

---

**Bottom-up construction of ontologies:  
the case of an ontology of pure substances**

Paul E. van der Vet

Nicolaas J.I. Mars

Memoranda Informatica 95-35

September 20, 1995

---

---

Technical report UT-KBS-

Knowledge-Based Systems Group  
Department of Computer Science

University of Twente  
P.O. Box 217  
7500 AE Enschede  
The Netherlands

---

## **Abstract**

Bottom-up construction of ontologies departs from the more usual taxonomic approach to ontology development. It allows for parsimonious specification of concepts. Defined concepts are written as compact expressions that allow flexible reasoning. Obvious applications include chemistry and engineering design. For such domains, the method enables better ontologies than the taxonomic method. Taxonomic approaches remain appropriate for other applications.

Bottom-up ontology construction is illustrated by presenting a drawn-out example involving an ontology of pure substances. This ontology is a simplified version of a part of the ontology developed for the Plinius project of the Knowledge-Based Systems Group, University of Twente. The viability of the approach is demonstrated by presenting a complete formalisation in pure logic programming.

# Chapter 1

## Foreword

In the Plinius project of the Knowledge-Based Systems Group at the University of Twente, ontology design is an actively pursued research subject. In 1991, we proposed an ontology for the domain covered by Plinius in a technical report and several publications. Since then, many discussions and a closer look at our needs in the project necessitated a large number of revisions. In the course of those discussions we also realised more clearly on which principles the Plinius ontologies are built. Of these, the present report discusses two. The first issue is the function of an ontology in the whole process which ultimately results in an implemented knowledge base. The second issue is a particular way of building ontologies called the bottom-up method. The report may thus seem to address one issue too many. As the title makes clear, the bottom-up method is the proper subject. However, we found that important aspects of the discussion are left hanging in mid-air if we omit a discussion of the first issue.

For the purposes of discussing bottom-up construction of ontologies, we have taken the subject of pure substances out of the larger ontology. Pure substances are sufficiently complex to merit discussion, but sufficiently simple to serve as an illustration. Also, the subject of pure substances takes relatively little domain knowledge.

This report discusses two variants of the bottom-up principle. In the simpler variant wholes are simply defined by telling which constituents there are. The other variant in addition specifies how the constituents are connected. Judging by the examples of non-chemical applications presented in chapter 6, it may seem that the applicability of the simpler variant is limited. The fact is, we don't know. Further research will have to clarify these matters, but as there is at least one important application we did not want to wait.

In preparing this report, we have profited from the many discussions with the other members of the Plinius/Condorcet research group: Piet-Hein Speel (now at Unilever Research Laboratories, Vlaardingen, the Netherlands), Wilco ter Stal, Jeroen Nijhuis, and in particular Hidde de Jong. We are also indebted to Stef Joosten (University of Twente) and to Tom Gruber (then at Stanford) for their comments on various versions of the ontology.

Enschede, September 20, 1995.

Paul E. van der Vet  
Nicolaas J.I. Mars

**Address of the authors:**

Knowledge-Based Systems Group  
Dept. of Computer Science, University of Twente  
P.O. Box 217, 7500 AE Enschede  
The Netherlands

Phone +31 53 89 36 90

Fax +31 53 33 96 05

Email {VET,MARS}@CS.UTWENTE.NL

# Contents

<b>1</b>	<b>Foreword</b>	<b>1</b>
<b>2</b>	<b>Introduction</b>	<b>5</b>
2.1	Ontology design . . . . .	5
2.2	Setting: the Plinius project . . . . .	5
2.3	Design principles . . . . .	7
2.3.1	Knowledge as assertions about entities . . . . .	7
2.3.2	Ontology as semantics . . . . .	7
2.3.3	Independence of particular knowledge representation languages . . . . .	8
2.3.4	The principle of conceptual construction kit . . . . .	8
2.3.5	Bottom-up approach . . . . .	8
2.3.6	Engineering perspective . . . . .	9
2.3.7	Quality assessment . . . . .	9
<b>3</b>	<b>Function of an ontology</b>	<b>10</b>
3.1	Ontology as semantics . . . . .	10
3.2	Utility . . . . .	11
3.3	Ontology and representation . . . . .	12
3.4	Concepts and instances . . . . .	12
<b>4</b>	<b>Chemical background</b>	<b>14</b>
4.1	The concept of sample . . . . .	14
4.2	Pure substances . . . . .	15
4.3	Bonding . . . . .	15
4.4	Formulae for pure substances . . . . .	16
4.5	Non-existing substances . . . . .	17
4.6	Structure . . . . .	18
<b>5</b>	<b>A simple ontology of pure substances</b>	<b>20</b>
5.1	Introduction . . . . .	20
5.2	Numbers . . . . .	21
5.3	Samples and their constitution . . . . .	22
5.4	Chemical elements . . . . .	23
5.5	Groups . . . . .	23
5.5.1	Considerations . . . . .	23
5.5.2	Definition of concepts of type group . . . . .	24
5.5.3	Absolute constitutions . . . . .	25
5.6	Ions . . . . .	25
5.6.1	Definition of concepts of type ion . . . . .	25
5.6.2	Attribute combinations . . . . .	26
5.7	Pure substances . . . . .	26
5.7.1	Definition of concepts of type pure substance . . . . .	26

---

5.7.2	Relative constitutions . . . . .	27
5.8	Examples . . . . .	28
5.9	Names for complex concepts . . . . .	29
5.10	Primitive or defined? . . . . .	29
5.11	Part-whole hierarchies . . . . .	30
5.12	Taxonomies . . . . .	31
5.12.1	The subconcept-relation . . . . .	31
5.12.2	“Arbitrary” . . . . .	31
5.12.3	Subconcept relations between complex concepts . . . . .	32
5.13	Configurations . . . . .	33
5.13.1	Configuration information . . . . .	33
5.13.2	Chemical graphs . . . . .	33
5.13.3	Other ways to specify configuration information . . . . .	35
5.14	Extendability: isotopes . . . . .	35
5.15	Graphs of ontologies . . . . .	36
<b>6</b>	<b>Bottom-up construction of ontologies</b>	<b>38</b>
6.1	Atomism . . . . .	38
6.2	Further examples . . . . .	39
6.2.1	Engineered artifacts . . . . .	39
6.2.2	Natural-language sentences . . . . .	40
6.2.3	Processes . . . . .	41
6.2.4	Diseases . . . . .	41
<b>7</b>	<b>Formalisation and implementation</b>	<b>43</b>
7.1	The choice of representation language . . . . .	43
7.2	The Prolog formalisation . . . . .	44
7.2.1	Set-up . . . . .	44
7.2.2	Atomic concepts . . . . .	45
7.2.3	Complex concepts . . . . .	45
7.2.4	Other features . . . . .	45
7.2.5	Toward implementation . . . . .	46
<b>8</b>	<b>Concluding remarks</b>	<b>47</b>
<b>A</b>	<b>List of chemical elements</b>	<b>49</b>
<b>B</b>	<b>Prolog implementation</b>	<b>52</b>
B.1	Introduction . . . . .	52
B.2	Program <code>atomic.pl</code> . . . . .	53
B.3	Program <code>complex.pl</code> . . . . .	53
B.4	Program <code>constitution.pl</code> . . . . .	54
B.5	Program <code>constituent.pl</code> . . . . .	55
B.6	Program <code>subconcept.pl</code> . . . . .	56
B.7	Program <code>graph.pl</code> . . . . .	57
B.8	Program <code>auxiliary.pl</code> . . . . .	57
<b>C</b>	<b>Bibliography</b>	<b>59</b>

# Chapter 2

## Introduction

### 2.1 Ontology design

Ontology design and the function of an ontology in designing and building (very) large knowledge bases are actively pursued research subjects at the Knowledge-Based Systems Group (Mars [1994]). An *ontology*<sup>1</sup> consists of a collection of unambiguously defined concepts (including relations). The collection can be a flat list or structure can be added in the form of relations between concepts. We have found it advantageous to incorporate structure to the extent that most concepts are fully defined in terms of a small set of primitives.

An ontology structures the domain of interest at the knowledge level (Newell [1982]). As a consequence, the development of an ontology proceeds independently from representational and implementational considerations. Examples of projects in which an ontology has been used are LILOG (Herzog & Rollinger [1991]), UMLS (Humphreys, Lindberg & Hole [1991]), and Cyc (Lenat & Guha [1990]).

There is at present no method, let alone a recipe, for designing ontologies. In the literature, first approaches have been published (see, for instance, Neches et al. [1991], Gruber [1993b], Mars [1994], and Gruber [1993a]). In the current situation, the best we can do is to articulate the design decisions underlying the particular ontology we are developing. That is also the point addressed by this report. In a next phase (not considered here) the decisions taken in the course of a number of ontology development jobs can be compared and assessed. In this manner, accumulated experience is used to pave the way for more methodical approaches.

In the present chapter, we summarise the design decisions that we have been aware of in preparing a particular ontology, the Plinius ontology of the chemical composition of materials. The more important design decisions will be elaborated in the next two chapters. Of these, the bottom-up approach to ontology construction is singled out for extensive discussion in subsequent chapters. The approach is a particular way to construct an ontology such that it fully defines most concepts in terms of a small set of primitive concepts. In the presentation of the bottom-up approach, we will draw heavily on the example of an ontology of pure substances. We start with a short description of the Plinius project, which constitutes the setting.

### 2.2 Setting: the Plinius project

The Plinius project, undertaken by the Knowledge-Based Systems Group at the University of Twente, the Netherlands, aims to realise a system for semi-automatic knowledge extraction from short texts. Detailed information on the project, including its aims (Mars & Schreiber [1985]; Mars & van der Vet [1990]), design decisions (Mars et al. [1994]), and results of the first three years (van der Vet et al. [1994]) are available elsewhere.

---

<sup>1</sup>A term introduced in this sense by Hayes [1985].

Superplasticity of Hot Isostatically Pressed Hydroxyapatite

Dense and translucent hydroxyapatite polycrystals ( $\text{Ca}_{10}(\text{PO}_4)_6(\text{OH})_2$  with a grain size of  $0.64 \mu\text{m}$ ) were obtained by hot isostatic pressing at 203 MPa and  $1000^\circ\text{C}$  for 2 h in Ar. The material exhibited superplastic elongation ( $> 150\%$ ) in a tension test at temperatures from  $1000 - 1100^\circ\text{C}$  and at strain rates from  $7.2 \times 10^{-5}$  to  $3.6 \times 10^{-4} \text{ s}^{-1}$ . Extensive strain hardening was observed. The stress exponent of the yield stress was larger than three.

**Figure 2.1:** TITLE AND ABSTRACT OF REPRESENTATIVE DOCUMENT DESCRIPTION. The title and abstract fields of document description nr. 1123 from the 1990 volume of *Engineered Materials Abstracts*, ©1990 Materials Information. We have taken the material as found on the EMA distribution tape, but for legibility have manually transcribed the abstract text to transform the formulae into their standard form.

The short texts are taken from the 1990 volume of *Engineered Materials Abstracts* (EMA) of Materials Information.<sup>2</sup> We have selected automatically the subset dealing with mechanical properties of ceramic materials (see van der Vet et al. [1994], section 2.4 for more details), comprising 406 texts. Of these, 293 texts form the so-called development corpus while the remaining 113 texts, called the test set, are locked away and will be used for testing the system somewhere in the near future. We will be concerned with the development corpus in the present report. See figure 2.1 for an example text. The output of Plinius consists of a knowledge base called the *interim knowledge base* and a set of *knowledge integration programs*.

The ontology used in the Plinius project supports the translation of natural-language sentences into assertions expressed in a knowledge representation language. It fulfills three functions directly relevant to the process:

1. The ontology provides concepts that serve as semantic translations of natural-language words and phrases in the lexicon. For instance, the ontology provides concepts for materials. The meaning of the natural-language word “hydroxylapatite” is given (via an intermediate step) as the appropriate concept in the lexicon. Thus, the ontology can be considered an operationalisation of the notion of semantics for the development corpus. By way of a practical consequence, the ontology aids in the resolution of lexical ambiguities.
2. The ontology enables co-operation between the knowledge bases used as resource in the knowledge extraction process. This is mainly achieved by using the same ontology for the semantic part of the lexicon and the background knowledge base.
3. The ontology implicitly and partly specifies the output of the process by demanding that only knowledge expressible in ontology concepts occur in the knowledge bases produced by the processes.

As a consequence of these primary functions, the ontology is also instrumental in decisions on further uses of the output. It helps assess whether and, if so, how the output can be utilised in systems for materials selection or materials design or be combined with other knowledge bases and databases for yet other purposes.

The Plinius ontology is under development. When finished, it will cover materials, their properties, processes that change properties, and processes to make ceramics. Since the Plinius ontology will become large, it is split over several parts that can be considered loosely coupled ontologies. Of these, the part dealing with the chemical composition of ceramic materials is used as starting point for the present report. We are interested in explaining a particular way of constructing an ontology rather than in presenting the full Plinius ontology of chemical composition.<sup>3</sup> Therefore

<sup>2</sup>Materials Information is a registered trademark of two professional societies of materials scientists and engineers: the Institute of Materials, London, and ASM International, Metals Park, Ohio. We are indebted to Materials Information, and to W. Jackson in particular, for making this material available to us.

<sup>3</sup>Treatments of the Plinius ontology of chemical composition are given by van der Vet & Mars [1993], van der Vet & Mars [1994], and van der Vet, Speel & Mars [1994]. The last source also contains a specification in Ontolingua.

we have simplified chemical composition to a specification in terms of pure substances only. In the following account, “the Plinius ontology” refers to the full set while “the ontology of pure substances” refers to the example at issue here.

## 2.3 Design principles

We have developed and are still developing the Plinius ontology in a craft-like manner. We only gradually became aware of the decisions underlying our approach, and most decisions have been identified in retrospect. Where many decisions are domain-dependent and thus can be discussed only in the context of a particular ontology, others appear more general. The distinction is important because the latter kind of decisions, to be called principles from now on, can be ported when developing new ontologies. We summarise the principles here and postpone a more elaborate discussion of the more important principles to later chapters.

We approach the subject of ontologies from an engineering perspective. We regard the design of an ontology as an essential step in the design and actual implementation of knowledge bases and knowledge-based systems. The importance of the availability of an ontology is particularly evident when the knowledge base gets large (Mars [1994]).

### 2.3.1 Knowledge as assertions about entities

Scientific knowledge is the kind of knowledge primarily studied by us. In science, it is vital to be able to assess whether a new proposal is in conflict with what we already know. For a knowledge base with scientific knowledge, a similar requirement holds. In a first approximation, the problem is solved in science by distinguishing between defining and non-defining properties of entities (whether individuals or classes of individuals). Knowledge consists of assertions about entities.<sup>4</sup> Two assertions can only conflict if they are about the same entity, and this can be found out by inspecting the defining properties of the entities. This strategy can be adopted for scientific knowledge bases, so that the detection of conflicts can be automated.<sup>5</sup> It requires at least the introduction of identifying labels, but it is better to have many concepts fully defined so that the system can automatically inspect the defining properties.

### 2.3.2 Ontology as semantics

In our way of working, the knowledge used by a knowledge-based system is expressed in a fully formal knowledge representation language at some point in the development process. At that point, an ontology has to be available. It is used to provide the semantics of *every* non-logical constant that occurs in the representation.<sup>6</sup> Pre-existing formal theories, however, are taken as given. Thus, our ontology is silent on the semantics of logical constants such as conjunction or quantification. It also incorporates set theory and number theory as being given.

The ontology is seen to be part of the conceptualisation. We will elaborate in chapter 3.

---

<sup>4</sup> A similar approach is chosen by Bunge [1977], who builds his ontology on the assumption that the world consists of things of which we can assert properties. His reason for doing so is that it appears to be the most sensible way to organise physical knowledge about the world. See also his ontology of molecules, which differs in certain respects from our own.

<sup>5</sup> This is an over-simplification. For a working system, we additionally need criteria to establish whether defining properties are sufficiently similar to conclude that the entities are identical. The catch-word here is “sufficiently”. In scientific practice, properties are directly or indirectly measured. Results of measurements are influenced by conditions of measurement and are also subject to random spread. This considerably complicates comparison of results obtained by different groups. A telling example is provided by melting points of oxides, where official committees aggregate published results to establish so-called recommended values (Coutures & Rand [1989]; Hlaváč [1982]).

