
Software
Engineering 1
Software
Engineering 2
Software
Engineering 3

WELCOME

Domain Science & Engineering
A Prerequisite for Requirements Engineering
An ASSC Tutorial

Melbourne: Wednesday, May 28, 2014: 9:00–12:00

Dines Bjørner
Fredsvej 11, DK–2840 Holte, Denmark

April 8, 2014: 12:26

Summary

- These lectures cover
  - a **new science & engineering of domains** as well as
  - a **new foundation for software development**.

  We treat the latter first.

- Instead of commencing with **requirements engineering**,
  - whose pursuit may involve repeated,
  - but unstructured forms of **domain analysis**,
  - we propose a predecessor phase of **domain engineering**.
- That is, we single out **domain analysis** as an activity to be pursued prior to **requirements engineering**.

- In emphasising domain engineering as a predecessor phase
  - we, at the same time, introduce **a number of facets**
  - that are **not present,** we think,
  - **in current software engineering** studies and practices.
- **One facet** is **the construction of separate domain descriptions.**
  - Domain descriptions are void of any reference to requirements
  - and encompass the modelling of domain phenomena
  - without regard to their being computable.

- **Another facet** is
  - **the pursuit of domain descriptions**
  - **as a free-standing activity.**
- In these lectures we emphasize
  - domain description development need not lead to software development.
  - This gives a new meaning to business process engineering, and should lead to
    - a deeper understanding of a domain
    - and to possible non-IT related business process re-engineering of areas of that domain.

- In these lectures we shall investigate
  - a method for analysing domains,
  - for constructing domain descriptions
  - and some emerging scientific bases.
- We shall also, less detailed, cover
  - basic principles, techniques and tools
  - for "deriving" requirements from domain descriptions.

- **Our contribution** to **domain analysis** is
  - that we view domain analysis
  - as a variant of formal concept analysis [Wille:ConceptualAnalysis1999],
    - a contribution which can be formulated by the "catch phrase"
    - **domain entities and their qualities form Galois connections**,
  - and further contribute with a methodology of
  - necessary corresponding principles and techniques of domain analysis.

- Those corresponding principles and techniques hinge on our view of domains as having the following **ontology**.
  - There are the **entities** that we can describe and then there is "the rest" which we leave un-described.
  - We **analyse** entities into
    - **endurant entities** and
    - **perdurant entities** ,

    that is,
    - **parts** and **materials** as **endurant entities** and
    - **discrete actions**, **discrete events** and **behaviours** as **perdurant entities** , respectively.
- Another way of looking at **entities** is as
  - **discrete entities** , or as
  - **continuous entities**.

- We also contribute to the **analysis** of **discrete endurants** in terms of the following notions:
  - **part types** and **material types**,
  - **part unique identifiers**,
  - **part mereology** and
  - **part attributes** and **material attributes** and
  - **material laws**.
- Of the above we point to the introduction, into **computing science** and **software engineering** of the notions of
  - **materials** and
  - **continuous behaviours**

  as novel.

- The example formalisations are expressed in
  - **RAISE**,
- but could as well have been expressed in, for example,
  - **Alloy**,
  - **Event B**,
  - **VDM** or
  - **Z**.

1. Introduction

- This is primarily a **methodology** set of lectures.
- By a **method** we shall understand
  - a set of **principles**
  - for **selecting** and **applying**
  - a number of **techniques** and **tools**
  - in order to **analyse** a **problem**
  - and **construct** an **artefact**.
- By **methodology** we shall understand
  - the study and knowledge about methods.

1.1. Domains: Some Definitions

- By a **domain** we shall here understand
  - an area of human activity
  - characterised by observable phenomena:
    - **entities**
      - whether **endurants** (manifest **parts** and **materials**)
      - or **perdurants** (**actions**, **events** or **behaviours**),
    - whether
      - **discrete** or
      - **continuous**;
    - and of their **properties**.

Example: 1 Some Domains Some examples are:

air traffic,
airport,
banking,
consumer market,
container lines,
fish industry,
health care,
logistics,
manufacturing,
pipelines,
securities trading,
transportation
etcetera. ■

1.1.1. Domain Analysis

- By **domain analysis** we shall understand
  - an inquiry into the domain,
  - its **entities**
  - and their **properties**.

Example: 2 A Container Line Analysis.

- *parts:*
  - container,
  - vessel,
  - terminal port, etc.;
- *actions:*
  - container loading,
  - container unloading,
  - vessel arrival in port, etc.;
- *events:*
  - container falling overboard;
  - container afire;
  - etc.;
- *behaviour:*
  - vessel voyage,
  - across the seas,
  - visiting ports, etc.

Length of a container is a container *property*.
Name of a vessel is a vessel *property*.
Location of a container terminal port is a port *property*.

1.1.2. Domain Descriptions

- By a **domain description** we shall understand
  - a **narrative description**
  - tightly coupled (say line-number-by-line-number)
  - to a **formal description**.
- To develop a **domain description**
  requires a thorough amount of **domain analysis**.

Example: 3 A Transport Domain Description.

- *Narrative:*
  - a transport net, n:N,
    consists of an aggregation of hubs, hs:HS,
    which we "concretise" as a set of hubs, **H-set**, and
    an aggregation of links, ls:LS, that is, a set **L-set**,
- *Formalisation:*
  - **type** N, HS, LS, Hs = **H-set**, Ls = **L-set**, H, L
    value
    obs_HS: N→HS,
    obs_LS: N→LS.
    obs_Hs: HS→**H-set**,
    obs_Ls: LS→**L-set**.

1.1.3. Domain Engineering

- By **domain engineering** we shall understand
  - the **engineering** of a domain description,
  - that is,
    - the rigorous construction of domain descriptions, and
    - the further analysis of these, creating **theories of domains**.

- The size, structure and complexity of interesting domain descriptions is usually such as to put a special emphasis on engineering:
  - the management and organisation of several, typically 5–6 collaborating domain describers,
  - the ongoing check of description quality, completeness and consistency, etcetera.

1.1.4. Domain Science

- By **domain science** we shall understand
  - two things:
    - the general study and knowledge of
      - how to create and handle domain descriptions
      - (a general theory of domain descriptions)

      and
    - the specific study and knowledge of a particular domain.
  - The two studies intertwine.

1.2. The Triptych of Software Development

- We suggest a "dogma":
  - before **software** can be **designed** one must understand[1] the **requirements**; and
  - before **requirements** can be expressed one must understand[2] the **domain**.
- We can therefore view software development as ideally proceeding in three (i.e., **TripTych**) phases:
  - an initial phase of **domain engineering**, followed by
  - a phase of **requirements engineering**, ended by
  - a phase of **software design**.

[1] Or maybe just: have a reasonably firm grasp of
[2] See previous footnote!

- In the domain engineering phase ($\mathcal{D}$)
  - a domain is analysed, described and "theorised",
  - that is, the beginnings of a specific domain theory is established.
- In the requirements engineering phase ($\mathcal{R}$)
  - a requirements prescription is constructed —
  - significant fragments of which are "derived",
  - systematically, from the domain description.
- In the software design phase ($\mathcal{S}$)
  - a software design
  - is derived, systematically, rigorously or formally,
  - from the requirements prescription.
- Finally the $\mathcal{S}$oftware is proven correct with respect to the $\mathcal{R}$equirements under assumption of the $\mathcal{D}$omain: $\mathcal{D}, \mathcal{S} \models \mathcal{R}$.

1.3. Issues of Domain Science & Engineering

- We specifically focus on the following issues of domain science &[3] engineering:
  - (i) which are the "things" to be described[4],
  - (ii) how to analyse these "things" into description structures[5],
  - (iii) how to describe these "things" informally and formally,
  - (iv) how to further structure descriptions[6], and a further study of
  - (v) mereology[7].

[3] When we put '&' between two terms that the compound term forms a whole concept.

[4] endurants [manifest entities henceforth called parts and materials] and perdurants [actions, events, behaviours]

[5] atomic and composite, unique identifiers, mereology, attributes

[6] *intrinsics, support technology, rules & regulations, organisation & management, human behaviour* etc.

[7] the study and knowledge of parts and relations of parts to other parts and a "whole".

1.4. Structure of Lectures

- First we introduce the problem. And that was done above.
- We start with an Example.
  - The example is that of a domain of road traffic.
- Then
  - we bring a rather careful analysis of
  - the concept of the **observable, manifest phenomena**
  - that we shall refer to as **entities**.

- That analysis focuses on
  - ◈ **endurant entities**, also called **parts** and **materials**,
    - ◎ those that can be observed at no matter what time,
    - ◎ i.e., entities of substance or continuant, and
  - ◈ **perdurant entities**: **action**, **event** and **behaviour** entities, those
    - ◎ that occur,
    - ◎ that happen,
    - ◎ that, in a sense, are accidents.

- **We think** that this "decomposition" of the "data analysis" problem into
  - discrete parts and continuous materials,
  - atomic and composite parts,
  - their unique identifiers and mereology, and
  - their attributes
  - **is novel**,
  - and differs from past practices in domain analysis.

- The formal descriptions will here be expressed in the **RAISE** Specification Language, RSL.
- We otherwise refer to [TheSEBook1wo].
- Appendix C of the tutorial notes brings a short primer, mostly on the syntactic aspects of RSL.
- But other model-oriented formal specification languages can be used with equal success; for example:
  - **Alloy**,
  - **Event B**,
  - **VDM** and
  - **Z**.

2. The Main Example – Example 3: Road Traffic System

2.1. Parts

2.1.1. Root Sorts

- The domain,
  - the stepwise unfolding of
  - whose description is
  - to be exemplified,

  is that of a composite traffic system
  - with a road net,
  - with a fleet of vehicles
  - of whose individual position on the road net we can speak, that is, monitor.

1. We analyse the composite traffic system into
   [a] a composite road net,
   [b] a composite fleet (of vehicles), and
   [c] an atomic monitor.

type
1. Δ
1a. N
1b. F
1c. M

value
1a. **obs_**N: $\Delta \rightarrow$ N
1b. **obs_**F: $\Delta \rightarrow$ F
1c. **obs_**M: $\Delta \rightarrow$ M

2.1.2. Sub-domain Sorts and Types

2. From the road net we can observe
   [a] a composite part, HS, of road (i.e., street) intersections (hubs) and
   [b] an composite part, LS, of road (i.e., street) segments (links).

type
2. HS, LS
value
2a. **obs_**HS: N → HS
2b. **obs_**LS: N → LS

3. From the fleet sub-domain, F, we observe a composite part, VS, of vehicles

type

3. VS

value

3. **obs_**VS: F → VS

4. From the composite sub-domain VS we observe
   [a] the composite part Vs, which we concretise as a set of vehicles
   [b] where vehicles, V, are considered atomic.

type
4a. Vs = V-**set**
4b. V
value
4a. **obs_**Vs: VS → V-**set**

- The "monitor" is considered atomic; it is an abstraction of the fact that
  - we can speak of the positions of each and every vehicle on the net
  - without assuming that we can indeed pin point these positions
  - by means of for example sensors.

2.1.3. Further Sub-domain Sorts and Types

- We now analyse the sub-domains of HS and LS.

5. From the hubs aggregate we decide to observe
   [a] the concrete type of a set of hubs,
   [b] where hubs are considered atomic; and
6. from the links aggregate we decide to observe
   [a] the concrete type of a set of links,
   [b] where links are considered atomic;

type
5a. Hs = H-**set**
6a. Ls = L-**set**
5b. H
6b. L
value
5. **obs**_Hs: HS → H-**set**
6. **obs**_Ls: LS → L-**set**

- We have no composite parts left to further analyse into parts
  - whether they be again composite
  - or atomic.

2.2. Properties

- Parts are distinguished by their properties:
  - the types and
  - the values of these.
- We consider three kinds of properties:
  - unique identifiers,
  - mereology and
  - attributes.

2.2.1. Unique Identifications

7. We decide the following:
   [a] each hub has a unique hub identifier,
   [b] each link has a unique link identifier and
   [c] each vehicle has a unique vehicle identifier.

type
7a. HI
7b. LI
7c. VI
value
7a. **uid**_H: H → HI
7b. **uid**_L: L → LI
7c. **uid**_V: V → VI

2.2.2. Mereology
2.2.2.1 Road Net Mereology

- By *mereology* we mean the study, knowledge and practice of understanding parts and part relations.

8. Each link is connected to exactly two hubs, that is,
   [a] from each link we can observe its mereology, that is, the identities of these two distinct hubs,
   [b] and these hubs must be of the net of the link;
9. and each hub is connected to zero, one or more links, that is,
   [a] from each hub we can observe its mereology, that is, the identities of these links,
   [b] and these links must be of the net of the hub.

value
8a. **mereo_**L: L → HI-**set**, **axiom** ∀ l:L·**card** **mereo_**L(l)=2
axiom
8b. ∀ n:N,l:L,hi:HI · l ∈ **obs_**Ls(**obs_**LS(n)) ∧ hi ∈ **mereo_**L(l)
8b. ⇒ ∃ h:H·h ∈ **obs_**Hs(**obs_**HS(n))∧**uid_**H(h)=hi
value
9a. **mereo_**H: H → LI-**set**
axiom
9b. ∀ n:N,h:H,li:LI · h ∈ **obs_**Hs(**obs_**HS(n)) ∧ li ∈ **mereo_**H(h)
9b. ⇒ ∃ l:L·l ∈ **obs_**Ls(**obs_**LS(n))∧**uid_**L(l)=li

2.2.2.2 Fleet of Vehicles Mereology

- We shall omit treatment of mereology of vehicles.

2.2.3. Attributes

- We shall model attributes of
  - links,
  - hubs and
  - vehicles.
- The composite parts,
  - aggregations of hubs, HS and Hs,
  - aggregations of links, LS and Ls and
  - aggregations of vehicles, VS and Vs,

  also have attributes, but we shall omit modelling them here.

2.2.3.1 Attributes of Links

10. The following are attributes of links.
    [a] Link states, lσ:LΣ, which we model as possibly empty sets of pairs of distinct identifiers of the connected hubs.
        • A link state expresses the directions that are open to traffic across a link.
    [b] Link state spaces, lω:LΩ which we model as the set of link states.
        • A link state space expresses the states that a link may attain across time.
    [c] Further link attributes are length, location, etcetera.

