RaCoSy: Railway Systems

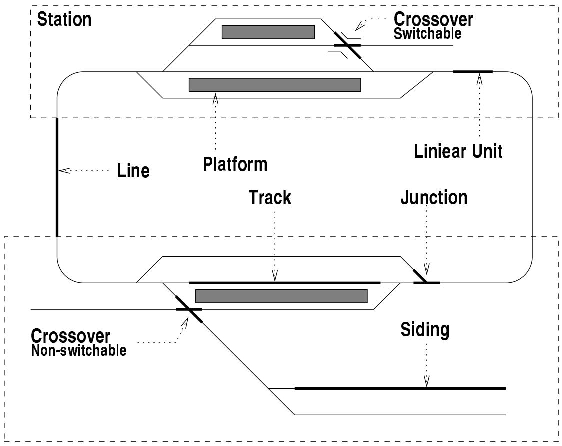
Lecture 50

- **Assumptions:** Familiarity with Concepts of Discrete Mathematics.
- **Aims & Objectives:**
  - To illustrate domain modelling principles and techniques,
  - to illustrate facets of a large scale domain, and
  - to lead up to railway optimisation formulations.
- **Treatment:** Currently only formulas are shown.

Nets, Lines, Stations, Units & Connectors

A Top–down Narrative

Figure 0.71: A Railway Net: Stations and Lines



- Nets, Lines, Stations
- Units as “Atomic” Components of Lines and Stations
- Unit Connectors
- Unit States
- Paths and Routes, Open and Closed
- *&c.*

Nets, Lines, Stations, Units & Connectors — Contd.

Formalisation

```
type N, L, S, U, C
value
  obs_Ls: N → L-set,      obs_Ss: N → S-set
  obs_Us: N → U-set,      obs_Us: L → U-set
  obs_Us: S → U-set,      obs_Cs: U-set → C-set
  obs_EndUs: L → U-set, obs_EndUs: S → U-set
axiom
  ∀ n:N, l,l′:L, s,s′:S, u,u′:U
    • l≠l′ ∧ s≠s′ ∧ u≠u′⇒
    (obs_Us(l) ∩ obs_Us(l′) = {} ∧
     obs_Us(s) ∩ obs_Us(s′) = {} ∧
     card(obs_Cs(u) ∩ obs_Cs(u′)) ≤ 1) ∧
  ∀ l:L •
    (let lines = obs_Ls(n), stations = obs_Ss(n) in
     l ∈ lines ⇒ ∃ c,c′:C,s,s′:S •
         ({s,s′} ⊆ stations ∧ s≠s′ ∧ u≠u′) ⇒
            (obs_Cs(obs_EndUs(l))
             ∩ obs_Cs(obs_EndUs(s)) = {c} ∧
            (obs_Cs(obs_EndUs(l))
             ∩ obs_Cs(obs_EndUs(s′)) = {c′}) end)
     etc. ...
```

Nets, Lines, Stations, Units & Connectors — Contd.
Discussion

- Sorts vs. Model:

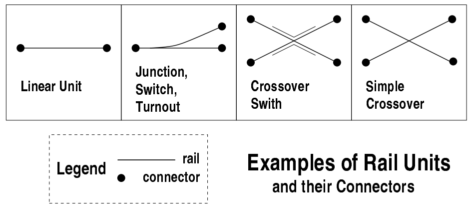
type
  L, S
  Line = U*
  Station = ... ?
value
  /* However obs_Ul affords Line view */
  obs_Ul: L → U*
  /* But why impose direction ? */
  /* Cf. first and last U of an L! */

- Observers
  One is free to define any (reasonable) observers
  as long as consistency is maintained
- Axioms
  makes-up for "loss" of model
- System Identification, i.e.: Alphabet:
  - N, L, S, U, C
  - obs_function names

Units & Connectors
A Bottom Up Formalisation

type
  U, C
value
  obs_U_Cs: U → C-**set**

Figure 0.72: Units



value
  is_Linear_U: U → **Bool**,
  is_Junction_U: U → **Bool**,
  is_Crossover_U: U → **Bool**
axiom
  forall u:U •
    is_Linear_U(u) ⇒ **card** obs_U_Cs(u) = 2,
    is_Junction_U(u) ⇒ **card** obs_U_Cs(u) = 3,
    is_Crossover_U(u) ⇒ **card** obs_U_Cs(u) = 4

Units & Connectors — Contd.
Paths and Unit States

type
  P = C × C,
  Σ = P-**set**,
  Ω = Σ-**set**
value
  obs_U_Σ: U → Σ,
  obs_U_Ω: U → Ω,

  /* All possible paths through a unit */
  U_Ps: U → P-**set**
  U_Ps(u) ≡
    { p | p:P • ∃ σ:Σ •
      σ ∈ obs_U_Ω(u) ∧ p ∈ σ
    },

  /* All connectors of a set of units */
  Us_Cs: U-**set** → C-**set**
  Us_Cs(us) ≡
    { c | c:C •
      ∃ u:U • u ∈ us ∧ c ∈ obs_U_Cs(u)
    }

