

---

## Lecture 4: Describing Domain Entities. Part I: Parts, Actions, Events

---

## 4. Describing Domain Entities

### 4.1. On Describing

- The purpose of description is
  - to use for example informal text
  - to present an entity (simple, action, event or behaviour)
  - so that the reader may
    - \* “picture” (“envisage”),  
that which is being described.
- The text describing the entity
  - is said to be a syntactic quantity.
  - and the entity is then said to be a semantic quantity:
  - the syntactic text *denotes* the semantic quantity.

- We also say that the syntactic quantity
    - designates,                      – specifies,                      – or title to, or
    - denotes,                         – points out,                     – characterises<sup>13</sup>
    - indicates,                        – gives a name
- the semantic quantity.

### 4.1.1. Informal Descriptions

- In the many examples<sup>14</sup> of Parts 2–3 we have made several references to quite a few domain entities.
- We do not claim that we have described these entities.

---

<sup>13</sup>— eight alternative terms for the same idea!

<sup>14</sup>Examples 5–23

### 4.1.1.1. Domain Instances Versus Domains

- What we can observe are instances of
  - a specific domain or
  - fragments (perhaps parts) of a specific domain.
- What we describe are
  - either abstractions of these instances
  - or abstractions of a set (i.e., a type) of these instance.

- If someone describes me
  - as an atomic part
  - with the action(s) and event(s) of my behaviour,then
  - that someone describes an instance of a person,
  - not the domain of all persons,
  - but in that description it is expected that *many fragments* of the description is also valid for
    - \* either a lot of persons
    - \* or all persons.
- We say that these *many fragments*
  - describe not an instances but fragments of abstractions
  - of a domain of persons.

### 4.1.1.2. Non-uniqueness of Domain Descriptions

- We say ‘a domain’, not ‘the domain’.
- Two or more domain describers may not exactly focus on the same entities and their properties.
- A domain description is always an abstraction.
  - Something is left out.
  - Not all entities and
    - not all properties of those entities included
    - may be deemed worthwhile to be included.
- A good domain description, to us, is a domain description
  - that covers what most stake holders can agree on to be relevant aspects of the domain,
  - that reveals generally unknown facets of the domain, and
  - that is terse and precise.

### 4.1.1.3. A Criterion for Description

- For us, to informally describe an entity ideally means the following:
  - *Let there be given what we can agree on to be an entity, call it  $e$ .*
  - *Let there be given what is claimed to be a description of that entity.*
  - *Let a person read and claim to have understood that description.*
  - *Now that person is confronted with some phenomenon  $e'$ .*
  - *Either that phenomenon is the same or it is of the same kind (type) as  $e$  or it is not.*
  - *If  $e'$  is of the same kind as  $e$  then the person must identify it as such, unequivocally.*
  - *If  $e'$  is not of the same kind as  $e$  then the person must identify it as not being so, likewise unequivocally.*

- If a description does not satisfy the above then it is not a *proper description*.
- The above “criterion” suffers, seriously, from our not having made precise what we mean by “*same*” and “*same kind*”.
  - These notes are not the place for a much needed investigation of the “sameness” problem.
  - It is basically a philosophical question.
  - But we should not overlook the fact that it is the domain describer and the domain stake holders who, finally, decide on “sameness”.



### 4.1.1.4. Reason for 'Description' Failure

- There can be three reasons for a description to not be proper:
  1. *either all phenomena are entities as described — the description is vacuous;*
  2. *or there are entities which were meant to be of the type or not meant to be of the type described but which*
    - “fall outside”, respectively
    - “fall inside”*the description;*
  3. *or the description does not make sense, is “gibberish”, ambiguous, or otherwise.*
- That is: a proper description, when applied to entities, “divides” their world into two non-empty and disjoint sets:
  - the set of all entities being described by the description,
  - and the rest !

### 4.1.1.5. Failure of Description Language

- But we have a problem !
  - One cannot give a precise definition of exactly the *denoting language*, that is,
    - \* of exactly, all and only those informal texts
    - \* which designate entities.
  - Firstly,
    - \* we have not given a sufficiently precise informal text characterisation of entities,
  - Secondly,
    - \* natural (cum national) languages, like English, defy such characterisations.
- We must do our best with informal language descriptions.

## 4.1.1.6. Guidance

- But there is help to be gotten!
- The whole purpose of Part 3 was to establish the pointers, i.e., guidelines, as to what must be described, generally:
  - parts, actions, events and behaviours,
  - and specifically:
    - \* whether atomic or composite parts,
    - \* their attributes, and, optionally, their mereology,
    - \* and, for composite parts, their subparts;
    - \* and, as a starter, the signatures of actions, events and behaviours.
- This part will continue the line
  - reviewed just above
  - and provide further hints, pointers, guidelines.

## 4.1.2. Formal Descriptions

- We shall,
  - in addition to the *description components*<sup>15</sup>, outlined in Part 3
  - now join the possibility of
    - \* improved description precision
    - \* through the use of formal description.
- We argue that formal description,
  - while being used in-separately with precise informal narrative.
  - improves precision
  - while enabling formal proofs of properties of that which is denoted by the description.

---

<sup>15</sup>parts, actions, events and behaviours; attributes and possibly unique identifiers of parts, and mereology of composite (atomic) parts; subparts of composite parts; etc.

- We shall here use the term ‘formal’ in the sense of mathematics.
- A formal description language is here defined to have
  - a formal syntax,
  - a formal semantics, and
  - a proof system.
- We shall “unravel” an example formal description language, **FDL**, in this section.
  - **FDL** has similarities to the **RAISE** Specification Language, **RSL**,
  - but, as our informal explanation of the meaning of **FDL** will show, it is not **RSL**.
  - The similarities are “purely” syntactical.

## 4.2. A Formal Description Language

### 4.2.1. Observing and Describing Entities

- We make the obvious distinction between
  - observing semantic values but
  - expressing syntactic structures.
- We observe
  - parts, but first express types and then their properties;
  - actions, but first express their signatures and then their definitions;
  - events, but first express their signatures and then their definitions; and
  - behaviours, but first express their signatures and then their definitions.

## 4.2.2. Observing and Describing Parts

- In order to describe a part we use such phrases as:
  - *a patient*
    - \* (whom we here consider to be an atomic part))
    - \* is characterised by as set of properties
      - a name .
      - a central personal registration identifier,
      - a gender,
      - a birth date,
      - a birth place,
      - a nationality,
      - a weight,
      - a height,
      - a insurance policy,
      - a medical record,
      - etcetera;

and

– *a transport net*

\* (whom we here consider to be a composite part))

\* is characterised by

• a set of properties

• a structure of, in this case two subparts, ie.,

• a **set of hubs** and

• a **set of links**,

and

• their mereology.

- Thus we take the nouns *name*, *central personal registration identifier*, *gender*, *birth date*, *birth place*, *nationality*, *weight*, *height*, *insurance policy*, *medical record*, ..., *set of hubs*, and *set of links* as type names.
- The names ‘patient’ and ‘transport net’ are also domain names.



- That is,
  - we go from instance of part
  - to the type of all parts “of the same kind”.
- One must take great care in not confusing the two: (type and value).
- Later we shall clarify the distinction between type and domain names.

### 4.2.2.1. Abstract Types

- By an *abstract type* we generally mean some further unexplained set of mathematical quantities.
  - Abstract types are in contrast to concrete types
  - by which we mean such mathematical quantities as sets, Cartesians, lists, maps and functions (in general).
  - Abstract types are also referred to as *sorts*.
- The FDL clause:

**type** A

- defines what we shall here, simplifying, take as a set of values said to be of type A.
- A is said to be a *type name* (here, more specifically, a *sort name*).

