
**Welcome Back — Thanks !**

# Lecture 2: 11:00–11:40 + 11:50–12:30
# Domains, Discrete Endurants

# 3. Domains

## 3.1. Delineations

We characterise a number of terms.

### 3.1.0.1 Domain

- By a **domain**$_\delta$ 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**.

## 3.1.0.2 Domain Phenomena

- By a **domain phenomenon**$_\delta$ 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 **part**s and **material**s.
- Those phenomena that can be observed of parts and materials
  - can usually be measured
  - and we call them **properties** of these **part**s and those **material**s.

# 3.1.0.3 Domain Entity

- By a **domain entity**$_\delta$ 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.

# 3.1.0.4 Endurant Entity

- We distinguish between
  - endurants and
  - perdurants.
- From Wikipedia:
  - *By an endurant$_\delta$ (also known as a continuant$_\delta$ or a substance$_\delta$) 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.*

# 3.1.0.5 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.*

## 3.1.0.6 Discrete Endurant

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

# 3.1.0.7 Continuous Endurant

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

### 3.1.0.8 Domain Parts and Materials

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

## 3.1.0.9 Domain Analysis

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

# 3.1.0.10 Domain Description

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

## 3.1.0.11 Domain Engineering

- By **domain engineering**$_\delta$ 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 36–105) is an example of the basis for a theory of road traffic systems.

# 3.1.0.12 Domain Science

- By domain science$_\delta$ 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.

# 3.1.0.13 Values & Types

- By a value$_\delta$ we mean some mathematical quantity.
- By a type$_\delta$ we mean
  - a largest set of values,
  - each characterised by the same predicate,
  - such that there are no other values,
  - not members of the set,
  - but which still satisfy that predicate.
- We do not give examples here of the kind of type predicates that may characterise types.

- When we observe a domain we observe **instances** of **entities**;
- but when we describe those instances
  - (which we shall call **value**s)
  - we describe, not the values,
  - but their **type** and **properties**:
    - parts and materials have **type**s and **value**s;
    - actions, events and behaviours, all, have **type**s and **value**s, namely as expressed by their **signature**s; and
    - actions, events and behaviours have **properties**, namely as expressed by their **function definition**s.
- Values are phenomena and types are concepts thereof.

# 3.1.0.14 Discrete Perdurant

- By a **discrete perdurant**$_\delta$ we shall understand
  - a perdurant
  - which we consider as taking place instantaneously,
  - in no time,
  - or where whatever time interval it may take to complete
  - is considered immaterial.

# 3.1.0.15 Continuous Perdurant

- By a continuous perdurant$_\delta$ we shall understand a perdurant whose temporal characteristics are likewise
  - prolonged, without interruption,
  - in an unbroken temporal series or pattern.

## 3.1.0.16 Extensionality

- By extensionality$_\delta$ `Merriam-Webster`[11] means
  - *"something which relates to, or is marked by extension,"*
  - *"that is, concerned with objective reality".*
- Our use basically follows this characterisation:
  - We think of extensionality as a syntactic notion,
  - one that characterises an exterior appearance or form
- We shall therefore think of
  - part types and material types
  - whether parts are atomic or composite, and
  - how composite parts are composed

  as extensional features.

[11] Extensionality. Merriam-Webster.com. 2011, http://www.merriam-webster.com (16 August 2012).

# 3.1.0.17 Intentionality

- By intentionality$_\delta$ Merriam-Webster[12] means:
  - *"done by intention or design"*,
  - *"intended"*,
  - *"of or relating to epistemological intention"*,
  - *"having external reference"*.
- Our use basically follows this characterisation:
  - we think of intentionality as a semantic notion,
  - one that characterises an intention.
- We shall therefore think of
  - part attributess and material attributes

  as intentional features.

[12] Intentionality. Merriam-Webster.com. 2011, http://www.merriam-webster.com (16 August 2012).

# 3.2. Formal Analysis of Entities

- This section is a transcription of
  - Ganter & Wille's [Wille:ConceptualAnalysis1999] *Formal Concept Analysis, Mathematical Foundations*, the 1999 edition, Pages 17–18.

## Definition: 1 Formal Context:

- A formal context$_\delta$ $\mathbb{K} := (\mathbb{E}, \mathbb{I}, \mathbb{Q})$ consists of two sets;
  - $\mathbb{E}$ of entities,
  - $\mathbb{Q}$ of qualities, and a
  - relation $\mathbb{I}$ between $\mathbb{E}$ and $\mathbb{Q}$.
- To express that $e$ is in relation $\mathbb{I}$ to a Quality $q$ we write
  - $e\mathbb{I}q$, or $(e, q) \in \mathbb{I}$ which we read as
  - *"the entity e* ***has*** *the quality q"*.

- Example endurant entities are
  - a specific vehicle,
  - another specific vehicle,
  - etcetera;
  - a specific street segment (link),
  - another street segment,
  - etcetera;
  - a specific road intersection (hub),
  - another specific road intersection,
  - etcetera,
  - a monitor.

  One can also list perdurant entities.
- Example endurant entity qualities are
  - has mobility,
  - has possible velocity,
  - has possible acceleration,
  - has length,
  - has location,
  - has traffic state,
  - can vehicles be sensed,
  - etcetera.

  One can also list perdurant entity qualities.

