# Course 02263 Mandatory Assignment 2
# Autumn 2022

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

November 3, 2022

**Abstract**

This document contains a mandatory exercise that must be solved as a part of course *02263 Formal Aspects of Software Engineering* in 2022.

## 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

**Assignment material:**

- This exercise text, a report skeleton, and specification skeletons are provided on DTU Learn under Assignments → Mandatory Assignment 2.

**Groups:**

- The assignment should be solved in the same **groups** as for mandatory assignment 1 (unless group changes have been approved by Anne Haxthausen before the start of the project period).

**Deliverables and their submission:**

- The solution to the assignment should be made in the form of a **group report**.
  - You must use the LaTeX report skeleton provided on DTU Learn.
  - Group number, full names, and study numbers of all persons of the group who have actively contributed to the group work should appear on the front page.
  - The report must contain all RSL specifications of the problems described in the assignment and some informal explanations.
- 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.) Furthermore, all specifications must have been type checked successfully with the RAISE tools, and your test specifications must have been translated successfully with the RAISE tools. Solutions that do not fulfil these requirements will not be considered.
- **Submission:** One of the group members must upload (a) a zip file containing all your specifications and (b) a pdf-file containing the group report on DTU Learn under *Assignments → Mandatory Assignment 2* **not later than 23:59 on Tuesday 29 November 2022**.

**Help:**

- The project work in your group must be solved **without help** from persons outside the group and without re-using other person's solutions. In particular groups must not collaborate.
- There will be **question time** (only) on the remaining Mondays of the semester. The exact time schedule for this can be found on DTU Learn.

  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.

**Suggested work plan:**

- Week 10: Make the tasks described in Section 3 (Formal Specification of Nets) and test the `NET` functions as described in Section 5 (Testing the Specifications). Write the report sections on this.
- Week 11: Make the tasks described in Section 4 (Formal Specification of Time Tables) and write the report section on this.

- Week 12: Test the `TIMETABLE` functions as described in Section 5 (Testing the Specifications) and write the report section on this. Make final reporting.
- Week 13: All group members review the report. One submit your report and formal specifications.

**Rules about sharing of work:**

1. Shared (not individualised) work: All persons must in collaboration define and explain the *Net* type, observers and generators, as well as the *TimeTable* type, observers and generators.
2. Shared (not individualised) work: All persons must in collaboration identify and informally describe *Net* requirements and *TimeTable* requirements.
3. Each person must contribute to formalising (i.e. writing the formal specification of) some parts of the *Net* requirements in functions. It is preferred that you each make an equal contribution to this, e.g. if you are 4 persons, you should each contribute with 25 % .[1] It is ok to collaborate about this.
4. Each person must contribute to formalising (i.e. writing the formal specification of) some parts of the *TimeTable* requirements in functions. It is preferred that you each make an equal contribution to this, e.g. if you are 4 persons, you should each contribute with 25 % .[2] It is ok to collaborate about this.
5. Each person must contribute to some parts of the test specifications. It is preferred that you each make an equal contribution to this, e.g. if you are 4 persons, you should each contribute with 25 % . It is ok to collaborate about this.
6. Each person must review all specifications and documentation written by each of the other persons. So it is your joint responsiblity to help making all parts correct.

Hence, it is e.g. not acceptable, if three persons shared items 3-5 above, by making one item each. You must demonstrate that you can both specify requirements and tests. It is advisable that each person has approximately the same procentage contribution to all tasks as these may get different weights.

In the introduction of the report, you can fill the following table out specifying how many percentage each person made of the work of items 3-5 above.

| study numbers | s000001 | s000002 | s000003 | s000004 | s000005 |
|---|---|---|---|---|---|
| formal spec of Net requirements | | | | | |
| formal spec of TimeTable requirements | | | | | |
| specification and documentation of tests | | | | | |

Ensure that sum og procentages in each row is 100 % .

Example 1: If you are 4 persons in the group and have made approximately equal sharing of work, your table will look like this:

| | s000001 | s000002 | s000003 | s000004 | none |
|---|---|---|---|---|---|
| formal spec of Net requirements | 25 % | 25 % | 25 % | 25 % | na |
| formal spec of TimeTable requirements | 25 % | 25 % | 25 % | 25 % | na |
| specification and documentation of tests | 25 % | 25 % | 25 % | 25 % | na |

[1] Note, that some functions are simple, while other are complicated and require more work.