## 4.2.2.2. Concrete Types

- Borrowing from **RSL**, and, in general, discrete mathematics, we introduce **FDL** clauses for expressing set, Cartesian, list, map and function types.
- Let  $A, B, \dots, C$  be (type or sort) names which denote some (not necessarily distinct) types, then

$$\mathbf{A\text{-set}}, (A \times B \times \dots \times C), A^*, A^\omega, A \xrightarrow{m} B, A \rightarrow B, A \xrightarrow{\sim} B$$

are *type expressions* which denote the following (left-to-right) concrete types:

- set of sets,
- set of finite lists,
- set of possibly infinite lists,
- set of maps,
- set of total functions, and
- set of partial functions.

- Choosing to describe a part
  - as a sort
  - rather than a concrete typereflects a *principle* of abstraction.
- Modelling a concrete type in terms of, for example,
  - a map type  $(A \xrightarrow{m} B)$
  - rather than as type of indexed sets  $((A \times B)\text{-set})$reflects a modelling *technique*.

### 4.2.2.3. Type Definitions

- Besides
  - the sort type definitions, e.g., **type** A
  - there are the concrete type definitions.
- Let D be some (unused) type name, then
  - type** D = Type\_Expression
  - is a concrete type definition
  - where **Type\_Expression** is of either of the forms
 

* <b>A-set</b> ,	* $A^*$ ,	* $A \rightarrow B$ ,
* <b>A-infset</b> ,	* $A^\omega$ ,	* $A \xrightarrow{\sim} B$ and
* $(A \times B \times \dots \times C)$ ,	* $A \xrightarrow{m} B$ ,	* $A B \dots C$ ,
  - where A, B, ..., C are either type names or, more generally, other such type expressions and where
  - $A|B|\dots|C$  expresses the “union” type of the A. B. ..., and C types.

**Example 24 (Transport Net Types)** Let us exemplify the above by starting a series of examples all focused on a domain of transport nets.

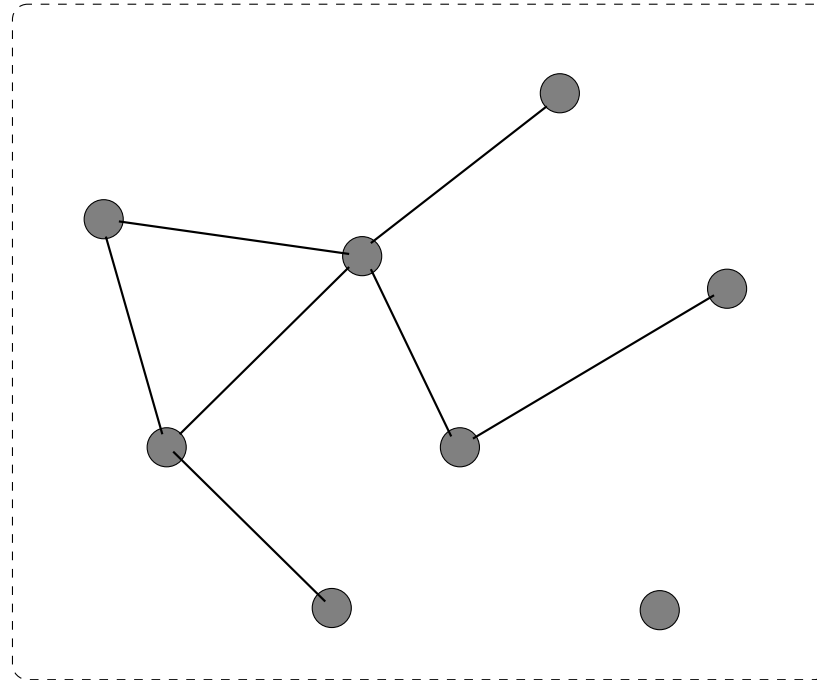
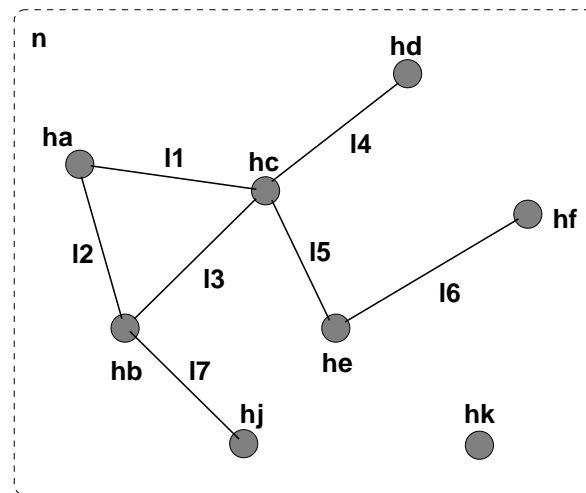


Figure 9: A transport net

- Figure 9 shows a net with
  - eight hubs and
  - seven links.

- To be able — here, in this text — to refer to fragments (here sub-parts),
  - of what is shown in Fig. 9,
  - we label the parts with names (Fig. 10);
  - these names stand for the designated parts.



$\text{obs\_Hs}(n) = \{\text{ha}, \text{hb}, \text{hc}, \text{hd}, \text{he}, \text{hf}, \text{hg}, \text{hj}, \text{hk}\}$

$\text{obs\_Ls}(n) = \{\text{l1}, \text{l2}, \text{l3}, \text{l4}, \text{l5}, \text{l6}, \text{l7}, \text{l8}, \text{l9}\}$

Figure 10: A transport net

32. We focus on the transport net domain.

That domain is “dominated” by the composite parts of nets,  $n:N$ .

32. **type** N

33. There are two subparts of nets:

(a) sets, **hs:HS**, of hubs (seen as one part) and

(b) sets, **ls:LS**, of links (also seen as one, but another part).

33(a). HS

33(b). LS

- As part of identifying the composite net type, **N**,
- we also identify two observers:
  - **obs\_HS** (observe [set of] hubs) and
  - **obs\_LS** (observe [set of] links)



- That is:

33(a).  $\text{obs\_HS}: N \rightarrow \text{HS}$

33(b).  $\text{obs\_LS}: N \rightarrow \text{LS}$

34. Hubs subparts of **HS** ('sets of hubs'), and

35. links are subparts of respectively **LS** ('sets of links'),

and are of types

(a) **H**, respectively

(b) **L**.

**type**

35(a). **H**

35(b). **L**

- We may, for convenience, bypassing a step (i.e., Items 33(a)–33(b)) instead express:

**type**

35(a). Hs = **H-set**

35(b). Ls = **L-set**

**value**

35(a). obs\_Hs: N  $\rightarrow$  **H-set**

35(b). obs\_Ls: N  $\rightarrow$  **L-set**



## 4.2.2.4. Type Properties

- In the following we shall be introducing a number of functions which analyse parts with respect to respective *properties*.
- There are three kinds of properties of interest to us:
  - the subparts of composite parts,
  - the mereology of composite parts,
  - and general attributes of parts (apart from their possible subparts and mereology).

- Every entity, whether simple, or an action, or an event, or a behaviour, has a unique identification.
  - The mere existence, in time and space, endows a part with a unique identification as follows.
    - \* No two spatial parts can occupy overlapping space,
    - \* so some abstract spatial location is a form of unique identification.
  - We consider the unique identity of a part of type **A**, say **AI**, as a general attribute.
  - We use the attribute values of **AI** to formulate mereologies.

Thus there are three kinds of property analysis functions.