<sup>6</sup> Guarino & Giaretta [1995] have identified seven different meanings of the term ‘ontology’. In their system, interpretation nr. 3 comes closest to what we mean.

### 2.3.3 Independence of particular knowledge representation languages

An ontology has to be independent of any particular choice of knowledge representation language. One rather obvious reason is, that the ontology contains specifications of the knowledge base to be built. The knowledge representation language has to be chosen so as to meet the specifications. Other advantages are: the ontology can serve to unify several co-operating knowledge bases written in different representation languages within a single system; and, more generally, the support of sharing and reuse.

In the Plinius project, the issue of the choice of representation language has been the subject of a separate study (Speel [1995]). In the course of these investigations, independence of particular representation languages has been demonstrated by representing and implementing a part of the Plinius ontology in a great variety of languages and systems. The version of Summer 1994 has been implemented in the KL-ONE systems CLASSIC, C-CLASSIC, BACK, BACK++, KRIS, and LOOM; and in Ontolingua version 4.

An older version of the same part of the Plinius ontology has been implemented in Ontolingua version 3, Prolog, LIFE, and three KL-ONE systems, the TAXON part of the COLAB system and, again, LOOM and CLASSIC. In addition, this older version has been represented but not implemented in Conceptual Graphs and *LLILOG*.<sup>7</sup>

As far as we are aware, no other ontology has been implemented in so many different ways. The experiences have taught us that, once the ontology is largely fixed, the choice of knowledge representation language is somewhat pre-determined. In particular, the ontology may contain constructs that are represented easily in some representation languages but only by using detours or not at all in others. See Speel [1995], chapters 4 and 5, for experiences with representing the Plinius ontology in KL-ONE-like languages.

### 2.3.4 The principle of conceptual construction kit

An ontology can be just a flat list of concepts with some explanation or display structure. Structure has the advantage of allowing for a parsimonious design since most concepts can be defined implicitly rather than be enumerated explicitly. This corresponds to what we call the principle of *conceptual construction kit*. An ontology built according to this principle consists of atomic concepts, serving as primitives, and construction rules that define all other concepts. The formal part of the ontology is given as a tuple  $\langle \mathcal{A}, \mathcal{C} \rangle$  with  $\mathcal{A}$  the set of atomic concepts and  $\mathcal{C}$  the set of construction rules. The tuple can be seen as a calculus with  $\mathcal{A}$  the alphabet and  $\mathcal{C}$  the transformation rules. The deductive closure of this calculus produces the explicit list of concepts.

The principle will be demonstrated in chapter 5.

### 2.3.5 Bottom-up approach

There are various ways to realise a conceptual construction kit. One approach starts with a concept for everything there is and proceeds by successive differentiation. It is a modern variant of the Aristotelian *genus-species* approach, where the *species* are differentiated from the *genus* and from each other by means of *differentiae*.

This approach is embodied, for instance, by KL-ONE-like systems.<sup>8</sup> The superconcept is the *genus*, every subconcept is a *species*, and the *differentiae* correspond to roles. The approach is not confined to KL-ONE-like systems. For instance, in several example ontologies illustrating the use of Ontolingua the same approach is used (Gruber [1993b]).

By contrast, our approach is based on *atomism*. Atomism proceeds bottom-up in that it builds objects out of smaller objects. For instance, chemical elements, kinds of atoms, are among the

<sup>7</sup>For Conceptual Graphs, there was no working system at that time. For *LLILOG*, there was a working system but it was not made available to us.

<sup>8</sup>KL-ONE is regarded here not as a family of description logics and associated systems (which it also is) but, rather, as a choice for a particular way of conceptualising a domain. This seems to be the idea underlying the original KL-ONE (Brachman & Levesque [1982]; Brachman & Schmolze [1985]).

more important atomic concepts. The construction rules construct concepts for more complex assemblies. The approach will be demonstrated in chapter 5. We will elaborate in chapter 6.

### 2.3.6 Engineering perspective

The task to be performed by the knowledge-based system has a great influence on the design of the ontology. In particular, the decision whether concepts are elaborated or not is governed by the outcome of a cost-benefit analysis.

For instance, in materials science it is important to classify materials according to the role they play in the production process. A typical ceramic artifact starts its career as a powder. It is turned into something called a greenling that incorporates the desired shape of the product. The greenling is densified to yield the finished product. A detailed conceptualisation of microscopic features at a scale of  $10^{-5}$ – $10^{-8}$  m allows unambiguous definition of these concepts. This has been omitted for Plinius, however, because it would serve mainly esthetic goals.

Engineering decisions are seen to rest on an identification of the task to be performed by the system. We must be silent on this, because our purpose is to discuss a particular way to build ontologies rather than their use in particular systems.

### 2.3.7 Quality assessment

Assessing the quality of an ontology presupposes a list of quality criteria. Again, such lists are not standard. For Plinius purposes, one of the more pressing criteria is completeness with respect to the Plinius task. Completeness, in turn, can be decomposed into *coverage* (is every concept of interest covered?) and *granularity* (is every relevant distinction made?). Assessment of any ontology on this criterion can only be carried through if there exists an operational specification of the domain. The Plinius development corpus serves as a starting point for assessing the completeness of the Plinius ontology. We have to add operational specifications that deal with the task, because we do not want to produce output for each and every text fragment.

A more elaborate discussion has to be based on considerations of the task to be performed by the system. This falls outside the scope of the present memo.

## Chapter 3

# Function of an ontology

### 3.1 Ontology as semantics

As explained in section 2.3.2, the design of an ontology is one of the activities that prepare for the representation of the knowledge that we want to use for reasoning. At some further point in the process of building a knowledge-based system, there will be a representation of the knowledge in a formal language. We require an ontology to supply the meaning of *every* non-logical constant that occurs in the representation. From the perspective of building the knowledge representation, the ontology is *limitative*. It rules out the use of a non-logical constant if its meaning cannot be supplied by the ontology.

A number of observations can be made.

(a) The meaning of logical constants is not fixed this way. In our view, a knowledge representation language ought to be a formal language. The logical constants are supplied by choosing a particular formal language, which is assumed to supply the meaning of the logical constants.

(b) The issue of meaning is a thorny one. A flat ontology consists of a list of concepts with some account that supplies meaning. Typically, the meaning of otherwise unexplicated concepts is anchored into consensus domain knowledge by means of a natural-language account. This implies that the natural-language account is an integral part of the ontology. In a structured ontology, certain concepts are defined in terms of others. We still need primitives to build other concepts. Moreover, it may be impossible or impractical to lay down full definitions. A natural-language account then has to supply the missing parts. In the ontology of pure substances discussed in chapter 5, concepts for chemical elements are among the primitives. We rely on a natural-language account to provide the meaning of those concepts. Chapter 4 in fact is an extensive natural-language account of the domain that, among other things, covers the issue of chemical elements. In the specification of the ontology itself it is sufficient to say that concepts for chemical elements are intended. In other cases, however, the natural-language account will be more substantial. What counts is that it is there and cannot be missed.

Apart from a natural-language part, an ontology contains a formal part. For a flat ontology, this is just the set of strings that stand for concepts. For a structured ontology, the definitions have to be supplied in a formal language. Since we are still at the knowledge level, computational issues do not play a role in choosing the formal language. The main function of the formal part is to reduce ambiguity. This can be realised by formalising as far as possible. We consider it unwanted to introduce additional axioms when formalising an ontology (unless, of course, they are needed as a detour because the original specification could not be expressed in the language).

(c) Since the ontology is limitative, it can be considered a specification (in the software engineering sense) of the concepts to be used in the knowledge base. Commitment to an ontology amounts to an agreement to use those and only those concepts.

## 3.2 Utility

The main advantage of having an ontology, and in fact of designing it as the first step in building a knowledge-based system, is that we gain *surveyability*. An ontology offers a vocabulary of unambiguous domain-related concepts. The meaning of the concepts has to be anchored in consensus domain knowledge. The ontology has to be *limitative*: it has to embrace every concept that is needed. This may necessitate intensional definitions. Since the ontology is exhaustive, it tells what can be expected in the knowledge base in terms of scope (which domain) and granularity (which distinctions). We believe that the requirement of exhaustivity is essential: without it, the ontology's utility is impaired.

Genesereth & Nilsson [1987] (Chapter 2) present an approach to designing knowledge-based systems that involves three stages: *conceptualisation*, in which we identify the objects in our domain of interest and their interrelations; *formalisation*, in which we write out the state of affairs in a formal language; and *implementation*, in which the system is implemented. The conceptualisation can be regarded as a specification of the knowledge that in the course of the second step, formalisation, is to be represented in some representation language. Genesereth and Nilsson provide a simple illustration in the form of a blocks world. Their conceptualisation consists of (i) a set of names of blocks and (ii) sets of tuples that stand for the relations between the blocks.

We regard the ontology as being part of the conceptualisation. The ontology only tells which concepts we choose to recognise and which relations may hold between them, but (in contrast to what is the case in the example of Genesereth and Nilsson) it does not describe any particular state of affairs. To arrive at a full conceptualisation we will have to add a description of the state of affairs; and since we have an ontology, we will do so in terms provided by the ontology.

To pursue the blocks world example, an ontology for the blocks world would list the blocks (either as flat names or in a more intricate way) or it would specify a concept of block but not its instances. The ontology would not specify which relations hold between blocks. Instead, the ontology would define the relations that might hold between blocks. This only provides a language for talking about the blocks world. It is not a full conceptualisation, because we have not yet specified which relations actually hold between the blocks. This is specified in the rest of the conceptualisation, which makes use of the vocabulary given by the ontology.

For a simple world like the blocks world, separate development of an ontology does not pay. But for real applications the conceptualisation is much more difficult. It is worth the trouble to develop the vocabulary separately, as a first step toward a full conceptualisation. One advantage is that, in an ontology, we can specify general structural aspects of our domain as distinct from the specification of the actual state of affairs. This is already apparent if we make the blocks world a bit more complex, by allowing for sides and edges as extra objects. We will want to distinguish between blocks, sides, and edges, even if only by the simple expedient of introducing three sets of objects rather than a single set. We can then achieve further clarity by adding structure to the ontology, for instance by demanding that every side is a side of a block, that every edge is an edge between two sides, and so on. These demands hold no matter which side forms part of which block or which edge is an edge between which two sides.

In further steps in developing a knowledge-based system, the ontology can be used to govern further choices. As has been said above, the conceptualisation can be regarded as a specification of the knowledge that in the course of the second step, formalisation, is to be represented in some representation language. A representation language is required to be a logical language with well-defined syntax and semantics. Logical constants and their associated inferential properties can be treated as being given. The ontology is required to supply the meaning of *every* non-logical constant that occurs in the representation.

The ontology used this way also supports *modularisation* of the knowledge base, another measure aimed at surveyability. The knowledge can be distributed over a number of knowledge bases and even be expressed in different representation languages because the ontology serves to unify the knowledge bases co-operating within a single system.

### 3.3 Ontology and representation

A potential source of confusion in the account above is the use of formal specifications at several stages of knowledge-based systems design. There may be a formal specification of concepts in the ontology, and there will be a formal representation of those concepts in the knowledge representation. Yet the two are quite distinct.

In the conceptualisation stage, we can in principle do without formal specifications, using only natural language. The attraction of formal definitions derives from their utility in combating ambiguity inherent in natural language. At this stage, formal expressions are used to achieve the goal of unambiguous specification of concepts. Not all concepts can be defined formally. Every defined concept will ultimately be defined in terms of atomic concepts. Atomic concepts can only acquire meaning by anchoring them in accepted domain knowledge. We have to take recourse to informal explications in natural language to do that. (Providing formal definitions for ‘atomic’ concepts only shifts the matter, because we then need other atomic concepts as basis for our definitions.) The informal explications are part and parcel of the ontology. A representation, on the other hand, is fully formal.

### 3.4 Concepts and instances

Another potential source of confusion is this. In typical accounts (such as that of Genesereth & Nilsson [1987]), the representation language is chosen to be a first-order language such as first-order predicate logic or a description logic. The knowledge representation itself is what logicians (but not chemists or physicists) call a *theory*: a set of axioms in the language. The conceptualisation typically specifies the intended model of the theory in standard model-theoretic terms, that is, as a set of objects that constitute the universe of discourse and a set of relations that hold between those objects.

The problem now is how the objects that constitute the universe of discourse are to be chosen. Genesereth & Nilsson [1987] take a liberal point of view and allow abstract and fictitious objects such as numbers and unicorns besides concrete individuals such as my table or the elephant Clyde. In the description logic community, on the other hand, there appears to be a distinct but implicit preference for concrete individuals that we can point out in reality. This gives rise to the important distinction between concepts and their instances. The distinction provides a clear semantics of the relation between a concept and its superconcept, see Brachman [1977] and Brachman [1985] for an extensive discussion.

We submit that the approach involving concepts and individual instances becomes problematic for scientific concepts. The description logic community’s insistence on concrete objects in the universe of discourse seems to serve the purpose of clarifying the relation between knowledge representation and reality. The knowledge engineer can point at the instances of the concepts that figure in the representation. Matters are not so simple in scientific applications. In mature scientific disciplines such as physics and chemistry, the relation between knowledge and reality defies simple schemes involving concepts and concrete instances. There still is a connection between the knowledge representation and reality, but access to reality proceeds via mediation by physicists and chemists.

For concrete examples, consider the following cases:

1. Whether there are instances of concepts occasionally depends on your philosophical position in the realism *versus* positivism debate. Do atoms and molecules exist? Do the forces exist that we add vectorially to obtain a net force? The latter example is taken from Cartwright [1983]. She concludes:

[Vector addition] is just a metaphor. *We* add forces (or the numbers that represent forces) when we do calculations. Nature does not ‘add’ forces. For the ‘component’ forces are not there, in any but a metaphorical sense, to be added. (*Ibid.*, p. 59; italics Cartwright’s.)

2. Many scientific concepts lack instances, even within the context of their use. An obvious example is the concept of virtual force. The problem is that we need some way to distinguish virtual forces such as the Coriolis force from non-virtual ('real'?) forces such as the one exerted by gravity. It seems incoherent to say that both kinds of force have instances because the distinction in fact turns around the question whether there are instances.
3. For other concepts it seems that the relation between the concept and its instances is not one of specialisation and generalisation, but rather one of concretisation and abstraction. For instance, for pure substances such as  $\text{H}_2\text{O}$  the problem is that there are plenty of samples that we will call water, but none consists of  $\text{H}_2\text{O}$ -molecules exclusively. Whether we are prepared to call a particular sample a sample of  $\text{H}_2\text{O}$  depends on the degree of purity, the nature of the impurities, and the intended use. Thus, what is an instance of the pure substance  $\text{H}_2\text{O}$  in one context may not be an instance of the pure substance  $\text{H}_2\text{O}$  in another context. The set-membership relation does not accommodate this difficulty.
4. Finally, there is the time frame problem. To pursue the  $\text{H}_2\text{O}$  example, " $\text{H}_2\text{O}$ " is intended to apply to all samples of water, past, present, and future. From a scientific point of view it is a problem that most members of the set are unavailable for inspection.

The problems are amenable to a treatment in terms of intensional logics: they are just the kind of problems for which intensional logics have been designed (compare Gamut [1991]). One possible approach would treat concrete sample identifiers as rigid designators across possible worlds. The assertion that a particular sample is an instance of the pure substance  $\text{H}_2\text{O}$  can be conceptualised as: there is at least one possible world in which the sample consists of pure  $\text{H}_2\text{O}$ . Other possible worlds may correspond to different experimental contexts, and for some of those we will want to say that the sample consists of a mixture (that, however, predominantly consists of  $\text{H}_2\text{O}$ ). These are fascinating issues that, however, fall outside the scope of the present account.

For engineering purposes, the problems can be sidestepped (rather than solved). As we shall see below, allowing for abstract and possibly fictitious objects in the universe of discourse is sufficient to arrive at a solution that is satisfactory for the kind of applications we have in mind. We do not insist on a sharp distinction between concepts and instances. But then we have to provide an alternative semantics of the relation between a concept and its superconcept. This will be outlined below.

