

# A Rôle for Mereology in Domain Science and Engineering

– to every mereology there corresponds a  $\lambda$ -expression

Dines Bjørner  
China, December 2012

DTU Informatics, November 16, 2012: 08:38

## 0. Summary

- We give an abstract model of parts and part-hood relations
- of software application domains such as
  - the financial service industry,
  - railway systems,
  - road transport systems,
  - health care,
  - oil pipelines,
  - secure [IT] systems,etcetera.
- We relate this model
  - to axiom systems for mereology, showing satisfiability, and
  - show that for every mereology there corresponds a class of **Communicating Sequential Processes**,
- that is: a  $\lambda$ -expression.

## 1. Introduction

- The term ‘mereology’ is accredited to the Polish mathematician, philosopher and logician Stanisław Leśniewski (1886–1939) who
  - was a nominalist: he rejected axiomatic set theory
  - and devised three formal systems,
    - \* *Protothetic*,
    - \* *Ontology*, and
    - \* *Mereology*
  - as a concrete alternative to set theory.
- In this seminar I shall be concerned with only
  - certain aspects of mereology,
  - namely those that appears most immediately relevant to domain science
  - (a relatively new part of current computer science).

## 1.1. Computing Science Mereology

- “Mereology (from the Greek *μερος* ‘part’) is the theory of parthood relations: of the relations of part to whole and the relations of part to part within a whole”<sup>1</sup>.
- In this talk we restrict ‘parts’ to be those that,
  - firstly, are spatially distinguishable, then,
  - secondly, while “being based” on such spatially distinguishable parts, are conceptually related.
- The relation: “being based”, shall be made clear in this talk.

---

<sup>1</sup>Achille Varzi: Mereology, <http://plato.stanford.edu/entries/mereology/> 2009 and [CasatiVarzi1999]

- Accordingly two parts,  $p_x$  and  $p_y$ , (of a same “whole”) are
  - are either “adjacent”,
  - or are “embedded within” one anotheras loosely indicated in Fig. 1.

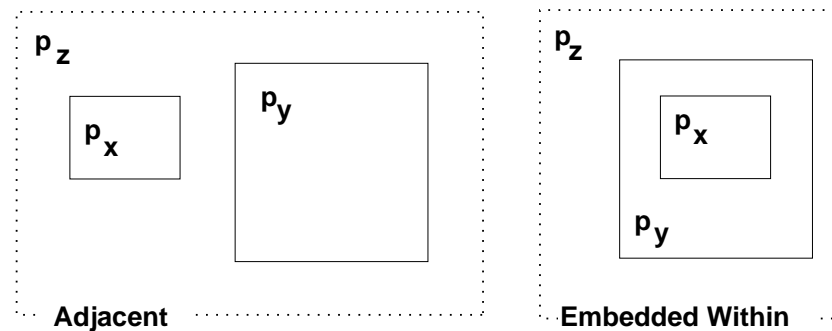


Figure 1: ‘Adjacent’ and ‘Embedded Within’ parts

- ‘Adjacent’ parts
  - are direct parts of a same third part,  $p_z$ ,
  - i.e.,  $p_x$  and  $p_y$  are “embedded within”  $p_z$ ;
  - or one ( $p_x$ ) or the other ( $p_y$ ) or both ( $p_x$  and  $p_y$ ) are parts of a same third part,  $p'_z$  “embedded within”  $p_z$ ;
  - etcetera;

as loosely indicated in Fig. 2 on the next slide.
- or one is “embedded within” the other — etc.  
as loosely indicated in Fig. 2 on the facing slide.

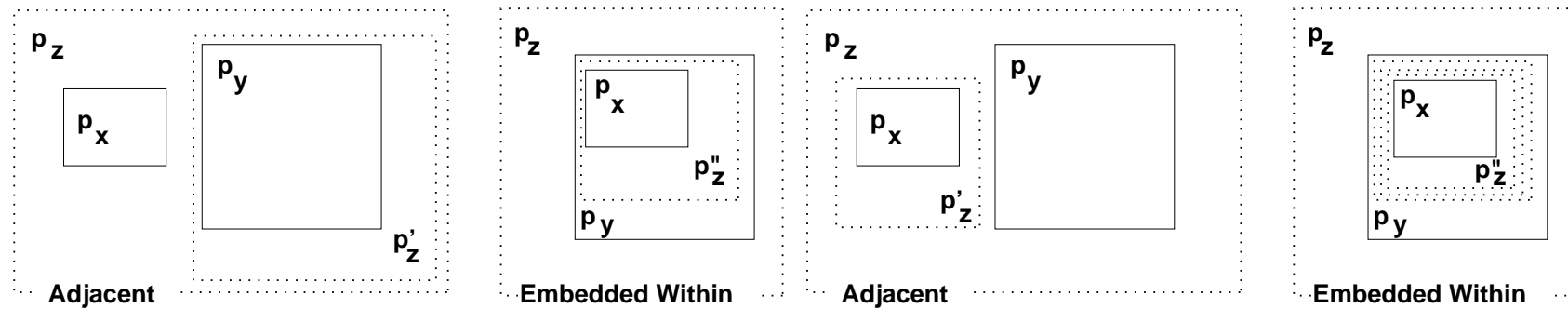


Figure 2: ‘Adjacent’ and ‘Embedded Within’ parts

- Parts, whether adjacent or embedded within one another, can **share** properties.
  - For adjacent parts this sharing seems, in the literature, to be diagrammatically expressed by letting the part rectangles “intersect”.
  - Usually properties are not spatial hence ‘intersection’ seems confusing.
  - We refer to Fig. 3 on the next slide.

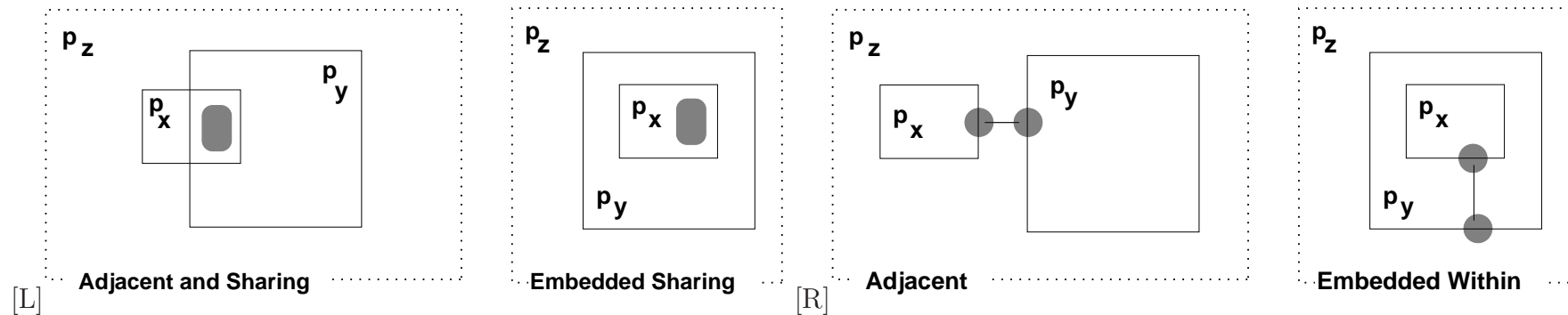


Figure 3: Two models, [L,R], of parts sharing properties

- Instead of depicting parts sharing properties as in Fig. 3[L]eft
  - \* where dashed rounded edge rectangles stands for ‘sharing’,
- we shall (eventually) show parts sharing properties as in Fig. 3[R]ight
  - \* where  $\bullet\text{---}\bullet$  connections connect those parts.



## 1.2. From Domains via Requirements to Software

- One reason for our interest in mereology is that we find that concept relevant to the modelling of domains.
- A derived reason is that we find the modelling of domains relevant to the development of software.
- Conventionally a first phase of software development is that of requirements engineering.
- To us domain engineering is (also) a prerequisite for requirements engineering [Bjørner: Montanari Festschrift (2008); PSI'09 (2009)].

- Thus
  - to properly
    - \* **design** Software we need to
    - \* **understand** its or their Requirements;
  - and to properly
    - \* **prescribe** Requirements one must
    - \* **understand** its Domain.
- To **argue**
  - correctness of Software
  - with respect to Requirements
  - one must usually **make assumptions** about the Domain:
  - $\mathbb{D}, \mathbb{S} \models \mathbb{R}$ .
- Thus **description** of Domains become an indispensable part of Software development.

## 1.3. Domains: Science and Engineering

- **Domain science** is the study and knowledge of domains.
- **Domain engineering** is the practice of “walking the bridge” from domain science to domain descriptions:
  - to **create domain descriptions** on the background of scientific knowledge of domains,
    - \* the specific domain “at hand”, or
    - \* domains in general; and
  - to **study domain descriptions** with a view to broaden and deepen scientific results about domain descriptions.
- This talk is based on the engineering and study of many descriptions, of
 

– air traffic,	– container lines,	– pipelines,	systems,
– banking,	– health care,	– railway systems,	– stock
– commerce <sup>2</sup> ,	– logistics,	– secure [IT]	exchanges,

etcetera.

---

<sup>2</sup>the consumer/retailer/wholesaler/producer supply chain

## 1.4. Contributions of This Talk

- A general contribution is that of providing elements of a domain science.
- Three specific contributions are those of
  - (i) giving a model that satisfies published formal, axiomatic characterisations of mereology;
  - (ii) showing that to every (such modelled) mereology there corresponds a **CSP** program and to conjecture the reverse; and, related to (ii),
  - (iii) suggesting complementing **syntactic** and **semantic** theories of mereology.

## 1.5. Structure of This Talk

We briefly overview the structure of this contribution.

- First, on Slides 15–31, **we loosely characterise how we look at mereologies: “what they are to us !”**.
- Then, on Slides 32–55,  
**we give an abstract, model-oriented specification of a class of mereologies** in the form of composite parts and composite and atomic subparts and their possible connections.
  - The abstract model as well as the axiom system (Sect. 5.) focuses on the **syntax of mereologies**.

- Following that (Slides 56–69),  
**we indicate how the model of** the previous section **satisfies the axiom system of that section.**
- In preparation for the next section Slides 70–92  
**presents characterisations of attributes of parts, whether atomic or composite.**
- Finally Slides 93–102 presents **a semantic model of mereologies,**  
one of a wide variety of such possible models.
  - This one emphasize the possibility of considering parts and subparts as processes and
  - hence a mereology as a system of processes.
- Lastly, Slides 103–106, concludes with some remarks on what we have achieved.

## 2. Our Concept of Mereology

### 2.1. Informal Characterisation

- Mereology, to us, is the study and knowledge
  - about how physical and conceptual parts relate and
  - what it means for a part to be related to another part:
    - \* *being disjoint,*
    - \* *being adjacent,*
    - \* *being neighbours,*
    - \* *being contained properly within,*
    - \* *being properly overlapped with,*
    - \* etcetera.

- By physical parts we mean
  - such spatial individuals
  - which can be pointed to.
- **Examples:**
  - *a road net*  
(*consisting of street segments and street intersections*);
  - *a street segment (between two intersections)*;
  - *a street intersection*;
  - *a road (of sequentially neighbouring street segments of the same name)*
  - *a vehicle*; and
  - *a platoon (of sequentially neighbouring vehicles)*.



