RAILWAY STAFF ROSTERING

Albena Strupchanska[1], **Martin Pěnička**[2], **Dines Bjørner**[3]
[1] *Linguistic Modelling Department, Bulgarian Academy of Sciences*
Address: 25A Acad. G. Bonchev Str. Sofia 1113, Bulgaria
Phone: (+359 2) 9796607 , Fax: (+359 2) 707273, e–mail: albena@lml.bas.bg
[2] *Czech Technical University, Faculty of Transportation Sciences, Department of Applied Mathematics*
Address: Na Florenci 25, 110 00 Praha 1, Czech Republic
Phone: (+420) 224 890 712, Fax: (+420) 224 890 702, e–mail: penicka@fd.cvut.cz
[3] *Technical University of Denmark, Computer Science and Engineering,*
Informatics and Mathematical Modelling
Address: Building 322, Richard Petersens Plads, DK–2800 Kgs.Lyngby, Denmark
Phone: (+45) 4525 3720, Fax: (+45) 4593 0074, e–mail: db@imm.dtu.dk

Abstract: The problem to be tackled is as follows: There is a railway net. Trains travel from station to station according to a schedule. There are depots (some, not all, stations) in a railway net to which sets of staff members are associated. Staff members are responsible for performing sets of actions in order to fullfil schedule demands. Given a schedule, a staff type, a set of depots and rules (related, for example to the assigninment of staff to trains), the problem is to construct work schedules for staff members such that they conform to the rules and to 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 conform to rules and cover the duties. The assignment of staff members to base rosters is done such that each staff member recieves a roster according to that staff member's personal characteristics (abilities, previous duties etc.). We relate this model to the descriptions provided in (Caprara et al., 1997; Caprara et al., 1999; Caprara et al., 2001; Ernst et al.; Kroon et al., 2000; Lentink et al., 2002).
Keywords: staff rostering, duties, rosters, staff.

1. INTRODUCTION

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 constructed for each staff type (engine men, conductors, catering staff, etc.).

There are two types of staff planning: long–term planning and short–term planning. We focus here on long term planning. Normally the long term planning task is separated into two stages: (1) staff scheduling and (2) staff rostering. (1) Staff scheduling yields short–term working schedules, called duties, for staff members, such that they satisfy schedule demands. After this stage it is easy to determine the overall required number of staff members, such that the working schedules can be performed. (2) Staff rostering arranges duties into long-term working schedules, called base rosters, and assigns specific staff members to them, such that each staff member performs a roster. During the stage of staff rostering we assume that we have enough staff members such that we can assign rosters to them.

In this paper we will explain and analyze the problem, first informally and then formally. Using a formal specification approach and the `RAISE Specification Language, RSL` (George et al., 1992), we will present a formal model of the domain

[0] The writing of this paper, as well as the papers (Pěnička et al., 2003; Bjørner, 2003), also contained in these proceedings, and their presentation at Budapest, is sponsored by the EU IST Research Training Network AMORE: Algorithmic Models for Optimising Railways in Europe: http://www.inf.uni-konstanz.de/algo/amore/. Contract no. HPRN-CT-1999-00104, Proposal no. RTN1-1999-00446

of staff rostering.

1.1 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 by schedule, depot, and rules will be defined further into the paper.

value
  gen_sross: SCH × StfTp × Dep × eRS → Ros

gen_sross expresses all rosters for a given staff type and depot. Usually rosters are generated per depot and we assume that after the staff scheduling stage, all duties generated per depot are moved 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.

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

1.2 Requirements and Software Design

We formally characterize schedules, duties, and rosters in such a way as to meet staff rostering demands. On the basis of such formal domain characterizations we can then express software requirements.

The actual software design relies on the identification of suitable operations research techniques, such which can provide reasonably optimal solutions at reasonable computing times. It is not the aim of this paper to show such operations research algorithms. Instead we formalize the domain of railway staff rostering such that we can later apply it to operation research techniques discovered in further research work.

1.3 Paper Structure

The paper consists of five sections. Each section consists of an informal description of the problem (ie., the narrative) and its formalization.

Section 2 introduces the topology of railway nets from the perspective of staff management. Section 3 introduces the notions of a staff member and of related characteristics — such that are taken into account in the early stage of planning. The last three sections gradually show the creation of rosters from a schedule, a set of depots and rules. Section 4 is concerned with separating journeys observed from a schedule into trips. Thus the notions of journey and trip are introduced. Section 5 introduces the notion of a duty and spefies the concept of the set of duties per each depot. Finally Section 6 introduces more characteristics of staff members and the notion of rosters. The section specifies the concept of staff rosters.