## Chapter 4

# Chemical background

### 4.1 The concept of sample

If the principle of knowledge as assertions about entities (see section 2.3.1) is accepted, the basis of any ontology consists of a choice of entities and a further choice which properties of those entities count as defining properties.

For chemistry in general, it might seem that substances are a sensible choice of entities. However, there are certain complications. One is, that the idea of a substance itself is an idealisation. No substance mentioned in a chemical textbook can be obtained in its ideal (pure) form. Another complication arises when we want to take care of conflicting information. There is a more fundamental choice of entities possible.

The choice of entities proposed here is based on the observation, that all we know about pure substances has become known through experimentation on samples. The concept of pure substance is an abstraction of the concept of sample.

A *sample* is any concrete and identifiable portion of a substance. In thermodynamics, the technical term for a concrete and identifiable part of the world is *system*. A system is identified by its boundaries: anything within is part of the system, anything else part of the environment. By using the term ‘sample’ rather than ‘system’ we emphasise that we are concerned with a special kind of systems, namely those that consist of a substance. In the simplified account presented here, the substance is always a pure substance. For an example, given a glass of water we can define a system that embraces both the water and the glass. We would not call this system a sample, however, if we are interested in the substance water. In that case, the sample would be the system that consists of the water while the glass is part of the surroundings.

Samples have properties. Their foremost and fundamental property is *uniqueness*. It is the only property of which we are certain. All other properties of samples are determined experimentally or predicted theoretically, and thus may be incorrect. The latter observation is essential because we want the Plinius system to be able to handle conflicts. The uniqueness property is therefore the only property we assert of samples. This is done by attaching to each sample a *unique label*. Obviously, the label serves as defining property and, since it is unique, it can be used as unambiguous concept for the sample in question.

The other property of samples relevant here is the one of chemical composition in terms of pure substances. To specify the chemical composition of a sample in terms of pure substances, we need a conceptual apparatus that enables us to express every pure substance we need and to distinguish between different pure substances if a distinction needs to be made. Chemists have developed such a system over the past two centuries. We will summarise their system here. It takes some background knowledge that will be introduced as we go along.

An important source of information is the set of recommendations for nomenclature and syntax of formulae published by the *International Union of Pure and Applied Chemistry* (IUPAC). These recommendations are regarded as *de facto* standards by the chemical community. Obviously,

recommendations have to cover all concepts of interest and have to enable their users to make the distinctions they want to make. For the present purpose, the recommendations for inorganic chemistry are the most relevant.<sup>1</sup> The latest recommendations, those of 1990, are published in what is called the *Red Book* (Leigh [1990]). Below, we will not repeat the citation but simply refer to the Red Book.

## 4.2 Pure substances

In chemistry, the term *pure substance* refers to some identifiable stuff. ‘Identifiable’ usually means that we can give a complete specification of the chemical composition. This specification, in turn, is given in terms of kinds of atoms and molecules. Pure substances represent the bottom line of chemical analysis in the sense that further analysis, proceeding down to the level of chemical elements, decomposes the substance. If a sample can be shown to consist of several pure substances, it is called a mixture.

Atoms come in kinds. An atom consists of a nucleus with positive charge surrounded by negatively charged electrons. The nucleus itself consists of two kinds of elementary particles: protons (with a positive charge) and neutrons (uncharged). The number of protons is known as the *atomic number* while the *mass number* is the sum of the number of protons and the number of neutrons. Since in neutral atoms the atomic number is equal to the number of electrons, it is the composition of the nucleus that determines the kind of the atom. The atomic number determines the chemical element. For most chemical elements, there exist variants which differ in the number of neutrons. These variants are called isotopes. Chemically, isotopes of the same element almost always behave so similarly that isotopy can be ignored. We will ignore isotopy for the time being. (On the problems caused by the discovery of isotopes and how they are solved in chemistry, see van der Vet [1987].)

The simplest pure substances are those that consist of loose atoms. Helium is an example. Other pure substances arise because atoms can become bonded (that is, held together). Examples are the two substances that make up the major part of the earth’s atmosphere, nitrogen (N<sub>2</sub>) and oxygen (O<sub>2</sub>). Other familiar examples are water (we mean here the chemists’ pure water, H<sub>2</sub>O) and ethanol (CH<sub>3</sub>CH<sub>2</sub>OH, called alcohol in daily life).

Chemists tend to distinguish between pure substances that consist of a single element (such as helium, nitrogen, and oxygen) and those that consist of two or more elements (such as water and ethanol). Confusingly, chemists call substances that consist of a single element ‘elements’, so that the term ‘element’ becomes ambiguous. We will reserve the term ‘element’ for kinds of atoms and use the term ‘elementary substance’ for any substance that consists of a single element.<sup>2</sup> The term ‘compound’ is used for substances that consist of two or more elements.

In the following, we will inspect the Red Book recommendations for writing chemical formulae for pure substances to assemble cues for the ontology. We turn to bonding first.

## 4.3 Bonding

To appreciate the conceptual organisation of chemistry, some understanding of bonding is essential. In a simplified but very useful picture, there are three kinds of bonds: the atomic, metallic, and ionic bonds.

The atomic bond (also called covalent or chemical bond) is regarded as the only ‘true’ chemical bond. It involves typically two, sometimes three or more atoms. A collection of atoms held together by atomic bonds is called a molecule. A satisfactory description can only be given in

<sup>1</sup>This also applies to the Plinius project. Ceramics are inorganic substances.

<sup>2</sup>As far as we are aware (see van der Vet [1987], pp. 21–28), Mendeleev was the first to note the ambiguity explicitly. It was he who proposed the terms ‘element’ and ‘elementary substance’ (or, rather, their Russian equivalents) in the sense used by us. Against this, the Red Book recommends the term ‘atom type’ for what we call ‘element’ and the term ‘element’ for what we call ‘elementary substance’. In practice, chemists keep on using the term ‘element’ in both senses.

terms of quantum mechanics. Examples of common substances that consist entirely of molecules are water, ethanol, sugar, nitrogen, and oxygen. There are instances of samples in which every atom is bonded by an atomic bond to at least one other atom, so that strictly speaking the whole sample forms a single molecule. Examples are diamond (that consists entirely of carbon atoms) and tungsten carbide (WC).

The metallic bond occurs only in solid and liquid states. The metallic bond, too, is a quantum phenomenon. Roughly, the atoms making up a sample share electrons with all other atoms. The shared electrons occupy a region in the energy spectrum known as the conducting band. Molecules cannot be identified. Examples are too well-known to recount here.

The ionic bond, finally, arises because most atoms and many molecules can permanently lose or gain a few electrons, becoming permanently charged. They are then called *ions*. Bonding occurs as a result of electrostatic (Coulombic) interaction. Like the metallic bond, the ionic bond occurs only in the solid and liquid states. An example of a common ionic substance is table salt (NaCl). Many ionic substances involve ions that are themselves molecules, such as soda ( $\text{Na}_2\text{CO}_3$ ) where the carbonate-ion ( $\text{CO}_3^{2-}$ ) is molecular.

As said above, the picture is simplified. It is so in two respects. First, there are interactions between atoms and molecules other than those mentioned, such as hydrogen bonds and Van der Waals interactions. Second, the classification of bonds as belonging to mutually exclusive categories is too simple. This is particularly evident for the ionic bond. Electrostatic interaction accounts only for cohesion. There must also be repulsion, otherwise the substance would collapse. Repulsion can be explained as arising from mutually overlapping electron densities, which is in fact the province of the atomic bond. As a result, no substance is held together by purely ionic bonds.

In the simple picture we want to say that the interactions in an ionic substance are a mixture of electrostatic and molecular interactions. Some accounts (for instance, that by Davidge [1979]) sketch a picture in which a bond is called, say, 65% ionic (and, hence, 35% atomic). Other accounts (for instance, that by Mackay & Mackay [1981]) prefer a dichotomy and employ a rough cut-off criterion based on the difference of electronegativity values.

The distinction between the atomic bond on the one hand and the ionic and metallic bonds on the other has consequences for any ontology of substances that employs numbers to specify composition. As we shall see below, atomic bonds imply absolute numbers while ionic and metallic bonds imply proportions. The way in which these implications are treated in our ontology of pure substances will become apparent below and in the next chapter. The main decision has been to employ a dichotomy. We treat examples such as alumina ( $\text{Al}_2\text{O}_3$ ) and zirconia ( $\text{ZrO}_2$ ) as being ionic. The tacit understanding is that the interaction is mixed.

## 4.4 Formulae for pure substances

The syntax of formulae for pure substances is based on the assumption that pure substances have a fixed composition and that this composition can be expressed as a specification of the constituent elements in natural-number proportions. We note at the outset that this is an abstraction. The numbers of individual atoms involved are enormous; for instance, in one gram of table salt there are roughly  $10^{22}$  atoms. It would be a miracle if any sample were really pure. In practice, no sample ever is pure. Whether the degree of impurity is acceptable depends on the application.

It is also very improbable that we can express chemical composition even for completely pure substances in natural-number proportions, unless there is a mechanism that enforces such proportions at the atomic-molecular level. In entirely molecular substances like water and ethanol, there is such a mechanism in the form of the atomic bond. Every constituent atom in the molecule has a finite and fixed number of atomic bonds with other atoms. Breaking a bond or substituting an atom for another results in a different molecule. A substance that consists of more than one kind of molecules is not a pure substance but, rather, a mixture. Therefore, in formulae such as  $\text{CH}_3\text{CH}_2\text{OH}$  for ethanol the numbers are both the absolute numbers of atoms in the molecule, and the relative numbers (proportions) of atoms in the pure substance ethanol.

The ionic and metallic bonds, on the other hand, are far less restrictive, and composition may vary as in nonstoichiometric phases and alloys. We will ignore variable composition in the context of the present report. A substance involving ionic or metallic bonds is called pure if it can be specified as consisting of elements in natural number proportions. Here the numbers are only proportions because a molecule cannot be identified. Examples include Fe (pure iron, where any sample consists of a number of Fe atoms bonded by metallic bonds) and NaCl (table salt, where any sample consists of  $\text{Na}^+$ - and  $\text{Cl}^-$ -ions in equal proportions).

The Red Book recommendations for writing formulae specifying pure substances can be summarised as follows. The basic form of any formula is a list of elements with subscripts to specify proportions. If the proportion is 1, the subscript is omitted. Parentheses and square brackets may be used to specify grouping.

Further recommendations depend on the nature of the substance. If it is molecular, the subscripts are also the absolute numbers of atoms in the molecule. Thus, atmospheric oxygen is  $\text{O}_2$  rather than simply O; and  $\text{NO}_2$  and  $\text{N}_2\text{O}_4$  are different substances (both exist). The only exception to this rule is substances for which the molecule strictly speaking covers the whole sample, as in diamond and tungsten carbide. For such substances, the subscripts do not specify absolute numbers but simplest proportions: diamond is written as C and tungsten carbide as WC.

If the substance involves ionic and/or metallic bonds, absolute numbers can only be specified for the molecular subunits, if any. For the rest, the subscripts express proportions only and are required to be written as simplest proportions. Thus, it is NaCl and Fe. Parentheses are used to identify the ions involved, but they are left out if the ion consists of a single atom or if the subscript specifying the ion's proportion is 1. An example is the formula for hydroxylapatite,  $\text{Ca}_5(\text{PO}_4)_3\text{OH}$ , where the  $\text{PO}_4$  ion has been surrounded by parentheses but the Ca and OH ions not. The presence of molecular subunits may give rise to formulae which seem to deviate from the simplest proportions rule: it is  $\text{Hg}_2\text{Cl}_2$  rather than HgCl because of the presence of a molecular ion,  $\text{Hg}_2^{2+}$ . On the other hand, the formula  $\text{Ca}_{10}(\text{PO}_4)_6(\text{OH})_2$  (given in the example abstract of figure 2.1) is ill-formed. The subscripts do not conform to the simplest proportions rule although there is no molecular subunit that might justify this deviation.

## 4.5 Non-existing substances

The Red Book syntax is entirely silent about whether a substance for which we can write a formula does, in fact, exist. Thus, the syntax allows formulae such as  $\text{Na}_{13}\text{O}_{17}$  and  $\text{He}_5\text{CrPt}_6$  even though the substances to which they refer are not known to exist. Note, however, that we can still interpret such formulae: if not, we could not have said that the corresponding substances are not known to exist. In Fregean terms, therefore, any formula that conforms to the syntax has *sense* but not necessarily *reference*.

The ability to write formulae for substances not known to exist serves two purposes. First, a chemist may want to inform her colleagues that she has tried to synthesise a particular substance never made before but failed to do so. The chemist has to specify the substance if she wants to communicate her result. Second, the syntax accommodates future developments to a certain extent. For instance, forty years ago every chemist would have subscribed to the belief that noble gases do not form compounds. The belief was shown unfounded when in 1962 Bartlett synthesized the compound  $\text{XePtF}_6$  under relatively mild conditions; many other noble gas compounds followed. Any recommendation that included information on the existence of substances would have had to be rewritten in 1962, but the Red Book recommendations allowed a formula in a straightforward way.

A more fundamental reason why recommendations like those of the Red Book do not include information about the existence of compounds is that such information is simply not available. We go in some detail here because the point is relevant for ontology building.

Suppose we had wanted to confine the ontology of pure substances to those substances that can exist, then we are confronted with the problem of specifying that set. The notion of existence of substances can be operationalised in terms of stability. A substance exists if and only if it is stable,

at least over some finite time interval. There are in principle two ways to determine the stability of a compound: it can be predicted theoretically or determined empirically. For ontological purposes we need the theoretical way because the empirical way is by definition open-ended: it has the set of stable substances depend on the ingenuity of synthetic chemists and on the ability to observe and identify substances over ever shorter time intervals.

The theoretical way attempts to predict stability on the basis of composition. The historically first candidate for providing predictions is equilibrium thermodynamics. It provides a clear-cut stability criterion. Unfortunately, it is not useful because it would, for instance, rule out almost every organic compound. Apparently stability is a kinetic, not a thermodynamic concept. Quantum chemistry might be a candidate even though quantum chemical calculations are very laborious. The problem with such calculations is that they involve very many assumptions, not all of which can be confirmed.

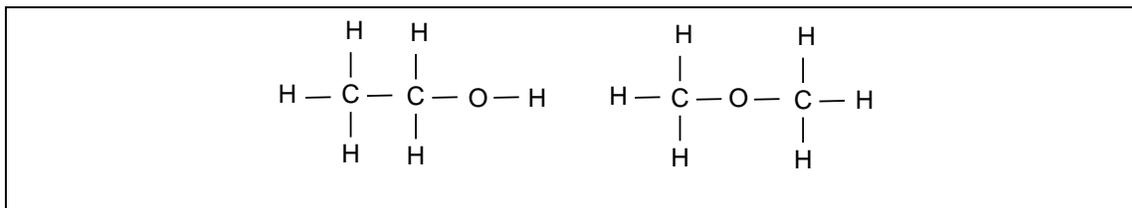
The lack of usable theories explains the mistakes that have been made in the past, and undoubtedly are still made, in predicting the stability of compounds. Famous examples involve the noble gas compounds mentioned above and polywater. Polywater, a reputed polymer of ordinary water, was the subject of a fierce debate in the 1970's. Quantum chemical calculations were used as argument by several parties, but some calculations predicted stability while others ruled it out. The debate ended when consensus was reached on the view that polywater, if it forms at all, is so unstable that it could not have been observed.

If domain experts do not possess a stability criterion, ontology developers cannot incorporate it into an ontology. Both the full Plinius ontology and the ontology of pure substances allow concepts for non-existing substances.

## 4.6 Structure

The simplest way of writing a formula just lists the elements and their proportions. The result is called the *empirical formula*. For instance, NaCl and H<sub>2</sub>O are empirical formulae. The Red Book allows formulae that are more expressive through the addition of structure information. In practice, chemists very often employ these more expressive formulae.

