Course 02263 Mandatory Assignment 2
Spring 2015

Anne E. Haxthausen
DTU Compute, Technical University of Denmark
aeha@dtu.dk

April 14, 2015

Abstract

This document contains a mandatory exercise that must be solved as a part of course *02263 Formal Aspects of Software Engineering* in 2015.

Contents

1 Practical Information

- The assignment should be solved in the same groups of two persons as for mandatory assignment 1.
- This exercise text, and specification skeletons will be provided on CampusNet under Assignment/Opgaver.
- There will be question time
  - Friday 24 April at 9-10
  - Friday 8 May at 9:30-approx. 11
  - Friday 12 May at 9-10

  in the usual class room in bldg. 303B. It is possible to get advice on *what* you are supposed to do, but not on the solution as it is an exam. You should solve as much as possible of the exercise *before* the question sessions.
- The solution to the assignment should be made in the form of a group report.
  - Full names, study numbers and signatures of all persons should appear on the front page.
  - The report must contain RSL specifications of the problems described in the assignment and some informal explanations.
  - We recommend that you use LaTeX to produce your report.
  - Due to the rules about individual evaluation, you must indicate who have made which parts. However, a certain degree of collaboration is also expected/required. Please ensure that each of you have contributed to each of the following kinds of tasks:
    - to define concrete types
    - to define observer and generator functions
    - to define informal requirements
    - to define predicates (for checking the requirements for nets and time tables)
    - to define test values representing concrete nets and time tables, and make test cases checking these test values

    Also please try to divide the tasks equally between you. Two persons can for instance divide the tasks as follows:
    - Person 1 defines the Net type, observers and generators, and half of all informal requirements + their formalization in predicates.
    - Person 2 defines the $TimeTable$ type, observers and generators, and half of all informal requirements + their formalization in predicates.

- It is a requirement (for passing) that all modules, types and values have names as required in the assignment text. (We need that when evaluating your solution, as we plan to run an automatic test on your specifications.)
- It is a requirement (for passing) that all specifications have been type checked successfully with the RAISE tools, and that your test specifications have been translated successfully with the RAISE tools.
- One of the group members must (1) upload (a) a zip file containing all your specifications and (b) a pdf-file containing the group report on CampusNet under Assignment/Opgaver and (2) deliver a printed version of the report (inside a closed envelope on which you have written your signatures and study numbers) in the box labelled 02263 in front of my office (room 054) in building 303B **not later than** noon on Monday May 18th, 2015.

2 Informal Description of the Problem Domain

A *tram* (Danish: sporvogn) is a vehicle that runs on rails in the streets. The goal of this assignment is to specify (1) data types for representing tram nets and tram time tables, and (2) specify predicates that can be used to check whether concrete values of these types satisfy some wellformedness requirements. Below we explain the notions of tram nets and tram time tables.

2.1 Tram Nets

Definitions and assumptions:

- A tram *net* consists of *stops* and *connections* between stops.
- A *stop* is a place where people can get on and off trams. A stop has a *name*, a *capacity* and a *minimum waiting time*. The *capacity* of a stop is the maximal a number of trams that are allowed to be at the stop at the same time. (If a stop has capacity n, it consists of n parallel tracks, each track having capacity for one tram.) The minimum waiting time is the minimum time a tram must wait at the station, from its arrival until its departure.
- A *connection* is a single track connecting two stops. In this assignment it is assumed that there is at most one track connecting two stops. (So trams can only pass each other at stops.) A connection has a *capacity* which is the maximal number of trams that are allowed to be on the single track at the same time.[1] A connection also has a *minimum driving time*, i.e. the time it must at least take for a tram to drive between the two stops.

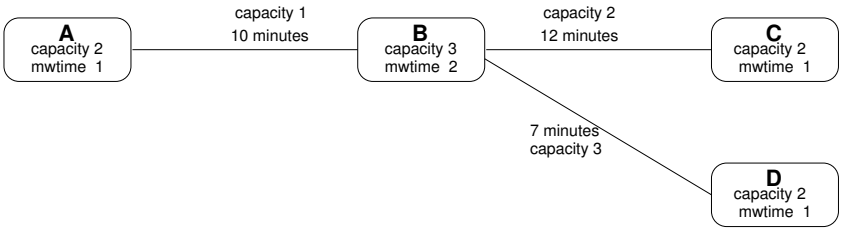

Figure 1: Example of a tram net.

Figure 1 shows an example of a net having four stops, A, B, C, and D with capacities 2, 3, 2, and 2, respectively, and minumum waiting times 1, 2, 1, and 1, respectively. Furthermore, the net has three connections with capacities 1, 2, and 3, respectively, and minimum driving times 10, 12, and 7, respectively.

[1] For instance, if a connection between stops B and C has the capacity 2, it means that at the same time at most two trams must be driving on the track from B to C or at most two trams must be driving on the track from C to B.