- *Subpart observer functions*,  $\text{obs}_B$ ,  $\text{obs}_C$ , ...,  $\text{obs}_D$ , which apply to composite parts (say of type  $A$ ), and yield their constituent subparts, say of type  $B$ ,  $C$ , ...,  $D$ :

$$\text{obs}_B: A \rightarrow B, \text{obs}_C: A \rightarrow C, \dots, \text{obs}_D: A \rightarrow D;$$

- *Mereology functions* which apply to composite parts (say of type  $A$ ), and yields elements of their mereologies.

- Let parts of type  $B$ ,  $C$ , ...,  $D$  be in some mereology relation to parts of type  $A$ .
- That is, there are mereology functions

$$\text{mereo}_{Ax}: B \rightarrow AI_x, \text{mereo}_{Ay}: C \rightarrow AI_y, \dots, \text{mereo}_{Az}: D \rightarrow AI_z$$

\* where  $Ax$ ,  $Ay$  and  $Az$  are some distinct identifiers and

\*  $AI_x$ ,  $AI_y$  and  $AI_z$  are some type expressions over type  $A$ , typically

$AI$ ,  $AI\text{-set}$ ,  $(AI \times \dots \times AI)$ , etc.

- *Attribute Functions* which apply to parts (say of type **A**) and yield their attributes (short of the mereologies) (say of types **E**, **F**, ..., **G**:

$$\text{attr\_E}: A \rightarrow E, \text{attr\_F}: A \rightarrow F, \dots, \text{attr\_G}: A \rightarrow G;$$

Among the general attribute functions are the unique identification functions, say **attr\_A**.

## 4.2.2.5. Subpart Type Observers

- Given a composite sort, named, say,  $A$ ,
- and postulating that its values contains subparts of type  $B$ ,
- one can observe these type  $B$  subparts of  $A$  using
- the likewise postulated *observer function*:

$$\text{obs}_B: A \rightarrow B$$

- If  $A$  values also contain subparts of types  $C, \dots, E$ , then there also exists the additional observer functions:  $\text{obs}_C, \dots, \text{obs}_E$ .
- We say that the observer functions are postulated.
  - We postulate them.
  - And we endow them with properties
  - so that they “stand out” from one another.
    - \* First examples of properties are given by the observer function signatures: from type  $A$  values observer function  $\text{obs}_B$  yields  $B$  values.
    - \* Further properties may be expressed through axioms.

**Example 25 (Subpart Type Observers)** From nets we observe

36. sets of hubs and

37. sets of links.

in either of two ways:

**value**

36.  $\text{obs\_HS}: N \rightarrow \text{HS}$

37.  $\text{obs\_LS}: N \rightarrow \text{LS}$

**value**

35(a).  $\text{obs\_Hs}: N \rightarrow \text{H-set}$

35(b).  $\text{obs\_Ls}: N \rightarrow \text{L-set}$





## 4.2.2.6. Unique Identifier Functions

- All parts have unique identifiers.
- This is a dogma.
- We may never need some (or any) of these unique part identifiers.
- But they are there nevertheless.

**Example 26 (Unique Hub and Link Identifiers)** From hubs and links we observe their unique

38. hub and

39. link

identifier attributes and their ‘observers’:

**type**

38. HI

39. LI

**value**

38. uid\_HI:  $H \rightarrow HI$

39. uid\_LI:  $L \rightarrow LI$

- Figure 11 shows the labelling
  - of links with unique link identifiers, and
  - of hubs with unique hub identifiers.
- It also shows sample unique identifier observer functions.

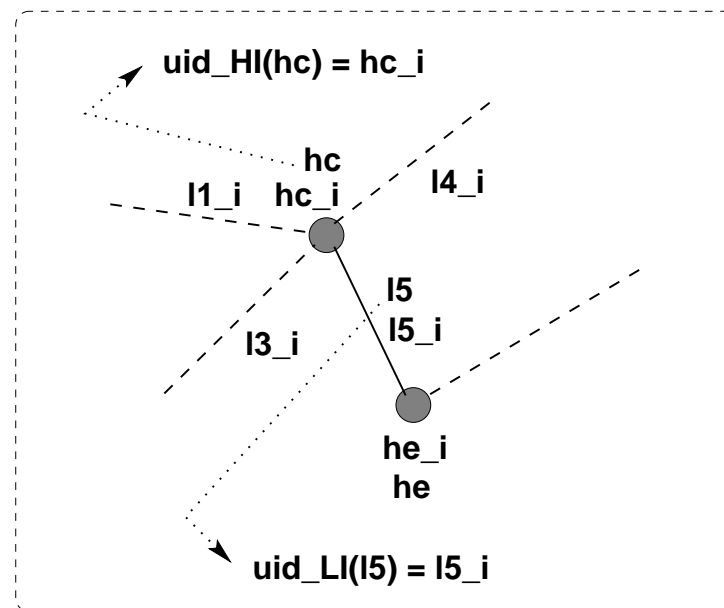


Figure 11: Fragment of a transport net emphasizing unique part identifiers and their observers



### 4.2.2.7. Mereologies and Their Functions

**Example 27 (Transport Net Mereology)** To express the mereology of transport nets we build on the unique identifications of hubs and links.

40. Links connect exactly two distinct hubs, **mereo\_HIs**.

41. Hubs are connected to zero, one or more distinct links, **mereo\_LIs**.

**type**

40. HIs = HI-set

**axiom**

40.  $\forall$  his:HIs **card** his=2

**type**

41. LIs = LI-set

**value**

40. mereo\_L: L  $\rightarrow$  HIs

41. mereo\_H: H  $\rightarrow$  LIs

- Figure 12 illustrates the idea of mereology observer.

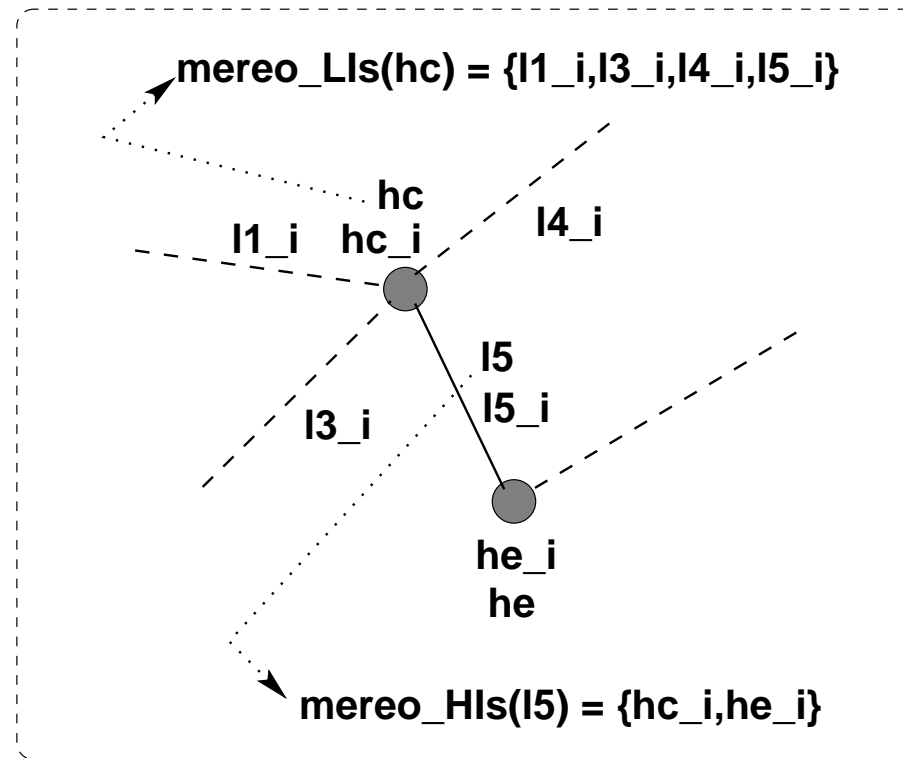


Figure 12: Fragment of a transport net emphasizing mereology observers

- The above (Items 40–41) form the basis for expressing the constraints on how hubs and links are connected.
42. Given a net, its link and hub observers and the derived link and hub identifier extraction functions, the mereology of all nets must satisfy the following:
- (a) All link identifiers observed from hubs must be of links of that net and
  - (b) All hub identifiers observed from links must be of hubs of that net.
43. We introduce two auxiliary functions for extracting all hub and link identifiers of a net.