Structure information is often added to disambiguate an otherwise ambiguous empirical formula. For instance, the empirical formula for ethanol, C<sub>2</sub>H<sub>6</sub>O, is also the empirical formula for the pure substance methoxymethane (also known as dimethyl ether or just ether). The two substances are different because the atoms are bonded in a different way, see figure 4.1. Since the empirical formula is ambiguous, chemists write more expressive formulae: CH<sub>3</sub>CH<sub>2</sub>OH for ethanol (as we in fact have done above) and CH<sub>3</sub>OCH<sub>3</sub> for methoxymethane. The phenomenon of several pure substances corresponding to the same empirical formula is known as *isomerism*.



**Figure 4.1:** STRUCTURAL ISOMERISM. The two molecules shown are ethanol (left) and methoxymethane (right). The two pure substances display quite different chemical and physical behaviour because the atoms are bonded in different ways.

Even when isomerism does not occur, chemists may prefer a more expressive formula. The reason is pragmatic: by making structure explicit, the reader does not have to derive the intended structure from the formula. An example is the formula for hydroxylapatite, that we have written as Ca<sub>5</sub>(PO<sub>4</sub>)<sub>3</sub>OH above. The empirical formula, Ca<sub>5</sub>HO<sub>13</sub>P<sub>12</sub>, is unambiguous but hard to interpret for a working chemist. In these cases, the possibility of isomerism might also play a role. It can be argued that the lack of ambiguity in at least some of these cases reflects our current state of

knowledge rather than some generally valid principle. Isomerism can only be ruled out if we know for sure that no isomer can ever be synthesised, but as we saw in section 4.5 we can never be sure. Prevention is better than cure. Inclusion of structure information forestalls ambiguities instead of curing them once they arise.

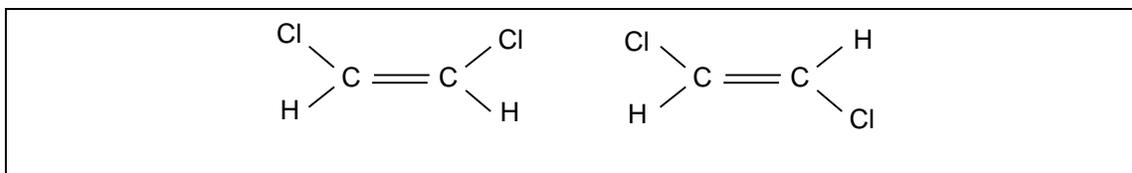
The ontology builder faces the question whether structure information has to be included in the ontology or not. The answer is obvious when isomerism needs to be taken into account. For Plinius, the current development corpus does not contain a single mention of an isomeric compound, and we can do without structure information. But then the pragmatic reason still holds.

We have decided to ignore isomerism in the first try at an ontology of pure substances. Some structure information will be incorporated for pragmatic reasons. One advantage follows from the argument of prevention rather than cure. For another advantage, the choice considerably facilitates translation of ontology concepts from and into recommended IUPAC chemical formulae and names.<sup>3</sup>

Ignoring isomerism in the first try is advantageous because it allows us to assess what price we have to pay if we decide to accommodate isomerism after all. There are grounds to suppose that, even in the context of Plinius, concentric extension of the corpus will force us to distinguish between isomers. So we close this chapter with a overview of isomerism.

There are two and perhaps three types of isomerism.

(a) **Structural isomerism.** Structural isomers consist of the same numbers of atoms but differ in the way they are bonded. An example has been given above: ethanol and methoxymethane, see figure 4.1.



**Figure 4.2:** STEREOISOMERISM. The two molecules shown are *cis*-1,2-dichloroethene (left) and *trans*-1,2-dichloroethene (right). These molecules would have been identical if rotation about the double bond were free, but under normal circumstances this is not the case. Therefore, at least one bond must be broken to convert one of these molecules into the other so that the two substances they represent are different. The difference is, for instance, manifest in the fact that the *cis*-isomer has an appreciable dipole moment while the *trans*-isomer has a zero dipole moment.

(b) **Stereoisomerism.** Stereoisomers are structurally identical but differ in the way the atoms are located relatively to each other in space. An example is given in figure 4.2.

(c) **Bond-stretch isomerism.** Just for completeness we mention bond-stretch isomerism, where the isomers differ in the length of one or more atomic bonds. Bond-stretch isomerism has been reported for an inorganic compound in 1971, but the observation turned out to be incorrect. Although theoretically bond-stretch isomerism is possible, no confirmed examples have been found until now. See Parkin [1992] for details.

<sup>3</sup>For the related problem of converting formulae into Red Book names, see Eggert, Jacob & Middlecamp [1992] and Eggert, Jacob & Middlecamp [1993].

# Chapter 5

## A simple ontology of pure substances

### 5.1 Introduction

In this chapter, we will develop a simple ontology of pure substances and discuss some of its properties. The ontology is intended to illustrate the design decisions identified in section 2.3. In particular, the ontology clarifies the bottom-up approach (section 2.3.5) to construct a conceptual construction kit (section 2.3.4).

It is obvious to define pure substances by their chemical constitution, in other words, by having a construction rule that defines pure substances in terms of chemical elements and natural numbers. We will discuss a first try that does not take isomerism or isotopy into account (compare sections 4.2 and 4.6). We will identify hierarchical relations for this simple version. We then turn to a summary of extensions needed to handle isomerism and isotopy. The present chapter thus discusses several ontologies rather than a single one. The relations between these ontologies are discussed in the last section.

The set-up is based on the following rules:

**Rule 1** Every definition implicitly defines a set of concepts. (This is the operationalisation of the principle of conceptual construction kit.)

**Rule 2** It is always allowed to define subsets of sets defined earlier.

**Rule 3** Concepts (including relations) not defined in this ontology cannot be used for constructing assertions. (This is the operationalisation of the requirement that the ontology be limitative, see sections 3.1 and 3.2.)

As noted by Gruber [1991] it is desirable to have both an informal and a formal definition of concepts. The formal definition serves to pin down the meaning as precisely as possible, while the informal counterpart serves to convey the intended meaning to human readers. The two are thus complementary and are not required to state exactly the same thing. In our ontology the informal definition plays an additional and very important role: it serves to anchor the formal definition into accepted domain knowledge. For instance, the formal definition of concepts for chemical elements defines elements in terms of atomic numbers. The informal definition explains that the numbers are to be understood as atomic numbers and thus define chemical elements as they are commonly understood in chemistry. This is enough for our purposes, since chemical elements are part of consensus knowledge. It is not correct to regard the set of formal definitions as constituting a formalisation, see section 3.3.

The resulting set-up is as follows. Every definition will be given here in a formal language, namely first-order predicate logic enriched with sets and arithmetic. As announced in section 2.3.2,

sets and arithmetic are taken ‘as is’. An informal explication that is part and parcel of the ontology precedes each formal definition.

The atomic concepts defined here are introduced by means of intensional definitions because they are all based on numbers. In two cases, primitives are introduced to pin down the meaning of the numbers involved. It is also possible to introduce atomic concepts by means of extensional definitions of sets. This is done in the full Plinius ontology, but not here.

Complex concepts are defined implicitly by means of construction rules. They take the form of definitions that supply rules for determining whether a complex concept is well-formed. Each definition thus introduces a large (as will be seen below, often technically infinite) number of concepts. The definitions themselves are not part of the ontology. Any ontology consists of concepts. This ontology is no exception, even though we cannot write out the whole list (and if we could, it would be very inconvenient). The function of definitions is to specify concepts; most definitions do so implicitly. Definitions are to be situated at a meta-level.

All concepts that fulfill the demands of a particular definition have something in common, and a *type* will be associated with each definition. Types will be written in sans serif font: for instance, element and pure substance. For each ontology concept we can determine the type by finding out to which definition it conforms.

It is important to distinguish types from concepts. Types are introduced by definitions. The definitions themselves are not part of the ontology (but the concepts they define are); they are situated at a meta-level. So, at least for the time being, types are also situated at the meta-level. They are a means to discuss properties of the ontology.

Concepts of the same type can be grouped into sets, thus achieving clarity of exposition. Sets of atomic concepts are denoted by single-letter uppercase symbols, as in  $E$  (the set of chemical elements). Sets of complex concepts are denoted by two-letter uppercase symbols: for instance,  $PS$  (the set of pure substances). An explicitly defined subset of a set defined earlier, as allowed by rule 2, will be denoted by capitalised three-letter symbols like  $Pho$ . Finally, if we want to denote a set irrespective of its kind (set of atomic concepts, set of complex concepts or defined subset), we use Euler Fraktur symbols like  $\mathcal{C}$ .

We allow ourselves to deviate from these conventions if more familiar symbols are available. Thus, we will denote the set of natural numbers as  $\mathbb{N}$  and the set of concepts for whole numbers unequal zero as  $\mathbb{Z}_{\neq 0}$ .

Since the word ‘element’ can be used for both chemical element and element of a set, we will speak about *members* of a set from now on. ‘Element’ then unambiguously refers to a chemical element.

## 5.2 Numbers

For the present ontology we only need a few sets of concepts for numbers. We take the number system, including arithmetical operations, as given. We use standard symbols. Just for completeness, we include the definitions here. A more principled introduction would have been possible because there are subset-relations between the sets introduced here. However, that seemed over-eager in view of the fact that all these sets are well-known.

### Definition 1 (Whole number)

A concept of type whole number is defined as any integer. The set of whole numbers is called  $\mathbb{Z}$ .  
□

### Definition 2 (Natural number)

A concept of type natural number is defined as any member of the set  $\mathbb{N}$ , defined as

$$\mathbb{N} =_{\text{Df}} \{n \mid n \in \mathbb{Z} \wedge n > 0\}$$

□

**Definition 3 (Whole number unequal zero)**

A concept of type whole number unequal zero (strictly positive or strictly negative whole number) is any member of the set  $\mathbb{Z}_{\neq 0}$ , defined as

$$\mathbb{Z}_{\neq 0} =_{\text{Df}} \{z \mid z \in \mathbb{Z} \wedge z \neq 0\}$$

□

**Definition 4 (Strictly positive rational number)**

A concept of type strictly positive rational number is any member of the set  $\mathbb{Q}_{>0}$ , defined as

$$\mathbb{Q}_{>0} =_{\text{Df}} \{q \mid q = n/m \wedge n \in \mathbb{N} \wedge m \in \mathbb{N}\}$$

□

It might be questioned whether it is necessary to have the full, denumerably infinite sets of whole and natural numbers for the purpose at hand. For instance, pure substances in inorganic chemistry surely consist of small numbers of atoms? The point is that, given the setup of the present ontology, it is no problem to have the full set. Formally it is no problem because we specify concepts implicitly. Computationally it is no problem either because any sensible implementation will take recourse to built-in routines for handling numbers. Finally, the numbers of atoms may be small, they are larger than most non-chemists think. Recent research into metal-ligand clusters has identified compounds comprising hundreds of atoms. At the time of writing, the record for an inorganic compound was held by a group at the University of Karlsruhe who had synthesised a cluster with the formula  $\text{Cu}_{146}\text{Se}_{73}[\text{P}(\text{C}_6\text{H}_5)_3]_{30}$  (Krautscheid et al. [1993]). Larger clusters will be made in the near future. Any limit is somewhat arbitrary, so if we can avoid setting a limit we should do that.

## 5.3 Samples and their constitution

Recalling the discussion in section 4.1, we further need concepts for samples and for the relation that expresses their chemical constitution. Samples are defined by a unique identifier. It is obvious to use a natural number for the purpose. We need a way to distinguish this number from numbers used for other purposes. We do this by writing a concept of type sample as the value of the function `sample(n)`, where  $n \in \mathbb{N}$  is the unique label. Concepts for samples are regarded as atomic concepts on the ground that the formally undefined function `sample(n)` is an essential ingredient of the definition.

**Definition 5 (Sample)**

A concept of type sample is any member of the set  $S$ , where  $S$  is the range of the function:

$$\text{sample}(n) \text{ with } n \in \mathbb{N}$$

□

For sample constitution, we will only introduce the `consists` predicate. We will run ahead of our discussion and assume that  $PS$ , the set of concepts of type pure substance, is known. (It is actually defined by definition 10 below.)

**Definition 6 (Chemical constitution specification)**

The chemical constitution of a sample is asserted by means of a sentence of the form

$$\text{consists}(s, p)$$

with  $s \in S$  and  $p \in PS$ .

□

The rest of this chapter can be regarded as a specification of the second argument of the `consists` predicate.

## 5.4 Chemical elements

Chemical elements are defined by their atomic number  $Z$ . It makes sense to introduce concepts for chemical elements so that elements are characterised by a natural number that stands for  $Z$ . As was the case for samples, we need a label to distinguish this number from numbers that serve other purposes. This is done in the same way as for concepts of type `sample`, namely by defining an element as the value of a particular function called `atom_kind`. The argument of the function is the atomic number  $Z$ . Consequently, concepts for chemical elements are regarded as atomic concepts.

No upper bound is defined for  $Z$  for reasons that are analogous to the reasons for allowing concepts for non-existing substances (compare the discussion in section 4.5). New elements are synthesized at a rate of about one per year. Currently, the highest atomic number reported is 111. Although every element with  $Z > 96$  is so unstable that practical use is very improbable, the absence of an upper bound makes the ontology useful for applications outside materials science.

This results in the following definition.

### Definition 7 (Chemical element)

A concept of type chemical element is any member of the set  $E$ , where  $E$  is the range of the function:

$$\text{atom\_kind}(Z) \text{ with } Z \in \mathbb{N}$$

$Z$  is called the atomic number.

□

Appendix A lists chemical elements with their official IUPAC names and symbols, atomic numbers and, where appropriate, atomic weights. The list includes all elements for which, at the time of writing of the present report, the IUPAC has published a recommended name and symbol.

## 5.5 Groups

### 5.5.1 Considerations

The next step is that of the construction rule for pure substances. Given the sets  $E$  and  $\mathbb{N}$ , all we can do is specify the conceptual analogues of empirical formulae for pure substances. In section 4.6, we have argued that empirical formulae are ambiguous when isomers are involved and impractical in many other cases. We announced there that the first try at an ontology will ignore isomerism but nevertheless will include some structure information for practical reasons.

The structure information to be included takes the form of a grouping level intermediate between chemical elements and pure substances. This level embraces concepts of type `group`. In this simple scheme, every group consists of chemical elements in amounts specified by natural numbers, while every pure substance consists of groups in natural number proportions.

Groups are chosen such that, for any given pure substance, every concept of type `group` corresponds to a molecular assembly or to an atom that is not connected to another atom by an atomic bond. The background is provided by the simplified picture of chemical bonding outlined in section 4.3. The idea applies especially to ionic compounds. Every ionic species corresponds to a concept of type `group`. Thus, in hydroxylapatite  $\text{Ca}_5(\text{PO}_4)_3\text{OH}$ , the groups are `Ca`, `PO4`, and `OH`. As can be seen, this choice for constructing groups resembles the choices made in writing formulae such as  $\text{Ca}_5(\text{PO}_4)_3\text{OH}$ ; parentheses would have surrounded `OH` if there had been more than one such ion in the unit formula.

It is a weakness of this approach that there is no way to decide automatically what the groups are. One way to do this utilises valencies of the atoms. This is the reason why, for instance, Napoli [1992] includes valency numbers in his representations of atoms. Against this, chemists will point out that valencies have limited predictive power. There have in the past been attempts to present a formal syntax of chemical species using valencies, Mulckhuysen [1960] probably being

the most elaborate one. All have failed because atomic bonding is a phenomenon that can only be understood in a quantum mechanical context. The complexity of quantum mechanical calculations and their sometimes unexpected outcomes defy simple schemes such as that involving valencies. Confirmed examples include the unusual valency of carbon in carbon monoxide (CO) and the unexpected negative valency of calcium in the  $\text{Ca}^{2-}$  ion.

