Transportation Nets

A Domain Model, a Set of Requirements and a Set of High-level Software Designs

Dines Bjørner
Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling
Richard Petersens Plads, Building 322
Technical University of Denmark
DK-2800 Kgs. Lyngby
Denmark

db@imm.dtu.dk, http://www.imm.dtu.dk/~db

January 4, 2006

Preface

Multi-modal transportation nets consists of segments and junctions. Segments are single-modal and are either roads, or rail tracks, or air-lanes or shipping lanes, but only exactly one of these. Junctions are multi-modal and can be thought of as a non-empty subset of road intersections, train stations, airports and harbours. We present a narrative and a formalisation of multi-modal transportation nets while exploring notions of single or multi-modal paths and routes, composition and decomposition of nets from, respectively into several such, cost of transports between junctions, and so on.

In this booklet we develop — according to the triptych principles of software engineering — a variety of software systems for transporation applications. Chapter 2 unfolds a domain model for multi-modal transpotation; Chap. 3 unfolds a variety of requirements models, and Chap. 4 develops this variety of requirements prescriptions into respective software packages.

Contents

Chapter 1

Triptych Development of Transportation Software

1.1 Multi-Modal Transportation Nets

By *transport* we mean that something, say *freight* is being *moved* along some *route* driven so by some *conveyour* propelled by some *motive force*.

Thus there are five phenomena involved in *transport*. They are *freight*, *movement*, *route*, *conveyour* and *motive force*.

Routes are part of *transportation nets*. The nets are seen as set of junctions connected by segments. The junctions are either street (or road) intersections, or are train stations, or are airports, or are harbours. The corresponding segments, consequentially, are roads, rail lines, air corridors (air lanes), or are shipping lanes. A route is then an alternating sequence of junctions and segments.

Conveyours consequentially are either cars (private automobiles, taxis, busses, and trucks), or are trains (passenger, freight, or mixed (for example auto trains)), or are aircraft, or are ships.

Multi-modality seemingly complicates matters: the ability to move along a route, by a car along a road, transferring to a train at a junction which is both a street intersection and a trail station, then transferring the train onto a ship at a station/harbour junction, and so on.

1.2 The TripTych Paradigm of Software Development

There is the dogma:

- Before software can be designed its requirements must be understood.
- Before requirements can be prescribed the underlying application domain must be understood.

- To understand the requirements their prescription must both
  - ⋆ be in the natural, professional language of the domain and of software engineering
  - ⋆ and must be formalised.
- To understand the domain its description must both
  - ⋆ be in the natural, professional language of the domain
  - ⋆ and must be formalised.

And there are therefore the consequences of the dogma:

- In software development
  - ⋆ we must first — and carefully — describe the domain,
  - ⋆ then, based on the domain description, prescribe the requirements,
  - ⋆ and, finally, based on the requirements prescription, design the software.
- The phases outlined above are linked:
  - ⋆ precise, formal relations can be established between the domain description and the requirements prescription, and
  - ⋆ precise, formal relations can be established between the requirements prescription and the software specification.

The consequences, we believe, of adhering to the above phase-wise development. are:

- That the emphasis
  - ⋆ on domain prescription as a prerequisite for requirements prescription secures the right computing system;
- and the emphasis
  - ⋆ on requirements prescription as a prerequisite for software specification secures that the computing system is right.

1.3 Aims of This Report

The aims of this report are twofold:

- to illustrate, to the reader, the essentials of the triptych paradigm of software engineering including what we consider the benefits of formalisation, and
- to illustrate, independently of the larger scope of the full development, the more narrow, but also, by far, the more novel aspect of the span of domain description.

That is, the really new thing here is the wide-ranging description which soon follows, of a description, informal and formal, of a domain, in this case the multi-modal transport domain.

1.4 Objectives of This Report

The objectives of this report are correspondingly twofold:

- to hopefully convince the reader that that is how software ought be professionally developed: first domain description, then requirements prescription and finally software specification (design and coding), and
- to hopefully further spur an area of research for computing science, here called domain theories or domain engineering.

1.5 Structure of This Report

It should therefore come as no surprise that the structure of this report follows the dogma outlined above. The next three chapters shall therefore outline a domain description of transportation nets, a small variety of requirements, and a similar variety of software designs.

The main reference to the triptych approach is [1].

Chapter 2

A Domain Model of Transportation

2.1 Net Topology

We conceptualise as segments the physically manifest phenomena of roads (between adjacent street intersections), rail tracks (between adjacent train stations), airlanes (between adjacent airports) and shpping lanes (between adjacent harbours). We likewise conceptualise as junctions street intersections, train stations, airports and harbours.

2.1.1 Nets, Segments and Junctions

1. Nets consists of one or more segments and two or more junctions.

   type
       N, S, J
   value
       obs_Ss: N $\rightarrow$ S-**set**
       obs_Js: N $\rightarrow$ J-**set**
   axiom
       $\forall$ n:N $\bullet$ **card** obs_Ss(n) $\geq 1 \wedge$ **card** obs_Js(n) ≥ 2

   Annotations:

   - N, S, J are considered abstract types, i.e., sorts. N, S and J are type names, i.e., names of types of values. Values of type N are nets, values of type S are segments and values of type J are junctions.
   - One can observe from nets, n, their (one or more) segments (obs_Ss(n)) and their (two or more) junctions (obs_Js(n)); n is a value of type N.
   - Functions have names, obs_Ss, and obs_Cs, and functions, f, have signatures, f: A $\rightarrow$ B (not illustrated), where A and B are type names. A designates the definition set of f and B the range set.

- A-**set** is a type expression. It denotes the type whose values are finite, possibly empty set of A values.
- These observer functions are postulated.
- They cannot be formally defined.
- They are "defined" once a net has been pointed out[1]
- The axiom expresses that in any net there is at lest one segment and at least two junctions.

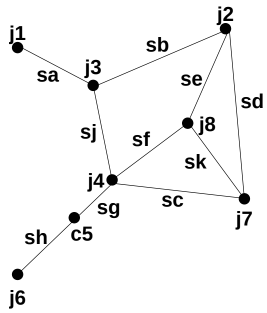


Figure 2.1: A simple net of segments and junctions

Applying the observer functions to the net of Fig. 2.1 yields:

obs_Ss(n) = {sa,sb,sc,sd,se,sf,sg,sh,sj,sk}
obs_Js(n) = {j1,j2,j3,j4,j5,j6,j7,j8}

Nets, segments and junctions are physically manifest, i.e., are phenomena.

2.1.2 Segment and Junction Identifications

2. We now assume that segments and junctions have unique identifications.

type
  Si, Ji
value
  obs_Si: S $\rightarrow$ Si
  obs_Ji: J $\rightarrow$ Ji

Segment and junction identifications are mental concepts.

[1] Take the transportation net Europe. By inspecting it, and by deciding which segments and which associated junctions to focus on (i.e., "the interesting ones") we know which are all the interesting roads, rail tracks, airlanes and shipping lanes, respectively the interesting (associated) street intersections, trains stations, airports and harbours.

3. No two segments have the same segment identifier. And no two junctions have the same junction identifier.

axiom
  ∀ n:N • **card** obs_Ss(n) ≡ **card** {obs_Si(s)|s:S • s ∈ obs_Ss(n)}
  ∀ n:N • **card** obs_Js(n) ≡ **card** {obs_Ji(c)|j:J • j ∈ obs_Js(n)}

Annotations:

- **card** set expresses the cardinality of the set set, i.e., its number of distinct elements.
- {f(a)|a:A • p(a)} expresses the set of all those B elements f(a) where a is of type A and has property p(a) [where we do not further state f, A and B. p is a predicate, i.e., a function, here from A into truth values of type **Bool**, for Boolean].
- The axioms now express that the number of segments in n is the same as the number of segment identifiers of n — which is a circumscription for: No two segments have the same segment identifier.
- Similar for junctions.

The constraints that limit identification of segments and junctions can be physically motivated: Think of the geographic (x, y, z co-ordinate) point spaces "occupied" by a segment or by a junction. They must necessarily be distinct for otherwise physically distinct segments and junctions. Segments may thus cross each other without the crossing point (in x, y space) being a junction, but, for example, one segment may, at the crossing point be physically above the other segment (tunnels, bridges, etc.).

2.1.3 Segment and Junction Reference Identifications

4. Segments are delimited by two distinct junctions. From a segment one can also observe, obs_Cis, the identifications of the delimiting junctions.

type
  Jip = {|{ji,ji′}:Ji-**set** • ji≠ji′|}
value
  obs_Jis: S → Jip

Annotations:

- {|a:A • p(a)|} is a subtype expression. It expresses a subset of type A, namely those A values which enjoys property p(a) [p is a predicate, i.e., a function, here from A into truth values in the type **Bool**]. In the above p(a) is ji≠ji′.

- In this case Jip is the subtype of Ji-**set** whose values are exactly 2 element sets of Ji elements.

5. Any junction has a finite, but non-zero number of segments connected to it. From a junction one can also observe, obs_Sis, the identifications of the connected segments.

type
  Si1 = {|sis:Si-**set**•**card** sis ≥1|}
value
  obs_Sis: J → Si1

Annotations:

- Si1 is the type whose values are non-empty, but still finite sets of Si values.

One cannot from a segment alone observe the connected junctions. One can only refer to them. Similarly: one cannot from a junction alon observe the connected segments. One can only refer to them. The identifications serve the role of being referents.

6. In any net, if s is a segment connected to connectors identified by ji and ji′, respectively, then there must exist connectors j and j′ which have these identifications and such that the identification si of s is observable from both j and j′.

axiom
  ∀ n:N, s:S • s ∈ obs_Ss(n) ⇒
    let {ji,ji′} = obs_Jis(s) **in**
    ∃! j,j′:J • {j,j′}⊆obs_Js(n) ∧ j≠j′ ∧
      obs_Si(s) ∈ obs_Sis(c) ∩ obs_Sis(c′) **end**

Annotations:

- We read the above axiom:
  - ⋆ for all nets n and for all segments s in n
  - ⋆ let ji and ji′be the two distinct junction identifications observable from s, then
  - ⋆ exists exactly two distinct junctions, j and j′ of the net, such that
  - ⋆ the segment identification of s is in both the sets of segment identifications observable from j and j′.

Figure 2.2 on the facing page illustrates the relation between observed identifications of segments and junctions.

The above constraints take on the mantle of being laws of nets: If segments and junctions otherwise have distinct identifications, then the above must follow as a law of man-made artifacts.

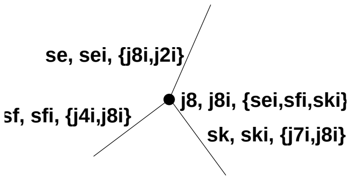


Figure 2.2: One junction and its connected segments

7. Vice-versa: In any net, if j is a junction connecting segments identified by si, si′, . . . , si″ then there must exist segments s, s′, . . . , s″ which have these identifications and such that the identification ji of j is observable from all s, s′, . . . , s″.

axiom
  ∀ n:N, j:J • j ∈ obs_Js(n) ⇒
    let sis = obs_Sis(c), ji = obs_Ji(j) **in**
    ∃! ss:S-**set** • ss⊆obs_Ss(n) ∧ **card** ss=**card** sis ∧
    sis = {|obs_Si(s)|s:S•s ∈ ss|} **end**

Annotations:

- Let us read the above axiom:
  - ⋆ for all nets, n, and all junctions, j, of that net
  - ⋆ let sis be the set of segment identifications observed from j, and let ji be the junction identifier of j, then
  - ⋆ there exists a unique set, ss, of segments of n with as many segments as there are segment identifications in sis, and such that
  - ⋆ sis is exactly the set of segment identifications of segments in ss.

2.1.4 Paths and Routes

8. By a path we shall understand a triplet of a junction identification, a segment identification and a junction identification.

type
  P = Ji × Si × Ji
value
  paths: N → P-**set**
  paths(n) ≡
    {(ji,si,ji′)|s:S,ji,ji′:Ji,si:Si• s ∈ obs_Ss(n)∧{ji,ji′} ∈ obs_Jis(s)∧si=obs_Si(s)}

Annotations:

- Paths are modelled as Cartesians.
- One can generate all the paths of a net.
- It is the set of path triplets, two for each segment of the net and such that the pair of junction identifications, ji and ji′, observable from a segment is at either "end" of the triplet, and such that the segment identification is common to the two triplets (and in the "middle").

Paths, and as we shall see next, routes are mental concepts.

9. By a route of a net we shall understand a list, i.e., a sequence of paths as follows:
   - A sequence of just one path of the net is a route.
   - If r and r′ are routes of the net such that the last junction identification, ji, of the last path, (_,_,ji) of r and the first junction identification, ji′, of the first path (ji′,_,_) of r′ are the same, i.e., ji=ji′, then $r\hat{\ }r'$ is a route.
   - Only routes that can be generated by uses of the first (the basis) and the second (the induction) clause above qualify as proper routes of a net.

```
type
   R = {|r:P*•wf_R(r)|}
value
   wf_R: P* → Bool
   wf_R(r) ≡
      ∀ i:Nat • {i,i+1}⊆inds(r) ⇒
         let (_,_,ji)=r(i), (ji′,_,_)=r(i+1) in ji = ji′ end

   routes: N → R-infset
   routes(n) ≡
      let rs = {⟨p⟩|p:P•p ∈ paths(n)}
               ∪ {r^r′|r,r′:R•{r,r′}⊆rs∧wf_R(r^r′)} in
      rs end
```

Annotations:

- Routes are well-formed sequences of paths.
- A sequence of paths is a well-formed route if adjacent path elements of the route share junction identification.
- Give a net we can compute all its routes as follows:
  - ⋆ let rs be the set of routes to be computed. It consists first of all the single path routes of the net.

- ⋆ Then rs also contains the concatenation of all pairs of routes, r and r′, such that these are members of rs and such that their concatenation is a well-formed route.
- ⋆ If the net is circular then the set rs is an infinite set of routes. The least fix point of the recursive equation in rs is the solution to the "routes" computation.

2.1.5 Segment and Junction Identifications of Routes

10. For future purposes we need be able to identifiy various segment and junction identifications as well as various segments and junctions of a route.

value

$\mathrm{xtr\_Jis}\colon \mathrm{R} \to \mathrm{Ci}\textbf{-set},\ \mathrm{xtr\_Sis}\colon \mathrm{R} \to \mathrm{Si}\textbf{-set}$
$\mathrm{xtr\_Jis(r)} \equiv \textbf{case}\ \mathrm{r}\ \textbf{of}\ \langle\rangle \to \{\},\ \langle(\mathrm{ji},\_,\mathrm{ji}')\rangle\widehat{\ }\mathrm{r}' \to \{\mathrm{ji},\mathrm{ji}'\}\cup \mathrm{xtr\_Jis(r')}\ \textbf{end}$
$\mathrm{xtr\_Sis(r)} \equiv \textbf{case}\ \mathrm{r}\ \textbf{of}\ \langle\rangle \to \{\},\ \langle(\_,\mathrm{si},\_)\rangle\widehat{\ }\mathrm{r}' \to \{\mathrm{si}\}\cup \mathrm{xtr\_Sis(r')}\ \textbf{end}$

$\mathrm{xtr\_Ss}\colon \mathrm{N} \times \mathrm{Ji} \to \mathrm{S}\textbf{-set}$
$\mathrm{xtr\_Ss(n,ji)} \equiv \{\mathrm{s}|\mathrm{s{:}S}\bullet \mathrm{s} \in \mathrm{obs\_Ss(n)} \wedge \mathrm{ji} \in \mathrm{obs\_Jis(s)}\}$

$\mathrm{xtr\_C}\colon \mathrm{N} \times \mathrm{Ji} \to \mathrm{C},\ \mathrm{xtr\_S}\colon \mathrm{N} \times \mathrm{Si} \to \mathrm{S}$
$\mathrm{xtr\_C(n,ji)} \equiv \textbf{let}\ \mathrm{j{:}J} \bullet \mathrm{j} \in \mathrm{obs\_Js(n)} \wedge \mathrm{ji{=}obs\_Ji(j)}\ \textbf{in}\ \mathrm{j}\ \textbf{end}$
$\mathrm{xtr\_S(n,si)} \equiv \textbf{let}\ \mathrm{s{:}S} \bullet \mathrm{s} \in \mathrm{obs\_Ss(n)} \wedge \mathrm{si{=}obs\_Si(s)}\ \textbf{in}\ \mathrm{s}\ \textbf{end}$

$\mathrm{first\_Ji}\colon \mathrm{R} \stackrel{\sim}{\to} \mathrm{Ji},\ \mathrm{last\_Ji}\colon \mathrm{R} \stackrel{\sim}{\to} \mathrm{Ji}$
$\mathrm{first\_Ji(r)} \equiv \textbf{case}\ \mathrm{r}\ \textbf{of}\ \langle\rangle \to \textbf{chaos},\ \langle(\mathrm{ji},\_,\_)\rangle\widehat{\ }\mathrm{r}' \to \mathrm{ji}\ \textbf{end}$
$\mathrm{last\_Ji(r)} \equiv \textbf{case}\ \mathrm{r}\ \textbf{of}\ \langle\rangle \to \textbf{chaos},\ \mathrm{r}'\widehat{\ }\langle(\_,\_,\mathrm{ji})\rangle \to \mathrm{ji}\ \textbf{end}$

$\mathrm{first\_Si}\colon \mathrm{R} \stackrel{\sim}{\to} \mathrm{Si},\ \mathrm{last\_Si}\colon \mathrm{R} \stackrel{\sim}{\to} \mathrm{Si}$
$\mathrm{first\_Si(r)} \equiv \textbf{case}\ \mathrm{r}\ \textbf{of}\ \langle\rangle \to \textbf{chaos},\ \langle(\_,\mathrm{si},\_)\rangle\widehat{\ }\mathrm{r}' \to \mathrm{si}\ \textbf{end}$
$\mathrm{last\_Si(r)} \equiv \textbf{case}\ \mathrm{r}\ \textbf{of}\ \langle\rangle \to \textbf{chaos},\ \mathrm{r}'\widehat{\ }\langle(\_,\mathrm{si},\_)\rangle \to \mathrm{si}\ \textbf{end}$

$\mathrm{first\_J}\colon \mathrm{R} \times \mathrm{N} \stackrel{\sim}{\to} \mathrm{J},\ \mathrm{last\_J}\colon \mathrm{R} \times \mathrm{N} \stackrel{\sim}{\to} \mathrm{J}$
$\mathrm{first\_J(r,n)} \equiv \mathrm{xtr\_J(first\_Ji(r),n)}$
$\mathrm{last\_J(r,n)} \equiv \mathrm{xtr\_J(last\_Ji(r),n)}$

$\mathrm{first\_S}\colon \mathrm{R} \times \mathrm{N} \stackrel{\sim}{\to} \mathrm{S},\ \mathrm{last\_S}\colon \mathrm{R} \times \mathrm{N} \stackrel{\sim}{\to} \mathrm{S}$
$\mathrm{first\_S(r,n)} \equiv \mathrm{xtr\_S(first\_Si(r),n)}$
$\mathrm{last\_S(r,n)} \equiv \mathrm{xtr\_S(last\_Si(r),n)}$

Annotations:

- Given a route one can extract the set of all its junction identifications.

