A Formal Model for Railway Staff Rostering

Albena Kirilova Strupchanska, Martin Pěnička and Dines Bjørner
Computer Science and Engineering, Informatics and Mathematical Modelling
Technical University of Denmark
Building 322, Richard Petersens Plads, DK–2800 Kgs. Lyngby, Denmark
{albena|martin|db}@imm.dtu.dk

15 October, 2002

Abstract

The problem to be tackled is as follows: There is a railway net. Trains travel from station to station according to the schedule. There are depots in a railway net to which sets of staff members are associated. Staff members are responsible for performing set of actions in order to fulfil the schedule demands. Given a schedule, a staff type, a set of depots and rules the problem is to construct work schedules for staff members located in the depots such that they conform to the rules and the schedule demands. This problem is approached by dividing it into two subproblems: (i) staff scheduling: from a given schedule, staff type, depots and some rules to produce duties (sequence of actions) for staff members and (ii) staff rostering: generation of base rosters from the duties, constructed in the previous stage and assignment of particular staff members to them. Base rosters are cyclic sequences of duties for some planning period such that they conforms to rules and cover the duties. The assignment of staff members to base rosters is done such that each staff member receives a roster according his/her characteristics (abilities, previous duties etc.).

We relate this model to the descriptions provided in [3, 4, 5, 6, 8, 9].

Contents

1 Introduction

T:1, F:1
cros0p1

1.1 Synopsis

Staff planning is a typical problem arising in the management of large transport companies, including railway companies. It is concerned with building the work schedules (duties and rosters) for staff members needed to cover a planned timetable. Each work schedule is built concerning a given staff type (engine men, conductors, cater staff, etc.).

There are two types of staff planning: long-term planning and short-term planning. We will be interested in long term planning. Normally the long term planning task is separated into two stages: staff scheduling and staff rostering. Staff scheduling is concerned with building short-term working schedules, called duties, for staff members such that they satisfy schedule demands. After this stage it is easy to determine the global number of staff members needed to hire such that the working schedules could be performed. Staff rostering is concerned with ordering of duties into long-term working schedules, called base rosters, and assignment of specific staff members to them such that each staff member performs a roster. During the stage of rostering we have the assumption that we have enough hired staff members such that we could assign rosters to them.

In this paper we will try to explain and analyze first informally and then formally the problem. Using a formal methods approach and RAISE Specification Language we will present a formal model of the domain of staff rostering.

1.2 The Major Functions

Given a schedule, a staff type, a depot and rules the task is to produce a set of rosters. What we understand in terms of schedule, depot and rules can be found further in the paper.

gen_sross : SCH × StfTp × Dep × eRS → Ros

The function above produces all the rosters for a staff type per depot. Usually rosters are generated per depot and we have the assumption that after the staff scheduling stage all duties generated per depot are shifted to the depot. If this is not the case we propose a function that integrates the two stages in staff rostering into one. So given a schedule, a staff type, set of depots and rules we produce all rosters per each depot for this staff type.

obtain_ross : SCH × StfTp × Dep-**set** × eRS → Ros-**set**

1.3 Requirements and Software Design

We emphasis that we formally characterized schedules, duties and rosters to meet staff rostering demands. On the basic of such formal characterization we can now express software requirements.

The actual software design relies on identification of suitable operation research techniques, that can provide reasonable optimal solution at reasonable computing times.

It is not the aim of this paper to show such operation research algorithms. Instead we formalize the domain of railway staff rostering such that later we could apply to it operation research techniques discovered in further research work done within AMORE group.

1.4 Paper Structure

The rest of the paper is one section with five subsections. Each section consist of formal description of the problem (narrative) and formalization of it (formal model). The first subsection introduces the topology of the railway net from staff management perspective. The second one introduces the notion of a staff member and related to it characteristics taken into account in the early stage of planning. And finally the last three subsections are the ones which gradually show the creation of rosters from a schedule, set of depots and rules. The first of them is concerned of separating the journeys observed from a schedule into trips. The notions of journey and trip are introduced there. The second one introduces the notion of a duty and produces the set of duties per each depot. Finally the third one introduces more characteristics of staff members and the notion of rosters. It generates the rosters for staff members too.

2 Narrative & Formal Model

T:2, F:2

2.1 Nets, Stations and Depots

cros1

In this section we will introduce the notions of nets, stations and depots which are related to the topology of the railway net from a staff manager point of view.

2.1.1 Narrative

We take as base concept for the railway net the topology of that net. From a railway net (Net) we can observe stations (Sta) and depots (Dep). Depots are personnel bases i.e places where staff members are located. The notion of staff member will be introduced in more details in the next section. From a station we can observe a set of depots to which the station can belong. From a depot we can observe a set of stations from which it is easy to reach the depot. Given a depot and a station we can observe the distance in time (TInt) between them. We will be interested in these stations and depots which are 'close' to each other.

There are at least two stations in a net (α_1). There is at least one depot in a net (α_2). The set of depots observed from a station consists of depots of the same railway net (α_3). The set of stations observed from a depot consists of stations of the same railway net (α_4).

2.1.2 Formal Model

We first state some abstract types, ie. sorts, and some observer functions.

F:3

Abstract Types and Observers:

```
scheme NETWORK =
  class
    type Net, Sta, Dep, TInt, StaNm, DepNm
    value
      obs_Stas : Net → Sta-set,
      obs_StaNm : Sta → StaNm,
      obs_Deps : Net → Dep-set,
      obs_DepNm : Dep → DepNm,
      obs_StaDeps: Sta → Dep-set,
      obs_DepStas: Dep → Sta-set,
      obs_StDepDistance : Sta × Dep → TInt
  end
```

We will then illustrate some axioms:

Axioms :

(α_1) **axiom** ∀ n : Net • **card** obs_Stas(n) ≥ 2

(α_2) **axiom** ∀ n : Net • **card** obs_Deps(n) ≥ 1

(α_3) **axiom** ∀ n: Net • ∀ s : Sta • s ∈ obs_Stas(n) ⇒
    (∀ d : Dep • d ∈ obs_StaDeps(s) ⇒
      d ∈ obs_Deps(n))

(α_4) **axiom** ∀ n: Net • ∀ d : Dep • d ∈ obs_Deps(n) ⇒
    (∀ s : Sta • s ∈ obs_DepStas(d) ⇒
      s ∈ obs_Stas(n))

2.2 Staff members

We will introduce the notions of staff members and related to them attributes according to a staff manager stake-holder's perspective.

2.2.1 Narrative

We will call staff all the people who are employed in a railway company and who could perform some actions in order to fulfil a schedule demands.

At the first stage of staff rostering - staff scheduling we will be interested in a part of the characteristics that can be related to staff members. Staff members are exchangeable at staff scheduling stage that is why we will call them anonymous staff members (AnonStfMbr). From anonymous staff member we could observe his/her home depot (obs_SMDep). A home depot of some staff member is the depot of the railway net from where he/she starts and finishes his/her sequence of actions. There is a notion of a staff type (StfTp). Some possible staff types are: engine men (engS), conductors (condS), cater staff (catS) etc. From anonymous staff member we could observe his/her staff type (obs_SMStfTp). The set of anonymous staff members we will call anonymous staff (AnonStaff).