- By a conceptual part we mean
  - an abstraction with no physical extent,
  - which is either present or not.
- **Examples:**
  - *a bus timetable*
    - \* *(not as a piece or booklet of paper,*
    - \* *or as an electronic device, but)*
    - \* *as an image in the minds of potential bus passengers; and*
  - *routes of a pipeline, that is, neighbouring sequences of pipes, valves, pumps, forks and joins, for example referred to in discourse: the gas flows through “such-and-such” a route”.*

- The mereological notion of **subpart**, that is: *contained within* can be illustrated by **examples**:
  - *the intersections and street segments are subparts of the road net;*
  - *vehicles are subparts of a platoon; and*
  - *pipes, valves, pumps, forks and joins are subparts of pipelines.*

- The mereological notion of **adjacency** can be illustrated by **examples**. We consider
  - *the various controls of an air traffic system, cf. Fig. 4 on Slide 23, as well as its aircrafts as adjacent within the air traffic system;*
  - *the pipes, valves, forks, joins and pumps of a pipeline, cf. Fig. 9 on Slide 28, as adjacent within the pipeline system;*
  - *two or more banks of a banking system, cf. Fig. 6 on Slide 25, as being adjacent.*

- The mereo-topological notion of **neighbouring** can be illustrated by **examples**:
  - *Some adjacent pipes of a pipeline are neighbouring (connected) to other pipes or valves or pumps or forks or joins, etcetera;*
  - *two immediately adjacent vehicles of a platoon are neighbouring.*

- The mereological notion of **proper overlap** can be illustrated by **examples**
  - some of which are of a general kind:
    - \* *two routes of a pipelines may overlap; and*
    - \* *two conceptual bus timetables may overlap with some, but not all bus line entries being the same;*
  - and some of really reflect adjacency:
    - \* *two adjacent pipe overlap in their connection,*
    - \* *a wall between two rooms overlap each of these rooms — that is, the rooms overlap each other “in the wall”.*

## 2.2. Six Examples

- We shall later
  - present a model that is claimed to abstract essential mereological properties of
    - \* air traffic,
    - \* buildings with installations,
    - \* machine assemblies,
    - \* financial service industry,
    - \* the oil industry and oil pipelines, and
    - \* railway nets.