Units & Connectors — Contd.
Path Axioms

axiom

/* The state is in the set of all states */

$\forall$ u:U $\cdot$ obs_U_Σ(u) $\in$ obs_U_Ω(u),

/* All connectors of paths in the state are connectors of the unit */

$\forall$ u:U, σ:Σ, (c,c′):P $\cdot$

$\sigma \in$ obs_U_Ω(u) $\wedge$ (c,c′) $\in \sigma \Rightarrow$

{c,c′} $\subseteq$ obs_U_Cs(u)

Units & Connectors — Contd.
Other Unit Axioms

axiom

forall u:U $\cdot$

is_Linear_U(u) $\Rightarrow$ U_Ps(u)$\neq${},

is_Junction_U(u) $\Rightarrow$

$\exists$ c1,c2,c3:C $\cdot$ **card** {c1,c2,c3} = 3 $\wedge$

{(c1,c2),(c2,c1)} $\cap$ U_Ps(u) $\neq$ {} $\wedge$

{(c1,c3),(c3,c1)} $\cap$ U_Ps(u) $\neq$ {} $\wedge$

{(c2,c3),(c3,c2)} $\cap$ U_Ps(u) = {},

is_Crossover_U(u) $\Rightarrow$

$\exists$ c1,c2,c3,c4:C $\cdot$ **card** {c1,c2,c3,c4} = 4 $\wedge$

{(c1,c4),(c4,c1)} $\cap$ U_Ps(u) $\neq$ {} $\wedge$

{(c2,c3),(c3,c2)} $\cap$ U_Ps(u) $\neq$ {} $\wedge$

{(c1,c3),(c3,c1)} $\cap$ U_Ps(u) = {} $\wedge$

{(c2,c4),(c4,c2)} $\cap$ U_Ps(u) = {}

Networks

type

N

value

obs_N_Us: N $\rightarrow$ U-**set**

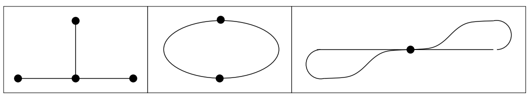
axiom

/* In a network, a connector connects no more than two units */

$\forall$ n:N, c:C $\cdot$

card { u | u:U $\cdot$ u $\in$ obs_N_Us(n) $\wedge$ c $\in$ obs_U_Cs(u) } $\leq$ 2

Figure 0.73: Unit Connectors



Examples on non-Rail Units

Networks — Contd.
Routes

type

Rt

value

obs_Rt_UP: Rt $\rightarrow$ (U $\times$ P)*,

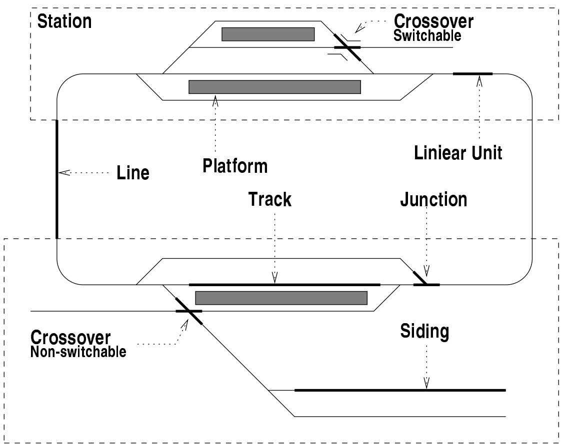
Rt_Ul: Rt $\rightarrow$ U*

Rt_Ul(r) $\equiv$ $\langle$ u | (u,p) **in** obs_Rt_UP(r) $\rangle$,

Rt_Pl: Rt $\rightarrow$ P*

Rt_Pl(r) $\equiv$ $\langle$ p | (u,p) **in** obs_Rt_UP(r) $\rangle$

Figure 0.74: Lines and Stations



Network Routes — Contd.

Route Axioms

axiom
  forall r:Rt $\cdot$
  /* Routes are not empty */
  obs_Rt_UP(r) $\neq \langle\rangle$,

  /* Routes contains units of a network */
  $\exists$ n:N $\cdot$ Rt_Us(r) $\subseteq$ obs_N_Us(n),

  /* Routes consist of possible paths through units */
  $\forall$ (u,p) : U $\times$ P $\cdot$
    (u,p) $\in$ **elems** obs_Rt_UP(r) $\Rightarrow$ p $\in$ U_Ps(u),

  /* Paths of a route are connected */
  let pl = Rt_Pl(r) **in**
    $\forall$ i:**Nat** $\cdot$ {i,i+1} $\subseteq$ **inds** pl $\Rightarrow$
    let (c,c′)=pl(i), (c″,c‴)=pl(i+1) **in** c′=c″ **end** **end**,

  /* Two successive paths of a route */
  /* do not go through the same unit */
  let ul = Rt_Ul(r) **in**
    $\forall$ i:**Nat** $\cdot$ {i,i+1} $\subseteq$ **inds** ul $\Rightarrow$ ul(i)$\neq$ul(i+1) **end**

Network Routes — Contd.