- Link states are usually dynamic attributes
- whereas
  - link state spaces,
  - link length and
  - link location (usually some curvature rendition)

  are considered static attributes.

type
10a. $\mathrm{L}\Sigma = (\mathrm{HI} \times \mathrm{HI})$**-set**
axiom
10a. $\forall\ \mathrm{l}\sigma{:}\mathrm{L}\Sigma \cdot 0 \leq$ **card** $\mathrm{l}\sigma \leq 2$
value
10a. **attr_**$\mathrm{L}\Sigma$: $\mathrm{L} \to \mathrm{L}\Sigma$
axiom
10a. $\forall\ \mathrm{l}{:}\mathrm{L} \cdot$ **let** $\{\mathrm{hi},\mathrm{hi}'\}=$**mereo_**$\mathrm{L}(\mathrm{l})$ **in attr_**$\mathrm{L}\Sigma(\mathrm{l}) \subseteq \{(\mathrm{hi},\mathrm{hi}'),(\mathrm{hi}',\mathrm{hi})\}$ **end**
type
10b. $\mathrm{L}\Omega = \mathrm{L}\Sigma$**-set**
value
10b. **attr_**$\mathrm{L}\Omega$: $\mathrm{L} \to \mathrm{L}\Omega$
axiom
10b. $\forall\ \mathrm{l}{:}\mathrm{L} \cdot$ **let** $\{\mathrm{hi},\mathrm{hi}'\}=$**mereo_**$\mathrm{L}(\mathrm{l})$ **in attr_**$\mathrm{L}\Sigma(\mathrm{l}) \in$ **attr_**$\mathrm{L}\Omega(\mathrm{l})$ **end**
type
10c. LOC, LEN, ...
value
10c. **attr_**LOC: $\mathrm{L} \to$ LOC, **attr_**LEN: $\mathrm{L} \to$ LEN, ...

2.2.3.2 Attributes of Hubs

11. The following are attributes of hubs:
    [a] Hub states, $\mathsf{h}\sigma{:}\mathsf{H}\Sigma$, which we model as possibly empty sets of pairs of identifiers of the connected links.
        • A hub state expresses the directions that are open to traffic across a hub.
    [b] Hub state spaces, $\mathsf{h}\omega{:}\mathsf{H}\Omega$ which we model as the set of hub states.
        • A hub state space expresses the states that a hub may attain across time.
    [c] Further hub attributes are location, etcetera.

- Hub states are usually dynamic attributes
- whereas
  - ⟐ hub state spaces and
  - ⟐ hub location

  are considered static attributes.

type
11a. HΣ = (LI × LI)**-set**
value
11a. **attr_**HΣ: H → HΣ
axiom
11a. ∀ h:H · **attr_**HΣ(h)⊆{(li,li′)|li,li′:LI·{li,li′}⊆**mereo_**H(h)}
type
11b. HΩ = HΣ**-set**
value
11b. **attr_**HΩ: H → HΩ
axiom
11b. ∀ h:H · **attr_**HΣ(h) ∈ **attr_**HΩ(h)
type
11c. LOC, ...
value
11c. **attr_**LOC: L → LOC, ...

2.2.3.3 Attributes of Vehicles

12. Dynamic attributes of vehicles include
    [a] position
        i. at a hub (about to enter the hub — referred to by the link it is coming from, the hub it is at and the link it is going to, all referred to by their unique identifiers or
        ii. some fraction "down" a link (moving in the direction from a from hub to a to hub — referred to by their unique identifiers)
        iii. where we model fraction as a real between 0 and 1 included.
    [b] velocity, acceleration, etcetera.
13. All these vehicle attributes can be observed.

type

12a. VP = atH | onL

12(a)i. atH :: fli:LI × hi:HI × tli:LI

12(a)ii. onL :: fhi:HI × li:LI × frac:FRAC × thi:HI

12(a)iii. FRAC = **Real**, **axiom** ∀ frac:FRAC · 0 ≤ frac ≤ 1

12b. VEL, ACC, ...

value

13. **attr_**VP:V→VP, **attr_**onL:V→onL, **attr_**atH:V→atH

13. **attr_**VEL:V→VEL, **attr_**ACC:V→ACC

2.2.3.4 Vehicle Positions

14. Given a net, n:N, we can define the possibly infinite set of potential vehicle positions on that net, vps(n).
    [a] vps(n) is expressed in terms of the links and hubs of the net.
    [b] vps(n) is the
    [c] union of two sets:
        i. the potentially[8] infinite set of "on link" positions
        ii. for all links of the net
        and
        i. the finite set of "at hub" positions
        ii. for all hubs in the net.

[8] The 'potentiality' arises from the nature of FRAC. If fractions are chosen as, for example, 1/5'th, 2/5'th, ..., 4/5'th, then there are only a finite number of "on link" vehicle positions. If instead fraction are arbitrary infinitesimal quantities, then there are infinitely many such.

value

14. vps: N → VP-**infset**

14b. vps(n) ≡

14a. **let** ls=**obs**_Ls(**obs**_LS(n)), hs=**obs**_Hs(**obs**_HS(n)) **in**

14(c)i. { onL(fhi,uid(l),f,thi) | fhi,thi:HI,l:L,f:FRAC ·

14(c)ii. l ∈ ls ∧ {fhi,thi}=**mereo**_L(l) }

14c. ∪

14(c)i. { atH(fli,**uid**_H(h),tli) | fli,tli:LI,h:H ·

14(c)ii. h ∈ hs ∧ {fli,tli}⊆**mereo**_H(h) }

14a. **end**

- Given a net and a finite set of vehicles
  - we can distribute these over the net, i.e., assign initial vehicle positions,
  - so that no two vehicles "occupy" the same position, i.e., are "crashed" !
- Let us call the non-deterministic assignment function, i.e., a relation, for vpr.

15. vpm:VPM is a bijective map from vehicle identifiers to (distinct) vehicle positions.
16. vpr has the obvious signature.
17. vpr(vs)(n) is defined in terms of
18. a non-deterministic selection, vpa, of vehicle positions, and
19. a non-deterministic assignment of these vehicle positions to vehicle identifiers —
20. being the resulting distribution.

type

15. VPM′ = VI $\overrightarrow{m}$ VP
15. VPM = {| vpm:VPM′ · **card dom** vpm = **card rng** vpm |}

value

16. vpr: V**-set** × N → VMP
17. vpr(vs)(n) ≡
18. **let** vpa:VP**-set** · vpa ⊆ vps(vs)(n) ∧ **card** vpa = vard vs **in**
19. **let** vpm:VPM · **dom** vpm = vps ∧ **rng** vpm = vpa **in**
20. vpm **end end**

2.3. Definitions of Auxiliary Functions

21. From a net we can extract all its link identifiers.
22. From a net we can extract all its hub identifiers.

value

21. xtr_LIs: N → LI-**set**
21. xtr_LIs(n) ≡ {**uid**_L(l)|l:L·l ∈ **obs**_Ls(**obs**_LS(n))}
22. xtr_HIs: N → HI-**set**
22. xtr_HIs(n) ≡ {**uid**_H(l)|h:H·h ∈ **obs**_Hs(**obs**_HS(n))}

23. Given a link identifier and a net get the link with that identifier in the net.
24. Given a hub identifier and a net get the hub with that identifier in the net.

value

26. get_H: HI → N $\xrightarrow{\sim}$ H
26. get_H(hi)(n) ≡ ι h:H·h ∈ **obs**_Hs(**obs**_HS(n))∧**uid**_H(h)=hi
26. **pre**: hi ∈ xtr_HIs(n)
26a. get_L: LI → N $\xrightarrow{\sim}$ L
26a. get_L(li)(n) ≡ ι l:L·l ∈ **obs**_Ls(**obs**_LS(n))∧**uid**_L(l)=li
26a. **pre**: hl ∈ xtr_LIs(n)

- The ι a:A·$\mathcal{P}$(a) expression
  - yields the unique value a:A
  - which satisfies the predicate $\mathcal{P}$(a).
  - If none, or more than one exists then the function is undefined.

2.4. Some Derived Traffic System Concepts

2.4.1. Maps

25. A road map is an abstraction of a road net. We define one model of maps below.

    [a] A road map, RM, is a finite definition set function, M, (a specification language map) from
    - hub identifiers (the source hub)
    - to (such finite definition set) functions
    - from link identifiers
    - to hub identifiers (the target hub).

type

25a. $\mathrm{RM}' = \mathrm{HI} \xrightarrow{m} (\mathrm{LI} \xrightarrow{m} \mathrm{HI})$

- If a hub identifier in the source or an rm:RM maps into the empty map then the "corresponding" hub is "isolated": has no links emanating from it.

26. These road maps are subject to a well-formedness criterion.
    [a] The target hubs must be defined also as source hubs.
    [b] If a link is defined from source hub (referred to by its identifier) **shi** via link **li** to a target hub **thi**, then, vice versa, link **li** is also defined from source **thi** to target **shi**.

type
26. RM = {| rm:RM′ · wf_RM(rm) |}
value
26. wf_RM: RM′ → **Bool**
26. wf_RM(rm) ≡
26a. ∪ { **rng**(rm(hi))|hi:HI·hi ∈ **dom** rm } ⊆ **dom** rm
26b. ∧ ∀ shi:HI·shi ∈ **dom** rm ⇒
26b. ∀ li:LI · li ∈ **dom** rm(shi) ⇒
26b. li ∈ **dom** rm((rm(shi))(li)) ∧ (rm((rm(shi))(li)))(li)=shi

27. Given a road net, n, one can derive "its" road map.
    [a] Let hs and ls be the hubs and links, respectively of the net n.
    [b] Every hub with no links emanating from it is mapped into the empty map.
    [c] For every link identifier uid_L(l) of links, l, of ls and every hub identifier, hi, in the mereology of l
    [d] hi is mapped into a map from uid_L(l) into hi'
    [e] where hi' is the other hub identifier of the mereology of l.

```
value
27.  derive_RM: N → RM
27.  derive_RM(n) ≡
27a.     let hs = obs_Hs(obs_HS(n)), ls = obs_Ls(obs_LS(n)) in
27b.     [ hi ↦ [ ] | hi:HI · ∃ h:H · h ∈ hs ∧ mereo_H(h) = {} ] ∪
27d.     [ hi ↦ [ uid_L(l) ↦ hi′
27e.                          | hi′:HI · hi′ = mereo_L(l)\{hi} ]
27c.         | l:L,hi:HI · l ∈ ls ∧ hi ∈ mereo_L(l) ] end
```

- **Theorem:** If the road net, n, is well-formed then wf_RM(derive_RM(n)).

2.4.2. Traffic Routes

28. A traffic route, **tr**, is an alternating sequence of hub and link identifiers such that
    [a] **li:LI** is in the mereology of the hub, **h:H**, identified by **hi:HI**, the predecessor of **li:LI** in route **r**, and
    [b] **hi':HI**, which follows **li:LI** in route **r**, is different from **hi**, and is in the mereology of the link identified by **li**.

type
28. $R' = (HI|LI)^*$
28. $R = \{|\ r{:}R' \cdot \exists\ n{:}N \cdot wf\_R(r)(n)\ |\}$
value
28. $wf\_R{:}\ R' \to N \to \mathbf{Bool}$
28. $wf\_R(r)(n) \equiv$
28. $\quad \forall\ i{:}\mathbf{Nat} \cdot \{i,i+1\} \subseteq \mathbf{inds}\ r \Rightarrow$
28a. $\quad\quad \underline{\mathbf{is\_}}HI(r(i)) \Rightarrow \underline{\mathbf{is\_}}LI(r(i+1)) \wedge r(i+1) \in \underline{\mathbf{mereo\_}}H(get\_H(r(i))(n)),$
28b. $\quad\quad \underline{\mathbf{is\_}}LI(r(i)) \Rightarrow \underline{\mathbf{is\_}}HI(r(i+1)) \wedge\ r(i+1) \in \underline{\mathbf{mereo\_}}L(get\_L(r(i))(n))$

29. From a well-formed road map (i.e., a road net) we can generate the possibly infinite set of all routes through the net.
    [a] **Basis Clauses:**
        i. The empty sequence of identifiers is a route.
        ii. The one element sequences of link and hub identifiers of links and hubs of a road map (i.e., a road net) are routes.
        iii. If hi maps into some li in rm then ⟨hi,li⟩ and ⟨li,hi⟩ are routes of the road map (i.e., of the road net).
    [b] **Induction Clause:**
        i. Let r⌢⟨i⟩ and ⟨i′⟩⌢r′ be two routes of the road map.
        ii. If the identifiers i and i′ are identical, then r⌢⟨i⟩⌢r′ is a route.
    [c] **Extremal Clause:**
        i. Only such routes that can be formed from a finite number of applications of the above clauses are routes.

value

29. gen_routes: M → Routes-**infset**
29. gen_routes(m) ≡
29(a)i. **let** rs = {⟨⟩}
29(a)ii. ∪ {⟨li,hi⟩,⟨hi,li⟩|li:LI,hi:HI•...}
29(b)i. ∪ {**let** r⌢⟨li⟩,⟨li′⟩⌢r′:R • {r⌢⟨li⟩,⟨li′⟩⌢r′}⊆rs,
29(b)i. r″⌢⟨hi⟩,⟨hi′⟩⌢r‴:R • {r″⌢⟨hi⟩,⟨hi′⟩⌢r‴}⊆rs **in**
29(b)ii. r⌢⟨li⟩⌢r′,r″⌢⟨hi⟩⌢r‴ **end**} **in**
29(c)i. rs **end**

2.4.2.1 Circular Routes

30. A route is circular if the same identifier occurs more than once.

value

30. is_circular_route: R → **Bool**
30. is_circular_route(r) ≡ ∃ i,j:**Nat** · {i,j}⊆**inds** r ∧ i≠j ⇒ r(i)=r(j)

2.4.2.2 Connected Road Nets

31. A road net is connected if there is a route from any hub (or any link) to any other hub or link in the net.

```
31.  is_conn_N: N → Bool
31.  is_conn_N(n) ≡
31.      let m = derive_RM(n) in
31.      let rs = gen_routes(m) in
31.      ∀ i,i′:(LI|HI) · {i,i′}⊆xtr_LIs(n)∪ xtr_HIs(n)
31.          ∃ r:R · r ∈ rs ∧ r(1)=i ∧ r(len r)=i′ end end
```

2.4.2.3 Set of Connected Nets of a Net

32. The set, **cns**, of connected nets of a net, **n**, is
    [a] the smallest set of connected nets, **cns**,
    [b] whose hubs and links together "span" those of the net **n**.

value
32. conn_Ns: N → N**-set**
32. conn_Ns(n) **as** cns
32a. **pre**: **true**
32b. **post**: conn_spans_HsLs(n)(cns)
32a. ∧ ∼∃ kns:N**-set** · **card** kns < **card** cns
32a. ∧ conn_spans_HsLs(n)(kns)

32b. conn_spans_HsLs: N → N → **Bool**
32b. conn_spans_HsLs(n)(cns) ≡
32b. ∀ cn:N·cn ∈ cns ⇒ is_connected_N(n)(cn)
32b. ∧ **let** (hs,ls) = (**obs_**Hs(**obs_**HS(n)),**obs_**Ls(**obs_**LS(n))),
32b. chs = ∪{**obs_**Hs(**obs_**HS(cn))|cn ∈ cns},
32b. cls = ∪{**obs_**Ls(**obs_**LS(cn))|cn ∈ cns} **in**
32b. hs = chs ∧ ls = cls **end**

2.4.2.4 Route Length

33. The length attributes of links can be
    [a] added and subtracted,
    [b] multiplied by reals to obtain lengths,
    [c] divided to obtain fractions,
    [d] compared as to whether one is shorter than another, etc., and
    [e] there is a "zero length" designator.