**axiom**

42.  $\forall n:N,$

42.     **let**  $ls=obs\_LS(n),hs=obs\_HS(n),$

42.          $lis=xtr\_LIs(n),his=xtr\_HIs(n)$  **in**

42(a).    $\forall h:H.h \in hs \Rightarrow mereo\_H(h) \subseteq lis \wedge$

42(b).    $\forall l:L.l \in ls \Rightarrow mereo\_L(l) \subseteq his$  **end**

**value**

43.  $xtr\_LIs: N \rightarrow LI\text{-set}, xtr\_HIs: N \rightarrow HI\text{-set}$

43.  $xtr\_LIs(n) \equiv \{uid\_LI(l) | l:L.l \in obs\_LS(n)\}$

43.  $xtr\_HIs(n) \equiv \{uid\_HI(h) | h:H.h \in obs\_HS(n)\}$



## 4.2.2.8. General Attributes and Their Functions

**Example 28 (Hub States and Hub State Spaces)** In addition to the unique identifiers and the mereology of parts there are the general attributes.

- An example are the states of hubs and links
- where these states indicate the direction of traffic
- for which the hubs and links are open.

44. With any hub,  $h$ , we thus associate

- (a) a hub state,  $h\sigma$ , and
- (b) a hub state space.

45. The relations between

- (a) the link identifiers of the hub, (c) the hub state spaces
- (b) the hub states, and

46. must satisfy the following

- (a) wrt. the potential set, **hs**, which is the “largest possible” hub state for  $h$ , one that allows traffic from any link  $l_i$  incident upon  $h$  to any link  $l_j$  emanent from  $h$ :
- (b) the hub state is any subset of **hs**, and
- (c) and such hub state is in that hub’s state space.



**type**44(a).  $H\Sigma = (LI \times LI)\text{-set}$ 44(b).  $H\Omega = H\Sigma\text{-set}$ **value**44(a).  $\text{attr\_H}\Sigma: H \rightarrow H\Sigma$ 44(b).  $\text{attr\_H}\Omega: H \rightarrow H\Omega$ **axiom**45.  $\forall n:N, h:H \cdot h \in \text{obs\_Hs}(n) \Rightarrow$ 41. **let**  $h\text{lis} = \text{mereo\_H}(h),$ 44(a).  $h\sigma = \text{attr\_H}\Sigma(h),$ 44(b).  $h\omega = \text{attr\_H}\Omega(h),$ 43.  $\text{lis} = \text{xtr\_LIs}(n)$  **in**46(a). **let**  $h\text{s} = \{(li, lj), (lk, li) \mid li:LI \cdot li \in h\text{lis} \wedge \{lj, lk\} \subseteq \text{lis}\}$  **in**46(b).  $h\sigma \subseteq h\text{s} \wedge$ 46(c).  $h\sigma \in h\omega$  **end end**

**Example 29 (Further Atomic Attributes)** In addition to the unique link identifier links also have, for example, lengths, widths, possibly heights, geographic (spatial) locations, etc.

**type**

LEN, WID, HEI, LOC, ...

**value**

attr\_LEN:  $L \rightarrow \text{LEN}$

+:  $\text{LEN} \times \text{LEN} \rightarrow \text{LEN}$

>:  $\text{LEN} \times \text{LEN} \rightarrow \mathbf{Bool}$

attr\_WID:  $L \rightarrow \text{WID}$

attr\_HEI:  $L \rightarrow \text{HEI}$

attr\_LOC:  $L \rightarrow \text{LOC}$

...



## 4.2.3. Describing Actions

### 4.2.3.1. Function Names

- Actions potentially change states.
- Actions are here considered deliberate phenomena
  - in that they are caused by willful applications
  - (by agents within the domain being described)
  - of functions
  - having a specific, deliberate purpose,
  - i.e., state change in mind.

**Example 30 (Transport Net Action Names)** Some examples are:

- *create* an *empty net* (no hubs, no links);
- *insert* a (new) hub;
- *insert* a (new) *link* (between a pair of distinct hubs of the net);
- *delete* (existing) *hub* (having no links into or out from it); and
- *delete* (existing) *link*.



### 4.2.3.2. Informal Function Descriptions

- The above examples just listed some actions by their function names.
- We did not describe these functions.
- We now do so, for two of these functions.

**Example 31 (Informal Transport Net Action Descriptions)** We detail two informal function descriptions.

- The *create empty net* function
  47. applies to nothing and yields a net,  $\mathbf{n}$ , of no hubs and no links.
- The *insert link* function
  48. applies to a net,  $\mathbf{n}$ , of at least two distinct hubs, as identified,  $\mathbf{hi}_j$ ,  $\mathbf{hi}_k$ , by the function application arguments, and a new link  $\ell$  not in  $\mathbf{n}$ , and yields a net,  $\mathbf{n}'$ .

- (a) The inserted link  $l$  is to be connected to the two distinct hubs identified by  $hj_i$  and  $hk_i$ , and these designate hubs of the net.
- (b)  $l$  is not in the original net.
- (c) The mereology of  $l$  designate  $hj_i$  and  $hk_i$ .
- (d) The state of these identified hubs do not allow any traffic.
- (e) No new hubs are added, hence the set of hub identifiers of the net is unchanged.
- (f) Only one link is added to the net, hence the set of net link identifiers is changed by only the addition of the  $l$  link identifier.
- (g) Let the hubs identified by  $hj_i$  and  $hk_i$  be  $hj$  and  $hk$ , respectively, before the insertion,
- (h) and by  $hj'$  and  $hk'$  after the insertion.
- (i) Now the mereology contributions by the two changed hubs reflect only the addition of link  $l$ .

---

We leave the informal descriptions of

49. *delete\_L*

50. *insert\_H*

51. *delete\_H*

to the reader.



### 4.2.3.3. Formal Function Descriptions

- We observe actions,
  - but describe the functions
  - which when applied amount to actions.
- There are two parts to describe a function:
  - (i) the function signature,  $\mathbf{a:A \rightarrow B}$ :
    - \* a distinct function name, say  $\mathbf{f}$ , and
    - \* a function type,  $\mathbf{A \rightarrow B}$ , that is,
      - type of arguments  $\mathbf{A}$  and
      - type of results  $\mathbf{B}$ , and
  - (ii) the function definition,  $\mathbf{f(a,b) \equiv C(a)}$ :
    - \* a symbolic function invocation,  $\mathbf{f(a,b)}$ , and
    - \* a definition body,  $\mathbf{C(a)}$ .
      - $\mathbf{C(a)}$  is a clause,
      - i.e., an expression in FDL,
      - whose evaluation
      - yields the function value.