At the second stage of staff rostering we will take into account all the characteristics that can be related to a staff member. We assume that staff member's personal information makes him distinguishable from other staff members. So we will call specific staff member (SpecStfMbr) an anonymous staff member with added personal information. From a specific staff member we can observe his personal information as well as home depot and staff type.

From anonymous and specific staff member we can observe staff member's name. It makes the relation between two abstractions of a staff member - anonymous and specific.

Given a staff type we can observe all the depots which are home depots for staff members of a given staff type (function deps_staff below). Given a staff type and a depot we can observe all anonymous staff members at this depot of this staff type and respectively their number (functions dstft and dstft_num below).

2.2.2 Formal Model

F:4

We first state some types and some observer functions.

Abstract Types and Observers:

```
scheme STAFF =
  extend NETWORK with
  class
    type
      AnonStfMbr, Name,
      SpecStfMbr, PersInfo,
      StfTp == engS | condS | catS,
      AnonStaff = Name $\underset{m}{\rightarrow}$ AnonStfMbr,
      Staff = Name $\underset{m}{\rightarrow}$ SpecStfMbr

    value
      obs_Name: AnonStfMbr $\rightarrow$ Name,
    obs_Name: SpecStfMbr → Name,
    obs_SMStfTp : AnonStfMbr → StfTp,
    obs_SMStfTp: SpecStfMbr → StfTp,
    obs_SMDep : AnonStfMbr → Dep,
    obs_SMDep: SpecStfMbr → Dep,
    obs_PersInfo: SpecStfMbr → PersInfo

end
```

We will then illustrate some axioms and functions:

Functions and Axioms:

```
    proj_SpecAnonStfMbr: SpecStfMbr → AnonStfMbr
    proj_SpecAnonStfMbr(ssm) as asm
     post obs_SMStfTp(ssm) = obs_SMStfTp(asm) ∧
            obs_SMDep(ssm) = obs_SMDep(asm),

    proj_AnonSpecStfMbr: AnonStfMbr × PersInfo → SpecStfMbr
    proj_AnonSpecStfMbr(asm, pinf) as ssm
     post obs_Name(asm) = obs_Name(ssm) ∧
            obs_PersInfo(ssm) = pinf ∧
            obs_SMStfTp(asm) = obs_SMStfTp(ssm) ∧
            obs_SMDep(asm) = obs_SMDep(ssm),

  axiom ∀ asm: AnonStfMbr • ∃! ssm: SpecStfMbr •
    obs_Name(asm) = obs_Name(ssm)

  axiom ∀ ssm, ssm′: SpecStfMbr • ssm ≠ ssm′ ⇒
    proj_SpecAnonStfMbr(ssm) = proj_SpecAnonStfMbr(ssm′)

  value
    depStfMbrs : Dep → AnonStaff
    depStfMbrs(d) as astf
     post (∀ asm: AnonStfMbr •
       astf = [obs_Name(asm) ↦ asm] ∧ obs_SMDep(asm) = d),

    deps_staff : StfTp → Dep-set
    deps_staff(stft) ≡
       {d | d : Dep •  ∃ asm : AnonStfMbr •
             obs_SMStfTp(asm) = stft ∧ obs_SMDep(asm) = d},

    dstft : Dep × StfTp → AnonStaff
    dstft(d, stft) as astf
     post (∀ asm: AnonStfMbr • astf = [obs_Name(asm) ↦ asm] ∧
             obs_SMDep(asm) = d ∧ obs_SMStfTp(asm) = stft),
```

$\text{dstft\_num} : \text{Dep} \times \text{StfTp} \rightarrow \textbf{Nat}$
$\text{dstft\_num}(\text{d}, \text{stft}) \equiv \textbf{card dom } \text{dstft}(\text{d}, \text{stft}),$

$\text{dsstft\_grs} : \text{Dep-}\textbf{set} \times \text{StfTp} \rightarrow (\text{Dep} \times \textbf{Nat})\textbf{-set}$
$\text{dsstft\_grs}(\text{ds}, \text{stft}) \equiv \{(\text{dep}, \text{n}) \mid \text{dep} : \text{Dep}, \text{n} : \textbf{Nat} \bullet$
$\quad \text{dep} \in \text{ds} \wedge \text{n} = \text{dstft\_num}(\text{dep}, \text{stft})\}$

2.3 Schedule, Journeys and Trips

In this section we will explain the notions of schedule, journeys and trips that help us to introduce further the notion of duties.

2.3.1 Narrative

Schedule and Exchange stations: A schedule includes information about all train journeys such that each train journey is uniquely determined by a train number and a date and time. A train number is a unique identifier of a train which remains the same from the first to the last station of its journey. We don't consider train names here as not all the trains in a railway net have names.

Some of the stations in the net are special from staff management perspective because it is possible either to exchange staff members or a staff member to start or to finish his work there. We will call such stations exchange stations. From a station we could observe all the staff types for which this station is an exchange station (obs_ExchgStas). Given a station and a staff type we could check if the station is an exchange station or not for this staff type (is_exchgst). Exchange stations are located near the depots in the railway net.

Journeys and Trips: Staff members are responsible for performing some actions in order to fulfil the schedule demands. Some of the actions are related to train journeys. Train journeys could be both actual journeys with passengers or freights or empty trains journeys. A train journey is a sequence of rides with the same train number. A ride is characterized by a departure station, a departure time, an arrival station, an arrival time and a train between these two stations. Given a schedule we can extract a set of train journeys (journ_set).

There are some restrictions about the maximal working time for a staff member without a rest. Taking into account these restrictions it is natural to divide a journey into an indivisible pieces of work for staff members. That is why we introduce the notion of a trip. A trip is a sequence of rides of a train journey such that the first and the last station of a trip are exchange stations and the duration of a trip is less or equal to maximal allowed uninterrupted working time (maxUnIntWrkHr). Each trip is characterized by a train, a departure time, a departure station, an arrival time, an arrival station and possibly additional attributes. From a trip we can observe train characteristics for instance kind of the engine, staff types and their numbers needed to perform a trip etc.

2.3.2 Formal Model

F:5

Fist we will state some types(abstract and concrete) and some observer functions.