## Definition: 2 Qualities Common to a Set of Entities:

- For any subset, $s\mathcal{E} \subseteq \mathcal{E}$, of entities we can define

$$\mathcal{DQ}: \ \mathcal{E} \to \mathcal{K} \to \mathcal{Q}$$
$$\mathcal{DQ}(s\mathbb{E})(\mathbb{E},\mathbb{I},\mathbb{Q}) \equiv \{\mathrm{q} \mid \mathrm{q}{:}\mathbb{Q},\ \mathrm{e}{:}\mathbb{E} \cdot \mathrm{e} \in s\mathbb{E} \wedge \ \mathrm{e}\mathbb{I}\mathrm{q}\ \},$$
$$\textbf{pre}: \ s\mathbb{E} \subseteq \mathbb{E}$$

*"the set of qualities common to entities in $\mathcal{SE}$".*

## Definition: 3 Entities Common to a Set of Qualities:

- For any subset, $s\mathcal{Q} \subseteq \mathcal{Q}$, of qualities we can define

$$\mathcal{DE}: \ \mathcal{Q} \to \mathcal{K} \to \mathcal{E}$$
$$\mathcal{DE}(s\mathbb{Q})(\mathbb{E}, \mathbb{I}, \mathbb{Q}) \equiv \{\mathrm{e} \mid \mathrm{e}{:}\mathbb{E},\ \mathrm{q}{:}\mathbb{Q} \cdot \mathrm{q} \in s\mathbb{Q} \wedge \ \mathrm{e}\mathbb{I}\mathrm{q}\ \},$$
$$\textbf{pre}: \ s\mathbb{Q} \subseteq \mathbb{Q}$$

*"the set of entities which have all qualities in $s\mathcal{Q}$".*

# Definition: 4 Formal Concept:

- A formal concept$_\delta$ of a context $\mathbb{K}$ is a pair:
  - $(s\mathbb{Q}, s\mathbb{E})$ where
    - $\mathcal{DQ}(s\mathbb{E})(\mathbb{E}, \mathbb{I}, \mathbb{Q}) = sQ$ and
    - $\mathcal{DE}(s\mathbb{Q})(\mathbb{E}, \mathbb{I}, \mathbb{Q}) = sE$;
  - $s\mathbb{Q}$ is called the extent$_\delta$ of $\mathcal{K}$ and
  - $s\mathbb{E}$ is called the intent$_\delta$ of $\mathcal{K}$.

- Now comes the "crunch":
  - *In the `TripTych` domain analysis*
  - *we strive to find termiiformalconcepts*
  - *and, when we think we have found one,*
  - *we assign a type to it.*

- In mathematical terms it turns out that **formal concepts** are **Galois connections**.
- We can, in other words, characterise **domain analysis** to be the "hunting" for **Galois connections**.
- Or, even more "catchy":
  - **domain types**,
  - whether they be **endurant entity types**
  - or they be **perdurant entity signatures**
  - are **Galois connections**.
- We shall put a **domain engineering** "touch"
  - on **formal concept analysis**
  - in Sects. 4.1.3 and 5.1.

• • •

- The entities referred to by $\mathbb{E}$
  - are the domain entities that we shall deal with in this seminar,
- and the qualities referred to by $\mathbb{Q}$
  - are the mereologies and attributes of discrete endurant entities
  - and the signatures of actions, events and behaviours of discrete perdurant entities;
  - with these terms becoming clearer as we progress through this seminar.

• • •

- Earlier in this section, two signatures were expressed as
  - $\mathcal{DQ}$: $\mathcal{E} \to \mathcal{K} \to \mathcal{Q}$ and
  - $\mathcal{DE}$: $\mathcal{Q} \to \mathcal{K} \to \mathcal{E}$
- The "switch" between using $\mathcal{K}$ for types and $\mathbb{K}$ for values of that type is "explained":
  - $\mathcal{K}$ is the Cartesian type: $\mathcal{E} \times \mathcal{I} \times \mathcal{Q}$, and
  - $\mathbb{K} = (\mathbb{E}, \mathbb{I}, \mathbb{Q})$ is a value of that type.

# 3.3. Discussion

- The crucial characterisation (above) is that of **domain entity** (Slide 107).
  - It is pivotal since all we describe are domain entities.
  - If we get the characterisation wrong we get everything wrong!
  - What might get the characterisation, or its interpretation, wrong is the interpretation of **domain entities**:
    - *"those phenomena that can be observed by*
      - *the human eye or*
      - *touched, for example, by human hands,"*

    and
    - *"manifest domain phenomena or*
    - *domain concepts, i.e., abstractions,*
    - *derived from a domain entities".*

- The whole thing hinges of
  - *what can be described,*
  - *what constitutes a description and*
  - *when is a text a bona fide description.*
- Another set of questions are
  - *of what we have chosen to constitute entities*
  - *which should we describe,*
  - *which not ?*

- Philosophers have dealt with these questions.
  - Recent writings are
    [Badiou1988,BarrySmith1993,ChrisFox2000] and
    [CasatiVarzi2010,HenryLaycock2011,WilsonScpall2012].
  - Going back in time we find
    [LeonardGoodman1940,Kripke1980,BowmanLClarke81].
  - Among the classics we mention
    [Russell1905,Russell1922,RudolfCarnap1928,StanislawLesniewksi1927-19

- We shall only indirectly contribute to this philosophical discussion
  - and do so by presenting the material of this paper;
  - having studied, over the years, fragments of the above cited publications
  - we have concluded with the suggestions of this paper:
    - following the principles, techniques and tools presented here
    - can lead the **domain engineer** to
    - a large class of **domain descriptionss**,
    - large enough for our "immediate future" needs !
- We shall, in the conclusion, return to the questions of
  - what can be described,
  - what constitutes a description and
  - when is a text a bona fide description ?

# 4. Discrete Endurant Entities

- For pragmatics reasons we structure our treatment of discrete endurant domain entities as follows:
  - First we treat the extensional aspects of parts,
  - then their properties: the intentional aspects.
- One could claim that when we say "first parts"
  - we mean fist: a syntactic analysis of parts
  - into atomic and composite parts,
  - etcetera;
- and when we say "then their properties"
  - we mean: then a partial semantic analysis,
  - something which "throws" light over parts,
  - since parts really are distinguishable only through their properties.

# 4.1. Parts

## 4.1.1. What is a Part ?

- By a $\mathsf{part}_\delta$ 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 Item 1(a) Slide 38,
  - F Item 1(b) Slide 38,
  - M Item 1(c) Slide 38,
  - HS Item 2(a) Slide 39,
  - LS Item 2(b) Slide 39,
  - VS Item 3 Slide 40,
  - Vs Item 4(a) Slide 41,
  - V Item 4(b) Slide 41,
  - Hs Item 5 Slide 44,
  - Ls Item 6 Slide 44,
  - H Item 5(a) Slide 44,
  - L Item 6(b) Slide 44.