## 2.2.1. Air Traffic

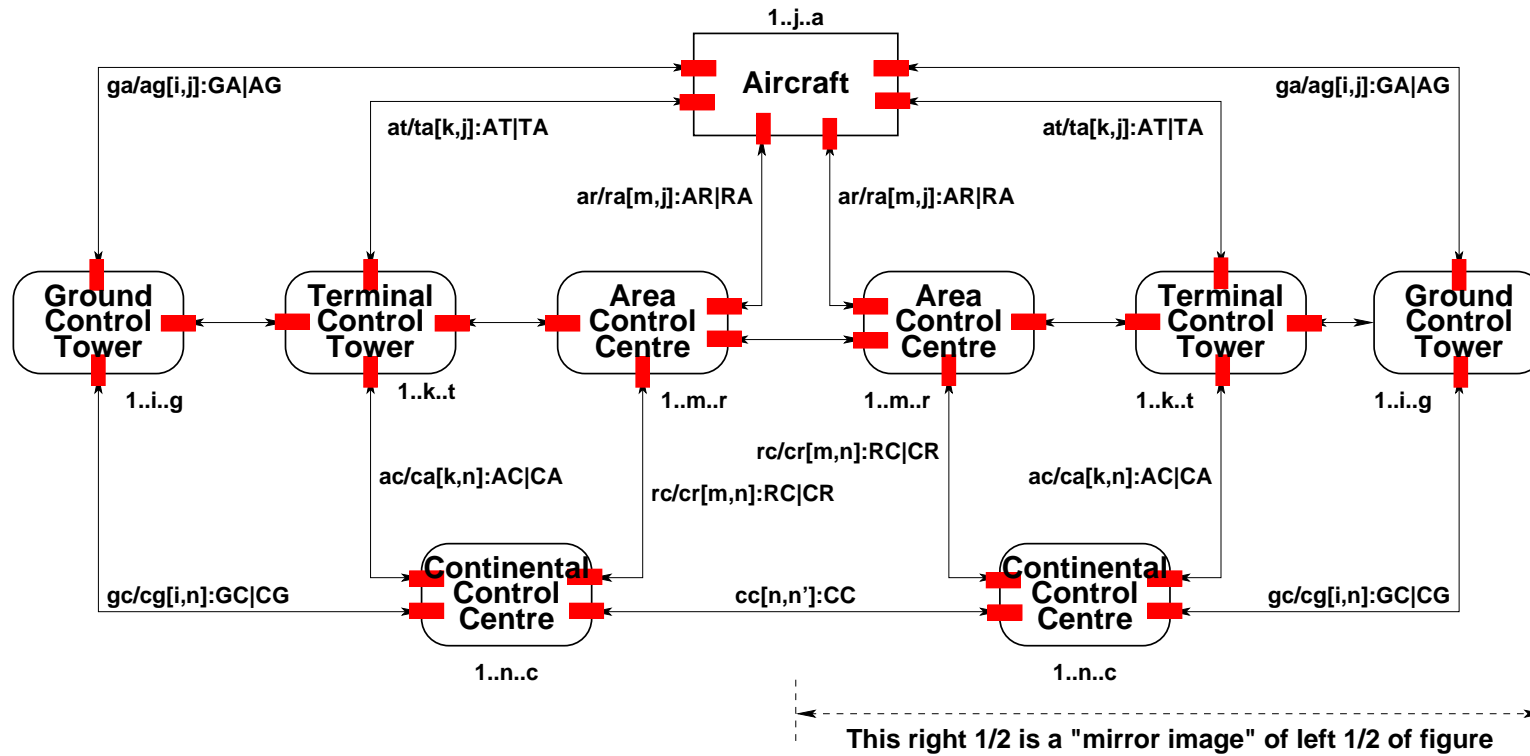


Figure 4: A schematic air traffic system

## 2.2.2. Buildings

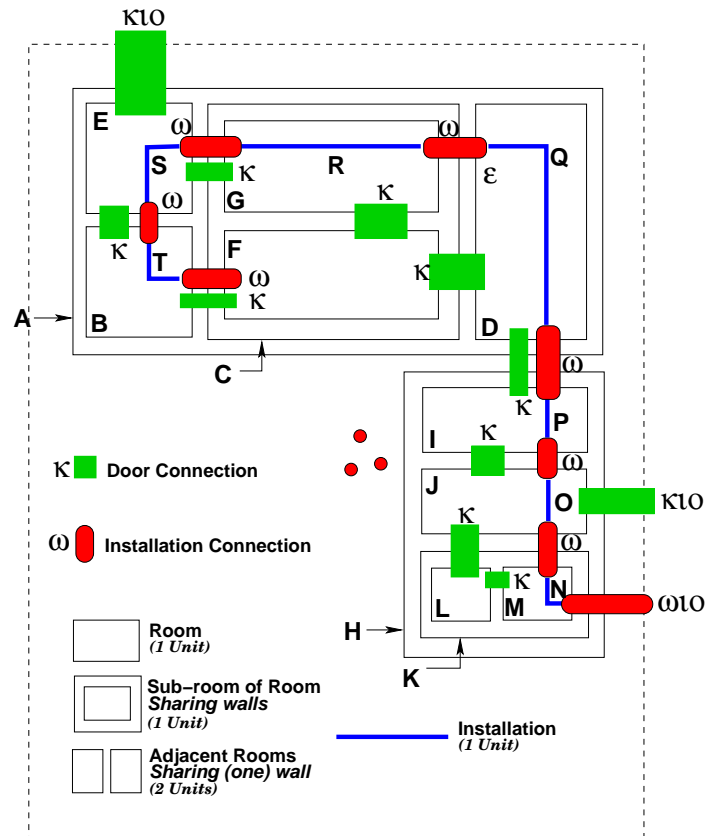


Figure 5: A building plan with installation



## 2.2.3. Financial Service Industry

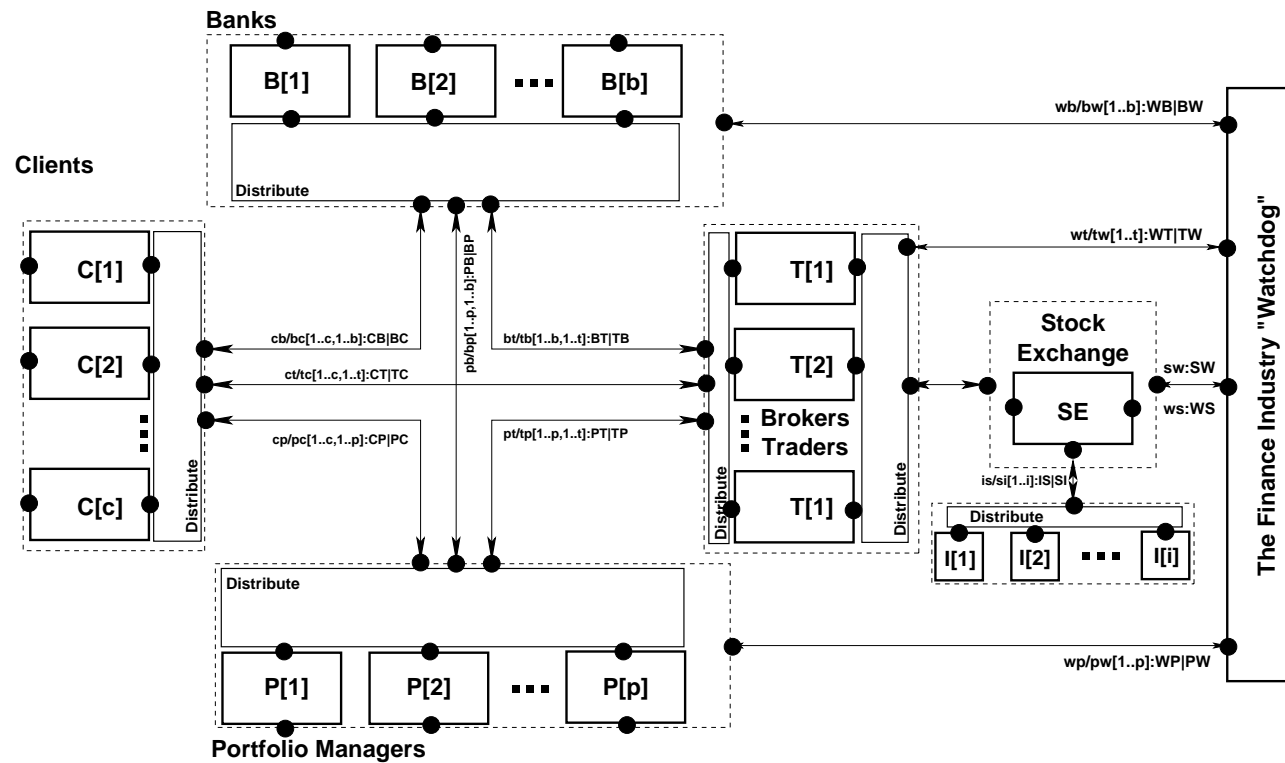


Figure 6: A financial service industry

## 2.2.4. Machine Assemblies

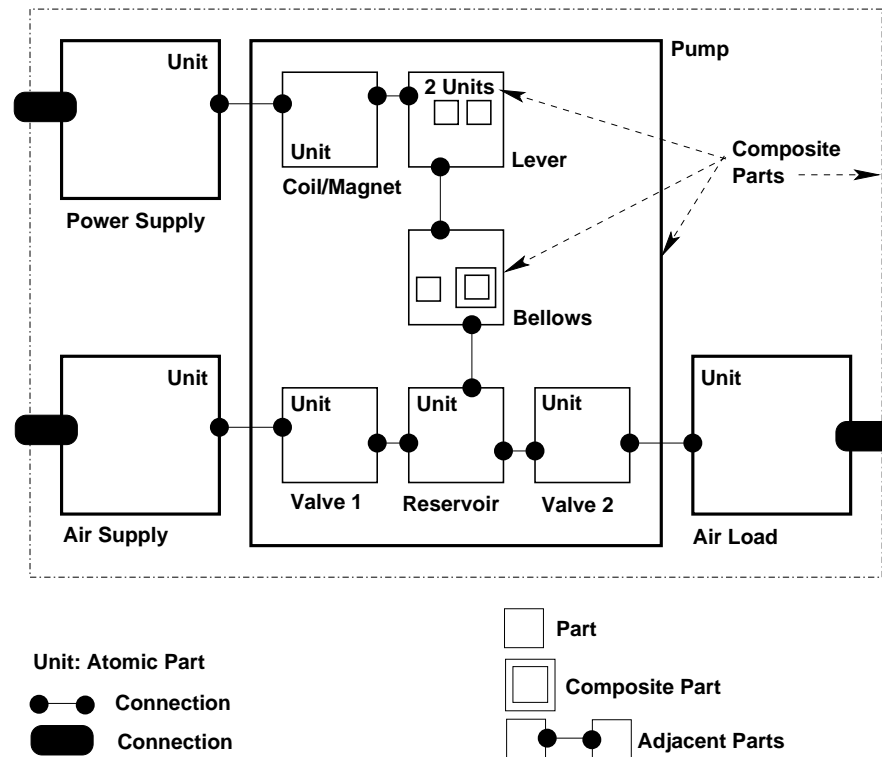


Figure 7: An air pump, i.e., a physical mechanical system



## 2.2.5.2. A Concretised Composite parts

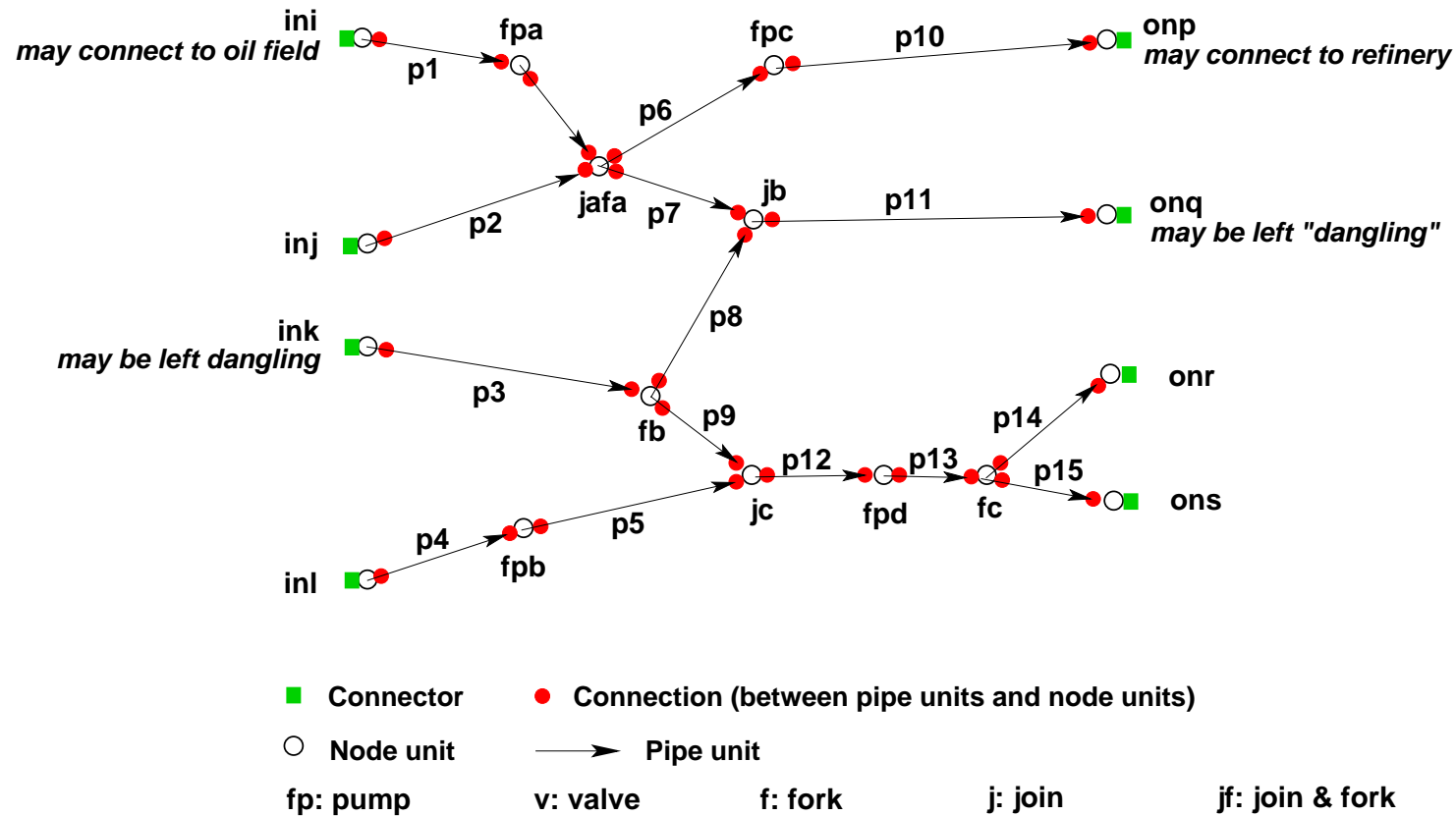


Figure 9: A pipeline system

## 2.2.6. Railway Nets

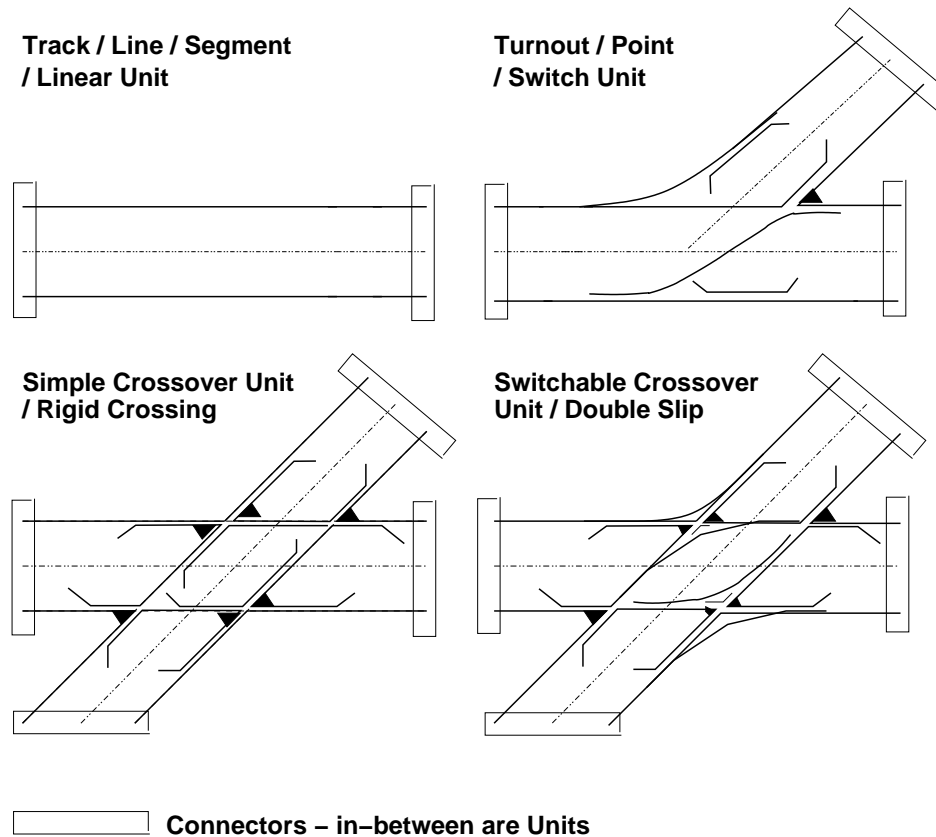


Figure 10: Four example rail units

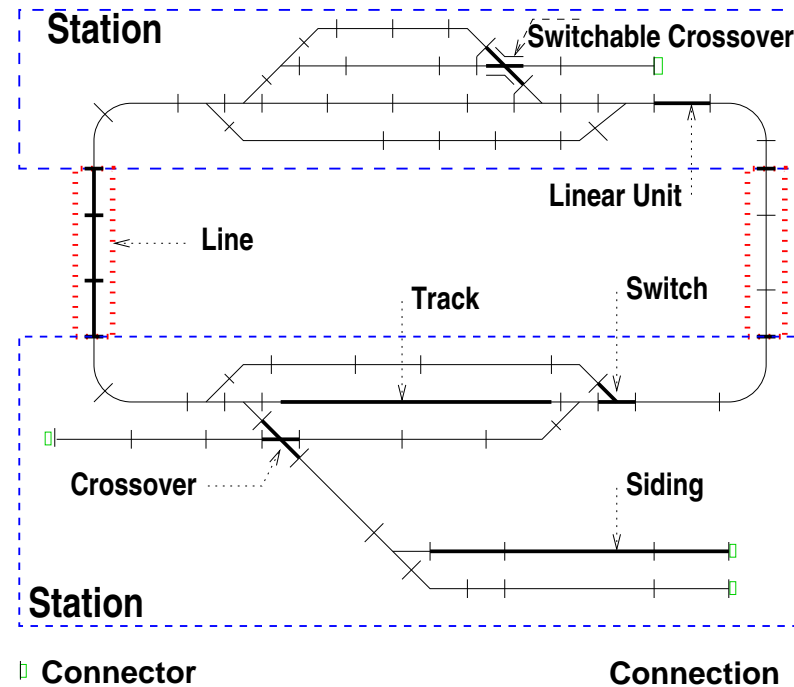


Figure 11: A “model” railway net. An Assembly of four Assemblies:  
 Two stations and two lines; Lines here consist of linear rail units;  
 stations of all the kinds of units shown in Fig. 10 on the preceding slide.  
 There are 66 connections and four “dangling” connectors

## 2.2.7. Discussion

- We have brought these examples only to indicate the issues of
  - a “whole” and atomic and composite parts,
  - adjacency, within, neighbour and overlap relations, and
  - the ideas of attributes and connections.
- We shall make the notion of ‘connection’ more precise in the next section.

### 3. An Abstract, Syntactic Model of Mereologies

- We distinguish between **atomic** and **composite parts**.
  - Atomic parts do not contain separately distinguishable parts.
  - Composite parts contain
    - at least one separately distinguishable part.
  - It is the domain analyser who decides
    - \* what constitutes “the whole”,
      - that is, how parts relate to one another,
    - \* what constitutes parts, and
    - \* whether a part is atomic or composite.
- We refer to the proper parts of a composite part as subparts.



## 3.1. Parts and Subparts

- Figure 12 illustrates composite and atomic parts.
- The *slanted sans serif* uppercase identifiers of Fig. 12 *A1*, *A2*, *A3*, *A4*, *A5*, *A6* and *C1*, *C2*, *C3* are meta-linguistic, that is.
  - they stand for the parts they “decorate”;
  - they are not identifiers of “our system”.

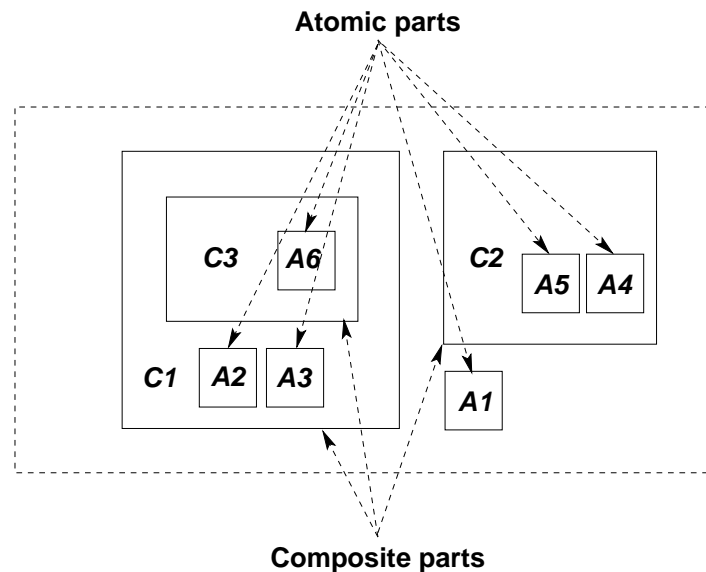


Figure 12: Atomic and composite parts

### 3.1.1. The Model

1. The “whole” contains a set of parts.
2. A part is either an atomic part or a composite part.
3. One can observe whether a part is atomic or composite.
4. Atomic parts cannot be confused with composite parts.
5. From a composite part one can observe one or more parts.