value

33a. $+,- : \text{LEN} \times \text{LEN} \to \text{LEN}$
33b. $* : \text{LEN} \times \textbf{Real} \to \text{LEN}$
33c. $/ : \text{LEN} \times \text{LEN} \to \textbf{Real}$
33d. $<,\leq,=,\neq,\geq,> : \text{LEN} \times \text{LEN} \to \textbf{Bool}$
33e. $\ell_0 : \text{LEN}$

34. One can calculate the length of a route.

value

34. length: R → N → LEN
34. length(r)(n) ≡
34. **case** r **of**:
34. ⟨⟩ → ℓ_0,
34. ⟨si⟩⌢r′ →
34. is_LI(si)→**attr**_LEN(get_L(si)(n))+length(r′)(n)
34. is_HI(si)→length(r′)(n)
34. **end**

2.4.2.5 Shortest Routes

35. There is a predicate, is_R, which,
    [a] given a net and two distinct hub identifiers of the net,
    [b] tests whether there is a route between these.

value

35. is_R: N → (HI×HI) → **Bool**
35. is_R(n)(fhi,thi) ≡
35a. fhi ≠ thi ∧ {fht,thi}⊆xtr_HIs(n)
35b. ∧ ∃ r:R · r ∈ routes(n) ∧ **hd** r = fhi ∧ r(**len** r) = thi

36. The shortest between two given hub identifiers
    [a] is an acyclic route, r,
    [b] whose first and last elements are the two given hub identifiers
    [c] and such that there is no route, r′ which is shorter.

value
36. shortest_route: N → (HI×HI) → R
36a. shortest_route(n)(fhi,thi) **as** r
36b. **pre**: pre_shortest_route(n)(fhi,thi)
36c. **post**: pos__shortest_route(n)(r)(fhi,thi)

36b. pre_shortest_route: N → (HI×HI) → **Bool**
36b. pre_shortest_route(n)(fhi,thi) ≡
36b. is_R(n)(fhi,thi) ∧ fhi≠thi ∧ {fhi,thi}⊂xtr_HIs(n)

36c. pos_shortest_route: N → R → (HI×HI) → **Bool**
36c. pos_shortest_route(n)(r)(fhi,thi) ≡
36c. r ∈ routes(n)
36c. ∧ ∼∃ r′:R · r′ ∈ routes(n) ∧ length(r′) < length(r)

2.5. States

- There are different notions of state. In our example these are some of the states:
  - the road net composition of hubs and links;
  - the state of a link, or a hub; and
  - the vehicle position.

2.6. Actions

- An action is what happens when a function invocation changes, or potentially changes a state.
- Examples of traffic system actions are:
  - insertion of hubs,
  - insertion of links,
  - removal of hubs,
  - removal of links,
  - setting of hub state ($\mathsf{h}\sigma$),
  - setting of link state ($\mathsf{l}\sigma$),
  - moving a vehicle along a link,
  - moving a vehicle from a link to a hub and
  - moving a vehicle from a hub to a link.

37. The **insert** action applies to a net and a hub and conditionally yields an updated net.
    [a] The condition is that there must not be a hub in the "argument" net with the same unique hub identifier as that of the hub to be inserted and
    [b] the hub to be inserted does not initially designate links with which it is to be connected.
    [c] The updated net contains all the hubs of the initial net "plus" the new hub.
    [d] and the same links.

value

37. ins_H: N → H $\xrightarrow{\sim}$ N
37. ins_H(n)(h) **as** n′, **pre**: pre_ins_H(n)(h), **post**: post_ins_H(n)(h)

37a. pre_ins_H(n)(h) ≡
37a. ∼∃ h′:H · h′ ∈ **obs**_Hs(n) ∧ **uid**_HI(h)=**uid**_HI(h′)
37b. ∧ **mereo**_H(h) = {}

37c. post_ins_H(n)(h)(n′) ≡
37c. **obs**_Hs(n) ∪ {h} = **obs**_Hs(n′)
37d. ∧ **obs**_Ls(n) = **obs**_Ls(n′)

2.7. Events

- By an **event** we understand
  - a state change
  - resulting indirectly from an unexpected application of a function,
  - that is, that function was performed "surreptitiously".
- Events can be characterised by a pair of (before and after) states, a predicate over these and, optionally, a **time** or **time interval**.
- Events are thus like actions:
  - change states,
  - but are usually
    - either caused by "previous" actions,
    - or caused by "an outside action".

38. Link disappearance is expressed as a predicate on the "before" and "after" states of the net. The predicate identifies the "missing" ℓink (!).

39. Before the disappearance of link ℓ in net n
    [a] the hubs h' and h'' connected to link ℓ
    [b] were connected to links identified by $\{l'_1, l'_2, \ldots, l'_p\}$ respectively $\{l''_1, l''_2, \ldots, l''_q\}$
    [c] where, for example, l'_i, l''_j are the same and equal to uid_$\Pi(\ell)$.

```
38.  link_dis: N × N → Bool
38.  link_dis(n,n′) ≡
38.      ∃ ℓ:L · pre_link_dis(n,ℓ) ⇒ post_link_dis(n,ℓ,n′)
39.  pre_link_dis: N × L → Bool
39.  pre_link_dis(n,ℓ) ≡ ℓ ∈ obs_Ls(n)
```

40. After link ℓ disappearance there are instead
    [a] two separate links, ℓ_i and ℓ_j, "truncations" of ℓ
    [b] and two new hubs h''' and h''''
    [c] such that ℓ_i connects h' and h''' and
    [d] ℓ_j connects h'' and h'''';
    [e] Existing hubs h' and h'' now have mereology
        i. $\{l'_1, l'_2, \ldots, l'_p\} \setminus \{\mathsf{uid\_\Pi}(\ell)\} \cup \{\mathsf{uid\_\Pi}(\ell_i)\}$ respectively
        ii. $\{l''_1, l''_2, \ldots, l''_q\} \setminus \{\mathsf{uid\_\Pi}(\ell)\} \cup \{\mathsf{uid\_\Pi}(\ell_j)\}$
41. All other hubs and links of n are unaffected.

42. We shall "explain" *link disappearance* as the combined, instantaneous effect of
    [a] first a **remove link** "event" where the **removed link** connected **hub**s hi_j and hi_k;
    [b] then the **insert**ion of two new, "fresh" **hub**s, h_α and h_β;
    [c] "followed" by the **insert**ion of two new, "fresh" links $\mathsf{l}_{j\alpha}$ and $\mathsf{l}_{k\beta}$ such that
        i. $\mathsf{l}_{j\alpha}$ connects hi_j and h_α and
        ii. $\mathsf{l}_{k\beta}$ connects hi_k and $\mathsf{h}_{k\beta}$

value

42. post_link_dis(n,ℓ,n′) ≡
42. **let** h_a,h_b:H ·
42. **let** {li_a,li_b}=**mereo_**L(ℓ) **in**
42. (get_H(li_a)(n),get_H(li_b)(n)) **end in**
42a. **let** n″ = rem_L(n)(**uid_**L(ℓ)) **in**
42b. **let** h_α,h_β:H · {h_α,h_β}∩**obs_**Hs(n)={} **in**
42b. **let** n‴ = ins_H(n″)(h_α) **in**
42b. **let** n⁗ = ins_H(n‴)(h_β) **in**
42c. **let** $l_{j\alpha}$,$l_{k\beta}$:L · {$l_{j\alpha}$,$l_{k\beta}$}∩**obs_**Ls(n)={}
42c. ∧ **mereo_**L($l_{j\alpha}$) = {**uid_**H(h_a),**uid_**H(h_α)}
42c. ∧ **mereo_**L($l_{k\beta}$) = {**uid_**H(h_b),**uid_**H(h_β)} **in**
42(c)i. **let** n⁗⁗ = ins_L(n⁗)($l_{j\alpha}$) **in**
42(c)ii. n′ = ins_L(n⁗⁗)($l_{k\beta}$) **end end end end end end end**

2.8. Behaviours
2.8.1. Traffic
2.8.1.1 Continuous Traffic

- For the road traffic system
  - ◈ perhaps the most significant example of a behaviour
  - ◈ is that of its traffic
    43. the continuous time varying discrete positions of vehicles, vp:VP[9],
    44. where time is taken as a dense set of points.

type

44. $\mathbb{cT}$
43. $\mathrm{cRTF} = \mathbb{cT} \rightarrow (\mathrm{V} \xrightarrow[m]{} \mathrm{VP})$

[9] For VP see Item 12a on Slide 45.

2.8.1.2 Discrete Traffic

- We shall model, not continuous time varying traffic, but

45. discrete time varying discrete positions of vehicles,
46. where time can be considered a set of linearly ordered points.

46. $\mathbb{dT}$
45. $\mathbb{dRTF} = \mathbb{dT} \xrightarrow{m} (\mathbb{V} \xrightarrow{m} \mathbb{VP})$

47. The road traffic that we shall model is, however, of vehicles referred to by their unique identifiers.

type

47. $\mathbb{RTF} = \mathbb{dT} \xrightarrow{m} (\mathbb{VI} \xrightarrow{m} \mathbb{VP})$

2.8.1.3 Time: An Aside

- We shall take a rather simplistic view of time.

48. We consider **d**$\mathbb{T}$, or just $\mathbb{T}$, to stand for a totally ordered set of time points.
49. And we consider $\mathbb{TI}$ to stand for time intervals based on $\mathbb{T}$.
50. We postulate an infinitesimal small time interval δ.
51. $\mathbb{T}$, in our presentation, has lower and upper bounds.
52. We can compare times and we can compare time intervals.
53. And there are a number of "arithmetics-like" operations on times and time intervals.

type

48. $\mathbb{T}$
49. $\mathbb{TI}$

value

50. δ:$\mathbb{TI}$
51. $\mathbb{MIN}, \mathbb{MAX}$: $\mathbb{T} \to \mathbb{T}$
51. $<,\leq,=,\geq,>$: $(\mathbb{T}\times\mathbb{T})|(\mathbb{TI}\times\mathbb{TI}) \to$ **Bool**
52. $-$: $\mathbb{T}\times\mathbb{T} \to \mathbb{TI}$
53. $+$: $\mathbb{T}\times\mathbb{TI},\mathbb{TI}\times\mathbb{T} \to \mathbb{T}$
53. $-,+$: $\mathbb{TI}\times\mathbb{TI} \to \mathbb{TI}$
53. $*$: $\mathbb{TI}\times$**Real** $\to \mathbb{TI}$
53. $/$: $\mathbb{TI}\times\mathbb{TI} \to$ **Real**

54. We postulate a global clock behaviour which offers the current time.

55. We declare a channel clk_ch.

value

54. clock: $\mathbb{T} \rightarrow$ **out** clk_ch **Unit**

54. clock(t) $\equiv$ ... clk_ch!t ... clock(t $\lceil\rceil$ t+δ)

channnel

55. clk_ch:$\mathbb{T}$

2.8.2. Globally Observable Parts

- There is given

56. a net, n:N,
57. a set of vehicles, vs:V-set, and
58. a monitor, m:M.

- The n:N, vs:V-set and m:M are observable from the road traffic system domain.

value

56. n:N = **obs_**N(Δ)
56. ls:L-set = **obs_**Ls(**obs_**LS(n)), hs:H-set = **obs_**Hs(**obs_**HS(n)),
56. lis:LI-set = {**uid_**L(l)|l:L·l ∈ ls}, his:HI-set = {**uid_**H(h)|h:H·h ∈ hs}
57. vs:V-set = **obs_**Vs(**obs_**VS(**obs_**F(Δ))), vis:V-set = {**uid_**V(v)|v:V·v
58. m:**obs_**M(Δ)

2.8.3. Road Traffic System Behaviours

59. Thus we shall consider our road traffic system, **rts**, as
    [a] the concurrent behaviour of a number of vehicles and, to "observe", or, as we shall call it, to monitor their movements,
    [b] the **mon**itor behaviour, based on
    [c] the monitor and its unique identifier,
    [d] an initial vehicle position map, and
    [e] an initial starting time.

value

59c. mi:MI = **uid_**(m)

59d. vpm:VPM = vpr(vs)(n)

59e. t_0:T = clk_ch?

59. rts() =

59a. ‖ {veh(**uid_**V(v))(v)(vpm(**uid_**V(v)))|v:V·v ∈ vs}

59b. ‖ mon(mi)(m)([t_0 ↦ vpm])

- where the "extra" monitor argument
  - records the discrete road traffic, RTF,
  - initially set to the singleton map from an initial start time, t_0 to the initial assignment of vehicle positions.

2.8.4. Channels

- In order for the monitor behaviour to assess the vehicle positions
  - these vehicles communicate their positions
  - to the monitor
  - via a vehicle to monitor channel.
- In order for the monitor to time-stamp these positions
  - it must be able to "read" a clock.

60. Thus we declare a set of channels indexed by the unique identifiers of vehicles and communicating vehicle positions.

channel

60. {vm_ch[mi,vi]|vi:VI•vi ∈ vis}:VP

2.8.5. Behaviour Signatures

61. The road traffic system behaviour, **rts**, takes no arguments; and "behaves", that is, continues forever.
62. The vehicle behaviours are indexed by the unique identifier, **uid_V(v):VI**, the vehicle part, **v:V** and the vehicle position; offers communication to the monitor behaviour; and behaves "forever".
63. The monitor behaviour takes monitor part, **m:M**, as argument and also the discrete road traffic, **drtf:dRTF**; the behaviour otherwise runs forever.

value

61. rts: **Unit** → **Unit**
62. veh: vi:VI → v:V → VP → **out** vm_ch[vi],mi:MI **Unit**
63. mon: mi:MI → m:M → dRTF → **in** {vm_ch[mi,vi]|vi:VI·vi ∈ vis},clk_c

2.8.6. The Vehicle Behaviour

64. A vehicle process
    - is indexed by the unique vehicle identifier vi:VI,
    - the vehicle "as such", v:V and
    - the vehicle position, vp:VPos.

    The vehicle process communicates
    - with the monitor process on channel vm[vi]
    - (sends, but receives no messages), and
    - otherwise evolves "in[de]finitely" (hence **Unit**).

65. We describe here an abstraction of the vehicle behaviour **at** a **Hub** (**hi**).
    [a] Either the vehicle remains at that hub informing the monitor,
    [b] or, internally non-deterministically,
        i. moves onto a link, **tli**, whose "next" hub, identified by **thi**, is obtained from the mereology of the link identified by **tli**;
        ii. informs the monitor, on channel **vm[vi]**, that it is now on the link identified by **tli**,
        iii. whereupon the vehicle resumes the vehicle behaviour positioned at the very beginning (**0**) of that link,
    [c] or, again internally non-deterministically,
    [d] the vehicle "disappears — off the radar" !