2. NETS, STATIONS AND DEPOTS

We introduce the notions of nets, stations and depots. These are related to the topology of the railway net. They are specified from the staff manager point of view.

2.1 Narrative

We take as a 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 staff bases, ie., places where staff members “live”. The notion of staff member will be introduced in more details in subsequent sections. From a station we can observe the set of depots to which the station is said to belong. From a depot we can observe a set of stations from which it is easy (for staff) 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 stations and depots which are 'close' to each other.

Some constraints are: 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).

2.2 Formal Model

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

```
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
  axiom
    ∀ n:Net •
      card obs_Stas(n) ≥ 2 ∧
      card obs_Deps(n) ≥ 1 ∧
      ∀ s:Sta • s ∈ obs_Stas(n) ⇒
```

```
            (∀ d:Dep • d ∈ obs_StaDeps(s) ⇒
                d ∈ obs_Deps(n))
        ∀ d:Dep • d ∈ obs_Deps(n) ⇒
            (∀ s:Sta • s ∈ obs_DepStas(d) ⇒
                s ∈ obs_Stas(n))
    end

scheme STAFF =
    extend NETWORK with
    class
        type
            AnonStfMbr, Name,
            SpecStfMbr, PersInfo,
            StfTp == engS | condS | catS,
            AnonStaff = Name →m AnonStfMbr,
            Staff = Name →m SpecStfMbr

        value
            obs_Name: AnonStfMbr → 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

            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)

            ∀ 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

            dstft_num: Dep×StfTp → Nat
            dstft_num(d,stft) ≡
                card dom dstft(d,stft)

            dsstft_grs:
                (Dep-set×StfTp)
                    → (Dep×Nat)-set
            dsstft_grs(ds,stft) ≡
                {(dep, n)|dep:Dep,n:Nat •
                        dep ∈ ds ∧
                        n=dstft_num(dep,stft)}
    end
```

3. SCHEDULE, JOURNEYS AND TRIPS

We explain the notions of schedule, journeys and trips. They help us to introduce the notion of duties.

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, a date, and a time. A train number is a unique identifier which remains the same from the first to the last station of its journey.

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

Journeys and Trips: Staff members are responsible for performing some actions in order to fullfil schedule demands. Some actions are related to train journeys. Train journeys can be both actual journeys with passengers or freights, or can be "empty" train 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, ie., the train between these two stations. Given a schedule we can extract a set of train journeys (journ_set).

There are some restrictions about maximal working time for a staff member between rests. Taking into account these restrictions, it is natural to divide a journey into indivisible pieces of work for staff members. To this end 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 the maximal allowed un–interrupted 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 engine', 'staff types' and the number needed (for each staff type) to perform a trip etc.

3.2 Formal Model

First we will state some abstract and concrete types and some observer functions.

```
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 -m-> (DateTime -m-> Journey)

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

       cons_inti: DateTime×DateTime → Bool,
       obs_TrnId: Trn → TrnId,
       trnchr: Ride → TrnChar,
       stfchr: TrnChar → StfTp -m-> Nat,
       obs_ExchgStas: Sta -~-> 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)) ∧
                 cons_inti(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 -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, if the first and the last stations of a trip are exchangeable stations, and if the train during the trip has the same characteristics as seen 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 ∗/
     /∗ of 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 stations, where we separate train journeies, there should be enough time for exchanging staff members or for 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 this technical time. Thus 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 characteristics of trips.

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

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

trip_fsta: Trip → Sta
trip_fsta(trp) ≡ sta(hd trp),

trip_lsta: Trip → Sta
trip_lsta(trp) ≡ nsta(last(trp)),

trip_trn: Trip → Trn
trip_trn(trp) ≡ trn(hd trp),

trip_trnchr: Trip → TrnChar
trip_trnchr(trp) ≡ trnchr(hd trp),

trip_stfchr: Trip → StfTp →m Nat
trip_stfchr(trp) ≡ stfchr(trip_trnchr(trp)),

trip_WrkTm: Trip → TInt
trip_WrkTm(tp) ≡
     trip_fnT(tp) − trip_stT(tp)
```

4. ACTIONS AND DUTIES

4.1 Narrative

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

Sequences of trips are characterized by start times, end times, and lists of rides. A rest is characterized by a start and an end time, a 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, the home depot, in a railway net. A depot represents starting and ending point of staff work segments. A natural constraint imposes that each staff member must return to the 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 (see Fig. 1). 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 such as: 'duty with external rest', 'overnight duty', 'heavy overnight duty', 'long duty', etc. Also each duty has some characteristics, such as:

- *Start time:* Specified explicitly when the first action of a duty is either a rest or a 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.