#### type

1.  $W = \mathbf{P\text{-}set}$
2.  $P = A \mid C$

#### value

3.  $is\_A: P \rightarrow \mathbf{Bool}, is\_C: P \rightarrow \mathbf{Bool}$

#### axiom

4.  $\forall a:A, c:C. a \neq c, \text{ i.e., } A \cap C = \{\mid\} \wedge is\_A(a) \equiv \sim is\_C(a) \wedge is\_C(c) \equiv \sim is\_A(c)$

#### value

5.  $obs\_Ps: C \rightarrow \mathbf{P\text{-}set}$     **axiom**  $\forall c:C. obs\_Ps(c) \neq \{\}$

- Fig. 13 and the expressions below illustrate the observer function  $\text{obs\_Ps}$ :

$$- \text{obs\_Ps}(C1) = \{C2, C3, A1\},$$

$$- \text{obs\_Ps}(C2) = \{A3, A4\},$$

$$- \text{obs\_Ps}(C3) = \{A6\}.$$

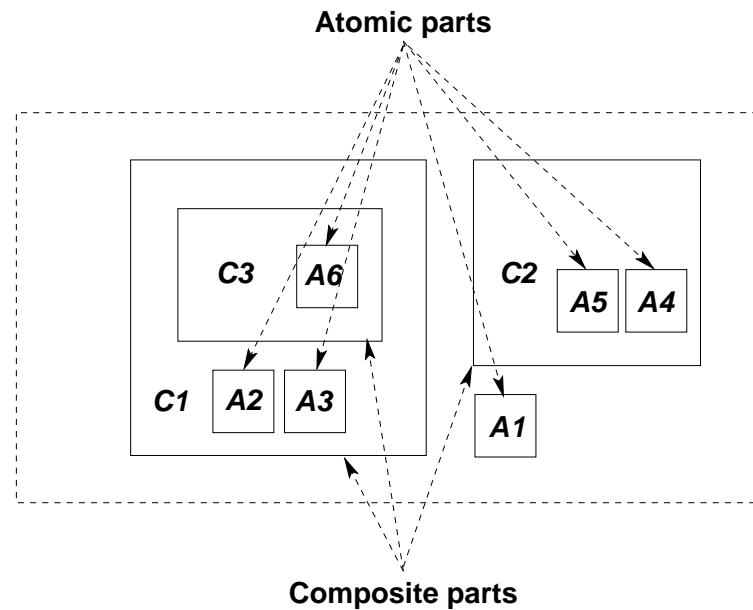


Figure 13: Atomic and composite parts

- Please note that this example is meta-linguistic.

- We can define an auxiliary function.
6. From a composite part, **c**, we can extract all atomic and composite parts
- (a) observable from **c** or
  - (b) extractable from parts observed from **c**.

## value

6.  $\text{xtr\_Ps}: C \rightarrow \text{P-set}$

6.  $\text{xtr\_Ps}(c) \equiv$

6(a). **let**  $\text{ps} = \text{obs\_Ps}(c)$  **in**

6(b).  $\text{ps} \cup \cup \{ \text{obs\_Ps}(c') \mid c':C \cdot c' \in \text{ps} \}$  **end**

## 3.2. 'Within' and 'Adjacency' Relations

### 3.2.1. 'Within'

7. One part,  $p$ , is said to be *immediately within*,  $\text{imm\_within}(p, p')$ , another part,

(a) if  $p'$  is a composite part

(b) and  $p$  is observable in  $p'$ .

**value**

7.  $\text{imm\_within}: P \times P \xrightarrow{\sim} \mathbf{Bool}$

7.  $\text{imm\_within}(p, p') \equiv$

7(a).  $\text{is\_C}(p')$

7(b).  $\wedge p \in \text{obs\_Ps}(p')$

### 3.2.2. 'Transitive Within'

- We can generalise the 'immediate within' property.

8. A part,  $p$ , is transitively within a part  $p'$ ,  $\text{within}(p, p')$ ,

(a) either if  $p$ , is immediately within  $p'$

(b) or if there exists a (proper) composite part  $p''$  of  $p'$  such that  $\text{within}(p'', p)$ .

#### value

8.  $\text{within}: P \times P \xrightarrow{\sim} \mathbf{Bool}$

8.  $\text{within}(p, p') \equiv$

8(a).  $\text{imm\_within}(p, p')$

8(b).  $\forall \exists p'': C \cdot p'' \in \text{obs\_Ps}(p') \wedge \text{within}(p, p'')$

### 3.2.3. 'Adjacency'

9. Two parts,  $p, p'$ , are said to be *immediately adjacent*,  $\text{imm\_adjacent}(p, p')(c)$ , to one another, in a composite part  $c$ , such that  $p$  and  $p'$  are distinct and observable in  $c$ .

#### value

9.  $\text{imm\_adjacent}: P \times P \rightarrow C \xrightarrow{\sim} \mathbf{Bool}$ ,
9.  $\text{imm\_adjacent}(p, p')(c) \equiv p \neq p' \wedge \{p, p'\} \subseteq \text{obs\_Ps}(c)$

### 3.2.4. Transitive 'Adjacency'

10. Two parts,  $p, p'$ , of a composite part,  $c$ , are  $\text{adjacent}(p, p')$  in  $c$
- (a) either if  $\text{imm\_adjacent}(p, p')(c)$ ,
  - (b) or if there are two  $p''$  and  $p'''$  of  $c$  such that
    - i.  $p''$  and  $p'''$  are immediately adjacent parts and
    - ii.  $p$  is equal to  $p''$  or  $p''$  is properly within  $p$  and  $p'$  is equal to  $p'''$  or  $p'''$  is properly within  $p'$

#### value

10.  $\text{adjacent}: P \times P \rightarrow C \xrightarrow{\sim} \mathbf{Bool}$

10.  $\text{adjacent}(p, p')(c) \equiv$

10(a).  $\text{imm\_adjacent}(p, p')(c) \vee$

10(b).  $\exists p'', p''': P .$

10((b))i.  $\text{imm\_adjacent}(p'', p''')(c) \wedge$

10((b))ii.  $((p = p'') \vee \text{within}(p, p'')(c)) \wedge ((p' = p''') \vee \text{within}(p', p''')(c))$



### 3.3. Unique Identifications

- Each physical part can be uniquely distinguished
    - for example by an abstraction of its spatial location.
  - In consequence we also endow conceptual parts with unique identifications.
11. In order to refer to specific parts we endow all parts, whether atomic or composite, with **unique id**entifications.
12. We postulate functions which observe these **unique id**entifications, whether as parts in general or as atomic or composite parts in particular.
13. such that any two parts which are distinct have **unique id**entifications.

**type**

11.  $\Pi$

**value**

12.  $\text{uid}_\Pi: P \rightarrow \Pi$

**axiom**

13.  $\forall p, p': P \cdot p \neq p' \Rightarrow \text{uid}_\Pi(p) \neq \text{uid}_\Pi(p')$

- Figure 14 illustrates the unique identifications of composite and atomic parts.

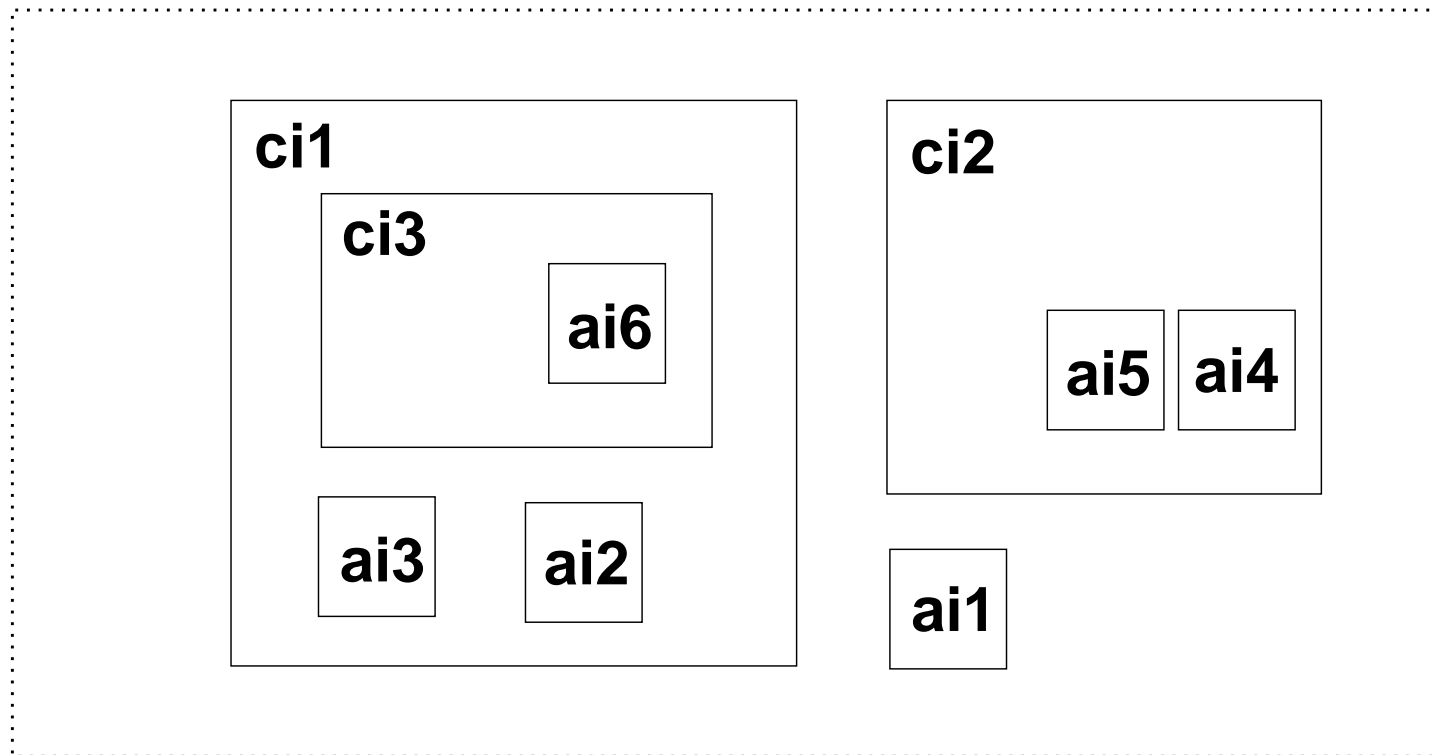


Figure 14:  $ai_j$ : atomic part identifiers,  $ci_k$ : composite part identifiers

- We exemplify the observer function  $\text{obs}_\Pi$  in the expressions below and on Fig. 15:
  - $\text{obs}_\Pi(C1) = ci1$ ,  $\text{obs}_\Pi(C2) = ci2$ , etcetera; and
  - $\text{obs}_\Pi(A1) = ai1$ ,  $\text{obs}_\Pi(A2) = ai2$ , etcetera.