    - ⋆ If the route is empty, then the set is empty.
    - ⋆ If the route is not empty than it consists of at least one path and the set of junction identifications is the pair of junction identifications of the path together with set of junction identifications of the remaining route.
    - ⋆ Possible double "counting up" of route adjacent junction identifications "collapse", in the resulting set into one junction identification. (Similarely for cyclic routes.)
- Given a route one can similarly extract the set of all its segment identifications.
- Given a net and a junction identification one can extract all the segments connected to the identified junction.
- Given a net and a junction identification one can extract the identified junction.
- Given a net and a segment identification one can extract the identified segment.
- Given a route one can extract the first junction identification of the route.
    - ⋆ This extraction should not be applied to empty routes.
    - ⋆ A non-empty route can always be thought of as its first path and the remaining route. The first junction identification of the route is the first junction identification of that (first) path.
- Given a route one can similarly extract the last junction identification of the route.
- Given a route one can similarly extract the first segment identification of the route.
- Given a route one can similarly extract the last segment identification of the route.
- And similarly for extracting the first and last junctions, respectively first and last segments of a route.

2.1.6 Circular and Pendular Routes

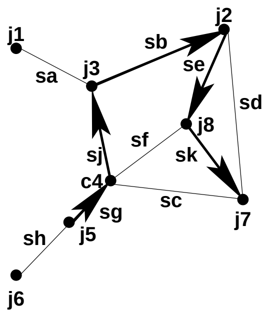
11. A route is circular if the same junction identification either occurs more than twice in the route, or if it occurs as both the first and the last junction identification of the route. Given a net we can compute the set of all non-circular routes by omitting from the above pairs of routes, r and r′, where the two paths share more than one junction identification.

```
non_circular_routes: N → R-set
non_circular_routes(n) ≡
    let rs = {⟨p⟩|p:P•p ∈ paths(n)}
            ∪ {r^r′|r,r′:R•{r,r′}⊆rs∧wf_R(r^r′)∧non_circular(r,r′)} in
    rs end
```

non_circular: R×R → **Bool**
non_circular(r,r′) ≡ **card** xtr_Jis(r) ∩ xtr_Jis(r′) =1

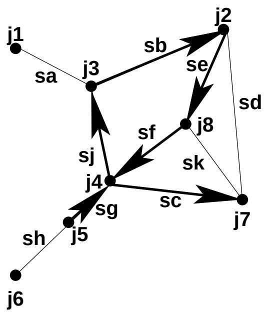
Annotations:

- To express the finite set of all non-circular routes
  - ⋆ is to re-express the set of all routes
  - ⋆ except contrained by the further predicate: non_circular.
- An otherwise well-formed route consisting of a first part r and a remaining part r′
  - ⋆ is non-circular if the two parts share at most one junction identification.



<(j5i,sgi,j4i),(j4i,sji,j3i),(j3i,sbi,j2i),(j2i,sei,j8i)>

Figure 2.3: A route, graphically and as an expression



<(j5i,sgi,j4i),(j4i,sji,j3i),(j3i,sbi,j2i),(j2i,sei,j8i),(j8i,sfi,j4i),(j4i,sci,j7i)>

Figure 2.4: A circular route, graphically and as an expression

12. Let a path be (ji_f, si, ji_t), then (ji_t, si, ji_f) is a *reverse path*. That is: the two junction identifications of a path are reversed in the reverse path. A route, rr, is the reverse route of a route r if the ith path of rr is the reverse path of the $n-i+1$'st

path of r where n is the length of the route r, i.e., its number of paths. A route is a *pendular* route if it is of an even length and the second half (which is a route) is the reverse of the first half route.

value
  reverse: $\mathrm{P} \to \mathrm{P}$
  reverse(jif,si,jit) $\equiv$ (jit,si,jif)

  reverse: $\mathrm{R} \to \mathrm{R}$
  reverse(r) $\equiv$
    case r **of**
      $\langle\rangle \to \langle\rangle$,
      $\langle(\mathrm{jif},\mathrm{si},\mathrm{jit})\rangle\widehat{\ }\mathrm{r}' \to \mathrm{reverse}(\mathrm{r}')\widehat{\ }\langle(\mathrm{jit},\mathrm{si},\mathrm{jif})\rangle$
    end

  reverse(r) $\equiv \langle\mathrm{reverse}(\mathrm{r}(\mathrm{i}))|\mathrm{i}\ \mathbf{in}\ [\,\mathrm{n}..1\,]\rangle$

  pendular: $\mathrm{R} \to \mathrm{R}$
  pendular(r) $\equiv \mathrm{r}\widehat{\ }\mathrm{reverse}(\mathrm{r})$

  is_pendular(r) $\equiv \exists\ \mathrm{r}',\mathrm{r}'':\mathrm{R} \bullet \mathrm{r}'\widehat{\ }\mathrm{r}'' = \mathrm{r} \wedge \mathrm{r}''=\mathrm{reverse}(\mathrm{r}')$

Annotations:

- The reverse of a path is a path with the same segment identification, but with reverse junction identifications.
- The reverse of a route, r, is
  - ⋆ the empty route if r is empty, and otherwise
  - ⋆ it is the reverse route of all of r except the first path of r concatenate (juxtaposed) with the singleton route of the reverse path of the first path of r.
- Given a route, r, we can constuct a pendular route whose first half is the route r and whose last half is the reverse route of r.
- A (an even length) route is a pendular route if it can be expressed as the concatenation of two (equal length) routes, r′ and r″ such that r″ is the reverse of r′, that is, if its second half is the reverse of its first half.

2.1.7 Connected Nets

13. A net is connected if for any two junctions of the net there is a route between them.

value
  is_connected: N $\rightarrow$ **Bool**
  is_connected(n) $\equiv$
    $\forall$ j,j′:J • {j,j′}$\subseteq$obs_Js(n) $\wedge$ j$\neq$j′ $\Rightarrow$
      let (ji,ji′) = (obs_Ji(j),obs_Ji(j′)) **in**
      $\exists$ r:R • r $\in$ routes(n) $\wedge$
        first_Ji(r) = ji $\wedge$ last_Ji(r) = ji′ **end**

Annotations:

- A net n is connected if
  - ⋆ for all two distinct connectors of the net
  - ⋆ where ji and ji′ are their junction identifications,
  - ⋆ there exists a route, r, of the net,
  - ⋆ whose first junction identification is ji and whose last junction identification is ji′.

2.1.8 Net Decomposition

14. One can decompose a net into all its connected subnets. If a net exhaustively consists of m disconnected nets, then for any pair of nets in different disconnected nets it is the case that they share no junctions and no segments. The set of disconnected nets is the smallest such set that together makes up all the segments and all the junctions of the ("original") net.

value
  decompose: N $\rightarrow$ N-**set**
  decompose(n) **as** ns
    obs_Ss(n) = $\cup$\{obs_Ss(n′)|n′:N•n′ $\in$ ns\} $\wedge$
    obs_Js(n) = $\cup$\{obs_Js(n′)|n′:N•n′ $\in$ ns\} $\wedge$
    {} = $\cap$\{obs_Ss(n′)|n′:N•n′ $\in$ ns\} $\wedge$
    {} = $\cap$\{obs_Js(n′)|n′:N•n′ $\in$ ns\} $\wedge$
    $\forall$ n′:N•n′ $\in$ ns $\Rightarrow$ connected(n′) $\wedge$ ...

Annotations:

- A set ns of nets constitutes a decomposition of a net, n,
  (a) if all the segments of n appear in some net of ns,
  (b) if all the junctions of n appear in some net of ns,
  (c) if no two or more distinct nets of ns share segments,
  (d) if no two or more distinct nets of ns share junctions, and

(e) if all nets of ns are connected.

- **Comment:** It appears that items 3 and 4 are unnecessary, that is, are properties once items 1, 2 and 5 hold.

That is, we have the following:

Lemma:
$\forall$ n:N •
  let ns = decompose (n) **in**
  $\forall$ n′,n″:N • {n′,n″}$\subseteq$ns $\wedge$ n′$\neq$n″ $\Rightarrow$
    obs_Ss(n′) $\cap$ obs_Ss(n″) = {} $\wedge$
    obs_Js(n′) $\cap$ obs_Js(n″) = {} **end**

The above 14 items define a lot of what there is to know about transportation nets if we only operate with the sorts that have been introduced (N, S, Si, J, Ji) and the observer functions that have likewise been introduced (obs_Ss, obs_Js, obs_Si, obs_Ji, obs_Jis and obs_Sis). The relationships between sorts, i.e., net, segment, segment identification, junction and junction identification values are expressed by the axioms. The above is a so-called property-oriented model of the topology of transportation nets. That model is abstract in that it does not hint at a mathematical model or at a data structure representation of nets, segments and junctions, let alone their topology. By topology we shall here mean how segments and junctions are "wired up". The axioms above guarantee that no segment of a net is left "dangling": It is always connected to two distinct junctions; and no junctions of a net is left isolated: It is always connected to some segments of the net.

We have tacitly assumed that all segments are two way segments, that is, transport can take place i either direction. Hence a segment gives rise to two paths.

2.2 Multi-Modal Nets

Interesting transportation nets are multi-modal. That is, consists of segments of different transport modalities: roads, rails, air-lanes, shipping lanes, and, within these of different categories. Thus roads can be either freeways, motor-ways, ordinary highways, and so on.

2.2.1 General Issues

15. We introduce a concept, M, of transport mode. M is a small set of distinct, but otherwise further undefined tokens. An m in M designates a transport modality.

type
  M

2.2.2 Segment and Junction Modes

16. With each segment, s, we can associate a single mode, m, and with each junction we can associate the set of modes of its connected segments.

value
  obs_M: S → M
  obs_Ms: J → M-**set**
axiom
  ∀ n:N, j:J • j ∈ obs_Js(n) ⇒
    let ss = xtr_Ss(n,obs_Ji(j)) **in**
    obs_Ms(j) = {obs_M(s)|s:S • s ∈ ss} **end**
  ∀ n:N, s:S • s ∈ obs_Ss(n) ⇒
    let {ji,ji′} = obs_Jis(s) **in**
    let {j,j′} = {xtr_J(n,ji),xtr_J(n,ji′)} **in**
    obs_M(s) ∈ obs_Ms(j) ∩ obs_Ms(j′) **end end**

Annotations:

- From a segment one can observe its mode.
- From a junction one can observe its set of modes.
- Let us read the first axiom:
  - ⋆ for all net, n, and all junctions, j, of that net
  - ⋆ let ss be the set of segments connected to j,
  - ⋆ now the set of modes of c is equal to the set of modes of the segments in ss.
- Let us read the second axiom:
  - ⋆ for all net, n, and all segments, s, of that net
  - ⋆ let ji and ji′ be the junction identifiers of the two junctions to which s is connected, and
  - ⋆ let j and j′ be the two corresponding junctions,
  - ⋆ then the segment mode is in both the set of modes of the two junctions.
- We can define a function, xtr_Ss, which from a net, n, and a junction identification, ji, extracts the set of segments, ss, connected to the junction identified by ji.
- xtr_Ss(n,ji) yields the set of segments, ss, in the net n for which ji is one of the observed junction identifications of s.
- And we can define a function, xtr_J, of signature N × Ji → J, which when applied to a net, n, and a junction identification, ji,
- extracts the junction in the net which has that junction identifier.

2.2.3 Single-Modal Nets and Net Projection

17. Given a multi-modal net one can project it onto a set og single modality nets, namely one for each modality registered in the multi-modal net.

type
  mmN = {|n:N • **card** xtr_Ms(n) > 1|}
  smN = {|n:N • **card** xtr_Ms(n) = 1|}
value
  xtr_Ms: N → M-**set**
  xtr_Ms(n) ≡ {obs_M(s) | s:S • s ∈ obs_Ss(n)}

  projs: N → smN-**set**
  projs(n) ≡ {proj(n,m) | m:M • m ∈ xtr_Ms(n)}

  proj: N × M → smN
  proj(n,m) **as** n′
    post
      let ss = obs_Ss(n), ss′ = obs_Ss(n′),
          js = obs_Js(n), js′ = obs_Js(n′) **in**
      ss′ = {s | s:S • s ∈ ss ∧ m=obs_M(s)} ∧
      js′ = {j | j:J • j ∈ js ∧ m ∈ obs_Ms(j)}
      end

Annotations:

- A multi-modal net is a net with more than one mode. mmN is thus the subtype of nets, n:N, which are multi-modal.
- A single-modal net is a net with exactly one mode. smN is thus the subtype of nets, n:N, which are multi-modal.
- The xtr_Ms function extracts the mode of every segment of a net.
- The projs function applies to any net, n:N, and yields the set of single-modal subnets of n, one for each mode of n. The projs function makes use of the proj function.
- The proj function applies to any n, n:N, and any mode of that net, and yields the single-modal subnet on n whose mode is the given mode.
  - ⋆ The proj function is expressed by a post condition, i.e., a predicate that characterises the necessary and sufficient relation between the argument net, n, and the result net n′.
  - ⋆ In a single-modal net, n′, projected from a multi-modal net, n, and of mode m, we keep exactly those segments, ss′, of n whose mode is m,
  - ⋆ and we keep exactly those junctions, js′, of n whose mode contains m.

⋆ No more is needed in order to express the necessary and sufficient condition for a single-modal net to be a subsnet of a proper net.

⋆ That is, some single-modal nets are not proper nets since in proper nets every junction have the set of modes of all the segments connected to the junction.

2.3 Segment and Junction Attributes

2.3.1 Segment and Junction Attribute Observations

We now enrich our segments and junctions.

18. Segments have lengths.

19. Junctions have modality-determined lengths between pairs of (same such modality) segments connected to the junction.

20. Segments have standard transportation times, i.e., time durations that it takes to transport any number of units of freight from one end of the segment to the other.

21. Junctions have standard transfer time per modality of transport between pairs of segments connected to the junction.

22. Junctions have standard arrival time per modality of transport.

23. Junctions have standard departure times per modality of transport.

24. Segments have standard costs of transporting a unit of freight from one end of the segment to the other end.

25. Junctions have standard costs of transporting a unit of freight from the end of one connecting segment to the beginning of another connecting segment.

We can now assess

- (i) length of a route,
- (ii) shortest routes between two junctions,
- (iii) duration time of standard transport along a route, including transfer, stopover and possible reloading times at junctions, and
- (iv) shortest duration time route of standard transport between two junctions.

type
  L, TI
value
  ms:M-**set**, **axiom** ms$\neq${}
  obs_L: S $\rightarrow$ L
  obs_L: Si $\times$ J $\times$ M $\times$ Si $\rightarrow$ L
  obs_TI: S $\rightarrow$ TI
  obs_TI: Si $\times$ J $\times$ Si $\rightarrow$ TI
  obs_TI: J $\times$ M $\tilde{\rightarrow}$ TI, **pre** obs_TI(j,m): m $\in$ obs_Ms(j)
  obs_TI: J $\times$ M $\times$ M $\tilde{\rightarrow}$ TI, **pre** obs_TI(j,m,m$'$): {m,m$'$}$\subseteq$obs_Ms(j)
  obs_arr_TI: J $\times$ M $\tilde{\rightarrow}$ TI, **pre** obs_arr_TI(j,m): m $\in$ obs_Ms(j)
  obs_dep_TI: J $\times$ M $\tilde{\rightarrow}$ TI, **pre** obs_dep_TI(j,m): m $\in$ obs_Ms(j)
  +: L $\times$ L $\rightarrow$ L
  +: TI $\times$ TI $\rightarrow$ TI

Annotations:

- L and Ti are sorts designating length and time values.
- ms denotes a non-empty set of modes.
- From a segment one can observe, obs_L, its length.
- From a segment one can observe, obs_TI, a time duration for a normal conveyour of the mode of the segment to travel the length of the segment.
- From a junction and a mode (of that junction) one can observe, obs_TI, a time duration for a normal conveyour of the mode to cross, i.e., to travel through the junction.
- From a junction and a pair of modes (m and m$'$ of that junction) one can observe, obs_TI, a time duration which represents the normal time it takes to transfer freight from a conveyour of mode m to a conveyour of mode m$'$. (The two modes may be the same.)
- From a junction and a mode (of that junction) one can observe, obs_arr_TI, a time duration for an item of freight destined for a normal conveyour of the mode to arrive and be “entry” processed (including loaded) at that junction.
- From a junction and a mode (of that junction) one can observe, obs_dep_TI, a time duration for an item of freight destined for a normal conveyour of the mode to arrive and be “exit” processed (including unloaded) at that junction.
- One can add lenths.
- One can add time durations.

2.3.2 Route Lengths

26. One can compute the length of a route of a net and one can find the shortest such route between two identified junctions.

```
value
   length: R → N ~→ L
   length(r)(n) ≡
      case r of
         ⟨⟩ → 0,
         ⟨(jf,si,jt)⟩ → obs_L(xtr_S(si,n)),
         ⟨(ji1,sii,ji2),(jj1,sij,jj2)⟩^r′ →
            let si=xtr_S(sii,n),sj=xtr_S(sij,n) in
            obs_L(si) + obs_L(sii,xtr_J(ji2,n),sij) + length(⟨(jj1,sij,jj2)⟩^r′) end
      end
      pre: r ∈ routes(n) ∧ ji2=jj1

value
   shortest_route: Ji × Ji → N ~→ R
   shortest_route(jf,jt)(n) ≡
      let rs = routes(n) in
      let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} in
      let sr:R • sr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ length(r)(n)<length(sr)(n) in
      sr end end end
      pre: {jf,jt}⊆obs_Jis(n) ∧ jf≠jt
```

Annotations:

- The length of a single modality route of a net
  - ⋆ is 0 if the route is empty,
  - ⋆ otherwise it is the length of the first segment of the route plus the length of the rest of the route computed as follows:
    - ⋄ If the route consists of just one segment, then 0,
    - ⋄ else, the length of the junction from incident segment to emanating segment plus
    - ⋄ the length of the rest of the route computed as otherwise specified above.
- The shortest route of a net between two of its identified junctions (the precondition) can be abstractly determined as follows:
  - ⋆ First we find all the routes, rs, of the net.
  - ⋆ Then we find those routes, crs, whose first and last connnection identifications are the given ones, cf and ct.
  - ⋆ Amongst those we find a shortest one, that is, one, in crs, for which there are no shorter routes, r, in crs.

2.3.3 Route Traversal Times

27. One can find the total time it takes to traverse a route, including the times it takes to pass through a junction, and one can find the quickest route between two identified junctions.

```
all_time: R → N → TI
all_time(r)(n) ≡
    obs_arr_TI(xtr_J(first_J(r),n),obs_M(first_S{r}))
    + time(r)(n)
    + obs_dep_TI(xtr_J(last_J{r},n),obs_M(last_S(r)))

time: R → N → TI
time(r)(n) ≡
    case r of
        ⟨⟩ → 0,
        ⟨(jf,si,jt)⟩ → obs_TI(xtr_S(si,n)),
        ⟨(ji1,sii,ji2),(jj1,sij,jj2)⟩^r′ →
            let si=xtr_S(sii,n),sj=xtr_S(sij,n) in
            obs_TI(si) + obs_TI(sii,xtr_J(ji2,n),sij) + time(⟨(jj1,sij,jj2)⟩^r′) end
    end
    pre: r ∈ routes(n) ∧ ji2=jj1

quickest_route: Ji × Ji → N → R
quickest_route(jf,jt)(n) ≡
    let rs = routes(n) in
    let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} in
    let qr:R • qr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ all_time(r)(n)<all_time(qr)(n) in
    qr end end end
```

Annotations:

-
-
-

2.3.4 Function Lifting

28. Notice how the two functions shortest_route and quickest_route differ only by the length, respectively the time functions. Hence:

```
type
    Q
```

FCT = R → N → Q
value
less: Q × Q → **Bool**
lowest: Ji × Ji → N → FCT → R
lowest(jf,jt)(n)(fct) ≡
let rs = routes(n) **in**
let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} **in**
let lr:R • lr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ less(fct(r)(n),fct(qr)(n)) **in**
lr **end end end**

Annotations:

•

•

•

29. Similarely one could also lift the 'less' predicate:

Q
PRE = Q × Q → **Bool**
FCT = R → N → Q
value
best: Ji × Ji → N → FCT → PRE → R
best(cf,ct)(n)(fct)(pre) ≡
let rs = routes(n) **in**
let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=cf ∧ last_Ji(r)=ct} **in**
let br:R • lr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ pre(fct(r)(n),fct(qr)(n)) **in**
br **end end end**

Annotations:

•

•

•

And so on.

