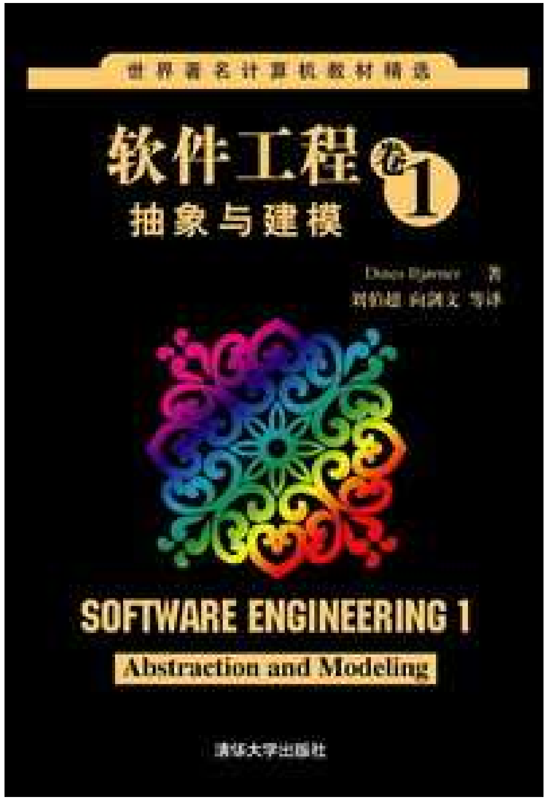

HAD A GOOD LUNCH ?

Begin of Lecture 6: First Session — Calculus I

Part and Material Discoverers

FM 2012 Tutorial, Dines Bjørner, Paris, 28 August 2012

Tutorial Schedule

- **Lectures 1–2** 9:00–9:40 + 9:50–10:30

- **Lectures 3–5** 11:00–11:15 + 11:20–11:45 + 11:50–12:30

Lunch **12:30–14:00**

- **Lectures 6–7** 14:00–14:40 + 14:50–15:30

- **Lectures 8–9** 16:00–16:40 + 16:50–17:30

11. Towards a Calculus of Domain Discoverers

- The ‘towards’ term is significant.
- We are not presenting
  - a “ready to serve”
  - comprehensive,
  - tested and tried

  calculus.
- We hope that the one we show you is interesting.
- It is, we think, the first time such a calculus is presented.

- By a **domain description calculus**
  - or, as we shall also call it,
    - either a **domain discovery calculus**
    - or a **calculus of domain discoverers**

  we shall understand an **algebra**, that is,
  - a set of meta-operations and
  - a pair of
    - a fixed domain and
    - a varying **repository**.
- The meta-operations will be outlined in this section.
- The fixed domain is of the kind of domains alluded to in the previous section of this tutorial.
- The varying repository contains fragments of a description of the fixed domain.

- The meta-operators are referred to as
  - either **domain analysis meta-functions**
  - or **domain discovery meta-functions**.
- The former are carried out by the **domain analyser** when inquiring (the domain) as to its properties.
- The latter are carried out by the **domain describer** when deciding upon which descriptions "to go for" !
- The two persons can be the same one **domain engineer**.
- The operators are referred to as meta-functions,
  - or meta-linguistic functions,
  - since they are
    - applied and
    - calculated
  - by humans, i.e., the **domain describer**s.
- They are directives which can be referred to by the **domain describer**s while carrying out their analytic and creative work.

11.1. Introductory Notions

- In order to present the operators of the calculus
  - ⟐ we must clear a few concepts.

11.1.1. Discovery

- By a **domain discovery calculus** we shall understand
  - ⟐ a set of operations (the **domain discoverer**s),
    - ⊚ which when applied to a domain
    - ⊚ by a human agent, the **domain describer**,

    and
    - ⊚ yield **domain description text**s.

- The **domain discoverers** are applied "mentally".
  - That is, not in a mechanisable way.
    - It is not like when procedure calls
    - invoke computations
    - of a computer.
  - But they are applied by the **domain describer**.
  - That person is to follow the ideas laid down for
  - these **domain discoverers**
    - (as they were in the earlier parts of this talk).
  - They serve to guide the **domain engineer**
    - to discoverer the desired domain entities
    - and their properties.
- In this section we shall review an ensemble of (so far) nine **domain discoverers** and (so far) four **domain analysers**.