- *End time:* Specified 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,

- *Paid time:* Defined as the elapsed time from start time to end time of a duty,

- *Working time:* Defined as the time interval between the start time and the end time of a duty minus the external rest, if any.

The above–mentioned 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 the engine men staff type we could observe:

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

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

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

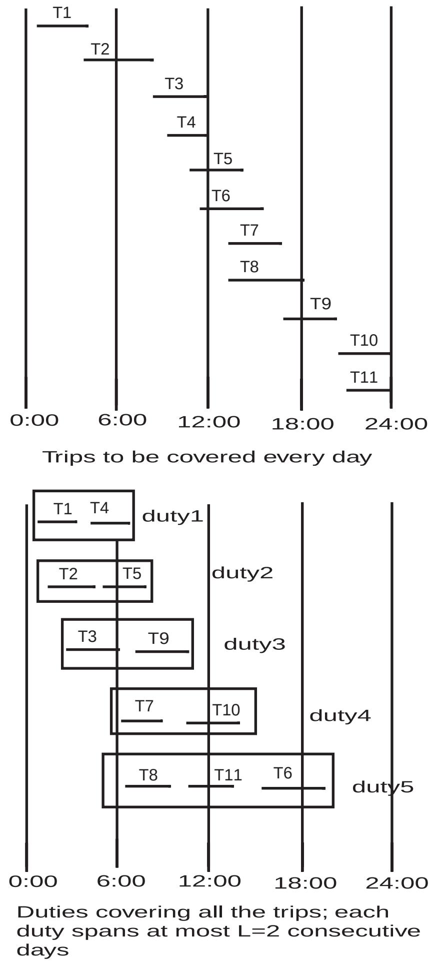


Fig. 1 Trips and Duties

4.2 Formal Model

```
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 shall take into account some depot and some staff type. The following is a function which defines a duty set for a depot. It expresses 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 to 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 characterise a duty for the depot. The function below expresses 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 d
    post
     ∃ d′:Dep •
        case ac of
         mk_trip(_,tripl,_) → d ∈
            st_stftdep(sta(hd (hd tripl)),stft),
         mk_rest(_,rsta,_,_) → d ∈
            st_stftdep(rsta,stft),
         mk_hra(_,hsta,_,_) → d ∈
            st_stftdep(hsta,stft)
        end ∧ d=d′

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 charac− ∗/
/∗ teristics 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(hd tripli,hd triplj,stft),
                _ → false
             end
     end

same: Trip×Trip×StfTp → Bool
same(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),
     _ → false
    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),
            _ → false
    end

belong: Trip* ×Duty → Bool
belong(tpl,dt) ≡
    ∃ ac:Ac • ac ∈ elems dt ∧
     case ac of
        mk_trip(st,tpl,et) → true,
        _ → false
     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, ie., restrictions, can be related to a maximal number of duties with specific characteristics per depot, to maximal number of duties per depot, etc. We will call these for 'rules on a depot level'.

The function below expresses a subset of a duty set, generated in a 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),
            _ → false
    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: The number of duties as a whole with a given characteristics, must, for example, not exceed some defined number etc. Also the generated duties as a whole have to cover all the trips that can be observed from a schedule. Finally given a schedule, a staff type, a set of depots, and rules we can express the set of duties per each depot such that the above–mentioned 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),
     _ → false
    end
```

The following are some auxiliary functions concerning duties and their 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,_,_) → st,
     mk_rest(sr,_,_,_) → sr,
     mk_hra(sh,_,_,_) → sh
    end

dt_endtime: Duty → DateTime
dt_endtime(dt) ≡
    case dt(len dt) of
     mk_trip(_,_,et) → et,
     mk_rest(_,_,_,er) → er,
     mk_hra(_,_,_,eh) → eh
    end

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

5. ROSTERS AND STAFF MEMBERS

We explain the notion of a roster and how it is related to staff members.

5.1 Narrative

Rosters: During the second stage of staff rostering the duties generated at a previous stage are ordered in rosters. These 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 auxiliary notions in order to explain the concept of roster and its stages of "construction".