- There are other ways than:

**value**

$$f: A \rightarrow B$$

$$f(a,b) \equiv \mathcal{C}(a)$$

in which to define a function.

- For example:

**value**

$$f: A \rightarrow B$$

$$f(a) \text{ as } b$$

$$\text{pre } \mathcal{P}(a)$$

$$\text{post } \mathcal{Q}(a,b)$$

## Example 32 (Formal Transport Net Action Descriptions) •

The *create net* function:

**value**

47. `create_N: Unit → N`

47. `create_N() as n`

47. `post obs_HS(n)={ } ∧ obs_LS(n)={ }`

- The *insert link* function:

**value**

48.  $\text{insert\_L}: (\text{HI} \times \text{HI}) \times \text{L} \rightarrow \text{N} \rightarrow \text{N}$

48.  $\text{insert\_L}((\text{hij}, \text{hik}), \text{l})(\text{n})$  **as**  $\text{n}'$

48(a). **pre**  $\text{hji} \neq \text{hki} \wedge \{\text{hji}, \text{hki}\} \subseteq \text{xtr\_HIs}(\text{n}) \wedge$

48(b).  $\wedge \text{l} \notin \text{obs\_LS}(\text{n})$

48(c).  $\wedge \text{mereo\_L}(\text{l}) = \{\text{hji}, \text{hki}\}$

48(d).  $\wedge \text{attr\_H}\Sigma(\text{get\_H}(\text{hji}))(\text{n}) = \{\} = \text{attr\_H}\Sigma(\text{get\_H}(\text{hki}))(\text{n})$

48(e). **post**  $\text{xtr\_HIs}(\text{n}) = \text{xtr\_HIs}(\text{n}')$

48(f).  $\wedge \text{xtr\_LIs}(\text{n}') = \text{xtr\_LIs}(\text{n}) \cup \{\text{uid\_LI}(\text{l})\}$

48(g).  $\wedge$  **let**  $\text{hj} = \text{get\_H}(\text{hij})(\text{n}), \text{hk} = \text{get\_H}(\text{hik})(\text{n}),$

48(h).  $\text{hj}' = \text{get\_H}(\text{hij})(\text{n}'), \text{vhk}' = \text{get\_H}(\text{hik})(\text{n}')$  **in**

48(i).  $\wedge \text{mereo\_H}(\text{hj}') = \text{mereo\_H}(\text{hj}) \cup \{\text{uid\_LI}(\text{l})\}$

48(i).  $\wedge \text{mereo\_H}(\text{hk}') = \text{mereo\_H}(\text{hk}) \cup \{\text{uid\_LI}(\text{l})\}$  **end**

52. From the postulated observer and attribute functions one can define the auxiliary *get* function:

**value**

52.  $\text{get\_H}: \text{HI} \rightarrow \text{N} \xrightarrow{\sim} \text{H}$

52.  $\text{get\_H}(\text{hi})(\text{n}) \equiv$

52.     **let**  $\text{h}:\text{H} \cdot \text{h} \in \text{obs\_HS}(\text{n}) \wedge \text{uid\_HI}(\text{h}) = \text{hi}$

52.     **in**  $\text{h}$  **end**

52.     **pre**  $\text{hi} \in \text{xtr\_HIs}(\text{n})$

- We do not narrate the informal description of “remaining” net actions (cf. Items 49– 51 on page 157), just their function signatures and pre-conditions.

49.  $\text{delete\_L}: \text{LI} \rightarrow \text{N} \xrightarrow{\sim} \text{N}$

49. **pre**  $\text{delete\_L}(li): li \in \text{xtr\_LIs}(n)$

50.  $\text{insert\_H}: \text{H} \rightarrow \text{N} \xrightarrow{\sim} \text{N}$

50. **pre**  $\text{insert\_H}(h): h \notin \text{obs\_HS}(n) \wedge \text{mereo\_H}(h) = \{\} \wedge \text{mereo\_}\Omega(h) = \{\{\}\}$

51.  $\text{delete\_H}: \text{HI} \rightarrow \text{N} \xrightarrow{\sim} \text{N}$

51. **pre**  $\text{delete\_H}(hi): hi \in \text{xtr\_HIs}(n) \wedge \text{mereo\_H}(\text{get\_H}(hi))(n) = \{\}$

- Given appropriate post-conditions the following theorems must be provable:

**theorems:**

$\forall h:\text{H} \cdot \text{pre-conditions are satisfied} \Rightarrow \text{delete\_H}(\text{uid\_HI}(h))(\text{insert\_H}(h)(n)) = n$

$\forall l:\text{L} \cdot \text{pre-conditions are satisfied} \Rightarrow \text{delete\_L}(\text{uid\_LI}(l))(\text{insert\_L}(l)(n)) = n$



## 4.2.3.4. Agents

- An *agent* is a behaviour
  - which invokes functions,
  - hence cause actions.
- So we simply “equate” agents with behaviours.

## 4.2.4. Describing Events

- We observe events. But we describe logical properties characterising classes of “the same kind” of events.

### 4.2.4.1. Deliberate and Inadvertent (Internal and External) Events

- Events are like actions: somehow a function was applied
  - either *deliberately* by an agent outside (that is, *external* to) the domain being described,
  - or *inadvertently* by a behaviour of (that is, *internal* to) the domain, but for another purpose than captured by the event.

**Example 33 (A Deliberate [External] Event)** We narrate a simple external cause / “internal” effect example:

- When one or more bank customers default on their loans
- and declare themselves unable to honour these loans
- then the bank may go bankrupt.





**Example 34 (An Inadvertent [Internal] Action Event)** We narrate a simple internal cause / “internal” effect example:

- When a bank customer, the agent, withdraws monies from an account
  - the balance of that account, if the withdrawal is completed,
    - \* may go negative,
    - \* or may go below the credit limit.
  - In either case we say that,
    - \* the withdrawal action, as was intended, succeeded,
    - \* but that an “*exceeded credit limit*” event occurred.



## 4.2.4.2. Event Predicates

- Instead of describing events
  - by directly characterising the
    - deliberate external, respectively
    - inadvertent internal
    - actions
- we suggest to describe these events
  - indirectly,
  - by characterising the logical effects,
  - say, in terms of predicates over *before/after* states.

**Example 35 (Formalisation of An External Event)** The event is that of a “*link segment disappearance*”.

53. Generally we can explain “*link segment disappearances*”, for example, as follows:

54. A  $l_i$ -identified link,  $l$ , between hubs  $hf$  and  $ht$  (identified in  $l$ ) is *removed*.

55. Two hubs,  $hf''$  and  $ht''$ , and two links,  $lf$  and  $lt$ , are *inserted* — where

- (a) hub values  $hf$  and  $ht$  (the hubs in the original net) become hub values  $hf'$  and  $ht'$  in the resulting net, that is, hub values  $hf$  and  $ht$  have same respective hub identifiers as  $hf'$  and  $ht'$ ,
- (b) hubs  $hf''$  and  $ht''$  are new,
- (c) links  $l'$  and  $l''$  are new,
- (d) link  $lf$  is *inserted* between  $hf'$  and  $hf''$ , that is, link  $lf$  identifies hubs  $hf'$  and  $hf''$ ,  
and link  $lt$  is *inserted* between  $ht'$  and  $ht''$ , that is, link  $lt$  identifies hubs  $ht'$  and  $ht''$ ,

- (e) hub  $hf'$  is, in the resulting net, connected to the links hub  $hf$  was connected to in the original net “minus” link  $l$  but “plus” link  $lf$ ,
- (f) hub  $ht'$  is, in the resulting net, connected to the links it was connected to in the original net “minus” link  $l$  but “plus” link  $lt$ ,
- (g) hub  $hf''$  is connected only to link  $lf$   
and hub  $ht''$  is connected only to link  $lt$ ,
- (h) the state space of  $hf'$  suitably includes all the possibilities of entering link  $lf$ <sup>16</sup>,
- (i) the state space of  $ht'$  suitably includes all the possibilities of entering link  $lt$ <sup>17</sup>,
- (j) the state spaces of  $hf''$  and  $ht''$  both contains just the empty set,
- (k) the states of  $hf''$  and  $ht''$  are both the empty set: “dead ends !”,
- (l) the sum of the lengths of links  $lf$  and  $lt$  is less than the length of link  $l$ , and
- (m) all other non-mereology attributes of  $lf$  and  $lt$  are the same as those of link  $l$ .

