

Course 02263 Mandatory Assignment 2

Spring 2016

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

April 13, 2016

Abstract

This document contains a mandatory exercise that must be solved as a part of course *02263 Formal Aspects of Software Engineering* in 2016.

Contents

1	Practical Information	2
2	Informal Description of the Problem Domain	4
2.1	Tram Nets	4
2.2	Time Tables	5
3	Formal Specification of Nets	6
4	Formal Specification of Time Tables	8
5	Testing the Specifications	10

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 15 April in the exercise session (after the lecture). In 033 in building 321.
 - Thursday 21 April at 12:00-13:00 (extra, outside schedule). In building ???
 - Friday 29 April at 9:00-11:00 (before the guest lecture). In 033 in building 321.
 - Wednesday 4 May 12:00-13:00 (extra, outside schedule). In building ???
 - Tuesday 10 May at 9:00-10:00 (used as a Friday at DTU). In 033 in building 321.
 - Wednesday 11 May at 9:00-10:00 (used as a Friday at DTU). In 033 in building 321.

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 L^AT_EX 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 functions for checking the requirements for nets as well as for 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 (1) the *Net* type, observers and generators, (2) half of all informal *Net* requirements + their formalization in functions, and (3) half of all informal *TimeTable* requirements + their formalization in functions.

* Person 2 defines the *TimeTable* type, observers and generators, (2) half of all informal *Net* requirements + their formalization in functions, and (3) half of all informal *TimeTable* requirements + their formalization in functions.

- 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 Friday May 13th, 2016.

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 functions 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* and a *capacity*. The *capacity* of a stop is the maximal 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.)
- A *connection* has two parallel tracks connecting two stops. (There is one track dedicated to each driving direction.) 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. Furthermore, a connection has a *minimum headway*, i.e. the minimum time that must pass between two following trains entering or leaving the same track of the connection.

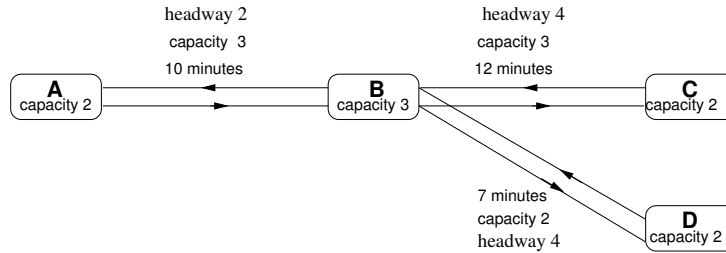


Figure 1: Example of a tram net.

Figure 1 shows an example of a net having four stops, A, B, C, and D with capacities 2, 3, 2, and 2, respectively. Furthermore, the net has three connections with minimum headways 2, 4 and 4, respectively, capacities 3, 3, and 2, respectively, and minimum driving times 10, 12, and 7, respectively.

¹For instance, if a connection between stops B and D has the capacity 2, it means that at the same time at most two trams may be driving on the track from B to D and at most two trams may be driving on the track from D to B , i.e. at the same time there can be up to four trams between B and D , two in each direction.

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, 14, and 60) (for simplicity in this assignment).

tram1			tram2		
A	0	1	D	0	2
B	12	14	B	11	13
C	28	32	A	28	32
B	46	48	B	47	50
A	59	60	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.rs1` that contains parts of a specification of tram nets:

```
scheme NET =
class
  type
    Net = ..., -- tram nets
    Headway = Time, -- minimum headways
    DrivingTime = Time, -- minimum driving times
    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 and capacity
    insertStop : StopId × Capacity × Net → Net
    ...,

    -- add a connection between given stops,
    -- with the given capacity, minimum driving time and minimum headway
    addConnection : StopId × StopId × Headway × Capacity × DrivingTime × 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
    minHeadway : StopId × StopId × Net  $\tilde{\rightarrow}$  Headway
    ...,

    -- get minimum driving time between two connected stops
    minDrivingTime : StopId × StopId × Net  $\tilde{\rightarrow}$  DrivingTime
    ...,

    -- get the capacity for a connection between two connected stops
    capacity : StopId × StopId × Net  $\tilde{\rightarrow}$  Capacity
    ...,

    -- get the capacity of a stop
    capacity : StopId × Net  $\tilde{\rightarrow}$  Capacity
    ...

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

1. You should now complete this specification.
 - Complete the definition of the 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 fulfil.
 - Give explicit definitions of functions that can be used to check the requirements for nets. The functions 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 functions 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:

```
NET
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  $\times$  TimeTable  $\rightarrow$  TimeTable
    ...,

    -- add a stop with arrival time and departure time to the plan for a given tram
    addStop: TramId  $\times$  StopId  $\times$  Time  $\times$  Time  $\times$  TimeTable  $\rightarrow$  TimeTable
    ...

  value /* observers */
    -- check whether a tram with a given name exists in a given time table
    isIn : TramId  $\times$  TimeTable  $\rightarrow$  Bool
    ...,

    -- you can add more observers here
    ...

  value /* predicates to check time tables */
    isWellformed : TimeTable  $\times$  Net  $\rightarrow$  Bool
    isWellformed(t, n)  $\equiv$  ...,

    ...

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 fulfil.
 - Give explicit definitions of functions that can be used to check the requirements for time tables. The functions 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 functions.

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 checking functions and aux functions. 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. You will be evaluated on how thoroughly you test you specifications.