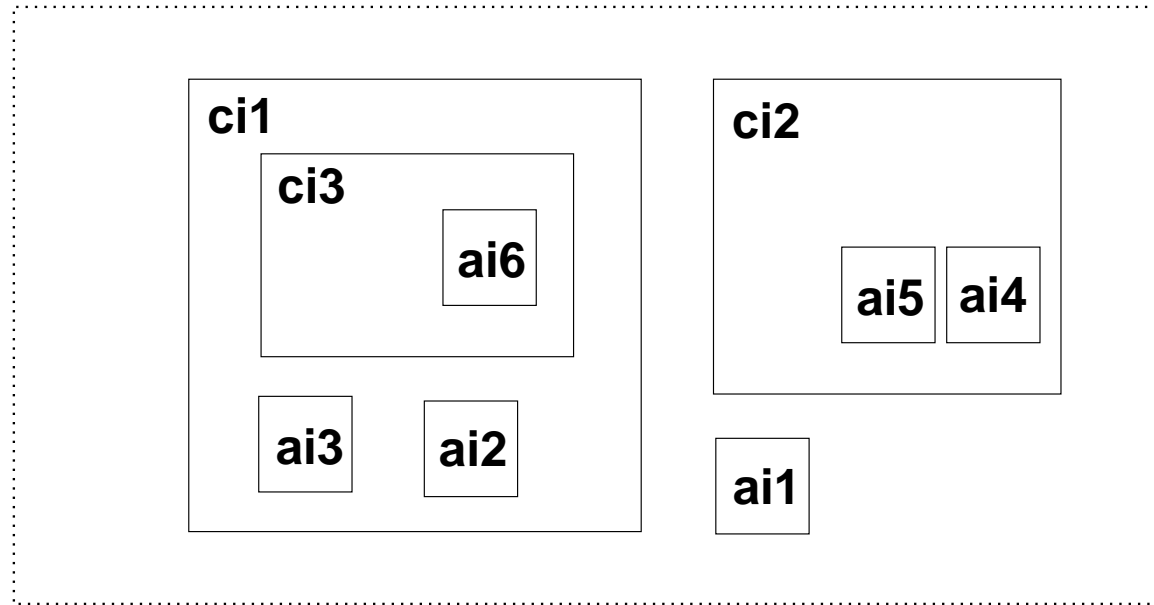


Figure 15:  $ai_j$ : atomic part identifiers,  $ci_k$ : composite part identifiers

14. We can define an auxiliary function which extracts all part identifiers of a composite part and parts within it.

**value**

14.  $\text{xtr\_}\Pi\text{s}: C \rightarrow \Pi\text{-set}$

14.  $\text{xtr\_}\Pi\text{s}(c) \equiv \{\text{uid\_}\Pi(c)\} \cup \{\text{uid\_}\Pi(p) \mid p:P \cdot p \in \text{xtr\_}\Pi\text{s}(c)\}$

## 3.4. Attributes

- We shall later
  - explain the concept of properties of parts,
  - or, as we shall refer to them, attributes
- For now we just postulate that
  15. parts have sets of attributes,  $\text{atr:ATR}$ , (whatever they are!),
  16. that we can observe attributes from parts, and hence
  17. that two distinct parts may share attributes
  18. for which we postulate a membership function  $\in$ .

### type

15. ATR

### value

16.  $\text{atr\_ATRs}: P \rightarrow \text{ATR-set}$

17.  $\text{share}: P \times P \rightarrow \mathbf{Bool}$

17.  $\text{share}(p, p') \equiv p \neq p' \wedge \exists \text{atr:ATR} \cdot \text{atr} \in \text{atr\_ATRs}(p) \wedge \text{atr} \in \text{atr\_ATRs}(p')$

18.  $\in: \text{ATR} \times \text{ATR-set} \rightarrow \mathbf{Bool}$

## 3.5. Connections

- In order to illustrate other than the **within** and **adjacency** part relations we introduce the notions of connectors and, hence, connections.
- Figure 16 on the facing slide illustrates connections between parts.
- A connector is, visually, a ●—● line that connects two distinct part boxes.

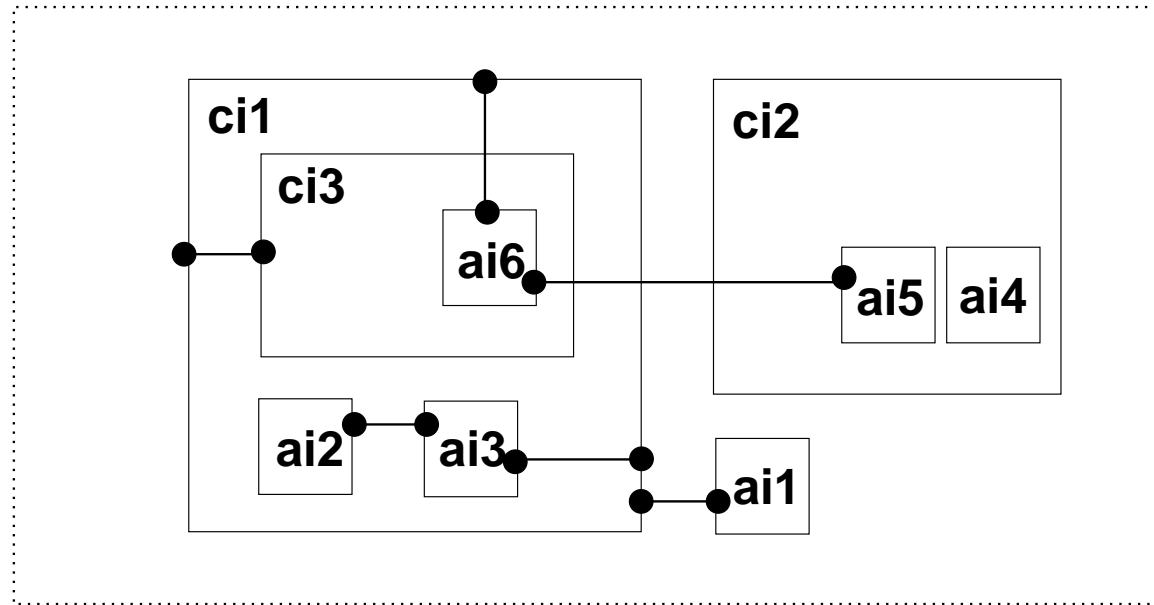


Figure 16: Connectors

19. We may refer to the connectors by the two element sets of the unique identifiers of the parts they connect.

For **example**:

- $\{ci_1, ci_3\}$ ,
- $\{ai_6, ci_1\}$ ,
- $\{ai_6, ai_5\}$  and
- $\{ai_2, ai_3\}$ ,
- $\{ai_3, ci_1\}$ ,
- $\{ai_1, ci_1\}$ .

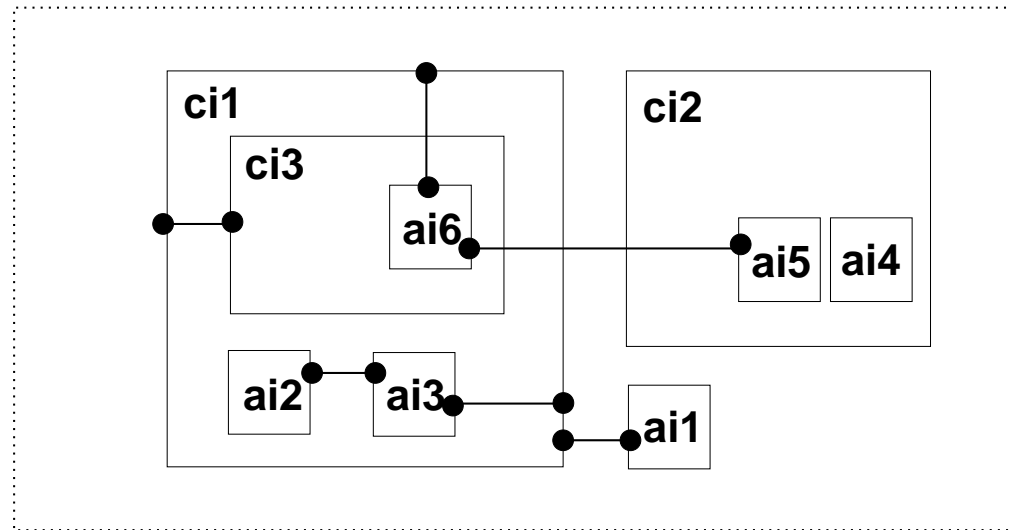


Figure 17: Connectors



20. From a part one can observe the unique identities of the other parts to which it is connected.

**type**

19.  $K = \{ | k : \Pi\text{-set} \cdot \mathbf{card} \ k = 2 \ | \}$

**value**

20.  $\mathbf{mereo\_Ks} : P \rightarrow K\text{-set}$

21. The set of all possible connectors of a part can be calculated.

**value**

21.  $\mathbf{xtr\_Ks} : P \rightarrow K\text{-set}$

21.  $\mathbf{xtr\_Ks}(p) \equiv \{ \{ \mathbf{uid\_}\Pi(p), \pi \} \mid \pi : \Pi \cdot \pi \in \mathbf{mereo\_}\Pi\mathbf{s}(p) \}$

### 3.5.1. Connector Wellformedness

- 22. For a composite part,  $s:C$ ,
- 23. all the observable connectors,  $ks$ ,
- 24. must have their two-sets of part identifiers  
identify parts of the system.

#### value

- 22.  $wf\_Ks: C \rightarrow \mathbf{Bool}$
- 22.  $wf\_Ks(c) \equiv$
- 23.     **let**  $ks = xtr\_Ks(c)$ ,  $\pi s = mereo\_Pis(c)$  **in**
- 24.      $\forall \{\pi', \pi''\} : \Pi\text{-set} \cdot \{\pi', \pi''\} \subseteq ks \Rightarrow$
- 24.      $\exists p', p'' : P \cdot \{\pi', \pi''\} = \{uid\_P(p'), uid\_P(p'')\}$  **end**

### 3.5.2. Connector and Attribute Sharing Axioms

25. We postulate the following axiom:

- (a) If two parts share attributes, then there is a connector between them; and
- (b) if there is a connector between two parts, then they share attributes.

26. The function **xtr\_Ks** (Item 21 on Slide 49) can be extended to apply to **Wholes**.

**axiom**

25.  $\forall w:W.$

25. **let**  $ps = \text{xtr\_Ps}(w)$ ,  $ks = \text{xtr\_Ks}(w)$  **in**

25(a).  $\forall p, p':P \cdot p \neq p' \wedge \{p, p'\} \subseteq ps \wedge \text{share}(p, p') \Rightarrow$

25(a).  $\{\text{uid\_}\Pi(p), \text{uid\_}\Pi(p')\} \in ks \wedge$

25(b).  $\forall \{\text{uid}, \text{uid}'\} \in ks \Rightarrow$

25(b).  $\exists p, p':P \cdot \{p, p'\} \subseteq ps \wedge \{\text{uid}, \text{uid}'\} = \{\text{uid\_}\Pi(p), \text{uid\_}\Pi(p')\}$

25(b).  $\Rightarrow \text{share}(p, p')$  **end**

**value**

26.  $\text{xtr\_Ks}: W \rightarrow \mathbf{K\text{-}set}$

26.  $\text{xtr\_Ks}(w) \equiv \cup \{\text{xtr\_Ks}(p) \mid p:P \cdot p \in \text{obs\_Ps}(p)\}$

### 3.5.3. Sharing

27. When two distinct parts share attributes,  
28. then they are said to be **sharing**:

27. sharing:  $P \times P \rightarrow \mathbf{Bool}$

28.  $\text{sharing}(p, p') \equiv p \neq p' \wedge \text{share}(p, p')$

## 3.6. Uniqueness of Parts

- There is one property of the model of wholes: **W**, Item 1 on Slide 34, and hence the model of composite and atomic parts and their unique identifiers “spun off” from **W** (Item 2 [Slide 34] to Item 25(b) [Slide 51]).
  - and that is that any two parts as revealed in different, say adjacent parts are indeed unique,
  - where we — simplifying — define uniqueness sôlely by the uniqueness of their identifiers.

### 3.6.1. Uniqueness of Embedded and Adjacent Parts

29. By the definition of the **obs\_Ps** function, as applied **obs\_Ps(c)** to composite parts, **c:C**, the atomic and composite subparts of **c** are all distinct and have distinct identifiers (**uiids**: unique immEDIATE identifiers).

value

29. **uiids**:  $C \rightarrow \mathbf{Bool}$

29.  $\mathbf{uiids}(c) \equiv \forall p, p': P. p \neq p' \wedge \{p, p'\} \subseteq \mathbf{obs\_Ps}(c) \Rightarrow \mathbf{card}\{\mathbf{uid}\Pi(p), \mathbf{uid}\Pi(p'), \mathbf{uid}\Pi(c)\} = 3$

30. We must now specify that that uniqueness is “propagated” to parts that are proper parts of parts of a composite part (**uids**: unique identifiers).