65. veh(vi)(v)(vp:atH(fli,hi,tli)) ≡
65a. vm_ch[mi,vi]!vp ; veh(vi)(v)(vp)
65b. ⌈⌉
65(b)i. **let** {hi′,thi}=**mereo**_L(get_L(tli)(n)) **in assert:** hi′=hi
65(b)ii. vm_ch[mi,vi]!onL(tli,hi,0,thi) ;
65(b)iii. veh(vi)(v)(onL(tli,hi,0,thi)) **end**
65c. ⌈⌉
65d. **stop**

66. We describe here an abstraction of the vehicle behaviour **on** a **L**ink (**ii**).
    Either
    [a] the vehicle remains at that link position informing the monitor,
    [b] or, internally non-deterministically,
    [c] if the vehicle's position on the link has not yet reached the hub,
        i. then the vehicle moves an arbitrary increment δ along the link informing the monitor of this, or
        ii. else, while obtaining a "next link" from the mereology of the hub (where that next link could very well be the same as the link the vehicle is about to leave),
            A. the vehicle informs the monitor that it is now at the hub identified by **thi**,
            B. whereupon the vehicle resumes the vehicle behaviour positioned at that hub.
67. or, internally non-deterministically,
68. the vehicle "disappears — off the radar" !

64. veh(vi)(v)(vp:onL(fhi,li,f,thi)) ≡
66a. vm_ch[mi,vi]!vp ; veh(vi)(v)(vp)
66b. ⌈⌉
66c. **if** f + δ<1
66(c)i. **then** vm_ch[mi,vi]!onL(fhi,li,f+δ,thi) ;
66(c)i. veh(vi)(v)(onL(fhi,li,f+δ,thi))
66(c)ii. **else let** li′:LI·li′ ∈ **mereo**_H(get_H(thi)(n)) **in**
66(c)iiA. vm_ch[mi,vi]!atH(li,thi,li′);
66(c)iiB. veh(vi)(v)(atH(li,thi,li′)) **end end**
67. ⌈⌉
68. **stop**

2.8.7. The Monitor Behaviour

69. The monitor behaviour evolves around the attributes of an own "state", m:M, a table of traces of vehicle positions, while accepting messages about vehicle positions and otherwise progressing "in[de]finitely".
70. Either the monitor "does own work"
71. or, internally non-deterministically accepts messages from vehicles.
    [a] A vehicle position message, vp, may arrive from the vehicle identified by vi.
    [b] That message is appended to that vehicle's movement trace,
    [c] whereupon the monitor resumes its behaviour —
    [d] where the communicating vehicles range over all identified vehicles.

69. mon(mi)(m)(rtf) ≡
70. mon(mi)(own_mon_work(m))(rtf)
71. ⌈⌉
71a. [] { **let** ((vi,vp),t) = (vm_ch[mi,vi]?,clk_ch?) **in**
71b. **let** rtf′ = rtf † [t ↦ rtf(max **dom** rtf) † [vi ↦ vp]] **in**
71c. mon(mi)(m)(rtf′) **end**
71d. **end** | vi:VI · vi ∈ vis }

70. own_mon_work: M → dRTF → M

- We do not describe the clock behaviour by other than stating that it continually offers the current time on channel clkm_ch. ■

LECTURE 2

3. Domains

4. Domains

We characterise a number of terms.

4.0.0.1 Domain

- By a **domain** we shall here understand
  - an area of human activity
  - characterised by observable phenomena:
    - **entities**
      - whether **endurants** [**is_endurant(e)**]
        (manifest **parts** and **materials**)
      - or **perdurants** [**is_perdurant(e)**]
        (**actions**, **events** or **behaviours**),
    - whether
      - **discrete** [**is_discrete(e)**] or
      - **continuous** [**is_continuous(e)**] ;
    - and of their **properties**.

4.0.0.2 Domain Phenomena

- By a **domain phenomenon** we shall understand
  - something that can be observed by the **human senses**
  - or by **equipment** based on laws of physics and chemistry.
- Those phenomena that can be observed by
  - the human eye or
  - touched, for example, by human hands,
  - we call **parts** and **materials**.
- Those phenomena that can be observed of parts and materials
  - can usually be measured
  - and we call them **properties** of these **parts** and those **materials**.

4.0.0.3 Domain Entity

- By a **domain entity** we shall understand
  - a **manifest domain phenomenon** or
  - a **domain concept**, i.e., an **abstraction**,
  - derived from a **domain entity**.
- The distinction between
  - a **manifest domain phenomenon** and
  - a **concept** thereof, i.e., a **domain concept**,

  is important.
- Really, what we describe are the **domain concept**s derived
  - from **domain phenomena** or
  - from other **domain concept**s.

4.0.0.4 Endurant Entity

- We distinguish between
  - endurants and
  - perdurants.
- From Wikipedia:
  - *By an endurant (also known as a continuant or a substance) we shall understand an entity*
    - *that can be observed, i.e., perceived or conceived,*
    - *as a complete concept,*
    - *at no matter which given snapshot of time.*
  - *Were we to freeze time*
    - *we would still be able to observe the entire endurant.*

4.0.0.5 Perdurant Entity

- From Wikipedia:
  - *Perdurant: Also known as occurrent, accident or happening.*
  - *Perdurants are those entities for which only a fragment exists if we look at them at any given snapshot in time.*
  - *When we freeze time we can only see a fragment of the perdurant.*
  - *Perdurants are often what we know as processes, for example 'running'.*
  - *If we freeze time then we only see a fragment of the running, without any previous knowledge one might not even be able to determine the actual process as being a process of running.*
  - *Other examples include an activation, a kiss, or a procedure.*

4.0.0.6 Discrete Endurant

- We distinguish between
  - discrete endurants and
  - continuous endurants.
- By a discrete endurant, that is, a part, we shall understand something which is
  - separate or distinct in form or concept,
  - consisting of distinct or separate parts.

4.0.0.7 Continuous Endurant

- By a **continuous endurant**, that is, a **material**, we shall understand an **endurant** whose spatial characteristics are
  - prolonged, without interruption,
  - in an unbroken spatial series or pattern.

4.0.0.8 Domain Parts and Materials

- By a **part** we mean
  - a **discrete endurant**,
  - a **manifest entity** which is fixed in shape and extent.
- By a **material**
  - a **continuous endurant**,
  - a **manifest entity** which typically varies in shape and extent.

4.0.0.9 Domain Analysis

- By **domain analysis** we shall understand an examination of a domain,
  - its **entities**,
  - their possible **composition**,
  - **properties**
  - and **relations** between **entities**,

4.0.0.10 Domain Description

- By a **domain description** we shall understand
  - a **narrative description**
  - tightly coupled (say line-number-by-line-number)
  - to a **formal description**.

4.0.0.11 Domain Engineering

- By **domain engineering** we shall understand
  - the **engineering** of a domain description,
  - that is,
    - the rigorous construction of domain descriptions, and
    - the further analysis of these, creating **theories of domains**[10], etc.

[10] Section (Slides 28–94) is an example of the basis for a theory of road traffic systems.

4.0.0.12 Domain Science

- By domain science we shall understand
  - two things:
    - the general study and knowledge of
      - how to create and handle domain descriptions
      - (a general theory of domain descriptions)

      and
    - the specific study and knowledge of a particular domain.
  - The two studies intertwine.

5. Discrete Endurant Entities
5.1. Parts – Syntactic Aspects
5.1.1. What is a Part ?

- By a **part** we mean an **observable manifest endurant**.

Discussion:

- We use the term 'part' where others use different terms, for example,
  - 'individual',
  - 'particular',
  - 'unit',
  - 'object',
  - 'thing',
  - or other.

Example: 5 Parts.

- Example parts have their types defined in the items as follows:
  - N,
  - F,
  - M,
  - HS,
  - LS,
  - VS,
  - Vs,
  - V,
  - Hs,
  - Ls,
  - H,
  - L.

5.1.2. Classes of "Same Kind" Parts

- We repeat:
  - the **domain describer** does not describe instances of parts,
  - but seeks to describe classes of parts of the same kind.
- Instead of the term 'same kind' we shall use either the terms
  - **part sort** or
  - **part type**.
- By a **same kind class of parts**, that is a **part sort** or **part type** we shall mean
  - a class all of whose members, i.e., **parts**,
  - enjoy "exactly" the same **properties**
  - where a **property** is expressed as a **proposition**.

Example: 6 Part Properties. We continue Example 4.

- Examples of part properties are:
  - *has unique identity*,
  - *has mereology*,
  - *has length*,
  - *has location*,
  - *has traffic movement restriction*,
  - *has position*,
  - *has velocity* and
  - *has acceleration*.

5.1.3. A Preview of Part Properties

- For pragmatic reasons we group endurant properties into two categories:
  - a group which we shall refer to as meta properties:
    - **is discrete**,
    - **is continuous**,
    - **is atomic**,
    - **is composite**,
    - **has observers**,
    - **is sort** and
    - **has concrete type**;
  - and a group which we shall refer to as part properties
    - **has unique existence**,
    - **has mereology** and
    - **has attributes**.
- The first group is treated in this section;
- the second group in the next section.

5.1.4. An Analysis Process: Endurants

- The domain analyser examines collections of parts.
  - In doing so the domain analyser discovers and thus identifies and lists a number of properties.
  - Each of the parts examined usually satisfies only a subset of these properties.
  - The domain analyser now groups parts into collections
    - such that each collection have its parts satisfy the same set of properties,
    - such that no two distinct collections are indexed, as it were, by the same set of properties, and
    - such that all parts are put in some collection.
  - The domain analyser now
    - assigns distinct type names (same as sort names)
    - to distinct collections.
- That is how we assign types to parts.

5.1.5. Part Property Values

- By a part property value, i.e., a property value of a part, we mean
  - the value
  - associated with an intentional property
  - of the part.

Example: 7 Part Property Values.

- A link, l:L, may have the following intentional property values:
  - LOCation value *loc_set*,
  - LENgth value *123 meters* and
  - *mereology* value $\{\kappa_i, \kappa_j\}$.

- Two parts of the same type are different
  - if for at least one of the intentional properties of that part type
  - they have different part property values.

slut

Example: 8 Distinct Parts.

- Two links, l_a,l_b:L, may have the following respective property values:
  - LOCation values *loc_set*$_a$, and *loc_set*$_b$,
  - LENgth value *123 meters* and *123 meters*, i.e., the same, and
  - *mereology* values $\{\kappa_i, \kappa_j\}$ and $\{\kappa_m, \kappa_n\}$ where $\{\kappa_i, \kappa_j\} \neq \{\kappa_m, \kappa_n\}$.
- When so, they are distinct, and the cadestral space *loc_set*$_a$ must not share any point with cadestral space *loc_set*$_b$.

5.1.6. Part Sorts

- By an **abstract type**, or a **sort**, we shall understand a type
  - which has been given a name
  - but is otherwise undefined, that is,
    - is a set of values of further undefined quantities.
      - where these are given properties
      - which we may express in terms of **axiom**s over sort (including **property**) values.

Example: 9 Part Sorts.

- The discovery of N, F and M was made as a result of examining the domain,
- Δ;
- HS and LS;
- Hs and H (Ls and L) ; and
- Vs and V.

5.1.7. Atomic Parts

- By an **atomic part** we mean a part which,
  - in a given context,
  - is deemed *not* to consist of meaningful, separately observable proper **sub-part**s.
- A **sub-part** is a **part**.

Example: 10 Atomic Types.

- We have exemplified the following atomic types:
  - H,
  - L,
  - V and
  - M.
- Implicit tests,
  - at domain indexes,
  - by the domain analyser,
  - for atomicity

  were performed as follows:
  - for H;
  - for L;
  - for V; and
  - for M.

5.1.8. Composite Parts

- By a composite part we mean *a part which*,
  - *in a given context*,
  - *is deemed to indeed consist of meaningful, separately observable proper sub-parts.*

Example: 11 Composite Types.

- We have exemplified the following composite types:
  - N, HS, LS, Hs, Ls, F, VS, Va,

  respectively.

5.1.9. Part Observers

- By a part observer or a material observer we mean
  - a meta-physical operator (a meta function),

    72. **obs_B**: P → B
  - that is, one performed by the domain analyser,
  - which "applies" (i.e., who applies it) to a composite part value[11], P,
  - and which yields the sub-part of type B,
  - of the examined part.

[11] or composite part type

- We name these **obs_**erver functions obs_X to indicate that they are observing parts of type X.
- The **obs_**erver functions are not computable.
  - They can not be mechanised.
  - Therefore we refer to them as mental.
  - They can be "implemented" as, for example, follows:

Example: 12 Implementation of Observer Functions.

- I take you around a particular road net, n, say in my town.
- I point out to you, one-by-one,
  all the street intersections, $h_1, h_2, \ldots, h_n$, of that net.
- You "write" them down:
  - as many characteristics as you (and I) can come across,
    - including some choice of **unique identifiers**,
    - their **mereologies**, and
    - **attributes**, "one-by-one".
- In the end we have identified, i.e., visited,
  all the hubs in my town's road net n. ■

Example: 13 Observer Functions.

- We have exemplified the following obs_erver functions:
  - **obs_**N,
  - **obs_**F,
  - **obs_**M,
  - **obs_**HS,
  - **obs_**LS,
  - **obs_**VS,
  - **obs_**Vs,
  - **obs_**Hs and
  - **obs_**Ls,

5.1.10. Part Types

- By a concrete type we shall understand a type, T,
  - which has been given both a name
  - and a defining type expression of, for example the form
    - T = A-set,
    - T = A-infset,
    - T = A×B×···×C,
    - T = A*,
    - T = A^ω,
    - T = A $\xrightarrow[m]{}$ B,
    - T = A→B,
    - T = A$\xrightarrow{\sim}$B, or
    - T = A|B|···|C.
  - where A, B, ..., C are type names or type expressions.

Example: 14 Concrete Types.

- Example concrete part types were exemplified in
  - Vs = V-set,
  - Hs = H-set,
  - Ls = L-set.

Example: 15 Has Composite Types.

- The discovery of concrete types were done as follows:
  - for HS, Hs = **H-set**,
  - for LS, Ls = **L-set**, and
  - for VS, Vs = **V-set**.

5.2. Part Properties

- We see three categories of part properties:
  - unique identifiers,
  - mereology and
  - (general) attributes.
- Each and every part has unique existence
  — which we model through unique identifiers.
- Parts relate (somehow) to other parts, that is, mereology
  — which we model a relations between unique identifiers.
- And parts usually have other, additional properties
  which we shall refer to as attributes
  — which we model as pairs of attribute types and attribute values.

5.2.1. Unique Identifiers

- Given that we can assume that each and every part, p:P, has unique existence
  - we can postulate the unique identifier observer function:
    - uid_P: P → PI

.

Example: 16 Unique Identifier Functions.

- We have only exemplified the following unique identifier meta-functions and types:
  - **uid**_H, HI,
  - **uid**_L, LI and
  - **uid**_V, VI.
- We did not find a need for defining unique identifier meta-functions for N, F, M, HS, Hs, LS, Ls, VS, and Vs. ■

5.2.1.1 A Dogma of Unique Existence

- We take, as a dogma, that
  - every two parts whose intentional property values differ for at least one property,
  - other than their unique identifiers,
  - are distinct and
  - thus have distinct unique identifiers.

5.2.2. Mereology

- **Mereology:** By mereology (Greek: $\mu\epsilon\rho o\varsigma$) we shall understand the study and knowledge about
  - the theory of *part-hood* relations:
    - of the relations of *part* to *whole* and
    - the relations of *part* to *part* within a *whole*.