2.3.5 Transporation Costs

30. We can further assess (i) transport costs, (ii) lowest (per unit) freight cost between two junctions, etc. We assume that if a freight item is transported into a junction

and out of that junction by the same modality conveyour, then it is not reloaded, i.e., along segments of the same modality.[2]

type
  K, F
value
  obs_K: (S|J) → K
  obs_F: (S|J) → F

  +: K × K → K

  cost: R → N → K
  cost(r)(n) ≡
    case r **of**
      ⟨⟩ → 0,
      ⟨(jf,si,jt)⟩ →
        obs_K(xtr_J(jf,n))+obs_K(xtr_S(si,n))+obs_K(xtr_J(jt,n))
      ⟨(jf,si,jt),(jf′,si′,jt′)⟩^r′ → **assert:** jt=jf′
        obs_K(xtr_J(jf,n))+obs_K(xtr_S(si,n))+...+cost(r′)
    end

  cheapest: Ji×Ji → N → ((K×K)→K) → ((K×K)→**Bool**) → R
  cheapest(jf,jt)(n) ≡
    best(jf,jt)(n)(λ(k1,k2):(K×K)•k1+k2)(λ(k1,k2):(K×K)•k1<k2)

Annotations:

-
-
-

2.4 Road Nets

We wish to view road nets at different levels of abstraction. At a most detailed such level we make no distinction between the road kinds, whether community roads, provincial roads, motor roads or freeways. At another level of abstraction we wish to make exactly those distinctions. And at least detailed level of abstraction we consider certain road junctions to designate road nets of smaller or larger communities.

[2]This grossly simplifying assumption will be removed later. For the time being it allows us to operate with the simple notion of routes that was introduced above. For the reloading case we need to decorate the route notion, effectively maing it into a bill of ladings notion: one that prescribes possible reloading at junctions.

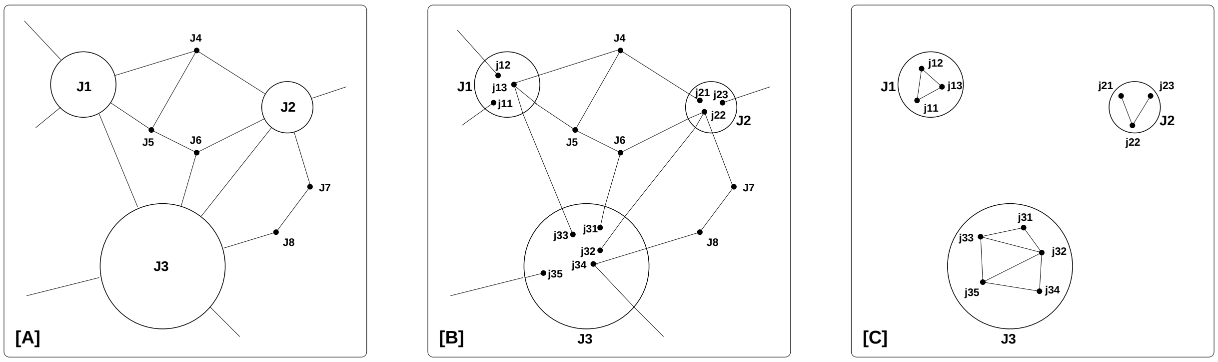


Figure 2.5: Gross [A] versus semi-detailed [B] road net — and community road nets [C]

31. Figure [A] 2.5 shows a road net. Instead of showing junctions J1, J2 and J3 as small black disks we show them as larger circles — for reasons that transpires from Fig. [B] 2.5.
32. Junctions J1, J2 and J3 are considered composite, that is, to represent communities.
33. We may consider the road net of Fig.[A] 2.5 to be an abstraction of the road net hinted at in Fig.[B] 2.5.
34. Junctions j11, j12, ..., j35 are considered simple embedded junctions.
35. We decide to allow three kinds of junctions:
    (a) composite,
    (b) simple embedded and
    (c) simple.

    They are as follows:

    (a) Composite junctions stand for road nets themselves. The junctions of those road nets are all simple embedded junctions.
    (b) Simple embedded junctions are the junctions, hence, of composite junction road nets.
    (c) Simple junctions are those junctions which are not composite (that is: are not standing for road nets) and are not simple embedded junctions (that is: simple, hence un-embedded junctions are those remaining junctions of a net which include modality road).
36. In Fig. [B] 2.5 we have left out the internal roads, that is, segments of junctions J1, J2 and J3, that is between the simple embedded junctions j11, j12 and j13, between j21, j22 and j23, and between j31, j32, j33, j34 ans j35.

37. The internal segments of junctions J1, J2 and J3 are shown in Fig. [C] 2.5 on the preceding page. They are to be considered complete nets "in and by" themselves.
38. We may consider the implied junction identifications Ji1, Ji2 and Ji3 to be names of communities.
39. We may consider the implied junction identifications ji11, ji12 and ji13 to abstract to J1, ji21, ji22 and ji23 to abstract to J2, and ji31, ji32, ji33, ji34 and ji35 to abstract to J3.
40. We shall assume that from these junction identifications, say jikℓ, one can observe the more abstract junction identifications, i.e., Jik.
41. We shall, conversely, assume that from segment junction identifications one can observe whether they are identifications of composite, of simple embedded or of simple junctions, and, if of composite junctions, that one can further observe which simple embedded junction of the composite junction the segment is connected to.
42. In summary: When consider any multi-modality net and from it project, that is, consider only the net, n_r, of modality road, then we may find that some junctions are composite while are are simple. When then examining the road nets, r_n, contained in composite junctions then we will find that their junctions are simple embedded. The embedded road nets, r_n, otherwise satisfy all the properties (i.e., axioms) of nets in general. To link up the segments of n_r incident upon, that is, connected to composite junctions (in n_r) we provide their junction identifications with two levels of observability: the abstract one that made us see that they were connected to composite junctions (cf. Fig. [A] 2.5 on the page before), and a concrete one that enables us to decide which ones of the simple embedded junctions they are "finally" linked to (cf. Fig. [B] 2.5 on the preceding page).

type
  M == road | ...
  Jc, Js, Jse
  Jic, Jis, Jise
  J = Jc | Js | Jse
  Cn
value
  is_composite_J: J → **Bool**
  is_simple_J: J → **Bool**
  is_simple_embedded_J: J → **Bool**
  obs_N: Jc → N
  obs_Jic: Jc → Jic, obs_Jis: Js → Jis, obs_Jise: Jse → Jise
  obs_Cn: Jic → Cn, obs_Cn: Jise → Cn
  obs_Jise: Jic → Jise
axiom

$\forall$ j:Jc • is_composite_J(j) $\wedge$ xtr_Ms(obs_N(j,road))={road},
$\forall$ j:Js • is_simple_J(j),
$\forall$ j:Jse • is_simple_embedded_J(j)

$\forall$ n:N,j:J • j $\in$ obs_Js(n) $\wedge$ is_composite_J(j) $\Rightarrow$
  let rn = obs_N(j) **in**

  end

2.5 Railway Nets

2.5.1 General

A transportation net of modality railway has segments be lines between stations and have junctions be stations.

43. We concretise the concept of modes. Mode m=railway will now designate railway nets:

    type
      M == road | railway | ...

44. From a multi-modal transportation net we can project the railway net, rn:RN:

    value
      proj: N $\times$ {railway} $\rightarrow$ RN

45. Junctions of a transportation net of modality railway have sub-junctions which are stations:

    value
      proj: J $\times$ {railway} $\rightarrow$ ST

46. Segments of a transportation net of modality railway become lines:

    value
      proj: S $\times$ {railway} $\rightarrow$ LI

2.5.2 Lines, Stations, Units and Connectors

Railway segments are thus called lines, and railway sub-junctions are thus called stations. A notion of connectors is introduced. It is not to be confused with the previous notion of junctions.

47. A railway net is a net of mode railway.
48. Its segments are lines of mode railway.
49. Its junctions are stations of mode railway.
50. A railway net consists of one or more lines and two or more stations.
51. A railway net consists of rail units.
52. A line is a linear sequence of one or more linear rail units.
53. The rail units of a line must be rail units of the railway net of the line.
54. A station is a set of one or more rail units.
55. The rail units of a station must be rail units of the railway net of the station.
56. No two distinct lines and/or stations of a railway net share rail units.
57. A station consists of one or more tracks.
58. A track is a linear sequence of one or more linear rail units.
59. No two distinct tracks share rail units.
60. The rail units of a track must be rail units of the station (of that track).
61. A rail unit is either a linear, or is a switch, or a is simple crossover, or is a switchable crossover, etc., rail unit.
62. A rail unit has one or more connectors.
63. A linear rail unit has two distinct connectors. A switch (a point) rail unit has three distinct connectors. Crossover rail units have four distinct connectors (whether simple or switchable), etc.
64. For every connector there are at most two rail units which have that connector in common.
65. Every line of a railway net is connected to exactly two distinct stations of that railway net.

66. A linear sequence of (linear) rail units is an acyclic sequence of linear units such that neighbouring units share connectors.

type
47. RN = {| n:smN • obs_M(n)=railway |}
48. LI = {| s:S • obs_M(s)=railway |}
49. ST = {| c:C • obs_M(c)=railway |}
Tr, U, K

value
50. obs_LIs: RN → LI-**set**
50. obs_STs: RN → ST-**set**
51. obs_Us: RN → U-**set**
52. obs_Us: LI → U-**set**
54. obs_Us: ST → U-**set**
57. obs_Trs: ST → Tr-**set**
61. is_Linear: U → **Bool**
61. is_Switch: U → **Bool**
61. is_Simple_Crossover: U → **Bool**
61. is_Switchable_Crossover: U → **Bool**
62. obs_Ks: U → K-**set**

66. lin_seq: U-**set** → **Bool**
lin_seq(us) ≡
∀ u:U • u ∈ us ⇒ is_Linear(u) ∧
∃ q:U* • **len** q = **card** us ∧ **elems** q = us ∧
∀ i:**Nat** • {i,i+1} ⊆ **inds** q ⇒ ∃ k:K •
obs_Ks(q(i)) ∩ obs_Ks(q(i+1)) = {k} ∧
len q > 1 ⇒ obs_Ks(q(i)) ∩ obs_Ks(q(**len** q)) = {}

axiom
50. ∀ n:RN • **card** obs_LIs(n) ≥ 1 ∧ **card** obs_STs(n) ≥ 2

52. ∀ n:RN, l:LI • l ∈ obs_LIs(n) ⇒ lin_seq(l)

53. ∀ n:RN, l:LI • l ∈ obs_LIs(n) ⇒ obs_Us(l) ⊆ obs_Us(n)

54. ∀ n:RN, s:ST • s ∈ obs_STs(n) ⇒ **card** obs_Us(s) ≥ 1

55. ∀ n:RN, s:ST • s ∈ obs_LIs(n) ⇒ obs_Us(s) ⊆ obs_Us(n)

56. ∀ n:RN,l,l′:LI•{l,l′}⊆obs_LIs(n)∧l≠l′⇒obs_Us(l)∩ obs_Us(l′)={}

56. $\forall$ n:RN,l:LI,s:ST•l $\in$ obs_LIs(n)$\wedge$s $\in$ obs_STs(n)$\Rightarrow$obs_Us(l)$\cap$ obs_Us(s)={}

56. $\forall$ n:RN,s,s′:ST•{s,s′}$\subseteq$obs_STs(n)$\wedge$s$\neq$s′$\Rightarrow$obs_Us(s)$\cap$ obs_Us(s′)={}

57. $\forall$ s:ST•**card** obs_Trs(s)$\geq$1

58. $\forall$ n:RN,s:ST,t:Tr•s $\in$ obs_STs(n)$\wedge$t $\in$ obs_Trs(s)$\Rightarrow$lin_seq(t)

59. $\forall$ n:RN,s:ST,t,t′:Tr•s $\in$ obs_STs(n)$\wedge${t,t′}$\subseteq$obs_Trs(s)$\wedge$t$\neq$t′
    $\Rightarrow$ obs_Us(t) $\cap$ obs_Us(t′) = {}

```
64. ∀ n:RN • ∀ k:K •
        k ∈ ∪{obs_Ks(u)|u:U•u ∈ obs_Us(n)}
            ⇒card{u|u:U•u ∈ obs_Us(n)∧k ∈ obs_Ks(u)}≤2

65. ∀ n:RN,l:LI •  l ∈ obs_LIs(n) ⇒
      ∃ s,s′:ST • {s,s′} ⊆ obs_STs(n) ∧ s≠s′ ⇒
        let sus=obs_Us(s),sus′=obs_Us(s′),lus=obs_Us(l) in
        ∃ u,u′,u″,u‴:U • u ∈ sus ∧
            u′ ∈ sus′ ∧ {u″,u‴} ⊆ lus ⇒
            let sks = obs_Ks(u), sks′ = obs_Ks(u′),
                lks = obs_Ks(u″), lks′ = obs_Ks(u‴) in
            ∃!k,k′:K•k≠k′∧sks ∩ lks={k}∧sks′ ∩ lks′={k′}
        end end
```

2.5.3 Paths, States and Routes

67.

68.

69.

70.

71.

2.5.4 Train Movement and Traffic

72.

73.

74.

75.

76.

2.5.5 Time Tables

77.

78.

79.

80.

81.

2.5.6 Etcetera

82.

83.

84.

85.

86.

2.6 Net Dynamics

By net dynamics we shall mean the changing possibilities of flow of conveyors (cars, trains, aircraft, ships, etc.) along segments and through junctions. We speak of direction of flow along segments in terms of *"from the junction at one end of the segment to the junction at the other end"*. And we speak of flow through a junction as *"proceeding from one segment incident upon the junction into a (udually different) segment emanating from that junction"*. Segments connected to a junction are both incident upon that junction and emanates from that junction.

2.6.1 Segment and Junction States

87. Segments may be open for traffic in either or both directions (between the segments' two junctions [identified by ji_x and ji_y]) or may be closed.

88. We model the state, $s\sigma : S\Sigma$, of a segment, $s : S$, as a set of pairs of junction identifications, namely of the two identifications of the junctions that the segment connects. This state, $s\sigma : S\Sigma$, is

   (a) either empty, i.e., the segment is closed ($\{\}$),
   (b) or has one pair, $\{(ji_x, ji_y)\}$, that is, the segment is open in direction from junction ji_x to junction ji_y,
   (c) or another pair $\{(ji_y, ji_x)\}$,
   (d) or both pairs $\{(ji_x, ji_y), (ji_y, ji_x)\}$, that is, is open in both directions.

89. Junctions may direct traffic from any subset of incident segments to any subset of emanating segments.
90. We model the state, $j\sigma : J\Sigma$, of a junction, $j : J$, as a set of pairs of segment identifications, namely of identifications of segments connected to the junction.
   (a) Let the set of identifications of segments connected to junction j be $\{si_1, si_2, ..., si_m)\}$.
   (b) If, in some state, $j\sigma$ of the junction, it is possible (allowed) to pass through the junction from the segment identified by si_j to the segment identified by si_k, then the pair (si_j, si_k) is in $j\sigma$.
   (c) The junction state may be empty, i.e., closed: no traffic is allowed through the junction.
   (d) Or the junction state may be "anarchic full", that is, it contains all combinations of the pairs of identifiers of segments incident upon the junction.

type
  SΣ = (Ji×Ji)-**set**
  JΣ = (Si×Si)-**set**
value
  obs_SΣ: S → SΣ
  obs_JΣ: J → JΣ

  xtr_Jis: SΣ → Ji-**set**, xtr_Jis(sσ) ≡ {ji|ji:Ji • (ji,_) ∈ obs_sσ ∨ (_,ji) ∈ obs_sσ}
  xtr_Sis: JΣ → Si-**set**, xtr_Sis(jσ) ≡ {si|si:Si • (si,_) ∈ obs_jσ ∨ (_,si) ∈ obs_jσ}
axiom
  ∀ s:S • xtr_Jis(obs_SΣ(s)) ⊆ xtr_Jip(s),
  ∀ j:J • xtr_Sis(obs_JΣ(j)) ⊆ xtr_Sis(j)

Annotations:

-
-
-
-

-
-

Observations:

- A junction, $j : J$, of just one segment, $s : S$, that is, s is a cul de sac, may either be closed, and vehicles trying to enter j will be queued up, or it is open, and vehicles entering j will be lead back to s.
- As a consequence segment s, in order for this latter routing to happen, must be open in both directions when j is "open".
- In general, if the state of a junction j (identified by ji) contains a pair (si_x, si_y) then the state of the designated segments, sx and sy, must respectively contain pairs (ji', ji), respectively (ji, ji''), where $\{ji, ji'\}$ and $(ji, ji''\}$ are the pairs of junction identifications associated with si_x and si_y respectively.
- And this must hold for all states of junctions and adjacent segments.
- This is captured in the axioms below.

axiom
...

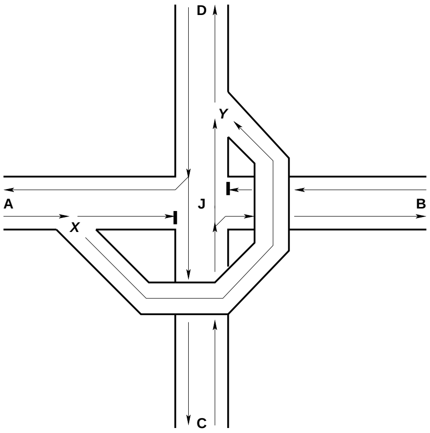


Figure 2.6: A Special "Carrefour" Junction

91. The junction of Fig. 2.6 shows four segments, identified by A, B, C and D.
92. The figure also suggests a state in which traffic lights prohibit movements from A into J, from B into J,

93. from C via J into A, and from D via J into B.
94. The "bypass" from A/X into Y/D appears to be such that traffic can always pass from A into D.
95. The current state alluded to in Fig. 2.6 on the page before appears to be:

$$j\sigma_J : \{(A,D),(C,B),(C,D),(D,A),(D,C)\}$$

96. (A,D) is potentially a member of every state that the junction can possibly be in — see next section.

