
6SENG001W Reasoning about Programs

Tutorial 3. Extend & Develop *Abstract Machines* using B Tools

Introduction

In this tutorial you are required to use the two B tools **Atelier B** & **ProB** to extend & develop *abstract machines*.

Any exercises that are not completed during the tutorial can be done as independent study or in your next tutorial.

Exercise 3.1

This exercise extends the [PaperRound.mch](#) abstract machine given in the lectures & used in tutorial 1.

Add the following operations to the PaperRound machine:



- *firsthouse*: outputs the number of the first house in the street that currently has a paper delivered.
- *lasthouse*: outputs the number of the last house that currently has a paper delivered.
- *haspaper*: takes a house number as parameter & outputs a message to indicate if the house has a paper delivered or not.
- *stopdelivery*: takes a house number as parameter & removes it from the set of houses that have paper deliveries.

It outputs a message to indicate that either:

- the house will no longer have a delivery, i.e. was removed successfully; or
- the house does not have a paper delivered, i.e. not in the set.

Hint: you need to work out what the *precondition* for this operation is before you define it.

You must **syntax & type check** the extended specification using **Atelier B**.

Exercise 3.2

Animate the extended *PaperRound* machine using **Atelier B**.

Ensure that all of the operations that you have added work correctly.

Do this systematically:

- Produce a **Test Table**, i.e. a list of "*test cases*" to use to test each new operation, in terms of the parameter (house number) & the paper round machine's state (houseset).
 - Then add these "*test cases*" to the **PaperRound.mch** file as a comment at the end of the file.
-

Exercise 3.3

Add a second state variable to the PaperRound machine to keep track of the households that have *magazines* delivered.

You must decide:

- What type this new variable should be.
- What its initial value should be.

Syntax & type check the extended specification, using **Atelier B**.

Exercise 3.4

Add the following operations to the new *PaperRound* machine with the *magazines* state variable:

- *deliverMagazine*: takes a house number as parameter & adds it to the houses that have a magazine delivered.
- *stopMagazine*: takes a house number as parameter & removes it from the houses that have a magazine delivered.
- *deliveries*: takes a house number as parameter & outputs a message to indicate what is delivered to this house.
- *stopalldeliverys*: takes a house number as parameter & removes it from the set of houses that have both a paper & magazine delivered.

It outputs a message to indicate that either:

- the house will no longer have either delivered; or
- there was a problem as it does not have both delivered.

You must **syntax, type check & animate** the extended specification, using **Atelier B** & **ProB**.

Exercise 3.5

After a few months the paper round manager realises that it is no longer profitable to deliver to customers that only want a magazine but no paper.

So a new rule is introduced that only customers that have a paper delivered can have a magazine delivered.

So now a house can have:

- just a paper delivered, or
- a paper & a magazine delivered,
- but NOT just a magazine delivered.

So amend the PaperRound machine so that this new rule applies to all of its states.

(Hint: think about the relationship between the houses that have papers delivered & those that can have magazine delivered.)

Exercise 3.6

Animate this modified *PaperRound* machine with the new delivery rule using [ProB](#).

You should discover that several of the operations that alter the state now no longer work properly or at all, i.e. are not "offered for selection" in [ProB](#).

Identify these operations by "attempting" to animate them in [ProB](#).

Then using :

- [ProB](#)'s analysis features &
- by documenting these "bugs" in a **Test Table**, i.e. a list of the "test cases" that caused errors.

Identify the errors & modify the operations so that they work correctly, i.e. **do not break the new delivery rule**.

Exercise 3.7

Due to a high profile child exploitation court case, the paper round manager realises that he has been breaking the law, as the child doing the paper round is carrying too many papers & magazines.

So he has to limit the number of houses that can have deliveries to just 10.

Amend the PaperRound machine (invariant & operations) so that it enforces this new requirement on its state.

Further add the following useful operation for the paper round manager:

- *howmany*: outputs how many more deliveries can be added to the paper round.

Finally, **syntax & type check** the amended specification, using [Atelier B](#), then animate & test the amended operations using [ProB](#).

Exercise 3.8

By referring to & using parts of the *Sets* abstract machine given at the end of lecture 2 & used in tutorial 2, define a **new machine** called *Grid*.

The *Grid* machine is intended to represent a grid & the position of a cursor within the grid.

Grid machine has the following components:

- The limits of a 9x7C grid of squares, i.e. x-axis 1 to 9 & y-axis 1 to 7.
(Hint: you do not have to represent the actual "grid squares", just the grid dimensions.)
- Two variables to represent the state of the system, i.e. the position of the cursor within the 9x7 grid.
- The cursor keys that can move the cursor around the screen, i.e. Up, Down, Left, Right.
- The following operations:
 - *move*: takes a direction as parameter & moves the cursor in that direction, provided it stays within the grid.

Alternatively, you can define 4 separate operations, one per direction.

- *reset*: places the cursor at the starting position within the grid.

- *position*: outputs the current position of the cursor as two numbers, i.e. its x-axis & y-axis coordinates.

Syntax, type check & animate the Grid specification, using **Atelier B** & **ProB**, to ensure that **the cursor moves around the grid correctly**.

Exercise 3.9

An abstract machine can be represented by a *Structure Diagram* (see Lecture 3) that presents an overview of its structure.

A *Structure Diagram* includes an abstract machine's state & operations, in particular its:

- Sets
- Constants
- State Variables
- State Invariant
- Operations

Draw the *Structure Diagram* for the following abstract machines:

- PaperRound (Final version.)
 - Grid
-

Last updated: 9/10/17
