Assignment 4, Part 1, Specification

Shengchen Zhou,400050783

April 9, 2018

This document is the MIS for Assignment 4 which implements a model of FreeCell poker card game.

Poker Card Meta Types Module

Module

PokerCardMetaTypes

Uses

N/A

Syntax

Exported Constants

None

Exported Types

RankT = {Ace, Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten, Jack, Queen, King}
SuitT = {Diamonds, Clubs, Hearts, Spades}
ColrT = {Red, Black}

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

Poker Card Module

Module

PokerCard

Uses

PokerCardMetaTypes

Syntax

Exported Types

PokerCard = ?

Exported Access Programs

Routine name	In	Out	Exceptions
PokerCard	RankT, SuitT	PokerCard	
r		RankT	
s		SuitT	
c		ColrT	
operator ==	PokerCard, PokerCard	$\mathbb{B}$	

Semantics

State Variables

R: RankT
S: SuitT

State Invariant

None

Assumptions

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

Access Routine Semantics

PokerCard(r,s):

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

r():

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

s():

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

c():

- output: $out := Red(if s() == Diamonds||Hearts), Black(if s() == Clubs||Spades)$
- exception: None

operator ==:

- output: $out := this.S == c.S \;\&\&\; this.R == c.R$
- exception: None

Poker Card Sequence Module

Module

PokerCardSeq

Uses

PokerCard

Syntax

Exported Types

PokerCardSeq = ?

Exported Access Programs

Routine name	In	Out	Exceptions
PokerCardSeq		PokerCardSeq	
init	PokerCard		
lastCard		PokerCard	GetCardOp_Illegal
card	$\mathbb{N}$	PokerCard	GetCardOp_Illegal
addCard	PokerCard		
removeLastCard			RemoveCardOp_Illegal
size		$\mathbb{N}$	

Semantics

State Variables

S: array of PokerCards

State Invariant

None

Assumptions

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

Access Routine Semantics

PokerCardSeq():

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

init(s):

- transition: $S = s$
- exception: None

lastCard():

- output: $out := S.back()$
- exception: S.empty ⇒ GetCardOp_Illegal

card(i):

- output: $out := S[i]$
- exception: i+1¿size() ⇒ GetCardOp_Illegal

addCard(c):

- transition: $S.push\_back(c)$
- exception: None

removeLastCard():

- transition: $PokerCardSeq = S.pop\_back$
- exception: S.empty ⇒ RemoveCardOp_Illegal

size():

- output: $out := S.size()$
- exception: None

Poker Card Foundation Module

Module

PokerCardFoundation

Uses

PokerCard, PokerCardSeq

Syntax

Exported Types

PokerCardFoundation = ?

Exported Access Programs

Routine name	In	Out	Exceptions
PokerCardFoudation		PokerCardFoudation	
canAppendCard	PokerCard	$\mathbb{B}$	
appendCard	PokerCard		AddCardOp_Illegal
card	$\mathbb{N}$	PokerCard	GetCardOp_Illegal
removeCard			GetCardOp_Illegal
size		$\mathbb{N}$	
canMoveCard2	$PokerCardFreeCell\|\|$ $PokerCardPile\|\|$ $PokerCardFoundation$	$\mathbb{B}$	
moveCard	PokerCardFreeCell $PokerCardPile\|\|$ $PokerCardFoundation$	$\mathbb{B}$	

Semantics

State Variables

S: sequence of pokercards

State Invariant

None

Assumptions

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

Access Routine Semantics

PokerCardFoundation():

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

canAppendCard(c):

- output: $out := \mathbb{B}$
- exception: None

appendCard(c):

- transition: $S.addCard(c)$
- exception: !canAppendCard(c) $\Rightarrow$ AddCardOp_Illegal

card(i):

- output: $out := S.card(i)$
- exception: i+1¿size() $\Rightarrow$ GetCardOp_Illegal

removeCard():

- transition $S.removeLastCard()$
- exception: S.size()==0 $\Rightarrow$ RemoveCardOp_Illegal

size():

- output: $out := S.size()$
- exception: None

canMoveCard2(c):

- output: $out := S.size() != 0 \&\& canAppendCard(C)$

- determine if able to move a PokerCard to a freecell or pile or foundation
- exception: None

moveCard2(c):

- transition: $C = S.lastCard()$
- output: $out := \mathbb{B}$
- move a pokecard to a freecell or pile or foundation
- exception: None