- Extensional relations between manifest parts are of the kind:
  - one part, p:P, is *"adjacent to"* (*"physically neighbouring"*) another part, q:Q,
  - one part, p:P, is *"embedded within"* (*"physically surrounded by"*) another part, q:Q, and
  - one part, p:P, *"overlaps with"* another part, q:Q.
- We model these relations, "equivalently", as follows:
  - in the mereology of p, **mereo_**P(p), there is a reference, **uid_**Q(q), to q, and
  - in the mereology of q, **mereo_**Q(q), there is a reference, **uid_**P(p), to p.

- Intentional relations between abstractions are of the kind:
  - part p:P
    - has an attribute
    - whose value
    - always stand in a certain relation
      - (for example, a copy of a fragment or the whole)
  - to another part q:Q's "corresponding" attribute value.

Example: 17 Shared Route Maps and Bus Time Tables. We continue and we extend Example 4.

- The 'Road Transport Domain' of Example 4
  - has its fleet of vehicles be that of a metropolitan city's busses
  - which ply some of the routes according to the city road map (i.e., the net) and
  - according to a bus time table — which we leave undefined.

- We can now re-interpret the road traffic monitor to represent a coordinating bus traffic authority, CBTA.
  - CBTA is now the "new" monitor, i.e., is a part.
  - Two of its attributes are:
    - a metropolitan area road map and
    - a metropolitan area bus time table
  - Vehicles are now busses
    - and each bus
      - follows a route of the metropolitan area road map
      - of which it has a copy, as a vehicle attribute,
      - "shared" with CBTA;
    - each bus additionally
      - runs according to the metropolitan area bus time table
      - of which it has a copy, as a vehicle attribute,
      - "shared" with CBTA.

- We model these attribute value relations, " equivalently", as above:
  - in the mereology of p, **mereo_**P(p), there is a reference, **uid_**Q(q), to q, and
  - in the mereology of q, **mereo_**Q(q), there is a reference, **uid_**P(p), to p.

Example: 18 Monitor and Vehicle Mereologies. We continue Example 17 on Slide 135.

```
73. value  mereo_M: VI-set
74. type   MI
75. value  uid_M: M → MI
76. value  mereo_V: V → MI
```


5.2.2.1 Unique Part Identifier Mereologies

- To express a unique part identifier mereology
  - assumes that the related parts
  - have been endowed, say explicitly,
  - with unique part identifiers.,
  - say of unique identifier types
  - $\Pi_j, \Pi_k, \ldots, \Pi_\ell$.

- A mereology **meta function** is now postulated:

77. **value** **mereo_P**: P → $(\Pi_j \mid \Pi_k \mid \ldots \mid \Pi_\ell)$**-set**,

  - or of some such signature,
  - one which applies to **part**s, **p**:P,
  - and yields **unique identifier**s
  - of other, "the related", parts —
  - where these "other parts" can be of any **part type**,
  - including P.

Example: 19 Road Traffic System Mereology.

- We have exemplified unique part identifier mereologies for
  - hubs, mereo_H Item 8a on Slide 38 and
  - links, mereo_L Item 9a on Slide 38.

5.2.3. Attributes

- **Attribute:** By a part attribute we mean
  - a part property
    - other than part unique identifier and
    - part mereology,
  - and its associated attribute property value.

Example: 20 Road Transport System Part Attributes. We have exemplified, Example 4, a number of part attribute observation functions:

- attr_LΣ,
- attr_LΩ,
- attr_LOC, attr_LEN,
- attr_HΣ,
- attr_HΩ,
- attr_LOC,
- attr_VP, attr_onL, attr_atH, attr_VEL and attr_ACC. ■

5.2.3.1 Stages of Attribute Analysis

- There are four facets to deciding upon part attributes:
  - (i) determining on which attributes to focus;
  - (ii) selecting appropriate attribute type names, (**viz.**, LΣ, LΩ, HΣ, HΩ, LEN, LOC, VP, atH, onL, VEL and ACC);
  - (iii) determining whether an attribute type is
    - a static attribute type (having constant value) (**viz.**, LEN, LOC), or
    - a dynamic attribute type (having variable values)) (**viz.**, LΣ, LΩ, HΣ, HΩ, VP, atH, onL, VEL, ACC);

    and
  - (iv) deciding upon possible concrete type definitions for (some of) those attribute types (**viz.**, LΣ, LΩ, HΣ, HΩ, VP, atH, onL).

Example: 21 Static and Dynamic Attributes. Continuing Example 4 we have:

- Dynamic attributes:
  - LΣ;
  - HΣ;
  - VP, atH, onL; and
  - VEL and ACC.
- All other attributes are considered static. ■

Example: 22 Concrete Attribute Types. From Example 4:

- $L\Sigma = (HI \times HI)$,
- $L\Omega = L\Sigma$**-set**,
- $H\Sigma = (LI \times LI)$**-set** and
- $H\Omega = H\Sigma$**-set**.

5.2.3.2 The attr_A Operator

- To observe a part attribute we therefore describe
  - the attribute observer signature

    78. attr_A: P → A,
  - where P is the part type being examined for attributes, and
  - A is one of the chosen attribute type names.
- The "hunt" for
  - part attributes, i.e., attribute types,
  - the resulting attribute function signatures and
  - the chosen concrete attribute types

  is crucial for achieving successful domain descriptions.

5.3. States

- By a **state** we mean
  - a collection of such **parts**
  - some of whose **part attribute values** are **dynamic**,
  - that is, can vary.

Example: 23 A Variety of Road Traffic Domain States. We continue Example 4.

- A link, l:L, constitutes a state by virtue of if its link traffic state lσ:**attr_**LΣ.
- A hub, h:H, constitutes a state by virtue of its
  - hub traffic state hσ:**attr_**HΣ, and
  - independently, its hub mereology **lis:LI-set:mereo_**H.
- A net, n:N, constitutes a state by virtue of if its link and hub states.
- A monitor, m:M, constitutes a state by virtue of if its vehicle position map vpm:**attr_**VPM.

LECTURE 3

6. Discrete Perdurant Entities

- From Wikipedia:
  - *Perdurant: Also known as occurrent, accident or happening.*
  - *Perdurants are those entities for which only a fragment exists if we look at them at any given snapshot in time.*
  - *When we freeze time we can only see a fragment of the perdurant.*
  - *Perdurants are often what we know as processes, for example 'running'.*
  - *If we freeze time then we only see a fragment of the running, without any previous knowledge one might not even be able to determine the actual process as being a process of running.*
  - *Other examples include an activation, a kiss, or a procedure.*
- A discrete perdurant is a perdurant which is a discrete entity.

- We shall consider the following discrete perdurants.
  - actions (Sect. 6.1),
  - events (Sect. 6.2), and
  - discrete behaviours (Sect. 6.3).
- Actions and events
  - occur instantaneously,
  - that is, in time, but taking no time, and to therefore be
    - discrete actions and
    - discrete events.

6.1. Actions

- By a function we understand a mathematical concept,
  - a thing
  - which when applied to a value, called its argument,
  - yields a value, called its result.
- A discrete action can be understood as
  - a function
  - invoked on a state value
  - and is one that potentially changes that value.
- Other terms for action are
  - function invocation and
  - function application.

Example: 24 Transport Net and Container Vessel Actions.

- *Inserting* and *removing* hubs and links in a net are considered actions.
- *Setting* the traffic signals for a hub (which has such signals) is considered an action.
- *Loading* and *unloading* containers from or unto the top of a container stack are considered actions.

6.1.1. Action Signatures

- By an **action signature** we understand a quadruple:
  - a **function name**,
  - a **function definition set type expression**,
  - a **total** or **partial function** designator ($\rightarrow$, respectively $\xrightarrow{\sim}$), and
  - a **function image set type expression**:

    fct_name: A $\rightarrow$ Σ $(\rightarrow|\xrightarrow{\sim})$ Σ [$\times$ **R**],

  where $(X \mid Y)$ means either X or Y, and $[Z]$ means that for some signatures there may be a **Z** component meaning that the action also has the effect of "leaving" a type **Z** value.

Example: 25 Action Signatures: Nets and Vessels.

insert_Hub: $N \to H \xrightarrow{\sim} N$;
remove_Hub: $N \to HI \xrightarrow{\sim} N$;
set_Hub_Signal: $N \to HI \xrightarrow{\sim} H\Sigma \xrightarrow{\sim} N$
load_Container: $V \to C \to StackId \xrightarrow{\sim} V$; and
unload_Container: $V \to StackId \xrightarrow{\sim} (V \times C)$.

6.1.2. Action Definitions

- There are a number of ways in which to characterise an action.
- One way is to characterise its underlying function by a pair of predicates:
  - precondition: a predicate over function arguments — which includes the state, and
  - postcondition: a predicate over function arguments, a proper argument state and the desired result state.
  - If the precondition holds, i.e., is **true**, then the arguments, including the argument state, forms a proper 'input' to the action.
  - If the postcondition holds, assuming that the precondition held, then the resulting state [and possibly a yielded, additional "result" (R)] is as they would be had the function been applied.

Example: 26 Transport Nets Actions.

- In Example 4 we gave an explicit example of an action:
  - ins_H: Items 37–37d,
- while implicit references to net actions were made in the event predicates
  - link_dis, pre_link_dis: Items 38–39c,
  - post_link_dis (Items 38–39c):
    - rem_L Item 42a and
    - ins_L Items 42(c)i–42(c)ii.

- What is not expressed, but tacitly assume in the above pre- and post-conditions is
  - that the state, here n, satisfy invariant criteria before (i.e. n) and after (i.e., n') actions,
  - whether these be implied by axioms
  - or by well-formedness predicates.

  over parts.
- This remark applies to any definition of actions, events and behaviours.
- There are other ways of defining functions.
- But the form of these are not germane to the aims of these lectures.

Modelling Actions, I/III

- The domain describer has decided that an entity is a perdurant and is, or represents an action: was *"done by an agent and intentionally under some description"* [Davidson1980].
  - The domain describer has further decided that the observed action is of a class of actions — of the "same kind" — that need be described.
  - By actions of the 'same kind' is meant that these can be described by the same function signature and function definition.

Modelling Actions, II/III

- The domain describer must decide on the underlying **function signature**.
  - The **argument type** and the **result type** of the signature are those of either previously identified
    - parts and/or materials,
    - unique part identifiers, and/or
    - attributes.

Modelling Actions, III/III

- Sooner or later the domain describer must decide on the **function definition**.
  - The form must be decided upon.
  - For pre/post-condition forms it appears to be convenient to have developed, "on the side", a **theory of mereology** for the part types involved in the function signature.

6.2. Events

- By an **event** we understand
  - *a state change*
  - *resulting indirectly from an unexpected application of a function,*
  - *that is, that function was performed "surreptitiously".*
- Events can be characterised by a pair of (before and after) states, a predicate over these and, optionally, a **time** or **time interval**.
- Events are thus like actions:
  - change states,
  - but are usually
    - either caused by "previous" actions,
    - or caused by "an outside action".

Example: 27 Events.

- *Container vessel:* A container falls overboard
  sometimes between times t and t'.
- *Financial service industry:* A bank goes bankrupt
  sometimes between times t and t'.
- *Health care:* A patient dies
  sometimes between times t and t'.
- *Pipeline system:* A pipe breaks
  sometimes between times t and t'.
- *Transportation:* A link "disappears"
  sometimes between times t and t'.

6.2.1. Event Signatures

- An event signature
  - *is a predicate signature*
  - *having an event name (evt),*
  - *a pair of state types ($\Sigma \times \Sigma$),*
  - *a total function space operator ($\rightarrow$)*
  - *and a* **Bool***ean type constant:*
  - *evt:* $(\Sigma \times \Sigma) \rightarrow \mathbf{Bool}$.
- Sometimes there may be a good reason
  - for indicating the type, ET, of an event cause value,
  - if such a value can be identified:
  - evt: ET $\times$ $(\Sigma \times \Sigma) \rightarrow \mathbf{Bool}$.

6.2.2. Event Definitions

- An event definition takes the form of
  - *a predicate definition:*
    - *a predicate name and argument list, usually just a state pair,*
    - *an existential quantification*
      - *over some part (of the state) or*
      - *over some dynamic attribute of some part (of the state)*
      - *or combinations of the above*
    - *a pre-condition expression over the input argument(s),*
    - *an implication symbol (⇒), and*
    - *a post-condition expression over the argument(s):*
  - $evt(\sigma,\sigma') = \exists$ (ev:ET) • $pre\_evt(ev)(\sigma) \Rightarrow post\_evt(ev)(\sigma,\sigma')$.

There may be variations to the above form.

Example: 28 Road Transport System Event.

- Example 4,
  - Items 38–42(c)ii
  - (Slides 74–77)

  exemplified an event definition.

Modelling Events I/II

- The **domain describer** has decided that an **entity** is a **perdurant** and is, or represents an **event**: occurred surreptitiously, that is, was not an action that was *"done by an agent and intentionally under some description"* [Davidson1980].
  - The domain describer has further decided that the observed event is of a class of events — of the "same kind" — that need be described.
  - By events of the 'same kind' is meant that these can be described by the same **predicate function signature** and **predicate function definition**.

Modelling Events, II/II

- First the domain describer must decide on the underlying **predicate function signature**.
  - The **argument type** and the **result type** of the signature are those of either previously identified
    - parts,
    - unique part identifiers, or
    - attributes.
- Sooner or later the domain describer must decide on the **predicate function definition**.
  - For predicate function definitions it appears to be convenient to have developed, "on the side", a **theory of mereology** for the part types involved in the function signature.

6.3. Discrete Behaviours

- We shall distinguish between
  - discrete behaviours (this section) and
  - continuous behaviours.
- Roughly discrete behaviours
  - proceed in discrete (time) steps —
  - where, in this lecture, we omit considerations of time.
  - Each step corresponds to an action or an event or a time interval between these.
  - Actions and events may take some (usually inconsiderable time),
  - but the domain analyser has decided that it is not of interest to understand what goes on in the domain during that time (interval).
  - Hence the behaviour is considered discrete.

- Continuous behaviours
  - are **continuous** in the sense of the **calculus** of **mathematical analysis**;
  - to qualify as a **continuous behaviour time** must be an essential aspect of the **behaviour**.
- **Discrete behaviours** can be modelled in many ways, for example using
  - `CSP [Hoare85+2004]`.
  - `MSC [MSCall]`,
  - `Petri Nets [m:petri:wr09]` and
  - `Statechart [Harel87]`.
- We refer to Chaps. 12–14 of `[TheSEBook2wo]`.
- In these lectures we shall use `RSL/CSP`.

6.3.1. What is Meant by 'Behaviour'?

- We give two characterisations of the concept of 'behaviour'.
  - a "loose" one and
  - a "slanted one.
- A loose characterisation runs as follows:
  - by a **behaviour** we understand
    - a set of sequences of
    - **actions**, **events** and **behaviours**.

- A "slanted" characterisation runs as follows:
  - by a **behaviour** we shall understand
    - either a **sequential behaviour** consisting of a possibly infinite sequence of zero or more actions and events;
    - or one or more **communicating behaviours** whose **output actions** of one behaviour may **synchronise** and **communicate** with **input actions** of another behaviour;
    - or two or more **behaviours** acting either as **internal non-deterministic behaviours** ($\sqcap$) or as **external non-deterministic behaviours** ($\Box$).

