Skeleton for Exercise 1:

Sequences.mch

The Tube system B machine and a tube Map:

tubemap.pdf

Tube.mch

6SENG005W Formal Methods

Week 08 Tutorial: Sequences in B

Introduction

These tutorial exercises refer to the **Week 07 Lecture: Sequences** notes.

Exercise 1

In the lecture we saw how a number of built-in operators could be reproduced, but two were left open:

- The concatenation operator ^
- The down-arrow operator \|/

Try to figure out how these could be defined if they did not exist yet.

Exercise 2

Th Tube.mch file represents a small part of the London Underground system around the Cavendish Campus using sequences.

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 nearby tube stations (16) on those lines.
- a variable `tube_journey` representing the planned tube journey as a sequence of stations.

The specification also has the following operations, for constructing a *tube journey*.

- `AddNextStationToJourney` -- add a station to the sequence representing 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.

Task 2.1

Load the Tube machine specification into ProB.

Initialise it and execute the three operations:

```
AddNextStationToJourney
RemoveStartStationFromJourney
JourneyStatus
```

Test both the *successful* and *error* cases, checking how the state has been modified.

Task 2.2

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

Use ProB to check that it does remove the first station from the journey.

Task 2.3

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 and Victoria` lines.

What kind of mapping is this -- a relation or a function?

The mapping's *properties* (type and value) must be defined as appropriate.

Task 2.4

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.

Task 2.5

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.

Define this mapping based on the line's sequence, so that if the sequence changes, the mapping is automatically updated (e.g. adding a new station to a line should only require editing the line's sequence, not the adjacency map).

a. Then using the above mapping, 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.

Task 2.6

Amend the `AddNextStation` operation so that a station can only be added if *"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 condition (in the `IF-THEN` part) 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 **Task 2.5**.