Open Routes

value
  /* Examine if a route is open */
  is_OpenRt: Rt $\rightarrow$ **Bool**
  is_OpenRt(r) $\equiv$
    $\forall$ (u,p) : U $\times$ P $\cdot$
      (u,p) $\in$ **elems** obs_Rt_UP(r) $\Rightarrow$ p $\in$ obs_U_Σ(u)

Network Routes — Contd.

Routable Sets of Units

value
  /* All possible routes through a set of units */
  Us_Rts: U-**set** $\rightarrow$ Rt-**set**
  Us_Rts(us) $\equiv$ { r | r:Rt $\cdot$ Rt_Us(r) = us },

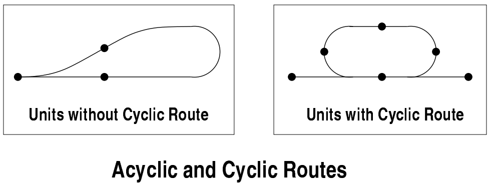
  /* There is a route through units */
  is_RoutableUs: U-**set** $\rightarrow$ **Bool**
  is_RoutableUs(us) $\equiv$ Us_Rts(us) $\neq$ {},

  /* All units of a route */
  Rt_Us: Rt $\rightarrow$ U-**set**
  Rt_Us(r) $\equiv$ **elems** Rt_Ul(r)

Network Routes — Contd.
Cyclic Routes

value
  is_Cyclic_Rt: Rt → **Bool**
  is_Cyclic_Rt(r) ≡
    let upl = obs_Rt_UP(r) **in**
      ∃ i,j:**Nat** • {i,j} ⊆ **inds** upl ∧ i≠j ∧
      let (u,(c,c′))=upl(i), (u′,(c″,c‴))=upl(j) **in**
        (u,c′)=(u′,c‴)
      end
    end

Figure 0.75: Cyclic Units



Network Routes — Contd.
Other Route Functions

value
  /∗ The first connector of a route ∗/
  Rt_firstC: Rt → C
  Rt_firstC(r) ≡ **let** (c,c′) = **hd** Rt_Pl(r) **in** c **end**,

  /∗ The last connector of a route ∗/
  Rt_lastC: Rt → C
  Rt_lastC(r) ≡
    let pl = Rt_Pl(r), (c,c′) = pl(**len** pl) **in** c′ **end**,

  /∗ The first unit of a route ∗/
  Rt_firstU: Rt → U
  Rt_firstU(r) ≡ **hd** Rt_Ul(r),

  /∗ The last unit of a route ∗/
  Rt_lastU: Rt → U
  Rt_lastU(r) ≡ **let** ul = Rt_Ul(r) **in** ul(**len** ul) **end**,

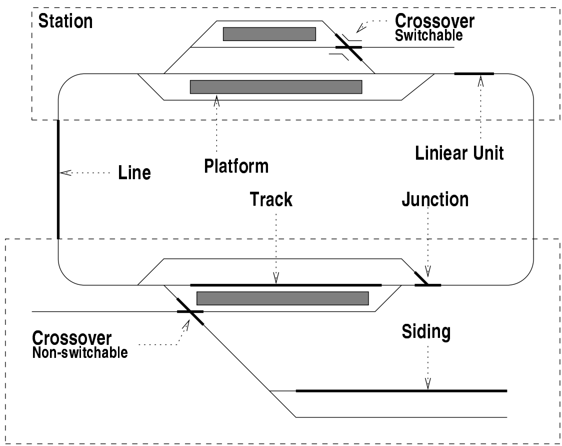
  /∗ Two routes are disjoint ∗/
  Rt_Disj: Rt × Rt → **Bool**
  Rt_Disj(r,r′) ≡ Rt_Us(r) ∩ Rt_Us(r′) = {}

Lines and Stations
Lecture 51

type
  L, S, Trk

value
  obs_N_Ls: N → L-**set**,
  obs_N_Ss: N → S-**set**,
  obs_L_Us: L → U-**set**,
  obs_S_Us: S → U-**set**,
  obs_S_Trks: S → Trk-**set**,
  obs_Trk_Us: Trk → U-**set**

Figure 0.76: Lines and Stations



Lines and Stations — Contd.
Reachability

```
value
  /* All lines that can be reached   */
  /* from a track in a given station */
  TrkLs: N × S × Trk ∼→ L-set
  TrkLs(n,s,t) ≡
    { l | l:L • l ∈ obs_N_Ls(n) ∧
      ∃ rt:Rt •
        rt ∈ Us_Rts(obs_S_Us(s)) ∧
        Rt_firstC(rt) ∈ Us_Cs(obs_Trk_Us(t)) ∧
        Rt_lastC(rt) ∈ Us_Cs(obs_L_Us(l)) }
  pre t ∈ obs_S_Trks(s) ∧ s ∈ obs_N_Ss(n),

  /* All tracks in a station that can */
  /* be reached  from a given line    */
  LTrks: N × L × S ∼→ Trk-set
  LTrks(n,l,s) ≡
    { t | t:Trk • t ∈ obs_S_Trks(s) ∧
      ∃ rt:Rt •
        rt ∈ Us_Rts(obs_S_Us(s)) ∧
        Rt_firstC(rt) ∈ Us_Cs(obs_L_Us(l)) ∧
        Rt_lastC(rt) isinn Us_Cs(obs_Trk_Us(t)) }
  pre l ∈ obs_N_Ls(n) ∧ s ∈ obs_N_Ss(n)
```


