1 Purpose of this assignment

This assignment is concerned with modelling a message bank system. The development is split into 2 assignments: 3a and 3b. Assignment 3b will build on 3a. The functionality of the message bank should be relatively easy to understand and developing a specification for this part of the assignment should be reasonably easy. This document will give the machines and the operations required and the behaviour of the machines. You are given variable names and their purpose, but not the invariants. This assignment is reasonably easy to get correct, although discharging proof obligations may be difficult.

The assignment uses sequences and you will need to look carefully at B’s sequence operations. See, for example, the information on sequences in the Concise B Summary and also in the lecture notes, B Set theory and basics.

This assignment develops

• your experience with developing adequate invariants and preconditions;
• specifying the body of operations;
• use of animation to check out scenarios;
• discharging proof obligations.

2 An Overview

In this assignment we attempt to model a messaging system like a phone message bank.

2.1 The Interface View

For the generic MessageBank machine we have customers and with any customer is associated a queue of messages, which can of course be empty. The available operation signatures are as follows:

custid ← NewCustomer: register a new customer and return a customer id, custid.
**RemoveCustomer**\(\text{(custid)}\): remove a customer. Before this operation is run, all the messages for the customer must have been removed.

**NewMsg**\(\text{(custid, msg)}\): send a message \((\text{msg})\) to customer \((\text{custid})\).

\[ \text{msg} \leftarrow \text{FirstMsg}\text{(custid)}: \text{read the first message for customer custid.} \]

\[ \text{msg} \leftarrow \text{CurrentMsg}\text{(custid)}: \text{re-read the current message for customer custid.} \]

\[ \text{msg} \leftarrow \text{NextMsg}\text{(custid)}: \text{read the next message for customer custid.} \]

**DeleteMsg**\(\text{(custid)}\): delete the current message for customer custid. After **DeleteMsg** the current message should be set to the predecessor (if any) of the deleted message.

### 2.2 Current and Next messages

The *current* message is the most recently read message. The current message will only have meaning after the first message has been read.

The *next* message is the message after *current*. There is no next message after the last message in the queue, and nor is there a next message if the current message is undefined.

### 2.3 The policy of the message bank

1. Each customer has a queue of messages. A *queue* is a first-in-first-out structure. The messages are stored in the order of their arrival. A new customer has an empty queue.

2. A customer must read their messages in the order they arrived. So to read any message in the queue all messages that preceded that message must have been read.

3. The message bank has a concept of the *current* message, which is the most recently read message still on the queue. Hence, until the first message on the queue has been read there is no current message.

4. Deletion deletes the current message. Consequently, a message cannot be deleted until it has been read.

A small scenario to illustrate the above operations on a single customer’s messages.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>first</td>
<td>current</td>
</tr>
<tr>
<td>FirstMsg</td>
<td>1</td>
<td>undefined</td>
</tr>
<tr>
<td>CurrentMsg</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DeleteMsg</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>NextMsg</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>DeleteMsg</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

### 3 What you have to do

Because this assignment and the next (assign3b) are going the develop the MessageBank machine two different ways, the machine for this assignment will be called MessageBankS (MessageBank using sequences) You should:

1. Develop MessageBankS machine in which each customer’s messages is modelled as a sequence.
2. The MessageBankS machine is shown with three variables:

- **customers**: the set of current customers;
- **messages**: the set of message queues;
- **current**: information that enables the location of the current message for each customer.

3. Document the invariant and the preconditions with short comments that express the purpose of a constraint from the requirements point of view. Don’t simply describe a predicate in English.

4. Discharge as many Proof Obligations as possible. There will be some POs that are difficult to discharge.

5. Create a test scenario for the Animator.

### 3.1 The MSG set

The set MSG is a deferred set and represents the set of all messages. Leave it uninstantiated. It will be essentially a set of numbers, although symbolic constants may be used during animation, for example HiJanehowru.

### 3.2 Be Careful

Use size for the size of a sequence, not card. While size is card the converse is not true. Sequences are always finite, size is defined with that knowledge. Thus, if ss is a sequence, then size(ss) ∈ N is always true, but if you happened to write card(ss) you would find yourself having to prove the ss is a finite set. If you used card then card(<>) will leave the theorem prover not knowing what to do, whereas it knows that size(<>) is 0.

The above comments only affect theorem proving.
MACHINE MessageBankS ( MSG , maxcustomer , maxmessage )
CONSTRAINTS maxcustomer ∈ N_1 ∧ maxmessage ∈ N_1
SEES CUSTOMER_CTX

VARIABLES
    customers ,
    messages ,
    current

INARIANT

INITIALISATION

OPERATIONS
    custid ←− NewCustomer ;
    RemoveCustomer ( custid ) ;
    NewMsg ( custid , msg ) ;
    msg ←− FirstMsg ( custid ) ;
    msg ←− CurrentMsg ( custid ) ;
    msg ←− NextMsg ( custid ) ;
    DeleteMsg ( custid ) ;

END

Figure 1: MessageBank machine

MACHINE CUSTOMER_CTX
SETS CUSTOMER
END

Figure 2: CUSTOMER context machine