[2] Note, that some functions are simple, while other are complicated and require more work.

# 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.[3] 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.

Cap: int

min_d_time:int

min head_ :int

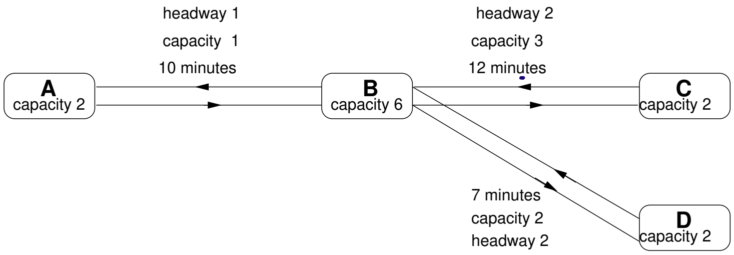
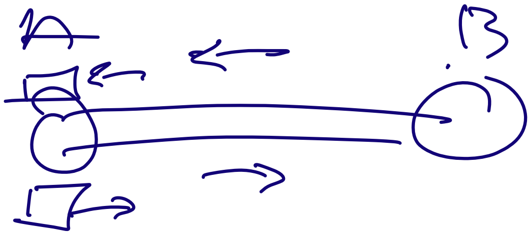
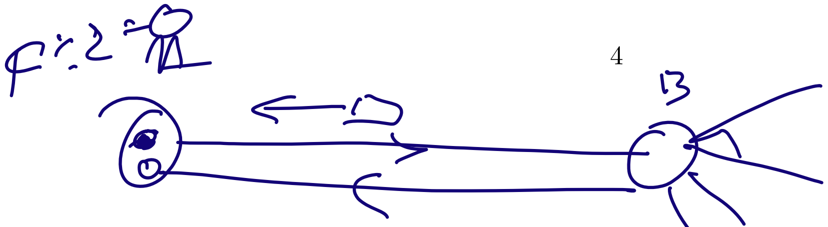


Figure 1: Example of a tram net.

Figure 1 shows an example of a net having four stops, A, B, C, and D with capacities 2, 6, 2, and 2, respectively. Furthermore, the net has three connections with minimum headways 1, 2 and 2, respectively, capacities 1, 3, and 2, respectively, and minimum driving times 10, 12, and 7, respectively.



[3] For 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 at all stops that it passes on its route. A tram need not to start and end at the same stop as it is the case in the example shown below.
- *Time* should be stated as whole minutes (e.g. 0, 14, and 60) (for simplicity in this assignment).

arrives at

leaves at

tram.name

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

| tram2 | | |
|---|---|---|
| D | 0 | 2 |
| B | 11 | 13 |
| A | 28 | 32 |
| B | 47 | 50 |
| D | 58 | 60 |

timetable

plan

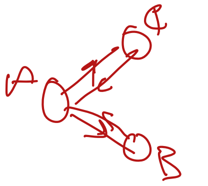
Figure 2: Example of a time table for the net shown in Figure 1.

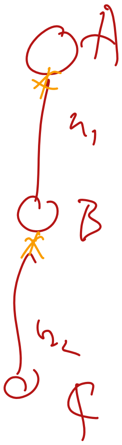
Figure 2 shows an example of a time table for two trams running in the net shown in Figure 1. The names of the trams are ”tram1” and “tram2”.

## 3 Formal Specification of Nets

On DTU Inside you can find a file `NET.rsl` 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      empty map [ ]

    −− insert a stop with a given name and capacity
    insertStop : StopId × Capacity × Net → Net
     ..., (StpId x Cap) → stop → Net w/no range yet

    −− add a connection between given stops,
    −− with the given minimum headway, capacity and minimum driving time   (What if stop B is not in Net?)
    addConnection : StopIdA × StopIdB × Headway × Capacity × DrivingTime × Net → Net
     ...
  value /∗ observers ∗/                 rang Net(SA) = range(stopA) + stopB   Net
    −− check whether a stop is in a network
    isIn : StopId × Net → Bool
     ...,

    −− get the capacity of a stop
    capacity : StopId × Net ⥲ Capacity
     ..., if stop in net → stop.capacity

    −− check whether two stops are directly connected in a network
    areDirectlyConnected : StopId × StopId × Net → Bool  (How to handle bad Nets?)
     ..., stp1 in range(stp2) ∧ stp2 in range(stp1)

  1 −− get minimum headway between two connected stops
    minHeadway : StopId × StopId × Net ⥲ Headway
     ...,
                        ~~max~~ minor not max?
  2 −− get the capacity for a connection between two connected stops
    capacity : StopId × StopId × Net ⥲ Capacity
     ...,

  3 −− get minimum driving time between two connected stops
    minDrivingTime : StopId × StopId × Net ⥲ DrivingTime
     ...