Abstract Types and Observers:

NETWORK, STAFF
scheme SCHEDULE =
extend STAFF **with**
class
  type
   Date, Hour, Trn, TrnId, LongDistance, Urban, ICE, TGV,
   StfAttr, NoStf,

```
 TrnChar = LongDistance| Urban| ICE| TGV,
 DateTime = Date × Hour,
 Ride′ == rd(sta: Sta, dt: DateTime, nsta: Sta, at: DateTime, trn: Trn),
 Ride = {|rd: Ride′ • wf_rd(rd)|},
 Journey′ = Ride*,
 Journey = {|j: Journey′ • wf_journ(j)|},
 Trip = Ride*,
 TrpAttr == Overnight|Other,
 SCH= TrnId $\overrightarrow{m}$ (DateTime $\overrightarrow{m}$ Journey)

value
 < : DateTime × DateTime → Bool, /∗ DateTime < DateTime∗/
 ≤: TInt × TInt → Bool, /∗TInt< TInt∗/
 − : DateTime × DateTime → TInt,
 − : TInt × TInt → TInt,
 ≤: DateTime × DateTime → Bool,
 ≥: TInt × TInt → Bool,

 consec_intime: DateTime × DateTime → Bool,
 obs_TrnId: Trn → TrnId,
 trnchr: Ride → TrnChar,
 stfchr: TrnChar → StfTp $\overrightarrow{m}$ Nat,
 obs_ExchgStas: Sta $\xrightarrow{\sim}$ StfTp-set,
 techTime: Sta × Trn × StfTp → TInt,
 maxUnIntWrkHr: StfTp → TInt,
 /∗ from a staff type (rules taken into account implicitly) we can observe
   the maximal permitted working time in minutes without a rest ∗/
 maxWrkHr: StfTp → TInt,
 /∗ from a staff type (rules taken into account implicitly) we can observe
   the maximal permitted working time∗/
 tripAttr: Trip → TrpAttr,

 wf_rd : Ride′ → Bool
 wf_rd(rd) ≡ dt(rd) < at(rd),

 wf_journ: Journey′ → Bool
 wf_journ(j) ≡
   (∀ i: Nat • {i, i + 1} ⊆ inds j ⇒
    obs_TrnId(trn(j(i))) = obs_TrnId(trn(j(i+1))) ∧
    nsta(j(i)) = sta(j(i+1)) ∧
    consec_intime(at(j(i)), dt(j(i+1)))),

 journ_set: SCH → Journey-set
 journ_set(sc) ≡ {j| j: Journey •(∀ trnid: TrnId, timdat: DateTime •
                                        trnid ∈ dom sc ∧
                                        timdat ∈ dom sc(trnid)⇒
```

```
                        j=sc(trnid)(timdat))},

  journ_set1: SCH → Journey-set
  journ_set1(sc) ≡ ∪ {rng tn| tn: (DateTime $\overrightarrow{m}$ Journey) •
                                              tn ∈ rng sc}
end
```

Each train journey is divided into trips with subject to a staff type. The following is a function that divides a journey into trips.

```
 trip_list : Journey × StfTp → Trip*
  trip_list(j, stft) as trpl
    post (∀ i : Nat • i ∈ inds trpl ⇒
              wf_stft_trip(trpl(i), stft)) ∧
              check_separation(trpl, stft),
```

A trip is well formed if it consists of consecutive rides, the first and the last stations of a trip are exchangeable stations and the train during the trip has the same characteristics from a staff member perspective.

```
  wf_stft_trip: Trip × StfTp → Bool
  wf_stft_trip(trp , stft) ≡
    is_exchgst(trip_fsta(trp), stft) ∧
    is_exchgst(trip_lsta(trp), stft) ∧
    ∼(possible_exchg_inside(trp, stft)) ∧
    trip_fnT(trp) − trip_stT(trp) ≤ maxUnIntWrkHr(stft) ∧
    same_trn(trp, stft),

  is_exchgst: Sta × StfTp → Bool
  is_exchgst(s, stft) ≡ stft ∈ obs_ExchgStas(s),

  possible_exchg_inside: Trip × StfTp → Bool
  possible_exchg_inside(trp, stft) ≡
    (∀ i: Nat • i ∈ {1..len trp −1} ⇒
       if is_exchgst(nsta(trp(i)), stft) then
        dt(trp(i + 1)) − at(trp(i)) ≥ tech_time(trp(i), stft)
        else false end),

  same_trn: Trip × StfTp → Bool
  same_trn(trp, stft) ≡
    (∀ i: Nat • {i, i + 1} ⊆ inds trp ⇒
     same_trnchr(trnchr(trp(i)),
                     trnchr(trp(i + 1)), stft)),


  same_trnchr: TrnChar × TrnChar × StfTp → Bool,
  /*checks if two trains are with the same characteristics from the staff point of view*/
```

```
check_separation: Trip* × StfTp → Bool
check_separation(trpl, stft) ≡
(∀ i : Nat • {i, i + 1} ⊆ inds trpl ⇒
  coincident_sta(trpl(i), trpl(i + 1)) ∧
  div_sta(trpl(i), trpl(i + 1), stft)),

coincident_sta: Trip × Trip → Bool
coincident_sta(trp1, trp2) ≡
  trip_lsta(trp1) = trip_fsta(trp2),
```

On the station where we separate the train journey there should be enough time for exchanging the staff members or a staff member to change a train. The time interval between departure and arrival time of a train at this station should be greater or equal to the technical time. Technical time is the smallest interval of time for which it is possible to exchange staff members or a staff member to change a train.

```
div_sta: Trip × Trip × StfTp→ Bool
div_sta(trp1, trp2, stft) ≡
  trip_stT(trp2) − trip_fnT(trp1) ≥ tech_time(last(trp1), stft),

tech_time: Ride × StfTp → TInt
tech_time(rd, stft) ≡
  techTime(sta(rd), trn(rd), stft),

last: Trip →~ Ride
last(trp) ≡ trp(len trp)
  pre len trp ≥ 1,
```

Finally given a schedule and a staff type we produce the trip set such that each journey that can be extracted from a schedule is divided into trips.

```
gen_tripss: SCH × StfTp → Trip-set
gen_tripss(sc, stft) ≡ ∪ {trips| trips: Trip-set •
                                    trips = gen_trips(sc, stft)},

gen_trips : SCH × StfTp → Trip-set
gen_trips(sc, stft) as trps
  post (∀ j: Journey • j ∈ journ_set(sc) ⇒
            trps = elems trip_list(j, stft))
```

The following are some functions that extract some characteristics of a trip.

```
trip_stT: Trip → DateTime
trip_stT(trp) ≡ dt(hd trp),

trip_fnT: Trip → DateTime
trip_fnT(trp) ≡ at(last(trp)),
```

trip_fsta: Trip $\rightarrow$ Sta
trip_fsta(trp) $\equiv$ sta(**hd** trp),

trip_lsta: Trip $\rightarrow$ Sta
trip_lsta(trp) $\equiv$ nsta(last(trp)),

trip_trn: Trip $\rightarrow$ Trn
trip_trn(trp) $\equiv$ trn(**hd** trp),

trip_trnchr: Trip $\rightarrow$ TrnChar
trip_trnchr(trp) $\equiv$ trnchr(**hd** trp),

trip_stfchr: Trip $\rightarrow$ StfTp $\underset{m}{\rightarrow}$ **Nat**
trip_stfchr(trp) $\equiv$ stfchr(trip_trnchr(trp)),

trip_WrkTm: Trip $\rightarrow$ TInt
trip_WrkTm(tp) $\equiv$ trip_fnT(tp) $-$ trip_stT(tp)

2.4 Actions and Duties

In this section we will explain the notion of duties.

2.4.1 Narrative

Actions: Each staff member performs some actions. Actions could be sequence of trips, rests and some human resource activities. Rests could be rest between trips, meal rests, rests away from home depot including sleeping in dormitories (external rest) etc. By human resource activities we mean activities performing from a staff member in order to increase his qualification (seminars, courses etc.).