We list the nine **domain discoverers**.

- [Slide 320] PART_SORTS,
  [Slide 317] MATEREIAL_SORTS,
  [Slide 324] PART_TYPES,
  [Slide 327] UNIQUE_ID,
  [Slide 328] MEREOLOGY,
  [Slide 332] ATTRIBUTES,
  [Slide 341] ACTION_SIGNATURES,
  [Slide 346] EVENT_SIGNATURES and
  [Slide 349] BEHAVIOUR_SIGNATURES.

11.1.2. Analysis

- In order to “apply” these **domain discoverers** certain conditions must be satisfied.
- Some of these condition inquiries can be represented by (so far) four **domain analysers**.
  - [Slide 306] IS_MATERIALS$_{\text{BASED}}$,
    [Slide 308] IS_ATOM,
    [Slide 308] IS_COMPOSITE and
    [Slide 312] HAS_A_CONCRETE_TYPE.

11.1.3. Domain Indexes

- In order to discover, the domain describer must decide on *"where & what in the domain"* to analyse and describe.
- One can, for this purpose, think of the domain as semi-lattice-structured.
  - The root of the lattice is then labelled Δ.
  - Let us refer to the domain as Δ.
  - We say that it has index $\langle\Delta\rangle$.
  - Initially we analyse the usually composite Δ domain to consist of one or more distinctly typed parts p_1:t_1, p_2:t_2, ..., p_m:t_m.
  - Each of these have indexes $\langle\Delta, t_i\rangle$.
  - So we view Δ, in the semi-lattice, to be the join of m sub-semi-lattices whose roots we shall label with t_1, t_2, ..., t_m.

- And so forth for any composite part type t_i, etcetera.
- It may be that any two or more such **sub-semi-lattice root types**, t_{i_j}, t_{i_j}, ..., t_{i_k} designate the same, **shared type** t_{i_x}, that is $\mathsf{t}_{i_j} = \mathsf{t}_{i_j} = \ldots = \mathsf{t}_{i_k} = \mathsf{t}_{i_x}$.
- If so then the k sub-semi-lattices are "collapsed" into one sub-semi-lattice.
- The building of the semi-lattice terminates when one can no longer analyse part types into further sub-semi-lattices, that is, when these part types are atomic.