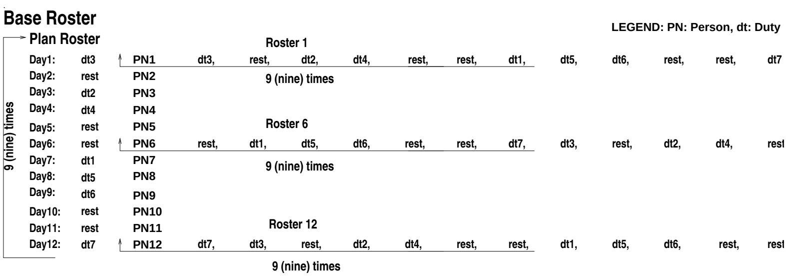
A 'plan roster' is a sequence of duties generated for anonymous staff members of the same staff type. A 'base roster' is a cyclic sequence of a plan roster such that it spans through a planning period determined by a schedule. In other words, a plan roster is that part of the base roster which is repeated several times and a base roster is just a cyclic sequence of duties (see *Fig. 2*). Each base roster has to satisfy some rules. The rules are about the order of duties in consecutive days, and their attributes. There are, additionally, some constraints concerning number of duties in a base roster with specific attributes. These rules we will call 'conventional rules at the roster level'.

So given a schedule, a staff type, a depot, and rules we can express 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 'conventional rules at the overall roster level'.

All the duties in a base roster has to be performed by a specific staff member. A roster is a cyclic sequence of (base roster) duties for a specific staff member such that that staff member can perform them. From a base roster, and a staff type we can express 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 on different days.

Staff Members: While allocating duties in a base roster to staff members, specific staff members are now considered. At this stage the specificity of staff members comes into play — as one is interested in their personal information. From a staff member personal information we can observe his/her private information (obs_PrInf), such as date of birth, place of living, address, etc. We can further observe staff qualifications (obs_Qual), special work requirements (obs_SpWrkReq), and the list of previous duties (obs_PrevDuty). Given a base roster and a staff member we can observe the staff roster.

Fig. 2 Rosters



5.2 Formal Model

```
scheme ROSTER =
extend DUTY with
    class
     type
        Info, WrkReq, Qualification
        PlRos = Duty*
        BRos = PlRos×Nat
        RoR = PlRos×StfTp → Bool
        RoRS = RoR-set
        OvR = BRos×StfTp → Bool
        OvRS = OvR-set
        eRS == RS | check_ror(rrs:RoRS)
                  | check_ovrs(ovrs:OvRS)
        Ros = SpecStfMbr ⇸ BRos

     value
        f:eRS → RS
        obs_PrInf: PersInfo → Info
        obs_SpWrkReq: PersInfo → WrkReq
        obs_PrevDuty: PersInfo → Duty*
        obs_PostDuty: PersInfo → Duty*
        obs_Qualf: PersInfo → Qualification
        obs_PlPer: SCH → Nat

        bros_length: BRos → Nat
        bros_length(bros) ≡
         let (plros, rnumb) = bros in
         len plros end
end
```

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

```
gen_dep_bross:
    SCH×StfTp×Dep×eRS → BRos-set
gen_dep_bross(sc,stft,dep,rs) as bross
    post
     ∀ bros:BRos •
        bros ∈ bross ⇒
            bros = genbros_dep(sc,stft,dep,rs) ∧
     ∼∃ bros′:BRos •
        bros′ = genbros_dep(sc,stft,dep,rs) ∧
        bros′ ∉ bross

genbros_dep: SCH×StfTp×Dep×eRS → BRos
genbros_dep(sc,stft,dep,rs) as bros
    post
     let ds = dep_dutyset(dep, stft) in
     cover_rds(bros, ds) end ∧
     wf_bros(bros,sc,stft,rs)

cover_rds: BRos×Duty-set → Bool,

wf_bros: BRos×SCH×StfTp×eRS → Bool
wf_bros(bros,sc,stft,rs) ≡
    let (plros,rnumb) = bros in
    same_qualific(plros, stft) ∧
    conform_cplrosrs(plros,stft,rs) ∧
    len plros ∗ rnumb = obs_PlPer(sc) in
    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),
        _ → false
     end
```

Sets of base rosters have to conform to some rules, such as for example: Having a 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),
        _ → false
     end
```

Given a base roster, a staff type, and a depot we can express 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 can 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 (ie., the cyclic part of the base roster).