56. All other links and hubs are unchanged.

<sup>16</sup>A substitution function replaces all link  $l$  identifiers with link  $lf$  identifiers.

<sup>17</sup>A substitution function replaces all link  $l$  identifiers with link  $lt$  identifiers.

**value**

53. link\_segment\_disappearance:  $\mathbb{N} \times \mathbb{N} \rightarrow \mathbf{Bool}$

53. link\_segment\_disappearance( $n, n'$ )  $\equiv$

53.  $\exists l:L, hf'', ht'':H, lf, lt:L \cdot$

54.  $\{l\} = \text{obs\_LS}(n) \setminus \setminus \text{obs\_LS}(n')$

55(a).  $\wedge \mathbf{let} \text{ hfi}=\text{uid\_HI}(hf), \text{ hti}=\text{uid\_HI}(ht) \mathbf{in}$

55(a).  $\mathbf{let} \text{ hf}'=\text{get\_H}(\text{hfi})(n'), \text{ ht}'=\text{get\_H}(\text{hti})(n') \mathbf{in}$

55(b).  $\{\text{hf}'', \text{ht}''\} \cap \text{obs\_HS}(n) = \{\} \wedge \{\text{hf}'', \text{ht}''\} \subseteq \text{obs\_HS}(n')$

55(c).  $\wedge \{\text{lf}, \text{lt}\} \cap \text{obs\_LS}(n) = \{\} \wedge \{\text{lf}, \text{lt}\} \subseteq \text{obs\_LS}(n')$

55(d).  $\wedge \text{mereo\_L}(l) = \{\text{hfi}, \text{uid\_HI}(\text{hf}'')\} \wedge \text{mereo\_L}(l') = \{\text{hti}, \text{uid\_HI}(\text{ht}'')\}$

55(e).  $\wedge \text{mereo\_H}(hf) = \text{mereo\_H}(hf') \setminus \{\text{uid\_LI}(l)\} \cup \{\text{uid\_LI}(lf)\}$

55(f).  $\wedge \text{mereo\_H}(ht) = \text{mereo\_H}(ht') \setminus \{\text{uid\_LI}(l)\} \cup \{\text{uid\_LI}(lt)\}$

55(g).  $\wedge \text{mereo\_H}(hf'') = \{\text{uid\_LI}(lf)\} \wedge \text{mereo\_H}(ht'') = \{\text{uid\_LI}(lt)\}$

55(h).  $\wedge \text{attr\_H}\Omega(hf') = \text{subst}(\text{uid\_LI}(l), \text{uid\_LI}(lf), \text{attr\_H}\Omega(hf))$

55(i).  $\wedge \text{attr\_H}\Omega(ht') = \text{subst}(\text{uid\_LI}(l), \text{uid\_LI}(lt), \text{attr\_H}\Omega(ht))$

55(j).  $\wedge \text{attr\_H}\Omega(hf'') = \{\{\}\} \wedge \text{attr\_H}\Omega(ht'') = \{\{\}\}$

55(k).  $\wedge \text{attr\_H}\Sigma(hf'') = \{\} \wedge \text{attr\_H}\Sigma(ht'') = \{\}$

55(l).  $\wedge \text{attr\_LEN}(lf) + \text{attr\_LEN}(t) < \text{attr\_LEN}(l)$

55(m).  $\wedge \forall X:\text{non\_mereo\_attributes}(l) \cdot \text{attr\_X}(lf) = \text{attr\_X}(lt) = \text{attr\_X}(l)$ <sup>18</sup>

56.  $\wedge \text{obs\_HS}(n') \setminus \{\text{hf}', \text{hf}'', \text{ht}', \text{ht}''\} = \text{obs\_HS}(n) \setminus \{\text{hf}, \text{ht}\}$

56.  $\wedge \text{obs\_LS}(n') \setminus \{\text{lf}, \text{lt}\} = \text{obs\_LS}(n) \setminus \{l\} \mathbf{end\ end}$

<sup>18</sup>The predicate in Line 55(m) on the facing page is to be explained.

- We can express a theorem relating the above to the
  - *remove* and
  - *insert* functions.

**theorem:**  $\text{link\_segment\_disappearance}(n, n') \Rightarrow$   
**let**  $l:L, hf'', ht'':H, lf, lt:L \cdot$  [ Lines 54–55, 55(a)–55(m), 56 ] **in**  
 $n' = \text{ins\_L}(lt)(\text{ins\_L}(lf)(\text{ins\_H}(ht'')(\text{ins\_H}(hf'')(\text{rem\_L}(\text{uid\_LI}(l))(n))))))$   
**end**

•

---

## End Lecture 4: 1st part: Describing Domain Entities [Parts–Events]

---

---

## Lecture 5: 2nd part: Describing Domain Entities [Behaviours]

---



## 4.2.5. Describing Behaviours

### 4.2.5.1. Behaviour Description Languages

- As for the description of parts, actions and events<sup>19</sup> there exists formal ways of describing behaviours as of sequences of actions, events and behaviours:
  - some are “textual”<sup>20</sup>:
    - \* CSP,
  - some are “graphical”, for example:
    - \* MSC,
    - \* Petri Nets and
    - \* State Charts.

---

<sup>19</sup> Part, action and event description languages were first mentioned in the ‘Summary’ footnotes 1– 2 on page 2: Alloy, CafeOBJ, Casl Event B, Maude RAISE/RSL , VDM and Z.

<sup>20</sup>The languages mentioned in Footnote 19 are textual.

### 4.2.5.1.1. Simple Sequential Behaviours

- A simple sequential behaviour
  - is a sequence of actions and events.

#### ★ *Snapshot Description of a Simple Sequential Behaviour* ★

- Snapshot of a behaviour, as it unfolds, could be described:

```

let  $\sigma'$  = action_1(arg_1)( $\sigma$ )  $\sqcap$  event_1( $\sigma$ )( $\sigma'$ ) in
let  $\sigma''$  = action_1(arg_1)( $\sigma'$ )  $\sqcap$  event_1( $\sigma'$ )( $\sigma''$ ) in
...
let  $\sigma''...'$  = action_1(arg_1)( $\sigma'...'$ )  $\sqcap$  event_1(sigm'...'a)( $\sigma''...'$ ) in
 $\sigma''...'$  end ... end end

```

where the internal non-deterministic operator,  $\sqcap$ , expresses that either its left side or right operand is chosen.

- We can name such simple sequential behaviours, for example: P.