Poker Card Pile Module

Module

PokerCardPile

Uses

PokerCard, PokerCardSeq

Syntax

Exported Types

PokerCardPile = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
PokerCardPile		PokerCardPile	
init	PokerCardSeq		
card	$\mathbb{N}$	PokerCard	GetCardOp_Illegal
canAppendCard	PokerCard	$\mathbb{B}$	
appendCard	PokerCard		AddCardOp_Illegal
removeCard			RemoveCardOp_Illegal
size		$\mathbb{N}$	
canMoveCard2	$PokerCardFreeCell\|\|$ $PokerCardPile\|\|$ $PokerCardFoundation$	$\mathbb{B}$	
moveCard	PokerCardFreeCell $PokerCardPile\|\|$ $PokerCardFoundation$	$\mathbb{B}$	

Semantics

State Variables

S: seq of pokercards

State Invariant

None

Assumptions

- The PokerCardPile constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once.

Access Routine Semantics

PokerCardPile():

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

init(s):

- transition: $S.init(s)$
- exception: None

card(i):

- output: $out := S.card(i)$
- exception: i+1¿size() $\Rightarrow$ GetCardOp_Illegal

canAppendCard(c):

- output: $out := \mathbb{B}$
- exception: None

appendCard(c):

- transition: $S.addCard(c)$

- exception: !canAppendCard(c) ⇒ AddCardOp_Illegal

removeCard():

- transition $S.removeLastCard()$
- exception: S.size()==0 ⇒ RemoveCardOp_Illegal

size():

- output: $out := S.size()$
- exception: None

canMoveCard2(c):

- output: $out := S.size()! = 0 \&\& canAppendCard(C)$
- exception: None

moveCard2(c):

- transition: $C = S.lastCard()$
- output: $out := S.size()! = 0 \&\& canAppendCard(C)$
- exception: None

Poker Card Free Cell Module

Module

PokerCardFreeCell

Uses

PokerCard, PokerCardSeq

Syntax

Exported Types

PokerCardFreeCell = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
PokerCardFreeCell			
canAppendCard	PokerCard	$\mathbb{B}$	
appendCard	PokerCard		AddCardOp_Illegal
card	$\mathbb{N}$	PokerCard	GetCardOp_Illegal
removeCard			RemoveCardOp_Illegal
size		$\mathbb{N}$	
canMoveCard2	$PokerCardFreeCell\|\|$ $PokerCardPile\|\|$ $PokerCardFoundation$	$\mathbb{B}$	
moveCard	PokerCardFreeCell $PokerCardPile\|\|$ $PokerCardFoundation$	$\mathbb{B}$	

Semantics

State Variables

S: seq of pokercards

State Invariant

None

Assumptions

- PokerCardFreeCell constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once.

Access Routine Semantics

PokerCardFreeCell():

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

canAppendCard(c):

- output: $out := S.size() ! = 0 \Rightarrow false$
- exception: None

appendCard(c):

- transition: $S.addCard(c)$
- exception: !canAppendCard(c) $\Rightarrow$ AddCardOp_Illegal

card(i):

- output: $out := S.card(0)$
- exception: none

removeCard():

- transition $S.removeLastCard()$

- exception: S.size()==0 ⇒ RemoveCardOp_Illegal

size():

- output: $out := S.size()$
- exception: None

canMoveCard2(c):

- output: $out := S.size() != 0 \&\& canAppendCard(C)$
- determine if able to move a PokerCard to a freecell or pile or foundation
- exception: None

moveCard2(c):

- transition: $C = S.lastCard()$
- output: $out := \mathbb{B}$
- move a pokecard to a freecell or pile or foundation
- exception: None

Game Deck Module

Module

GameDeck

Uses

PokerCardFoudation, PokerCardPile, PokerCardFreeCell

Syntax

Exported Types

GameDeck = ?

Exported Constants

None

Exported Access Programs

Routine name	In	Out	Exceptions
GameDeck	PokerCardFreeCell PokerCardPile PokerCardFoundation	GameDeck	
init			
cell	$\mathbb{N}$	PokerCardFreeCell	Ind_Illegal
pile	$\mathbb{N}$	PokerCardPile	Ind_Illegal
foundation	$\mathbb{N}$	PokerCardFoundation	Ind_Illegal
canFreeCell2FreeCell	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
freeCell2FreeCell	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
canPile2Pile	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
pile2Pile	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
canFoundation2Foundation	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
foundation2Foundation	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
canFreeCell2Pile	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
freeCell2Pile	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
canPile2FreeCell	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
pile2FreeCell	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
canFreeCell2Foundation	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
freeCell2Foundation	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
canPile2Foundation	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
Pile2Foundation	$\mathbb{N}$, $\mathbb{N}$	$\mathbb{B}$	Ind_Illegal
validMoves		$\mathbb{B}$	
winning		$\mathbb{B}$	

Semantics

State Variables

C: PokerCardFreeCell
P: PokerCardPile
F: PokerCardFoundation

State Invariant

None

Assumptions

- The init() can only be called once.

