U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Master in Informatics and Computing Engineering (M.EIC)
M.EIC037 | Formal Methods for Critical Systems 2021/22

Verification of object-oriented programs in Dafny using state abstraction

1. Deque [Selected]

Assume the following implementation of a double-ended-queue (DEQUE), with a bounded capacity, by means of a circular array, supporting the basic operations in time O(1). A test scenario is also provided.

```
type T = int // for demo purposes, but could be another type

class {:autocontracts} Deque {
    // (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;

    constructor (capacity: nat) {
        list := new T[capacity];
        start := 0;
        size := 0;
    }

    predicate method isEmpty() {
        size == 0
    }

    predicate method isFull() {
        size == list.Length
    }

    function method front() : T {
        list[start]
    }

    function method back() : T {
        list[(start + size - 1) % list.Length]
    }

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

    method pop_back() {
        size := size - 1;
```

```
    }

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

    method pop_front() {
        start := if start + 1 < list.Length then start + 1 else 0;
        size := size - 1;
    }
}

// A simple test scenario.
method testDeque()
{
    var q := new Deque(3);
    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();
}
```

To describe the operations' contracts and prove the correctness of their implementation, we will use the technique of state abstraction (see the Set example).

a) Add the following ghost fields to describe the state of the Deque in an abstract way (independently of the internal state representation):

```
    // (Public) ghost variables used for specification purposes only
    ghost var elems: seq<T>;  // front at head, back at tail
    ghost const capacity: nat;
```

b) Add a class invariant (predicate `Valid()`) to ensure: (i) the consistency of the concrete state variables; (ii) the consistency between the concrete and abstract (ghost) state variables (abstraction relation).

c) Add relevant pre- and pos-conditions to all methods, constructors, functions, and predicates, based on the abstract state variables only. You also need to add instructions to update the abstract state variables in the body of constructors and methods.

d) Check if all test assertions are successfully checked by Dafny and fix any problems found.

e) Redo with state abstraction functions instead of ghost variables.

2. Priority Queue

The tutorial Development of verified data structures in Dafny: priority queue implemented with a binary heap is incomplete, because the expressions of the class invariant (`heapInv`) and the loop invariants (`heapifyUpInv` and `heapifyDownInv`) are missing. Copy the code from the section "Putting it all together" to your working area, fill in the expressions for those invariants (based on the explanations and pictures in the tutorial), and confirm that Dafny checks successfully all the code.