- This latter characterisation of behaviours
  - is "slanted" in favour of a CSP, i.e., a communicating sequential behaviour, view of behaviours.
  - We could similarly choose to "slant" a behaviour characterisation in favour of
    - Petri Nets, or
    - MSCs, or
    - Statecharts, or other.

6.3.2. Behaviour Narratives

- Behaviour narratives may take many forms.
  - A behaviour may best be seen as composed from several interacting behaviours.
    - Instead of narrating each of these,
    - as was done in Example 4,
    - one may proceed by first narrating the interactions of these behaviours.
  - Or a behaviour may best be seen otherwise,
    - for which, therefore, another style of narration may be called for,
    - one that "traverses the landscape" differently.
  - Narration is an art.
  - Studying narrations – and practice – is a good way to learn effective narration.

6.3.3. Channels

- We remind the listener that we are focusing exclusively on domain behaviours.
  - Domain behaviours, as we shall see in Sect. 5.4.6, take their "root" in **part**s.
  - We shall find, even when "parts" take the form of concepts, that these do not "overlap".
    - They may share properties,
    - but we can consider them "disjoint".
  - Hence communication between processes
    - can be thought of as communication between "disjoint parts",
    - and, as such, can be abstracted as taking place
    - in a non-physical medium which we shall refer to as **channel**s.

- By a **channel** we shall understand
  - *a means of communicating entities*
  - *between [two] behaviours.*
- To express channel communications we, at present, make use of **RSL**'s **out**put (ch ! v) / **in**put (ch ?) clauses and **channel** declarations,

```
type    M
channel ch M,
value   ch!v, ch?,
```

- Variations of the above clauses are

```
type    ChIdx, ChJdx
channel {ch[ i ]|i:ChIdx•𝒫(i,...)}:M, {ch[ i,j ]|i:ChIdx,j:ChJdx•𝒫(i,j,...)}:M
value   ch[ i ]!v, ch[ i ]?, ch[ i,j ]!v, ch[ i,j ]?
```

- where $\mathcal{P}$ is a suitable predicate
  - over channel indices and
  - possibly global domain values.

6.3.4. Behaviour Signatures

- By a **behaviour signature** we shall understand *a*
  - *a function signature*
  - *augmented by a clause which declares*
    - *the* **in** *channels on which the function accepts inputs and*
    - *the* **out** *channels on which the function offers output.*

value behaviour: A → **in** in_chs **out** out_chs B

- where (i)
  - the form **in** in_chs **out** out_chs
    - may be just **in** in_chs
    - or **out** out_chs
    - or both **in** in_chs **out** out_chs

that is, **behaviour** accepts input(s), or offers output(s), or both;

value behaviour: A → **in** in_chs **out** out_chs B

- where (ii)
  - A typically is of the forms
    - **Unit** if the behaviour "takes no arguments",
      - that is: behaviour(),

      or
    - PI×P if the behavior is directly based on a part, p:P, for
      - that is: behaviour(uid_P(p),p);

value behaviour: A → **in** in_chs **out** out_chs B

- where (iii)
  - in_chs and out_chs are of the form either
    - ch or
    - {ch[i]|i:ChIdx·$\mathcal{Q}$(i,...)} or
    - {ch[i,j]|i:ChIdx,j:ChJdx·$\mathcal{R}$(i,j,...)},

    $\mathcal{Q}$, $\mathcal{R}$ are appropriate predicates; and
- where (iv)
  - either
  - B is
    - either just **Unit** when the behaviour is typically a never-ending (i.e., cyclic) behaviours,
    - or is some result type C.

6.3.5. Behaviour Definitions

- This section is about the basic form of **behaviour function definitions**.
  - We shall only be concerned with behaviours which define **part behaviours**.
  - By a **part behaviour** we shall understand
    - *a behaviour whose state*
    - *is that of the part for which it is the behaviour.*
- There are basically two cases for which we are interested in the form of the behaviour definition:
  - the **atomic part behaviour**, and
  - the **composite part behaviour**.

6.3.5.1 Atomic Part Behaviours

- Let p:P be an atomic part of type P.
- Then the basic form of a cyclic atomic behaviour definition is

value
  atomic_core_part_behaviour(uid_P(p))(p) ≡
    let p′ = $\mathcal{A}$(uid_P(p))(p) **in**
    atomic_core_part_behaviour(uid_P(p))(p′) **end**
    post: uid_P(p) = uid_P(p′),

$\mathcal{A}$: PI → P → **in** ... **out** ... P,

- where $\mathcal{A}$ usually is a terminating function
  - which synchronises and
  - communicates with other part behaviours.

Example: 29 Atomic Part Behaviours.

- Example 4 illustrates cyclic atomic behaviours:
  - vehicle at Hub,
  - vehicle on Link and
  - monitor.

6.3.5.2 Composite Part Behaviours

- Let p:P be a composite part of type P.
- Then the basic form of a cyclic atomic behaviour definition for p:P is

value

composite_part_behaviour(uid_P(p))(p) ≡
  composite_core_part_behaviour(uid_P(p))(p)
  ‖ { part_behaviour(uid_P(p′))(p′)|p′:P•p′ ∈ **obs_**(p)}

composite_core_part_behaviour: PI → P → **in** ... **out** ... **Unit**
composite_core_part_behaviour(uid_P(p))(p) ≡
  let p′ = $\mathcal{C}$(uid_P(p))(p) **in**
  composite_core_part_behaviour(uid_P(p))(p′) **end**
  post: uid_P(p) = uid_P(p′)

$\mathcal{C}$: PI → P → **in** ... **out** ... P,

- where $\mathcal{C}$ usually is a terminating function
  - which synchronises and
  - communicates with other part behaviours.

Example: 30 Compositional Behaviours.

- Example 4, Sect. 2.8.3
  - illustrated compositionality,
  - cf. Items 59– 59b on Slide 84.

7. Continuous Perdurants

- We shall not cover continuous
  - actions,
  - event, and
  - behaviours.

LECTURE 3

8. Requirements Engineering

- We shall give a terse overview of some facets of requirements engineering.
  - Namely those which "relate" domain engineering to requirements engineering.
  - The relation is the following:
    - one can "derive",
      - not automatically,
      - but systematically,
    - domain requirements and significant aspects of
    - interface requirements
  - from domain descriptions.

8.1. A Requirements "Derivation"

8.1.1. Definition of Requirements

IEEE Definition of 'Requirements'

- By a requirements we understand (cf. IEEE Standard 610.12 [ieee-610.12]):
  - *"A condition or capability needed by a user to solve a problem or achieve an objective".*

8.1.2. The Machine = Hardware + Software

- By 'the **machine**' we shall understand the
  - **software** to be developed and
  - **hardware** (equipment + base software) to be configured

for the domain application.

8.1.3. Requirements Prescription

- The core part of the requirements engineering of a computing application is the **requirements prescription**.
  - A requirements prescription tells us which parts of the domain are to be supported by 'the machine'.
  - A requirements is to satisfy some **goal**s.
  - Usually the **goal**s cannot be prescribed in such a manner that they can serve directly as a basis for software design.
  - Instead we derive the requirements from the domain descriptions and then argue
    (incl. prove) that the **goal**s satisfy the requirements.
  - In this colloquium we shall not show the latter
    but shall show the former.

8.1.4. Some Requirements Principles

The "Golden Rule" of Requirements Engineering

- Prescribe only such requirements
  - that can be objectively shown to hold
  - for the designed software.

An "Ideal Rule" of Requirements Engineering

- When prescribing (including formalising) requirements,
  - formulate tests (theorems, properties for model checking)
  - whose actualisation show adherence to the requirements.

- We shall not show adherence to the above rules.

8.1.5. A Decomposition of Requirements Prescription

- We consider three forms of requirements prescription:
  - the **domain requirements**,
  - the **interface requirements** and
  - the **machine requirements**.
- Recall that the machine is the hardware and software (to be required).
  - **Domain requirements** are those whose technical terms are from the domain only.
  - **Machine requirements** are those whose technical terms are from the machine only.
  - **Interface requirements** are those whose technical terms are from both.

8.1.6. An Aside on Our Example

- We shall continue our "ongoing" example.
- Our requirements is for a tollway system.
- By a **requirements goal** we mean
  - *an objective*
  - *the system under consideration*
  - *should achieve* [LamsweerdeIEEE2001].
- The **goal**s of having a tollway system are:
  - to decrease transport times
    between selected hubs of a general net; and
  - to decrease traffic accidents and fatalities
    while moving on the tollway net
    as compared to comparable movements on the general net.

- The tollway net, however, must be paid for by its users.
  - Therefore tollway net entries and exits occur at tollway plazas
  - with these plazas containing entry and exit toll collectors
  - where tickets can be issued,
    respectively collected and
    travel paid for.
- We shall very briefly touch upon these toll collectors,
  in the Extension part (as from Slide 209) below.
- So all the other parts of the next section
  serve to build up to the Extension section.

8.2. Domain Requirements

- Domain requirements cover all those aspects of the domain —
  - parts and materials,
  - actions,
  - events and
  - behaviours —

  which are to be supported by 'the machine'.

- Thus domain requirements are developed by systematically "revising" cum "editing" the domain description:
  - which parts are to be **projected:** left in or out;
  - which general descriptions are to be **instantiated** into more specific ones;
  - which non-deterministic properties are to be made more **determinate**; and
  - which parts are to be **extended** with such computable domain description parts which are not feasible without IT.

- Thus
  - projection,
  - instantiation,
  - determination and
  - extension

  are basic engineering tasks of domain requirements engineering.

- An example may best illustrate what is at stake.
- The example is that of a tollway system —
  - in contrast to the general nets.
  - See Fig. 1 on the next slide.

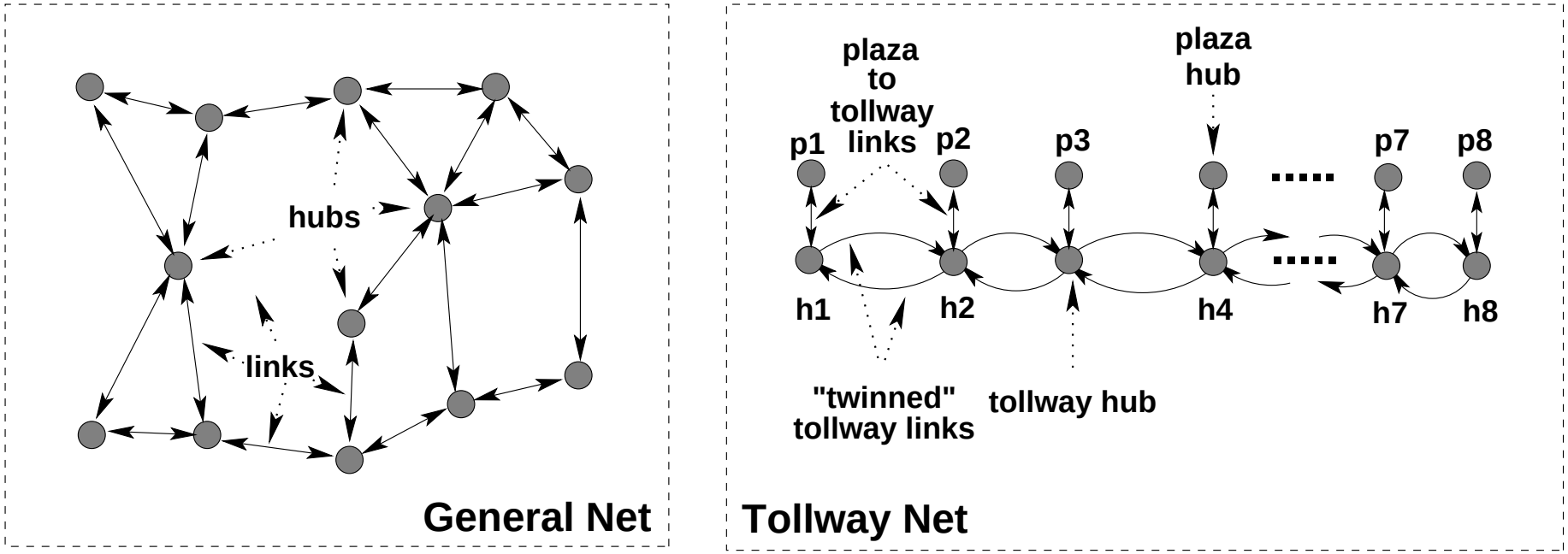

Figure 1: General and Tollway Nets

8.2.1. Projection

- By **domain projection** we mean that a subset of the **domain description** is kept.
- In the tollway example we actually keep all the parts, their properties and therefore the types and functions derived from these,
- Thus we keep:
  - 1a–1c (N, F, M)
  - 2–2b (HS, LS),
  - 5a–6b (Hs, Ls, H, L),
  - 7a–7b (HI, LI),
  - 10a–10c (LΣ, LΩ, LEN, LOC) and
  - 11a–11c (HΣ, HΩ, LOC) ,
  - 3–4b, 7c (VS, Vs, V),
  - 8a–9b (**mereo**_L),
  - 12a–12(a)iii, 13 (VP, atH, onL, FRAC, **attr**_VP),
- We do not keep any actions or events (!),
- But we keep the behaviours:
  - 59–59b (trs),
  - 61–63 (trs, veh, mon),
  - 65–65d, 64–68 (veh),
  - 69–71d (mon).

8.2.2. Instantiation

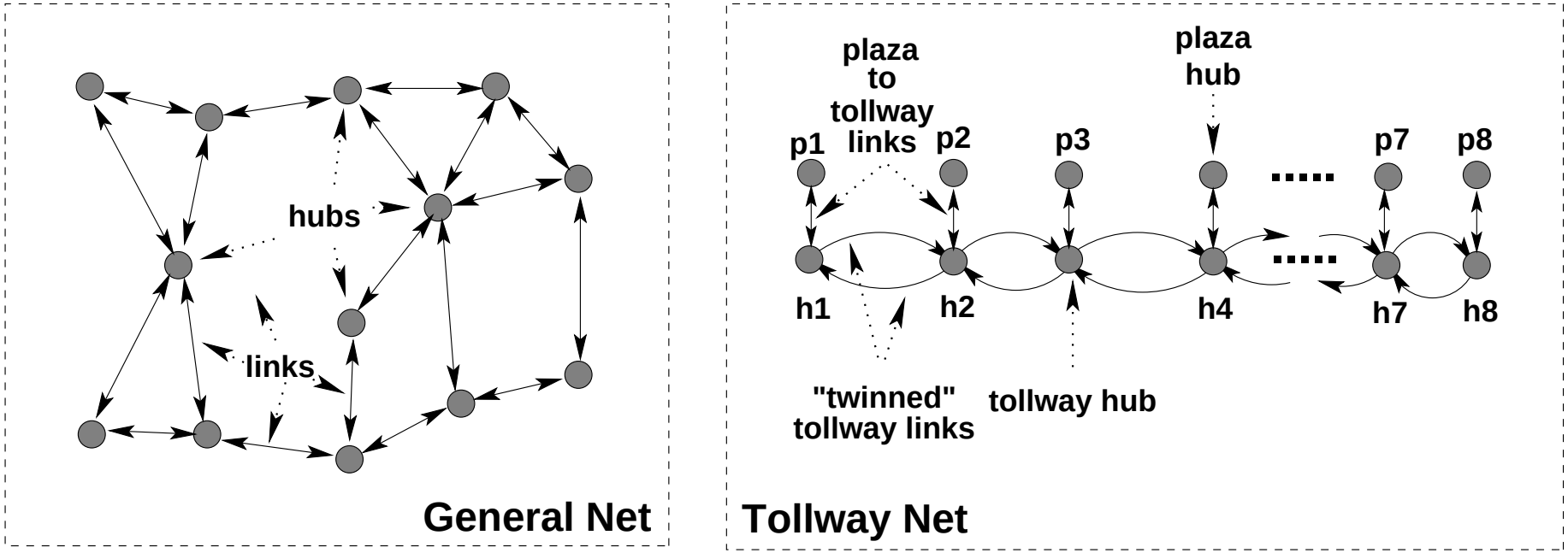

Figure 2: General and Tollway Nets

- From the general net model of earlier formalisations we instantiate, that is, make more concrete, the tollway net model now described.