2.6.2 Segment and Junction State Spaces

97.
98.
99.
100.
101.

type
  SΩ = SΣ-**set**
  JΩ = JΣ-**set**
value
  obs_SΩ: S → SΩ
  obs_JΩ: J → JΩ
axiom
  ∀ s:S • obs_SΣ(s) ⊆ obs_SΩ(s),
  ∀ j:J • obs_JΣ(j) ⊆ obs_JΩ(j)

2.6.3 Open and Closed Routes

102.
103.
104.
105.
106.

value
is_closed_R: R → **Bool**
is_open_R: R → **Bool**

is_closed_R() ≡

is_open_R() ≡

Chapter 3

Requirements Models for Transportation Software

The present chapter will sketch the domain and interface requirements for three different software packages. The three are more or less arbitrarily chosen and are:

- software for the support of maintenance of transportation nets,
- software for the support of logistics planning over transportation nets and
- software for the support of animation, that is, dynamic visualisation of transportation net traffic.

The next section will give a synopsis (an introductory narrative) for each of these three sets of requirements as well as an overview of the sections in which the details are sketched.

The principles and techniques for transforming a domain description to a domain and interface requirements prescription are given in Vol. 3, Part IV, Chap. 19, Sects. 19.4–19.5 of [1].

A basic principle of domain requirements, that is, those requirements which can be expressed solely in terms of domain terminology, is that we stepwise transform a domain model into the domain requirements model.

- **A Summary Domain Model** Sect. 3.2, Pages 48–59

therefore prefixes the three domain requirements models. We have simply copied all formulas of Chap. 2.

From these formulas, and driven by the notions covered in the next three bulletted items, we then apply the domain-to-requirements transformation operations of *projection, determination, instantiation, extension* and *fitting*.

3.1 Overview of Requirements

We now present set of informative texts that — hopefully gently — should lead the reader in the appropriate directions as concerns the three separate requirements are about. Please recall that the scope is the same in all three cases, namely that of transportation nets.

- **Net Maintenance:** Sect. 3.4, Pages 59–69

    1. **The current situation:** Maintenance of transportation nets is becoming an increasingly complex "affair": it is costly, net (segment and junction) breakdowns attract the critical attention of the general public and hence of politicians, preventive — rather than corrective — maintenance is hence becoming "de rigeur"[1], and overhead (i.e., preventive maintenance) costs must be controlled — and are not so well controlled in the specific "transportation net owner".
    2. **Needs and ideas:** There is therefore a need for more rigorous monitoring of when maintenance ought take place and control of such maintenance. The idea is to require a computing system to help in the monitoring and control.
    3. **Span:** The span of the problem is that of road net maintenance.
    4. **Concepts and facilities:** The concepts and facilities of the required computing system are initially thought to be: (1) With every segment and with every junction (i.e., item) of a net is associated a number of maintenance attributes such as: quality state of segment and junction (viz.: a %age which "speaks out" about the "goodness" of the item ranging from 100% tip-top condition to 30% [dangerous state]), recommended time between maintenance reviews, date of last time item was maintained, etc. (2) Every item is represented in a database. (3) Database queries can help (3.1) analyse need for "out-of-order" maintenance, (3.2) list items up for regular maintenance, etc. (4) Database updates can set or compute new values of the maintenance and other attributes of items, etc.
    5. **Synopsis:** The transport net maintenance software shall provide for the timely, that is, "up to date" recording of the maintenance state of a transport net, and for (query and update) functions that allow the user to ascertain the maintenance state of specifiable (i.e., computable) subsets of segments and junctions, to select geographically neigbouring collections of segments and junctions whose maintenance state need be monitored, etc., and for the refreshment (i.e., update) of the maintenance state of segments and junctions.
    6. **Design brief:** The task is, based on an adequate domain model of multi-modal transportation nets, and in collaboration with designated transportation net stakeholders, to prescribe commonly agreed requirements for software to support the maintenance of road transportation net.

---

[1]MSN Encarta Dictionary. de rigeur: strictly required by the current fashion or by etiquette (http://encarta.msn.com)

- **Net Logistics:** Sect. 3.5, Pages 59–81

 1. **The current situation:** In very large multi-modal transportation nets there are many ways of travelling from one junction to another junction. Some means of transport are cheap, some are slow, may require both day and night travel, and some routes imply many transfers between transport facilitators (conveyours). Alternative and/or competing scheduled or regular transports with the above attributes (cost, time, day/night only/both, frequency of transfers, etc.) compound the "picture". The situation easily becomes too "complex" to manually handle the above and other forms of logistics.
 2. **Needs and ideas:** There is therefore a need for more systematic means of evaluating which routes which satisfy transportation properties. The idea is to require a computing system, i.e., a software package to perform transport logistics.
 3. **Span:** The planning of transport routes which satisfy a number of transportation properties.
 4. **Concepts and facilities:** The concepts and facilities of the required computing system are initially thought to be: (1) With every segment and with every junction (i.e., item) of a net is associated a number of logistics attributes such as: (1.1) length, (1.2) recommended (expected average) transport time along a segment between junctions, (1.3) recommended (expected average) transport time through a junction, from a segment (of one modality) to another segment (of possibly another modality), (1.4) transportation costs along a complete segment, (1.5) transportation costs through a junction (from a segment (of one modality) to another segment (of possibly another modality)), etc. (2) Every item is represented in a database. (3) Database queries can help (3.1) analyse with respect to shortest, fastest, cheapest, etc. routes through a net, to come up with possibly alternative routes, etc.
 5. **Synopsis:**
 6. **Design brief:**

- **Net Animation:** Sect. 3.6, Pages 59–92

 1. **The current situation:** Traffic is becoming forever more congested, is unpredictable (i.e., future traffic conditions cannot be simply computed based on statistical trends) and is difficult to control.
 2. **Needs and ideas:** There is clearly some need for means to ascertain possible future traffic patterns based on single or multiple combinations of traffic assumptions. An idea could be that a computerised tool for (visual display) animating traffic based on such assumptions be a possible support for traffic controllers.

3. **Span:** For illustration we suggest the rather narrow span of animating only regular, unhindered and normal flow and schedule train traffic.
4. **Concepts and facilities:** The concepts and facilities of the required computing system are initially thought to be: (1) To every simple segment is associated a curved line, i.e., a curve, in 2-dimensional space, and (2) to every such curve is associated a velocity and acceleration profile — which designates the speed with which a train travels along the segment. (3) There is then associated with a given net a train timetable which lists a number of train travel prescriptions, with each travel prescription being a list of junction (i.e., station) visits where each station visit records the time of train arrival, the station name (i.e., junction identifier), and the time of train departure. Etcetera.
5. **Synopsis:**
6. **Design brief:**

In the informative texts above we have been rather "cavalier"[2] wrt. the terms maintenance, logistics and animation. Each of the respective requirements sections shall therefore start with a terminology subsection in which we present careful definitions of these and derived terms.

- Sections 3.7–3.8 cover remaining issues of interface and machine requirements.

Section 3.3 introduces Sects. 3.4–3.6.

3.2 A Summary Domain Model

3.2.1 Domain: Net Topology

Domain: Nets, Segments and Junctions

type
  N, S, J
value
  obs_Ss: N $\rightarrow$ S-**set**
  obs_Js: N $\rightarrow$ J-**set**
axiom
  $\forall$ n:N • **card** obs_Ss(n) $\geq$ 1 $\wedge$ **card** obs_Js(n) $\geq$ 2

[2]Merriam–Webster. cavalier [noun]: marked by or given to offhand and often disdainful dismissal of important matters

Domain: Segment and Junction Identifications

.

type
  Si, Ji
value
  obs_Si: $S \to Si$
  obs_Ji: $J \to Ji$

axiom
  $\forall$ n:N • **card** obs_Ss(n) $\equiv$ **card** {obs_Si(s)|s:S • s $\in$ obs_Ss(n)}
  $\forall$ n:N • **card** obs_Js(n) $\equiv$ **card** {obs_Ji(c)|j:J • j $\in$ obs_Js(n)}

Domain: Segment and Junction Reference Identifications

type
  Jip = {|{ji,ji′}:Ji-**set** • ji$\neq$ji′|}
value
  obs_Jis: $S \to$ Jip

type
  Si1 = {|sis:Si-**set**•**card** sis $\geq$1|}
value
  obs_Sis: $J \to$ Si1

axiom
  $\forall$ n:N, s:S • s $\in$ obs_Ss(n) $\Rightarrow$
    let {ji,ji′} = obs_Jis(s) **in**
    $\exists!$ j,j′:J • {j,j′}$\subseteq$obs_Js(n) $\wedge$ j$\neq$j′ $\wedge$
      obs_Si(s) $\in$ obs_Sis(c) $\cap$ obs_Sis(c′) **end**

axiom
  $\forall$ n:N, j:J • j $\in$ obs_Js(n) $\Rightarrow$
    let sis = obs_Sis(c), ji = obs_Ji(j) **in**
    $\exists!$ ss:S-**set** • ss$\subseteq$obs_Ss(n) $\wedge$ **card** ss=**card** sis $\wedge$
    sis = {|obs_Si(s)|s:S•s $\in$ ss|} **end**

Domain: Paths and Routes

type
  $P = Ji \times Si \times Ji$
value
  paths: N $\rightarrow$ P-**set**
  paths(n) $\equiv$
    $\{(ji,si,ji')|s{:}S,ji,ji'{:}Ji,si{:}Si\bullet\ s \in obs\_Ss(n) \wedge \{ji,ji'\} \in obs\_Jis(s) \wedge si=obs\_Si(s)\}$

type
  $R = \{|r{:}P^*\bullet wf\_R(r)|\}$
value
  wf_R: $P^* \rightarrow$ **Bool**
  wf_R(r) $\equiv$
    $\forall$ i:**Nat** $\bullet$ $\{i,i+1\} \subseteq$ **inds**(r) $\Rightarrow$
      let (_,_,ji)=r(i), (ji′,_,_)=r(i+1) **in** ji = ji′ **end**

  routes: N $\rightarrow$ R-**infset**
  routes(n) $\equiv$
    let rs = $\{\langle p\rangle|p{:}P\bullet p \in paths(n)\}$
         $\cup\ \{r^\frown r'|r,r'{:}R\bullet\{r,r'\}\subseteq rs \wedge wf\_R(r^\frown r')\}$ **in**
    rs **end**

Domain: Segment and Junction Identifications of Routes

value
  xtr_Jis: R $\rightarrow$ Ci-**set**, xtr_Sis: R $\rightarrow$ Si-**set**
  xtr_Jis(r) $\equiv$ **case** r **of** $\langle\rangle \rightarrow \{\}$, $\langle(ji,\_,ji')\rangle^\frown r' \rightarrow \{ji,ji'\} \cup$ xtr_Jis(r′) **end**
  xtr_Sis(r) $\equiv$ **case** r **of** $\langle\rangle \rightarrow \{\}$, $\langle(\_,si,\_)\rangle^\frown r' \rightarrow \{si\} \cup$ xtr_Sis(r′) **end**

  xtr_Ss: N $\times$ Ji $\rightarrow$ S-**set**
  xtr_Ss(n,ji) $\equiv$ $\{s|s{:}S\bullet s \in obs\_Ss(n) \wedge ji \in obs\_Jis(s)\}$

  xtr_C: N $\times$ Ji $\rightarrow$ C, xtr_S: N $\times$ Si $\rightarrow$ S
  xtr_C(n,ji) $\equiv$ **let** j:J $\bullet$ j $\in$ obs_Js(n) $\wedge$ ji=obs_Ji(j) **in** j **end**
  xtr_S(n,si) $\equiv$ **let** s:S $\bullet$ s $\in$ obs_Ss(n) $\wedge$ si=obs_Si(s) **in** s **end**

  first_Ji: R $\stackrel{\sim}{\rightarrow}$ Ji, last_Ji: R $\stackrel{\sim}{\rightarrow}$ Ji
  first_Ji(r) $\equiv$ **case** r **of** $\langle\rangle \rightarrow$ **chaos**, $\langle(ji,\_,\_)\rangle^\frown r' \rightarrow$ ji **end**
  last_Ji(r) $\equiv$ **case** r **of** $\langle\rangle \rightarrow$ **chaos**, $r'^\frown\langle(\_,\_,ji)\rangle \rightarrow$ ji **end**

  first_Si: R $\stackrel{\sim}{\rightarrow}$ Si, last_Si: R $\stackrel{\sim}{\rightarrow}$ Si

first_Si(r) $\equiv$ **case** r **of** $\langle\rangle \to$ **chaos**, $\langle$(_,si,_)$\rangle^r'$ $\to$ si **end**
last_Si(r) $\equiv$ **case** r **of** $\langle\rangle \to$ **chaos**, r$'$^$\langle$(_,si,_)$\rangle \to$ si **end**

first_J: R $\times$ N $\xrightarrow{\sim}$ J, last_J: R $\times$ N $\xrightarrow{\sim}$ J
first_J(r,n) $\equiv$ xtr_J(first_Ji(r),n)
last_J(r,n) $\equiv$ xtr_J(last_Ji(r),n)

first_S: R $\times$ N $\xrightarrow{\sim}$ S, last_S: R $\times$ N $\xrightarrow{\sim}$ S
first_S(r,n) $\equiv$ xtr_S(first_Si(r),n)
last_S(r,n) $\equiv$ xtr_S(last_Si(r),n)

Domain: Circular and Pendular Routes

non_circular_routes: N $\to$ R-**set**
non_circular_routes(n) $\equiv$
 let rs = $\{\langle$p$\rangle$|p:P•p $\in$ paths(n)$\}$
  $\cup$ $\{$r^r$'$|r,r$'$:R•$\{$r,r$'\}\subseteq$rs$\wedge$wf_R(r^r$'$)$\wedge$non_circular(r,r$'$)$\}$ **in**
 rs **end**
non_circular: R$\times$R $\to$ **Bool**
non_circular(r,r$'$) $\equiv$ **card** xtr_Jis(r) $\cap$ xtr_Jis(r$'$) =1

value
 reverse: P $\to$ P
 reverse(jif,si,jit) $\equiv$ (jit,si,jif)

 reverse: R $\to$ R
 reverse(r) $\equiv$
  case r **of**
   $\langle\rangle \to \langle\rangle$,
   $\langle$(jif,si,jit)$\rangle^r'$ $\to$ reverse(r$'$)^$\langle$(jit,si,jif)$\rangle$
  end

 reverse(r) $\equiv$ $\langle$reverse(r(i))|i **in** [n..1]$\rangle$

 pendular: R $\to$ R
 pendular(r) $\equiv$ r^reverse(r)

 is_pendular(r) $\equiv$ $\exists$ r$'$,r$''$:R • r$'^{r}''$ = r $\wedge$ r$''$=reverse(r$'$)

Domain: Connected Nets

value
is_connected: N → **Bool**
is_connected(n) ≡
∀ j,j′:J • {j,j′}⊆obs_Js(n) ∧ j≠j′ ⇒
let (ji,ji′) = (obs_Ji(j),obs_Ji(j′)) **in**
∃ r:R • r ∈ routes(n) ∧
first_Ji(r) = ji ∧ last_Ji(r) = ji′ **end**

Domain: Net Decomposition

value
decompose: N → N-**set**
decompose(n) **as** ns
obs_Ss(n) = ∪{obs_Ss(n′)|n′:N•n′ ∈ ns} ∧
obs_Js(n) = ∪{obs_Js(n′)|n′:N•n′ ∈ ns} ∧
{} = ∩{obs_Ss(n′)|n′:N•n′ ∈ ns} ∧
{} = ∩{obs_Js(n′)|n′:N•n′ ∈ ns} ∧
∀ n′:N•n′ ∈ ns ⇒ connected(n′) ∧ ...

Lemma:
∀ n:N •
let ns = decompose (n) **in**
∀ n′,n″:N • {n′,n″}⊆ns ∧ n′≠n″ ⇒
obs_Ss(n′) ∩ obs_Ss(n″) = {} ∧
obs_Js(n′) ∩ obs_Js(n″) = {} **end**

3.2.2 Domain: Multi-Modal Nets

Domain: General Issues

type
M

Domain: Segment and Junction Modes

value
obs_M: S → M
obs_Ms: J → M-**set**

axiom
$\forall$ n:N, j:J • j $\in$ obs_Js(n) $\Rightarrow$
let ss = xtr_Ss(n,obs_Ji(j)) **in**
obs_Ms(j) = {obs_M(s)|s:S • s $\in$ ss} **end**
$\forall$ n:N, s:S • s $\in$ obs_Ss(n) $\Rightarrow$
let {ji,ji$'$} = obs_Jis(s) **in**
let {j,j$'$} = {xtr_J(n,ji),xtr_J(n,ji$'$)} **in**
obs_M(s) $\in$ obs_Ms(j) $\cap$ obs_Ms(j$'$) **end end**

Domain: Single-Modal Nets and Net Projection

type
mmN = {|n:N • **card** xtr_Ms(n) > 1|}
smN = {|n:N • **card** xtr_Ms(n) = 1|}
value
xtr_Ms: N $\rightarrow$ M-**set**
xtr_Ms(n) $\equiv$ {obs_M(s) | s:S • s $\in$ obs_Ss(n)}

projs: N $\rightarrow$ smN-**set**
projs(n) $\equiv$ {proj(n,m) | m:M • m $\in$ xtr_Ms(n)}

proj: N $\times$ M $\rightarrow$ smN
proj(n,m) **as** n$'$
post
let ss = obs_Ss(n), ss$'$ = obs_Ss(n$'$),
js = obs_Js(n), js$'$ = obs_Js(n$'$) **in**
ss$'$ = {s | s:S • s $\in$ ss $\wedge$ m=obs_M(s)} $\wedge$
js$'$ = {j | j:J • j $\in$ js $\wedge$ m $\in$ obs_Ms(j)}
end

3.2.3 Domain: Segment and Junction Attributes

Domain: Segment and Junction Attribute Observations

type
L, TI
value
ms:M-**set**, **axiom** ms$\neq${}
obs_L: S $\rightarrow$ L
obs_L: Si $\times$ J $\times$ M $\times$ Si $\rightarrow$ L
obs_TI: S $\rightarrow$ TI
obs_TI: Si $\times$ J $\times$ Si $\rightarrow$ TI

obs_TI: J × M $\stackrel{\sim}{\to}$ TI, **pre** obs_TI(j,m): m ∈ obs_Ms(j)
obs_TI: J × M × M $\stackrel{\sim}{\to}$ TI, **pre** obs_TI(j,m,m′): {m,m′}⊆obs_Ms(j)
obs_arr_TI: J × M $\stackrel{\sim}{\to}$ TI, **pre** obs_arr_TI(j,m): m ∈ obs_Ms(j)
obs_dep_TI: J × M $\stackrel{\sim}{\to}$ TI, **pre** obs_dep_TI(j,m): m ∈ obs_Ms(j)
+: L × L → L
+: TI × TI → TI

Domain: Route Lengths

value
length: R → N $\stackrel{\sim}{\to}$ L
length(r)(n) ≡
case r **of**
⟨⟩ → 0,
⟨(jf,si,jt)⟩ → obs_L(xtr_S(si,n)),
⟨(ji1,sii,ji2),(jj1,sij,jj2)⟩^r′ →
let si=xtr_S(sii,n),sj=xtr_S(sij,n) **in**
obs_L(si) + obs_L(sii,xtr_J(ji2,n),sij) + length(⟨(jj1,sij,jj2)⟩^r′) **end**
end
pre: r ∈ routes(n) ∧ ji2=jj1

value
shortest_route: Ji × Ji → N $\stackrel{\sim}{\to}$ R
shortest_route(jf,jt)(n) ≡
let rs = routes(n) **in**
let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} **in**
let sr:R • sr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ length(r)(n)<length(sr)(n) **in**
sr **end end end**
pre: {jf,jt}⊆obs_Jis(n) ∧ jf≠jt