2.2 Time Tables

Assume given a tram net as described above. We will now describe what is meant by a time table for trams driving in this net.

Definitions and assumptions:

- Each tram is identified by a unique *name*.
- A *time table* associates with the *name* of each tram to be used, a *plan* for that tram.
- A *plan* for a tram states the names of those stops in the net where the tram will stop, and for each stop the arrival time and the departure time for that stop. In this assignment it is assumed that a tram must stop on all stops that it passes on its route.
- *Time* should be stated as whole minutes (e.g. 0, 15, and 100) (for simplicity in this assignment).

tram1		
A	0	1
B	12	14
C	28	32
B	46	48
A	59	60

tram2		
D	0	2
B	10	13
A	28	32
B	47	50
D	58	60

Figure 2: Example of a time table for the net shown in Figure 1.

Figure 2 shows an example of a time table for two trams running in the net shown in Figure 1. The names of the trams are "tram1" and "tram2".

3 Formal Specification of Nets

On CampusNet you can find a file `NET.rsl` that contains parts of a specification of tram nets:

```
scheme NET =
class
  type
   Net = ..., -- tram nets
   Time = Nat, -- times in number of minutes
   Capacity = Nat, -- capacities
   StopId = Text  -- names of stops

  value /* generators */
    empty :  Net,  -- the empty net

    -- insert a stop with a given name, capacity and minimum waiting time
    insertStop : StopId × Capacity × Time × Net → Net,

    -- add a connection between given stops,
    -- with a given capacity and a given minimum driving time
    addConnection : StopId × StopId × Capacity × Time × Net → Net

  value /* observers */
    -- check whether a stop is in a network
    isIn : StopId × Net → Bool,

    -- check whether two stops are directly connected in a network
    areDirectlyConnected : StopId × StopId × Net → Bool,

    -- get minimum driving time between two connected stops
    minDrivingTime : StopId × StopId × Net ~→ Time,

    -- get the capacity for a connection between two connected stops
    capacity : StopId × StopId × Net ~→ Capacity,

    -- get the capacity of a stop
    capacity : StopId × Net ~→ Capacity,

    -- get the minimum waiting time of a stop
    minWaitingTime : StopId × Net ~→ Time

  value /* predicates to check nets */
    isWellformed : Net → Bool
    isWellformed(n) ≡ ...
end
```

1. You should now complete this specification.

   - Complete the definition of the type type `Net` of values for representing nets. You are allowed to introduce auxiliary types, if needed.
   - Make explicit definitions of the stated generators and observers. Note that the generator functions should be total and without any pre conditions such that they can also be used to build illegal networks.
   - Write informally the requirements that any net must fulfill.
   - Give explicit definitions of predicates that can be used to check the requirements for nets. The predicates must have the type `Net` $\rightarrow$ **Bool** and they must be defined in terms of the observer functions. You are allowed to introduce auxiliary functions (e.g. derived observer functions) if that is convenient either for expressing the predicates or for later use.

     The function `isWellformed` should be defined such that it checks all requirements.

2. Type check the module.

3. Is the module translatable to SML? If not, refine it into a translatable form.

4 Formal Specification of Time Tables

On CampusNet you can find a file `TIMETABLE.rsl` that contains parts of a specification of time tables:

```
scheme TIMETABLE = extend NET with
class
  type
    TimeTable = ..., -- time tables
    ...,
    TramId = Text -- tram names

    value /* generators */
      -- the empty timetable
      empty: TimeTable,

      -- add to a time table an empty plan for a new tram
      addTram: TramId × TimeTable → TimeTable,

      -- add a stop with arrival time and departure time to the plan for a given tram
      addStop: TramId × StopId × Time × Time × TimeTable → TimeTable

  value /* observers */
    -- check whether a tram with a given name exists in a given time table
    isIn : TramId × TimeTable → Bool,

    -- you can add more observers here
    ...

  value /* predicates to check time tables */
    isWellformed : TimeTable × Net → Bool
    isWellformed(t, n) ≡ ...,

    ...

end
```

1. You should now complete this specification.

   - Complete the definition of a type `TimeTable` of values for representing time tables. You are allowed to introduce auxiliary types, if needed.
   - Make explicit definitions of the stated generators and observers. Note that the generator functions should be total and without any pre conditions such that they can also be used to build illegal time tables.
   - Write informally the requirements that any time table must fulfill.
   - Give explicit definitions of predicates that can be used to check the requirements for time tables. The predicates must be defined in terms of the observers. You are allowed to introduce auxiliary functions (e.g. derived observer functions) if that is convenient for expressing the predicates.

     The function `isWellformed` should be defined such that it checks all requirements.

2. Type check the module.

3. Is the module translatable to SML? If not, refine it into a translatable form.

5 Testing the Specifications

Write one or several RSL modules that can be used to test your predicates. It is a requirement (to pass) that your tests include tests for checking that the network and time table given in Figures 1 and 2, respectively, are wellformed.