The sequence of trips is characterized with a start time, an end time and a list of rides. A rest is characterized by a start and an end time, station name and also some attributes. We will assume that a rest starts and ends at the same station. Human resource activities has the same characteristics as rests.

Duties: Each staff member is related to a given depot, home depot, in a railway net, which represents starting and ending point of his work segments. A natural constraint imposes that each staff member must return to his home depot within some period of time. This leads to the introduction of the concept of duty as a list of actions spanning L consecutive days such that its start and end actions are related to the same depot. A duty conforms to some rules and satisfy some requirements like continuance, working hours per duty etc. Each duty is concerned with members of the same staff type. From a duty we can observe duty attributes for example: 'duty with external rest', 'overnight duty', 'heavy overnight duty', 'long duty' etc. Also each duty has some characteristics as:

1. Start time: it is given explicitly when the first action of a duty is either rest or human resource activity; in case of a trip it is defined as the departure time of its first ride minus the sum of technical departure time and briefing time,
2. End time: it is given explicitly when the last action of a duty is either rest or human resource activity; in case of a trip it is defined as the arrival time of its last ride plus the sum of technical arrival time and debriefing time,
3. Paid time: it is defined as the elapsed time from the start time to the end time of the duty,
4. Working time: it is defined as the duration of the time interval between the start time and the end time of the duty, minus the external rest, if any.

Mentioned above characteristics are common for every duty. There are other possible characteristics of a duty but they strictly depend on a staff type. For instance taking into account engine men staff type we could observe:

5. Driving time: it is defined as the sum of the trip durations plus all rest periods between consecutive trips which are shorter than M minutes e.g. 30 minutes,

Duties attributes and characteristics are taken into account in scheduling process while selecting feasible, efficient and acceptable duties per each depot and in sequencing duties into rosters. This will be introduced in the next sections.

Given the schedule, staff type, set of depots and rules we can generate duty sets per each depot.

2.4.2 Formal Model

F:6

First we will state some types and observer functions.

Abstract Types and Observers:

```
scheme DUTY =
extend SCHEDULE with
class
   type
    RestAttr, HRAttr, DtChar,
    Ac == mk_trip(st: DateTime, tripl: Trip*, et: DateTime)|
            mk_rest(sr: DateTime, rsta: Sta, ratt: RestAttr, er: DateTime)|
            mk_hra(sh: DateTime, hsta: Sta, hatt: HRAttr, eh: DateTime),
    Duty = Ac*,
    DtAttr==ExtRest|Long|Overnight|HeavyOvernight,
    AcR = Ac × StfTp → Bool,
    AcRS = AcR-set,
    DuR = Duty × StfTp → Bool,
    DuRS = DuR -set,
    DepR = Dep × Duty-set × StfTp→ Bool,
    DepRS = DepR-set,
    OvDR = (Duty-set)-set × StfTp → Bool,
    OvDRS = OvDR-set,
    RS == check_acr(ar: AcRS)|check_dur(dur: DuRS)|
             check_dpr(dpr: DepRS)|check_ovdsr(ovdsr: OvDRS)
   value
    dt_maxlenght: StfTp → TInt,
    dt_char: Duty → DtChar,
    dt_attr: Duty → DtAttr
end
```

Each duty is generated taking into account some depot and some staff type. The following is a function which generates a duty set for a depot. It generates all possible duties for the depot.

```
gendep_dutys : Trip-set × StfTp × Dep × RS → Duty-set
gendep_dutys(trps, stft, dep, rs) as ds
   post (∀ d: Duty • d ∈ ds ⇒
            d = gen_duty(trps, stft, dep, rs)) ∧
     ∼(∃ d′: Duty • d′ = gen_duty(trps, stft, dep, rs) ∧
        d′ ∉ ds),
```

Each duty has to start and to end at the same depot and has to conform some rules. Rules are related to the sequence of actions in a duty, maximal number of actions with a given characteristics, rest time between actions, overall rest time, overall working time etc. These rules we will call rules at a duty level. Given a trip set, a staff type, a depot and rules we can generate a duty for the depot. The function below generates a duty such that its fist and its last action starts and respectively finishes at the depot, the depot is a home depot for staff members of the given staff type and the duty satisfy the rules.

```
gen_duty : Trip-set × StfTp × Dep × RS → Duty
gen_duty(trps, stft, dep, srs) as d
  post is_wfd(d, stft, srs) ∧ ac_dep(hd d, stft) = dep ∧
   dt_endt(d) − dt_startt(d) ≤ dt_maxlenght(stft) ∧
   (∃ trpl : Trip* • belong(trpl, d) ⇒
                                          trip_stft(trpl, stft, dep)),

is_wfd: Duty × StfTp × RS → Bool
is_wfd(dt, stft, rs) ≡
  ac_dep(hd dt, stft) = ac_dep(dt(len dt), stft) ∧
   comp_dtTrips(dt, stft) ∧ conf_dt_rules(dt, stft, rs),

ac_dep : Ac × StfTp ~→ Dep
ac_dep(ac, stft) as dep
  post (∃ dep′:Dep •
   case ac of
     mk_trip(st, tripl, et) →
             dep ∈  st_stftdep(sta(hd (hd tripl)), stft),
     mk_rest(sr, rsta, ratt, er) →
             dep ∈  st_stftdep(rsta, stft),
     mk_hra(sh, hsta, hatt, eh) →
           dep ∈ st_stftdep(hsta, stft)
    end
  ∧ dep = dep′),

  st_stftdep: Sta × StfTp → Dep-set
  st_stftdep(st, stft) ≡
   {dep| dep: Dep • dep ∈ obs_StaDeps(st)∧
     is_exchgst(st, stft)},

  /∗ checks if all the trips in a duty has the same characteristics from staff point of view ∗/
  comp_dtTrips: Duty × StfTp → Bool
  comp_dtTrips(dt, stft) ≡
   (∀ i: Nat • i ∈ inds dt ⇒
     case dt(i) of
       mk_trip(sti, tripli, eti) →
       (∀ j: Nat • j ∈ inds dt ∧ j ≠ i ⇒
        case dt(j) of
          mk_trip(stj, triplj, etj) →
           same_trpchr(hd tripli, hd triplj, stft)
        end)
       end),
same_trpchr: Trip × Trip × StfTp → **Bool**
same_trpchr(trp1, trp2, stft) ≡
 same_trnchr(trip_trnchr(trp1), trip_trnchr(trp2), stft),

conf_dt_rules: Duty × StfTp × RS → Bool
conf_dt_rules(dt, stft, rs) ≡ satf(dt, stft, rs) ∧
 (∀ i : Nat • i ∈ inds dt ⇒ conf_ac(dt(i), stft, rs)),

conf_ac: Ac × StfTp × RS → Bool
conf_ac(ac, stft, rs) ≡
 case rs of
    check_acr(acrs) → (∀ acr: AcR • acr ∈ acrs ⇒ acr(ac, stft))
 end,

/*checks if the rules for sequencing actions in a duty are satisfied*/
satf: Duty × StfTp × RS → Bool
satf(dt, stft, rs) ≡
 case rs of
    check_dur(durs) → (∀ dur:DuR • dur ∈ durs ⇒ dur(dt, stft))
 end,

belong: Trip* × Duty → Bool
belong(tpl,dt) ≡ (∃ ac: Ac • ac ∈ elems dt ∧
 case ac of
    mk_trip(st,tpl,et) → true
 end),

trip_stft: Trip* × StfTp × Dep → Bool
trip_stft(trpl, stft, dep) ≡
    let stfm = trip_stfchr(hd trpl) in
     stft ∈ dom stfm ∧ dstft_num(dep, stft) > 0 end,
```