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

Need

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

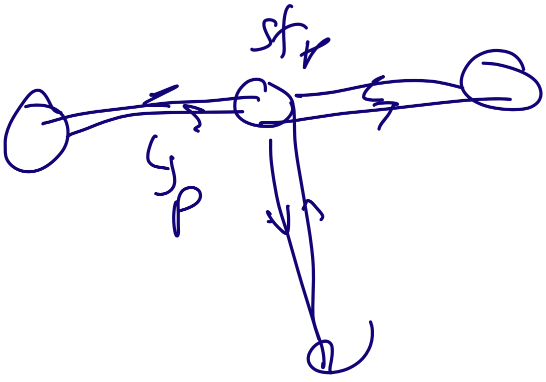
Stop : (stopId x capacity)

Connections: ( stop A x stop B x capacity x headway x drive_time )

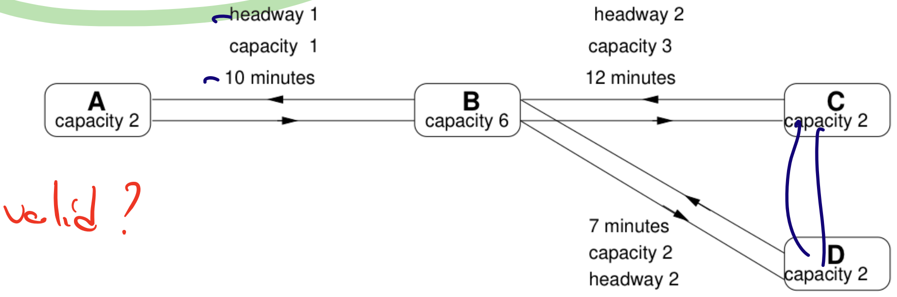
Net

Stop ⟼ {Conn-set}

$\forall C \in N(A) : outst == A$



$N(SA) = \{SA, SB\}$



net may is valid ?

## well formed

> dual-symmetry for all stop

if A → B then check B → A

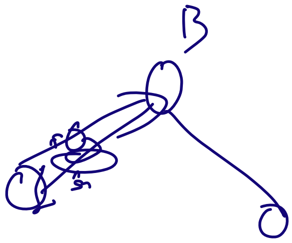
> no A → A

> if ¢ ∉ Net then no $ → ¢

> ∀ conns ; cap > 0

> for station S

S. cap =



> Capacitiy of st >= Σ Connections

## 4 Formal Specification of Time Tables

On DTU Inside 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 × 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
```

Time table

stop x arr x dep

leaus

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

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

Timet    tid x [stop x time x time]   ⟨(st, t, t), . . . ⟩

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. ( define wellformed )
   - 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.

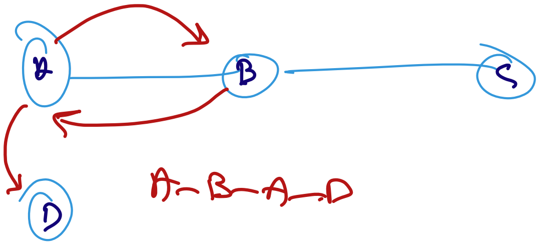
     The function `isWellformed` should be defined such that it checks all requirements.

2. Type check the module.

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

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

at dpt



Plan not empty

arrtime != deptime

∀ stop in plan need to be in network for given n

two adjacent stops in plan cannot be A-A

two adjacent stops need to be directly connected

for A B : stop

Com. dt <= B.at - A.dpt

for A: stop ∈ plan

A.dpt > A.at

dpt and at need to increase as the plan moves on

## 5 Testing the Specifications

Write one or several RSL modules that can be used to test your checking functions and other 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 your specifications.

for trains A,B $\curvearrowright$ $\overset{\text{from}}{\text{stop}}$ X Y: stop

"A goes before B"

$B.dpt >= A.dpt + C_{xy} . h$