79. The net is now concretely modelled as a pair of sequences.
80. One sequence models the plaza hubs, their plaza-to-tollway link and the connected tollway hub.
81. The other sequence models the pairs of "twinned" tollway links.
82. From plaza hubs one can observe their hubs and the identifiers of these hubs.
83. The former sequence is of m such plaza "complexes" where $m \geq 2$; the latter sequence is of $m - 1$ "twinned" links.
84. From a tollway net one can abstract a proper net.

```
type
79. TWN = PC* × TL*
80. PC = PH × L × H
81. TL = L × L
value
80. obs_H: PH → H, obs_HI: PH → HI
axiom
83. ∀ (pcl,tll):TWN •
83.   2≤len pcl∧len pcl=len tll+1
value
84. abs_HsLs: TWN → (Hs×Ls)
84. abs_HsLs(pcl,tll) as (hs,ls)
84.   pre: wf_TWN(pcl,tll)
84.   post:
84.     hs = {h,h′|(h,_,h′):PC • (h,_,h′)∈ elems pcl}
84.   ∧ ls = {l|(_,l,_):PC • (_,l,_)∈ elems pcl} ∪
84.            {l,l′|(l,l′):TL•(l,l′)∈ elems tll}
```

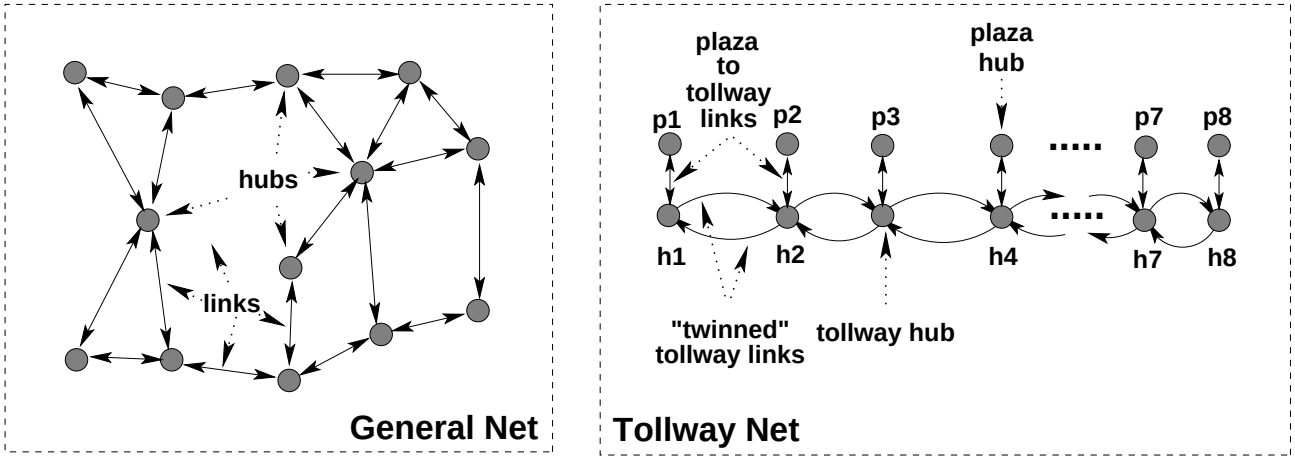

Figure 3: General and tollway Nets

8.2.2.1 Model Well-formedness wrt. Instantiation

- Instantiation restricts general nets to tollway nets.
- Well-formedness deals with proper mereology:
  that observed identifier references are proper.
- The well-formedness of instantiation of the tollway system model can be defined as follows:

85. The i'plaza complex, (p_i, l_i, h_i), is instantiation-well-formed if
    [a] link l_i identifies hubs p_i and h_i, and
    [b] hub p_i and hub h_i both identifies link l_i; and if
86. the i'th pair of twinned links, tl_i, tl'_i,
    [a] has these links identify the tollway hubs of the i'th and $i+1$'st plaza complexes ((p_i, l_i, h_i) respectively $(p_{i+1}, l_{i+1}, h_{i_1})$).


```
value
  Instantiation_wf_TWN: TWN → Bool
  Instantiation_wf_TWN(pcl,tll) ≡
85.   ∀ i:Nat · i ∈ inds pcl⇒
85.     let (pi,li,hi)=pcl(i) in
85a.      obs_LIs(li)={obs_HI(pi),obs_HI(hi)}
85b.    ∧ obs_LI(li)∈ obs_LIs(pi)∩ obs_LIs(hi)
86.     ∧ let (li′,li″) = tll(i) in
86.       i < len pcl ⇒
86.         let (pi′,li‴,hi′) = pcl(i+1) in
86a.          obs_HIs(li) = obs_HIs(li′)
86a.          = {obs_HI(hi),obs_HI(hi′)}
    end end end
```

8.2.3. Determination

- By **domain determination** we mean, as illustrated in this example,
  - making **part property values**
  - less **in-determinate**, i.e.,
  - more **determinate**.
- The state sets contain only one set.
  - Twinned tollway links allow traffic only in opposite directions.
  - Plaza to tollway hubs allow traffic in both directions.
  - tollway hubs allow traffic to flow freely from
    - plaza to tollway links
    - and from incoming tollway links
    - to outgoing tollway links
    - and tollway to plaza links.

8.2.3.1 Model Well-formedness wrt. Determination

- We need define well-formedness wrt. determination.
- Please study Fig. 4.

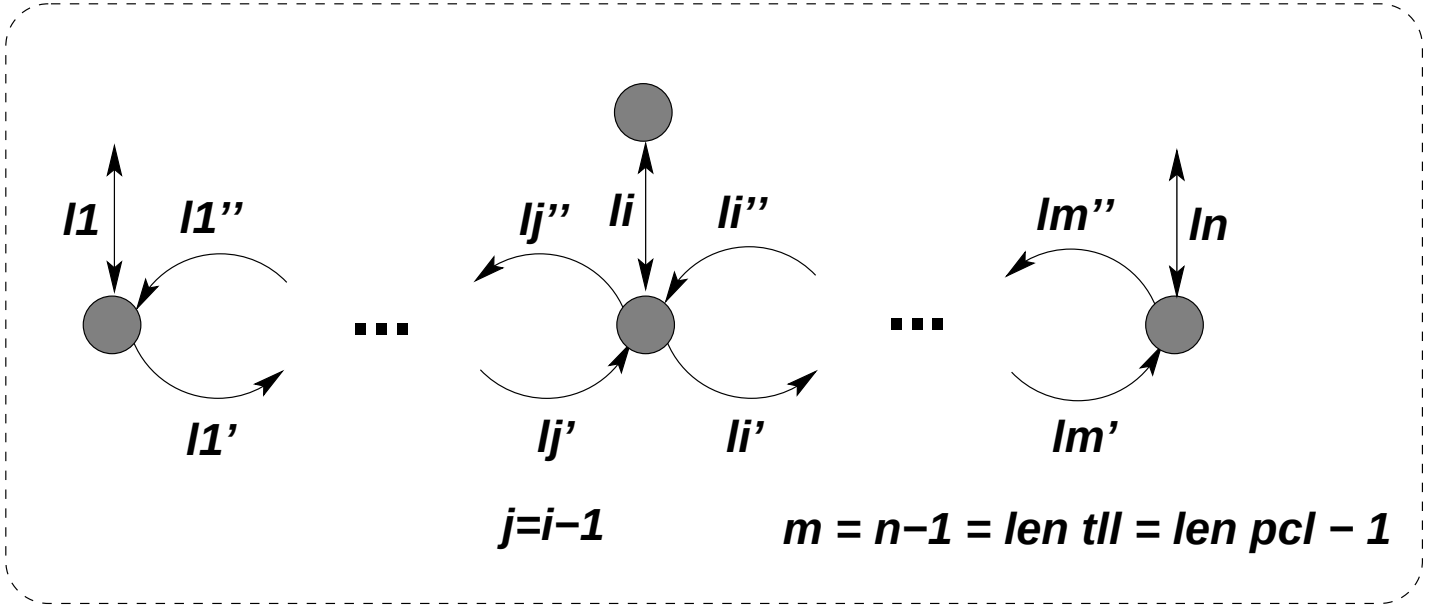

Figure 4: Hubs and Links

87. All hub and link state spaces contain just one hub, respectively link state.
88. The i'th plaza complex, pcl(i):(p_i, l_i, h_i) is determination-well-formed if
    [a] l_i is open for traffic in both directions and
    [b] p_i allows traffic from h_i to "revert"; and if
89. the i'th pair of twinned links (li', li'') (in the context of the i+1st plaza complex, pcl(i+1):$(p_{i+1}, l_{i+1}, h_{i+1})$) are determination-well-formed if
    [a] link l'_i is open only from h_i to h_{i+1} and
    [b] link l''_i is open only from h_{i+1} to h_i; and if
90. the jth tollway hub, h_j (for $1 \leq j \leq \mathbf{len}$ pcl) is determination-well-formed if, depending on whether j is the first, or the last, or any "in-between" plaza complex positions,
    [a] [the first:] hub $i = 1$ allows traffic in from l_1 and l''_1, and onto l_1 and l'_1.
    [b] [the last:] hub $j = i + 1 = \mathbf{len}$ pcl allows traffic in from $l_{\mathbf{len}\,\mathsf{tll}}$ and $l''_{\mathbf{len}\,\mathsf{tll}-1}$, and onto $l_{\mathbf{len}\,\mathsf{tll}}$ and $l'_{\mathbf{len}\,\mathsf{tll}-1}$.
    [c] [in-between:] hub $j = i$ allows traffic in from l_i, l''_i and l'_i and onto l_i, l'_{i-1} and l''_i.

```
value
88. Determination_wf_TWN: TWN → Bool
88. Determination_wf_TWN(pcl,tll) ≡
88.   ∀ i:Nat• i ∈ inds tll ⇒
88.     let (pi,li,hi) = pcl(i),
88.         (npi,nli,nhi) = pcl(i+1), in
88.         (li′,li″) = tll(i) in
87.       obs_HΩ(pi)={obs_HΣ(pi)}∧obs_HΩ(hi)={obs_HΣ(hi)}
87.     ∧ obs_LΩ(li)={obs_LΣ(li)}∧obs_LΩ(li′)={obs_LΣ(li′)}
87.     ∧ obs_LΩ(li″)={obs_LΣ(li″)}
88a.    ∧ obs_LΣ(li)
88a.         = {(obs_HI(pi),obs_HI(hi)),(obs_HI(hi),obs_HI(pi))}
88a.    ∧ obs_LΣ(nli)
88a.         = {(obs_HI(npi),obs_HI(nhi)),(obs_HI(nhi),obs_HI(npi))}
88b.    ∧ {(obs_LI(li),obs_LI(li))}⊆obs_HΣ(pi)
88b.    ∧ {(obs_LI(nli),obs_LI(nli))}⊆obs_HΣ(npi)
89a.    ∧ obs_LΣ(li′)={(obs_HI(hi),obs_HI(nhi))}
89b.    ∧ obs_LΣ(li″)={(obs_HI(nhi),obs_HI(hi))}
90.     ∧ case i+1 of
90a.       2 → obs_HΣ(h_1)=
90a.             {(obs_LΣ(l_1),obs_LΣ(l_1)), (obs_LΣ(l_1),obs_LΣ(l_1″)),
90a.             (obs_LΣ(l″_1),obs_LΣ(l_1)), (obs_LΣ(l″_1),obs_LΣ(l′_1))},
90b.       len pcl → obs_HΣ(h_i+1)=
90b.            {(obs_LΣ(l_len pcl),obs_LΣ(l_len pcl)),
90b.             (obs_LΣ(l_len pcl),obs_LΣ(l′_len tll)),
90b.             (obs_LΣ(l″_len tll),obs_LΣ(l_len pcl)),
90b.             (obs_LΣ(l″_len tll),obs_LΣ(l′_len tll))},
90c.       _ → obs_HΣ(h_i)=
90c.             {(obs_LΣ(l_i),obs_LΣ(l_i)), (obs_LΣ(l_i),obs_LΣ(l′_i)),
90c.             (obs_LΣ(l_i),obs_LΣ(l″_i−1)), (obs_LΣ(l″_i),obs_LΣ(l′_i)),
90c.             (obs_LΣ(l″_i),obs_LΣ(l′_i−1)), (obs_LΣ(l″_i),obs_LΣ(l′_i))}
88.    end end
```

8.2.4. Extension

- By domain extension we understand the
  - *introduction of domain entities, actions, events and behaviours that were not feasible in the original domain,*
  - *but for which, with computing and communication,*
  - *there is the possibility of feasible implementations,*
  - *and such that what is introduced become part of the emerging domain requirements prescription.*

8.2.4.1 Backgorund

- The road traffic monitoring domain of Example 4,
  - notably the sections on vehicle and monitor behaviours, (Items 65–71d Slides 89–93),
  - illustrated the intangible abstraction of road traffic
  - in the form of the recording of a discrete version of that traffic:[12]

46. $\mathbb{dT}$
45. $\mathbb{dRTF} = \mathbb{dT} \overrightarrow{m} (\mathbb{VI} \overrightarrow{m} \mathbb{VP})$

- by the road traffic system:

value

59. trs() =
59a. ‖ {veh(**uid**_V(v))(v)(vpm(**uid**_V(v)))|v:V·v ∈ vs}
59b. ‖ mon(mi)(m)([t_0 ↦ vpm])

[12] In dRTF we change V into a reference to vehicles VI.

- We say that the **road traffic, dRTF** is **intangible**
  - since the **dRTF** function,
  - being a function, is an intangible.
- The **domain extension** is now making that "function" a **tangible** notion.
- There is no presumption,
  - in defining the **mon**itor behaviour,
  - that there is indeed a mechanised behaviour,
  - i.e., a computerised process
  - that "implements" that **mon**itor.
- Since
  - one can speak of the **mon**itor behaviour,
  - one can, as well define it.