The set of all duties for a depot has to obey to some rules too. The rules/restrictions could be related to a maximal number of duties with specific characteristics per depot, maximal number of duties per depot etc. We will call these rules rules on a depot level.
The function below selects a subset of a duty set, generated on previous stage, such that it satisfies the rules on the depot level and there is enough staff at the depot to perform the duty set.

```
seldep_dutys : Trip-set × StfTp × Dep × RS → Duty-set
seldep_dutys(trps, stft, dep, rs) as ds
 post let ds1 = gendep_dutys(trps, stft, dep, rs) in
           ds ⊆ ds1 ∧ conf_dts_deprules(dep, ds, stft, rs) ∧
    enough_staff(ds, stft, dep) end,

conf_dts_deprules: Dep × Duty-set × StfTp × RS → Bool
```

```
conf_dts_deprules(dep,ds,stft, rs) ≡
 case rs of
       check_dpr(dprs) → (∀ dpr: DepR • dpr ∈ dprs ⇒ dpr(dep, ds, stft))
     end,

enough_staff: Duty-set × StfTp × Dep → Bool
enough_staff(ds, stft, dep) ≡
 dutys_staff_numb(ds, stft) ≤ dstft_num(dep, stft),

dutys_staff_numb: Duty-set × StfTp → Nat,
/*the number of people should be equal to the number of duties,
but in case of a conductor staff type the number of people may
be more than the number of duties as two conductors may have the same duties*/
```

Finally given a trip set, a staff type, a depot set and rules we can generate a set of duties per each depot.

```
gen_dutys : Trip-set × StfTp × Dep-set × RS → (Duty-set)-set
gen_dutys(trps, stft, deps, rs) as dss
 post (∀ ds: Duty-set • ds ∈ dss ⇒
           (∃! dep: Dep • dep ∈ deps ∧
              ds = seldep_dutys(trps, stft, dep, rs))) ∧
            card dss = card deps,
```

The union of generated sets of duties per each depot has to conform to some overall rules e.g. the number of duties as a whole with a given characteristics not to exceed some defined number etc. Also the generated duties as a whole has to cover all the trips that can be observed from a schedule. Finally given a schedule, a staff type, set of depots and rules we can generate set of duties per each depot such that the mentioned above constraints are satisfied.

```
sel_dutyss : SCH × StfTp × Dep-set × RS → (Duty-set)-set
sel_dutyss(sc, stft, deps, rs) as dss
 post let trps = gen_tripss(sc, stft) in
            dss = gen_dutys(trps, stft, deps, rs) ∧ cover(dss, trps)
     ∧ conf_dts_ovr(dss, stft, rs)
          end,

cover : (Duty-set)-set × Trip-set → Bool,

conf_dts_ovr: (Duty-set)-set  × StfTp × RS → Bool
conf_dts_ovr(dss, stft, rs) ≡
 case rs of
    check_ovdsr(ovdsrs) → (∀ ovdsr: OvDR • ovdsr ∈ ovdsrs ⇒
                                       ovdsr(dss, stft))
 end,
```

The following are some functions concerning a duty and its characteristics.

```
duty_dep: Duty × StfTp → Dep
duty_dep(dt, stft) as dep
 post dep ∈ st_stftdep(dt_fsta(dt),stft),

dt_fsta: Duty → Sta
dt_fsta(dt) ≡
 case hd dt of
    mk_trip(_, tripl, _) → trip_fsta(hd tripl),
    mk_rest(_, rsta, _, _) → rsta,
    mk_hra(_, hsta, _, _) → hsta
end,

dt_lsta: Duty → Sta
dt_lsta(dt) ≡
 case dt(len dt) of
    mk_trip(_, tripl, _) → trip_fsta(hd tripl),
    mk_rest(_, rsta, _, _) → rsta,
    mk_hra(_, hsta, _, _) → hsta
end,

dt_starttime: Duty → DateTime
dt_starttime(dt) ≡
 case hd dt of
    mk_trip(st, tripl, et) → st,
    mk_rest(sr, rsta, ratt, er) → sr,
    mk_hra(sh, hsta, hatt, eh) → sh
 end,

dt_endtime: Duty → DateTime
dt_endtime(dt) ≡
 case dt(len dt) of
    mk_trip(st, tripl, et) → et,
    mk_rest(sr, rsta, ratt, er) → er,
    mk_hra(sh, hsta, hatt, eh) → eh
 end,

duty_stft_num: Duty → StfTp $\overrightarrow{m}$ Nat
duty_stft_num(dt) as stfm
 post (∃ trpl: Trip* • belong(trpl, dt) ⇒
    stfm = trip_stfchr(hd trpl))
```

2.5 Rosters and Staff Members

In this section we will explain the notion of a roster and how it is related to staff members.

2.5.1 Narrative

Rosters: During the second stage of staff rostering the duties generated at previous stage are ordered in rosters which are long term working schedules assigned to specific staff members. For each depot in a depot set, a separate staff rostering problem is solved considering only the corresponding duties. We will introduce two help notions in order to explain the concept of roster and its stages of generation.

A plan roster is a sequence of duties generated for anonymous staff members of the same staff type. A base roster is a cyclic sequence of a plan roster such that it spans trough a planning period determined by a schedule. In other words, a plan roster is that part of the base roster which is repeated several times and a base roster is just a cyclic sequence of duties. Each base roster has to satisfy some rules. The rules are about the order of duties in a consecutive days and their attributes. Also there are some constraints concerning number of duties in a base roster with determined attributes. These rules we will call conventionally rules at the roster level.

So given a schedule, a staff type, a depot and rules we can generate base rosters for the given depot. These base rosters have to cover all the duties corresponding to this depot and have to conform to some rules. The rules at this level we will call conventionally rules at the overall roster level.

All the duties in a base roster has to be performed by a specific staff member. We will call roster a cyclic sequence of duties (base roster) for a specific staff member such that he/she could perform them. So from a base roster and a staff type we can generate rosters. The number of staff members assigned to the base roster is equal to the length of the plan roster. All staff members perform the base roster but starting at a different day.

Staff Members: During the assignment of duties in a base roster to staff members we consider specific staff members. At this stage we are working with specific staff members as we are interested in their personal information. From a staff member personal information we could observe his/her private information (obs_PrInf) as date of birth, place of living, address etc. Also we could observe his qualification (obs_Qual), special work requirements (obs_SpWrkReq) and the list of his/her previous duties (obs_PrevDuty).

Given a base roster and a staff member we can observe his roster which is considered to his/her attributes.