# 4.1.2. 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**$_\delta$, 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*.

# 4.1.3. 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.

# 4.1.4. Formal Concept Analysis: 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.

# 4.1.5. Part Property Values

- By a part property value$_\delta$, i.e., a property value$_\delta$ 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*$_b$.

# 4.1.6. Part Sorts

- By an **abstract type**$_\delta$, or a **sort**$_\delta$, 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 [Milne1990:RSL:SemFound,Milne1990:RSL:ProofTheory].
      - where these are given properties
      - which we may express in terms of **axiom**s over sort (including **property**) values.
- All of the above examples are examples of **sort**s.

## Example: 9 Part Sorts.

- The discovery of N, F and M was made as a result of examining the domain, $\Delta$, at domain index $\langle\Delta\rangle$;
- HS and LS at domain index $\langle\Delta$,N$\rangle$;
- Hs and H (Ls and L) at domain indexes $\langle\Delta$,HS$\rangle$ ($\langle\Delta$,LS$\rangle$); and
- Vs and V at domain index $\langle\Delta$,VS$\rangle$. ■

# 4.1.7. Atomic Parts

- By an **atomic part**$_\delta$ 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 (Item 5(b) on Slide 43),
  - L (Item 6(b) on Slide 43),
  - V (Item 4(b) on Slide 41) and
  - M (Item 1(c) on Slide 38).
- Implicit tests,
  - at domain indexes,
  - by the domain analyser,
  - for atomicity

  were performed as follows:
  - for H at $\langle \Delta, \mathsf{N,HS,Hs,H} \rangle$;
  - for L at $\langle \Delta, \mathsf{N,LS,Ls,L} \rangle$;
  - for V at $\langle \Delta, \mathsf{F,VS,Vs,V} \rangle$; and
  - for M at $\langle \Delta, \mathsf{M} \rangle$. ■

# 4.1.8. Composite Parts

- By a composite part$_\delta$ 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 (Items 2(a)– 2(b) on Slide 39),
    HS (Item 5 on Slide 43),
    LS (Item 6 on Slide 43),
    Hs (Item 5(a) on Slide 43),
    Ls (Item 6(a) on Slide 43),
    F (Item 3 on Slide 40),
    VS (Item 4(a) on Slide 41),
    Va (Item 4(a) on Slide 41),

  respectively.
- Tests for compostionality of these were implicitly performed;
  - for N at index $\langle \Delta, \mathsf{N} \rangle$;
  - for HS and LS at index $\langle \Delta, \mathsf{N}, \mathsf{HS} \rangle$ and $\langle \Delta, \mathsf{N}, \mathsf{LS} \rangle$;
  - for Hs and Ls at indexes $\langle \Delta, \mathsf{N}, \mathsf{HS}, \mathsf{Hs} \rangle$ and $\langle \Delta, \mathsf{N}, \mathsf{LS}, \mathsf{Ls} \rangle$;
  - for F at index $\langle \Delta, \mathsf{F} \rangle$;
  - for VS at index $\langle \Delta, \mathsf{F}, \mathsf{VS} \rangle$; and
  - for Vs at index $\langle \Delta, \mathsf{F}, \mathsf{VS}, \mathsf{Vs} \rangle$.

# 4.1.9. Part Observers

- By a part observer$_\delta$ or a material observer$_\delta$ we mean
  - a meta-physical operator$_\delta$ (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[13], P,
  - and which yields the sub-part of type B,
  - of the examined part.

[13] 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 (Item 1(a) on Slide 38),
  - **obs_**F (Item 1(b) on Slide 38),
  - **obs_**M (Item 1(c) on Slide 38),
  - **obs_**HS (Item 2(a) on Slide 39),
  - **obs_**LS (Item 2(b) on Slide 39),
  - **obs_**VS (Item 3 on Slide 40),
  - **obs_**Vs (Item 4(a) on Slide 41),
  - **obs_**Hs (Item 5 on Slide 44) and
  - **obs_**Ls (Item 6 on Slide 44),

where we list their "definitions", not their many uses. ■

# 4.1.10. Part Types

- By a concrete type$_\delta$ 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$^\omega$,
    - T = A $\overrightarrow{m}$ 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**: Item 4(a) on Slide 41,
  - Hs = H-**set**: Item 5(a) Slide 44,
  - Ls = L-**set**: Item 6(a) Slide 44.

## Example: 15 Has Composite Types.

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

# 4.2. Part Properties

- (I) By a **property**[14] we mean a pair
  - ◈ a (finite) collection of one or more propositions.
- (II) By an **endurant property**
  - ◈ a **property** which holds of an **endurant** —
  - ◈ which we *model* as a *pair* of a **type** and a **value** (of that type)[15].
- (III) By a **perdurant property**$_\delta$ we shall mean
  - ◈ a **property** which holds of an **perdurant** —
  - ◈ which we, as a minimum, *model* as a *pair* of a **perdurant name** and a **function type**,
  - ◈ that is, as a **function signature**.

[14] By saying 'a property' we definitely mean to distinguish our use of the term from one which refers to legal property such as physical (land) or intangible (legal rights) property.

[15] The **type value** may be a singleton, or lie within a range of discrete values, or lie within a range of continuous values. The ranges may be finite or may be infinite.

- **Property Value Scales:**
  - With **intentional properties** we associate a **property value scale**.
  - By a **property value scale**$_\delta$ of a **part type** we shall mean
    - a **value range** that **part**s of that **type**
    - will have their **property value**s range over.

**Example: 16 Property Value Scales.** We continue Example 4.

- The mereology **property value scale**$_\delta$ for hubs of a net range over finite sets of link identifiers of that net.
- The mereology **property value scale**$_\delta$ for links of a net range over two element sets of hub identifiers for that net.
- The range of location values for any one hub of a net is restricted to not share any cadestral point with any other hub's location value for that net.

- **Discussion:**
  - The notion of 'property' is central to much philosophical discussion; we mention a few (that we have studied):
    - [Chris Fox: The Ontology of Language: Properties, Individuals and Discourse, 2000],
    - [Simons: Parts – A Study in Ontology, 1987] and
    - [Mellor & Oliver (eds.): Properties].[16]

  Their reading has influenced our work.