8.2.4.2 The Extension

- We now "implement" a version of the above monitor behaviour.
- The proposed domain extension builds upon
  - the monitor
  - and the ability of vehicles
    - to communicate their vehicle positions
    - to the monitor, cf.
      - Items 65a and 65a Slide 90,
      - Items 66a, 66(c)i and 66(c)iiA Slide 91 and
      - Item 71a Slide 94.

- Instead of this "directness"
  - we interpret links and hubs of the tollway system
  - as behaviours
  - endowed with sensors.
- Vehicle behaviours now interact with link and hub behaviours
  - communicating their positions
  - which the link and hub behaviours
  - communicate to a tollway system monitor.
- The **domain extension** then consists of
  - the extension of links and hubs with sensors and
  - the modelling of their vehicle interactions and
  - their interaction with the tollway system monitor.

8.2.4.3 The Formalisation

- We introduce
  91. rather simple **link** and **hub** behaviours, and
  92. an array of channels for the interaction of **vehicle** behaviours with **link** and **hub** behaviours.
- And we modify
  93. the **vehicle** and **monitor** behaviours and
  94. the **vehicle**/**monitor** **channel**
- the latter to now serve at the **channel**
- for **link** and **hub** interactions
- with the refined **monitor** behaviour.

```
value
84.  (hs,ls):(Hs×Ls) = abs_HsLs(twn)
22.    his:HI-set = {uid_H(h)|h:H·h ∈ hs}
21.    lis:LI-set = {uid_L(l)|l:L·l ∈ ls}
channel
92.  {vlh_ch[ vi,si ]|vi:VI,si:(LI|HI)·vi ∈ vis∧si ∈ lis ∪ his}:VP
94.  {lhm_ch[ si,mi ]|si:(LI|HI)·si ∈ lis ∪ his}:(VI×VP)
value
92.  link: li:LI → L → in { vlh_ch[ vi,si ]|si:LI·si ∈ lis }  Unit
92.  hub: hi:HI → H → in { vlh_ch[ vi,si ]|si:HI·si ∈ his }  Unit
91.  link(li)(l) ≡
91.    (...⌈⌉ [] {let (vi,vp) = vlh_ch[ vi,li ]? in lhm_ch[ li,mi ]!(vi,vp)|vi:VI·vi ∈ vis end});linl
91.  hub(hi)(h) ≡
91.    (...⌈⌉ [] {let (vi,vp) = vlh_ch[ vi,hi ]? in lhm_ch[ hi,mi ]!(vi,vp)|vi:VI·vi ∈ vis end});hu
59.    trs() =
59a.     ‖ {veh(uid_V(v))(v)(vpm(uid_V(v)))|v:V·v ∈ vs}
59b.     ‖ mon(mi)(m)([ t0 ↦ vpm ])
91.      ‖ {link(uid_L(l))(l)|l:L·l ∈ ls}
91.      ‖ {hub(uid_H(h))(h)|h:H·h ∈ hs}
```

- The modifications to the **veh**icle behaviour is shown in
  - Items 65a′, 65(b)ii′, 66a′, 66(c)i′, 66(c)iiA′ and 71a′
  - (Slides 216–217).

65. veh(vi)(v)(vp:atH(fli,hi,tli)) ≡
65a′. **vlh_ch[vi,hi]**!(vi,vp) ; veh(vi)(v)(vp)
65b. ⌈⌉
65(b)i. **let** {hi′,thi}=**mereo**_L(get_L(tli)(n)) **in assert:** hi′=hi
65(b)ii′. **vlh_ch[vi,tli]**!(vi,onL(hi,tli,0,thi)) ;
65(b)iii. veh(vi)(v)(onL(hi,tli,0,thi)) **end**
65c. ⌈⌉
65d. **stop**

64. veh(vi)(v)(vp:onL(fhi,li,f,thi)) ≡
66a′. **vlh_ch[vi,li]**!(vi,vp) ; veh(vi)(v)(vp)
66b. ⌈⌉
66c. **if** f + δ<1
66(c)i′. **then vlh_ch[vi,li]**!(vi,onL(fhi,li,f+δ,thi)) ;
66(c)i. veh(vi)(v)(onL(fhi,li,f+δ,thi))
66(c)ii. **else let** li′:LI·li′ ∈ **mereo_**H(get_H(thi)(n)) **in**
66(c)iiA′. **vlh_ch[vi,thi]**!(vi,atH(li,thi,li′)) ;
66(c)iiB. veh(vi)(v)(atH(li,thi,li′)) **end end**
67. ⌈⌉
68. **stop**

69. mon(mi)(m)(rtf) ≡
70. mon(mi)(own_mon_work(m))(rtf)
71. ⌈⌉
71a′. [] { **let** ((vi,vp),t) = (**lhm_ch[si,mi]?**,clk_ch?) **in**
71b. **let** rtf′ = rtf † [t ↦ rtf(max **dom** rtf) † [vi ↦ vp]] **in**
71c. mon(mi)(m)(rtf′) **end**
71d. **end** | si:(LI|HI) · si ∈ lis ∪ his}

- The extension, in this example, does not really amount to much.
  - We say that we have extended links and hubs with sensors.
  - But we have not really modelled these sensors.
  - We have modelled their intent, but not their extent.
  - A more complete extension,
    - which has to be done, but which is not shown in these lectures,
    - would now model these sensors
    - as they rely on the **unique vehicle identifier** to be sensed.

- We shall, regrettably, omit this aspect of our presentation of the extension.
  - Whichever sensor technology is chosen, it must be described.
  - A description includes both it proper and its erroneous functioning.
  - Such (IT equipment &c.) descriptions may be expressed in a number of steps:
    - First, as here, a `RSL/CSP` `[CARH:Electronic,TheSEBook1wo]`. model.
    - Then a "derived" description models temporal properties using `Duration Calculus, DC [zcc+mrh2002]`, or `Temporal Logic of Actions,` **TLA+**.
    - Finally a timed-automata `[AluDil:94,olderogdirks2008]` model which "implements" the `DC` model.

8.3. Interface Requirements Prescription

- A systematic reading of the domain requirements shall
  - result in an identification of all shared
    - parts and materials,
    - actions,
    - events and
    - behaviours.
- An entity is said to be a **shared entity**
  - if it is present
  - in some related forms,
  - in both
    - the **domain** and
    - the **machine**.

- Each such **shared phenomenon** shall then be individually dealt with:
  - **part** and **materials sharing** shall lead to interface requirements for **data initialisation and refreshment;**
  - **action sharing** shall lead to interface requirements for **interactive dialogues between the machine and its environment;**
  - **event sharing** shall lead to interface requirements for **how events are communicated between the environment of the machine and the machine.**
  - **behaviour sharing** shall lead to interface requirements for **action and event dialogues between the machine and its environment.**

8.3.1. Shared Parts

- The main **shared parts** of the main example of this section are
  - the net, hence the hubs and the links.
- As domain parts they repeatedly undergo changes with respect to the values of a great number of attributes and otherwise possess attributes — most of which have not been mentioned so far:
  - length, cadestral information, namings,
  - wear and tear (where-ever applicable),
  - last/next scheduled maintenance (where-ever applicable),
  - state and state space, and
  - many others.

- We "split" our interface requirements development into two separate steps:
  - the development of $d_{r.net}$
    - (the common domain requirements for the shared hubs and links),
  - and the co-development of $d_{r.db:i/f}$
    - (the common domain requirements for the interface between $d_{r.net}$ and DB_{rel} —
  - under the assumption of an available relational database system DB_{rel}

- When planning the common domain requirements for the net, i.e., the hubs and links,
  - we enlarge our scope of requirements concerns beyond the two so far treated ($d_{r.toll}$, $d_{r.maint}$.)
  - in order to make sure that the shared relational database of nets, their hubs and links, may be useful beyond those requirements.

- We then come up with something like
  - hubs and links are to be represented as tuples of relations;
  - each net will be represented by a pair of relations
    - a hubs relation and a links relation;
    - each hub and each link may or will be represented by several tuples;
  - etcetera.
- In this database modelling effort it must be secured that "standard" actions on nets, hubs and links can be supported by the chosen relational database system DB_{rel}.

8.3.1.1 Data Initialisation

- As part of $d_{r.net}$ one must prescribe **data initialisation**, that is provision for
  - an interactive user interface dialogue with a set of proper display screens,
    - one for establishing net, hub or link attributes (names) and their types and,
    - for example, two for the input of hub and link attribute values.
  - Interaction prompts may be prescribed:
    - next input,
    - on-line vetting and
    - display of evolving net, etc.
  - These and many other aspects may therefore need prescriptions.
- Essentially these prescriptions concretise the insert link action.

8.3.1.2 Data Refreshment

- As part of $d_{r.net}$ one must also prescribe **data refreshment**:
  - an interactive user interface dialogue with a set of proper display screens
    - one for updating net, hub or link attributes (names) and their types and,
    - for example, two for the update of hub and link attribute values.
  - Interaction prompts may be prescribed:
    - next update,
    - on-line vetting and
    - display of revised net, etc.
  - These and many other aspects may therefore need prescriptions.
- These prescriptions concretise remove and insert link actions.

8.3.2. Shared Actions

- The main **shared actions** are related to
  - the entry of a vehicle into the tollway system and
  - the exit of a vehicle from the tollway system.

8.3.2.1 Interactive Action Execution

- As part of $d_{r.toll}$ we must therefore prescribe
  - the varieties of successful and less successful sequences
  - of interactions between vehicles (or their drivers) and the toll gate machines.
- The prescription of the above necessitates determination of a number of external events, see below.
- (Again, this is an area of embedded, real-time safety-critical system prescription.)

8.3.3. Shared Events

- The main **shared external events** are related to
  - the entry of a vehicle into the tollway system,
  - the crossing of a vehicle through a tollway hub and
  - the exit of a vehicle from the tollway system.
- As part of $d_{r.toll}$ we must therefore prescribe
  - the varieties of these events,
  - the failure of all appropriate sensors and
  - the failure of related controllers:
    - gate opener and closer (with sensors and actuators),
    - ticket "emitter" and "reader" (with sensors and actuators),
    - etcetera.
- The prescription of the above necessitates extensive fault analysis.

8.3.4. Shared Behaviours

- The main **shared behaviours** are therefore related to
  - the journey of a vehicle through the tollway system and
  - the functioning of a toll gate machine during "its lifetime".
- Others can be thought of, but are omitted here.
- In consequence of considering, for example, the journey of a vehicle behaviour, we may "add" some further, extended requirements:
  - requirements for a vehicle statistics "package";
  - requirements for tracing supposedly "lost" vehicles;
  - requirements limiting tollway system access in case of traffic congestion; etcetera.

8.4. Machine Requirements

- The machine requirements
  - make hardly any concrete reference to the domain description;
  - so we omit its treatment altogether.

8.5. Discussion of Requirements "Derivation"

- We have indicated
  - how the **domain engineer**
  - and the **requirements engineer**
  - can work together
  - to "derive" significant fragments
  - of a **requirements prescription**.

- This puts **requirements engineering** in a new light.
  - Without a previously existing **domain descriptions**
  - the **requirements engineer** has to do double work:
    - both **domain engineering**
    - and **requirements engineering**
  - but without the **principles** of **domain description**,
    - as laid down in these lectures
  - that job would not be so straightforward as we now suggest.

9. Conclusion

9.1. What Have We Achieved

- We claim that there are four major contributions having been lectured upon:
  - the separation of **domain engineering** from **requirements engineering**,
  - the separate treatment of **domain science & engineering**:
    - as "free-standing" with respect, ultimately, to computer science,
    - and endowed with quite a number of **domain analysis principles** and **domain description principles**; and

- the identification of a number of techniques
  - for "deriving" significant fragments of **requirements prescriptions** from **domain descriptions** —
  - where we consider this whole relation between **domain engineering** and **requirements engineering** to be novel.

- Yes, we really do consider the possibility of a systematic
  - 'derivation' of significant fragments of **requirements prescriptions** from **domain descriptions**
  - to cast a different light on **requirements engineering**.

- What we have not shown in these lectures is
  - the concept of **domain facets**;
  - this concept is dealt with in `[dines:facs:2008]` —
  - but more work has to be done to give a firm theoretical understanding of **domain facets** of
    - **domain intrinsics**,
    - **domain support technology**,
    - **domain scripts**,
    - **domain rules and regulations**,
    - **domain management and organisation**, and
    - **human domain**behaviour.
- Facets will be illustrated in my keynote tomorrow !

9.2. General Remarks

- Perhaps belaboring the point:
  - one can pursue creating and studying domain descriptions
  - without subsequently aiming at requirements development,
  - let alone software design.
- That is, domain descriptions
  - can be seen as
    - "free-standing",
    - of their "own right",
    - useful in simply just understanding
    - domains in which humans act.

- Just like it is deemed useful
  - that we study "Mother Nature",
  - the physical world around us,
  - given before humans "arrived";
- so we think that
  - there should be concerted efforts to study and create **domain models**,
  - for use in
    - studying "our man-made domains of discourses";
    - possibly proving laws about these domains;
    - teaching, from early on, in middle-school, the domains in which the middle-school students are to be surrounded by;
    - etcetera

- How far must one formalise such **domain descriptions** ?
  - Well, enough, so that possible laws can be mathematically proved.
  - Recall that **domain descriptions** usually will or must be developed by **domain researchers** — not necessarily **domain engineers** —
    - in research centres, say universities,
    - where one also studies physics.

- And, when we base requirements development on domain descriptions,
  - as we indeed advocate,
  - then the requirements engineers
  - must understand the formal domain descriptions,
  - that is, be able to perform formal
    - domain projection,
    - domain instantiation,
    - domain determination,
    - domain extension,

    etcetera.

- This is similar to the situation in classical engineering
  - which rely on the sciences of physics,
  - and where, for example,
    - *Bernoulli's equations*,
    - *Maxwell's equations*,
    - *Navier-Stokes equations*,
    - etcetera
  - were developed by physicists and mathematicians,
  - but are used, daily, by engineers:
    - read and understood,
    - massaged into further differential equations, etcetera,
    - in order to calculate (predict, determine values), etc.

- Nobody would hire non-skilled labour
  - for the engineering development of airplane designs
    - unless that "labourer" was skilled in *Navier-Stokes equations*,

    or
  - for the design of mobile telephony transmission towers
    - unless that person was skilled in *Maxwell's equations*.

- So we must expect a future, we predict,
  - where a subset of the software engineering candidates from universities
    - are highly skilled in the development of
      - formal **domain descriptions**
      - formal **requirements prescriptions**
  - in at least one domain, such as
    - *transportation*, for example,
      - air traffic,
      - railway systems,
      - road traffic and
      - shipping;

      or
    - *manufacturing*,
    - *services* (health care, public administration, etc.),
    - *financial industries*, or the like.

9.3. Acknowledgements

- I thank
  - the organisers of ASSC
    and especially Dr. Clive V. Boughton
    for having invited me to give this tutorial,
  - and I thank you, the audience, for having behaved nicely !

Questions ?