedge (0 <= \mathbb{N}_4 < \mathbb{N}_2)$

AlternatingColour: $\text{CardT}_1 \times \text{CardT}_2 \rightarrow \mathbb{B}$
output: $out := (\text{CardT}_1.\text{getColour}() = \text{RED} \wedge \text{CardT}_2.\text{getColour}() = \text{BLACK}) \vee (\text{CardT}_1.\text{getColour}() = \text{BLACK} \wedge \text{CardT}_2.\text{getColour}() = \text{RED})$
IncreasingRank: $\text{CardT}_1 \times \text{CardT}_2 \rightarrow \mathbb{B}$
output: $out := \text{CardT}_1.\text{getRank}() = \text{CardT}_2.\text{getRank}() - 1$

DecreasingRank: $\text{CardT} \times \text{CardT} \rightarrow \mathbb{B}$
output: $out := \text{CardT}_1.\text{getRank}() = \text{CardT}_2.\text{getRank}() + 1$

StackEmpty: $\text{StackT} \rightarrow \mathbb{B}$
output: $out := \text{StackT.size}() = 0 \vee (\text{StackT.size}() = 1 \wedge \neg\text{StackT.peek.isValid}())$

CellFree: $\mathbb{N} \times$ seq of CardT $\rightarrow \mathbb{B}$
output: $out := \neg$(seq of CardT)[$\mathbb{N}$].isValid

SameSuit: CardT$_1 \times$ CardT$_2 \rightarrow \mathbb{B}$
output: $out :=$ CardT$_1$.getSuit() = CardT$_2$.getSuit()

BoardEmpty: seq of StackT $\rightarrow \mathbb{B}$
output: $out := \forall(s :$ StackT$|s \in$ (seq of StackT : StackEmpty(s))

FreeCellEmpty: seq of CardT $\rightarrow \mathbb{B}$
output: $out := \forall(c :$ CardT$|c \in$ seq of CardT : $\neg c$.isValid())

FoundationComplete: seq of CardT $\rightarrow \mathbb{B}$
output: $out := \forall(c :$ CardT$|c \in$ seq of CardT : c.getRank() = KING) $\wedge\, \forall(s_1 :$ CardT$|s_1 \in$ seq of CardT : s_1.getSuit $\neq$ NAS $\wedge\, \forall(s_2 :$ CardT$|s_2 \in$ seq of CardT$\backslash s_1$: s_1.getSuit() $\neq$ s_2.getSuit()))