[16] A reading of the contents listing of [Mellor & Oliver] reveals an interpretation of *parts and properties:*

I Function and Concept, Gottlob Frege
II The World of Universals, Bertrand Russell
III On our Knowledge of Universals, Bertrand Russell
IV Universals, F. P. Ramsey
V On What There Is, W. V. Quine
VI Statements about Universals, Frank Jackson
VII 'Ostrich Nominalism'|'Mirage Realism', Michael Devitt
VIII Against 'Ostrich' Nominalism, D. M. Armstrong
IX On the Elements of Being: I, Donald C. Williams
X The Metaphysic of Abstract Particulars, Keith Campbell
XI Tropes, Chris Daly
XII Properties, D. M. Armstrong
XIII Modal Realism at Work: Properties, David Lewis
XIV New Work for a Theory of Universals, David Lewis
XV Causality and Properties, Sydney Shoemaker
XVI Properties and Predicates, D. H. Mellor.

- The notion of 'property' is also central to the recent notion of **concept analysis** [Ganter and Wille: Formal Concept Analysis — Mathematical Foundations, 1999].
  - Here the term **concept** is understood as *a property of a part*.
  - There is no associated type and value notions such as we have expressed in (II) on Slide 156 and Footnote 15 on Slide 156.
  - We shall have more to say about the relations between our concept of **domain analysis** and Will & Ganter's **concept analysis**
    - starting on Slide 123 and
    - in Item (iii) starting on Slide 460.
- We shall now unravel our 'Property Theory'[17] of parts.

[17] — with apologies to [Turner:1990,Turner:1992,ChrisFox2000].

- 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.

# 4.2.1. Unique Identifiers

## Example: 17 Unique Identifier Functions.

- We have only exemplified the following unique identifier meta-functions and types:
  - **uid_**H, HI Item 7(a) on Slide 47,
  - **uid_**L, LI Item 7(b) on Slide 47 and
  - **uid_**V, VI Item 7(c) on Slide 47.
- We did not find a need for defining unique identifier meta-functions for N, F, M, HS, Hs, LS, Ls, VS, and Vs. ■

# 4.2.1.1 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.

# 4.2.1.2 A Simplification on Specification of Intentional Properties

- So we make a simplification in our treatment of intentional part properties
  - By postulating distinct unique identifiers
  - we are forcing distinctness of parts
  - and can dispense with,
    - that is, do not have to explicitly ascribe such intentional properties
    - whose associated values would then have to differ in order to guarantee distinctness of parts,

# 4.2.1.3 Discussion

- Parts have unique existence.
  - Whether they be spatial or conceptual.
  - Two **manifest parts** cannot overlap spatially.
  - A part is a **conceptual part** if it is an abstraction of a part.
  - Two **conceptual parts** are identical
    - if they have identical properties,
    - that is, abstract the same manifest part,
    - otherwise they are distinct.
  - We shall therefore associate with each part
    - a **unique identifier**,
    - whether we may need to refer to that property or not.
  - There are only **manifest parts** and **conceptual parts**.

# 4.2.1.4 The uid_P Operator

- More specifically we postulate, for every part, p:P, a meta-function:

  73. **uid_**P: P → Π

- where Π is the type of the unique identifiers of parts p:P.

- In practice
  - we "construct" the **unique identifier type name** for parts of type P by "suffixing" I to P, and
  - we explicitly "postulate define" the meta-function shown in Item 73 on the preceding slide.
- How is the uid_PI meta-function "implemented" ?
  - Well, for a **domain description** it suffices to postulate it.
  - If we later were to develop software in support of the described domain, then there are many ways of "implementing" the uid_PIs.

# 4.2.1.5 Constancy of Unique Identifiers — Some Dogmas

- We postulate the following dogmas:
  - parts may be "added" to or "removed" from a domain;
  - parts that are "added" to a domain have **unique identifiers** that are not identifiers of any other part of the history of the domain;
  - parts that are "removed" from a domain will not have their identifiers reused should parts subsequently be "added" to the domain; and
  - domains do not allow for the changing (**upd**ate) of **unique identifier values**.

# 4.2.2. Mereology

- **Mereology:** By mereology$_\delta$ (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*.

- In the following please observe the type font distinctions:
  - *part*, etc., and
  - part (etc.).
- In the above definition of the term mereology
  - we have used the terms
    - *part-hood*,
    - *part* and
    - *whole*
  - in a more general sense than we use the term part.

- In this the "more general sense"
  - we interpret *part* to include,
    - besides what the term part covers in this seminar,
    - also concepts, abstractions, derived from the concept of part.

- That is, by *part* we mean
  - not only manifest phenomena
  - but also intangible phenomena
    - that may be abstract models of parts,
    - or may be (further) abstract models of *part*s.

**Example: 18 Manifest and Conceptual Parts.** We refer to Example 4.

- A net, n:N (Item 1(a) on Slide 38), is a manifest part
- whereas a map, rm:RM (Item 26 on Slide 65), is a *part*. ■

# 4.2.2.1 Extensional and Intentional Part Relations

- Henceforth we shall "merge" the two terms
  - *part* and
  - part

  into one meaning.
- So henceforth the term part shall refer to
  - both manifest, tangible and discrete endurants
  - and to abstractions of these.

- We are forced to do so by necessity.
  - Instead of describing the **manifest phenomena**
  - we are describing conceptual models of these;
- that is,
  - instead of describing **manifest parts**
  - we are describing their **part types** and **part properties**.

- Thus we choose "mereology" to model relations between both
  - parts and
  - *part*s.
- We can thus distinguish between two kinds of such relations:
  - extensional part relations which typically are spatial relations between manifest parts and
  - intentional part relations which typically are conceptual relations between abstract parts.

- 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: 19 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: 20 Monitor and Vehicle Mereologies.** We continue Example 19 on Slide 176.