30. **uids**:  $C \rightarrow \mathbf{Bool}$

30. **uids**(c)  $\equiv$

30.  $\forall c':C.c' \in \text{obs\_Ps}(c) \Rightarrow \text{uids}(c')$

30.  $\wedge \mathbf{let} \text{ps}'=\text{xtr\_Ps}(c'), \text{ps}''=\text{xtr\_Ps}(c'') \mathbf{in}$

30.  $\forall c'':C.c'' \in \text{ps}' \Rightarrow \text{uids}(c'')$

30.  $\wedge \forall p',p'':P.p' \in \text{ps}' \wedge p'' \in \text{ps}'' \Rightarrow \text{uid\_}\Pi(p') \neq \text{uid\_}\Pi(p'') \mathbf{end}$

## 4. An Axiom System

- Classical axiom systems for mereology focus on just one sort of “things”, namely  $\mathcal{P}$ arts.
  - Leśniewski had in mind, when setting up his mereology to have it supplant set theory.
    - \* So parts could be composite and consisting of other, the sub-parts — some of which would be atomic;
    - \* just as sets could consist of elements which were sets — some of which would be empty.



## 4.1. Parts and Attributes

- In our axiom system for mereology we shall avail ourselves of two sorts:
  - $\mathcal{P}$ arts, and
  - $\mathcal{A}$ tttributes.<sup>3</sup>
  - **type**  $\mathcal{P}, \mathcal{A}$
- $\mathcal{A}$ tttributes are associated with  $\mathcal{P}$ arts.
- We do not say very much about attributes:
  - We think of attributes of parts to form possibly empty sets.
  - So we postulate a primitive predicate,  $\in$ , relating  $\mathcal{P}$ arts and  $\mathcal{A}$ tttributes.
- $\in: \mathcal{A} \times \mathcal{P} \rightarrow \mathbf{Bool}$ .

---

<sup>3</sup>Identifiers  $\mathbf{P}$  and  $\mathbf{A}$  stand for model-oriented types (parts and atomic parts), whereas identifiers  $\mathcal{P}$  and  $\mathcal{A}$  stand for property-oriented types (parts and attributes).

## 4.2. The Axioms

- The axiom system to be developed in this section is a variant of that in [CasatiVarzi1999].
- We introduce the following relations between parts:

part_of:	$\mathbb{P} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 59
proper_part_of:	$\mathbb{PP} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 60
overlap:	$\mathbb{O} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 61
underlap:	$\mathbb{U} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 62
over_crossing:	$\mathbb{OX} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 63
under_crossing:	$\mathbb{UX} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 64
proper_overlap:	$\mathbb{PO} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 65
proper_underlap:	$\mathbb{PU} : \mathcal{P} \times \mathcal{P} \rightarrow \mathbf{Bool}$	Slide 66

- Let  $\mathbb{P}$  denote **part-hood**;  $p_x$  is part of  $p_y$ , is then expressed as  $\mathbb{P}(p_x, p_y)$ .<sup>4</sup>
  - (1) Part  $p_x$  is part of itself (reflexivity).
  - (2) If a part  $p_x$  is part  $p_y$  and, vice versa, part  $p_y$  is part of  $p_x$ , then  $p_x = p_y$  (antisymmetry).
  - (3) If a part  $p_x$  is part of  $p_y$  and part  $p_y$  is part of  $p_z$ , then  $p_x$  is part of  $p_z$  (transitivity).

$$\forall p_x : \mathcal{P} \bullet \mathbb{P}(p_x, p_x) \quad (1)$$

$$\forall p_x, p_y : \mathcal{P} \bullet (\mathbb{P}(p_x, p_y) \wedge \mathbb{P}(p_y, p_x)) \Rightarrow p_x = p_y \quad (2)$$

$$\forall p_x, p_y, p_z : \mathcal{P} \bullet (\mathbb{P}(p_x, p_y) \wedge \mathbb{P}(p_y, p_z)) \Rightarrow \mathbb{P}(p_x, p_z) \quad (3)$$

---

<sup>4</sup>Our notation now is not **RSL** but a conventional first-order predicate logic notation.

- Let  $\mathbb{P}\mathbb{P}$  denote **proper part-hood**.
  - $p_x$  is a proper part of  $p_y$  is then expressed as  $\mathbb{P}\mathbb{P}(p_x, p_y)$ .
  - $\mathbb{P}\mathbb{P}$  can be defined in terms of  $\mathbb{P}$ .
  - $\mathbb{P}\mathbb{P}(p_x, p_y)$  holds if
    - \*  $p_x$  is part of  $p_y$ , but
    - \*  $p_y$  is not part of  $p_x$ .

$$\mathbb{P}\mathbb{P}(p_x, p_y) \triangleq \mathbb{P}(p_x, p_y) \wedge \neg\mathbb{P}(p_y, p_x) \quad (4)$$

- **Overlap**,  $\mathbb{O}$ , expresses a relation between parts.
  - Two parts are said to overlap
    - \* if they have “something” in common.
  - In classical mereology that ‘something’ is parts.
  - To us parts are spatial entities and these cannot “overlap”.
  - Instead they can ‘share’ attributes.

$$\mathbb{O}(p_x, p_y) \triangleq \exists a : \mathcal{A} \bullet a \in p_x \wedge a \in p_y \quad (5)$$

- **Underlap**,  $\mathbb{U}$ , expresses a relation between parts.
  - Two parts are said to underlap
    - \* if there exists a part  $p_z$
    - \* of which  $p_x$  is a part
    - \* and of which  $p_y$  is a part.

$$\mathbb{U}(p_x, p_y) \triangleq \exists p_z : \mathcal{P} \bullet \mathbb{P}(p_x, p_z) \wedge \mathbb{P}(p_y, p_z) \quad (6)$$

- Think of the underlap  $p_z$  as an “umbrella” which both  $p_x$  and  $p_y$  are “under”.

- **Over-cross**,  $\mathbb{O}\mathbb{X}$ ,
  - $p_x$  and  $p_y$  are said to over-cross if
  - $p_x$  and  $p_y$  overlap and
  - $p_x$  is not part of  $p_y$ .

$$\mathbb{O}\mathbb{X}(p_x, p_y) \triangleq \mathbb{O}(p_x, p_y) \wedge \neg \mathbb{P}(p_x, p_y) \quad (7)$$

- **Under-cross**,  $\mathbb{UX}$ ,

- $p_x$  and  $p_y$  are said to under cross if
- $p_x$  and  $p_y$  underlap and
- $p_y$  is not part of  $p_x$ .

$$\mathbb{UX}(p_x, p_y) \triangleq \mathbb{U}(p_x, p_z) \wedge \neg \mathbb{P}(p_y, p_x) \quad (8)$$



- **Proper Overlap**,  $\mathbb{PO}$ , expresses a relation between parts.
  - $p_x$  and  $p_y$  are said to properly overlap if
  - $p_x$  and  $p_y$  over-cross and if
  - $p_y$  and  $p_x$  over-cross.

$$\mathbb{PO}(p_x, p_y) \triangleq \mathbb{OX}(p_x, p_y) \wedge \mathbb{OX}(p_y, p_x) \quad (9)$$

- **Proper Underlap**,  $\mathbb{PU}$ ,
  - $p_x$  and  $p_y$  are said to properly underlap if
  - $p_x$  and  $p_y$  under-cross and
  - $p_x$  and  $p_y$  under-cross.

$$\mathbb{PU}(p_x, p_y) \triangleq \mathbb{UX}(p_x, p_y) \wedge \mathbb{UX}(p_y, p_x) \quad (10)$$

## 4.3. Satisfaction

- We shall sketch a proof that
  - the *model* of the previous section
  - *satisfies* is a model for the *axioms* of this section.
- To that end we first define the notions of
  - *interpretation*,
  - *satisfiability*,
  - *validity* and
  - *model*.

## Interpretation:

- By an interpretation of a predicate we mean
  - an assignment of a truth value to the predicate
  - where the assignment may entail
  - an assignment of values, in general, to the terms of the predicate.

## Satisfiability:

- By the satisfiability of a predicate we mean
  - that the predicate is true for some interpretation.

## Valid:

- By the validity of a predicate we mean
  - that the predicate is true for all interpretations.

## Model:

- By a model of a predicate we mean
  - an interpretation for which the predicate holds.

### 4.3.1. A Proof Sketch

We assign

31.  $\mathbf{P}$  as the meaning of  $\mathcal{P}$
  32.  $\mathbf{ATR}$  as the meaning of  $\mathcal{A}$ ,
  33.  $\mathbf{imm\_within}$  as the meaning of  $\mathbb{P}$ ,
  34.  $\mathbf{within}$  as the meaning of  $\mathbb{PP}$ ,
  35.  $\in_{(\text{of type: } \mathbf{ATR} \times \mathbf{ATR-set} \rightarrow \mathbf{Bool})}$  as the meaning of  $\in_{(\text{of type: } \mathcal{A} \times \mathcal{P} \rightarrow \mathbf{Bool})}$  and
  36.  $\mathbf{sharing}$  as the meaning of  $\mathbb{O}$ .
- With the above assignments it is now easy to prove that
    - the other axiom-operators
    - $\mathbf{U}$ ,  $\mathbf{PO}$ ,  $\mathbf{PU}$ ,  $\mathbf{OX}$  and  $\mathbf{UX}$
    - can be modelled by means of
    - $\mathbf{imm\_within}$ ,  $\mathbf{within}$ ,  $\in_{(\text{of type: } \mathbf{ATR} \times \mathbf{ATR-set} \rightarrow \mathbf{Bool})}$  and  $\mathbf{sharing}$ .

## 5. An Analysis of Properties of Parts

- So far we have not said much about “*the nature*” of parts
  - other than composite parts having one or more subparts and
  - parts having attributes.
- In preparation also for the next section we now take a closer look at the concept of ‘attributes’.
  - We consider three kinds of attributes:
    - \* their unique identifications [uid\_Π]
      - which we have already considered;
    - \* their connections, i.e., their mereology [mereo\_P]
      - which we also considered;
    - \* and their “other” attributes [prop\_P]
      - which we shall refer to as properties.

## 5.1. Mereological Properties

### 5.1.1. An Example

- Road nets,  $n:N$ , consists of
  - a set of street intersections (hubs),  $h:H$ ,
  - uniquely identified by  $hi$ 's (in  $HI$ ), and
  - a set of street segments (links),  $l:L$ ,
  - uniquely identified by  $li$ 's (in  $LI$ ).
- such that
  - from a street segment one can observe a two element set of street intersection identifiers, and
  - from a street intersection one can observe a set of street segment identifiers.

- Constraints between values of link and hub identifiers must be satisfied.
  - The two element set of street intersection identifiers express that the street segment is connected to exactly two existing and distinct street intersections, and
  - the zero, one or more element set of street segment identifiers express that the street intersection is connected to zero, one or more existing and distinct street segments.
- An axiom expresses these constraints.
- We call the hub identifiers of hubs and links, the link identifiers of links and hubs, and their fulfilment of the axiom the connection **mereology**.



**type**

$N, H, L, HI, LI$

**value**

$obs\_Hs: N \rightarrow H\text{-set}, obs\_Ls: N \rightarrow L\text{-set}$

$uid\_HI: H \rightarrow HI, uid\_LI: L \rightarrow LI$

$mereo\_HIs: L \rightarrow HI\text{-set}$  **axiom**  $\forall l:L \cdot \mathbf{card} \ mereo\_HIs(l)=2$

$mereo\_LIs: H \rightarrow LI\text{-set}$

**axiom**

$\forall n:N.$

**let**  $hs=obs\_Hs(n), ls=obs\_Ls(n)$  **in**

$\forall h:H \cdot h \in hs \Rightarrow$

$\forall li:LI \cdot li \in mereo\_LIs(h) \Rightarrow \exists l:L \cdot uid\_LI(l)=li$

$\wedge \forall l:L \cdot l \in ls \Rightarrow$

$\exists h,h':H \cdot \{h,h'\} \subseteq hs \wedge mereo\_HIs(l) = \{uid\_HI(h), uid\_HI(h')\}$

**end**



## 5.1.2. Unique Identifier and Mereology Types

- In general we allow for any embedded (within) part to be connected to any other embedded part of a composite part or across adjacent composite parts.
- Thus we must, in general, allow
  - for a family of part types  $P1, P2, \dots, Pn$ ,
  - for a corresponding family of part identifier types  $\Pi1, \Pi2, \dots, \Pi n$ ,
  - and for corresponding observer **unique identification** and **mereology** functions:

**type**

$$P = P1 \mid P2 \mid \dots \mid Pn$$

$$\Pi = \Pi1 \mid \Pi2 \mid \dots \mid \Pi n$$

**value**

$$\text{uid}_{\Pi j}: P_j \rightarrow \Pi_j \text{ for } 1 \leq j \leq n$$

$$\text{mereo}_{\Pi s}: P \rightarrow \Pi\text{-set}$$

- **Example:** Our example relates to the abstract model given earlier.