Lines and Stations — Contd.
Line–Station Connections

```
value
  /* Examine if a route of a line connects to a station */
  LS_connection: L × S → Bool
  LS_connection(l,s) ≡
    ∃ rt:Rt •
      rt ∈ Us_Rts(obs_L_Us(l)) ∧
      Rt_lastC(rt) ∈ Us_Cs(obs_S_Us(s))

  /* Examine if a station connects to a route of a line */
  SL_connection: S × L → Bool
  SL_connection(s,l) ≡
    ∃ rt:Rt •
      rt ∈ Us_Rts(obs_L_Us(l)) ∧
      Rt_firstC(rt) ∈ Us_Cs(obs_S_Us(s))

  /* Examine if two stations are connected via a line */
  SLSConnection: S × L × S → Bool
  SLSConnection(s,l,s′) ≡
    SL_connection(s,l) ∧ LS_connection(l,s′)
```


Lines and Stations — Contd.
Line and Station Units

```
value
  /* All units of the lines in a network */
  N_L_Us: N → U-set
  N_L_Us(n) ≡
    { u | u:U •
      ∃ l:L • l ∈ obs_N_Ls(n) ∧ u ∈ obs_L_Us(l)
    },

  /* All units of the stations in a network */
  N_S_Us: N → U-set
  N_S_Us(n) ≡
    { u | u:U •
      ∃ s:S • s ∈ obs_N_Ss(n) ∧ u ∈ obs_S_Us(s)
    }
or:
  N_L_Us(n) ≡
    ∪ { obs_L_Us(l) | l:L • l ∈ obs_N_Ls(n)}

  N_S_Us(n) ≡
    ∪ { obs_S_Us(s) | s:S • s ∈ obs_N_Ss(n)}
```


Lines and Stations — Contd.
Line and Station Axioms

```
axiom
  forall n:N, l,l′:L, s,s′:S, t,t′:Trk, c:C, u:U •

  /* Lines are routable and consist of linear units */
  is_RoutableUs(obs_L_Us(l)),
  u ∈ obs_L_Us(l) ⇒ is_Linear_U(u),

  /* Tracks are routable and consist of lineal units */
  is_RoutableUs(obs_Trk_Us(t)),
  u ∈ obs_Trk_Us(t) ⇒ is_Linear_U(u),

   /* Lines in a network do not intersect */
   {l,l′} ⊆ obs_N_Ls(n) ⇒
     obs_L_Us(l) ⊆ obs_N_Us(n) ∧
     l≠l′ ⇒ obs_L_Us(l) ∩ obs_L_Us(l′) = {},

   /* Stations in a network do not intersect */
   {s,s′} ⊆ obs_N_Ss(n) ⇒
     obs_S_Us(s) ⊆ obs_N_Us(n) ∧
     s≠s′ ⇒ obs_S_Us(s) ∩ obs_S_Us(s′) = {},
```

Lines and Stations — Contd.
Line & Station Axioms — Contd.

/* Lines and stations do not intersect */
$l \in obs\_N\_Ls(n) \land s \in obs\_N\_Ss(n) \Rightarrow$
$\quad obs\_L\_Us(l) \cap obs\_S\_Us(s) = \{\}$,

/* Lines connect stations */
$l \in obs\_N\_Ls(n) \Rightarrow$
$\quad \exists\, s,s':S \bullet$
$\quad\quad s \neq s' \land \{s,s'\} \subseteq obs\_N\_Ss(n) \land$
$\quad\quad SLSConnection(s,l,s')$,

/* Tracks of a station do not intersect */
$\{t,t'\} \subseteq obs\_S\_Trks(s) \Rightarrow$
$\quad obs\_Trk\_Us(t) \subseteq obs\_S\_Us(s) \land$
$\quad t \neq t' \Rightarrow obs\_Trk\_Us(t) \cap obs\_Trk\_Us(t') = \{\}$,

/* Stations do not have common connectors */
$\{s,s'\} \subseteq obs\_N\_Ss(n) \land s \neq s' \Rightarrow$
$\quad Us\_Cs(obs\_S\_Us(s)) \cap Us\_Cs(obs\_S\_Us(s')) = \{\}$

