More Abstract Data Types

Bounded stacks

interface BStackI$\langle T :: \text{type}, \mathit{capacity} : \text{int}\rangle$
 ghost public readonly var $s : \text{Seq}\,\langle T\rangle := [\,]$

 invariant $\mathit{length}(s) \le \mathit{capacity}$

 public method *size*() : int
  postcondition $\mathit{result} = \mathit{length}(s)$

 public method *capacity*() : int
  postcondition $\mathit{result} = \mathit{capacity}$

 public method *push*($x : T$)
  precondition $\mathit{length}(s) < \mathit{capacity}$
  changes s
  postcondition $s = [x]\hat{}\, s_0$

 public method *pop*( )
  precondition $\mathit{length}(s) > 0$
  changes s
  postcondition $s = s_0[1, ..\mathit{length}(s_0)]$

 public method *top*() : T
  precondition $\mathit{length}(s) > 0$
  postcondition $\mathit{result} = s(0)$
end BStackI

A possible implementation

var a : array$\langle T\rangle$

var p : int

With linking invariant

$$p = \mathit{length}(s) \le \mathit{capacity} = a.\mathrm{length}$$
$$\wedge\ \forall i \in \{0, ..p\} \cdot s(i) = a(p - 1 - i)$$

In detail

class BStack$\langle T :: \mathrm{type}, \mathit{capacity} :\ \mathrm{int}\rangle$

implements BStackI$\langle T :: \mathrm{type}, \mathit{capacity} :\ \mathrm{int}\rangle$

// ghost public readonly var s : $\mathrm{Seq}\ \langle T\rangle := [\,]$

// top is a the left of s

private var a : array$\langle T\rangle$

private var p : int

invariant $p = \mathit{length}(s) \le \mathit{capacity} = a.\mathrm{length}$

invariant $\forall i \in \{0, ..p\} \cdot s(i) = a(p - 1 - i)$

public method *size*() : int

postcondition $\mathit{result} = \mathit{length}(s)$

return p

end *size*

public method *capacity*() : int

postcondition $\mathit{result} = \mathit{capacity}$

return *capacity* ;

end *capacity*

public method *push*($x : T$)
  precondition $length(s) < capacity$
  changes s
  postcondition $s = [x]\hat{\ } s_0$
  $a(p) := x \quad p := p + 1$
  ghost $s := [x]\hat{\ } s$
end *push*

public method *pop*( )
  precondition $length(s) > 0$
  changes s
  postcondition $s = s_0[1, ..length(s)]$
  $p := p - 1$
  ghost $s := s[1, ..length(s)]$
end *pop*

public method *top*() : T
  precondition $length(s) > 0$
  postcondition $result = s(0)$
  return $a(p - 1)$
end *top*

end BStack

Bounded queue

interface BQueueI$\langle T$ :: type, $capacity$: int$\rangle$

  ghost public readonly var s : Seq $\langle T\rangle$:= [ ]

  invariant $length(s) \leq capacity$

  public method *size*() : int

    postcondition $result = length(s)$

  public method *capacity*() : int

    postcondition $result = capacity$

  public method *add*($x : T$)

    precondition $length(s) < capacity$

    changes s

    postcondition $s = s_0\hat{}[x]$

  public method *remove*( )