Domain: Route Traversal Times

type
Q
FCT = R → N → Q
value
less: Q × Q → **Bool**
lowest: Ji × Ji → N → FCT → R
lowest(jf,jt)(n)(fct) ≡
let rs = routes(n) **in**


```
        let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} in
        let lr:R • lr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ less(fct(r)(n),fct(qr)(n)) in
        lr end end end



    Q
    PRE = Q × Q → Bool
    FCT = R → N → Q
value
    best: Ji × Ji → N → FCT → PRE → R
    best(cf,ct)(n)(fct)(pre) ≡
        let rs = routes(n) in
        let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=cf ∧ last_Ji(r)=ct} in
        let br:R • lr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ pre(fct(r)(n),fct(qr)(n)) in
        br end end end
```

Domain: Transporation Costs

```
type
    K, F
value
    obs_K: (S|J) → K
    obs_F: (S|J) → F

    +: K × K → K

    cost: R → N → K
    cost(r)(n) ≡
        case r of
            ⟨⟩ → 0,
            ⟨(jf,si,jt)⟩ →
                obs_K(xtr_J(jf,n))+obs_K(xtr_S(si,n))+obs_K(xtr_J(jt,n))
            ⟨(jf,si,jt),(jf′,si′,jt′)⟩^r′ → assert: jt=jf′
                obs_K(xtr_J(jf,n))+obs_K(xtr_S(si,n))+...+cost(r′)
        end

    cheapest: Ji×Ji → N → ((K×K)→K) → ((K×K)→Bool) → R
    cheapest(jf,jt)(n) ≡
        best(jf,jt)(n)(λ(k1,k2):(K×K)•k1+k2)(λ(k1,k2):(K×K)•k1<k2)
```

3.2.4 Domain: Road Nets

type
M == road | ...
Jc, Js, Jse
Jic, Jis, Jise
J = Jc | Js | Jse
Cn
value
is_composite_J: J → **Bool**
is_simple_J: J → **Bool**
is_simple_embedded_J: J → **Bool**
obs_N: Jc → N
obs_Jic: Jc → Jic, obs_Jis: Js → Jis, obs_Jise: Jse → Jise
obs_Cn: Jic → Cn, obs_Cn: Jise → Cn
obs_Jise: Jic → Jise
axiom
∀ j:Jc • is_composite_J(j) ∧ xtr_Ms(obs_N(j,road))={road},
∀ j:Js • is_simple_J(j),
∀ j:Jse • is_simple_embedded_J(j)

∀ n:N,j:J • j ∈ obs_Js(n) ∧ is_composite_J(j) ⇒
  let rn = obs_N(j) **in**
    ...
  end

3.2.5 Domain: Railway Nets

Domain: General

value
proj: J × {railway} → ST

value
proj: S × {railway} → LI

Domain: Lines, Stations, Units and Connectors

type
RN = {| n:smN • obs_M(n)=railway |}
LI = {| s:S • obs_M(s)=railway |}

```
  ST  = {| c:C • obs_M(c)=railway |}
  Tr, U, K

value
  obs_LIs: RN → LI-set
  obs_STs: RN → ST-set
  obs_Us: RN → U-set
  obs_Us: LI → U-set
  obs_Us: ST → U-set
  obs_Trs: ST → Tr-set
  is_Linear: U → Bool
  is_Switch: U → Bool
  is_Simple_Crossover: U → Bool
  is_Switchable_Crossover: U → Bool
  obs_Ks: U → K-set

  lin_seq: U-set → Bool
  lin_seq(us) ≡
    ∀ u:U • u ∈ us ⇒ is_Linear(u) ∧
      ∃ q:U* • len q = card us ∧ elems q = us ∧
        ∀ i:Nat • {i,i+1} ⊆ inds q ⇒ ∃ k:K •
          obs_Ks(q(i)) ∩ obs_Ks(q(i+1)) = {k} ∧
        len q > 1 ⇒ obs_Ks(q(i)) ∩ obs_Ks(q(len q)) = {}

axiom

  ∀ n:RN • card obs_LIs(n) ≥ 1 ∧ card obs_STs(n) ≥ 2
  ∀ n:RN, l:LI • l ∈ obs_LIs(n) ⇒ lin_seq(l)
  ∀ n:RN, l:LI • l ∈ obs_LIs(n) ⇒ obs_Us(l) ⊆ obs_Us(n)
  ∀ n:RN, s:ST • s ∈ obs_STs(n) ⇒ card obs_Us(s) ≥ 1
  ∀ n:RN, s:ST • s ∈ obs_LIs(n) ⇒ obs_Us(s) ⊆ obs_Us(n)
  ∀ n:RN,l,l′:LI•{l,l′}⊆obs_LIs(n)∧l≠l′⇒obs_Us(l)∩ obs_Us(l′)={}
  ∀ n:RN,l:LI,s:ST•l ∈ obs_LIs(n)∧s ∈ obs_STs(n)⇒obs_Us(l)∩ obs_Us(s)={}
  ∀ n:RN,s,s′:ST•{s,s′}⊆obs_STs(n)∧s≠s′⇒obs_Us(s)∩ obs_Us(s′)={}
  ∀ s:ST•card obs_Trs(s)≥1
  ∀ n:RN,s:ST,t:Tr•s ∈ obs_STs(n)∧t ∈ obs_Trs(s)⇒lin_seq(t)
  ∀ n:RN,s:ST,t,t′:Tr•s ∈ obs_STs(n)∧{t,t′}⊆obs_Trs(s)∧t≠t′
    ⇒ obs_Us(t) ∩ obs_Us(t′) = {}
  n:RN • ∀ k:K •
    k ∈ ∪{obs_Ks(u)|u:U•u ∈ obs_Us(n)}
      ⇒card{u|u:U•u ∈ obs_Us(n)∧k ∈ obs_Ks(u)}≤2
  ∀ n:RN,l:LI •  l ∈ obs_LIs(n) ⇒
    ∃ s,s′:ST • {s,s′} ⊆ obs_STs(n) ∧ s≠s′ ⇒
```

let sus=obs_Us(s),sus′=obs_Us(s′),lus=obs_Us(l) **in**
∃ u,u′,u″,u‴:U • u ∈ sus ∧
  u′ ∈ sus′ ∧ {u″,u‴} ⊆ lus ⇒
  let sks = obs_Ks(u), sks′ = obs_Ks(u′),
    lks = obs_Ks(u″), lks′ = obs_Ks(u‴) **in**
  ∃!k,k′:K•k≠k′∧sks ∩ lks={k}∧sks′ ∩ lks′={k′}
end end

Domain: Paths, States and Routes

Domain: Train Movement and Traffic

Domain: Time Tables

Domain: Etcetera

3.2.6 Domain: Net Dynamics

Domain: Segment and Junction States

type
  SΣ = (Ji×Ji)-**set**
  JΣ = (Si×Si)-**set**
value
  obs_SΣ: S → SΣ
  obs_JΣ: J → JΣ

  xtr_Jis: SΣ → Ji-**set**, xtr_Jis(sσ) ≡ {ji|ji:Ji • (ji,_) ∈ obs_sσ ∨ (_,ji) ∈ obs_sσ}
  xtr_Sis: JΣ → Si-**set**, xtr_Sis(jσ) ≡ {si|si:Si • (si,_) ∈ obs_jσ ∨ (_,si) ∈ obs_jσ}
axiom
  ∀ s:S • xtr_Jis(obs_SΣ(s)) ⊆ xtr_Jip(s),
  ∀ j:J • xtr_Sis(obs_JΣ(j)) ⊆ xtr_Sis(j)

axiom
  ...

Domain: Segment and Junction State Spaces

type
  SΩ = SΣ-**set**
  JΩ = JΣ-**set**
value

obs_SΩ: S → SΩ
obs_JΩ: J → JΩ
axiom
∀ s:S • obs_SΣ(s) ⊆ obs_SΩ(s),
∀ j:J • obs_JΣ(j) ⊆ obs_JΩ(j)

Domain: Open and Closed Routes

value
is_closed_R: R → **Bool**
is_open_R: R → **Bool**

is_closed_R() ≡

is_open_R() ≡

3.3 Three Sets of Requirements

3.4 Net Maintenance

3.4.1 Introduction

Maintenance: Terminology

1. **:**
2. **:**
3. **:**
4. **Maintenance:** By maintenance we understand either of four kinds of actions:
   (a) **Adaptive Maintenance:**
   (b) **Corrective Maintenance:**
   (c) **Extensional Maintenance:**

(d) **Perfective Maintenance:**

(e) **Preventive Maintenance:**

5. **:**

6. **:**

7. **:**

8. **:**

9. **:**

Maintenance: Detailed Span

3.4.2 Maintenance: Topology

Maintenance: Segments and Junctions

type
  N, S, J
value
  obs_Ss: N $\rightarrow$ S-**set**
  obs_Js: N $\rightarrow$ J-**set**
axiom
  $\forall$ n:N • **card** obs_Ss(n) $\geq 1 \wedge$ **card** obs_Js(n) ≥ 2

Maintenance: Segment and Junction Identifications

.

type
  Si, Ji
value
  obs_Si: S $\rightarrow$ Si
  obs_Ji: J $\rightarrow$ Ji

axiom
  $\forall$ n:N • **card** obs_Ss(n) $\equiv$ **card** {obs_Si(s)|s:S • s $\in$ obs_Ss(n)}
  $\forall$ n:N • **card** obs_Js(n) $\equiv$ **card** {obs_Ji(c)|j:J • j $\in$ obs_Js(n)}

Maintenance: Segment and Junction Reference Identifications

type
  Jip = $\{|\{ji,ji'\}$:Ji-**set** • $ji \neq ji'|\}$
value
  obs_Jis: S → Jip

type
  Si1 = $\{|$sis:Si-**set**•**card** sis $\geq 1|\}$
value
  obs_Sis: J → Si1

axiom
  ∀ n:N, s:S • s ∈ obs_Ss(n) ⇒
    let $\{ji,ji'\}$ = obs_Jis(s) **in**
    ∃! j,j′:J • $\{j,j'\}$⊆obs_Js(n) ∧ $j \neq j'$ ∧
      obs_Si(s) ∈ obs_Sis(c) ∩ obs_Sis(c′) **end**

axiom
  ∀ n:N, j:J • j ∈ obs_Js(n) ⇒
    let sis = obs_Sis(c), ji = obs_Ji(j) **in**
    ∃! ss:S-**set** • ss⊆obs_Ss(n) ∧ **card** ss=**card** sis ∧
    sis = $\{|$obs_Si(s)$|$s:S•s ∈ ss$|\}$ **end**

Maintenance: Paths and Routes

type
  P = Ji × Si × Ji
value
  paths: N → P-**set**
  paths(n) ≡
    $\{$(ji,si,ji′)$|$s:S,ji,ji′:Ji,si:Si• s ∈ obs_Ss(n)∧$\{ji,ji'\}$ ∈ obs_Jis(s)∧si=obs_Si(s)$\}$

type
  R = $\{|$r:P*•wf_R(r)$|\}$
value
  wf_R: P* → **Bool**
  wf_R(r) ≡
    ∀ i:**Nat** • $\{i,i+1\}$⊆**inds**(r) ⇒

```
      let (_,_,ji)=r(i), (ji′,_,_)=r(i+1) in ji = ji′ end

  routes: N → R-infset
  routes(n) ≡
    let rs = {⟨p⟩|p:P•p ∈ paths(n)}
           ∪ {r⌢r′|r,r′:R•{r,r′}⊆rs∧wf_R(r⌢r′)} in
    rs end
```

Maintenance: Segment and Junction Identifications of Routes

```
value
  xtr_Jis: R → Ci-set, xtr_Sis: R → Si-set
  xtr_Jis(r) ≡ case r of ⟨⟩ → {}, ⟨(ji,_,ji′)⟩⌢r′ → {ji,ji′}∪ xtr_Jis(r′) end
  xtr_Sis(r) ≡ case r of ⟨⟩ → {}, ⟨(_,si,_)⟩⌢r′ → {si}∪ xtr_Sis(r′) end

  xtr_Ss: N × Ji → S-set
  xtr_Ss(n,ji) ≡ {s|s:S•s ∈ obs_Ss(n) ∧ ji ∈ obs_Jis(s)}

  xtr_C: N × Ji → C, xtr_S: N × Si → S
  xtr_C(n,ji) ≡ let j:J • j ∈ obs_Js(n) ∧ ji=obs_Ji(j) in j end
  xtr_S(n,si) ≡ let s:S • s ∈ obs_Ss(n) ∧ si=obs_Si(s) in s end

  first_Ji: R ~→ Ji, last_Ji: R ~→ Ji
  first_Ji(r) ≡ case r of ⟨⟩ → chaos, ⟨(ji,_,_)⟩⌢r′ → ji end
  last_Ji(r) ≡ case r of ⟨⟩ → chaos, r′⌢⟨(_,_,ji)⟩ → ji end

  first_Si: R ~→ Si, last_Si: R ~→ Si
  first_Si(r) ≡ case r of ⟨⟩ → chaos, ⟨(_,si,_)⟩⌢r′ → si end
  last_Si(r) ≡ case r of ⟨⟩ → chaos, r′⌢⟨(_,si,_)⟩ → si end

  first_J: R × N ~→ J, last_J: R × N ~→ J
  first_J(r,n) ≡ xtr_J(first_Ji(r),n)
  last_J(r,n) ≡ xtr_J(last_Ji(r),n)

  first_S: R × N ~→ S, last_S: R × N ~→ S
  first_S(r,n) ≡ xtr_S(first_Si(r),n)
  last_S(r,n) ≡ xtr_S(last_Si(r),n)
```

Maintenance: Circular and Pendular Routes

Maintenance: Connected Nets

Maintenance: Net Decomposition

3.4.3 Maintenance: Multi-Modal Nets

Maintenance: General Issues

type
  M

Maintenance: Segment and Junction Modes

value
  obs_M: S $\rightarrow$ M
  obs_Ms: J $\rightarrow$ M-**set**
axiom
  $\forall$ n:N, j:J $\bullet$ j $\in$ obs_Js(n) $\Rightarrow$
    let ss = xtr_Ss(n,obs_Ji(j)) **in**
    obs_Ms(j) = {obs_M(s)|s:S $\bullet$ s $\in$ ss} **end**
  $\forall$ n:N, s:S $\bullet$ s $\in$ obs_Ss(n) $\Rightarrow$
    let {ji,ji$'$} = obs_Jis(s) **in**
    let {j,j$'$} = {xtr_J(n,ji),xtr_J(n,ji$'$)} **in**
    obs_M(s) $\in$ obs_Ms(j) $\cap$ obs_Ms(j$'$) **end end**

Maintenance: Single-Modal Nets and Net Projection

type
  mmN = {|n:N $\bullet$ **card** xtr_Ms(n) > 1|}
  smN = {|n:N $\bullet$ **card** xtr_Ms(n) = 1|}
value
  xtr_Ms: N $\rightarrow$ M-**set**
  xtr_Ms(n) $\equiv$ {obs_M(s) | s:S $\bullet$ s $\in$ obs_Ss(n)}

  projs: N $\rightarrow$ smN-**set**
  projs(n) $\equiv$ {proj(n,m) | m:M $\bullet$ m $\in$ xtr_Ms(n)}

  proj: N $\times$ M $\rightarrow$ smN
  proj(n,m) **as** n$'$
    post
      let ss = obs_Ss(n), ss$'$ = obs_Ss(n$'$),

$\quad\quad \mathrm{js = obs\_Js(n),\ js' = obs\_Js(n')}$ **in**
$\quad \mathrm{ss' = \{s \mid s{:}S \bullet s \in ss \wedge m{=}obs\_M(s)\} \wedge}$
$\quad \mathrm{js' = \{j \mid j{:}J \bullet j \in js \wedge m \in obs\_Ms(j)\}}$
$\quad$ **end**

3.4.4 Maintenance: Segment and Junction Attributes

Maintenance: Segment and Junction Attribute Observations

type
$\quad \mathrm{L,\ TI}$
value
$\quad \mathrm{ms{:}M}$**-set**, **axiom** $\mathrm{ms \neq \{\}}$
$\quad \mathrm{obs\_L{:}\ S \to L}$
$\quad \mathrm{obs\_L{:}\ Si \times J \times M \times Si \to L}$
$\quad \mathrm{obs\_TI{:}\ S \to TI}$
$\quad \mathrm{obs\_TI{:}\ Si \times J \times Si \to TI}$
$\quad \mathrm{obs\_TI{:}\ J \times M \stackrel{\sim}{\to} TI,}$ **pre** $\mathrm{obs\_TI(j,m){:}\ m \in obs\_Ms(j)}$
$\quad \mathrm{obs\_TI{:}\ J \times M \times M \stackrel{\sim}{\to} TI,}$ **pre** $\mathrm{obs\_TI(j,m,m'){:}\ \{m,m'\} \subseteq obs\_Ms(j)}$
$\quad \mathrm{obs\_arr\_TI{:}\ J \times M \stackrel{\sim}{\to} TI,}$ **pre** $\mathrm{obs\_arr\_TI(j,m){:}\ m \in obs\_Ms(j)}$
$\quad \mathrm{obs\_dep\_TI{:}\ J \times M \stackrel{\sim}{\to} TI,}$ **pre** $\mathrm{obs\_dep\_TI(j,m){:}\ m \in obs\_Ms(j)}$
$\quad \mathrm{+{:}\ L \times L \to L}$
$\quad \mathrm{+{:}\ TI \times TI \to TI}$

Maintenance: Route Lengths

value
$\quad \mathrm{length{:}\ R \to N \stackrel{\sim}{\to} L}$
$\quad \mathrm{length(r)(n) \equiv}$
$\quad\quad$ **case** r **of**
$\quad\quad\quad \langle\rangle \to 0,$
$\quad\quad\quad \mathrm{\langle(jf,si,jt)\rangle \to obs\_L(xtr\_S(si,n)),}$
$\quad\quad\quad \mathrm{\langle(ji1,sii,ji2),(jj1,sij,jj2)\rangle^\frown r' \to}$
$\quad\quad\quad\quad$ **let** $\mathrm{si{=}xtr\_S(sii,n),sj{=}xtr\_S(sij,n)}$ **in**
$\quad\quad\quad\quad \mathrm{obs\_L(si) + obs\_L(sii,xtr\_J(ji2,n),sij) + length(\langle(jj1,sij,jj2)\rangle^\frown r')}$ **end**
$\quad\quad$ **end**
$\quad\quad$ **pre**: $\mathrm{r \in routes(n) \wedge ji2{=}jj1}$

value
$\quad \mathrm{shortest\_route{:}\ Ji \times Ji \to N \stackrel{\sim}{\to} R}$
$\quad \mathrm{shortest\_route(jf,jt)(n) \equiv}$

let rs = routes(n) **in**
let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} **in**
let sr:R • sr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ length(r)(n)<length(sr)(n) **in**
sr **end end end**
pre: {jf,jt}⊆obs_Jis(n) ∧ jf≠jt

Maintenance: Route Traversal Times

type
Q
FCT = R → N → Q
value
less: Q × Q → **Bool**
lowest: Ji × Ji → N → FCT → R
lowest(jf,jt)(n)(fct) ≡
let rs = routes(n) **in**
let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} **in**
let lr:R • lr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ less(fct(r)(n),fct(qr)(n)) **in**
lr **end end end**