2.5.2 Formal Model

F:7

Abstract Types and Observers: We first state some types(abstract and concrete) and some observer functions.

```
scheme ROSTER =
  extend DUTY with
  class
    type
      Info, WrkReq, Qualification,
    PlRos = Duty$^*$,
    BRos = PlRos $\times$ **Nat**,
    RoR = PlRos $\times$ StfTp $\to$ **Bool**,
    RoRS = RoR-**set**,
    OvR = BRos $\times$ StfTp $\to$ **Bool**,
    OvRS = OvR-**set**,
    eRS == RS | check_ror(rrs : RoRS) | check_ovrs(ovrs : OvRS),
    Ros = SpecStfMbr $\overrightarrow{m}$ BRos

  **value**
    f : eRS $\to$ RS,
    obs_PrInf : PersInfo $\to$ Info,
    obs_SpWrkReq : PersInfo $\to$ WrkReq,
    obs_PrevDuty : PersInfo $\to$ Duty$^*$,
    obs_PostDuty : PersInfo $\to$ Duty$^*$,
    obs_Qualf : PersInfo $\to$ Qualification,
    obs_PlPer : SCH $\to$ **Nat**,

    bros_length : BRos $\to$ **Nat**
    bros_length(bros) $\equiv$
     **let** (plros, rnumb) = bros **in**
       **len** plros **end**
**end**
```

The following function generates all possible base rosters for a given duty set (related to a depot).

```
    gen_dep_bross : SCH $\times$ StfTp $\times$ Dep $\times$ eRS $\to$ BRos-**set**
    gen_dep_bross(sc, stft, dep, rs) **as** bross
     **post** ($\forall$ bros : BRos $\bullet$
              bros $\in$ bross $\Rightarrow$
                bros = genbros_dep(sc, stft, dep, rs)) $\wedge$
       $\sim$($\exists$ bros$'$ : BRos $\bullet$
             bros$'$ = genbros_dep(sc, stft, dep, rs) $\wedge$
             bros$'$ $\notin$ bross),

    genbros_dep : SCH $\times$ StfTp $\times$ Dep $\times$ eRS $\to$ BRos
    genbros_dep(sc, stft, dep, rs) **as** bros **post**
      **let** ds = dep_dutyset(dep, stft) **in**
        cover_rds(bros, ds)
      **end** $\wedge$ wf_bros(bros, sc, stft, rs),

    cover_rds : BRos $\times$ Duty-**set** $\to$ **Bool**,

    wf_bros : BRos $\times$ SCH $\times$ StfTp $\times$ eRS $\to$ **Bool**
    wf_bros(bros, sc, stft, rs) $\equiv$
      **let** (plros, rnumb) = bros **in**
    same_qualific(plros, stft) ∧
    conform_cplrosrs(plros, stft, rs) ∧
    len plros ∗ rnumb = obs_PlPer(sc)
    end,

same_qualific : PlRos × StfTp → Bool
same_qualific(plros, stft) ≡
    (∀ i : Nat • {i, i + 1} ⊆ inds plros ⇒
     sm_qual(plros(i), plros(i + 1))),

sm_qual : Duty × Duty → Bool,

conform_cplrosrs : PlRos × StfTp × eRS → Bool
conform_cplrosrs(plros, stft, rs) ≡
    conform_plrosrs(plros, stft, rs) ∧
    let cycros = ⟨plros(len plros)⟩ ^ ⟨hd plros⟩ in
        conform_plrosrs(cycros, stft, rs)
    end,

conform_plrosrs : PlRos × StfTp × eRS → Bool
conform_plrosrs(plros, stft, rs) ≡
    case rs of
        check_ror(rrs) →
            (∀ rr : RoR • rr ∈ rrs ⇒ rr(plros, stft))
    end,
```

The generated, on previous stage, set of base rosters has to conform to some rules as maximal percentage of base rosters with particular characteristics etc.

```
sel_dep_bross: SCH × StfTp × Dep × eRS → BRos-set
sel_dep_bross(sc, stft, dep, rs) as bross
 post let bross1 = gen_dep_bross(sc, stft, dep, rs) in
          bross ⊆ bross1 ∧
          conform_bros_rules(bross, stft, rs)
         end,

conform_bros_rules : BRos-set × StfTp × eRS → Bool
conform_bros_rules(bross, stft, rs) ≡
    (∀ bros: BRos • bros ∈ bross ⇒
    case rs of
        check_ovrs(ovrs) →
            (∀ ovr : OvR •
                ovr ∈ ovrs ⇒ ovr(bros, stft))
    end),
```

Having a base roster and a staff type and a depot we can produce rosters for the specific staff members of the given staff type.

```
gen_ssmros : BRos × StfTp × Dep → Ros
gen_ssmros(bros, stft, dep) as ros
 post let sms = dstft_gr(dep, stft) in
          ros = assignment(bros, sms) ∧
          card dom ros = bros_length(bros)
         end,

dstft_gr : Dep × StfTp → Staff
dstft_gr(dep, stft) ≡
   let anstaff = dstft(dep, stft) in
      get_staff(anstaff)
   end,

get_staff: AnonStaff → Staff
get_staff(anstaff) as staff
 post (∀ asm: AnonStfMbr • asm ∈ rng anstaff ⇒
          (∃! ssm: SpecStfMbr • ssm ∈ rng staff ∧
             obs_Name(asm) = obs_Name(ssm)))
```

Given a base roster and staff we assign specific staff members to the base roster such that we receive a set of rosters. The number of rosters is equal to the length of the base roster. All the rosters are permutations of the base roster. So at this stage of planning we assign specific staff members to duties in the plan roster (cyclic part of the base roster).

```
assignment : BRos × Staff → Ros
assignment(bros, staff) as ros
 post (∀ dt : Duty • duty_in_bros(dt, bros)⇒
          (∃! ssm : SpecStfMbr • ssm ∈ dom ros ∧
             dt = first_bros_duty(ros(ssm)) ∧
             conform_rsm(ros(ssm),ssm) ∧ permutation(ros(ssm), bros)))
   pre card rng staff > bros_length(bros),

duty_in_bros: Duty × BRos → Bool
duty_in_bros(dt, bros) ≡
   let (plros, rnumb) = bros in
    dt ∈ elems plros end,

first_bros_duty: BRos → Duty
first_bros_duty(bros) ≡
 let (plros, rnumb) = bros in
   hd plros end,
```

Each roster is assigned to a specific staff member according to his/her qualification, special work requirements and previous duties such that he/she could perform it.

```
conform_rsm : BRos × SpecStfMbr → Bool
conform_rsm(bros, ssm) ≡
```

```
    satisfy_qual(bros, obs_Qualf(obs_PersInfo(ssm))) ∧
    satisfy_predt(bros, obs_PrevDuty(obs_PersInfo(ssm))) ∧
    satisfy_swr(bros, obs_SpWrkReq(obs_PersInfo(ssm))),

satisfy_qual : BRos × Qualification → Bool,
satisfy_predt : BRos × Duty* → Bool,
satisfy_swr : BRos × WrkReq → Bool,

permutation: BRos × BRos → Bool,
```

