Lab Session 8
ADTs and Typestates

Software Verification

Nova School of Science and Technology
Mário Pereira   mjp.pereira@fct.unl.pt

Version of October 28, 2025

The goal of this lab session is to practice on the construction of ADTs and its specification using *Typestates*.

While working on the different exercises, try to change the specification of predicates, functions, and lemmas. This is crucial to understand why the given pre- and post-conditions are necessary to prove this ADT correct.

1 Amortized Queues

The most efficient implementation of queue the data structure resorts to the use of a pointer-based linked-list, where one maintains a pointer to the first element in the list (the *head*) and a pointer to the last element (the *tail*). Such an implementation guarantees worst-case constant execution time for every operation: creation, checking for emptyness, inserting a new element, popping an element of the queue, and even concatenation.

Under a setting where dynamic memory is not supported, one can still derive a queue implementation with worst-case constant execution time for every operation, except *pop* and *concatenation*. However, for the pop operation, it is possible to implement it in *amortized constant-time*. Concatenation is doomed, we can only implement it in linear time.

The goal of this lab session is to implement and formally verify such a data structure in Why3.

1.1 Data Structure Implementation

Consider the following Why3 definition of polymorphic amortized queues:

```
type t 'a = {
  mutable front : list 'a;
  mutable rear : list 'a;
  mutable size : int;
}
```

The `front` list keeps its elements in insertion order and is used, essentially, to support the pop operation. On the other hand, `rear` keeps its elements in reverse order of insertion and is used to support insertion of new elements. Finally, field `size` stands for the total number of elements in the data structure.

As an example, if we have

```
front = Cons 1 (Cons 2 (Cons 3 Nil))
```

and

```
rear = Cons 6 (Cons 5 (Cons 4 Nil))
```

and, of course `size = 6`, this data structure corresponds to the *logical* queue

$$1; 2; 3; 4; 5; 6$$

1.2 Type Invariant and Ghost Representation

Exercise 1. Extend the `t 'a` datatype with a *ghost representation* for the queue structure. This means adding a new field to the record definition, one with a mathematical type, in a way that such field describes *mathematically* the contents of the data structure. □

Exercise 2. Equip the `t 'a` datatype definition with a proper *Type Invariant*. Your invariant should capture the following properties:

- Field `size` is the length of the ghost model field.
- If `front` is the empty list, so is `rear`.
- The ghost model field is equal to the sequence of elements in `front`, concatenated with the reversed sequence of elements in `rear`.

□

1.3 Operations Specification

Exercise 3. Provide proper specification for function `create`. □

Exercise 4. Provide proper specification for function `is_empty`. □

Exercise 5. Provide proper specification for function `push`. □

Exercise 6. Provide proper specification for function `pop`. □

Exercise 7. The proof of function `pop`, namely the updates on `q` preserve the type invariant, relies on the use of lemma `rev_of_list_commut`. This lemma basically states that if one reverts a list `l` and then converts it to a mathematical sequence, it stands for the same sequence of elements as if one would first convert `l` into a sequence and only then reverse such sequence.

Prove this lemma by induction on `l`. □

Exercise 8. Provide proper specification for function `transfer`. This function stands for the concatenation of the elements in `q1` to the end of `q2`, by emptying `q1`. □

1.4 Typestates

Exercise 9. Define a predicate `empty` that represents the *typestate* where a queue `q` is empty. □

Exercise 10. Define a predicate `not_empty` that represents the *typestate* where a queue `q` is non-empty. □

Exercise 11. Refine the specification of the queue operations to use the typestates defined in previous questions. □