Access Routine Semantics

GameDeck(c,p,f):

- transition: $C[i] = c[i], P[i] = p[i], F[i] = f[i]$
- exception: None

init():

- transition:
- exception: None

cell(i)

- output: $out := C[i]$
- exception: i+1¿NC $\Rightarrow$ Ind_Illegal

pile(i):

- output: $out := P[i]$
- exception: i+1¿NP $\Rightarrow$ Ind_Illegal

foundation(i):

- output: $out := F[i]$
- exception: i+1¿NP $\Rightarrow$ Ind_Illegal

canFreeCell2FreeCell(i1,i2):

- output: $out := this \rightarrow C[i1].canMoveCard2(\&this \rightarrow C[i2])$
- exception: $i1 + 1 > NC || i2 + 1 > NC \Rightarrow Ind\_Illegal$

FreeCell2FreeCell(i1,i2):

- output: output: $out := this \rightarrow C[i1].moveCard2(\&this \rightarrow C[i2])$
- exception: $i1 + 1 > NC || i2 + 1 > NC \Rightarrow Ind\_Illegal$

canPile2Pile(i1,i2):

- output: output: $out := this \rightarrow P[i1].canMoveCard2(\&this \rightarrow P[i2])$
- exception: $i1 + 1 > NP \| i2 + 1 > NP \Rightarrow Ind\_Illegal$

pile2Pile(i1,i2):

- output: $out := this \rightarrow P[i1].moveCard2(\&this \rightarrow P[i2])$
- exception: $i1 + 1 > NP \| i2 + 1 > NP \Rightarrow Ind\_Illegal$

canFreeCell2Pile(i1,i2):

- output: $out := this \rightarrow C[i1].canMoveCard2(\&this \rightarrow P[i2])$
- exception: $i1 + 1 > NC \| i2 + 1 > NP \Rightarrow Ind\_Illegal$

FreeCell2Pile(i1,i2):

- output: $out := this \rightarrow C[i1].moveCard2(\&this \rightarrow P[i2])$
- exception: $i1 + 1 > NC \| i2 + 1 > NP \Rightarrow Ind\_Illegal$

canPile2FreeCell(i1,i2):

- output: $out := this \rightarrow P[i1].canMoveCard2(\&this \rightarrow C[i2])$
- exception: $i1 + 1 > NP \| i2 + 1 > NC \Rightarrow Ind\_Illegal$

pile2FreeCell(i1,i2):

- output: $out := this \rightarrow P[i1].moveCard2(\&this \rightarrow C[i2])$
- exception: $i1 + 1 > NP \| i2 + 1 > NC \Rightarrow Ind\_Illegal$

canFreeCell2Foundation(i1,i2)

- output: $out := this \rightarrow C[i1].canMoveCard2(\&this \rightarrow F[i2])$
- exception: $i1 + 1 > NC \| i2 + 1 > NF \Rightarrow Ind\_Illegal$

freeCell2Foundation(i1,i2):

- output: $out := this \rightarrow C[i1].moveCard2(\&this \rightarrow F[i2])$
- exception: $i1 + 1 > NC \| i2 + 1 > NF \Rightarrow Ind\_Illegal$

canPile2Foundation(i1,i2):

- output: $out := this \rightarrow P[i1].canMoveCard2(\&this \rightarrow F[i2])$
- exception: $i1 + 1 > NP \| i2 + 1 > NF \Rightarrow Ind\_Illegal$

pile2Foundation(i1,i2):

- output: $out := this \rightarrow P[i1].moveCard2(\&this \rightarrow F[i2])$
- exception: $i1 + 1 > NP \| i2 + 1 > NF \Rightarrow Ind\_Illegal$

validMoves():

- output: $out := \mathbb{B}$
- exception: None

Winning()

- output: $out := \mathbb{B}$
- exception: None

Exceptions

Ind_Illegal
GetCardOp_Illegal
RemoveCardOp_Illegal
AddCardOp_Illegal