Q
PRE = Q × Q → **Bool**
FCT = R → N → Q
value
best: Ji × Ji → N → FCT → PRE → R
best(cf,ct)(n)(fct)(pre) ≡
let rs = routes(n) **in**
let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=cf ∧ last_Ji(r)=ct} **in**
let br:R • lr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ pre(fct(r)(n),fct(qr)(n)) **in**
br **end end end**

Maintenance: Transporation Costs

type
K, F
value
obs_K: (S|J) → K
obs_F: (S|J) → F

+: K × K → K


```
cost: R → N → K
cost(r)(n) ≡
   case r of
      ⟨⟩ → 0,
      ⟨(jf,si,jt)⟩ →
         obs_K(xtr_J(jf,n))+obs_K(xtr_S(si,n))+obs_K(xtr_J(jt,n))
      ⟨(jf,si,jt),(jf′,si′,jt′)⟩^r′ → assert: jt=jf′
         obs_K(xtr_J(jf,n))+obs_K(xtr_S(si,n))+...+cost(r′)
   end

cheapest: Ji×Ji → N → ((K×K)→K) → ((K×K)→Bool) → R
cheapest(jf,jt)(n) ≡
   best(jf,jt)(n)(λ(k1,k2):(K×K)•k1+k2)(λ(k1,k2):(K×K)•k1<k2)
```

3.4.5 Maintenance: Road Nets

```
type
   M == road | ...
   Jc, Js, Jse
   Jic, Jis, Jise
   J = Jc | Js | Jse
   Cn
value
   is_composite_J: J → Bool
   is_simple_J: J → Bool
   is_simple_embedded_J: J → Bool
   obs_N: Jc → N
   obs_Jic: Jc → Jic, obs_Jis: Js → Jis, obs_Jise: Jse → Jise
   obs_Cn: Jic → Cn, obs_Cn: Jise → Cn
   obs_Jise: Jic → Jise
axiom
   ∀ j:Jc • is_composite_J(j) ∧ xtr_Ms(obs_N(j,road))={road},
   ∀ j:Js • is_simple_J(j),
   ∀ j:Jse • is_simple_embedded_J(j)

   ∀ n:N,j:J • j ∈ obs_Js(n) ∧ is_composite_J(j) ⇒
      let rn = obs_N(j) in
         ...
      end
```

3.4.6 Maintenance: Railway Nets

Maintenance: General

value
proj: J × {railway} → ST

value
proj: S × {railway} → LI

Maintenance: Lines, Stations, Units and Connectors

```
type
    RN = {| n:smN • obs_M(n)=railway |}
    LI = {| s:S • obs_M(s)=railway |}
    ST = {| c:C • obs_M(c)=railway |}
    Tr, U, K

value
    obs_LIs: RN → LI-set
    obs_STs: RN → ST-set
    obs_Us: RN → U-set
    obs_Us: LI → U-set
    obs_Us: ST → U-set
    obs_Trs: ST → Tr-set
    is_Linear: U → Bool
    is_Switch: U → Bool
    is_Simple_Crossover: U → Bool
    is_Switchable_Crossover: U → Bool
    obs_Ks: U → K-set

    lin_seq: U-set → Bool
    lin_seq(us) ≡
        ∀ u:U • u ∈ us ⇒ is_Linear(u) ∧
            ∃ q:U* • len q = card us ∧ elems q = us ∧
                ∀ i:Nat • {i,i+1} ⊆ inds q ⇒ ∃ k:K •
                    obs_Ks(q(i)) ∩ obs_Ks(q(i+1)) = {k} ∧
                len q > 1 ⇒ obs_Ks(q(i)) ∩ obs_Ks(q(len q)) = {}

axiom

    ∀ n:RN • card obs_LIs(n) ≥ 1 ∧ card obs_STs(n) ≥ 2
```

$\forall$ n:RN, l:LI • l $\in$ obs_LIs(n) $\Rightarrow$ lin_seq(l)
$\forall$ n:RN, l:LI • l $\in$ obs_LIs(n) $\Rightarrow$ obs_Us(l) $\subseteq$ obs_Us(n)
$\forall$ n:RN, s:ST • s $\in$ obs_STs(n) $\Rightarrow$ **card** obs_Us(s) $\geq$ 1
$\forall$ n:RN, s:ST • s $\in$ obs_LIs(n) $\Rightarrow$ obs_Us(s) $\subseteq$ obs_Us(n)
$\forall$ n:RN,l,l′:LI•{l,l′}$\subseteq$obs_LIs(n)$\wedge$l$\neq$l′$\Rightarrow$obs_Us(l)$\cap$ obs_Us(l′)={}
$\forall$ n:RN,l:LI,s:ST•l $\in$ obs_LIs(n)$\wedge$s $\in$ obs_STs(n)$\Rightarrow$obs_Us(l)$\cap$ obs_Us(s)={}
$\forall$ n:RN,s,s′:ST•{s,s′}$\subseteq$obs_STs(n)$\wedge$s$\neq$s′$\Rightarrow$obs_Us(s)$\cap$ obs_Us(s′)={}
$\forall$ s:ST•**card** obs_Trs(s)$\geq$1
$\forall$ n:RN,s:ST,t:Tr•s $\in$ obs_STs(n)$\wedge$t $\in$ obs_Trs(s)$\Rightarrow$lin_seq(t)
$\forall$ n:RN,s:ST,t,t′:Tr•s $\in$ obs_STs(n)$\wedge${t,t′}$\subseteq$obs_Trs(s)$\wedge$t$\neq$t′
  $\Rightarrow$ obs_Us(t) $\cap$ obs_Us(t′) = {}
n:RN • $\forall$ k:K •
  k $\in$ $\cup${obs_Ks(u)|u:U•u $\in$ obs_Us(n)}
    $\Rightarrow$**card**{u|u:U•u $\in$ obs_Us(n)$\wedge$k $\in$ obs_Ks(u)}$\leq$2
$\forall$ n:RN,l:LI • l $\in$ obs_LIs(n) $\Rightarrow$
  $\exists$ s,s′:ST • {s,s′} $\subseteq$ obs_STs(n) $\wedge$ s$\neq$s′ $\Rightarrow$
    let sus=obs_Us(s),sus′=obs_Us(s′),lus=obs_Us(l) **in**
    $\exists$ u,u′,u″,u‴:U • u $\in$ sus $\wedge$
      u′ $\in$ sus′ $\wedge$ {u″,u‴} $\subseteq$ lus $\Rightarrow$
      let sks = obs_Ks(u), sks′ = obs_Ks(u′),
        lks = obs_Ks(u″), lks′ = obs_Ks(u‴) **in**
      $\exists$!k,k′:K•k$\neq$k′$\wedge$sks $\cap$ lks={k}$\wedge$sks′ $\cap$ lks′={k′}
    end end

Maintenance: Paths, States and Routes

Maintenance: Train Movement and Traffic

Maintenance: Time Tables

Maintenance: Etcetera

3.4.7 Maintenance: Net Dynamics

Maintenance: Segment and Junction States

type
  SΣ = (Ji$\times$Ji)-**set**
  JΣ = (Si$\times$Si)-**set**
value
  obs_SΣ: S $\to$ SΣ
  obs_JΣ: J $\to$ JΣ

  xtr_Jis: SΣ $\to$ Ji-**set**, xtr_Jis(sσ) $\equiv$ {ji|ji:Ji • (ji,_) $\in$ obs_sσ $\vee$ (_,ji) $\in$ obs_sσ}

xtr_Sis: JΣ → Si-**set**, xtr_Sis(jσ) ≡ {si|si:Si • (si,_) ∈ obs_jσ ∨ (_,si) ∈ obs_jσ}
axiom
∀ s:S • xtr_Jis(obs_SΣ(s)) ⊆ xtr_Jip(s),
∀ j:J • xtr_Sis(obs_JΣ(j)) ⊆ xtr_Sis(j)

axiom
...

Maintenance: Segment and Junction State Spaces

type
SΩ = SΣ-**set**
JΩ = JΣ-**set**
value
obs_SΩ: S → SΩ
obs_JΩ: J → JΩ
axiom
∀ s:S • obs_SΣ(s) ⊆ obs_SΩ(s),
∀ j:J • obs_JΣ(j) ⊆ obs_JΩ(j)

Maintenance: Open and Closed Routes

value
is_closed_R: R → **Bool**
is_open_R: R → **Bool**

is_closed_R() ≡

is_open_R() ≡

3.5 Net Logistics

3.5.1 Logistics: Introduction

Logistics: Terminology

1. **Logistics:** By logistics we understand any non-empty combination of the following six kinds of actions:

   (a) **Planning:**

i. **Allocation:**
ii. **Scheduling:**

(b) **Execution:**

i. **Monitoring:**
ii. **Control:**

2. **:**
3. **:**
4. **:**
5. **:**
6. **:**

Logistics: Detailed Span

3.5.2 Logistics: Net Topology

Logistics: Nets, Segments and Junctions

type
  N, S, J
value
  obs_Ss: N $\rightarrow$ S-**set**
  obs_Js: N $\rightarrow$ J-**set**
axiom
  $\forall$ n:N • **card** obs_Ss(n) $\geq$ 1 $\wedge$ **card** obs_Js(n) $\geq$ 2

Logistics: Segment and Junction Identifications

.

type
  Si, Ji
value
  obs_Si: S $\rightarrow$ Si
  obs_Ji: J $\rightarrow$ Ji

axiom
  $\forall$ n:N • **card** obs_Ss(n) $\equiv$ **card** {obs_Si(s)|s:S • s $\in$ obs_Ss(n)}
  $\forall$ n:N • **card** obs_Js(n) $\equiv$ **card** {obs_Ji(c)|j:J • j $\in$ obs_Js(n)}

Logistics: Segment and Junction Reference Identifications

type
  Jip = $\{|\{ji,ji'\}$:Ji-**set** • ji$\neq$ji'$|\}$
value
  obs_Jis: S $\to$ Jip

type
  Si1 = $\{|$sis:Si-**set**•**card** sis $\geq$1$|\}$
value
  obs_Sis: J $\to$ Si1

axiom
  $\forall$ n:N, s:S • s $\in$ obs_Ss(n) $\Rightarrow$
    let $\{$ji,ji'$\}$ = obs_Jis(s) **in**
    $\exists!$ j,j':J • $\{$j,j'$\}\subseteq$obs_Js(n) $\wedge$ j$\neq$j' $\wedge$
      obs_Si(s) $\in$ obs_Sis(c) $\cap$ obs_Sis(c') **end**

axiom
  $\forall$ n:N, j:J • j $\in$ obs_Js(n) $\Rightarrow$
    let sis = obs_Sis(c), ji = obs_Ji(j) **in**
    $\exists!$ ss:S-**set** • ss$\subseteq$obs_Ss(n) $\wedge$ **card** ss=**card** sis $\wedge$
    sis = $\{|$obs_Si(s)$|$s:S•s $\in$ ss$|\}$ **end**

Logistics: Paths and Routes

type
  P = Ji $\times$ Si $\times$ Ji
value
  paths: N $\to$ P-**set**
  paths(n) $\equiv$
    $\{$(ji,si,ji')$|$s:S,ji,ji':Ji,si:Si• s $\in$ obs_Ss(n)$\wedge\{$ji,ji'$\}$ $\in$ obs_Jis(s)$\wedge$si=obs_Si(s)$\}$

type
  R = $\{|$r:P*•wf_R(r)$|\}$
value
  wf_R: P* $\to$ **Bool**
  wf_R(r) $\equiv$
    $\forall$ i:**Nat** • $\{$i,i+1$\}\subseteq$**inds**(r) $\Rightarrow$

let (_,_,ji)=r(i), (ji′,_,_)=r(i+1) **in** ji = ji′ **end**

routes: N → R-**infset**
routes(n) ≡
  let rs = {⟨p⟩|p:P•p ∈ paths(n)}
      ∪ {r^r′|r,r′:R•{r,r′}⊆rs∧wf_R(r^r′)} **in**
  rs **end**

Logistics: Segment and Junction Identifications of Routes

value
xtr_Jis: R → Ci-**set**, xtr_Sis: R → Si-**set**
xtr_Jis(r) ≡ **case** r **of** ⟨⟩ → {}, ⟨(ji,_,ji′)⟩^r′ → {ji,ji′}∪ xtr_Jis(r′) **end**
xtr_Sis(r) ≡ **case** r **of** ⟨⟩ → {}, ⟨(_,si,_)⟩^r′ → {si}∪ xtr_Sis(r′) **end**

xtr_Ss: N × Ji → S-**set**
xtr_Ss(n,ji) ≡ {s|s:S•s ∈ obs_Ss(n) ∧ ji ∈ obs_Jis(s)}

xtr_C: N × Ji → C, xtr_S: N × Si → S
xtr_C(n,ji) ≡ **let** j:J • j ∈ obs_Js(n) ∧ ji=obs_Ji(j) **in** j **end**
xtr_S(n,si) ≡ **let** s:S • s ∈ obs_Ss(n) ∧ si=obs_Si(s) **in** s **end**

first_Ji: R $\xrightarrow{\sim}$ Ji, last_Ji: R $\xrightarrow{\sim}$ Ji
first_Ji(r) ≡ **case** r **of** ⟨⟩ → **chaos**, ⟨(ji,_,_)⟩^r′ → ji **end**
last_Ji(r) ≡ **case** r **of** ⟨⟩ → **chaos**, r′^⟨(_,_,ji)⟩ → ji **end**

first_Si: R $\xrightarrow{\sim}$ Si, last_Si: R $\xrightarrow{\sim}$ Si
first_Si(r) ≡ **case** r **of** ⟨⟩ → **chaos**, ⟨(_,si,_)⟩^r′ → si **end**
last_Si(r) ≡ **case** r **of** ⟨⟩ → **chaos**, r′^⟨(_,si,_)⟩ → si **end**

first_J: R × N $\xrightarrow{\sim}$ J, last_J: R × N $\xrightarrow{\sim}$ J
first_J(r,n) ≡ xtr_J(first_Ji(r),n)
last_J(r,n) ≡ xtr_J(last_Ji(r),n)

first_S: R × N $\xrightarrow{\sim}$ S, last_S: R × N $\xrightarrow{\sim}$ S
first_S(r,n) ≡ xtr_S(first_Si(r),n)
last_S(r,n) ≡ xtr_S(last_Si(r),n)

Logistics: Circular and Pendular Routes

non_circular_routes: N → R-**set**

```
    non_circular_routes(n) ≡
        let rs = {⟨p⟩|p:P•p ∈ paths(n)}
                 ∪ {r⌢r′|r,r′:R•{r,r′}⊆rs∧wf_R(r⌢r′)∧non_circular(r,r′)} in
        rs end
    non_circular: R×R → Bool
    non_circular(r,r′) ≡ card xtr_Jis(r) ∩ xtr_Jis(r′) =1
```

value

```
    reverse: P → P
    reverse(jif,si,jit) ≡ (jit,si,jif)

    reverse: R → R
    reverse(r) ≡
        case r of
            ⟨⟩ → ⟨⟩,
            ⟨(jif,si,jit)⟩⌢r′ → reverse(r′)⌢⟨(jit,si,jif)⟩
        end

    reverse(r) ≡ ⟨reverse(r(i))|i in [ n..1 ]⟩

    pendular: R → R
    pendular(r) ≡ r⌢reverse(r)

    is_pendular(r) ≡ ∃ r′,r″:R • r′⌢r″ = r ∧ r″=reverse(r′)
```

Logistics: Connected Nets

value

```
    is_connected: N → Bool
    is_connected(n) ≡
        ∀ j,j′:J • {j,j′}⊆obs_Js(n) ∧ j≠j′ ⇒
            let (ji,ji′) = (obs_Ji(j),obs_Ji(j′)) in
            ∃ r:R • r ∈ routes(n) ∧
                first_Ji(r) = ji ∧ last_Ji(r) = ji′ end
```

Logistics: Net Decomposition

value

```
    decompose: N → N-set
    decompose(n) as ns
```

obs_Ss(n) = ∪{obs_Ss(n′)|n′:N•n′ ∈ ns} ∧
obs_Js(n) = ∪{obs_Js(n′)|n′:N•n′ ∈ ns} ∧
{} = ∩{obs_Ss(n′)|n′:N•n′ ∈ ns} ∧
{} = ∩{obs_Js(n′)|n′:N•n′ ∈ ns} ∧
∀ n′:N•n′ ∈ ns ⇒ connected(n′) ∧ ...