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

# 4.2.2.2 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:

78. **value** **mereo_**P: P $\rightarrow$ ($\Pi_j$ | $\Pi_k$ | ... | $\Pi_\ell$)**-set**,
  - or of some such signature,
  - one which applies to parts, p:P,
  - and yields unique identifiers
  - of other, "the related", parts —
  - where these "other parts" can be of any part type,
  - including P.

## Example: 21 Road Traffic System Mereology.

- We have exemplified unique part identifier mereologies for
  - hubs, mereo_H Item 8(a) on Slide 48 and
  - links, mereo_L Item 9(a) on Slide 48.

■

**Example: 22 Pipeline Mereology.** This is a somewhat lengthy example from a domain now being exemplified.

- We start by narrating a pipeline domain of pipelines and pipeline units.

79. A pipeline consists of pipeline units.
80. A pipeline unit is either
    (a) a well unit output connected to a pipe or a pump unit;
    (b) a pipe, a pump or a valve unit input and output connected to two distinct pipeline units other than a well;
    (c) a fork unit input connected to a pipeline unit other than a well and output connected to two pipeline units other than wells and sinks;
    (d) a join unit input connected to two pipeline units other than wells and output connected to a a pipeline unit other than a sink; and
    (e) a sink unit input connected to a valve.

**type**
79. PL
**value**
79. **obs_**Us: PL → U-**set**
**type**
80. U = WeU | PiU | PuU | VaU | FoU | JoU | SiU
**value**
80. **uid_**U: U → UI
80. **mereo_**U: U → UI-**set** × UI-**set**
80. i_**mereo_**U,o_**mereo_**U: U → UI-**set**
80. i_UIs(u) ≡ **let** (ius,_) = **mereo_**U(u) **in** ius **end**
80. o_UIs(u) ≡ **let** (_,ous) = **mereo_**U(u) **in** ous **end**
**axiom**
∀ pl:PL,u:U • u ∈ **obs_**Us(pl) ⇒
80(a). **is_**WeU(u) → **card** i_UIs(u)=0 ∧ **card** o_UIs(u)=1,
80(b). (**is_**PiU|**is_**PuU|**is_**VaU)(u) → **card** i_UIs(u)=1=**card** o_UIs(u),
80(c). **is_**FoU(u) → **card** i_UIs(u)=1 ∧ **card** o_UIs(u)=2,
80(d). **is_**JoU(u) → **card** i_UIs(u)=2 ∧ **card** o_UIs(u)=1,
80(e). **is_**SiU(u) → **card** i_UIs(u)=1 ∧ **card** o_UIs(u)=0

- The UI "typed" **value** and **axiom** Items 80 "reveal" the mereology of pipelines.

# 4.2.2.3 Concrete Part Type Mereologies

- Let $\mathsf{A}_i$ and $\mathsf{B}_j$, for suitable $i, j$ denote distinct part types and let $\mathsf{B}_j\mathsf{I}$
- Let there be the following concrete type definitions:

**type**

$$\mathrm{a}_1{:}\mathrm{A}_1 = \mathrm{bs}{:}\mathrm{B}_1\text{-}\mathbf{set}$$
$$\mathrm{a}_2{:}\mathrm{A}_2 = \mathrm{bc}{:}\mathrm{B}_{2_1} \times \mathrm{B}_{2_2} \times ... \times \mathrm{B}_{2_n}$$
$$\mathrm{a}_3{:}\mathrm{A}_3 = \mathrm{bl}{:}\mathrm{B}_3{}^*$$
$$\mathrm{a}_4{:}\mathrm{A}_4 = \mathrm{bm}{:}\mathrm{BI}_4 \xrightarrow[m]{} \mathrm{B}_4$$

- The above part type definitions can be interpreted mereologically:
  - Part $\mathsf{a}{:}\mathsf{A}_1$ has sub-parts $\mathsf{b}_{1_i},\mathsf{b}_{1_2},...,\mathsf{b}_{1_m}{:}\mathsf{B}_1$ of $\mathsf{bs}$ parthood related to just part $\mathsf{a}{:}\mathsf{A}_1$.
  - Parts $\mathsf{a}{:}\mathsf{A}_2$ has sub-parts $\mathsf{b}_{2_1},\mathsf{b}_{2_2},\ldots,\mathsf{b}_{2_m}{:}\mathsf{B}_2$ of $\mathsf{bc}$ parthood related only to parts $\mathsf{a}{:}\mathsf{A}_1$
  - Parts $\mathsf{a}{:}\mathsf{A}_3$ has sub-parts $\mathsf{b}_{3_i}$, for all indices $i$ of the list $\mathsf{b}\ell$, parthood related to parts $\mathsf{a}{:}\mathsf{A}_3$, and to part $\mathsf{b}_{3_{i-1}}$ and part $\mathsf{b}_{3_{i+1}}$, for $1<i<\mathbf{len}\,\mathsf{b}\ell$ by being "neighbours" and also to other $\mathsf{b}_{3_j}$ if the index $j$ is known to $\mathsf{b}_{3_i}$ for $i\neq j$.
  - Parts $\mathsf{a}{:}\mathsf{A}_4$ have all parts $\mathsf{bm}(\mathsf{bi}_j)$ for index $\mathsf{bi}_j$ in the definition set $\mathbf{dom}\,\mathsf{bm}$, be parthood related to $\mathsf{a}{:}\mathsf{A}_4$ and to other such $\mathsf{bm}{:}\mathsf{B}_4$ parts if they know their indexes.

**Example: 23 A Container Line Mereology.** This example brings yet another domain into consideration.

81. Two parts, sets of container vessels, CV-**set**, and sets of container terminal ports, CTP-**set**, are crucial to container lines, CL.
82. Crucial parts of container vessels and container terminal ports are their structures of *bays*, bs:BS.
83. A bay structure consists of an indexed set of *bay*s.
84. A *bay* consists of an indexed set of *rows*
85. A *row* consists of an index set of *stacks*.
86. A *stack* consists of a linear sequence of *container*s.