Lines and Stations — Contd.
Station Names

type
$\quad Sn$
value
$\quad obs\_S\_Sn: S \to Sn$
axiom
$\quad \forall\, n:N,\, s,s':S \bullet$
$\quad\quad \{s,s'\} \subseteq obs\_N\_Ss(n) \land s \neq s' \Rightarrow obs\_S\_Sn(s) \neq obs\_S\_Sn(s')$

Trains
Train Routes

type
$\quad TR = Rt$
value
$\quad AddTR: TR \times TR \xrightarrow{\sim} TR$
$\quad AddTR(tr',tr) \equiv tr' \frown tr$
$\quad$ **pre** $connectableTR(tr',tr)$,

$\quad RemTR: TR \times TR \xrightarrow{\sim} TR$
$\quad RemTR(tr',tr)$ **as** tr''
$\quad$ **post** $tr' = tr \frown tr''$
$\quad$ **pre** $prefixTR(tr,tr')$,

$\quad connectableTR: TR \times TR \to$ **Bool**
$\quad connectableTR(tr,tr') \equiv \exists\, tr'':TR \bullet tr''=tr \frown tr'$,

$\quad prefixTR: TR \times TR \to$ **Bool**
$\quad prefixTR(tr,tr') \equiv \exists\, tr'':TR \bullet tr'=tr \frown tr''$,

$\quad suffixTR: TR \times TR \to$ **Bool**
$\quad suffixTR(tr,tr') \equiv \exists\, tr'',tr''':TR \bullet tr'=tr'' \frown tr \frown tr'''$

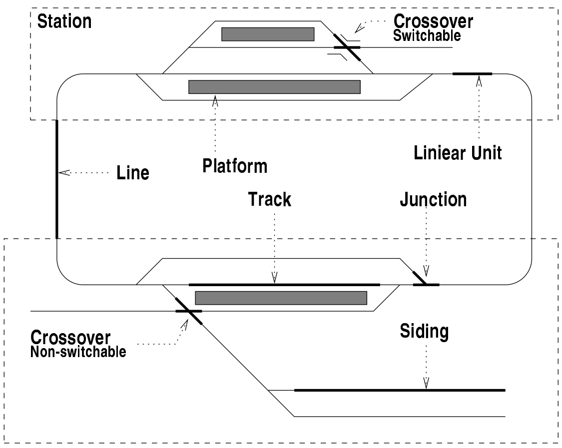
Trains — Contd.
Trains at Stations and Tracks

value
$\quad TR\_at\_S: TR \times S \to$ **Bool**
$\quad TR\_at\_S(tr,s) \equiv Rt\_Us(tr) \subseteq obs\_S\_Us(s)$,

$\quad TR\_at\_Trk: TR \times Trk \to$ **Bool**
$\quad TR\_at\_Trk(tr,trk) \equiv Rt\_Us(tr) \subseteq obs\_Trk\_Us(trk)$,

$\quad TR\_at\_StaTrk: TR \times S \to$ **Bool**
$\quad TR\_at\_StaTrk(tr,s) \equiv$
$\quad\quad \exists\, trk:Trk \bullet trk \in obs\_S\_Trks(s) \land TR\_at\_Trk(tr,trk)$

Figure 0.77: Lines and Stations



Managed Rail Nets

type
  T,
  MR′ = T → N,
  MR = {| mr:MR′ • wf_MR(mr) |}
value
  wf_MR: MR′ → **Bool**
  wf_MR(mr) ≡
    ∀ t:T • ∃ t′:T • t′>t ∧
      ∀ t″:T • t≤t″≤t′ ⇒ MoN(mr(t),mr(t″)),

  MoN: N × N → **Bool**

Managed Rail Nets — Contd.
Monotonicity

value
  /∗ Removed or inserted stations contain only closed units ∗/
  rem_ins_S_closed: N × N → **Bool**
  rem_ins_S_closed(n,n′) ≡
    ∀ s:S • s ∈ (obs_N_Ss(n)\obs_N_Ss(n′)) ∪
          (obs_N_Ss(n′)\obs_N_Ss(n)) ⇒
        closed_Us(obs_S_Us(s)),

  /∗ Removed or inserted lines contain only closed units ∗/
  rem_ins_L_closed: N × N → **Bool**
  rem_ins_L_closed(n,n′) ≡
    ∀ l:L • l ∈ (obs_N_Ls(n)\obs_N_Ls(n′)) ∪
          (obs_N_Ls(n′)\obs_N_Ls(n)) ⇒
        closed_Us(obs_L_Us(l)),

  closed_Us: U-**set** → **Bool**
  closed_Us(us) ≡
    ∀ u:U • u ∈ us ⇒ obs_U_Σ(u) = {}
axiom
  ∀ n,n′:N • MoN(n,n′) ⇒
    rem_ins_S_closed(n,n′) ∧
    rem_ins_L_closed(n,n′)