Lemma:
∀ n:N •
let ns = decompose (n) **in**
∀ n′,n″:N • {n′,n″}⊆ns ∧ n′≠n″ ⇒
obs_Ss(n′) ∩ obs_Ss(n″) = {} ∧
obs_Js(n′) ∩ obs_Js(n″) = {} **end**

3.5.3 Logistics: Multi-Modal Nets

Logistics: General Issues

type
M

Logistics: Segment and Junction Modes

value
obs_M: S → M
obs_Ms: J → M-**set**
axiom
∀ n:N, j:J • j ∈ obs_Js(n) ⇒
let ss = xtr_Ss(n,obs_Ji(j)) **in**
obs_Ms(j) = {obs_M(s)|s:S • s ∈ ss} **end**
∀ n:N, s:S • s ∈ obs_Ss(n) ⇒
let {ji,ji′} = obs_Jis(s) **in**
let {j,j′} = {xtr_J(n,ji),xtr_J(n,ji′)} **in**
obs_M(s) ∈ obs_Ms(j) ∩ obs_Ms(j′) **end end**

Logistics: Single-Modal Nets and Net Projection

type
mmN = {|n:N • **card** xtr_Ms(n) > 1|}
smN = {|n:N • **card** xtr_Ms(n) = 1|}
value

$\text{xtr\_Ms}: N \rightarrow M\textbf{-set}$
$\text{xtr\_Ms}(n) \equiv \{\text{obs\_M}(s) \mid s{:}S \bullet s \in \text{obs\_Ss}(n)\}$

$\text{projs}: N \rightarrow \text{smN}\textbf{-set}$
$\text{projs}(n) \equiv \{\text{proj}(n,m) \mid m{:}M \bullet m \in \text{xtr\_Ms}(n)\}$

$\text{proj}: N \times M \rightarrow \text{smN}$
$\text{proj}(n,m)\ \textbf{as}\ n'$
  post
    $\textbf{let}\ ss = \text{obs\_Ss}(n),\ ss' = \text{obs\_Ss}(n'),$
      $js = \text{obs\_Js}(n),\ js' = \text{obs\_Js}(n')\ \textbf{in}$
    $ss' = \{s \mid s{:}S \bullet s \in ss \wedge m{=}\text{obs\_M}(s)\} \wedge$
    $js' = \{j \mid j{:}J \bullet j \in js \wedge m \in \text{obs\_Ms}(j)\}$
    end

3.5.4 Logistics: Segment and Junction Attributes

Logistics: Segment and Junction Attribute Observations

type
  L, TI
value
  $ms{:}M\textbf{-set},\qquad \textbf{axiom}\ ms \neq \{\}$
  $\text{obs\_L}: S \rightarrow L$
  $\text{obs\_L}: Si \times J \times M \times Si \rightarrow L$
  $\text{obs\_TI}: S \rightarrow TI$
  $\text{obs\_TI}: Si \times J \times Si \rightarrow TI$
  $\text{obs\_TI}: J \times M \xrightarrow{\sim} TI,\qquad \textbf{pre}\ \text{obs\_TI}(j,m){:}\ m \in \text{obs\_Ms}(j)$
  $\text{obs\_TI}: J \times M \times M \xrightarrow{\sim} TI,\qquad \textbf{pre}\ \text{obs\_TI}(j,m,m'){:}\ \{m,m'\} \subseteq \text{obs\_Ms}(j)$
  $\text{obs\_arr\_TI}: J \times M \xrightarrow{\sim} TI,\qquad \textbf{pre}\ \text{obs\_arr\_TI}(j,m){:}\ m \in \text{obs\_Ms}(j)$
  $\text{obs\_dep\_TI}: J \times M \xrightarrow{\sim} TI,\qquad \textbf{pre}\ \text{obs\_dep\_TI}(j,m){:}\ m \in \text{obs\_Ms}(j)$
  $+: L \times L \rightarrow L$
  $+: TI \times TI \rightarrow TI$

Logistics: Route Lengths

value
  $\text{length}: R \rightarrow N \xrightarrow{\sim} L$
  $\text{length}(r)(n) \equiv$
    case r **of**
      $\langle\rangle \rightarrow 0,$

```
        ⟨(jf,si,jt)⟩ → obs_L(xtr_S(si,n)),
        ⟨(ji1,sii,ji2),(jj1,sij,jj2)⟩^r′ →
            let si=xtr_S(sii,n),sj=xtr_S(sij,n) in
            obs_L(si) + obs_L(sii,xtr_J(ji2,n),sij) + length(⟨(jj1,sij,jj2)⟩^r′) end
    end
    pre: r ∈ routes(n) ∧ ji2=jj1
```

```
value
  shortest_route: Ji × Ji → N ~→ R
  shortest_route(jf,jt)(n) ≡
    let rs = routes(n) in
    let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} in
    let sr:R • sr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ length(r)(n)<length(sr)(n) in
    sr end end end
    pre: {jf,jt}⊆obs_Jis(n) ∧ jf≠jt
```

Logistics: Route Traversal Times

```
type
  Q
  FCT = R → N → Q
value
  less: Q × Q → Bool
  lowest: Ji × Ji → N → FCT → R
  lowest(jf,jt)(n)(fct) ≡
    let rs = routes(n) in
    let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} in
    let lr:R • lr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ less(fct(r)(n),fct(qr)(n)) in
    lr end end end
```

```
  Q
  PRE = Q × Q → Bool
  FCT = R → N → Q
value
  best: Ji × Ji → N → FCT → PRE → R
  best(cf,ct)(n)(fct)(pre) ≡
    let rs = routes(n) in
    let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=cf ∧ last_Ji(r)=ct} in
    let br:R • lr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ pre(fct(r)(n),fct(qr)(n)) in
    br end end end
```

Logistics: Transporation Costs

type
  K, F
value
  obs_K: (S|J) → K
  obs_F: (S|J) → F

  +: K × K → K

  cost: R → N → K
  cost(r)(n) ≡
    case r **of**
      ⟨⟩ → 0,
      ⟨(jf,si,jt)⟩ →
        obs_K(xtr_J(jf,n))+obs_K(xtr_S(si,n))+obs_K(xtr_J(jt,n))
      ⟨(jf,si,jt),(jf′,si′,jt′)⟩^r′ → **assert:** jt=jf′
        obs_K(xtr_J(jf,n))+obs_K(xtr_S(si,n))+...+cost(r′)
    end

  cheapest: Ji×Ji → N → ((K×K)→K) → ((K×K)→**Bool**) → R
  cheapest(jf,jt)(n) ≡
    best(jf,jt)(n)(λ(k1,k2):(K×K)•k1+k2)(λ(k1,k2):(K×K)•k1<k2)

3.5.5 Logistics: Road Nets

type
  M == road | ...
  Jc, Js, Jse
  Jic, Jis, Jise
  J = Jc | Js | Jse
  Cn
value
  is_composite_J: J → **Bool**
  is_simple_J: J → **Bool**
  is_simple_embedded_J: J → **Bool**
  obs_N: Jc → N
  obs_Jic: Jc → Jic, obs_Jis: Js → Jis, obs_Jise: Jse → Jise
  obs_Cn: Jic → Cn, obs_Cn: Jise → Cn
  obs_Jise: Jic → Jise
axiom
  ∀ j:Jc • is_composite_J(j) ∧ xtr_Ms(obs_N(j,road))={road},

$\forall$ j:Js • is_simple_J(j),
$\forall$ j:Jse • is_simple_embedded_J(j)

$\forall$ n:N,j:J • j $\in$ obs_Js(n) $\wedge$ is_composite_J(j) $\Rightarrow$
  let rn = obs_N(j) **in**
    ...
  end

3.5.6 Logistics: Railway Nets

Logistics: General

value
  proj: J $\times$ {railway} $\rightarrow$ ST

value
  proj: S $\times$ {railway} $\rightarrow$ LI

Logistics: Lines, Stations, Units and Connectors

type
  RN = {| n:smN • obs_M(n)=railway |}
  LI = {| s:S • obs_M(s)=railway |}
  ST = {| c:C • obs_M(c)=railway |}
  Tr, U, K

value
  obs_LIs: RN $\rightarrow$ LI-**set**
  obs_STs: RN $\rightarrow$ ST-**set**
  obs_Us: RN $\rightarrow$ U-**set**
  obs_Us: LI $\rightarrow$ U-**set**
  obs_Us: ST $\rightarrow$ U-**set**
  obs_Trs: ST $\rightarrow$ Tr-**set**
  is_Linear: U $\rightarrow$ **Bool**
  is_Switch: U $\rightarrow$ **Bool**
  is_Simple_Crossover: U $\rightarrow$ **Bool**
  is_Switchable_Crossover: U $\rightarrow$ **Bool**
  obs_Ks: U $\rightarrow$ K-**set**

  lin_seq: U-**set** $\rightarrow$ **Bool**
  lin_seq(us) $\equiv$

$$
\begin{array}{l}
\quad \forall\ u{:}U \bullet u \in us \Rightarrow is\_Linear(u) \wedge \\
\qquad \exists\ q{:}U^* \bullet \mathbf{len}\ q = \mathbf{card}\ us \wedge \mathbf{elems}\ q = us \wedge \\
\qquad\quad \forall\ i{:}\mathbf{Nat} \bullet \{i,i+1\} \subseteq \mathbf{inds}\ q \Rightarrow \exists\ k{:}K \bullet \\
\qquad\qquad obs\_Ks(q(i)) \cap obs\_Ks(q(i+1)) = \{k\} \wedge \\
\qquad\quad \mathbf{len}\ q > 1 \Rightarrow obs\_Ks(q(i)) \cap obs\_Ks(q(\mathbf{len}\ q)) = \{\}
\end{array}
$$

axiom

$$
\begin{array}{l}
\forall\ n{:}RN \bullet \mathbf{card}\ obs\_LIs(n) \geq 1 \wedge \mathbf{card}\ obs\_STs(n) \geq 2 \\
\forall\ n{:}RN,\ l{:}LI \bullet l \in obs\_LIs(n) \Rightarrow lin\_seq(l) \\
\forall\ n{:}RN,\ l{:}LI \bullet l \in obs\_LIs(n) \Rightarrow obs\_Us(l) \subseteq obs\_Us(n) \\
\forall\ n{:}RN,\ s{:}ST \bullet s \in obs\_STs(n) \Rightarrow \mathbf{card}\ obs\_Us(s) \geq 1 \\
\forall\ n{:}RN,\ s{:}ST \bullet s \in obs\_LIs(n) \Rightarrow obs\_Us(s) \subseteq obs\_Us(n) \\
\forall\ n{:}RN,l,l'{:}LI \bullet \{l,l'\} \subseteq obs\_LIs(n) \wedge l \neq l' \Rightarrow obs\_Us(l) \cap obs\_Us(l') = \{\} \\
\forall\ n{:}RN,l{:}LI,s{:}ST \bullet l \in obs\_LIs(n) \wedge s \in obs\_STs(n) \Rightarrow obs\_Us(l) \cap obs\_Us(s) = \{\} \\
\forall\ n{:}RN,s,s'{:}ST \bullet \{s,s'\} \subseteq obs\_STs(n) \wedge s \neq s' \Rightarrow obs\_Us(s) \cap obs\_Us(s') = \{\} \\
\forall\ s{:}ST \bullet \mathbf{card}\ obs\_Trs(s) \geq 1 \\
\forall\ n{:}RN,s{:}ST,t{:}Tr \bullet s \in obs\_STs(n) \wedge t \in obs\_Trs(s) \Rightarrow lin\_seq(t) \\
\forall\ n{:}RN,s{:}ST,t,t'{:}Tr \bullet s \in obs\_STs(n) \wedge \{t,t'\} \subseteq obs\_Trs(s) \wedge t \neq t' \\
\quad \Rightarrow obs\_Us(t) \cap obs\_Us(t') = \{\} \\
n{:}RN \bullet \forall\ k{:}K \bullet \\
\quad k \in \cup\{obs\_Ks(u) | u{:}U \bullet u \in obs\_Us(n)\} \\
\qquad \Rightarrow \mathbf{card}\{u | u{:}U \bullet u \in obs\_Us(n) \wedge k \in obs\_Ks(u)\} \leq 2 \\
\forall\ n{:}RN,l{:}LI \bullet\ l \in obs\_LIs(n) \Rightarrow \\
\quad \exists\ s,s'{:}ST \bullet \{s,s'\} \subseteq obs\_STs(n) \wedge s \neq s' \Rightarrow \\
\qquad \mathbf{let}\ sus=obs\_Us(s),sus'=obs\_Us(s'),lus=obs\_Us(l)\ \mathbf{in} \\
\qquad \exists\ u,u',u'',u'''{:}U \bullet u \in sus \wedge \\
\qquad\quad u' \in sus' \wedge \{u'',u'''\} \subseteq lus \Rightarrow \\
\qquad\quad \mathbf{let}\ sks = obs\_Ks(u),\ sks' = obs\_Ks(u'), \\
\qquad\qquad lks = obs\_Ks(u''),\ lks' = obs\_Ks(u''')\ \mathbf{in} \\
\qquad\quad \exists!k,k'{:}K \bullet k \neq k' \wedge sks \cap lks = \{k\} \wedge sks' \cap lks' = \{k'\} \\
\qquad \mathbf{end}\ \mathbf{end}
\end{array}
$$

Logistics: Paths, States and Routes

Logistics: Train Movement and Traffic

Logistics: Time Tables

Logistics: Etcetera

3.5.7 Logistics: Net Dynamics

Logistics: Segment and Junction States

type
  $\mathrm{S}\Sigma = (\mathrm{Ji}\times\mathrm{Ji})$**-set**
  $\mathrm{J}\Sigma = (\mathrm{Si}\times\mathrm{Si})$**-set**
value
  obs_SΣ: $\mathrm{S} \rightarrow \mathrm{S}\Sigma$
  obs_JΣ: $\mathrm{J} \rightarrow \mathrm{J}\Sigma$

  xtr_Jis: $\mathrm{S}\Sigma \rightarrow$ Ji-**set**, xtr_Jis(sσ) $\equiv$ {ji|ji:Ji • (ji,_) $\in$ obs_sσ $\vee$ (_,ji) $\in$ obs_sσ}
  xtr_Sis: $\mathrm{J}\Sigma \rightarrow$ Si-**set**, xtr_Sis(jσ) $\equiv$ {si|si:Si • (si,_) $\in$ obs_jσ $\vee$ (_,si) $\in$ obs_jσ}
axiom
  $\forall$ s:S • xtr_Jis(obs_SΣ(s)) $\subseteq$ xtr_Jip(s),
  $\forall$ j:J • xtr_Sis(obs_JΣ(j)) $\subseteq$ xtr_Sis(j)

axiom
  ...

Logistics: Segment and Junction State Spaces

type
  $\mathrm{S}\Omega = \mathrm{S}\Sigma$**-set**
  $\mathrm{J}\Omega = \mathrm{J}\Sigma$**-set**
value
  obs_SΩ: $\mathrm{S} \rightarrow \mathrm{S}\Omega$
  obs_JΩ: $\mathrm{J} \rightarrow \mathrm{J}\Omega$
axiom
  $\forall$ s:S • obs_SΣ(s) $\subseteq$ obs_SΩ(s),
  $\forall$ j:J • obs_JΣ(j) $\subseteq$ obs_JΩ(j)

Logistics: Open and Closed Routes

value
is_closed_R: R → **Bool**
is_open_R: R → **Bool**

is_closed_R() ≡

is_open_R() ≡

3.6 Net Animation

3.6.1 Animation: Introduction

Animation: Terminology

1. **Animation:** By animation we understand a combination of
   (a) **Visualisation** and
   (b) **Dyamics**.
2. **Visualisation** is the display of some technical and/or artistic image of a phenomenon or concept
   (a) **Image:**
      i. **Technical:**
      ii. **Artistics:**
   (b) **Rendering:** Many different kinds of images are possible:
      iii. **Photos:**
      iv. **Scaled Diagrams:**
      v. **Conceptual Diagrams:** Maps, ...
      vi. **Tabulated Functions:** Histograms, curves, ...
      vii. **:**
      viii. **:**
3. **Dynamics:**
   (a) **Time scales:**
   (b) **Discreteness:**
   (c) **Continuity:**

Animation: Detailed Span

3.6.2 Animation: Net Topology

Animation: Nets, Segments and Junctions

type
  N, S, J
value
  obs_Ss: N → S-**set**
  obs_Js: N → J-**set**
axiom
  $\forall$ n:N • **card** obs_Ss(n) $\geq$ 1 $\wedge$ **card** obs_Js(n) $\geq$ 2

Animation: Segment and Junction Identifications

.

type
  Si, Ji
value
  obs_Si: S → Si
  obs_Ji: J → Ji

axiom
  $\forall$ n:N • **card** obs_Ss(n) $\equiv$ **card** {obs_Si(s)|s:S • s $\in$ obs_Ss(n)}
  $\forall$ n:N • **card** obs_Js(n) $\equiv$ **card** {obs_Ji(c)|j:J • j $\in$ obs_Js(n)}

Animation: Segment and Junction Reference Identifications

type
  Jip = {|{ji,ji′}:Ji-**set** • ji≠ji′|}
value
  obs_Jis: S → Jip

type
  Si1 = {|sis:Si-**set**•**card** sis ≥1|}
value
  obs_Sis: J → Si1

axiom
$\forall$ n:N, s:S $\bullet$ s $\in$ obs_Ss(n) $\Rightarrow$
let {ji,ji′} = obs_Jis(s) **in**
$\exists!$ j,j′:J $\bullet$ {j,j′}$\subseteq$obs_Js(n) $\wedge$ j$\neq$j′ $\wedge$
obs_Si(s) $\in$ obs_Sis(c) $\cap$ obs_Sis(c′) **end**

axiom
$\forall$ n:N, j:J $\bullet$ j $\in$ obs_Js(n) $\Rightarrow$
let sis = obs_Sis(c), ji = obs_Ji(j) **in**
$\exists!$ ss:S-**set** $\bullet$ ss$\subseteq$obs_Ss(n) $\wedge$ **card** ss=**card** sis $\wedge$
sis = {|obs_Si(s)|s:S$\bullet$s $\in$ ss|} **end**

Animation: Paths and Routes

type
P = Ji $\times$ Si $\times$ Ji
value
paths: N $\to$ P-**set**
paths(n) $\equiv$
{(ji,si,ji′)|s:S,ji,ji′:Ji,si:Si$\bullet$ s $\in$ obs_Ss(n)$\wedge${ji,ji′} $\in$ obs_Jis(s)$\wedge$si=obs_Si(s)}

type
R = {|r:P*$\bullet$wf_R(r)|}
value
wf_R: P* $\to$ **Bool**
wf_R(r) $\equiv$
$\forall$ i:**Nat** $\bullet$ {i,i+1}$\subseteq$**inds**(r) $\Rightarrow$
let (_,_,ji)=r(i), (ji′,_,_)=r(i+1) **in** ji = ji′ **end**

routes: N $\to$ R-**infset**
routes(n) $\equiv$
let rs = {⟨p⟩|p:P$\bullet$p $\in$ paths(n)}
$\cup$ {r^r′|r,r′:R$\bullet${r,r′}$\subseteq$rs$\wedge$wf_R(r^r′)} **in**
rs **end**

Animation: Segment and Junction Identifications of Routes

value
xtr_Jis: R $\to$ Ci-**set**, xtr_Sis: R $\to$ Si-**set**
xtr_Jis(r) $\equiv$ **case** r **of** ⟨⟩ $\to$ {}, ⟨(ji,_,ji′)⟩^r′ $\to$ {ji,ji′}$\cup$ xtr_Jis(r′) **end**

xtr_Sis(r) ≡ **case** r **of** ⟨⟩ → {}, ⟨(_,si,_)⟩^r′ → {si}∪ xtr_Sis(r′) **end**

xtr_Ss: N × Ji → S-**set**
xtr_Ss(n,ji) ≡ {s|s:S•s ∈ obs_Ss(n) ∧ ji ∈ obs_Jis(s)}

xtr_C: N × Ji → C, xtr_S: N × Si → S
xtr_C(n,ji) ≡ **let** j:J • j ∈ obs_Js(n) ∧ ji=obs_Ji(j) **in** j **end**
xtr_S(n,si) ≡ **let** s:S • s ∈ obs_Ss(n) ∧ si=obs_Si(s) **in** s **end**

first_Ji: R $\xrightarrow{\sim}$ Ji, last_Ji: R $\xrightarrow{\sim}$ Ji
first_Ji(r) ≡ **case** r **of** ⟨⟩ → **chaos**, ⟨(ji,_,_)⟩^r′ → ji **end**
last_Ji(r) ≡ **case** r **of** ⟨⟩ → **chaos**, r′^⟨(_,_,ji)⟩ → ji **end**

first_Si: R $\xrightarrow{\sim}$ Si, last_Si: R $\xrightarrow{\sim}$ Si
first_Si(r) ≡ **case** r **of** ⟨⟩ → **chaos**, ⟨(_,si,_)⟩^r′ → si **end**
last_Si(r) ≡ **case** r **of** ⟨⟩ → **chaos**, r′^⟨(_,si,_)⟩ → si **end**

first_J: R × N $\xrightarrow{\sim}$ J, last_J: R × N $\xrightarrow{\sim}$ J
first_J(r,n) ≡ xtr_J(first_Ji(r),n)
last_J(r,n) ≡ xtr_J(last_Ji(r),n)

first_S: R × N $\xrightarrow{\sim}$ S, last_S: R × N $\xrightarrow{\sim}$ S
first_S(r,n) ≡ xtr_S(first_Si(r),n)
last_S(r,n) ≡ xtr_S(last_Si(r),n)

Animation: Circular and Pendular Routes

non_circular_routes: N → R-**set**
non_circular_routes(n) ≡
  let rs = {⟨p⟩|p:P•p ∈ paths(n)}
      ∪ {r^r′|r,r′:R•{r,r′}⊆rs∧wf_R(r^r′)∧non_circular(r,r′)} **in**
  rs **end**
non_circular: R×R → **Bool**
non_circular(r,r′) ≡ **card** xtr_Jis(r) ∩ xtr_Jis(r′) =1

value
reverse: P → P
reverse(jif,si,jit) ≡ (jit,si,jif)

reverse: R → R
reverse(r) ≡

```
    case r of
        ⟨⟩ → ⟨⟩,
        ⟨(jif,si,jit)⟩^r′ → reverse(r′)^⟨(jit,si,jif)⟩
    end

reverse(r) ≡ ⟨reverse(r(i))|i in [ n..1 ]⟩

pendular: R → R
pendular(r) ≡ r^reverse(r)

is_pendular(r) ≡ ∃ r′,r″:R • r′^r″ = r ∧ r″=reverse(r′)
```