**type**
81. CP, CVS, CTPS
**value**
81. **obs_**CVS: CL → CVS
81. **obs_**CTPS: CL → CTPS
**type**
81. CVS = CV**-set**
81. CTPS = CTP**-set**
**value**
82. **obs_**BS: (CV|CTP) → BS
**type**
83. BI, BS, B = BI $\overrightarrow{m}$ B
**value**
84. **obs_**RS: B → RS
**type**
84. RI, RS, R = RI $\overrightarrow{m}$ R
**value**
85. **obs_**SS: R → SS
**type**
85. SI, SS, C = SI $\overrightarrow{m}$ S
86. S = C*

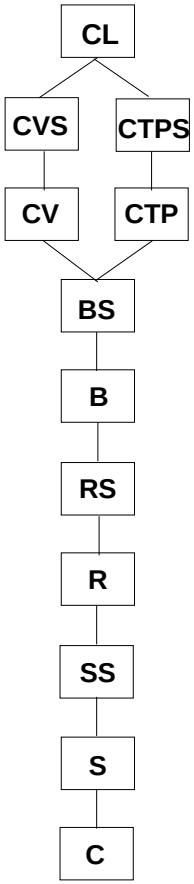



Figure 1: A container line domain index lattice

- In Fig. 1 on the facing slide is shown a container line domain index lattice.
  - At the top ("root") there is the container line domain type name.
  - Immediately below it are the, in this case, two sub-domains (that we consider), CVS and CTPS.
  - For each of these two there are the corresponding CV and CTP sun-domains.
  - For each of these one can observe the container bays, hence, definition-wise, shared sub-domain.
  - It is then defined in terms of a sequence of increasingly more "narrowly" defined sub-domains.
  - The lattice "ends" with the atomic sub-domain of containers, C. ■

# 4.2.2.4 Variability of Mereologies

- The mereology of parts (of type P) may be
  - a constant, i.e., static, or
  - a variable, i.e., dynamic.
- That is, for some, or all, parts of a part type may need to be updated.
  - We express the update of a part mereology as follows:

    87. **value** **upd_mereo_P**: $(\Pi_i|\Pi_i|\ldots|\Pi_i)$**-set** $\rightarrow$ P $\rightarrow$ P
  - where **upd_mereo_P**$(\{\pi_a, \pi_b, \ldots, \pi_c\})$(p)
  - results in a part p′:P where
    - all part properties of p′
      - other than its mereology
      - are as they "were" in p
      - but the mereology of p′ is $\{\pi_a, \pi_b, \ldots, \pi_c\}$.

**Example: 24 Insert Link.** We continue Example 4, Item 42 on Slide 87:

- In the post_link_dis predicate we referred to the undefined link insert function, ins_L.
- We now define that function:

88. The insert_Link action applies to a net, n, and a link, l,
89. and yields a new net, n′.

90. The conditions for a successful insertion are
    (a) that the link, l, is not in the links of net n,
    (b) that the unique identifier of l is not in the set of unique identifiers of the net n, and
    (c) that the mereology of link l has been prepared to be, i.e., is the two element set of unique identifiers of two hubs in net n.
91. The result of a successful insertion is
    (a) that the links of the new net, n′, are those of the previous net, n, "plus" link l;
    (b) that the hubs, "originally" h_a,h_b, connected by l, are only mereo-logically updated to each additional include the unique identifier of l; and
    (c) that all other hubs of n and n′ are unchanged.

88. ins_L: N → L → N
89. ins_L(n)(l) **as** n′
90. **pre**:
90(a). l ∉ **obs_**Ls(**obs_**LS(n))
90(b). ∧ **uid_**L(l) ∉ **in** xtr_LIs(n)
90(c). ∧ **mereo_**L(l) ⊆ xtr_HIs(n)
91. **post**:
91(a). **obs_**Ls(**obs_**LS(n′))=**obs_**Ls(**obs_**LS(n))∪{l}
91. ∧ **let** {hi_a,hi_b}=**mereo_**L(l) **in**
91. **let** {h_a,h_b}={get_H(hi_a)(n),get_H(hi_b)(n)} **in**
91(b). get_H(hi_a)(n′)=**upd_mereo_**H(h_a)(**mereo_**H(h_a)∪{**uid_**L(l)})
91(b). ∧ get_H(hi_b)(n′)=**upd_mereo_**H(h_b)(**mereo_**H(h_b)∪{**uid_**L(l)})
91(c). ∧ **obs_**Hs(**obs_**HS(n))=**obs_**Hs(**obs_**HS(n))\{hi_a,hi_b}∪{h_a′,h_b′} **end end**

- As for the very many other **function definitions** in this seminar
  - we illustrate one form of **function definition annotations**,
  - and not always consistently the same "style".
- We do not pretend that our **function definitions**
  - are novel, let alone a contribution of this seminar;
  - instead we rely on the listener
  - having learnt, more laboriously than we this seminar can muster,
  - an appropriate **function definition narrative style**. ■

• • •

# 4.2.3. Attributes

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

**Example: 25 Road Transport System Part Attributes.** We have exemplified, Example 4, a number of part attribute observation functions:

- attr_LΣ Item 10(a) on Slide 52,
- attr_LΩ Item 10(b) on Slide 52,
- attr_LOC, attr_LEN Item 10(c) on Slide 52,
- attr_HΣ Item 11(a) on Slide 54,
- attr_HΩ Item 11(b) on Slide 54,
- attr_LOC Item 11(c) on Slide 54,
- attr_VP, attr_onL, attr_atH, attr_VEL and attr_ACC Item 13 on Slide 56. ■

# 4.2.3.1 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: 26 Static and Dynamic Attributes.** Continuing Example 4 we have:

- Dynamic attributes:
  - LΣ Item 10(a) on Slide 52;
  - HΣ Item 11(a) on Slide 54;
  - VP, atH, onL Items 12(a)–12((a))ii on Slide 56; and
  - VEL and ACC both Item 13 on Slide 56.
- All other attributes are considered static.

**Example: 27 Concrete Attribute Types.** From Example 4:

- LΣ=(HI×HI) Item 10(a) on Slide 52,
- LΩ=LΣ**-set** Item 10(b) on Slide 52,
- HΣ=(LI×LI)**-set** Item 11(a) on Slide 54 and
- HΩ=HΣ**-set** Item 11(b) on Slide 54.

