6SENG001W Reasoning about Programs

Tutorial 7: Sequences in B

Introduction

These tutorial exercises refer to the notes for **Lecture 7: Sequences**.

In this tutorial you are required to use the two B tools Atelier B & ProB to animate & extend a B machine that represents a small part of the London Underground system around the Cavendish Campus, called *Tube.mch*.

This Tube specification illustrates how sequences (Lecture 7's topic) can be used in a specification.

The *Tube* (partial) specification is given in this B machine `Tube.mch`.

The specification includes information about 4 of the tube lines around the Cavendish Campus, see the Tube Map:

- the *tube lines*: Bakerloo, Victoria, Northern, Circle.
- a small number of tube stations (16) on those lines, that are within a relatively small area of the tube map around this area of London.
- a variable `tube_journey` that represents the planned tube journey as a sequence (i.e. a queue) of stations.

The specification also has the following operations, that relate to constructing a *tube journey* using the lines & stations.

- `AddNextStationToJourney` -- add a station to the sequence of stations that represents the tube journey.
- `RemoveStartStationFromJourney` -- remove the first tube station from the journey. Currently it is just a *"dummy"* operation, i.e. its body is just `skip`.
- `JourneyStatus` -- an enquiry that provides a *"status report"* about the planned journey.

Exercise 7.1

Review the **Lecture 7: Sequences** notes, in particular familiarise yourself with the **AMN versions of the Sequences symbols**.

See ProB's "Help" menu for a summary of the B syntax.

Exercise 7.2

Create a new B "Project" using Atelier B, then create a new "Component" & then paste the *Tube* B machine into it.

Type check it using Atelier B.

Exercise 7.3

Load the Tube machine specification into ProB.

Animate it & execute the three operations:

```
AddNextStationToJourney

RemoveStartStationFromJourney

JourneyStatus
```

Test both the *successful* & *error* cases of the operations.

After executing each operation **check** how the state has been modified.

Do this by use ProB's "`Eval`" terminal to check the values of the three state variable that make up the state.

Also use the "`Eval`" terminal to test the truth value of the:

- Operations's *preconditions* &
- IF statements's *conditions*.

Exercise 7.4

Replace the "`skip`" body of the `RemoveStartStationFromJourney`, by commands that actually perform the intended action.

Type check it using **Atelier B** & animate it using ProB to check that it does remove the first station from the journey.

Exercise 7.5

Add a constant mapping to the Tube's state that maps an individual station to the tube line(s) its on.

For example, `Regents_Park` is only on the `Bakerloo` line, but `Oxford_Circus` is on both the `Bakerloo & Victoria` lines.

What kind of mapping is this -- a relation or a function? Consider each of the possible options & think about what the implications are of each one in relation to the stations & lines.

The mapping's *properties* must be defined & it should be initialised as appropriate.

Exercise 7.6

For each line add a constant sequence that represents the tube line, as the sequence of stations, in just one direction only, i.e. either East to West or North to South.

Exercise 7.7

For each line add a constant mapping to the Tube's state that maps an individual station to its *adjacent* tube station in one direction, either East to West or North to South.

For example, for the `Bakerloo` line going North to South: `Baker_Street` next to `Regents_Park`, `Regents_Park` next to `Oxford_Circus`, etc.

Then using the above mapping:

a. Produce the opposite direction adjacency mapping using a built in B operator.

b. Then by combining both of the above define it for each line in both directions.

c. Finally, using all of the above define the *"adjacency mapping"* for the whole of this Tube System.
Hint: use the union of all of the adjacent station mappings.

Exercise 7.8

Add the following enquiry operations:

a. `journeyLength` -- returns the *length of the journey*.

b. `returnJourney` -- returns the *sequence of stations* that make up the return journey.

c. `firstStation` -- returns the *first station* on the journey.

d. `lastStation` -- returns the *last station* on the journey.

e. `goOnLines` -- returns the set of *tube lines* that the journey's stations are on.

f. `journeyLeft` -- inputs a *number* that is the number of stations travelled through on the journey. Then provided the journey involves at least that *number* of stations, it output the remaining sequence of stations on the journey. If there are less than *number* stations on the journey, it reports an error.

Exercise 7.9

Amend the `AddNextStationToJourney` operation so that a station can only be added to the end of the sequence of stations that represents the journey provided that *"it is adjacent to the current last station"*.

For example, based on the Tube map, if the last station was `Baker_Street` it could be either `Regents_Park` or `Great_Portland_Street`, but not `Oxford_Circus`.

Hint: this means adding an extra pre-condition to the operation when adding a station, to make sure that its adjacent to the last station. So use the *adjacent tube stations* defined in **Exercise 7.7**. This extra pre-condition should not be added to the `PRE`'s conditions, as you need to report this as an error, but in the `THEN` part, as an `IF` condition.

Coursework

Begin working on the Maze & Robot B Specification Coursework.

Last updated: 1/12/17