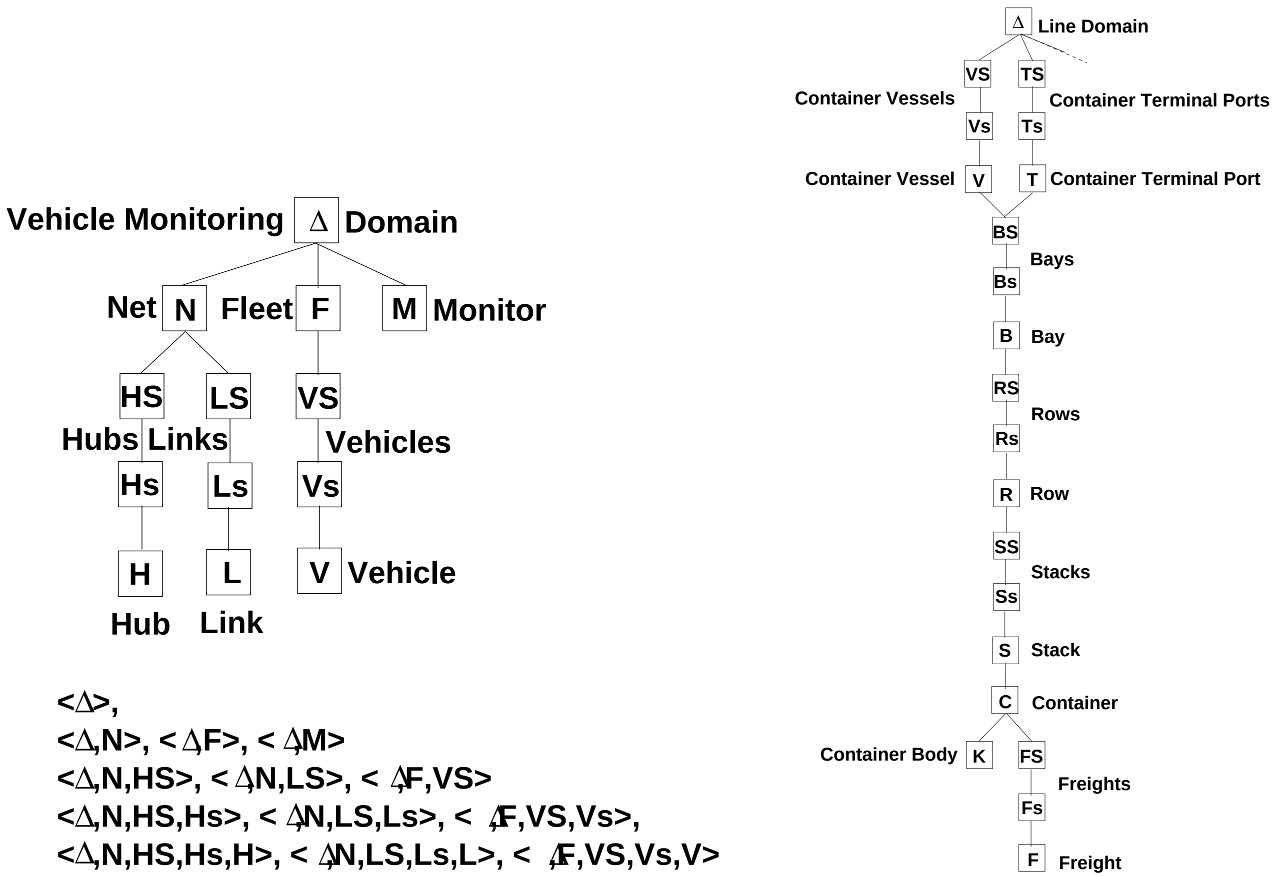

Figure 3: Domain indices

- That is, the roots of the **sub-tree**s of the Δ tree are labelled with **type name**s.
  - Every point in the semi-lattice can be identified by a **domain index**.
    - The root is defined to have index $\langle\Delta\rangle$.
    - The immediate **sub-semi-lattice**s of Δ have domain **index**es $\langle\Delta,\mathsf{t}_1\rangle$, $\langle\Delta,\mathsf{t}_2\rangle$, ..., $\langle\Delta,\mathsf{t}_m\rangle$.
    - And so forth.
    - If $\ell^\frown\langle\mathsf{t}\rangle$ is a prefix of another **domain index**, say $\ell^\frown\langle\mathsf{t},\mathsf{t}'\rangle$, then **t** designates a composite type.

- For every **domain index**, $\ell\frown\langle \mathsf{t}\rangle$,
  that index designates the type **t domain type texts**.
- These texts consists of several sub-texts.
- There are the texts directly related to the **parts**, p:P:
  - the **observer functions**, obs_$\cdots$, if type **t** is composite,
  - the **unique identifier functions**, uid_P,
  - the **mereology function**, mereo_P, and
  - the **attribute functions**, attr_$\cdots$.
  - To the above "add"
    - possible **auxiliary types** and **auxiliary functions**
    - as well as possible **axioms**.

- Then there are the texts related to
  - actions,
  - events, and
  - behaviours

  "based" (primarily) on parts p:P.
- These texts consists of
  - function signatures (for actions, events, and behaviours),
  - function definitions for these, and
  - channel
    - declarations and
    - channel message type definitions

    for behaviours.

We shall soon see examples of the above.

- But not all can be "discovered" by just examining the domain from the point of view of a sub-semi-lattice type.
  - Many interesting action, event and behaviour signatures depend on **domain type texts** designated by "roots" of disjoint sub-trees of the semi-lattice.
  - Each such root has its own **domain index**.
  - Together a **meet** of the semi-lattice is defined by the set of disjoint domain indices: $\{\ell_i, \ell_j, \cdots, \ell_k\}$.
- It is thus that we arrive at a proper semi-lattice structure relating the various entities of the domain rooted in Δ.