# 4.2.3.2 The attr_A Operator

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

    92. **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**.

# 4.2.3.3 Variability of Attributes

- Static attributes are constants.
- Dynamic attributes are variables.
- To express the update of any one specific dynamic attributevalue we use the meta-operator:

93. **value** **upd_attr_**A: A → P → P

- where **upd_attr_**A(a)(p) results in a part p′:P where
  - ⋄ all part properties of p′
    - ◦ other than its the attribute value for attribute A
      - ∗ are as they "were" in p
    - ◦ but the attribute value for attribute A is a.

**Example: 28 Setting Road Intersection Traffic Lights.** We refer to Example 4, Items 11(a) (HΣ) and 11(b) (HΩ) on Slide 55.

- The intent of the hub state model
  (a hub state as a set of pairs of unique link identifiers) is
  - that it expresses the possibly empty set of allowed hub traversals,
  - from a link incident upon the hub
  - to a link emanating from that hub.

94. In order to "change" a hub state the set_hub_state action is performed,
95. It takes a hub and a hub state and yields a changed hub.
    The argument hub state must be in the state space of the hub.
    The result of setting the hub state is that the resulting hub has the argument state as its (updated) hub state.

**value**

94. set_hub_state: $\mathrm{H} \to \mathrm{H}\Sigma \to \mathrm{H}$
95. set_hub_state(h)(h$\sigma$) $\equiv$ **upd_attr**_HΣ(h)(h$\sigma$)
95. **pre**: h$\sigma \in$ **attr**_HΩ(h)

- The hub state has not changed if **attr**_HΣ(h) = h$\sigma$. ■

# 4.2.4. Properties and Concepts

- Some remarks are in order.

## 4.2.4.1 Inviolability of Part Properties

- Given any **part** $p$ of **type** $P$
  - one cannot "remove" any one of its properties
  - and still expect the the **part** to be of **type** $P$.
- Properties are what "makes" parts.
- To put the above remark in "context"
  let us review Ganter & Wille's **formal concept analysis**
  [Ganter & Wille: Formal Concept Analysis, 1999].

# 4.2.4.2 Ganter & Wille: Formal Concept Analysis

- This review is based on [Ganter & Wille: Formal Concept Analysis, 1999].

⟐ TO BE WRITTEN

# 4.2.4.3 The Extensionality of Part Attributes

- TO BE WRITTEN

# 4.2.5. Properties of Parts

- The properties of parts and materials are fully captured by
  - (i) the unique part identifiers,
  - (ii) the part mereology and
  - (iii) the full set of part attributes and material attributes
- We therefore postulate a property function
  - when when applied to a part or a material
  - yield this triplet, (i–iii), of properties
  - in a suitable structure.

**type**
  Props = {|PI|**nil**|} × {|(PI-**set**×...×PI-**set**)|**nil**|} × Attrs
**value**
  props: Part|Material → Props

- where
  - Part stands for a part type,
  - Material stands for a material type,
  - PI stand for unique part identifiers and
  - PI-set×...×PI-set for part mereologies.
- The {|...|} denotes a proper specification language sub-type and **nil** denotes the empty type.

# 4.3. States

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

**Example: 29 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
  - indepenently, 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.

# 4.4. An Example Domain: Pipelines

- We close this lecture with a "second main example", albeit "smaller", in text size, than Example 4.
- The domain is that of pipelines.
- The reason we bring this example is the following:
  - Not all **domain endurants** are **discrete domain endurants**.
  - Some domains possess **continuous domain endurants**.
  - We shall call them materials.
  - Two such materials are
    - **liquids**, like `oil` (or `petroleum`), and
    - **gaseous**, like `natural gas`.

- The description of such materials-based domains requires
  - additional description concepts and
  - new description techniques.
- The examples illustrates these new concepts and techniques
- as do the examples of Sect. 6.1.

## Example: 30 Pipeline Units and Their Mereology.

96. A pipeline consists of connected units, u:U.
97. Units have unique identifiers.
98. And units have mereologies, ui:UI:
    (a) pump, pu:Pu, pipe, pi:Pi, and valve, va:Va, units have one input connector and one output connector;
    (b) fork, fo:Fo, [join, jo:Jo] units have one [two] input connector[s] and two [one] output connector[s];
    (c) well, we:We, [sink, si:Si] units have zero [one] input connector and one [zero] output connector.
    (d) Connectors of a unit are designated by the unit identifier of the connected unit.
    (e) The auxiliary sel_UIs_in selector funtion selects the unique identifiers of pipeline units providing input to a unit;
    (f) sel_UIs_out selects unique identifiers of output recipients.

**type**
96. U = Pu | Pi | Va | Fo | Jo | Si | We
97. UI

**value**
97. uid_U: U → UI
98. mereo_U: U → UI-**set** × UI-**set**
98. wf_mereo_U: U → **Bool**
98. wf_mereo_U(u) ≡
98(a). **let** (iuis,ouis) = **mereo_**U(u) **in**
98(a). is_(Pu|Pi|Va)(u) → **card** iusi = 1 = **card** ouis,
98(b). is_Fo(u) → **card** iuis = 1 ∧ **card** ouis = 2,
98(b). is_Jo(u) → **card** iuis = 2 ∧ **card** ouis = 1,
98(c). is_We(u) → **card** iuis = 0 ∧ **card** ouis = 1,
98(d). is_Si(u) → **card** iuis = 1 ∧ **card** ouis = 0 **end**
98(e). sel_UIs_in: U → UI-**set**
98(e). sel_UIs_in(u) ≡ **let** (iuis,_)=**mereo_**U(u) **in** iuis **end**
98(f). sel_UIs_out: U → UI-**set**
98(f). sel_UIs_out(u) ≡ **let** (_,ouis)=**mereo_**U(u) **in** ouis **end**

## Example: 31 Pipelines: Nets and Routes.

99. A pipeline net consists of several properly connected pipeline units.

    Example 30 on Slide 209 already described pipeline units.

    Here we shall concentrate on their connectedness, i.e., the wellformednes of pipeline nets.