We are now sufficiently prepared for stating the shape of complex concepts of type group. A group consists of a specification of the constituent chemical elements with their amounts. Amounts are absolute rather than relative numbers, because the atoms are bonded by chemical bonds. An element and its amount in a group belong together, and they are best collected into an ordered pair  $\langle e, n \rangle$ , with  $e \in E$  and  $n \in \mathbb{N}$ . We do not want to be bothered by complications caused by ordering of such pairs in the concept for a whole group and by redundancy. This can be obtained by collecting all ordered pairs for a group into a set. To illustrate, three of the many ways to write the concept for the phosphate ( $\text{PO}_4$ ) group are (see table 5.1 for the mapping of atomic numbers occurring in the present chapter onto element symbols):

$$\begin{aligned} & \{ \langle \text{atom\_kind}(15), 1 \rangle, \langle \text{atom\_kind}(8), 4 \rangle \}_{GR} \\ & \{ \langle \text{atom\_kind}(8), 4 \rangle, \langle \text{atom\_kind}(15), 1 \rangle \}_{GR} \\ & \{ \langle \text{atom\_kind}(15), 1 \rangle, \langle \text{atom\_kind}(8), 4 \rangle, \langle \text{atom\_kind}(15), 1 \rangle \}_{GR} \end{aligned}$$

where the subscript  $GR$  serves as a reminder of the fact that the set in question is a concept of type group. All these constructs are just different ways of writing the same concept. The ordering of members in a set is unimportant so that the first and second variants are the same. An member that is specified twice, here  $\langle \text{atom\_kind}(15), 1 \rangle$  in the third variant, counts as a single member so that the third variant is the same as the first two.

From a chemical point of view, two further constraints must be imposed on concepts of type group. One constraint demands for obvious reasons that the set be non-empty. The other constraint demands that no element occur in more than one ordered pair in any concept of type group. The latter constraint rules out a group such as

$$\{ \langle \text{atom\_kind}(15), 1 \rangle, \langle \text{atom\_kind}(8), 3 \rangle, \langle \text{atom\_kind}(8), 1 \rangle \}_{GR}$$

which, if written as the chemical formula  $\text{PO}_3\text{O}$ , is seen to be ill-formed. This constraint will return in the definition of concepts of type pure substance, below. We will refer to it as the *chemical constraint*.

### 5.5.2 Definition of concepts of type group

The considerations sufficiently prepare for the full definition of concepts of type group.

#### Definition 8 (Group)

A concept of type group is any member of the set  $GR$ , intensionally defined as

$$GR =_{\text{Df}} \{ g \mid g \subset E \times \mathbb{N} \wedge g \neq \emptyset \wedge \forall vxy [(\langle v, x \rangle \in g \wedge \langle v, y \rangle \in g) \rightarrow x = y] \}$$

A group is constructed such that it collects all atoms that are bonded to each other by atomic bonds.

□

The chemical constraint is handled by the last clause in the intensional definition. It effectively demands that, if two tuples  $\langle v, x \rangle$  and  $\langle v, y \rangle$  (with  $v$  an element and  $x$  and  $y$  natural numbers) are member of the same group, then  $x = y$ . That entails  $\langle v, x \rangle = \langle v, y \rangle$ . Since any concept of type group is a set, this means that we have just listed the same member of the set twice and the two count as a single member.

### 5.5.3 Absolute constitutions

It is useful to note that a group concept in fact specifies a list of parts. We can in principle use the same construction to specify the list of parts of a car; the list can be regarded as a concept for that car. A car parts list can also be written as a set of pairs such that the first member specifies a kind of parts and the second member the absolute amount. The chemical constraint applies to other part lists as well. We would call a car parts list sloppy or confusing if it contained two entries for the same kind of wheels. The amounts obviously are required to be natural numbers. It makes no sense to put zero or minus two wheels on a parts list. Rational numbers or qualitative values specify relative rather than absolute amounts. The generalisation therefore applies to the kinds of parts.

We will call the generalised version of the parts list with absolute amounts an *absolute constitution*.<sup>1</sup> Absolute constitutions can be defined, thus preparing for the generalisation of the bottom-up approach to be presented in chapter 6. Each absolute constitution, denoted  $acons$ , is a non-empty set of tuples:

$$acons =_{\text{Df}} \{ \langle c_1, n_1 \rangle, \langle c_2, n_2 \rangle, \dots, \langle c_m, n_m \rangle \}$$

where each  $c_i$  is a concept for a kind of parts and each  $n_i$  is a natural number. The chemical constraint implies that, in each set  $acons$ , all  $c_i$  are unique but any two or more  $n_i$  can be equal.

The definition of concepts of type group serves as a template for the definition of absolute constitutions. Definition 8 defines a set of absolute constitutions by laying down from which set  $\mathcal{C}$  can be chosen. A car parts list results from another choice of  $\mathcal{C}$ 's. We will write  $\mathcal{ACONS}(\mathcal{C})$  to denote an absolute constitution, where  $\mathcal{C}$  is the set from which the kinds of components or parts have to be taken. We define:

**Generalised definition 1 (Absolute constitution)**

A set of *absolute constitutions* is any set  $\mathcal{ACONS}(\mathcal{C})$  such that

$$\mathcal{ACONS}(\mathcal{C}) =_{\text{Df}} \{ acons \mid acons \subset \mathcal{C} \times \mathbb{N} \wedge acons \neq \emptyset \wedge \forall vxy [(\langle v, x \rangle \in acons \wedge \langle v, y \rangle \in acons) \rightarrow x = y] \}$$

where  $\mathcal{C}$  is a previously defined set of component types.  $\mathbb{N}$  is the set of natural numbers, as defined by definition 2.

□

Using the generalised definition 1, we could have written definition 8 as:

$$GR =_{\text{Df}} \mathcal{ACONS}(E)$$

## 5.6 Ions

### 5.6.1 Definition of concepts of type ion

Ions are groups with a charge. The charge can be specified by a number  $z \in \mathbb{Z}_{\neq 0}$  (*i.e.*, whole numbers unequal zero). The definition is obvious:

**Definition 9 (Ion)**

A concept of type ion is any member of the set  $IO$ , intensionally defined as

$$IO =_{\text{Df}} \{ \langle g, c \rangle \mid g \in GR \wedge c \in \mathbb{Z}_{\neq 0} \}$$

where  $c$  specifies the charge of the ion.

□

<sup>1</sup>The term 'constitution' is borrowed from chemistry, where it contrasts with 'configuration'. This contrast is precisely what we mean. Compare section 5.13 below.

## 5.6.2 Attribute combinations

Definition 9 can also be generalised. It defines a set of concepts such that any concept is fully characterised by two attributes chosen from sets defined before. The relationship between the two attributes is not formally defined. A combination fully and unambiguously defines a concept only within the context of the ontology, because the informal part explicates the missing information. We will call such constructions *attribute combinations*. Any attribute combination is formally specified by giving the sets  $\mathcal{C}_i$  from which the attributes have to be chosen. The  $\mathcal{C}_i$  have to be collected in a tuple because we want to accommodate cases where the same set occurs two or more times. Also, ordering information may be important. We obtain:

### Generalised definition 2 (Attribute combination)

A set of *attribute combinations* is any set  $ACOMB(\langle \mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_i, \dots, \mathcal{C}_m \rangle)$  (where  $m \geq 1$  and finite) such that:

$$ACOMB(\langle \mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_i, \dots, \mathcal{C}_m \rangle) =_{\text{Df}} \{ acomb \mid acomb \in \mathcal{C}_1 \times \mathcal{C}_2 \times \dots \times \mathcal{C}_i \times \dots \times \mathcal{C}_m \}$$

where every set  $\mathcal{C}_i$  is required to be defined previously.

□

Using generalised definition 2, we can define concepts of type ion in a more compact way:

$$IO =_{\text{Df}} ACOMB(GR, \mathbb{Z}_{\neq 0})$$

## 5.7 Pure substances

### 5.7.1 Definition of concepts of type pure substance

For pure substances, the construction employed for groups can be used with ‘group’ substituted for ‘element’. The amounts are specified as natural numbers. For pure substances, however, they are proportions rather than absolute amounts because we have collected all molecules in groups (compare also the discussion in section 4.4). Therefore the amounts have to fulfill the demand that the greatest common divisor is one.

Any concept of type pure substance is expressed as a non-empty set of ordered pairs  $\langle g, n \rangle$  with  $g \in GR$  and  $n \in \mathbb{N}$ , where the  $n$ ’s are such that their greatest common divisor is one. This expression is wholly canonical given the conventions for concepts of type group. The chemical constraint here takes the form of the demand, that no group may occur in more than one ordered pair in any concept of type pure substance.

The chemical constraint involves a subtlety that has to be pointed out to non-chemists. The groups  $\{\langle \text{atom\_kind}(80), 1 \rangle\}_{GR}$  and  $\{\langle \text{atom\_kind}(80), 2 \rangle\}_{GR}$  are different. The first group denotes Hg-atoms not bonded by atomic bonds and the second group denotes molecular subunits consisting of two Hg-atoms, compare section 4.4. Therefore a concept of type pure substance in which both groups occur does not violate the chemical constraint and therefore is well-formed. In practice, both might occur in the same pure substance. In this case we are not aware of confirmed examples.

The definition becomes:

### Definition 10 (Pure substance)

A concept of type pure substance is any member of the set  $PS$ , intensionally defined as

$$PS =_{\text{Df}} \{ p \mid p \subset GR \times \mathbb{N} \wedge p \neq \emptyset \wedge \text{gcd}(\text{range}(p)) = 1 \wedge \forall xy [( \langle v, x \rangle \in p \wedge \langle v, y \rangle \in p ) \rightarrow x = y] \}$$

□

The rule of simplest proportions is formalised in the clause  $\text{gcd}(\text{range}(p)) = 1$ .  $p$  is a set of tuples of the form  $\langle g, n \rangle$  with  $g$  a group and  $n$  a natural number. The function  $\text{range}(p)$  returns the set of all  $n$ ’s that occur in the tuples. The function  $\text{gcd}(y)$ , where  $y$  is required to be a set of whole numbers, returns the greatest common divisor of all numbers in  $y$ . Then, simplest proportions have been used if and only if  $\text{gcd}(\text{range}(p)) = 1$ .

### 5.7.2 Relative constitutions

Just as the definition of groups can be generalised to yield a definition of absolute constitutions, so the definition of pure substances can be generalised to yield a definition of (what will be called) *relative constitutions*. A relative constitution is also a parts list, but this time we are only interested in the proportions of the various kinds of parts. Any relative constitution  $rcons$  can be expressed as a non-empty set of tuples:

$$rcons = \{\langle c_1, a_1 \rangle, \langle c_2, a_2 \rangle, \dots, \langle c_i, a_i \rangle, \dots, \langle c_m, a_m \rangle\}$$

where each  $c_i$  is a concept for a kind of part and each  $a_i$  a concept for a proportion. The chemical constraint applies. In the same  $rcons$ , no  $c_i$  may occur in more than one tuple.

Since the  $a_i$  express proportions, they are rational numbers  $\neq 0$  that can be scaled by demanding that, for any set  $rcons$ , their sum equals some pre-determined value that will be called the *scaling constant*. Often, the scaling constant is one but another choice was seen in the definition of concepts of type pure substance. The numbers occurring in those concepts of course are also rational numbers, but scaling is chosen such that all denominators are equal to one.

The sum of proportions can be calculated by means of a function  $\text{sumrange}(s)$ , where  $s$  is a set of ordered pairs. The function can be defined recursively as follows:

$$\begin{aligned} \text{sumrange}(\{\langle x, y \rangle\}) &= y \\ \text{sumrange}(s \cup \{\langle x, y \rangle\}) &= \text{sumrange}(s) + y \end{aligned}$$

In principle it is also possible to choose qualitative values for the  $a_i$ . However, qualitative values in relative constitutions can always be replaced by an ordering of unknown quantitative values. We also know that the unknown values add up to the scaling constant for any set  $rcons$ . We can discount cases involving a set  $rcons$  with only a single tuple, because in that case no qualitative value makes sense. In the next simplest case, that involving two tuples, we can imagine one proportion specified as being **large** and the other as being **small**. This can be rendered in qualitative terms by saying  $a_1 > a_2$ . Additionally,  $a_1 + a_2 = sc$  (with  $sc$  the scaling constant).

We obtain:

#### Generalised definition 3

A set of *relative constitutions* is any set  $\mathcal{RCONS}(\mathcal{C})$  such that

$$\mathcal{RCONS}(\mathcal{C}) =_{\text{Df}} \{rcons \mid rcons \subset \mathcal{C} \times \mathbb{Q}_{>0} \wedge rcons \neq \emptyset \wedge \text{sumrange}(rcons) = sc \wedge \forall vxy [(\langle v, x \rangle \in rcons \wedge \langle v, y \rangle \in rcons) \rightarrow x = y]\}$$

where  $\mathcal{C}$  is a previously defined set of component types and  $sc$  is a scaling constant.  $\mathbb{Q}_{>0}$  is defined by definition 4.

□

Generalised definition 3 immediately allows us to define  $PS$ , the set of concepts of type pure substance, as

$$PS =_{\text{Df}} \mathcal{RCONS}(GR)$$

Substituting  $GR$  by using generalised definition 1 yields the more interesting variant:

$$PS =_{\text{Df}} \mathcal{RCONS}(\mathcal{ACONS}(E))$$

This expression is the equivalent in our ontology of the assertion that any pure substance consists of chemical elements.

$Z$	Symbol	Name	$Z$	Symbol	Name
1	H	hydrogen	20	Ca	calcium
2	He	helium	22	Ti	titanium
8	O	oxygen	35	Br	bromine
9	F	fluorine	53	I	iodine
11	Na	sodium	56	Ba	barium
15	P	phosphorus	80	Hg	mercury
17	Cl	chlorine			

**Table 5.1:** ATOMIC NUMBERS MAPPED ONTO ELEMENT SYMBOLS AND NAMES. The table maps atomic numbers onto element symbols and names for every atomic number  $Z$  used in the present chapter to construct a concept `atom_kind( $Z$ )`.

## 5.8 Examples

We will illustrate the ontology of pure substances by constructing concepts for six pure substances: helium, oxygen, ozone, table salt, barium titanate, and hydroxylapatite. For readability, we will utilise abbreviations for groups of the form  $g_i$ , with  $i$  a running index. Table 5.1 maps atomic numbers onto the more familiar element symbols and names. See appendix A for more information.

### Example 1 (Helium)

Helium is an elementary substance that consists of unbonded atoms. The group involved is:

$$g_1 =_{\text{Df}} \{\langle \text{atom\_kind}(2), 1 \rangle\}_{GR}$$

The concept for the pure substance helium is:

$$\{\langle g_1, 1 \rangle\}_{PS}$$

where the subscript  $PS$  serves as a reminder of the fact that the set involved is a concept of type pure substance.

### Example 2 (Elementary substances containing oxygen)

The gas oxygen consists of diatomic molecules  $O_2$ . Since the two oxygen atoms are bonded through an atomic bond, they have to be collected in a single group:

$$g_2 =_{\text{Df}} \{\langle \text{atom\_kind}(8), 2 \rangle\}_{GR}$$

The concept for the pure substance oxygen is:

$$\{\langle g_2, 1 \rangle\}_{PS}$$

Mark that the setup allows unambiguous distinction between chemical elements and elementary substances, and between elementary substances and compounds.