Managed Rail Nets — Contd.
Station Removal

Theorem
  ∀ mr:MR, t:T, s:S • S_removed(mr,t,s) ⇒
    ∼∃ l:L • l ∈ obs_N_Ls(mr(t)) ∧
    (SL_connection(s,l) ∨ LS_connection(l,s))

value
  S_removed: MR × T × S → **Bool**
  S_removed(mr,t,s) ≡
    s ∈ obs_N_Ss(mr(t)) ∧
    ∃ t′:T • t′>t ∧ ∀ t″:T • t<t″≤t′ ⇒ s ∉ obs_N_Ss(mr(t″))

Traffic

```
type
  TF′ = T → RS,
  TF = {| tf:TF′ • wf_TF(tf) |},

  RS′ :: net:N  trns:TP,
  RS = {| rs:RS′ • wf_RS(rs) |},

  TP = Tn →m TS,
  Tn,TS
value
  obs_TS_TR: TS → TR,
  obs_TS_Velocity: TS → ...
  obs_TS_Acc: TS → ...
  ...
```


Traffic — Contd.
Traffic Wellformedness

```
value
  /* Trains do not jump */
  wf_TF: TF′ → Bool
  wf_TF(tf) ≡ continous_movement(tf),

  /* Trains do not jump from one spot to another */
  continous_movement: TF′ → Bool
  continous_movement(tf) ≡
   ∀ t:T, tn:Tn • tn ∈ TF_Tns(tf,t) ⇒
      train_removed(tf,tn,t) ∨
      train_wf_move(tf,tn,t),

  /* In the traffic tf the train tn is removed at time t */
  train_removed: TF′ × Tn × T → Bool
  train_removed(tf,tn,t) ≡
    tn ∈ TF_Tns(tf,t) ∧
    ∃ t′:T • t′>t ∧
      ∀ t″:T • t<t″≤t′ ⇒ tn ∉ TF_Tns(tf,t″),
```


Traffic Wellformedness — Contd.

```
  /* In the traffic tf the train tn is performing
     (a wellformed continuous) move at time t */
  train_wf_move: TF′ × Tn × T → Bool
  train_wf_move(tf,tn,t) ≡
    ∃ t′:T • t′>t ∧ ∀ t″:T • t≤t″≤t′ ⇒
       tn ∈ TF_Tns(tf,t″) ∧
       intersecting_move(TF_TR(tf,tn,t),TF_TR(tf,tn,t″)),

  intersecting_move: TR × TR → Bool
  intersecting_move(tr,tr′) ≡
    ∃ tr1,tr3:TR •
      prefixTR(tr1,tr) ∧
      let tr2=RemTR(tr,tr1) in
        connectableTR(tr2,tr3) ∧
        tr′ = AddTR(tr2,tr3)
      end,

  TF_Tns: TF′ × T → Tn-set
  TF_Tns(tf,t) ≡ dom trns(tf(t)),

  TF_TR: TF′ × Tn × T → TR
  TF_TR(tf,tn,t) ≡ obs_TS_TR(trns(tf(t))(tn))
  pre tn ∈ TF_Tns(tf,t)
```


Traffic Wellformedness — Contd.
Rail State Well–formedness

```
value
  wf_RS: RS′ → Bool
  wf_RS(rs) ≡ TP_onrails(net(rs),trns(rs)),

  /* The trains of a TP are on units of the network */
  TP_onrails: N × TP → Bool
  TP_onrails(n,tp) ≡
    ∀ tn:Tn • tn ∈ dom tp ⇒
      Rt_Us(obs_TS_TR(tp(tn))) ⊆ obs_N_Us(n)
```

Traffic Wellformedness — Contd.
Traffic Quality
Lecture 52

```
value
  isbetter_TF: TF × TF → Bool

  /∗ Trains are on open routes ∗/
  TF_openroutes: TF → Bool
  TF_openroutes(tf) ≡ ∀ t:T • TP_openroutes(trns(tf(t))),

  /∗ Trains are on lines or within stations of the network ∗/
  TF_on_S_or_L: TF → Bool
  TF_on_S_or_L(tf) ≡ ∀ t:T • TP_on_S_or_L(net(tf(t)),trns(tf(t))),

  TP_openroutes: TP → Bool
  TP_openroutes(tp) ≡
    ∀ tn:Tn • tn ∈ dom tp ⇒ is_OpenRt(obs_TS_TR(tp(tn))),

  TP_on_S_or_L: N × TP → Bool
  TP_on_S_or_L(n,tp) ≡
    ∀ tn:Tn • tn ∈ dom tp ⇒
      Rt_Us(obs_TS_TR(tp(tn))) ⊆ N_L_Us(n) ∪ N_S_Us(n)
```


Traffic Wellformedness — Contd.
Traffic Quality: Train Routes

```
value
  /∗ No two trains share units ∗/
  TF_disj_TR: TF → Bool
  TF_disj_TR(tf) ≡ ∀ t:T • TP_disj(trns(tf(t))),

  /∗ A trainroute does not run through the same unit twice ∗/
  TF_trdisj: TF → Bool
  TF_trdisj(tf) ≡ ∀ t:T • TP_trdisj(trns(tf(t))),

  TP_disj: TP → Bool
  TP_disj(tp) ≡
    ∀ tn,tn′:Tn • {tn,tn′} ⊆ dom tp ∧ tn≠tn′ ⇒
      Rt_Disj(obs_TS_TR(tp(tn)),obs_TS_TR(tp(tn′))),

  TP_trdisj: TP → Bool
  TP_trdisj(tp) ≡
    ∀ tn:Tn • tn ∈ dom tp ⇒
    let tr=obs_TS_TR(tp(tn)) in
      card Rt_Us(tr) = len Rt_Ul(tr)
    end
```