100. A pipeline net is well-formed if

    (a) all routes of the net are **acyclic**, and

    (b) there are a non-empty set of **well-to-sink routes** that connect any well to some sink, and

    (c) all other routes of the net are **embedded** in the well-to-sink routes

**type**
99. PLN′
99. PLN = {| pln:PLN′ · **is_wf_**PLN(pln) |}
**value**
99. **obs_**Us: PLN → U-**set**
100. **is_wf_**PLN: PLN′ → **Bool**
100. **is_wf_**PLN(pln) ≡
100. **let** rs = routes{pln} **in**
100(b). well_to_sink_routes(pln)≠{}
100(c). ∧ embedded_routes(pln) **end**

101. An **acyclic route** is a route where any element occurs at most once.
102. A **well-to-sink route** of a net, **pln**, is a route whose first element designates a **well** in **pln** and whose last element designates a **sink** in **pln**.
103. One non-empty route, $\mathsf{r}'$, is embedded in another route, $\mathsf{r}$ if the latter can be expressed as the concatenation of three routes: $\mathsf{r} = \mathsf{r}''\frown\mathsf{r}'\frown\mathsf{r}'''$ where $\mathsf{r}''$ or $\mathsf{r}'''$ may be empty routes ($\langle\rangle$).

**type**
105. R′ = UI*
100(a). R = {r:R′·is_acyclic(r)}
**value**
100(a). is_acyclic: R → **Bool**
100(a). is_acyclic(r) ≡ ∀ i,j:**Nat**·i≠j∧{i,j}⊆**inds** r⇒r[ i ]≠r[ j ]

100(b). well_to_sink_routes: PLN → R-**set**
100(b). well_to_sink_routes(pln) ≡
100(b). {r|r:R·r ∈ routes(pln) ∧ ∃ we:WE,si:Si ·
100(b). {we,si}⊆**obs**_Us(pln) ⇒ r[ 1 ]=we ∧ r[ **len** r ]=si}

104. One non-empty route, er, is_embedded in another route, r,
   (a) if there are two indices, i, j, into r
   (b) such that the sequence of r elements from and including i to and including j is er.

**value**

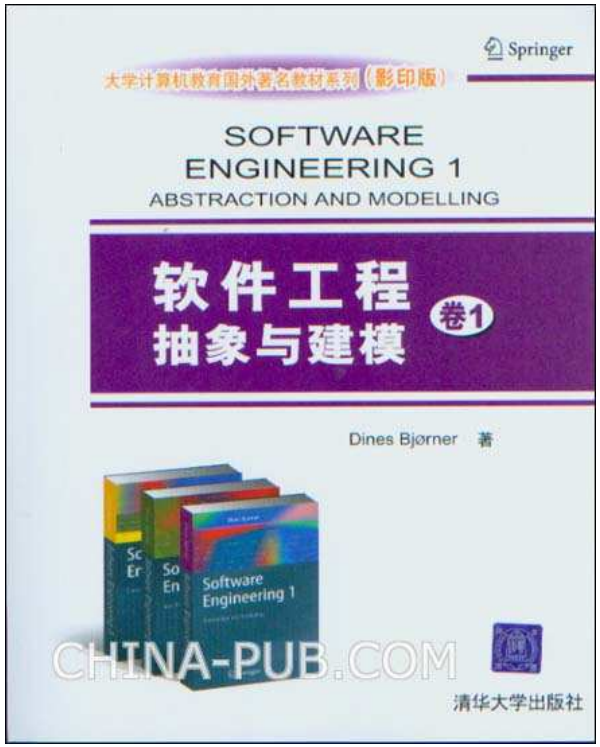
104. is_embedded: R × R → **Bool**
104. is_embedded(er,r) ≡
104(a). ∃ i,j:**Nat**·{i,j}⊆**inds** r
104(b). ⇒ er = ⟨r[k]|k:**Nat** · i≤k≤j⟩
104. **pre**: er≠⟨⟩

105. A route, r, of a pipeline net is a sequence of unique unit identifiers, satisfying the following properties:
    (a) if r[i]=ui$_i$ has ui$_i$ designate a unit, u, of the pipeline then $\langle$ui$_i\rangle$ is a route of the net;
    (b) if r$_i\frown\langle$ui$_i\rangle$ and $\langle$ui$_j\rangle\frown$r$_j$ are routes of the net
        i. where u$_i$ and u$_j$ are the units (of the net) designated by ui$_i$ and ui$_j$
        ii. and ui$_j$ is in the output mereology of u$_i$ and ui$_i$ is in the input mereology of u$_j$
        iii. then r$_i\frown\langle$ui$_i\rangle\frown\langle$ui$_j\rangle\frown$r$_j$ is a route of the net.
    (c) Only such routes that can be constructed by a finite number of "applications" of Items 105(a) and 105(b) are routes.

105. routes: PLN → R-**set**
105. routes(pln) ≡
105(a). **let** rs = {⟨**uid**_UI(u)⟩|u:U•u ∈ **obs**_Us(pln)}
105((b))iii. ∪ { $r_i^\frown\langle ui_i\rangle^\frown\langle ui_j\rangle^\frown r_j$
105(b). | $r_i^\frown\langle ui_i\rangle,\langle ui_j\rangle^\frown r_i$:R • {$r_i^\frown\langle ui_i\rangle,\langle ui_j\rangle^\frown r_j$}⊆rs
105((b))i. ∧ **let** $u_i,u_j$:U•{$u_i,u_i$}⊆**obs**_Us(pln)∧$ui_i$=**uid**_U($u_i$)∧$ui_j$=**uid**_U($u_j$)
105((b))ii. **in** $ui_i$ ∈ iuis($u_j$) ∧ $ui_j$ ∈ ouis($u_i$) **end** }
105(c). **in** rs **end**

- Section 6.1 will continue with several examples
  - Example 43 on Slide 285,
  - Example 44 on Slide 287,
  - Example 45 on Slide 291,
  - Example 46 on Slide 295 and
  - Example 47 on Slide 298

  following up on the two examples of this section.






**See You After Lunch: 14:00 — Thanks !**