In the atmosphere, another elementary substance involving oxygen occurs as well: ozone,  $O_3$ . All three atoms are bonded together by atomic bonds, so the group involved is

$$g_3 =_{\text{Df}} \{\langle \text{atom\_kind}(8), 3 \rangle\}_{GR}$$

and the pure substance

$$\{\langle g_3, 1 \rangle\}_{PS}$$

### Example 3 (Table salt)

Table salt,  $NaCl$ , is an ionic compound that consists of two kinds of ions: Na ions and Cl ions. Therefore, we need two groups:

$$g_4 =_{\text{Df}} \{\langle \text{atom\_kind}(11), 1 \rangle\}_{GR}$$

$$g_5 =_{\text{Df}} \{\langle \text{atom\_kind}(17), 1 \rangle\}_{GR}$$

The concept for the pure substance is easily seen to be:

$$\{\langle g_4, 1 \rangle, \langle g_5, 1 \rangle\}_{PS}$$

**Example 4 (Barium titanate)**

Barium titanate,  $\text{BaTiO}_3$ , is commonly written as being composed of Ba- and  $\text{TiO}_3$ -ions. The groups then are:

$$\begin{aligned} g_6 &=_{\text{Df}} \{\langle \text{atom\_kind}(56), 1 \rangle\}_{GR} \\ g_7 &=_{\text{Df}} \{\langle \text{atom\_kind}(22), 1 \rangle, \langle \text{atom\_kind}(8), 3 \rangle\}_{GR} \end{aligned}$$

and the pure substance

$$\{\langle g_6, 1 \rangle, \langle g_7, 1 \rangle\}_{PS}$$

**Example 5 (Hydroxylapatite)**

Hydroxylapatite,  $\text{Ca}_5(\text{PO}_4)_3\text{OH}$ , finally, consists of three groups:

$$\begin{aligned} g_8 &=_{\text{Df}} \{\langle \text{atom\_kind}(20), 1 \rangle\}_{GR} \\ g_9 &=_{\text{Df}} \{\langle \text{atom\_kind}(15), 1 \rangle, \langle \text{atom\_kind}(8), 4 \rangle\}_{GR} \\ g_{10} &=_{\text{Df}} \{\langle \text{atom\_kind}(8), 1 \rangle, \langle \text{atom\_kind}(1), 1 \rangle\}_{GR} \end{aligned}$$

so that the substance becomes

$$\{\langle g_8, 5 \rangle, \langle g_9, 3 \rangle, \langle g_{10}, 1 \rangle\}_{PS}$$

## 5.9 Names for complex concepts

In contrast to many other systems, complex concepts go unnamed in our ontology. This precludes the occurrence of inconsistencies caused by the presence of two different concepts corresponding to the same name. However, the expressions for complex concepts can become quite complicated. For humans, names facilitate the understanding of the expressions. We have in fact already done this when introducing the  $g_i$ 's in section 5.8.

Names must be regarded as abbreviations that fall outside the scope of the ontology. A consistent naming scheme must be employed, but it is not the business of the ontology to specify such a scheme. A naming scheme is inconsistent when a name corresponds to two or more different concepts. The converse (a concept that corresponds to two or more names) is not inconsistent but may be inconvenient.

In the Plinius context there is another link between names, or rather natural-language terms, and concepts. It is embodied in the lexicon, that is required (among other things) to translate natural-language constituents into ontology concepts. The mapping between natural-language constituents and ontology concepts is many-to-many. In natural language there are many ways to express the same concept. Thus, in the development corpus there are four different ways to indicate the pure substance hydroxylapatite: besides hydroxylapatite we have hydroxylapatite and ASCII-versions (which include typesetting codes) of the formulae  $\text{Ca}_5(\text{PO}_4)_3\text{OH}$  and  $\text{Ca}_{10}(\text{PO}_4)_6(\text{OH})_2$  (the latter actually incorrect according to IUPAC recommendations). The systematic IUPAC name, calcium hydroxyl phosphate, could also have been used but does not occur. Conversely, any of these terms is conceptually ambiguous because the pure substance or the material might be meant.

## 5.10 Primitive or defined?

It has been observed by Gruber [1991] that a good ontology consists of two mutually supplementary parts: a natural-language part for explanatory purposes and a formal part for unambiguous definitions. The two parts together define the concepts. The important difference is that automated reasoning is supported by the formal part only. (We assume that the formalisation does

not introduce additional axioms. We would regard such an introduction unwanted. If it is possible and desirable to formalise further, then it can and should be done in the ontology itself.)

It is therefore important to inquire how much of the definitions of concepts given above is formal. For the atomic concepts the answer is simple. Concepts of type natural number are only posited. Concepts of types sample and element are defined by means of the functions `sample` and `atom_kind` which are also only posited. In these cases, meaning is acquired through the natural-language account that anchors the definitions into domain knowledge. Formally, we just have names and automatic reasoning about their meanings is impossible.

For the complex concepts, the answer is less obvious. The construction rules define concepts such that they coincide with their definitions. All the information available to distinguish any concept from other concepts is stored in the expression for the concept itself. It has been repeatedly emphasised above that the expression involves partial rather than full information. This is particularly evident for constitutions, where even the fact that the constituents are connected to form a whole is left implicit. A similar observation applies to attribute combinations. An ion is fully defined by a group and a charge, but the ontology has no concept for charge and is *a fortiori* unable to infer anything about charges.

Although the information is partial, it is still sufficient to unambiguously identify the type of the concept. However, the types are *not* fully defined with respect to each other. The construction is such that automated procedures can always tell the type of any concept, but automated reasoning about the differences and relations between the concepts is not possible.

The conclusion is, that complex concepts are defined to the extent that they can always be assigned to the correct type automatically and are primitive in the sense that there is no way to automatically reason about their relations if they are of different types. One way to solve the problem partly is to define special relations to hold between complex concepts of different types, such as `constituent` and subset relations. We proceed to define those now.

## 5.11 Part-whole hierarchies

Constitutions (whether absolute or relative) implicitly define part-whole relations. Any constitution specifies which parts constitute the whole denoted by the constitution, since the constituents are listed as the first members of the tuples. This can be made explicit by means of a relation called `constituent(x, y)`, where  $x$  is a constituent of  $y$ . Constitutions can be nested, so that we have to allow for transitive closure of `constituent` relations. We forego the special definition for the ontology of pure substances and immediately introduce the following generalised definition:

### Generalised definition 4 (constituent relation)

The assertion `constituent(x, y)` expresses that  $x$  is a part or constituent of  $y$ . Its truth value is determined by:

$$\text{constituent}(x, y) \Leftrightarrow \exists z(\langle x, z \rangle \in y) \vee \exists w(\langle x, z \rangle \in w \wedge \text{constituent}(w, y))$$

provided that all tuples occur in sets that denote absolute or relative constitutions defined before.  $\square$

This is sufficient to be able to find out that, for instance, `atom_kind(8)` (the element oxygen) is a constituent of the concept for the phosphate group, which in turn is a constituent of the pure substance hydroxylapatite. As a consequence, `atom_kind(8)` is a constituent of the pure substance hydroxylapatite. (See example 5 in section 5.8).

Concepts of type ion are not written as constitutions but as constitutions, groups, with an extra attribute, the charge. These concepts need a special version of the constituent relation that disregards the charge:

$$\text{constituent\_of\_ion}(x, y) \Leftrightarrow y = \langle g, c \rangle \wedge \text{constituent}(x, g)$$

## 5.12 Taxonomies

### 5.12.1 The subconcept-relation

The important subconcept relation can be introduced by using a construct that will be called **arbitrary**. The intuitive idea is as follows. Given any particular concept  $c_1$  which is a constitution, we can construct a new concept  $c_2$  from  $c_1$  by leaving one or more constituents unspecified. It is intended that for the unspecified constituent any constituent can be filled in, provided it is of the same type. Then  $c_2$  is a superconcept of  $c_1$ . Type information can be retained by writing **arbitrary**( $X$ ), where  $X$  is the set of all concepts of the intended type.

We now proceed to put matters on a formal basis. We first introduce the relation of *subconcept* that may hold between two concepts  $c_1$  and  $c_2$ . (As before, concepts will be written in lower case, sets of concepts in uppercase.) The subconcept relation will be written as the infix operator  $\sqsubseteq$  familiar from the KL-ONE tradition. The expression

$$c_1 \sqsubseteq c_2$$

means that  $c_1$  is a subconcept of  $c_2$ .

We demand that the relation expressed by  $\sqsubseteq$  be reflexive, antisymmetric, and transitive. Thus, for each  $c_1$ ,  $c_2$ , and  $c_3$ ,

$$(c_1 \sqsubseteq c_2 \wedge c_2 \sqsubseteq c_1) \Leftrightarrow c_1 = c_2 \quad (5.1)$$

(from which it follows that  $\sqsubseteq$  is reflexive) and

$$(c_1 \sqsubseteq c_2 \wedge c_2 \sqsubseteq c_3) \Rightarrow c_1 \sqsubseteq c_3 \quad (5.2)$$

In other words,  $\sqsubseteq$  induces a partial ordering on the set of concepts.

We cannot follow the KL-ONE mainstream to define the semantics of the subconcept relation by means of subset relations between sets of concrete objects in the Universe of Discourse. As we discussed in section 3.4, we also allow concepts in our Universe of Discourse. Then still subset relations are too simple and more appropriate constructions will have to be used. We will outline our solution for concepts involving chemical constitution. Subconcept relations between such concepts can be established by means of assertions involving samples, as follows:

$$c_1 \sqsubseteq c_2 \Leftrightarrow \forall x(\text{consists}(\text{sample}(x), c_1) \Rightarrow \text{consists}(\text{sample}(x), c_2))$$

### 5.12.2 “Arbitrary”

The meaning of **arbitrary**-constructs can now be defined in terms of the  $\sqsubseteq$  relation:

$$c \sqsubseteq \text{arbitrary}(C) \Leftrightarrow c \in C \quad (5.3)$$

where  $c$  is a concept and  $C$  is a set of concepts.  $C$  needs to be known before **arbitrary**( $C$ ) can be used. This definition covers simple cases. The concept for chemical element in general is **arbitrary**( $E$ ), and the concept for pure substance in general is **arbitrary**( $PS$ ). Likewise, let, for instance, *Hal* stand for the set of halogens (*i.e.*, the chemical elements of column VII in the periodic table: F, Cl, Br, and I):

$$\text{Hal} =_{\text{Df}} \{\text{atom\_kind}(z) \mid z \in \{9, 17, 35, 53\}\} \quad (5.4)$$

Then **arbitrary**(*Hal*) is the concept for halogen in general because, for instance:

$$\text{atom\_kind}(9) \sqsubseteq \text{arbitrary}(\text{Hal})$$

but

$$\text{atom\_kind}(11) \not\sqsubseteq \text{arbitrary}(\text{Hal})$$

### 5.12.3 Subconcept relations between complex concepts

Complex concepts exist in two variants: they are constitutions (written as sets of tuples) or attribute combinations (written as tuples). For convenience, a separate notion of *subtuple*, symbol  $\sqsubseteq_t$ , is introduced first. The obvious definition requires the two tuples to be of equal length. Furthermore, the members of the tuples at corresponding positions have to be identical or the first has to be a subconcept of the second. Formally:

$$\langle c_1, c_2, \dots, c_n \rangle \sqsubseteq_t \langle d_1, d_2, \dots, d_n \rangle \Leftrightarrow c_1 \sqsubseteq d_1 \wedge c_2 \sqsubseteq d_2 \wedge \dots \wedge c_n \sqsubseteq d_n \quad (5.5)$$

Note that the two tuples involved in a subtuple relation do not have to be concepts defined in the ontology; only their members are required to be so. For instance, a tuple consisting of a chemical element and a natural number is not a concept. But the definition above supports assertions such as

$$\langle \text{atom\_kind}(11), 5 \rangle \sqsubseteq_t \langle \text{arbitrary}(E), 5 \rangle$$

and

$$\langle \text{atom\_kind}(11), 5 \rangle \sqsubseteq_t \langle \text{arbitrary}(E), \text{arbitrary}(\mathbb{N}) \rangle$$

For the subconcept relation between two attribute combinations, written as tuples  $t_1$  and  $t_2$ , we obtain:

$$t_1 \sqsubseteq t_2 \Leftrightarrow t_1 \sqsubseteq_t t_2 \quad (5.6)$$

For defining the subconcept relation between two constitutions (whether relative or absolute) written as sets of tuples  $s_1$  and  $s_2$ , we use a recursive construction. The base case is formed by singleton sets. Hence:

$$\begin{aligned} \{t_1\} \sqsubseteq \{t_2\} &\Leftrightarrow t_1 \sqsubseteq_t t_2 \\ \{t_1\} \cup s_1 \sqsubseteq \{t_2\} \cup s_2 &\Leftrightarrow t_1 \sqsubseteq_t t_2 \wedge s_1 \sqsubseteq s_2 \end{aligned} \quad (5.7)$$

The present machinery is sufficient to handle most subconcept relations of interest. To illustrate, we often want to create a superconcept to stand for a set of constitutions that all have the same constituent. This is common in chemistry, where for instance the expression “phosphates” refers to all pure substances that have the phosphate group as one of their constituents. Concepts for hydroxylapatite (example 5 in section 5.8) and aluminium phosphate  $\text{AlPO}_4$  are both subconcepts of the concept of a phosphate.

By rule 2 we are allowed to define *Pho*, the set of phosphates. In example 5, section 5.8,  $g_9$  was defined to stand for the phosphate group. Hence:

$$\text{Pho} =_{\text{Df}} \{p \mid p \in PS \wedge \text{constituent}(g_9, p)\}$$

As a result, we can immediately employ  $\text{arbitrary}(\text{Pho})$  to stand for the pure substance superconcept ‘phosphate’.

Because subconcept relations based on constitution are so common, it pays to introduce a special shortcut for them. Let  $\text{arb\_const}(X, y)$  stand for a concept such that any concept  $x$  is a subconcept if  $x \in X$  and  $\text{constituent}(y, x)$ :

$$x \sqsubseteq \text{arb\_const}(X, y) \Leftrightarrow x \in X \wedge \text{constituent}(y, x)$$

Then, to define a concept for phosphates, it is no longer necessary to separately introduce *Pho*. We can immediately write:

$$\text{arb\_const}(PS, g_9)$$

## 5.13 Configurations

### 5.13.1 Configuration information

We close the discussion of the ontology of pure substances with a summary of ways to include structure information. Constitutions convey partial information: they only tell which kinds of parts there are and how many. It is established by fiat that the parts in fact constitute a constitution. We have stipulated that, for example, the constituents of a group are connected to form a group. This is inconvenient if we want to distinguish between different ways to constitute a whole out of the same set of parts. For concepts of type group this is manifest by the inability to account for isomers. For the car parts example, we cannot distinguish between a car and a building kit to make that car, because both obviously consist of the same set of parts. In these cases, the wholes have an identical constitution but different *configurations*. We make the intended distinctions by specifying *configuration information*.

Configuration information is added to the constitution of the whole. This can be expressed by demanding that the concept for any whole is a tuple  $(cons, conf)$ , where *cons* is an absolute or relative constitution and *conf* specifies the configuration. The addition of configuration information will be relevant primarily for absolute constitutions, because in those cases the whole very often is a concept for a class of identifiable objects that are constituted of parts connected in a particular way. It can also be imagined, however, that we want to distinguish between relative constitutions on the basis of connections between parts.

The notion of configuration does not render the notion of constitution useless. It is useful to be able to distinguish between wholes with different configurations but identical constitution, on the one hand, and wholes with different constitutions, on the other. Also, the addition of configuration information is an investment that can be earned back only if the finer distinctions have to be made. This will not always be the case.

The primary way to specify configuration information is by means of *graphs*. We will first discuss the use of graphs for specifying configuration information of groups at some length. We will then turn to the general issue of adding configuration information to chemical concepts.

The aim is to give a preliminary assessment of the price that has to be paid to accommodate configuration rather than to present a definitive proposal.

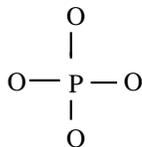
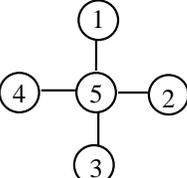
### 5.13.2 Chemical graphs

The occurrence of isomerism presupposes the presence of atomic bonds (section 4.6), so handling isomerism is needed at the level of groups. The primary way to specify configuration information is by means of a *graph*.

In computational chemistry, the use of graphs for specifying structure is common. Drawn structure formulae such as the ones depicted in figures 4.1 and 4.2 are naturally interpreted as graphs. The literature on chemical graphs is voluminous. Research has addressed issues at the conceptual, representational, and implementational levels. We have the impression that in AI there is a deplorable tendency to re-invent the idea and its applications. We first discuss previous work on chemical graphs and then present a quick summary of their use for specifying configuration information.

The idea of a chemical graph it is more than a century old. Harary [1972] credits the mathematician A. Cayley, who in 1857 used graphs to calculate the number of structural isomers that correspond to a given empirical formula. The name of the graph-theoretical concept of valency (that stands for the number of edges connected to a particular node) is inspired by this chemical application. Meyer [1991] credits the mathematician Mooers with the idea to use graphs for computer storage and manipulation of molecular structures and lists a large number of applications. Balaban [1995] provides a recent review with ample references.