Traffic Wellformedness — Contd.
Traffic Quality: Rail Nets

```
value
  /∗ The managed railnet of a traffic is wellformed ∗/
  TF_wf_MR: TF → Bool
  TF_wf_MR(tf) ≡ ∃ mr:MR • ∀ t:T • mr(t) = net(tf(t))
```


Schedules

```
type
  SC = TF-infset
```

Schedule Descriptions

```
type
  SCdesc = T →m (RS → Bool)

value
  SCdesc_SC: SCdesc → SC
  SCdesc_SC(scd) ≡
    { tf | tf:TF • TF_sat_SCdesc(tf,scd) },

  TF_sat_SCdesc: TF × SCdesc → Bool
  TF_sat_SCdesc(tf,scd) ≡
    ∀ t:T • t ∈ dom scd ⇒ scd(t)(tf(t))
```

Timetables

type
  TT = SC

value
  SC_sat_TT: SC × TT → **Bool**
  SC_sat_TT(sc,tt) ≡ sc ⊆ tt

Timetables — Contd.
Timetable Descriptions

type
  TTdesc = Tn $\overrightarrow{m}$ J,
  J′ = Sn $\overrightarrow{m}$ SV,
  J = {| j:J′ • wf_J(j) |},
  SV′ :: arrival : T depart : T,
  SV = {| sv:SV′ • wf_SV(sv) |}

value
  /∗ Arrival before departure ∗/
  wf_SV: SV′ → **Bool**
  wf_SV(sv) ≡ arrival(sv) ≤ depart(sv),

  /∗ Station visits are disjoint ∗/
  wf_J: J′ → **Bool**
  wf_J(j) ≡
    ∀ sn,sn′:Sn • {sn,sn′} ⊆ **dom** j ∧ sn≠sn′ ⇒
      disj_SV(j(sn),j(sn′)),

  disj_SV: SV × SV → **Bool**
  disj_SV(sv,sv′) ≡
    arrival(sv)>depart(sv′) ∨ arrival(sv′)>depart(sv)

Timetables — Contd.
Timetable Description Semantics

value
  TTdesc_TT: TTdesc → TT
  TTdesc_TT(tt) ≡ { tf | tf:TF • TF_sat_TTdesc(tf,tt) },

  TF_sat_TTdesc: TF × TTdesc → **Bool**
  TF_sat_TTdesc(tf,tt) ≡
    ∀ tn:Tn, sn:Sn •
      tn ∈ **dom** tt ∧ sn ∈ **dom** tt(tn) ⇒
      TF_sat_SV(tf,tn,sn,tt(tn)(sn)),

  /∗ For the duration of the stationvisit, the station
    is part of the network and the train is at a track
    within the station ∗/

  TF_sat_SV: TF × Tn × Sn × SV → **Bool**
  TF_sat_SV(tf,tn,sn,sv) ≡
    ∀ t:T • arrival(sv)≤t≤depart(sv) ⇒
      ∃ s:S • obs_S_Sn(s)=sn ∧ s ∈ obs_N_Ss(net(tf(t))) ∧
      TR_at_StaTrk(TF_TR(tf,tn,t),s)

Rescheduling

value
  TF_on_SC: TF × SC → **Bool**
  TF_on_SC(tf,sc) ≡ tf ∈ sc,

  /∗ Examine if all possible traffics are on schedule ∗/

  TFs_on_SC: TF-**set** × SC → **Bool**
  TFs_on_SC(tfs,sc) ≡
    ∀ tf:TF • tf ∈ tfs ⇒ TF_on_SC(tf,sc),

  /∗ Examine if no possible traffics are on schedule ∗/

  TFs_not_on_SC: TF-**set** × SC → **Bool**
  TFs_not_on_SC(tfs,sc) ≡
    ∀ tf:TF • tf ∈ tfs ⇒ ∼TF_on_SC(tf,sc),

Rescheduling — Contd.

disruption: TF-**set** $\times$ SC $\rightarrow$ **Bool**
disruption(tfs,sc) $\equiv$ TFs_not_on_SC(tfs,sc),

/* Traffic should adhere to the new schedule */

new_SC: TF-**set** $\times$ SC $\rightarrow$ **Bool**
new_SC(tfs,sc) $\equiv$ ~disruption(tfs,sc),