- The **domain discoverer**s are therefore provided with arguments:
  - either a single **domain index**, $\mathbb{DOMAIN\_FUNCTION}(\ell)$,
  - or a pair, $\mathbb{DOMAIN\_FUNCTION}(\ell)(\{\ell_i, \ell_j, \cdots, \ell_k\})$,
    - the single **domain index** ℓ and
    - a set of **domain indices**, $\{\ell_i, \ell_j, \cdots, \ell_k\}$

  where $\mathbb{DOMAIN\_FUNCTION}$ is any of the
    - **domain discoverer**s or
    - **domain analyser**s

  listed earlier.

11.1.4. The ℜepository

- We have yet to give the full signature of the **domain discoverer**s and **domain analyser**s.
  - One argument of these meta-functions
    - was parts of the actual domain
    - as designated by the domain indices.
  - Another argument
    - is to be the **ℜepository** of **description text**s
    - being inspected (together with the sub-domain) when
      - analysing that sub-domain and
    - being updated
      - when "generating" the "discovered" description texts.

- We can assume, without loss of generality, that
  - the ℜepository of description texts
  - is the description texts discovered so far.
- The result of domain analysis is either undefined or a truth value. We can assume, without any loss of generality that that result is not recorded.
- The result of domain discovery is either undefined or is a description text consisting of two well-defined fragments:
  - a narrative text, and
  - a formal text.
- Those well-defined texts are "added" to the text of the ℜepository of description texts.
  - For pragmatic reasons,
  - when we explain the positive effect of domain discovery,
  - then we show just this "addition" to the ℜepository.

102. The proper type of the discover functions is therefore:

102. 𝔻𝕀𝕊ℂ𝕆𝕍𝔼ℝ_𝔽𝕌ℕℂ𝕋𝕀𝕆ℕ: $\mathsf{Index} \rightarrow \mathsf{Index\text{-}set} \rightarrow \Re \stackrel{\sim}{\rightarrow} \Re$

- In the following we shall omit the **ℜepository** argument and result.

103. So, instead of showing the discovery function invocation and result as:

103. 𝔻𝕀𝕊ℂ𝕆𝕍𝔼ℝ_𝔽𝕌ℕℂ𝕋𝕀𝕆ℕ$(\ell)(\ell\text{set})(\rho) = \rho'$

- where ρ' incorporates a pair of texts and **RSL** formulas,

104. we shall show the discover function signature, the invocation and the result as:

104. 𝔻𝕀𝕊ℂ𝕆𝕍𝔼ℝ_𝔽𝕌ℕℂ𝕋𝕀𝕆ℕ: $\mathsf{Index} \rightarrow \mathsf{Index\text{-}set} \stackrel{\sim}{\rightarrow} (\mathsf{Narr\_Text} \times \mathsf{RSL\_Text})$

104. 𝔻𝕀𝕊ℂ𝕆𝕍𝔼ℝ_𝔽𝕌ℕℂ𝕋𝕀𝕆ℕ$(\ell)(\ell\text{set})$: $(\mathsf{narr\_text},\mathsf{RSL\_text})$

11.2. Domain Analysers

- Currently we identify four analysis functions.
- As the discovery calculus evolves
  - (through further practice and research)
  - we expect further analysis functions to be identified.

11.2.1. IS_MATERIALS_BASED

- You are reminded of the *Continuous Endurant Modelling* frame on Slide 132.

IS_MATERIALS_BASED

- An early decision has to be made as to whether a domain is significantly based on materials or not:

105. IS_MATERIALS_BASED($\langle\Delta_{\text{Name}}\rangle$).

- If Item 105 holds of a domain Δ_{Name}
  - then the **domain describer** can apply
  - MATERIAL_SORTS (Item 107 on page 317).

Example: 45 Pipelines and Transports: Materials or Parts.

- IS_MATERIALS_BASED($\langle\Delta_{\mathsf{Pipeline}}\rangle$) = **true**.
- IS_MATERIALS_BASED($\langle\Delta_{\mathsf{Transport}}\rangle$)= **false**.

