Prof. Dr. Peter Thiemann
Sergio Feo-Arenis
Sergiy Bogomolov

Summer Term 2014

Softwaretechnik

http://proglang.informatik.uni-freiburg.de/teaching/swt/2014/

Exercise Sheet 3

Exercise 1: The B Method and ProB (7 Points)

This exercise focuses on the specification of a simple elevator.

Tool

For this exercise, we will use the tool ProB, available for download at `http://www.stups.uni-duesseldorf.de/ProB`. Follow the installation instructions on the webpage.

Elevator specification

The set of floors served by the elevator is modeled by a constant interval of integers. There is a single cab (also known as cage) in this specification. The cab is equipped with a door, that may be either closed or open. For simplicity, the specification does not account for floor doors.
The elevator is specified in the B abstract machine Elevator provided below (for further details on the syntax, see ProB's built-in help files, for additional information on concrete vs. abstract variables, see e.g. `http://home.gna.org/brillant/manref/B-Typing.html`). The meaning of the cab and door concrete variables should be self-explanatory. The req variable contains the pending requests as a subset of floors. Requests may originate either from inside the cab or from a floor: the machine does not care (i.e., it processes these two kinds of requests identically). There is initially no pending request. The machine contains three operations: an operation to move the cab to another floor, an operation to toggle the state of the doors, and an operation to add a new request. A request is considered served (and, hence, removed from req) when closing the cab door at the corresponding floor.

```
MACHINE
   Elevator
SETS
   /* Possible states of the cab door. */
   DSTATE = {closed, open}
CONSTANTS
   /* Floors range from flmin to flmax. */
   flmin,
   flmax,
   floors
PROPERTIES
   flmin : INT &
   flmax : INT &
   flmin < flmax &
```

```
   floors = (flmin..flmax)
CONCRETE_VARIABLES
   /* Cab's current floor. */
   cab,
   /* State of the cab door. */
   door
ABSTRACT_VARIABLES
   /* Pending requests. */
   req
INVARIANT
   cab : floors &
   req <: floors &
   door : DSTATE
   &
   >>> To Be Completed <<<
INITIALISATION
   cab := flmin ||
   req := {} ||
   door := closed
OPERATIONS
   request (fl) =
   PRE
   >>> To Be Completed <<<
   END
   ;
   move =
   PRE
   >>> To Be Completed <<<
   END
   ;
   toggle =
   PRE
   >>> To Be Completed <<<
   END
END
```

Tasks

Create the file `Elevator.mch` using ProB with the template above and complete the following tasks:

1. Specify in the INVARIANT clause of the machine `Elevator` the following property: *"if the door is open then the current floor is among the pending requests."*
2. Specify the `request (fl)` operation: adds the floor *fl* to the set (not a multiset) of pending requests. It is allowed for the given floor fl to already be a pending request.
3. Specify the `move` operation: moves the cab to an arbitrary floor that is among the pending requests. The current floor must not be a pending request, and the door must be closed.

4. Specify the `toggle` operation: switches the state of the cab door. Opens the door if it is closed, and closes it otherwise. The cab door must be toggled only if the cab's current floor is a pending request. If the operation closes the door, the request is removed from the set of pending requests.

5. Show that the operations you defined are consistent. I.e., for `INVARIANT` *I* and operation `PRE` *P* `THEN` *S* `END`, show that `I & P => [S]I`.

6. Save the file and use the model checking feature of ProB to verify that the operations you defined are consistent for values of *flmin* and *flmax* between 0 and 10. Hint: use the menu "Preferences − Animation Preferences" to set appropriate values of MININT and MAXINT. Additionally, set the preferences "Max number of initialisations computed" and "Max number of Enablings per Operation Computed" to a sufficiently large number to ensure all states are explored. Report the number of states and the time required to model check your specification.

Exercise 2: B Types (3 Points)

Consider the typing rules for set expressions in B as presented in the lecture. For each of the following set expressions: if the expression is well-typed give (or describe) its value, or state why it is not well-typed otherwise.

1. `NAT`
2. `POW(1)`
3. `{1} * {{1}, NAT}`
4. `1 :  2`
5. `card({{{}, {1, {}}, {2}}})`
6. `{x | x = 1 or x = TRUE}`
7. $\{1\} \cap \{\{1\}\}$

Exercise 3: OOA (10 Points)

In this exercise, you are requested to perform the first steps towards creating an OOA model for an event management system (EMS). For the organization of a music festival, the EMS should provide the following essential features:

- Users of an EMS can create accounts. After successful login, users can create new events and register to existing events. Users can have different roles (regular participant, organizer, roadie, VIP).
- By default, the creator of a new event is an organizer of the created event, and registering to an event implies the role of a regular participant. Organizers can create new users and register them to events on their behalf. Moreover, organizers can assign roles to registered users.
- The registration as a regular participant implies the purchase of a ticket. The payment of tickets can be done in different ways (cash, credit card, invoice). There are different types of tickets: tickets can be valid for one day or for a longer duration. Moreover, each ticket has an assigned area (backstage, regular, all areas).

Tasks

1. Identify the most important classes of the EMS based on the features described above.
2. Identify associations and compositions of the identified classes.
3. Identify attributes and operations for each class.
4. Provide a UML class diagram containing the items you have identified.
5. Provide a sequence diagram for the use case "buy ticket online".

Submission

- Submit this sheet *before* the lecture of Thursdays.
- Late submissions will not be accepted.
- Deadline: Thursday 11:59 a.m..