37. With each part we associate a unique identifier,  $\pi$ .

38. And with each part we associate a set,  $\{\pi_1, \pi_2, \dots, \pi_n\}$ ,  $n \geq 0$  of zero, one or more other unique identifiers, different from  $\pi$ .

39. Thus with each part we can associate a set of zero, one or more connections, viz.:  $\{\pi, \pi_j\}$  for  $0 \leq j \leq n$ .

**type**

37.  $\Pi$

**value**

37.  $\text{uid}_\Pi: P \rightarrow \Pi$

38.  $\text{mereo}_\Pi s: P \rightarrow \Pi\text{-set}$

**axiom**

38.  $\forall p:P. \text{uid}_\Pi(p) \notin \text{mereo}_\Pi s(p)$

**value**

39.  $\text{xtr}_K s: P \rightarrow K\text{-set}$

39.  $\text{xtr}_K s(p) \equiv$

39. **let**  $(\pi, \pi s) = (\text{uid}_\Pi, \text{mereo}_\Pi s)(p)$  **in**

39.  $\{\{\pi', \pi''\} \mid \pi', \pi'': \Pi. \pi' = \pi \wedge \pi'' \in \pi s\}$  **end**



## 5.2. Properties

- By the properties of a part we mean
  - such properties additional to those of
  - unique identification and mereology.
- Perhaps this is a cryptic characterisation.
  - Parts, whether atomic or composite, are there for a purpose.
  - The unique identifications and mereologies of parts are there to refer to and structure (i.e., relate) the parts.
  - So they are there to facilitate the purpose.
  - The properties of parts help towards giving these parts “their final meaning”.
  - (We shall support his claim (“their final meaning”) in the next section.)

- Let us illustrate the concept of properties.
- **Examples:**
  - Typical properties of street segments are:
    - \* length,
    - \* cartographic location,
    - \* surface material,
    - \* surface condition,
    - \* traffic state —  
whether open in one, the  
other, both or closed in all  
directions.

– Typical properties of street intersections are:

- \* design<sup>5</sup>
- \* location,
- \* surface material,
- \* surface condition,
- \* traffic state —  
open or closed between any two  
pairs of in/out street segments.

– Typical properties of road nets are:

- \* name,
- \* owner,
- \* public/private,
- \* free/tool road,
- \* area,
- \* etcetera.




---

<sup>5</sup>for example,

- |                               |                   |                       |
|-------------------------------|-------------------|-----------------------|
| • a simple ‘carrefour’, or    | a stack or        | a trumpet or          |
| • a (circular) roundabout, or | a clover-stack or | a directional or      |
| • a free-way interchange      | a turbine or      | a full Y or           |
| a cloverleaf or               | a roundabout or   | a hybrid interchange. |

40. Parts are characterised (also) by a set of one or more distinctly named and not necessarily distinctly typed property values.
- (a) Property names are further undefined tokens (i.e., simple quantities).
  - (b) Property types are either sorts or are concrete types such as integers, reals, truth values, enumerated simple tokens, or are structured (sets, Cartesians, lists, maps) or are functional types.
  - (c) From a part
    - i. one can observe its sets of property names
    - ii. and its set (i.e., enumerable map) of distinctly named and typed property values.
  - (d) Given an property name of a part one can observe the value of that part for that property name.
  - (e) For practical reasons we suggest **property** named **property** value observer function — where we further take the liberty of using the **property** type name in lieu of the **property** name.



**type**

40.  $\text{Props} = \text{PropNam} \xrightarrow{m} \text{PropVAL}$

40(a).  $\text{PropNam}$

40(b).  $\text{PropVAL}$

**value**

40((c))i.  $\text{obs\_Props}: P \rightarrow \text{Props}$

40((c))ii.  $\text{xtr\_PropNams}: P \rightarrow \text{PropNam-set}$

40((c))ii.  $\text{xtr\_PropNams}(p) \equiv \mathbf{dom} \text{ obs\_Props}(p)$

40(d).  $\text{xtr\_PropVAL}: P \rightarrow \text{PropNam} \xrightarrow{\sim} \text{PropVAL}$

40(d).  $\text{xtr\_PropVAL}(p)(pn) \equiv (\text{obs\_Props}(p))(pn)$

40(d). **pre:**  $pn \in \text{xtr\_PropNams}(p)$

- Here we leave **PropNames** and **PropVALues** undefined.

## • Example:

### type

NAME, OWNER, LEN, DESIGN, PP == public | private, ...

$L\Sigma$ ,  $H\Sigma$ ,  $L\Omega$ ,  $H\Omega$

### value

obs\_Props:  $N \rightarrow \{ | [ \text{"name"} \mapsto nm, \text{"owner"} \mapsto ow, \text{"public/private"} \mapsto pp, \dots ]$   
 $| nm:NAME, ow:OWNER, \dots, pp:PP | \}$

obs\_Props:  $L \rightarrow \{ | [ \text{"length"} \mapsto len, \dots, \text{"state"} \mapsto l\sigma, \text{"state space"} \mapsto l\omega:L\Omega ]$   
 $| len:LEN, \dots, l\sigma:L\Sigma, l\omega:L\Omega | \}$

obs\_Props:  $H \rightarrow \{ | [ \text{"design"} \mapsto des, \dots, \text{"state"} \mapsto h\sigma, \text{"state space"} \mapsto h\omega ]$   
 $| des:DESIGN, \dots, h\sigma:H\Sigma, h\omega:H\Omega | \}$

prop\_NAME:  $N \rightarrow NAME$

prop\_OWNER:  $N \rightarrow OWNER$

prop\_LEN:  $L \rightarrow LEN$

prop\_LΣ:  $L \rightarrow L\Sigma$ , obs\_LΩ:  $L \rightarrow L\Omega$

prop\_DESIGN:  $H \rightarrow DESIGN$

prop\_HΣ:  $H \rightarrow H\Sigma$ , obs\_HΩ:  $H \rightarrow H\Omega$

...

## 5.3. Attributes

- There are (thus) three kinds of part attributes:
  - **unique identifier** “observers” (**uid\_**),
  - **mereology** “observers” (**mereo\_**), and
  - **property** “observers” (**prop\_...**, **obs\_Props**)
- We refer to the section on ‘Attributes’ in the previous section, and to Items 15–16.

**type**

15.'  $\text{ATR} = \Pi \times \Pi\text{-set} \times \text{Props}$

**value**

16.'  $\text{atr\_ATR}: P \rightarrow \text{ATR}$

**axiom**

$\forall p:P \cdot \text{let } (\pi, \pi_s, \text{props}) = \text{atr\_ATR}(p) \text{ in } \pi \notin \pi_s \text{ end}$

- In preparation for redefining the **share** function of Item 17 on Slide 45 we must first introduce a modification to property values.

41. A property value, **pv:PropVal**, is

- either a simple property value (as was hitherto assumed),
- or is a unique part identifier.

**type**

40.  $\text{Props} = \text{PropNam} \xrightarrow{m} \text{PropVAL\_or\_}\Pi$

41.  $\text{PropVAL\_or\_}\Pi :: \text{mk\_Simp:PropVAL} \mid \text{mk\_}\Pi:\Pi$

42. The idea a property name  $pn$ , of a part  $p'$ , designating a  $\Pi$ -valued property value  $\pi$  is
- (a) that  $\pi$  refers to a part  $p'$
  - (b) one of whose property names must be  $pn$
  - (c) and whose corresponding property value must be a proper, i.e., simple property value,  $v$ ,
  - (d) which is then the property value in  $p'$  for  $pn$ .

## value

42.  $\text{get\_VAL}: P \times \text{PropName} \rightarrow W \rightarrow \text{PropVAL}$

42.  $\text{get\_VAL}(p, pn)(w) \equiv$

44.     **let**  $pv = (\text{obs\_Props}(p))(pn)$  **in**

42.     **case**  $pv$  **of**

42.          $\text{mk\_Simp}(v) \rightarrow v,$

42(a).          $\text{mk\_}\Pi(\pi) \rightarrow$

42(a).             **let**  $p':P.p' \in \text{xtr\_Ps}(w) \wedge \text{id\_}\Pi(p') = \pi$  **in**

42(c).              $(\text{obs\_Props}(p'))(pn)$  **end**

42.     **end end**

42(c).     **pre:**  $pn \in \text{obs\_PropNams}(p)$

42(b).          $\wedge pn \in \text{obs\_PropNams}(p')$

42(c).          $\wedge \text{is\_PropVAL}((\text{obs\_Props}(p'))(pn))$

- The three bottom lines above, Items 42(b)–42(c), imply the general constraint now formulated.

43. We now express a constraint on our modelling of attributes.

- (a) Let the attributes of a part  $p$  be  $(\pi, \pi s, \mathbf{props})$ .
- (b) If a property name  $pn$  in  $\mathbf{props}$  has (associates to) a  $\Pi$  value, say  $\pi'$
- (c) then  $\pi'$  must be in  $\pi s$ .
- (d) and there must exist another part,  $p'$ , distinct from  $p$ , with unique identifier  $\pi'$ , such that
- (e) it has some property named  $pn$  with a simple property value.

**value**

43. wf\_ATTR:  $\text{ATR} \rightarrow \text{W} \rightarrow \mathbf{Bool}$

43(a).  $\text{wf\_ATTR}(\pi, \pi s, \mathbf{props})(w) \equiv$

43(a).  $\pi \notin \pi s \wedge$

43(b).  $\forall \pi': \Pi \cdot \pi' \in \mathbf{rng} \ \mathbf{props} \Rightarrow$

43(c).  $\mathbf{let} \ pn: \text{PropNam} \cdot \mathbf{props}(pn) = \pi' \ \mathbf{in}$

43(c).  $\pi' \in \pi s$

43(d).  $\wedge \exists p': P \cdot p' \in \text{xtr\_Ps}(w) \wedge \text{uid\_}\Pi(p') = \pi' \Rightarrow$

43(e).  $pn \in \text{obs\_PropNams}(\text{obs\_Props}(p'))$

43(e).  $\wedge \exists \text{mk\_SimpVAL}(v): \text{VAL} \cdot (\text{obs\_Props}(p'))(pn) = \text{mk\_SimpVAL}(v) \ \mathbf{end}$

#### 44. Two distinct parts share attributes

- (a) if the unique part identifier of one of the parts is in the mereology of the other part, or
- (b) if a property value of one of the parts refers to a property of the other part.



**value**

44. share:  $P \times P \rightarrow \mathbf{Bool}$

44. share( $p, p'$ )  $\equiv$

44.  $p \neq p' \wedge$

44. **let**  $(\pi, \pi s, \text{props}) = \text{atr\_ATR}(p), (\pi', \pi s', \text{props}') = \text{atr\_ATR}(p'),$

44.  $\text{pns} = \text{xtr\_PropNams}(p), \text{pns}' = \text{xtr\_PropNams}(p') \text{ in}$

44(a).  $\pi \in \pi s' \vee \pi' \in \pi s \vee$

44(b).  $\exists \text{pn:PropNam} \cdot \text{pn} \in \text{pns} \cap \text{pns}' \Rightarrow$

44(b). **let**  $\text{vop} = \text{props}(\text{pn}), \text{vop}' = \text{props}'(\text{pn}) \text{ in}$

44(b). **case**  $(\text{vop}, \text{vop}') \text{ of}$

44(b).  $(\text{mk\_}\Pi(\pi''), \text{mk\_Simp}(v)) \rightarrow \pi'' = \pi',$

44(b).  $(\text{mk\_Simp}(v), \text{mk\_}\Pi(\pi'')) \rightarrow \pi = \pi'',$

44(b).  $\_ \rightarrow \mathbf{false}$

44. **end end end**

- **Comment:**  $v$  is a shared attribute.