11.2.2. IS_ATOM, IS_COMPOSITE

- During the discovery process
  - discrete part types arise (i.e., the names are yielded)
  - and these may either denote **atomic** or **composite** parts.
- The **domain describer**
  - must now decide as to
  - whether a named, **discrete type is atomic** or **is composite**.

IS_ATOM

- The IS_ATOM analyser serves that purpose:

value

$\mathbb{IS\_ATOM}$: Index $\stackrel{\sim}{\rightarrow}$ **Bool**

$\mathbb{IS\_ATOM}(\ell\widehat{\ }\langle t\rangle) \equiv$ **true** | **false** | **chaos**

- The analysis is undefined for ill-formed indices.

Example: 46 Transport Nets: Atomic Parts (II). We refer to Example 3 (Slide 16).

$\mathbb{IS\_ATOM}(\langle\Delta,\mathsf{N},\mathsf{HS},\mathsf{Hs},\mathsf{H}\rangle)$, $\mathbb{IS\_ATOM}(\langle\Delta,\mathsf{N},\mathsf{LS},\mathsf{Ls},\mathsf{L}\rangle)$

$\sim\mathbb{IS\_ATOM}(\langle\Delta,\mathsf{N},\mathsf{HS},\mathsf{Hs}\rangle)$, $\sim\mathbb{IS\_ATOM}(\langle\Delta,\mathsf{N},\mathsf{LS},\mathsf{Ls}\rangle)$ ■

IS_COMPOSITE

- The IS_COMPOSITE analyser is
  - similarly applied by the domain describer
  - to a part type t
  - to help decide whether t is a composite type.

value

$\text{IS\_COMPOSITE}\colon \text{Index} \xrightarrow{\sim} \textbf{Bool}$

$\text{IS\_COMPOSITE}(\ell^\frown\langle t\rangle) \equiv \textbf{true} \mid \textbf{false} \mid \textbf{chaos}$

Example: 47 Transport Nets: Composite Parts. We refer to Example 3 (Slide 16)

$\mathbb{IS\_COMPOSITE}(\langle\Delta\rangle)$,
$\mathbb{IS\_COMPOSITE}(\langle\Delta,\mathsf{N}\rangle)$
$\mathbb{IS\_COMPOSITE}(\langle\Delta,\mathsf{N},\mathsf{HS},\mathsf{Hs}\rangle)$,
$\mathbb{IS\_COMPOSITE}(\langle\Delta,\mathsf{N},\mathsf{LS},\mathsf{Ls}\rangle)$
$\sim\mathbb{IS\_COMPOSITE}(\langle\Delta,\mathsf{N},\mathsf{HS},\mathsf{Hs},\mathsf{H}\rangle)$,
$\sim\mathbb{IS\_COMPOSITE}(\langle\Delta,\mathsf{N},\mathsf{LS},\mathsf{Ls},\mathsf{L}\rangle)$ ■

11.2.3. HAS_A_CONCRETE_TYPE

- Sometimes we find it expedient
  - to endow a "discovered" sort with a concrete type expression, that is,
  - "turn" a sort definition into a concrete type definition.

HAS_A_CONCRETE_TYPE

106. Thus we introduce the analyser:

106 HAS_A_CONCRETE_TYPE: Index $\stackrel{\sim}{\to}$ **Bool**
106 HAS_A_CONCRETE_TYPE($\ell^\frown\langle t\rangle$): **true** | **false** | **chaos**

Example: 48 Transport Nets: Concrete Types . We refer to Example 3 (Slide 16) while exemplifying four cases:

HAS_A_CONCRETE_TYPE(⟨Δ,N,HS,Hs⟩)
HAS_A_CONCRETE_TYPE(⟨Δ,N,LS,Ls⟩)
∼ HAS_A_CONCRETE_TYPE(⟨Δ,N,HS,Hs,H⟩)
∼ HAS_A_CONCRETE_TYPE(⟨Δ,N,LS,Ls,L⟩) ■

- We remind the listener that
  - it is a decision made by the domain describer
  - as to whether a part type is
    - to be considered a sort or
    - be given a concrete type.
- We shall later cover a domain discoverer related to the positive outcome of the above inquiry.