### 4.2.5.1.3. Simple Concurrent Behaviours

- A simple concurrent behaviour
  - is a set of two or more simple sequential behaviours.
- We describe a simple concurrent behaviour
  - by a list of named behaviour descriptions
  - separated by the **parallel** behaviour composition operator  $\parallel$ ,
  - for example,  $P \parallel Q \parallel \dots \parallel R$ .
  - A variant form is  $\parallel \{P(i) \mid i: \text{Index} \bullet \text{predicate}(i)\}$  which expresses ‘distribution’ of the behaviour composition operator ( $\parallel$ ) over ‘expanded’ terms, that is,  $P(i_j) \parallel P(i_k) \parallel \dots \parallel P(i_\ell)$ .

## 4.2.5.1.4. Communicating Behaviours

- A communicating behaviour
  - is a behaviour which expresses willingness to engage
  - in a (synchronisation and) communication
  - with another communicating behaviour
  - or with the environment.
- In order to express ‘communication’ (between behaviours)
  - a notion of an output/input *channel* is introduced
  - with behaviours allowed ‘access’ to channel (which are therefore shared).
- In **CSP** channels are typed with the type of the values that can be output on a channel “between” behaviours.

- In CSP
  - output of a value (say of expression  $e$ ) onto channel  $ch$  is expressed by the statement  $ch!e$  whereas
  - input of a value from channel  $ch$  is expressed by the expression  $ch?$ .

**type**

$M$

**channel**

$ch:M$

**value**

$S: \mathbf{Unit} \rightarrow \mathbf{Unit}, \quad S() = P() \parallel Q()$

$P: \mathbf{Unit} \rightarrow \mathbf{out} \ ch \ \mathbf{Unit}, \quad P() \equiv \dots \ ch!e \ \dots$

$Q: \mathbf{Unit} \rightarrow \mathbf{in} \ ch \ \mathbf{Unit}, \quad Q() \equiv \dots \ ch? \ \dots$

- We thus describe a communicating behaviour by allowing one or more clauses:
  - statements of the kind  $ch!e$  and
  - expressions of the kind  $ch?$ .

### 4.2.5.1.5. External Non-deterministic Behaviours

- A behaviour,  $\mathcal{P}$ , composed from behaviours  $\mathcal{P}_i, \mathcal{P}_j, \dots, \mathcal{P}_k$  is said to exhibit internal non-determinism if the behaviour is
  - either as is behaviour  $\mathcal{P}_i$ ,
  - or as is behaviour  $\mathcal{P}_j$ ,
  - $\dots$ ,
  - or as is behaviour  $\mathcal{P}_k$ ,
  - and is influenced in being so by the environment of behaviour  $\mathcal{P}$ .
- We describe such behaviours as follows:
 
$$P: P\_i \sqcup P\_j \sqcup \dots \sqcup P\_k$$
 where  $P\_i$  (etcetera) describes behaviour  $\mathcal{P}_i$  (etcetera).
- A variant description of internal non-determinism is
 
$$\sqcup \{P(i) \mid i: \text{Index} \cdot \text{predicate}(i)\}.$$

- External influence is, for example, expressed
  - if behaviour descriptions  $P_i$  (etcetera)
  - contain either an output (**ch!e**) or an input (**ch?**) clause
  - and the environment offers to accept input, respectively offers output
  - “along” the name channel.

### 4.2.5.1.6. Internal Non-deterministic Behaviours

- A behaviour,  $\mathcal{P}$ , composed (somehow) from behaviours  $\mathcal{P}_i, \mathcal{P}_j, \dots, \mathcal{P}_k$  is said to exhibit internal non-determinism if the behaviour is
  - either as is behaviour  $\mathcal{P}_i$ ,
  - or as is behaviour  $\mathcal{P}_j$ ,
  - $\dots$ ,
  - or as is behaviour  $\mathcal{P}_k$ ,
  - and is not influenced in being so by the environment of behaviour  $\mathcal{P}$ .
- We describe such behaviours as follows:
 
$$P: P\_i \sqcap P\_j \sqcap \dots \sqcap P\_k$$
 where  $P\_i$  (etcetera) describes behaviour  $\mathcal{P}_i$  (etcetera).
- A variant description of internal non-determinism is
 
$$\sqcap \{P(i) \mid i: \text{Index} \cdot \text{predicate}(i)\}.$$



- For internal non-determinism to work for expressions like the above
  - we must assume that they do not contain such
  - output (**ch!**e) or an input (**ch?**) clauses
  - for which the environment may accept input, respectively offer output.

## 4.2.5.1.7. General Communicating Behaviours

- A general communicating behaviour
  - is a set of sequences of actions, events and
  - (simple sequential, simple concurrent, communicating and non-deterministic)
  - behaviours
  - such that at least two separately identifiable behaviours of a set
  - share at least one channel and
  - contain respective **ch!** and **ch?** clauses.

## Example 36 (A Road Pricing (Transport) System Behaviour)

This example is quite extensive.

57. A road pricing (transport) system,  $\Delta_{\text{RPS}}$  contains

- (a) a net  $n$  — as outlined in earlier examples — of hubs and links,
- (b) a fleet  $f$  of vehicles and
- (c) a central road pricing monitor  $m$ .

58. From  $\Delta_{\text{RPS}}$  we can observe the

- (a) a net,  $\mathbf{n:N}$ ,
- (b) a fleet of vehicles,  $\mathbf{f:F}$ , and
- (c) a road pricing monitor,  $\mathbf{m:M}$ .

59. From the net,  $\mathbf{n:N}$ , we observe

- (a) a set of hubs and
- (b) a set of links

60. From the fleet,  $\mathbf{f:F}$ , we observe

- (a) a set of vehicles.

**type**57.  $\Delta\_RPS, N, F, M$ **value**57(a).  $obs\_N: \Delta\_RPS \rightarrow N$ 57(b).  $obs\_F: \Delta\_RPS \rightarrow F$ 57(c).  $obs\_M: \Delta\_RPS \rightarrow M$ 59(a).  $obs\_Hs: N \rightarrow \mathbf{H\text{-set}}$ 59(b).  $obs\_Ls: N \rightarrow \mathbf{L\text{-set}}$ 60(a).  $obs\_Vs: F \rightarrow \mathbf{V\text{-set}}$ 

- We need “prepare” some part names:

**type**57(a).  $n:N$ 57(b).  $f:F$ 57(c).  $m:M$ 59(a).  $hs:Hs=obs\_Hs(n)$ 59(b).  $ls:Ls=obs\_Ls(n)$ 60(a).  $vs:Vs=obs\_VSs(f)$

61. With the road pricing behaviour we associate separate behaviours,
- (a) one for the net which is seen as the parallel composition of
    - i. a set of hub behaviours
    - ii. a set of link behaviours;
  - (b) one for the fleet of vehicles which is seen as the parallel composition of
    - i. a set of vehicle behaviours;
  - (c) and a central road pricing monitor behaviour.

61. road\_pricing\_system: **Unit**  $\rightarrow$  **Unit**

61. road\_pricing\_system()  $\equiv$  net() || fleet() || monitor(...)

61(a). net()  $\equiv$

61((a))i. || {hub(uid\_HI(h))(h)(vis)|h:H·h  $\in$  hs} ||

61((a))ii. || {link(uid\_LI(l))(l)(vis)|l:L·l  $\in$  ls}

61(b). fleet()  $\equiv$

61((b))i. || {vehicle(obs\_VI(v))(v)(vp)|v:V·v  $\in$  vs}

61(c). monitor(...)  $\equiv$  ...

- The *vis* arguments of the *hub* and *link* behaviours “carry” the identifiers of current vehicles currently at the hub or on the link.
- The *vp* argument of the *vehicle* behaviour “carries” the current vehicle position.
- The (...) argument of the *monitor* behaviour records the history status of all vehicles on the net.

