Program Verification

Assignment 2

Niek Haarman, Tanja de Jong, Tinus Pool

July 3, 2015

For this assignment we specified and verified a sequential and concurrent `Queue` implementation. Below is a report about our findings.

1 Sequential Queue implementation

For the sequential Queue, we decided to go with a `LinkedList` implementation, which can be seen as a FIFO queue where items are appended at the tail, and taken from the head of the queue. We took the Java implementation of `LinkedList` and stripped it down to contain only code relevant to the exercise, and created a PVL implementation based on that.

The Java implementation keeps a reference to the first and last `Node` of the list as fields. This allows for easy appending and extraction of values. We chose to create a `contents()` function which describes the current contents of the list as a `seq<int>`. That way we can easily ensure items are added and removed properly. To do that, we declared a recursive `contents()` function on `Node`. This approach requires that the first `Node` has (recursively) at least a read permission on the `val` and `next` fields of all subsequent `Nodes`, as seen in Listing 1. This immediately leads to a problem, since we cannot easily obtain a write permission to append an item to the queue: either we have full write permissions on the contents of the `LinkedList`'s `last` field, or we have recursive read permissions for all `Nodes`. To solve this, we dropped the `last` field as a whole, and gave full recursive *write* permission to the `Nodes`. Appending an item now traverses the entire list, starting from the head. The up-to-date and working version can be found in the attachments.

We had to specify the methods `peek`, `poll` and `offer`. The latter two delegate their work to the 'private' functions `unlinkFirst()` and `linkLast(int)` respectively. Since we used the `contents()` function in our specifications, we can easily verify and test that our implementation works as intended.

To verify the sequential implementation, run the following command:

```
vct Integer.pvl Node.pvl Queue.pvl Test.pvl --silver=silicon --inline
```

```
class Node {
  Node next;
  int val;

  resource state() = Value(val) ** Value(next) ** next->state();

  requires state();
  seq<int> contents() =
  unfolding state() in (
    next == null
     ? seq<int>{val}
     : seq<int>{val} + next.contents()
  );
}
```

Listing 1: Basic Node specification