Problem 1

Assume that the sum of an empty sequence is zero.

Problem 2

You should NOT try to write to memory with something like:

```
DO ~(LMemory.Read(a)=c(a)) =>
  LMemory.Write(a,c(a))
OD
```

The idea is that, when writing back you write to memory once.

Remember, you cannot decide in the implementation whether the write to memory will succeed or not. Hint: You may want to write back more often. But try not to write back more often than you need to.

Traces of the implementation must be a subset of the traces of the specificatioin. You should carefully prove that this is true for your implementation by referring to the actions that your implementation performs.

It is likely that you will need to introduce some intermediate automaton C between implementation and specification. If you do that, you should prove that the implementation implements C and that C implements the specification.

Here is the definition of a backward simulation relation BR in case you need to use it in some proof that traces of automaton T are a subset of the traces of automaton S:

1. BR is a total relation, that is:

```
    FORALL t in states(T)
      EXISTS s in states(S)   such that  BR(t,s)
```

2. All states related to initial states of T
   are initial states in S:

```
FORALL t in initial(T)
  FORALL s in states(S) such that BR(t,s)
    s in initial(S)
```

3. It is always possible to extend BR along every transition in the backwards direction:

```
FORALL t in states(T)
  FORALL t' in states(T)
    FORALL (t',pi,t) in steps(T)
      FORALL s in states(S) such that BR(t,s)
        EXISTS s' in states(S) such that
          (s',pi,s) in steps(S)
```

In showing some of the trace inclusions you may want to use use invariants. If you use any invariants you must prove that every step preserves them.