    precondition $length(s) > 0$

    changes s

    postcondition $s = s_0[1, ..length(s_0)]$

  public method *next*() : T

    precondition $length(s) > 0$

    postcondition $result = s(0)$

end BQueueI

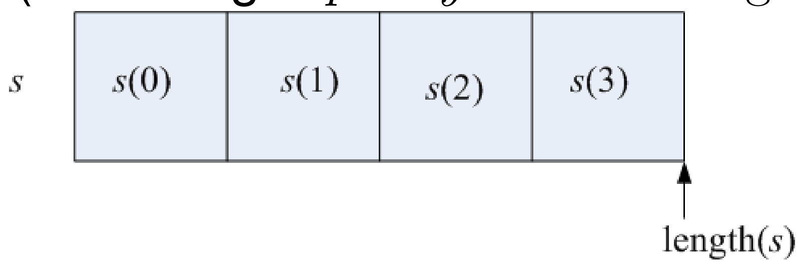
A possible implementation

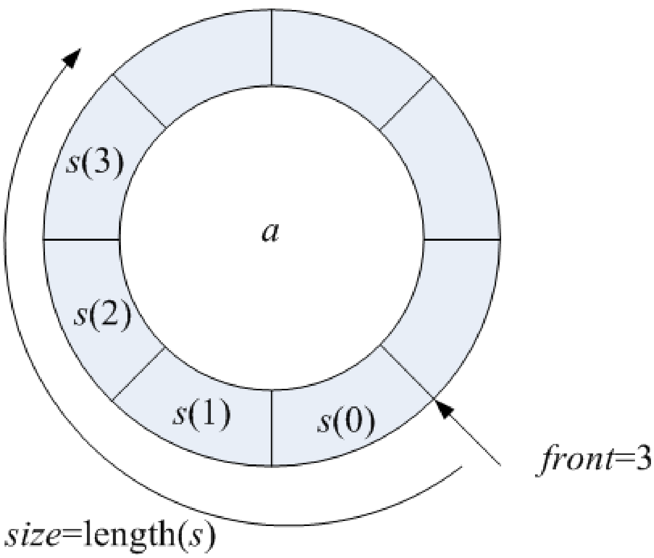
var a : array$\langle T\rangle$
var $front : int$
var $size : int$

With linking invariant

$$size = \text{length}(s) \le a.\text{length} = \mathit{capacity}$$
$$\wedge\ \mathit{front} \in \{0, ..\mathit{capacity}\}$$
$$\wedge\ \forall i \in \{0, ..\mathit{size}\} \cdot s(i) = a((\mathit{front} + i) \bmod \mathit{capacity})$$

In a picture (assuming $\mathit{capacity} = 8$ and $\text{length}(s) = 4$)





Note that even if $\text{length}(s) = \mathit{capacity}$ there are still $\mathit{capacity}$ different ways of representing each value in the abstract space.

Mutable bounded sets

interface MutSetI$\langle T$ :: type, *capacity* : int$\rangle$
  ghost public readonly var s : Set $\langle T \rangle := \emptyset$

  invariant $|s| \leq \mathit{capacity}$

  public method *size*() : int
    postcondition $\mathit{result} = |s|$

  public method *capacity*() : int
    postcondition $\mathit{result} = \mathit{capacity}$

  public method *add*($x : T$)
    precondition $|s| < \mathit{capacity} \lor x \in s$
    changes s
    postcondition $s = s_0 \cup \{x\}$

  public method *remove*($x : T$)
    precondition $x \in s$
    changes s
    postcondition $s = s_0 - \{x\}$

  public method *contains*($x : T$) : $\mathbb{B}$
    postcondition $\mathit{result} = (x \in s)$

end MutSetI

A bit vector implementation:

Assumptions: T is a type of finite size $|T|$ and there is a one-one mapping ord_T from T to $\{0, .. |T|\}$.

var v : array $\langle$bool$\rangle$

With linking invariant

$$v.\mathrm{length} = |T| \\ \wedge\ \forall x \in T \cdot v(\mathrm{ord}_T(x)) = (x \in s)$$

An array implementation:

var a : array $\langle T \rangle$

var $size$: int

with linking invariant

$$|s| = size \leq a.\mathrm{length} = capacity \\ \wedge\ s = \{a(i) \mid i \in \{0, ..size\}\}$$

It follows that

$$\forall i, j \in \{0, ..size\} \cdot i \neq j \Rightarrow a(i) \neq a(j)$$

Unbounded stacks

interface UStackI$\langle T :: \text{type}\rangle$
  ghost public readonly var $s : \text{Seq}\,\langle T\rangle := [\,]$

  public method *size*() : int
    postcondition $result = length(s)$

  public method *push*($x : T$)
    precondition $length(s)$
    changes s
    postcondition $s = [x]\hat{}\,s_0$

  public method *pop*( )
    precondition $length(s) > 0$
    changes s
    postcondition $s = s_0[1, ..length(s_0)]$

  public method *top*() : T
    precondition $length(s) > 0$
    postcondition $result = s(0)$

end UStackI

A possible implementation

var $a : \text{DynamicArray}\,\langle T \rangle$

With linking invariant

$$s = \text{reverse}(a.s)$$

where

$$\begin{aligned} \text{reverse}([\,]) &= [\,] \\ \text{reverse}([x]\,\hat{}\,s) &= \text{reverse}(s)\,\hat{}\,[x] \end{aligned}$$