Animation: Connected Nets

```
value
    is_connected: N → Bool
    is_connected(n) ≡
        ∀ j,j′:J • {j,j′}⊆obs_Js(n) ∧ j≠j′ ⇒
            let (ji,ji′) = (obs_Ji(j),obs_Ji(j′)) in
            ∃ r:R • r ∈ routes(n) ∧
                first_Ji(r) = ji ∧ last_Ji(r) = ji′ end
```

Animation: Net Decomposition

```
value
    decompose: N → N-set
    decompose(n) as ns
        obs_Ss(n) = ∪{obs_Ss(n′)|n′:N•n′ ∈ ns} ∧
        obs_Js(n) = ∪{obs_Js(n′)|n′:N•n′ ∈ ns} ∧
        {} = ∩{obs_Ss(n′)|n′:N•n′ ∈ ns} ∧
        {} = ∩{obs_Js(n′)|n′:N•n′ ∈ ns} ∧
        ∀ n′:N•n′ ∈ ns ⇒ connected(n′) ∧ ...
```

Lemma:

```
    ∀ n:N •
        let ns = decompose (n) in
        ∀ n′,n″:N • {n′,n″}⊆ns ∧ n′≠n″ ⇒
            obs_Ss(n′) ∩ obs_Ss(n″) = {} ∧
            obs_Js(n′) ∩ obs_Js(n″) = {} end
```

3.6.3 Animation: Multi-Modal Nets

Animation: General Issues

type
  M

Animation: Segment and Junction Modes

value
  obs_M: S $\to$ M
  obs_Ms: J $\to$ M-**set**
axiom
  $\forall$ n:N, j:J $\bullet$ j $\in$ obs_Js(n) $\Rightarrow$
    let ss = xtr_Ss(n,obs_Ji(j)) **in**
    obs_Ms(j) = {obs_M(s)|s:S $\bullet$ s $\in$ ss} **end**
  $\forall$ n:N, s:S $\bullet$ s $\in$ obs_Ss(n) $\Rightarrow$
    let {ji,ji$'$} = obs_Jis(s) **in**
    let {j,j$'$} = {xtr_J(n,ji),xtr_J(n,ji$'$)} **in**
    obs_M(s) $\in$ obs_Ms(j) $\cap$ obs_Ms(j$'$) **end end**

Animation: Single-Modal Nets and Net Projection

type
  mmN = {|n:N $\bullet$ **card** xtr_Ms(n) > 1|}
  smN = {|n:N $\bullet$ **card** xtr_Ms(n) = 1|}
value
  xtr_Ms: N $\to$ M-**set**
  xtr_Ms(n) $\equiv$ {obs_M(s) | s:S $\bullet$ s $\in$ obs_Ss(n)}

  projs: N $\to$ smN-**set**
  projs(n) $\equiv$ {proj(n,m) | m:M $\bullet$ m $\in$ xtr_Ms(n)}

  proj: N $\times$ M $\to$ smN
  proj(n,m) **as** n$'$
    post
      let ss = obs_Ss(n), ss$'$ = obs_Ss(n$'$),
        js = obs_Js(n), js$'$ = obs_Js(n$'$) **in**
      ss$'$ = {s | s:S $\bullet$ s $\in$ ss $\wedge$ m=obs_M(s)} $\wedge$
      js$'$ = {j | j:J $\bullet$ j $\in$ js $\wedge$ m $\in$ obs_Ms(j)}
      end

3.6.4 Animation: Segment and Junction Attributes

Animation: Segment and Junction Attribute Observations

type
  L, TI
value
  ms:M-**set**,   **axiom** $\text{ms} \neq \{\}$
  $\text{obs\_L: S} \rightarrow \text{L}$
  $\text{obs\_L: Si} \times \text{J} \times \text{M} \times \text{Si} \rightarrow \text{L}$
  $\text{obs\_TI: S} \rightarrow \text{TI}$
  $\text{obs\_TI: Si} \times \text{J} \times \text{Si} \rightarrow \text{TI}$
  $\text{obs\_TI: J} \times \text{M} \xrightarrow{\sim} \text{TI},$ **pre** $\text{obs\_TI(j,m): m} \in \text{obs\_Ms(j)}$
  $\text{obs\_TI: J} \times \text{M} \times \text{M} \xrightarrow{\sim} \text{TI},$ **pre** $\text{obs\_TI(j,m,m}'\text{): } \{\text{m,m}'\} \subseteq \text{obs\_Ms(j)}$
  $\text{obs\_arr\_TI: J} \times \text{M} \xrightarrow{\sim} \text{TI},$ **pre** $\text{obs\_arr\_TI(j,m): m} \in \text{obs\_Ms(j)}$
  $\text{obs\_dep\_TI: J} \times \text{M} \xrightarrow{\sim} \text{TI},$ **pre** $\text{obs\_dep\_TI(j,m): m} \in \text{obs\_Ms(j)}$
  $+\text{: L} \times \text{L} \rightarrow \text{L}$
  $+\text{: TI} \times \text{TI} \rightarrow \text{TI}$

Animation: Route Lengths

value
  $\text{length: R} \rightarrow \text{N} \xrightarrow{\sim} \text{L}$
  $\text{length(r)(n)} \equiv$
    case r **of**
      $\langle\rangle \rightarrow 0,$
      $\langle\text{(jf,si,jt)}\rangle \rightarrow \text{obs\_L(xtr\_S(si,n))},$
      $\langle\text{(ji1,sii,ji2),(jj1,sij,jj2)}\rangle\widehat{\ }\text{r}' \rightarrow$
        let si=xtr_S(sii,n),sj=xtr_S(sij,n) **in**
        $\text{obs\_L(si)} + \text{obs\_L(sii,xtr\_J(ji2,n),sij)} + \text{length}(\langle\text{(jj1,sij,jj2)}\rangle\widehat{\ }\text{r}')$ **end**
    end
    pre: $\text{r} \in \text{routes(n)} \wedge \text{ji2=jj1}$

value
  $\text{shortest\_route: Ji} \times \text{Ji} \rightarrow \text{N} \xrightarrow{\sim} \text{R}$
  $\text{shortest\_route(jf,jt)(n)} \equiv$
    let rs = routes(n) **in**
    let $\text{crs} = \{\text{r}|\text{r:R}\bullet\text{r} \in \text{rs} \wedge \text{first\_Ji(r)=jf} \wedge \text{last\_Ji(r)=jt}\}$ **in**
    let $\text{sr:R} \bullet \text{sr} \in \text{crs} \wedge \sim\exists\ \text{r:R} \bullet \text{r} \in \text{crs} \wedge \text{length(r)(n)}<\text{length(sr)(n)}$ **in**
    sr **end end end**
    pre: $\{\text{jf,jt}\} \subseteq \text{obs\_Jis(n)} \wedge \text{jf} \neq \text{jt}$

Animation: Route Traversal Times

type
  Q
  FCT = R → N → Q
value
  less: Q × Q → **Bool**
  lowest: Ji × Ji → N → FCT → R
  lowest(jf,jt)(n)(fct) ≡
    let rs = routes(n) **in**
    let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=jf ∧ last_Ji(r)=jt} **in**
    let lr:R • lr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ less(fct(r)(n),fct(qr)(n)) **in**
    lr **end end end**

  Q
  PRE = Q × Q → **Bool**
  FCT = R → N → Q
value
  best: Ji × Ji → N → FCT → PRE → R
  best(cf,ct)(n)(fct)(pre) ≡
    let rs = routes(n) **in**
    let crs = {r|r:R•r ∈ rs ∧ first_Ji(r)=cf ∧ last_Ji(r)=ct} **in**
    let br:R • lr ∈ crs ∧ ∼∃ r:R • r ∈ crs ∧ pre(fct(r)(n),fct(qr)(n)) **in**
    br **end end end**

Animation: Transporation Costs

type
  K, F
value
  obs_K: (S|J) → K
  obs_F: (S|J) → F

  +: K × K → K

  cost: R → N → K
  cost(r)(n) ≡
    case r **of**
      ⟨⟩ → 0,
      ⟨(jf,si,jt)⟩ →
        obs_K(xtr_J(jf,n))+obs_K(xtr_S(si,n))+obs_K(xtr_J(jt,n))
      ⟨(jf,si,jt),(jf′,si′,jt′)⟩⌢r′ → **assert:** jt=jf′

```
        obs_K(xtr_J(jf,n))+obs_K(xtr_S(si,n))+...+cost(r′)
    end

cheapest: Ji×Ji → N → ((K×K)→K) → ((K×K)→Bool) → R
cheapest(jf,jt)(n) ≡
    best(jf,jt)(n)(λ(k1,k2):(K×K)•k1+k2)(λ(k1,k2):(K×K)•k1<k2)
```

3.6.5 Animation: Road Nets

```
type
    M == road | ...
    Jc, Js, Jse
    Jic, Jis, Jise
    J = Jc | Js | Jse
    Cn
value
    is_composite_J: J → Bool
    is_simple_J: J → Bool
    is_simple_embedded_J: J → Bool
    obs_N: Jc → N
    obs_Jic: Jc → Jic, obs_Jis: Js → Jis, obs_Jise: Jse → Jise
    obs_Cn: Jic → Cn, obs_Cn: Jise → Cn
    obs_Jise: Jic → Jise
axiom
    ∀ j:Jc • is_composite_J(j) ∧ xtr_Ms(obs_N(j,road))={road},
    ∀ j:Js • is_simple_J(j),
    ∀ j:Jse • is_simple_embedded_J(j)

    ∀ n:N,j:J • j ∈ obs_Js(n) ∧ is_composite_J(j) ⇒
        let rn = obs_N(j) in
            ...
        end
```

3.6.6 Animation: Railway Nets

Animation: General

```
value
    proj: J × {railway} → ST
```

value
proj: $S \times \{railway\} \to LI$

Animation: Lines, Stations, Units and Connectors

type
RN = {| n:smN • obs_M(n)=railway |}
LI = {| s:S • obs_M(s)=railway |}
ST = {| c:C • obs_M(c)=railway |}
Tr, U, K

value
obs_LIs: RN → LI-**set**
obs_STs: RN → ST-**set**
obs_Us: RN → U-**set**
obs_Us: LI → U-**set**
obs_Us: ST → U-**set**
obs_Trs: ST → Tr-**set**
is_Linear: U → **Bool**
is_Switch: U → **Bool**
is_Simple_Crossover: U → **Bool**
is_Switchable_Crossover: U → **Bool**
obs_Ks: U → K-**set**

lin_seq: U-**set** → **Bool**
lin_seq(us) ≡
∀ u:U • u ∈ us ⇒ is_Linear(u) ∧
∃ q:U* • **len** q = **card** us ∧ **elems** q = us ∧
∀ i:**Nat** • {i,i+1} ⊆ **inds** q ⇒ ∃ k:K •
obs_Ks(q(i)) ∩ obs_Ks(q(i+1)) = {k} ∧
len q > 1 ⇒ obs_Ks(q(i)) ∩ obs_Ks(q(**len** q)) = {}

axiom

∀ n:RN • **card** obs_LIs(n) ≥ 1 ∧ **card** obs_STs(n) ≥ 2
∀ n:RN, l:LI • l ∈ obs_LIs(n) ⇒ lin_seq(l)
∀ n:RN, l:LI • l ∈ obs_LIs(n) ⇒ obs_Us(l) ⊆ obs_Us(n)
∀ n:RN, s:ST • s ∈ obs_STs(n) ⇒ **card** obs_Us(s) ≥ 1
∀ n:RN, s:ST • s ∈ obs_LIs(n) ⇒ obs_Us(s) ⊆ obs_Us(n)
∀ n:RN,l,l′:LI•{l,l′}⊆obs_LIs(n)∧l≠l′⇒obs_Us(l)∩ obs_Us(l′)={}
∀ n:RN,l:LI,s:ST•l ∈ obs_LIs(n)∧s ∈ obs_STs(n)⇒obs_Us(l)∩ obs_Us(s)={}
∀ n:RN,s,s′:ST•{s,s′}⊆obs_STs(n)∧s≠s′⇒obs_Us(s)∩ obs_Us(s′)={}

$$
\begin{array}{l}
\quad \forall\, \mathrm{s{:}ST}\bullet \mathbf{card}\ \mathrm{obs\_Trs(s)} \geq 1 \\
\quad \forall\, \mathrm{n{:}RN,s{:}ST,t{:}Tr}\bullet \mathrm{s} \in \mathrm{obs\_STs(n)} \wedge \mathrm{t} \in \mathrm{obs\_Trs(s)} \Rightarrow \mathrm{lin\_seq(t)} \\
\quad \forall\, \mathrm{n{:}RN,s{:}ST,t,t'{:}Tr}\bullet \mathrm{s} \in \mathrm{obs\_STs(n)} \wedge \{\mathrm{t,t'}\} \subseteq \mathrm{obs\_Trs(s)} \wedge \mathrm{t} \neq \mathrm{t'} \\
\qquad \Rightarrow \mathrm{obs\_Us(t)} \cap \mathrm{obs\_Us(t')} = \{\} \\
\quad \mathrm{n{:}RN} \bullet \forall\, \mathrm{k{:}K} \bullet \\
\qquad \mathrm{k} \in \cup\{\mathrm{obs\_Ks(u)} \mid \mathrm{u{:}U}\bullet \mathrm{u} \in \mathrm{obs\_Us(n)}\} \\
\qquad\quad \Rightarrow \mathbf{card}\{\mathrm{u} \mid \mathrm{u{:}U}\bullet \mathrm{u} \in \mathrm{obs\_Us(n)} \wedge \mathrm{k} \in \mathrm{obs\_Ks(u)}\} \leq 2 \\
\quad \forall\, \mathrm{n{:}RN,l{:}LI} \bullet\ \mathrm{l} \in \mathrm{obs\_LIs(n)} \Rightarrow \\
\qquad \exists\, \mathrm{s,s'{:}ST} \bullet \{\mathrm{s,s'}\} \subseteq \mathrm{obs\_STs(n)} \wedge \mathrm{s} \neq \mathrm{s'} \Rightarrow \\
\qquad\quad \mathbf{let}\ \mathrm{sus{=}obs\_Us(s),sus'{=}obs\_Us(s'),lus{=}obs\_Us(l)}\ \mathbf{in} \\
\qquad\quad \exists\, \mathrm{u,u',u'',u'''{:}U} \bullet \mathrm{u} \in \mathrm{sus} \wedge \\
\qquad\qquad \mathrm{u'} \in \mathrm{sus'} \wedge \{\mathrm{u'',u'''}\} \subseteq \mathrm{lus} \Rightarrow \\
\qquad\qquad \mathbf{let}\ \mathrm{sks} = \mathrm{obs\_Ks(u)},\ \mathrm{sks'} = \mathrm{obs\_Ks(u')}, \\
\qquad\qquad\qquad \mathrm{lks} = \mathrm{obs\_Ks(u'')},\ \mathrm{lks'} = \mathrm{obs\_Ks(u''')}\ \mathbf{in} \\
\qquad\qquad \exists!\mathrm{k,k'{:}K}\bullet \mathrm{k} \neq \mathrm{k'} \wedge \mathrm{sks} \cap \mathrm{lks} = \{\mathrm{k}\} \wedge \mathrm{sks'} \cap \mathrm{lks'} = \{\mathrm{k'}\} \\
\qquad\quad \mathbf{end\ end}
\end{array}
$$

Animation: Paths, States and Routes

Animation: Train Movement and Traffic

Animation: Time Tables

Animation: Etcetera

3.6.7 Animation: Net Dynamics

Animation: Segment and Junction States

$$
\begin{array}{l}
\mathbf{type} \\
\quad \mathrm{S\Sigma} = (\mathrm{Ji} \times \mathrm{Ji})\textbf{-set} \\
\quad \mathrm{J\Sigma} = (\mathrm{Si} \times \mathrm{Si})\textbf{-set} \\
\mathbf{value} \\
\quad \mathrm{obs\_S\Sigma}{:}\ \mathrm{S} \to \mathrm{S\Sigma} \\
\quad \mathrm{obs\_J\Sigma}{:}\ \mathrm{J} \to \mathrm{J\Sigma} \\
\\
\quad \mathrm{xtr\_Jis}{:}\ \mathrm{S\Sigma} \to \mathrm{Ji}\textbf{-set},\ \mathrm{xtr\_Jis}(\mathrm{s}\sigma) \equiv \{\mathrm{ji} \mid \mathrm{ji{:}Ji} \bullet (\mathrm{ji},\_) \in \mathrm{obs\_s}\sigma \vee (\_,\mathrm{ji}) \in \mathrm{obs\_s}\sigma\} \\
\quad \mathrm{xtr\_Sis}{:}\ \mathrm{J\Sigma} \to \mathrm{Si}\textbf{-set},\ \mathrm{xtr\_Sis}(\mathrm{j}\sigma) \equiv \{\mathrm{si} \mid \mathrm{si{:}Si} \bullet (\mathrm{si},\_) \in \mathrm{obs\_j}\sigma \vee (\_,\mathrm{si}) \in \mathrm{obs\_j}\sigma\} \\
\mathbf{axiom} \\
\quad \forall\, \mathrm{s{:}S} \bullet \mathrm{xtr\_Jis(obs\_S\Sigma(s))} \subseteq \mathrm{xtr\_Jip(s)}, \\
\quad \forall\, \mathrm{j{:}J} \bullet \mathrm{xtr\_Sis(obs\_J\Sigma(j))} \subseteq \mathrm{xtr\_Sis(j)} \\
\\
\mathbf{axiom} \\
\quad \ldots
\end{array}
$$

Animation: Segment and Junction State Spaces

type
  SΩ = SΣ-**set**
  JΩ = JΣ-**set**
value
  obs_SΩ: S → SΩ
  obs_JΩ: J → JΩ
axiom
  ∀ s:S • obs_SΣ(s) ⊆ obs_SΩ(s),
  ∀ j:J • obs_JΣ(j) ⊆ obs_JΩ(j)

Animation: Open and Closed Routes

value
  is_closed_R: R → **Bool**
  is_open_R: R → **Bool**

  is_closed_R() ≡

  is_open_R() ≡

3.7 Common Interface and Machine Requirements

3.8 Distinct Interface and Machine Requirements

3.8.1 Maintenance: Interface and Machine Requirements

3.8.2 Logistics: Interface and Machine Requirements

3.8.3 Animation: Interface and Machine Requirements

Chapter 4

Software Design Models of Transportation Software

Chapter 5

Discussion

Chapter 6

Closing

Bibliographical Notes

Bibliography

[1] Dines Bjørner. *Software Engineering*, volume 1: Abstraction and Modelling, vol. 2: Specification of Systems and Languages, vol. 3: Domains, Requirements and Software Design of *Texts in Theoretical Computer Science, the EATCS Series*. Springer–Verlag, 2005. Chapters 12–14 of vol.2 primarily authored by Christian Krog Madsen.