62. We associate channels as follows:

- (a) one for each pair of vehicles and hubs,
- (b) one for each pair of vehicles and links and
- (c) one for monitor (connected to vehicles).

## 62. channel

62(a).  $\{vh\_ch[uid\_VI(v),uid\_HI(h)] \mid v:V,h:H \cdot v \in vd \wedge h \in hs\}:VH\_Msg$

62(b).  $\{vl\_ch[uid\_VI(v),uid\_LI(h)] \mid v:V,l:L \cdot v \in vs \wedge h \in ls\}:VL\_Msg$

62(c).  $mon:VM\_Msg$

- We omit detailing the channel message types.



63. Vehicles are positioned

- (a) either on a link, in direction from one hub to a next, some fraction down that link,
- (b) or at a hub, in direction from one link to a next where
- (c) the fraction is a real between 0 and 1.

**type**

63.  $VP = \text{onL} \mid \text{atH}$

63(a).  $\text{onL} == \text{mk\_onL}(\text{li:LI}, \text{fhi:HI}, \text{f:FRA}, \text{thi:HI})$

63(b).  $\text{atH} == \text{mk\_atH}(\text{hi:HI}, \text{fli:LI}, \text{tli:LI})$

63(c).  $\text{FRA} = \mathbf{Real\ axiom} \ \forall \text{fra:FRA} \cdot 0 \leq \text{fra} \leq 1$

64. The *vehicle* behaviour is modelled as a CSP process which communicates with hubs, links and the monitor.
65. The *vehicle* behaviour is a relation over its position.  
If on a link, at some position,
- (a) then the vehicle may “remain” at that position,
  - (b) chosen so internally non-deterministically,
  - (c) or, if the vehicle position is not “infinitesimally” close to the “next” hub,
  - (d) then the vehicle will move further on along the link,
  - (e) some small fraction  $\delta$ ,
  - (f) else the vehicle moves into the next hub in direction of the link named  $li'$
  - (g) where  $li'$  is in the set of links connected to that hub —
  - (h) while notifying the link, the hub and the monitor of its entering the link and entering the hub.

**value**

65(e).  $\delta$ :**Real axiom**  $0 < \delta \ll 1$  vehc6

64. vehicle: VI  $\rightarrow$  V  $\rightarrow$  VP  $\rightarrow$

64. **out,in** {vl\_ch[ vi,li ] || li:LI- $\in$  xtr\_LIs(ls)} **out** m\_ch **Unit**

65. vehicle(vi)(v)(vp:mk\_onL(li,fhi,f,thi))  $\equiv$

65(a). vehicle(vi)(v)(vp)

65(b).  $\sqcap$

65(c). **if**  $f + \delta < 1$

65(d). **then** vehicle(vi)(v)(mk\_onL(li,fhi,f+ $\delta$ ,thi))

65(e). **else let** li':LI- '  $\in$  mereo\_H(get\_H(thi)(n)) **in**

65(g). vh\_ch[ vi,thi ]!enterH || vl\_ch[ vi,li ]!leaveL || m\_ch!leaveL\_enterH(vi,li,thi);

65(h). vehicle(vi)(v)(mk\_atH(thi,li,li')) **end end**

66. If the vehicle is at a hub,

- (a) then the vehicle may “remain” at that same position,
- (b) chosen so internally non-deterministically,
- (c) or move on to the next link,
- (d) in direction of a next hub,
- (e) while notifying the hub and monitor of leaving the hub and the link and the monitor of entering the link.

66.  $\text{vehicle}(vi)(v)(vp:\text{mk\_atH}(hi,fl_i,tli)) \equiv$

66(a).  $\text{vehicle}(vi)(v)(vp)$

66(b).  $\sqcap$

66(d). **let**  $\{hi',thi\} = \text{mereo\_L}(\text{getL}(tli)(n))$  **in assert:**  $hi' = hi$

66(e).  $\text{vh\_ch}[vi,hi]!\text{leaveH} \parallel \text{vl\_ch}[vi,tli]!\text{enterL} \parallel \text{m\_ch}!\text{leaveH\_enterL}(vi,hi)$

66(c).  $\text{vehicle}(vi)(v)(\text{ml\_onL}(tli,hi,0,thi))$  **end**

67. The monitor behaviour records the (dynamic) history of all vehicles on the net: alternating sequences of hub and link identifiers.
68. The monitor contains a price table which to every link and hub records the fee for moving along that link or hub.

### type

$$67. \quad VW' = VI \xrightarrow{m} (HI|LI)^*$$

$$67. \quad VW = \{ |vw:VW'.wf\_VW(vh)| \}$$

$$68. \quad \text{Fee, PT} = (LI|HI) \xrightarrow{m} \text{Fee}$$

### value

$$67. \quad wf\_VW(vh) \equiv$$

$$67. \quad \forall hll:(HI|LI)^* \cdot hll \in \mathbf{rng} \text{ } vh$$

$$67. \quad \forall i:\mathbf{Nat} \cdot \{i, i+1\} \subseteq \mathbf{inds} \text{ } hll \Rightarrow$$

$$67. \quad is\_HI(hll(i)) \wedge is\_LI(hll(i+1)) \vee is\_LI(hll(i)) \wedge is\_HI(hll(i+1))$$

69. The monitor behaviour non-deterministically externally alternates between

(a) input of messages from vehicles

i. either when entering a link

in which case the vehicle history is updated with that link's identifier (for that vehicle),

ii. or when entering a hub

in which case the vehicle history is updated with that hub's identifier (for that vehicle).

(b) and accepting inquiries and requests relating vehicle histories and fees (designated by (...) below).

## value

69. monitor:  $PT \rightarrow VH \rightarrow \mathbf{in\ m\_ch\ Unit}$

69. monitor(pt)(vh)  $\equiv$

69(a). (**case** m\_ch? **of**

69((a))i. leaveH\_enterL(vi,hi,li)  $\rightarrow$  monitor(pt)(vh  $\dagger$  [ vi  $\mapsto$  vh(vi)  $\hat{\langle$  li  $\rangle$  ])

69((a))ii. leaveL\_enterH(vi,li,hi)  $\rightarrow$  monitor(pt)(vh  $\dagger$  [ vi  $\mapsto$  vh(vi)  $\hat{\langle$  hi  $\rangle$  ])

69(a). **end**)

69(b).  $\square$  (...)

- We omit description of other monitor actions (Line 69(b)).

70. Link behaviours maintain a state which records the set of vehicles “currently” on the link.

71. The link behaviour expresses willingness to

(a) accept messages from vehicles

(b) entering links

in which case the “*vehicle  $v_i$  on link*” state has  $v_i$  added, or

(c) leaving links

in which case the “*vehicle  $v_i$  on link*” state has  $v_i$  removed,

(d) where these vehicles range over all fleet vehicles.



**type**

70. VIS = VI-set

**value**

71. link: li:LI  $\rightarrow$  L  $\rightarrow$  VIS  $\rightarrow$  **in** cl\_Unitlink1

71. link(li)(l)(vis)  $\equiv$

71(a).  $\square$  {**let** m = cl\_vl[vi,li']? **in assert:** li'=li

71(a). **case** m **of**

71(b). enterL  $\rightarrow$  link(li)(l)(vis  $\cup$  {vi})

71(c). leaveL  $\rightarrow$  link(li)(l)(vis  $\setminus$  {vi})

71(d). **end** | vi:V.v  $\in$  xtr\_VIs(vs) **end**}

- We leave it to the reader to suggest a *hub* behaviour description.



---

## End Lecture 5: Describing Domain Entities. Part II: Behaviours

---