Finally we generate the rosters for the given depot and staff type such that for each base roster generated at the previous stage we generate rosters.

```
gen_sross : SCH × StfTp × Dep × eRS → Ros
gen_sross(sc, stft, dep, rs) as ros
 post let bross = sel_dep_bross(sc, stft, dep, rs) in
    (∀ bros : BRos • bros ∈ bross ⇒
     ros = gen_ssmros(bros, stft, dep))
    end,
```

All rosters are generated taking into account a staff type. So using the function above we can generate all rosters per depot for all staff types related to this depot. In this case to generate rosters per depot we will need only the schedule, the depot and the rules.

```
dep_rosters : SCH × Dep × eRS → StfTp →m Ros
dep_rosters(sc, dep, rs) as stft_ross
 post (∃! stft : StfTp •
            stft ∈ dep_stftypes(dep) ⇒
               let rset = gen_sross(sc, stft, dep, rs) in
                stft_ross = [ stft ↦ rset ]
            end) ∧
    card dep_stftypes(dep) = card dom stft_ross,

dep_stftypes : Dep → StfTp-set
dep_stftypes(dep) ≡ {stft| stft: StfTp • ∃ ssm: SpecStfMbr • obs_SMDep(ssm) = dep},
```

Base rosters and respectively rosters are generated per depot and we have the assumption that after the staff scheduling stage all duties generated per depot are shifted to the depot. If this is not the case we could observe all the duties generated in staff scheduling stage per depot (dep_dutyset) which will help us to integrate the two stages in staff planning into one. So given a schedule, a staff type, a set of depots and rules we will produce all rosters per each depot in the depot set for the given staff type.

```
obtain_ross : SCH × StfTp × Dep-set × eRS → Ros-set
obtain_ross(sc, stft, deps, rs) as rosset
 post let dtss = sel_dutyss(sc, stft, deps, f(rs)) in
            (∀ ross : Ros • ross ∈ rosset ⇒
               (∃! dep : Dep • dep ∈ deps ⇒
                    ross = gen_sross(sc, stft, dep, rs) $\wedge$
                    dep_dutyset(dep, stft) $\in$ dtss)) **end** $\wedge$
                  **card** rosset = **card** deps,

     dep_dutyset: Dep $\times$ StfTp $\to$ Duty-**set**
     dep_dutyset(dep, stft) $\equiv$
      {dt| dt: Duty $\bullet$ dep = duty_dep(dt, stft)}
```

The rest is a small part of the possible functions for operating with staff members in depots.

```
hire_sm: SpecStfMbr $\times$ Staff $\xrightarrow{\sim}$ Staff
hire_sm(ssm, stf) $\equiv$ stf $\cup$ [obs_Name(ssm) $\mapsto$ ssm]
  **pre** ($\forall$ ssm$'$: SpecStfMbr $\bullet$ ssm$'$ $\in$ **rng** stf $\Rightarrow$
        obs_Name(ssm$'$) $\neq$ obs_Name(ssm)) $\wedge$ ssm $\notin$ **rng** stf,

fire_sm: SpecStfMbr $\times$ Staff $\xrightarrow{\sim}$ Staff
fire_sm(ssm, stf) $\equiv$ stf \ {obs_Name(ssm)}
  **pre** obs_Name(ssm) $\in$ **dom** stf,

hired_sm: SpecStfMbr $\times$ Staff $\to$ **Bool**
hired_sm(ssm, stf) $\equiv$ ssm $\in$ **rng** stf,

add_specsm: AnonStfMbr $\times$ PersInfo $\times$ Name $\to$ SpecStfMbr
add_specsm(asm, pinf, nm) **as** ssm
  **post** obs_Name(asm) = nm $\wedge$
   obs_SMStfTp(asm) = obs_SMStfTp(ssm) $\wedge$
   obs_SMDep(asm) = obs_SMDep(ssm) $\wedge$
   obs_PersInfo(ssm) = pinf,

get_specsm : AnonStfMbr $\times$ PersInfo $\to$ SpecStfMbr
get_specsm(asm, pinf) **as** ssm
  **post** obs_Name(asm) = obs_Name(ssm) $\wedge$
  obs_PersInfo(ssm) = pinf,

dep_staff : Dep $\to$ Staff
dep_staff(dep) $\equiv$
  **let** anstaff = depStfMbrs(dep) **in**
   get_staff(anstaff)
  **end**
```