The DENDRAL project (Lindsay et al. [1993]) has been immensely fruitful for chemical graph research. The work is seldom cited by AI authors, so we expand a little. Brown and co-workers have developed CONGEN, a suite of methods that for the first time fully solves the problem of

		<table border="1"> <thead> <tr> <th>Nodes</th> <th>Edges</th> </tr> </thead> <tbody> <tr> <td><math>\langle 1, O \rangle</math></td> <td><math>\{1, 5\}</math></td> </tr> <tr> <td><math>\langle 2, O \rangle</math></td> <td><math>\{2, 5\}</math></td> </tr> <tr> <td><math>\langle 3, O \rangle</math></td> <td><math>\{3, 5\}</math></td> </tr> <tr> <td><math>\langle 4, O \rangle</math></td> <td><math>\{4, 5\}</math></td> </tr> <tr> <td><math>\langle 5, P \rangle</math></td> <td></td> </tr> </tbody> </table>	Nodes	Edges	$\langle 1, O \rangle$	$\{1, 5\}$	$\langle 2, O \rangle$	$\{2, 5\}$	$\langle 3, O \rangle$	$\{3, 5\}$	$\langle 4, O \rangle$	$\{4, 5\}$	$\langle 5, P \rangle$	
Nodes	Edges													
$\langle 1, O \rangle$	$\{1, 5\}$													
$\langle 2, O \rangle$	$\{2, 5\}$													
$\langle 3, O \rangle$	$\{3, 5\}$													
$\langle 4, O \rangle$	$\{4, 5\}$													
$\langle 5, P \rangle$														

**Figure 5.1:** CONNECTION TABLE. Connection table or graph representation of molecular structure, using the phosphate ion as example. From left to right: (i) The structure as a chemist would write it, without, however, indicating the nature of the bond. (ii) The common way of drawing graphs to indicate the numbering of nodes. (iii) The sets of nodes and edges defining the graph written as a connection table.

exhaustively generating all structural isomers corresponding to a given empirical formula (Carhart et al. [1975]). CONGEN is based on earlier graph-theoretical work (Brown, Hjelmeland & Masinter [1974]; Brown & Masinter [1974]). Nourse and co-workers have enriched CONGEN with a method that exhaustively generates all stereoisomers corresponding to a given empirical formula (Nourse [1977]; Nourse et al. [1979]; Nourse et al. [1980]). A later version called GENOA (Carhart et al. [1981]) is more efficient by making use of overlapping and alternative substructures. It is again based on graphs. The great utility of graphs was also shown by using them to predict the arrangement of atoms in space (Fella, Nourse & Smith [1983]).

Other AI applications are described by Levinson [1984], Gelernter, Rose & Chen [1990], Napoli [1992], and Valdés-Pérez [1995], among others. Ellis [1993] describes a method for efficient storage and retrieval of objects, illustrated by chemical graphs. An interesting application outside AI is the derivation of topological characteristics of molecules on the basis of graphs. The characteristics prove to be remarkably effective in predicting certain properties of the substances (Rouvray [1990]).

Turning now to a summary of the subject, the standard version of a chemical graph is a tuple  $\langle \mathcal{A}, \mathcal{B} \rangle$  with  $\mathcal{A}$  a set of nodes that stand for atoms and  $\mathcal{B}$  a set of edges that stand for bonds. It is a typed, undirected, and possibly cyclic graph. To each node, a type specifying the chemical element is attached. The bonds are undirected. Bonds can be typed by specifying the nature of the bond (single, double, ...) but this is not always necessary. Chemists call the extensional specification of  $\mathcal{A}$  and  $\mathcal{B}$  a *connection table*, see figure 5.1 for an example.

Chemical graphs are determined up to the scheme used to label the nodes. Two graphs,  $\langle \mathcal{A}_1, \mathcal{B}_1 \rangle$  with labelling scheme  $\mathcal{L}_1$  and  $\langle \mathcal{A}_2, \mathcal{B}_2 \rangle$  with labelling scheme  $\mathcal{L}_2$ , are chemically identical (specify the same molecule) iff there exists a mapping  $\mathcal{L}_1 \mapsto \mathcal{L}_2$  and *vice versa* such that exchanging one set of labels for the other does not change the types of nodes connected by edges.

Chemical graphs as specifications of configuration information in concepts of type group can handle two of the three types of isomerism identified in section 4.6. A chemical graph with unlabeled bonds is sufficient for distinguishing structural isomers. The graph specifies all the information needed because the absolute constitution of the group can be derived in a straightforward way from the connection table or any other representation of the graph. Labeling the bonds by a measure of bond length would be sufficient to distinguish bond-stretch isomers.

The chemical graphs for stereoisomers are identical. Nourse, cited above, has demonstrated that it is possible to predict the number of stereoisomers that correspond to a particular graph, provided that bond lengths are known. One kind of stereoisomers (*cis-trans* isomers like the pair depicted in figure 4.2) can be distinguished by specifying the symmetry group as defined in group theory. Chiral pairs, presenting the other kind of stereoisomers, are mirror images of each other and therefore the symmetry group is the same. Mirror images can only be distinguished by their absolute configuration in space.

Chemical graphs open the way for specifying a constituent relation that differs from the **constituent** relation as defined by generalised definition 4 in section 5.11. The subgraph relation can be employed to define a **subconstituent** relation that holds between parts of molecules. This is amply discussed in the literature.

### 5.13.3 Other ways to specify configuration information

Graphs are not always attractive. In some cases a graph is over-specific and in others it is inappropriate for making the intended distinctions. For instance, a graph is inappropriate for handling stereoisomers.

A graph is over-specific if the costs of its inclusion are not compensated by gains. For example, defining a concept of type *group* as a chemical graph entails that we have to specify each and every group as a graph. This in turn means that we have to know the structure, which represents a large investment even if we disregard bond labels.<sup>2</sup> If only a few structural isomers are expected in the domain, it may be advantageous to seek a more modest way of specifying configuration information. Lack of canonicity may then become a problem. When graphs are used only to distinguish the isomers that occur in the domain under study, we run into trouble when the domain has to be enlarged.

Lack of canonicity also hinders the use of simpler schemes. Chemists often dispense with graphs by writing their formulae in particular ways: ethanol is written as  $\text{CH}_3\text{CH}_2\text{OH}$  and methoxymethane as  $\text{CH}_3\text{OCH}_3$  (see figure 4.1 for the graphs). There are no official rules underlying this practice. Figueras [1983] has investigated an algorithm to automatically convert formulae like  $\text{CH}_3\text{CH}_2\text{OH}$  and  $\text{CH}_3\text{OCH}_3$  into connection tables. If successful, the rules used by the program are suited to design a canonical format that is cheaper than graphs. Unfortunately, too many ambiguities were found.

To round off, configuration information is not needed for concepts of type pure substance. There obviously are different ways of piecing the constituents together, namely by metallic or ionic bonds. The inclusion of the kind of bonds is redundant in the sense, that concepts are unambiguously identified in the absence of this information: for pure substance, only constitution counts. Also, the kind of bonds can be decided by making use of background knowledge.

## 5.14 Extendability: isotopes

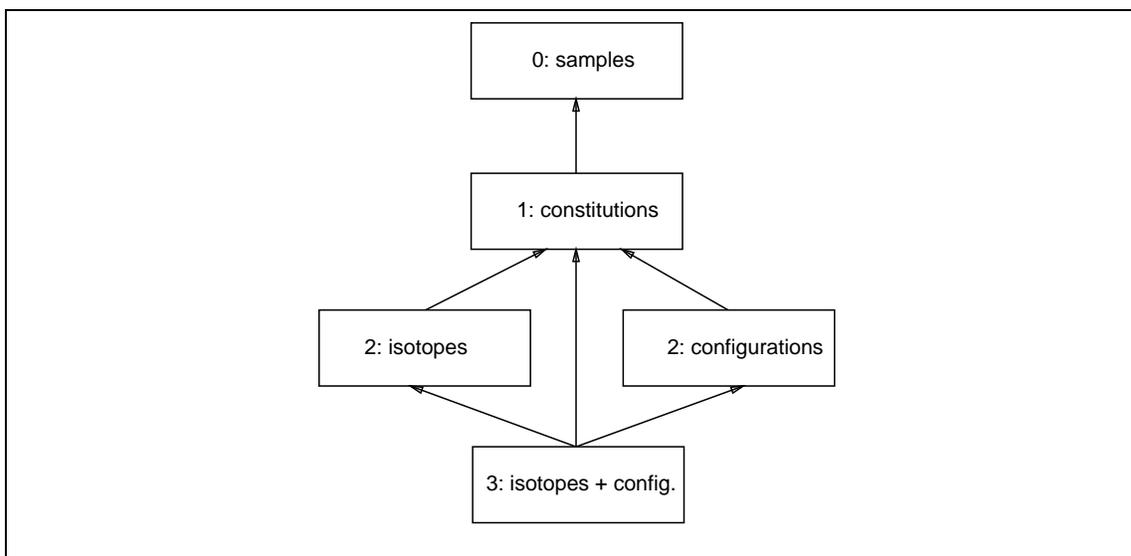
Among the criteria for assessing ontologies proposed by Gruber [1993b] is monotonic extendability. It should be possible to define new concepts (Gruber says ‘terms’) based on the existing vocabulary without having to revise existing definitions. In the *KL-ONE* and closely related *Ontolingua* approaches, the taxonomic hierarchy can always be extended by adding a subconcept to a particular concept. This does not alter any of the definitions already present. As was seen in section 5.12, something very similar can be done in our approach, even if it is bottom-up rather than top-down.

A much more difficult extension in any approach involves exchanging atomic concepts for other atomic concepts, because then at least some existing definitions will have to be revised. Our ontology is no exception, but we shall see that its layered design necessitates only minimal changes.

The example we will use here is concerned with isotopes. It will be recalled from section 4.2 that isotopes are variants of the same chemical element that differ in their mass number. As the mass number is the sum of the numbers of protons and neutrons in the atom’s nucleus and the number of protons (the atomic number) is fixed for any element, a difference in mass number is caused by a difference in the number of neutrons. The recommended symbol according to the Red Book writes the mass number as a superscript in front of the element symbol:  $^{17}\text{O}$ , also in formulae:  $\text{Ca}_5(\text{PO}_4)_3^{17}\text{OH}$ .

Extending the ontology to accommodate isotopes is easy. It takes a re-definition of the function `atom_kind` to take two arguments rather than one. In `atom_kind(Z, A)`, the first argument  $Z$  is as before the atomic number, while  $A \in \mathbb{N}$  is the mass number. For the second argument we could also have chosen the number of neutrons, but the present choice conforms more closely to chemical practice. The value of the function for any two arguments is a concept for an isotope. Let us call the set of concepts of type isotope  $I$ .

<sup>2</sup>This may change, however, if publicly accessible chemical databases with structure information proliferate.



**Figure 5.2: GRAPH OF ONTOLOGIES.** The figure illustrates the relations between ontologies at several levels as discussed in the main text. The levels are indicated by short terms. The arrows correspond to transformations that discard part of the information.

For chemical elements, the value of  $A$  does not matter. Therefore, the new construction allows concepts for elements to be written as `atom_kind( $Z$ , arbitrary( $\mathbb{N}$ ))`. The concepts for isotopes of any given element are then subconcepts of the concept for the element, which is the way chemists construe the relationship, too.

The only other change required is to define any concept of type `group` to be a set of tuples  $\langle i, n \rangle$  with  $i \in I$  and (as before)  $n \in \mathbb{N}$  subject to the chemical constraint. The definitions of concepts of types `ion` and `pure substance` remain the same, because these concepts do not ‘see’ elements or isotopes. They only ‘see’ groups, and those we have appropriately re-defined.

## 5.15 Graphs of ontologies

As has become apparent above, the present chapter does not specify a single ontology of pure substances: it defines several, at various levels of detail. At the top-level (section 5.3) we just have concepts for uniquely defined samples that can be said to consist of a pure substance.<sup>3</sup> For clarity we will call this level 0 (zero). We arrive at a more detailed level by choosing how to specify the substance, but the choice is entirely open: flat strings qualify just as well as more complex constructions.

A more detailed ontology, at what we call level 1, is specified in sections 5.4 through 5.7. It details pure substances by means of a bottom-up ontology and thus decides on a particular way to detail level 0. Isomerism and isotopy are ignored. In other words, pure substances are characterised by their constitution in terms of chemical elements only. Again, this leaves many options open for filling in yet further details.

At level 2 we have even more detailed ontologies. The present account specifies two possibilities; there certainly are far more. The first possibility is to include details about bonding in order to be able to distinguish structural isomers. This is done in section 5.13. The other possibility is to accommodate isotopy as specified in section 5.14. We may also combine the two possibilities (not discussed here but obvious), arriving at level that can be called level 3 in the present discussion. Of course, simple numbering schemes run into trouble in more involved examples.

One way to chart the relations between the different ontologies looks at whether we have to add

<sup>3</sup>At least, that is the choice made here. In the full Plinius ontology, samples are said to consist of a material.

or discard information in going from one to the other. A transformation is *strictly information-discarding* iff, in going from one to the other, we discard information but do not have to add information. At the knowledge level, strictly information-discarding transformations are interesting because they correspond to abstractions. An ontology that ignores isomerism and isotopy is more abstract than an ontology that takes account of these phenomena. At the implementation level they are interesting because the corresponding transformations can be done automatically. A *graph of ontologies* is obtained by having the nodes stand for ontologies and the directed edges for strictly information-discarding transformations. See figure 5.2.

Strictly information-discarding transformations between ontologies are supported by Ontolingua (Gruber [1993a]), which has the `include-theory` relation for this purpose. It is used to have the definitions laid down by the included ontology (Ontolingua says ‘theory’) taken for granted by the including ontology.

Whether a strictly information-discarding transformation results in a useful abstraction is undecided. The following example may make this clear. In an Ontolingua set-up, definitions for numbers are stored in a dedicated ontology to enable their use in any other ontology. An ontology  $O$  which includes the number ontology can be transformed into an ontology  $O'$  by discarding the appropriate `include-theory` relation. The transformation is obviously strictly information-discarding. However, in many cases the definitions in  $O$  rely so heavily on numbers that  $O'$  is useless. For instance, substitution of the primitive `number` for every occurrence of a number in the ontology of pure substances results in an ontology that is unable to make the appropriate distinctions. Some will even call the result incoherent.

## Chapter 6

# Bottom-up construction of ontologies

### 6.1 Atomism

An ontology built in a bottom-up fashion defines concepts as standing for objects called *wholes* that are composed of certain indivisible or smallest objects. The ontology defines wholes as constitutions of parts when structure can be disregarded, and as graphs with smallest units for nodes otherwise. In a bottom-up conceptual construction kit  $\langle \mathcal{A}, \mathcal{C} \rangle$  (section 2.3.4),  $\mathcal{A}$  is the set of concepts for the smallest units while the construction rules in  $\mathcal{C}$  are chosen such that they specify allowed ways to combine smallest units into larger wholes. This can be done recursively, so that several levels of composition are distinguished, but ultimately every whole is described as being constituted of smallest units.

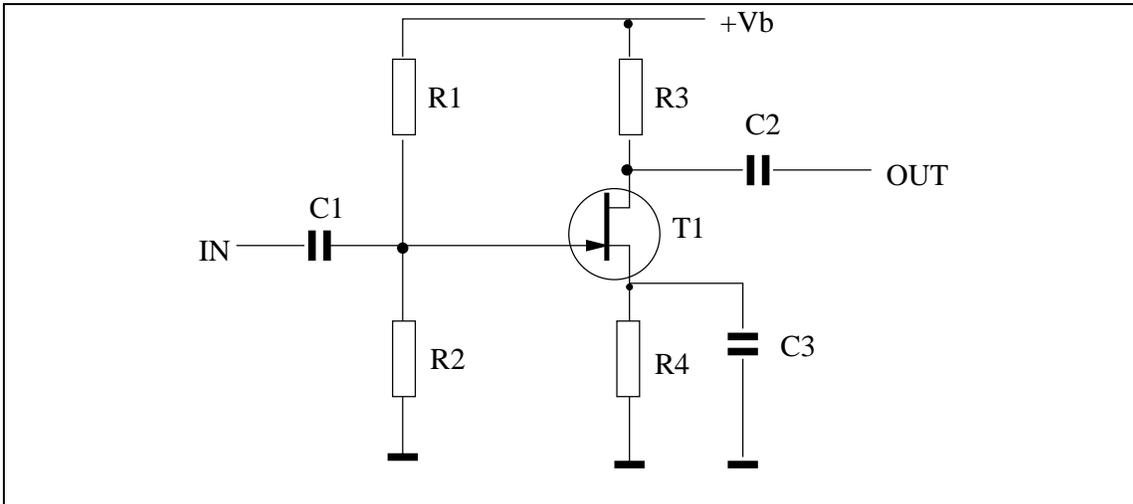
The background is provided by the tradition of *atomism*. According to atomism, the objects we perceive are composed of indivisible units called atoms. In principle, the specification of an object in terms of atoms and their interactions constitutes the fullest possible description of the object and allows derivation of all its other properties. Atomism thus has a *descriptive* aspect and a more ambitious *explanatory* aspect. We will be concerned with the descriptive aspect only.