## 5.4. Discussion

- We have now witnessed four kinds of observer function:
  - the above three kinds of mereology and property ‘observers’ and the
  - part (and subpart) **obs\_**ervers,.
- These observer functions are postulated.
  - They cannot be defined.
  - They “just exist” by the force
    - \* of our ability to observe and
    - \* decide upon their values
    - \* when applied by us, the domain observers.

- Parts are either composite or atomic.
  - Analytic functions are postulated. They help us decide
    - \* whether a part is composite or atomic, and,
    - \* from composite parts their immediate subparts.
- Both atomic and composite parts have all three kinds of attributes:
  - unique identification,
  - mereology (connections), and
  - properties.
- Analytic functions help us observe, from a part,
  - its unique identification,
  - its mereology, and
  - its properties.

- Some attribute values
  - may be static, that is, constant, others
  - may be inert dynamic, that is, can be changed.
- It is exactly the inert dynamic attributes which are the basis for the next sections semantic model of parts as processes.
- In the above model
  - we have not modelled distinctions between static and dynamic properties.
  - You may think, instead of such a model, that an **always** temporal operator,  $\Box$ , being applied to appropriate predicates.

## 6. A Semantic CSP Model of Mereology

- The model of Sect. 3 can be said to be an abstract model-oriented definition of the syntax of mereology.
- Similarly the axiom system of Sect. 4 can be said to be an abstract property-oriented definition of the syntax of mereology.
- With the analysis of attributes of parts, Sect. 5, we have begun a semantic analysis of mereology.
- We now bring that semantic analysis a step further.

## 6.1. A Semantic Model of a Class of Mereologies

- We show that to every mereology there corresponds a program of cooperating sequential processes CSP.
- We assume that the listener has practical knowledge of Hoare's CSP.

### 6.1.1. Parts $\equiv$ Processes

- The model of mereology (Slides 32–55) given earlier focused on (i) parts and (ii) connectors.
- To parts we associate CSP processes.
- Part processes are indexed by the unique part identifiers.
- The connectors form the mereological attributes of the model.

## 6.1.2. Connectors $\equiv$ Channels

- The **CSP** channels are indexed by the two-set (hence distinct) part identifier connectors.
- From a whole we can extract (**xtr\_Ks**, Item 26 on Slide 51) all connectors.
- They become indexes into an array of channels.
  - Each of the connector channel index identifiers
  - indexes exactly two part processes.

- Let  $w:W$  be the whole under analysis.

**value**

$w:W$

$ps:P\text{-set} = \cup \{xtr\_Ps(c) | c:C \cdot c \in w\} \cup \{a | a:A \cdot a \in w\}$

$ks:K\text{-set} = xtr\_Ks(w)$

**type**

$K = \Pi\text{-set axiom } \forall k:K \cdot \text{card } k=2$

$ChMap = \Pi \xrightarrow{m} K\text{-set}$

**value**

$cm:ChMap = [uid\_ \Pi(p) \mapsto xtr\_Ks(p) | p:P \cdot p \in ps]$

**channel**

$ch[k | k:K \cdot k \in ks] \text{ MSG}$

- We leave channel messages.  $m:MSG$ , undefined.



### 6.1.3. Process Definitions

value

system:  $W \rightarrow \mathbf{process}$

system( $w$ )  $\equiv$

$\parallel \{ \text{comp\_process}(\text{uid\_}\Pi(c))(c) \mid c:C \cdot c \in w \} \parallel \parallel \{ \text{atom\_process}(\text{uid\_}\Pi(a),a) \mid a:A \cdot a \in w \}$

comp\_process:  $\pi:\Pi \rightarrow c:C \rightarrow \mathbf{in,out} \{ \text{ch}(k) \mid k:K \cdot k \in \text{cm}(\pi) \} \mathbf{process}$

comp\_process( $\pi$ )( $c$ )  $\equiv [ \mathbf{assert:} \pi = \text{uid\_}\Pi(c) ]$

$\mathcal{M}_C(\pi)(c)(\text{atr\_ATR}(c)) \parallel$

$\parallel \{ \text{comp\_process}(\text{uid\_}\Pi(c'))(c') \mid c':C \cdot c' \in \text{obs\_Ps}(c) \} \parallel$

$\parallel \{ \text{atom\_process}(\text{uid\_}\Pi(a))(a) \mid a:A \cdot a \in \text{obs\_Ps}(c) \}$

$\mathcal{M}_C: \pi:\Pi \rightarrow C \rightarrow \text{ATR} \rightarrow \mathbf{in,out} \{ \text{ch}(k) \mid k:K \cdot k \in \text{cm}(\pi) \} \mathbf{process}$

$\mathcal{M}_C(\pi)(c)(c\_attrs) \equiv \mathcal{M}_C(c)(C\mathcal{F}(c)(c\_attrs)) \quad \mathbf{assert:} \text{atr\_ATR}(c) \equiv c\_attrs$

$C\mathcal{F}: c:C \rightarrow \text{ATR} \rightarrow \mathbf{in,out} \{ \text{ch}[\text{em}(i)] \mid i:KI \cdot i \in \text{cm}(\text{uid\_}\Pi(c)) \} \text{ATR}$

ATR and atr\_ATR are defined in Items 15.' and 16.' (Slide 83).

atom\_process:  $a:A \rightarrow \mathbf{in,out} \{ \text{ch}[\text{cm}(k)] \mid k:K \cdot k \in \text{cm}(\text{uid}_\Pi(a)) \} \text{ process}$   
 $\text{atom\_process}(a) \equiv \mathcal{M}_{\mathcal{A}}(a)(\text{atr\_ATR}(a))$

$\mathcal{M}_{\mathcal{A}}: a:A \rightarrow \text{ATR} \rightarrow \mathbf{in,out} \{ \text{ch}[\text{cm}(k)] \mid k:K \cdot k \in \text{cm}(\text{uid}_\Pi(a)) \} \text{ process}$   
 $\mathcal{M}_{\mathcal{A}}(a)(a\_attrs) \equiv \mathcal{M}_{\mathcal{A}}(a)(A\mathcal{F}(a)(a\_attrs)) \quad \mathbf{assert:} \text{atr\_ATR}(a) \equiv a\_attr$

$A\mathcal{F}: a:A \rightarrow \text{ATR} \rightarrow \mathbf{in,out} \{ \text{ch}[\text{em}(k)] \mid k:K \cdot k \in \text{cm}(\text{uid}_\Pi(a)) \} \text{ ATR}$

- The meaning processes  $\mathcal{M}_C$  and  $\mathcal{M}_A$  are generic.
  - Their sole purpose is to provide a never ending recursion.
  - “In-between” they “make use” of Composite, respectively Atomic specific  $\mathcal{F}$ unctions
  - here symbolised by  $C\mathcal{F}$ , respectively  $A\mathcal{F}$ .
- Both  $C\mathcal{F}$  and  $A\mathcal{F}$ 
  - are expected to contain input/output clauses referencing the channels of their signatures;
  - these clauses enable the sharing of attributes.
- We illustrate this “sharing” by the schematised function  $\mathcal{F}$  standing for either  $C\mathcal{F}$  or  $A\mathcal{F}$ .

**value**

$\mathcal{F}: p:(C|A) \rightarrow \text{ATR} \rightarrow \mathbf{in,out} \{ \text{ch}[\text{em}(k)] \mid k:K \cdot k \in \text{cm}(\text{uid}_\Pi(p)) \} \text{ATR}$

$\mathcal{F}(p)(\pi, \pi s, \text{props}) \equiv$

$\square \{ \mathbf{let} \text{ av} = \text{ch}[\text{em}(\{\pi, j\})] ? \mathbf{in}$   
 $\quad \dots ; [\text{optional}] \text{ch}[\text{em}(\{\pi, j\})] ! \text{in\_reply}(\text{props})(\text{av});$   
 $\quad (\pi, \pi s, \text{in\_update\_ATR}(\text{props})(j, \text{av})) \mathbf{end}$   
 $\mid \{\pi, j\}:K \cdot \{\pi, j\} \in \pi s \}$

$\square \square \{ \dots ;$   
 $\quad \text{ch}[\text{em}(\{\pi, j\})] ! \text{out\_reply}(\text{props});$   
 $\quad (\pi, \pi s, \text{out\_update\_ATR}(\text{props})(j))$   
 $\mid \{\pi, j\}:K \cdot \{\pi, j\} \in \pi s \}$

$\square (\pi, \pi s, \text{own\_work}(\text{props}))$

**assert:**  $\pi = \text{uid}_\Pi(p)$

$\text{in\_reply}: \text{Props} \rightarrow \Pi \times \text{VAL} \rightarrow \text{VAL}$

$\text{in\_update\_ATR}: \text{Props} \rightarrow \Pi \times \text{VAL} \rightarrow \text{Props}$

$\text{out\_reply}: \text{Props} \rightarrow \text{VAL}$

$\text{out\_update\_ATR}: \text{Props} \rightarrow \Pi \rightarrow \text{Props}$

$\text{own\_work}: \text{Props} \rightarrow \text{Props}$

## 6.2. Discussion

### 6.2.1. General

- A little more meaning has been added to the notions of parts and connections.
- The **within** and **adjacent to** relations between parts (composite and atomic) reflect a phenomenological world of geometry, and
- the **connected** relation between parts
  - reflect both physical and conceptual world understandings:
    - \* physical world in that, for example, radio waves cross geometric “boundaries”, and
    - \* conceptual world in that ontological classifications typically reflect lattice orderings where *overlaps* likewise cross geometric “boundaries”.

## 6.2.2. Partial Evaluation

- The **composite\_processes** function “first” “functions” as a compiler.  
The ‘compiler’ translates an assembly structure into three process expressions:
  - the  $\mathcal{M}_c(c)(c\_attrs)$  invocation,
  - the parallel composition of composite processes,  $c'$ , one for each composite sub-part of  $c$ , and
  - the parallel composition of atomic processes,  $a$ , one for each atomic sub-part of  $c$
  - with these three process expressions “being put in parallel”.
  - The recursion in **composite\_processes** ends when a sub-...-composites consist of no sub-sub-...-composites.
- Then the compiling task ends and the many generated  $\mathcal{M}_c(c)(c\_attrs)$  and  $\mathcal{M}_a(a)(a\_attrs)$  process expressions are invoked.

## 7. Closing

### 7.1. Relation to Other Work

- Douglas T. Ross: Plex, CAD, APT, SADT, IDEF0, ...
- Leonard Goodman 1940: Calculus of Individuals
- R. Casati and A. Varzi: Parts and Places: the structures of spatial representation.
- B. Ganter and R. Wille: Formal Concept Analysis — Mathematical Foundations.
- Etcetera.

## 7.2. What Has Been Achieved ?

- We have given a model-oriented specification of mereology.
- We have indicated that the model satisfies a widely known axiom system for mereology.
- We have suggested that (perhaps most) work on mereology amounts to syntactic studies.
- So we have suggested one of a large number of possible, schematic semantics of mereology.
- And we have shown that to every mereology there corresponds a set of communicating sequential process (**CSP**).



## 7.3. Future Work

- We need to characterise, in a proper way,
  - the class of **CSP** programs
  - for which there corresponds a mereology.
- Are you game ?
- One could also wish for an extensive editing and publication of Doug Ross' surviving notes.

## 7.4. Acknowledgements

- I thank
  - Dr. Claudio Calosi and
  - Dr. Pierluigi Graziani,University of Urbino, Italy, for inviting this paper for
  - a Springer Verlag *Synthese Library* volume on
  - *Mereology and the Sciences*. Due Summer 2012
- I further thank Patricia M. Ross for permission to dedicate this paper to the memory of her husband of many years.

**QUESTIONS ?**