2.6 Apendix

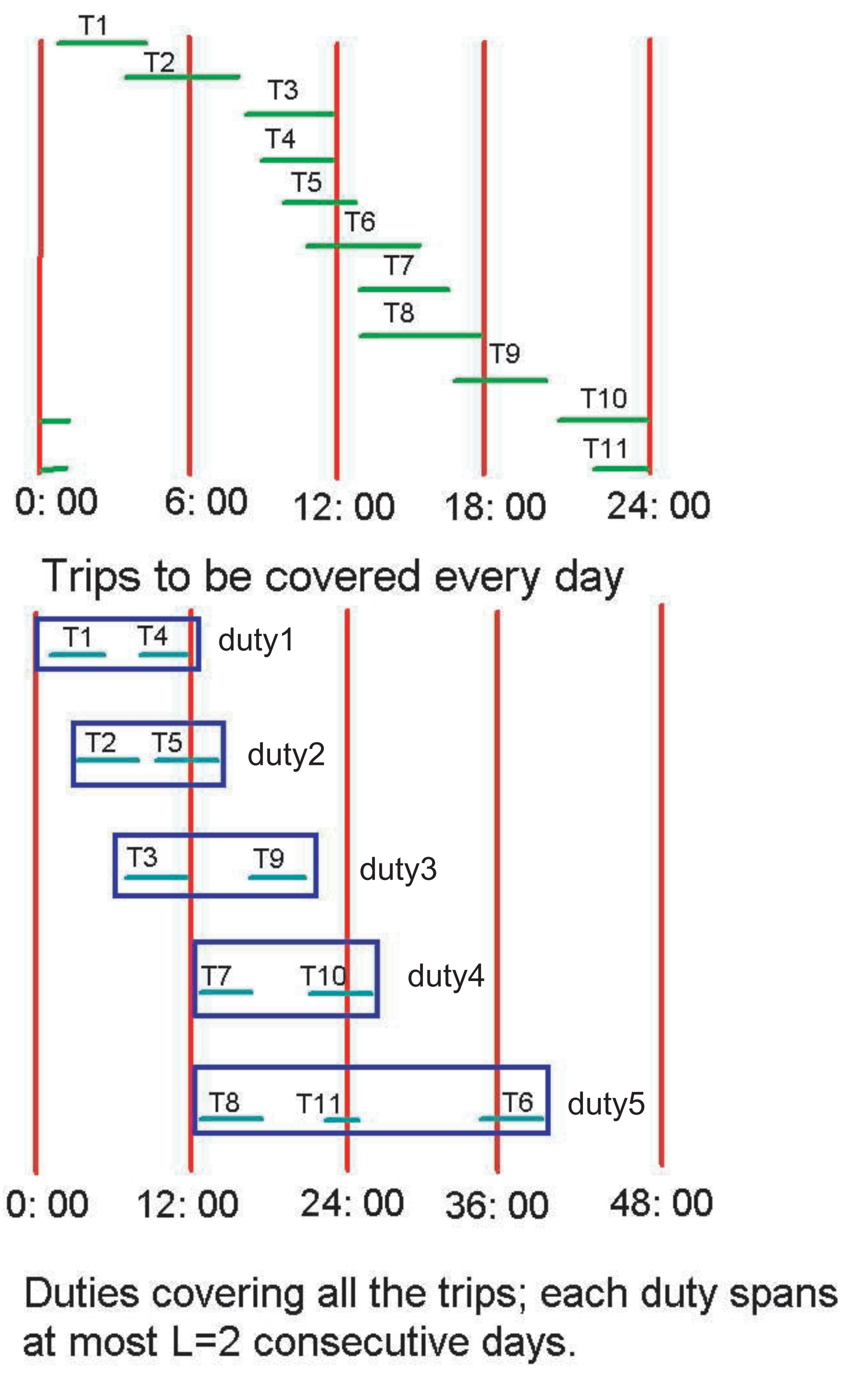

Fig. 1. Trips and duties

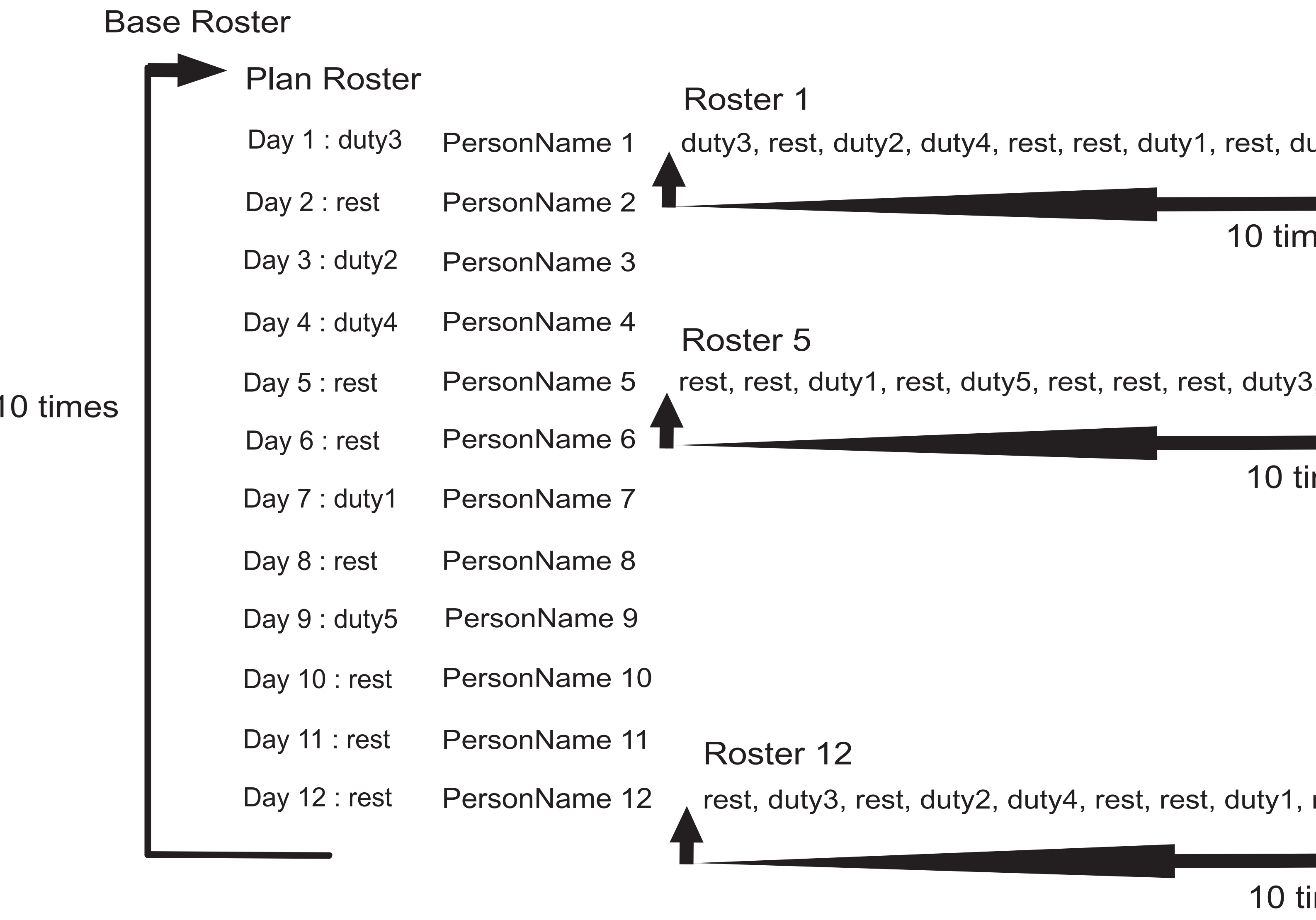

Fig. 2. Plan Roster, Base Roster and Roster

References

[1] Leo Kroon. Models for rolling stock planning. Research report, Utrecht, Netherlands. AMORE meeting Patras, Oct/Nov 2001.

[2] Gábor Maróti. Maintenance Routing. Research report, CWI, Amsterdam and NS Reizigers, Utrecht, Netherlands. AMORE meeting Patras, Oct/Nov 2001.

[3] Leo Kroon and Matteo Fischetti. Crew Scheduling for Netherlands Railways Destination: Customer. ERIM Report Series Research In Management, Netherlands, December 2000.

[4] A. Caprara, M. Fischetti, P. Toth, D. Vigo, P.L. Guida, "Algorithms for Railway Crew Management". Publication in Mathematical Programming 79 (1997) 125-141.

[5] A. Caprara, M. Fischetti, P.L. Guida, P. Toth, D. Vigo. "Solution of Large-Scale railway Crew Planning Problems: the Italian Experience", in N.H.M. Wilson (ed.) Computer-Aided Transit Scheduling, Lecture Notes in Economics and Mathematical Systems 471, Springer-Verlag (1999) 1-18.

[6] A. Caprara, M. Monaci, P. Toth. "A Global Method for Crew Planning in Railway Applications", in J. Daduna, S. Voss (eds.) Computer-Aided Transit Scheduling, Lecture Notes in Economics and Mathematical Systems 505, Springer-Verlag (2001) 17-36.

[7] Torsten Fashe. Crew assignment via constraint programming: integrating column generation and heuristic tree search. Research report, University of Paderborn, Department of Mathematics and Computer Science, Germany. AMORE meeting Patras, Oct/Nov 2001.

[8] A. Ernst, H. Jiang, M. Krishnamoorthy, H. Nott and D. Sier. "Rail Crew Scheduling and Rostering: Optimisation Algorithms", CSIRO Mathematical and Information Sciences, Australia.

[9] R. Lentink, M. Odijk and E.Rijn. "Crew Rostering for the High Speed Train", ERIM Report Series Research In Management, Netherlands, Februaty 2002.