11.3. Domain Discoverers

- A domain discoverer is a mental tool.
  - It takes a written form shown earlier.
  - It is to be "applied" by a human, the domain describer.
  - The domain describer applies the domain discoverer to a fragment of the domain, as it is: "out there" !

- 'Application' means the following.
  - The **domain describer** examines the domain as directed by the explanation given for the **domain discoverer** — as here, in these lectures.
  - As the brain of the **domain describer** views, examines, analyses, a domain index-designated fragment of the domain,
    - ideas as to which domain concepts to capture arise
    - and these take the form of pairs of **narrative** and **formal texts**.

11.3.1. MATERIAL_SORTS

MATERIAL_SORTS – I/II

107. The MATERIAL_SORTS discovery function applies to a domain, usually designated by $\langle\Delta_{\mathsf{Name}}\rangle$ where Name is a pragmatic hinting at the domain by name.
108. The result of the domain discoverer applying this meta-function is some narrative text
109. and the **type**s of the discovered materials
110. usually affixed a comment
    (a) which lists the "somehow related" part types
    (b) and their related materials observers.

MATERIAL_SORTS II/II

107. MATERIAL_SORTS: $\langle\Delta\rangle \rightarrow$ (**Text** $\times$ RSL)
107. MATERIAL_SORTS($\langle\Delta_{\text{Name}}\rangle$):
108. [narrative text ;
109. **type** M_a, M_b, ..., M_c **materials**
110. **comment**: related part **type**s: P_i, P_j, ..., P_k
110. obs_M_n : $P_m \rightarrow M_n$, ...]
105. **pre**: IS_MATERIALS_BASED($\langle\Delta_{\text{Name}}\rangle$)

Example: 49 Pipelines: Material.

- MATERIAL_SORTS(⟨$\Delta_{\text{Oil Pipeline System}}$⟩):

  [The oil pipeline system is focused on oil ;

  type O **material**

  comment related part type: U, obs_O: U → O]

11.3.2. PART_SORTS

PART_SORTS I/II

111. The part type discoverer PART_SORTS
    (a) applies to a simply indexed domain, $\ell^\frown\langle \mathsf{t}\rangle$,
    (b) where t denotes a composite type, and yields a pair
        i. of narrative text and
        ii. formal text which itself consists of a pair:
            A. a set of type names
            B. each paired with a part (sort) observer.

PART_SORTS II/II

value

111. PART_SORTS: Index $\stackrel{\sim}{\rightarrow}$ (**Text**×RSL)

111(a). PART_SORTS(ℓ⌢⟨t⟩):

111((b))i. [narrative, possibly enumerated texts ;

111((b))iiA. **type** t_1,t_2,...,t_m,

111((b))iiB. **value** obs_t_1:t→t_1,obs_t_2:t→t_2,...,obs_t_m:t→t_m

111(b). **pre**: IS_COMPOSITE(ℓ⌢⟨t⟩)]

Example: 50 Transport: Part Sorts. We apply a concrete version of the above sort discoverer to the vehicle monitoring domain Δ. See Example 36 (Slide 188).

- PART_SORTS($\langle\Delta\rangle$):
  [the vehicle monitoring domain contains three sub-parts: net, fleet and monitor ;
  type N, F, M,
  value obs_N: $\Delta \rightarrow$ N, obs_F: $\Delta \rightarrow$ F, obs_M: $\Delta \rightarrow$ M]
- PART_SORTS($\langle\Delta$,N$\rangle$):
  [the net domain contains two sub-parts: sets of hubs and sets of link ;
  type HS, LS,
  value obs_HS: N $\rightarrow$ HS, obs_LS: N $\rightarrow$ LS]

- PART_SORTS(⟨Δ,F⟩):
  [the fleet domain consists of one sub-domain:
  set of vehicles;
  type VS,
  value obs_VS: F → VS]

11.3.3. PART_TYPES

PART_TYPES I/II

112. The PART_TYPES discoverer applies to a composite sort, t, and yields a pair
    (a) of narrative, possibly enumerated texts [omitted], and
    (b) some formal text:
        i. a type definition, $\mathsf{t}_c = \mathsf{te}$,
        ii. together with the sort definitions of so far undefined type names of te.
        iii. An observer function observes t_c from t.
        iv. The PART_TYPES discoverer is not defined if the designated sort is judged to not warrant a concrete type definition.