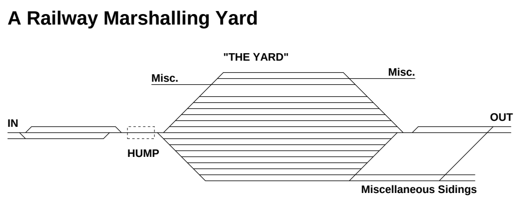
new_TT: TT $\times$ SC $\rightarrow$ **Bool**
new_TT(tt,sc) $\equiv$ SC_sat_TT(sc,tt)

Marshalling Yards

type
  MY
value
  obs_MY_Us: MY $\rightarrow$ U-**set**,
  obs_MY_incoming: MY $\rightarrow$ C-**set**,
  obs_MY_outgoing: MY $\rightarrow$ C-**set**,
  obs_MY_hump: MY $\rightarrow$ U,

  is_MY_inout_Rt: Rt $\times$ MY $\rightarrow$ **Bool**
  is_MY_inout_Rt(rt,my) $\equiv$
    Rt_Us(rt) $\subseteq$ obs_MY_Us(my) $\wedge$
    Rt_firstC(rt) $\in$ obs_MY_incoming(my) $\wedge$
    Rt_lastC(rt) $\in$ obs_MY_outgoing(my)

Figure 0.78: Marshalling Yard



Marshalling Yards — Contd.
Marshalling Yard Axioms

axiom
  forall my:MY, u:U, c,c′:C •

    /* Humps are linear */

    is_Linear_U(obs_MY_hump(my)),

    /* Incoming and outgoing connectors are disjoint */

    obs_MY_incoming(my) $\cap$ obs_MY_outgoing(my) = {},

    /* There is a route from any incoming connector
      to any outgoing connector */

    c $\in$ obs_MY_incoming(my) $\wedge$ c′ $\in$ obs_MY_outgoing(my) $\Rightarrow$
      $\exists$ rt:Rt • is_MY_inout_Rt(rt,my) $\wedge$
        (c,c′) = (Rt_firstC(rt),Rt_lastC(rt)),

    /* Between any two connectors in a marshalling yard
      there is no more than one route */

    ~$\exists$ rt,rt′:Rt • rt $\neq$ rt′ $\wedge$
      Rt_Us(rt) $\cup$ Rt_Us(rt′) $\subseteq$ obs_MY_Us(my) $\wedge$
      (Rt_firstC(rt),Rt_lastC(rt)) = (Rt_firstC(rt′),Rt_lastC(rt′)),

Marshalling Yards — Contd.
Marshalling Yard Axioms — Contd.

/* All units in a marshalling yard are part of a route
  from an incoming connector to a outgoing connector
  which goes through the hump */

u $\in$ obs_MY_Us(my) $\Rightarrow$
  $\exists$ rt:Rt • is_MY_inout_Rt(rt,my) $\wedge$
    {u,obs_MY_hump(my)} $\subseteq$ Rt_Us(rt),

/* Units are never open in direction from the outgoing side
  towards the incoming side */

$\forall$ rt:Rt • is_MY_inout_Rt(rt,my) $\Rightarrow$
  (u,(c,c′)) $\in$ **elems** obs_Rt_UP(rt) $\Rightarrow$ (c′,c) $\notin$ obs_U_Σ(u)

Marshalling Yards — Contd.
Train Bodies

type
  TB = W*,
  W
value
  obs_TS_TB: TS → TB,
  obs_W_kind: W → ...

Marshalling Plans

type
  MP
value
  obs_MP_incoming: MP → TB-**set**,
  obs_MP_outgoing: MP → TB-**set**

Marshalling Yards — Contd.
Marshalling Descriptions

type
  MD′ = MS*,
  MD = {| md:MD′ • wf_MD(md) |},
  MS :: incoming: TB-**set** outgoing: TB-**set**
value
  wf_MD: MD′ → **Bool**
  wf_MD(md) ≡
    ∀ i:**Nat** • {i,i+1} ⊆ **inds** md ⇒
      wf_state_shift(md(i),md(i+1)),

  wf_state_shift: MS × MS → **Bool**
  wf_state_shift(ms,ms′) ≡
    ∃ w:W, tb,tb′:TB •
    let ims=incoming(ms)\{tb} ∪ {**tl** tb},
        oms=outgoing(ms)\{tb′} ∪ {tb′⌢**hd** tb}
    in ms′ = mk_MS(ims,oms) **end**

Marshalling Yards — Contd.
Feasibility of Marshalling Plans

value
  feasible_MP: MP → **Bool**
  feasible_MP(mp) ≡
    ∃ md:MD •
      incoming(**hd** md) = obs_MP_incoming(mp) ∧
      outgoing(md(**len** md)) = obs_MP_outgoing(mp),

  feasible_MP_wrt_MY: MP × MY → **Bool**
  feasible_MP_wrt_MY(mp,my) ≡
    feasible_MP(mp) ∧
    card obs_MY_incoming(my) ≥ **card** obs_MP_incoming(mp) ∧
    card obs_MY_outgoing(my) ≥ **card** obs_MP_outgoing(mp)

Review of the Railway Systems Model

-
-
-
-
-