```
assignment : BRos×Staff → Ros
assignment(bros,staff) as ros
    pre
     card rng staff > bros_length(bros)
    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)

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 qualifications, special work requirements, and previous duties, and such that they are performable by that staff member.

```
conform_rsm: BRos×SpecStfMbr → Bool
conform_rsm(bros, ssm) ≡
    sat_qual(bros,obs_Qualf(obs_PersInfo(ssm))) ∧
    sat_predt(bros,obs_PrevDuty(obs_PersInfo(ssm))) ∧
    sat_swr(bros,obs_SpWrkReq(obs_PersInfo(ssm)))

sat_qual: BRos×Qualification → Bool,
sat_predt : BRos×Duty* → Bool,
sat_swr : BRos×WrkReq → Bool,

permutation: BRos×BRos → Bool,
```

Finally we express the rosters for the given depot and staff type. For each base roster generated in the previous stage we generate these 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 staff types. So using the function above, we can generate all rosters per depot for all staff types related to this depot. To generate (all) 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 respective rosters are generated per depot, under 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 can observe all the duties generated in the staff scheduling stage per depot (dep_dutyset): This will aid us in integrating the two stages of staff planning into one. So given a schedule, a staff type, a set of depots, and rules we can now express 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) ∧
            dep_dutyset(dep,stft) ∈ dtss end ∧
     card rosset = card deps,

dep_dutyset: Dep×StfTp → Duty-set
dep_dutyset(dep,stft) ≡
    {dt|dt:Duty • dep=duty_dep(dt,stft)}
```

The rest are some of the possible functions for handling staff members in depots.

```
hire_sm: SpecStfMbr×Staff ~→ Staff
hire_sm(ssm, stf) ≡
    stf ∪ [ obs_Name(ssm)↦ssm ]
    pre
     ∀ ssm′:SpecStfMbr •
        ssm′ ∈ rng stf ⇒
            obs_Name(ssm′)≠obs_Name(ssm) ∧
     ssm ∉ rng stf

fire_sm: SpecStfMbr×Staff ~→ Staff
fire_sm(ssm,stf) ≡
```

```
    stf \ {obs_Name(ssm)}
    pre obs_Name(ssm) ∈ dom stf

hired_sm: SpecStfMbr×Staff → Bool
hired_sm(ssm, stf) ≡ ssm ∈ rng stf,

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

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

dep_staff: Dep → Staff
dep_staff(dep) ≡
    let anstaff = depStfMbrs(dep) in
    get_staff(anstaff) end
```

6. CONCLUSION

6.1 Summary

We have analysed the problem of staff rostering. This required a careful analysis of the topology of railway nets, including "what are" stations and depots, and the railway staff related to serving on trains and "located" at depots. Then followed a careful analysis of "what are" schedules, journeys, and trips, including a large number of auxiliary concepts relating schedules, journeys, trips and staff. After that followed a careful analysis of railway staff actions and duties — those "things" for which their roster is to be "built". Since rosters must satisfy many constraints we also found a need to analyse, ie., specify many auxiliary notions. We were then ready to define proper rosters and real staff members. And hence to define the main functions of the problem of staff rostering under many constraints.

6.2 Some Remarks

In this example of applying formal specification cum analysis techniques to understanding the domain and requirements for what is normally considered an operations research problem, our specifications became alarmingly detailed. But careful consideration reveals that in normal optimisation work many properties are not considered — they were here — or are overlooked, or are not even discovered. We do not claim to have discovered all necessary and sufficient properties — but to have made a great stride towards that goal.

References

Bjørner, D. (2003). *New Results and Trends in Formal Techniques & Tools for the Development of Software for Transportation Systems — A Review.* In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems.* L'Harmattan Hongrie. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany.

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

Caprara, A., M. Fischetti, P.L. Guida, P. Toth and 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.

Caprara, A., M. Monaci and 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.

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

George, C., Haff, P., Haxthausen, A., Havelund, K., Milne, R., Nielsen, C.B., Prehn, S., and Wagner, K.R. (1992). *The* RAISE *Specification Language.* The BCS Practitioners Series. Prentice-Hall International.

Kroon, L. and M. Fischetti. *Crew Scheduling for Netherlands Railways.* ERIM Report Series Research In Management, Netherlands, December 2000.

Lentink, R., M. Odijk and E.Rijn. *Crew Rostering for the High Speed Train,* ERIM Report Series Research In Management, Netherlands, February 2002.

Pěnička, M., Strupchanska, A. K., and Bjørner, D. (2003). *Train maintenance routing.* In *FORMS2003: Symposium on Formal Methods for Railway Operation and Control Systems.* L'Harmattan Hongrie. Conf. held at Techn.Univ. of Budapest, Hungary. Editors: G. Tarnai and E. Schnieder, Germany.