PART_TYPES II/II

112. PART_TYPES: Index $\stackrel{\sim}{\rightarrow}$ (**Text**×RSL)
112. PART_TYPES(ℓ⌢⟨t⟩):
112(a). [narrative, possibly enumerated texts ;
112((b))i. **type** t_c = te,
112((b))ii. t_α, t_β, ..., t_γ,
112((b))iii. **value** obs_t_c: t → t_c
112((b))iv. **pre**: HAS_CONCRETE_TYPE(ℓ⌢⟨t⟩)]
112((b))ii. **where:** type expression te contains
112((b))ii. type names t_α, t_β, ..., t_γ

Example: 51 **Transport: Concrete Part Types.** Continuing Examples 36–50 and Example 3 – we omit narrative informal texts.

$\mathbb{PART\_TYPES}(\langle\Delta,\mathsf{F},\mathsf{VS}\rangle)$:
type V, Vs=V-**set**, **value** obs_Vs: VS→Vs
$\mathbb{PART\_TYPES}(\langle\Delta,\mathsf{N},\mathsf{HS}\rangle)$:
type H, Hs=H-**set**, **value** obs_Hs: HS→Hs
$\mathbb{PART\_TYPES}(\langle\Delta,\mathsf{N},\mathsf{LS}\rangle)$:
type L, Ls=L-**set**, **value** obs_Ls: LS→Ls ■

11.3.4. UNIQUE_ID

UNIQUE_ID

113. For every part type **t** we postulate a unique identity analyser function uid_t.

value
113. UNIQUE_ID: Index → (**Text**×RSL)
113. UNIQUE_ID(ℓ⌢⟨t⟩):
113. [narrative, possibly enumerated text ;
113.   **type** ti
113.   **value** uid_t: t → ti]

Example: 52 Transport Nets: Unique Identifiers. Continuing Example 3:

UNIQUE_ID(⟨Δ,HS,Hs,H⟩): **type** H, HI, **value** uid_H→HI
UNIQUE_ID(⟨Δ,LS,Ls,L⟩): **type** L, LI, **value** uid_L→LI ■

11.3.5. MEREOLOGY

- Given a part, p, of type t, the mereology, MEREOLOGY, of that part
  - is the set of all the unique identifiers of the other parts to which part p is part-ship-related
  - as "revealed" by the $\mathsf{mereo\_ti}_i$ functions applied to p.
- Henceforth we omit the otherwise necessary narrative texts.

MEREOLOGY I/II

114. Let type names t_1, t_2, ..., t_n
denote the types of all parts of a domain.

115. Let type names ti_1, ti_2, ..., ti_n[26], be the corresponding
type names of the unique identifiers of all parts of that domain.

116. The mereology analyser MEREOLOGY is a generic function
which applies to a pair of an index and an index set
and yields some structure of unique identifiers.
We suggest two possibilities,
but otherwise leave it to the domain analyser
to formulate the mereology function.

117. Together with the "discovery" of the mereology function
there usually follows some axioms.

MEREOLOGY II/II

type

114. $t_1, t_2, ..., t_n$

115. $t_{idx} = ti_1 \mid ti_2 \mid ... \mid ti_n$

116. MEREOLOGY: Index $\tilde{\to}$ Index-**set** $\tilde{\to}$ (**Text**×RSL)

116. MEREOLOGY$(\ell \frown \langle t \rangle)(\{\ell_i \frown \langle t_j \rangle, ..., \ell_k \frown \langle t_l \rangle\})$:

116. [narrative, possibly enumerated texts ;

116. **either:** {}

116. **or:** **value** mereo_t: t → ti_x

116. **or:** **value** mereo_t: t → ti_x-**set** × ti_y-**set** × ... × ti_x-**set**

117. **axiom** $\mathcal{P}$redicate over values of t′ and t_{idx}]

where none of the ti_x, ti_y, . . . , ti_z are equal to ti.

