Assignment 4, Specification

Dominik Buszowiecki

April 5, 2019

Game Board ADT Module

Template Module

BoardT

Uses

None

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
new BoardT	Grid	BoardT	invalid_argument
nextStage		BoardT	
toGrid		Grid	

Semantics

State Variables

S : Grid *# 2D array*

State Invariant

$\forall e \in S(|S[0]| = |e|)$ *# All elements in S are sequences of the same size*

Assumptions & Design Decisions

- The BoardT constructor is called before any other access routine is called on that instance. Once a BoardT has been created, the constructor will not be called on it again.

Access Routine Semantics

BoardT(G):

- transition: $S := G$
- output: $out := self$
- exception: $exc := (|G| = 0 \vee \neg(\forall e \in G : |G[0]| = |e|) \Rightarrow \text{invalid\_argument})$
  # Exception checks state invariant

nextStage():

- transition: $S := G$ such that $(\forall i, j : \mathbb{N} | \text{IsInBounds}(S, i, j) : G[i][j] = \text{updateCell}(S, i, j))$
- output: $out := self$
- exception: none

toGrid():

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

Local Types

Grid = seq of (seq of $\mathbb{B}$)

Local Functions

UpdateCell : Grid $\times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$
UpdateCell$(S, x, y) \equiv$

$S[x][y] =$ True	NumAdj$(S, x, y) < 2$	False
	NumAdj$(S, x, y) = 2$	True
	NumAdj$(S, x, y) = 3$	True
	NumAdj$(S, x, y) > 3$	False
$S[x][y] =$ False		NumAdj$(S, x, y) = 3$

NumAdj : Grid $\times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$
NumAdj$(S, x, y) \equiv$
$+(i, j : \mathbb{N} | i \in \{x+1, x-1\} \wedge j \in \{y+1, y-1\} \wedge$ IsInBounds$(S, i, j) \wedge S[i][j] =$ True $: 1)$
 # Assumed that when x or y is 0, x-1 or y-1 does not return anything, i or j can only become {x+1} or {y+1}

IsInBounds : Grid $\times \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$
IsInBounds$(S, x, y) \equiv x < |S| \wedge y < |S[0]|$

View Module

Module

View

Uses

GameBoardT

Syntax

Exported Access Programs

Routine name	In	Out	Exceptions
readStage	s : string		file_not_found
initStage	$\mathbb{N}, \mathbb{N}$		out_of_range
simulate	$\mathbb{N}$		
writeStage	s : string		

Semantics

State Variables

gameBoard : GameBoardT *# 2D array*

Environment Variables

gridFile: File containing a grid representation of the game

State Invariant

None

Assumptions & Design Decisions

- The input file will match the given specification, each line will have equal number of characters
- The user will either call readStage or initStage before simulating or witting the board

Access Routine Semantics

readStage(s):

- transition: $gameBoard$:= GameBoard(G) where G is a seq of(seq of $\mathbb{B}$)

  G is generated from the file gridFile associated with the string s. It is generated with the following condition:

  $$(\forall i : \mathbb{N} | i < |L| : G[i] = \text{stringToRow}(L))$$

  Where L is a seq of string, each string in L corresponds to a line in file s. Therefore, $L[2][4]$ would represent the 5th character in the 3rd row. An example of file s is provided below:

  ```
  ______0______0__
  ___0________0____
  _____0___0_______
  ```

  Each "0" corresponds to a populated cell, where a "_" is an empty cell (note: an empty cell can be represented by anything but 0 when the file is read)

initStage(x, y):

- transition: $gameBoard := G$ such that
  $(|G.\text{toGrid}()| = x \wedge |G.\text{toGrid}()[0]| = y \wedge (\forall i, j : \mathbb{N} | i < x \wedge j < y : G.\text{toGrid}()[i][j] = False)$

- exception: $exc := (x \leq 0 \vee y \leq 0 \Rightarrow \text{invalid\_argument})$

simulate(n):

- transition: *# Procedural Specification*
  printStage()
  for all i in $[0..n]$:
  $gameBoard = gameBoard.\text{nextStage}()$
  printStage()

- exception: $exc := (x \leq 0 \vee y \leq 0 \Rightarrow \text{invalid\_argument})$

writeStage(s):

- transition: Writes G into the file with name s. G can be represented be the following.

  $(\forall i, j : \mathbb{N} | i < |L| \wedge j < |L[0]| : L[i][j] = \text{"\_"} \Rightarrow G[i][j] = \text{False} | L[i][j] = \text{"0"} \Rightarrow G[i][j] = \text{True})$

  Where L is a seq of string, each string in L corresponds to a line in file s.

Local Functions

stringToRow : string $\rightarrow$ seq of $\mathbb{B}$
stringToRow(s) $\equiv$ L such that $(\forall i : \mathbb{N} | i < |s| : L[i] = (S[i] = "0"))$

printStage :
printStage() $\equiv$
Displays s in terminal where s is:
s = $(+i : \mathbb{N} | i < |gameBoard.\text{toGrid}()| : \text{rowToString}(gameBoard.\text{toGrid}()[i]) + "\backslash n")$

rowToString : seq of $\mathbb{B} \rightarrow$ string
stringToRow(s) $\equiv$ L such that $(\forall i : \mathbb{N} | i < |s| : s[i] \Rightarrow L[i] = "[\#]" \wedge \neg s[i] \Rightarrow L[i] = "[\ ]")$

Critique of Design

Write a critique of the interface for the modules in this project. Is there anything missing? Is there anything you would consider changing? Why?

Potential discussion points:

- The stack module provides a toSeq module that violates essentiality. To address this, another module could be built to provide the toSeq service through a function that takes a stack as input and return a sequence.