Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software 2020/21

TP5. Design by contract and verification of object-oriented programs in Dafny (Part II) - Solutions

1. Deque

a)

b)

```
predicate Valid() {
  // (i) consistency of the concrete state variables;
  size <= list.Length && start < list.Length
  // (ii) consistency between concrete and abstract (ghost) state variables
  && capacity == list.Length
  && elems == if start + size <= list.Length
              then list[start .. start+size]
              else list[start..] + list[.. (start + size) - list.Length]
}
```

c)

```
constructor (capacity: nat)
  requires capacity > 0
  ensures elems == [] && this.capacity == capacity
{
    list := new T[capacity];
    start := 0;
    size := 0;
    elems := [];
    this.capacity := capacity;
}

predicate method isEmpty()
   ensures  isEmpty() <==> |elems| == 0
{
    size == 0
}

predicate method isFull()
   ensures  isFull() <==> |elems| == capacity
{
    size == list.Length
}

function method front() : T
  requires !isEmpty()
  ensures front() == elems[0]
{
    list[start]
}
```

```
  function method back() : T
    requires !isEmpty()
    ensures back() == elems[ |elems| - 1 ]
  {
      list[(start + size - 1) % list.Length]
  }

  method push_back(x : T)
    requires !isFull()
    ensures elems == old(elems) + [x]
  {
      if start + size + 1 <= list.Length {
        list[start + size] := x;
      }
      else {
         list[start + size - list.Length] := x;
      }
      size := size + 1;
      elems := elems + [x];
  }

  method pop_back()
    requires !isEmpty()
    ensures elems == old(elems[..|elems|-1])
  {
      size := size - 1;
      elems := elems[..|elems|-1];
  }

  method push_front(x : T)
    requires !isFull()
    ensures elems == [x] + old(elems)
  {
      start := if start > 0 then start - 1 else list.Length - 1;
      list[start] := x;
      size := size + 1;
      elems := [x] + elems;
  }

  method pop_front()
    requires !isEmpty()
    ensures elems == old(elems[1..])
  {
      start := if start + 1 < list.Length then start + 1 else 0;
      size := size - 1;
      elems := elems[1..];
  }
```

d)

e) Using state abstraction, and also generics (which requires initializer to be passed to new T[]).

```
class {:autocontracts} Deque<T> {
  // (Private) concrete state variables
```

```
const list: array<T>; // circular array, from list[start] (front) to
                      //   list[(start+size-1) % list.Length] (back)
var start : nat;
var size : nat;

 // State abstraction functions
 function elems(): seq<T> {
     if start + size <= list.Length
     then list[start .. start+size]
     else list[start..] + list[.. start + size - list.Length]
 }
 function capacity(): nat {
   list.Length
 }

predicate Valid() {
  size <= list.Length && start < list.Length
}

constructor (capacity: nat, initializer: nat -> T)
  requires capacity > 0
  ensures elems() == [] && this.capacity() == capacity
{
    list := new T[capacity](initializer);
    start := 0;
    size := 0;
}

predicate method isEmpty()
   ensures isEmpty() <==> |elems()| == 0
{
    size == 0
}

predicate method isFull()
   ensures isFull() <==> |elems()| == capacity()
{
    size == list.Length
}

function method front() : T
  requires !isEmpty()
  ensures front() == elems()[0]
{
    list[start]
}

function method back() : T
  requires !isEmpty()
  ensures back() == elems()[ |elems()| - 1 ]
{
    list[(start + size - 1) % list.Length]
}

method push_back(x : T)
```

```
      requires !isFull()
      ensures elems() == old(elems()) + [x]
    {
        if start + size + 1 <= list.Length {
          list[start + size] := x;
        }
        else {
           list[start + size - list.Length] := x;
        }
        size := size + 1;
    }

    method pop_back()
      requires !isEmpty()
      ensures elems() == old(elems()[..|elems()|-1])
    {
        size := size - 1;
    }

    method push_front(x : T)
      requires !isFull()
      ensures elems() == [x] + old(elems())
    {
        start := if start > 0 then start - 1 else list.Length - 1;
        list[start] := x;
        size := size + 1;
    }

    method pop_front()
      requires !isEmpty()
      ensures elems() == old(elems()[1..])
    {
        start := if start + 1 < list.Length then start + 1 else 0;
        size := size - 1;
    }
}

// A simple test scenario.
method testDeque() {
     var q := new Deque<int>(3, i => 0);
    assert q.isEmpty();
    q.push_front(1);
    assert q.front() == 1;
    assert q.back() == 1;
    q.push_front(2);
    assert q.front() == 2;
    assert q.back() == 1;
    q.push_back(3);
    assert q.front() == 2;
    assert q.back() == 3;
    assert q.isFull();
    q.pop_back();
    assert q.front() == 2;
    assert q.back() == 1;
    q.pop_front();
    assert q.front() == 1;
```

```
    assert q.back() == 1;
    q.pop_front();
    assert q.isEmpty();
}
```

2. [Home Work] Priority Queue

TODO