[26] We here assume that all parts have unique identifications.

Example: 53 Transport Net Mereology. Examples:

- MEREOLOGY(⟨Δ,N,HS,Hs,H⟩)({⟨Δ,N,LS,Ls,L⟩}):
  value mereo_H→LI-**set**
- MEREOLOGY(⟨Δ,N,LS,Ls,L⟩)({⟨Δ,N,HS,Hs,H⟩}):
  value mereo_L→HI-**set**
  axiom see Example 10 Slide 83.

11.3.6. ATTRIBUTES

- A general attribute analyser analyses parts beyond their unique identities and possible mereologies.
  - Part attributes have names.
  - We consider these names to also abstractly name the corresponding attribute types.

ATTRIBUTES I/II

118. Attributes have types.
We assume **attribute type names** to be distict from **part type names**.

119. ATTRIBUTES applies to parts of type **t** and yields a pair of

(a) narrative text and

(b) formal text, here in the form of a pair

i. a set of one or more attribute types, and

ii. a set of corresponding attribute observer functions **attr_at**, one for each attribute sort **at** of **t**.

ATTRIBUTES II/II

type
118. $\mathrm{at} = \mathrm{at}_1 \mid \mathrm{at}_2 \mid \ldots \mid \mathrm{at}_n$
value
119. ATTRIBUTES: Index → (**Text**×RSL)
119. ATTRIBUTES(ℓ⌢⟨t⟩):
119(a). [narrative, possibly enumerated texts ;
119((b))i. **type** at_1, at_2, ..., at_m
119((b))ii. **value** $\mathrm{attr\_at}_1\mathrm{:t}\rightarrow\mathrm{at}_1,\mathrm{attr\_at}_2\mathrm{:t}\rightarrow\mathrm{at}_2,\ldots,\mathrm{attr\_at}_m\mathrm{:t}\rightarrow\mathrm{at}_m$]

- where $m \leq n$

Example: 54 Transport Nets: Part Attributes. We exemplify attributes of composite and of atomic parts — omitting narrative texts:

ATTRIBUTES(⟨Δ⟩):
type Domain_Name, ...
value attr_Domain_Name: $\Delta \rightarrow$ Domain_Name, ...

- where
  - Domain_Name could include *State Roads* or *Rail Net*.
  - etcetera.

ATTRIBUTES(⟨Δ,N⟩):

type

Sub_Domain_Name ex.: *State Roads*
Sub_Domain_Location ex.: *Denmark*
Sub_Domain_Owner ex.: *The Danish Road Directorate*
...
Length ex.: *3.786 Kms.*

value

attr_Sub_Domain_Name: N → Sub_Domain_Name
attr_Sub_Domain_Location: N → Sub_Domain_Location
attr_Sub_Domain_Owner: N → Sub_Domain_Owner
...
attr_Length: N → Length

ATTRIBUTES(⟨Δ,N,LS,Ls,L⟩):
  type LOC, LEN, ...
  value attr_LOC: L → LOC, attr_LEN: L → LEN, ...

ATTRIBUTES(⟨Δ,N,LS,Ls,L⟩)({,⟨Δ,N,HS,Hs,H⟩}):
  type
    LΣ=HI-**set**
    LΩ=LΣ-**set**
  value
    attr_LΣ:L→LΣ
    attr_LΩ:L→LΩ

- where
  - LOC might reveal some Bézier curve[27] representation of the possibly curved three dimensional location of the link in question,
  - LEN might designate length in meters,
  - LΣ designates the state of the link,
  - LΩ designates the space of all allowed states of the link. ■

[27] http://en.wikipedia.org/wiki/Bézier_curve

End of Lecture 6: First Session — Calculus I

Part and Material Discoverers

FM 2012 Tutorial, Dines Bjørner, Paris, 28 August 2012


Software
Engineering 1


Software
Engineering 2


Software
Engineering 3

SHORT BREAK