Chemistry and particle physics are the paradigm cases. A complicated picture involving several levels of composition has been developed. According to the current picture, the smallest constituents of matter are quarks and leptons. Quarks in turn are the constituents of particles like protons and neutrons. The term ‘atoms’ in natural science no longer refers to the smallest units but to complex objects that consist of protons, neutrons, and electrons (a kind of leptons). Atoms are the constituents of molecules. The concepts at these levels can all be defined as absolute constitutions of quarks and leptons. We can go further up, but the idea is sufficiently clear. A bottom-up approach to ontology development is obvious.

The bottom-up approach is not confined to particle physics and chemistry. A domain (or part of it) lends itself to the development of a bottom-up ontology if it involves objects that can be characterised as being composed out of a finite number of smallest objects. There may be levels of composition between smallest objects and the object we want to characterise, and composed and smallest objects can be abstract or concrete.

The purpose of this chapter is to outline further applications. We will present four examples. It is not intended to claim that the bottom-up approach is sensible or even feasible for every application. We will return to this point in our concluding remarks (chapter 8).

The first two examples address wholes that can be described in terms of constitutions without or with configuration information. As will be seen, configuration information is indispensable particularly in the second example. The other two examples involve attribute combinations. The list is certainly incomplete; for instance, certain biological applications seem promising. For instance, Fogle [1990] has discussed an ontology of genetics that is built around the concept of



**Figure 6.1:** ELECTRONIC CIRCUIT. The scheme shows a simple  $n$ -channel FET amplifier with source bias. A possible choice of parts is given in figure 6.2.

T1	=	2N5246	{(transistor(2N5246), 1)
R1	=	20 M $\Omega$	(resistor( $20 \times 10^6$ ), 1),
R2	=	1 M $\Omega$	(resistor( $10^6$ ), 1),
R3	=	7 k $\Omega$	(resistor( $7 \times 10^3$ ), 1),
R4	=	3 k $\Omega$	(resistor( $3 \times 10^3$ ), 1),
C1 = C2	=	0.01 $\mu$ F	(capacitor( $10^{-8}$ ), 2),
C3	=	10 $\mu$ F	(capacitor( $10^{-5}$ ), 1)}

**Figure 6.2:** Parts list for the electronic circuit depicted in figure 6.1, left as an electronic engineer would write it and right as an absolute constitution. We also need concepts for input and output, and for power supply and ground. The parts can be connected together in so many ways that configuration has to be added to obtain sufficiently unambiguous concepts.

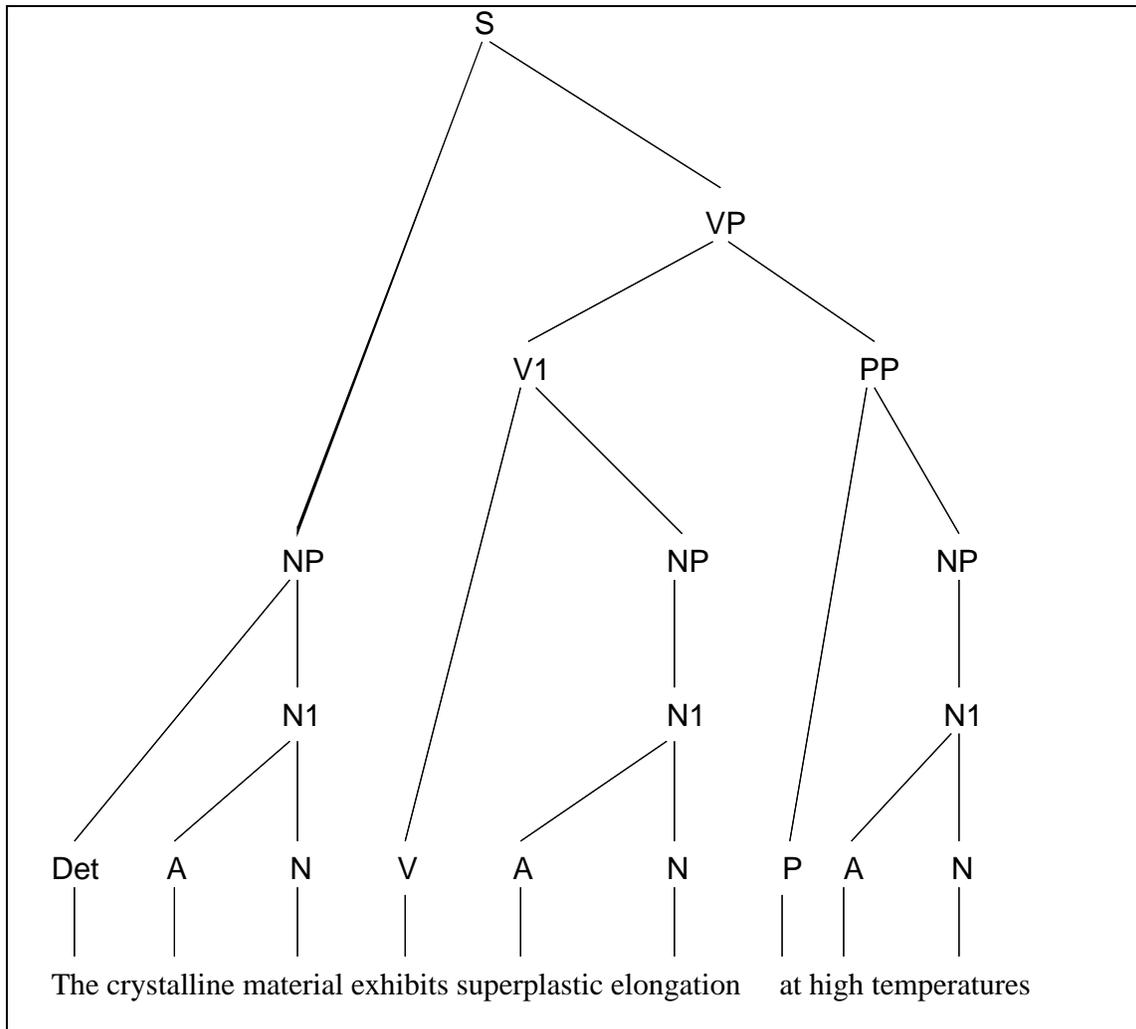
domain set for active description (DSAT). DSATs are absolute constitutions.

## 6.2 Further examples

### 6.2.1 Engineered artifacts

Engineered artifacts like cars, electronic circuits, and coffee machines can be described as constituted in a particular way out of smallest units. The choice of smallest units depends on the task at hand. For production scheduling, it appears sensible to identify parts delivered by suppliers as smallest units. Levels intermediate between finished product and smallest parts may be chosen such that they correspond to semifinished products like wheels and engine blocks.

For defining concepts we have the choice of using constitutions only or adding configuration information. The choice obviously depends on the application. It has been said above that, lacking configuration information, we cannot distinguish between a car and a building kit to make that car. If we are certain that we will only consider fully assembled cars, the need to make the distinction disappears. Another situation where constitutions will do is that addressed by component catalogs (compare, for instance, Adams & Dahl [1994] and Bradley, Agogino & Wood [1994]). A component manufacturer may want to include part-whole information in the catalog. Constitutions are sufficient if every assembly is uniquely characterised by its set of components in the context of the catalog.



**Figure 6.3:** PARSE TREE OF A SIMPLE SENTENCE, showing the syntactic composition. The sentence is a simplified version of the second sentence of the abstract quoted in figure 2.1. The context-free grammar rules needed to describe the sentence are specified in figure 6.4.

In other cases, the possible variation in configuration that we want to consider is such that structure is indispensable. A graph is again the primary way to specify configuration information. Consider a simple electronic circuit as depicted in figure 6.1 and its parts list as specified in figure 6.2. Even for describing this circuit we need configuration information because there are so many ways to assemble the parts into a whole that absolute constitution is ambiguous. Furthermore, the presence of input and output entails that if this circuit is part of a larger whole, it is best described as a subgraph. The simple **constituent** relation is inappropriate.

### 6.2.2 Natural-language sentences

A natural-language sentence is composed of smallest units that, depending on the required level of detail, can be identified as words (tokens), word parts, single characters (including spaces, periods, and the like), or even sounds as in speech recognition. Syntactic analysis identifies the way these constituents are related to form a sentence. Linguists distinguish levels intermediate between the sentence and its smallest constituents, as manifest by categories like noun phrase and prepositional phrase.

For an illustration, consider the case where the smallest units are word forms (tokens) and

N1	→	A N
NP	→	N1
NP	→	Det N1
PP	→	P NP
V1	→	V NP
VP	→	V1 PP
S	→	NP VP

**Figure 6.4:** CONTEXT-FREE GRAMMAR RULES. Listed is the complete set of context-free grammar rules needed to describe the structure depicted in figure 6.3. Wholes are at the left-hand sides, their constituents at the right-hand sides. Constituents are specified in the order in which they have to occur. The first level of analysis (working bottom-up) is that of pre-terminals. Pre-terminals correspond to part-of-speech categories and include Det (determiner), A (adjective), N (noun), P (preposition), and V (verb). N1 is a whole intermediate between an NP (noun phrase) and its constituents; V1 is intermediate between a VP (verb phrase) and its constituents. We further have a PP (prepositional phrase) and an S (sentence).

the largest wholes are sentences. The result of syntactic analysis as depicted in figure 6.3 is a structure that consists of nested parts. A typical lexicon of tokens maps each token onto a part-of-speech category like adjective or noun. This corresponds to a set of atomic concepts divided over subsets. Grammar rules (figure 6.4) specify how wholes may be composed of parts and thus correspond with construction rules. Order is important, so configuration information cannot be missed. However, ordering is always linear.

The example of natural-language sentences is obvious but not very informative because we have only re-formulated the notion of a context-free grammar.

### 6.2.3 Processes

Processes can very often be specified as a nested sequence of subprocesses. For instance, hot isostatic pressing is one way to make an object of hydroxylapatite. The process starts with powder preparation, followed by forming the powder into the desired shape (this is called a greenling). The greenling is subjected to a high temperature and a high pressure to obtain the finished product. Each of these subprocesses can be further subdivided. The first step, for instance, may involve repeated cycles of purification, milling, and sieving to obtain a sufficient degree of powder homogeneity.

An interesting option is to model processes at each level as attribute combinations  $\langle m_1, m_2 \rangle$ , where  $m_1$  is a concept for the starting material and  $m_2$  a concept for the product. We have done this for the Plinius ontology. The ordering of processes is implicitly specified because process  $p_1$  can only follow process  $p_2$  if the starting material of  $p_1$  is a product of  $p_2$ .

### 6.2.4 Diseases

So far, we have discussed examples where the constitutions of wholes in terms of parts were rather obvious. The bottom-up approach however also accommodates abstract combinations. An example is provided by an ontology prepared earlier as part of our Sapiens project (Mars, Speel & van der Vet [1991]; Speel, Mars & van der Vet [1991]). Concepts for diseases are defined as attribute combinations  $\langle g, f, o, e, d \rangle$ . The first four members of the tuple are all taken from predefined sets of concepts, respectively for gross anatomy ( $g$ ), functional anatomy ( $f$ ), organism affected ( $o$ ) and cause (etiology,  $e$ ). Not all four have to be specified. Given the tacit agreement that the tuple specifies partial information defining a whole, concepts for almost all known diseases and many unknown ones are unambiguously defined. For example, it is easy to introduce a concept for bacterial carditis, a disease of the heart ( $g$ ) in humans ( $o$ ) that is caused by bacterial infection ( $e$ ).

---

The attribute combination  $\langle g, f, o, e \rangle$  specifies partial information which is not guaranteed to be sufficient. Therefore we added a fifth member  $d$ , called the disambiguator, to the tuple. It is a label to distinguish diseases that cannot be told apart by means of the first four members, given the sets of concepts from which they have to be taken. In a sense,  $d$  supplies configuration information because it distinguishes two different ways in which the constituents (the first four members of the tuples) can be related.

## Chapter 7

# Formalisation and implementation

### 7.1 The choice of representation language

To pave the way for implementations using the concepts of the ontology, it is sensible to provide at least one formalisation in a language for automated reasoning. This does not entail that in the finished system there will be something like a separate ontology module. It is unclear what function such a module would have. One potentially useful function could be a type checker for manually coded input; but it is better to define an interface that makes it impossible to code incorrect input. For the present account, the function of a formalisation in an executable language is to give a flavour of what represented knowledge in terms of ontology concepts might be.

One of the design principles discussed before (section 2.3.3) is the independence of any particular representation language. Previous versions of the ontology presented in chapter 5 have been formalised and often also implemented in a large variety of languages and systems, in particular those of the KL-ONE family, see Speel [1995]. Yet, as we also discussed above, the independence of any *particular* knowledge representation language does not imply that every language is equally convenient. This can be illustrated now. The design of our ontology can be made explicit in the form of a logic to help choose a representation language.

The idea underlying the ontology of pure substances is to express knowledge as simple expressions of the form

`predicate(argument1, argument2, ...)`

where `predicate` and the `argumenti` are concepts defined by the ontology.

The `predicate` expresses some relation between concepts. All predicates are strongly typed in their arguments, where the types are those defined by the ontology. In the simple example of pure substances, we have encountered three such predicates:

`consists(c1, c2)` for chemical constitution specification (definition 5 in section 5.3). Concept `c1` is required to be of type `sample` and concept `c2` to be of type `pure substance`.

`constituent(c1, c2)` for the relation between a part and the whole of which it is a constituent (generalised definition 4 in section 5.11). Type constraints on `c1` and `c2` follow from the definition of the `constituent` predicate: both `c1` and `c2` are required to be an atomic concept for a part or a constitution (relative or absolute).

`c1 ⊆ c2` for the subconcept relation defined in section 5.12. `c1` and `c2` are required to be of the same type.

Note that type constraints ensure well-formedness rather than correctness or truth. An assertion to the effect that a particular group is a constituent of a chemical element is well-formed but incorrect.

The logic we sketch here is first order: we have so far not found need for quantifying over predicates. The arguments, if complex concepts, are expressions involving sets and tuples, often nested several levels deep. Whether we have to allow for quantification over constituents of arguments is an open question for the moment. There are inferences that take recourse to constituents of complex concepts. For instance, the evaluation of the assertions `constituent(c1, c2)` and `c1 ⊆ c2` proceeds by inspection of the structure of complex concepts (provided they are well-formed).

This logic deviates from more traditional approaches in AI, such as exemplified by the KL-ONE approach. There, instances of concepts rather than concepts themselves are arguments to predicates. In such approaches, concepts correspond to one-place predicates that, when given  $x$  as argument, assert that  $x$  is an instance of the concept in question. From a syntactical point of view, however, there is nothing wrong with our approach.

The sketch aids in deciding about the knowledge representation language. In the course of preparing the KL-ONE versions, a distinct friction was felt. In retrospect the friction was mainly due to the quite different design principles underlying the approaches. In the ontology of pure substances, the bottom-up approach leads to two orthogonal hierarchies (part-whole and subconcept) that are both defined implicitly. In KL-ONE, by contrast, the subconcept hierarchy is the main structuring device and it has to be specified explicitly by the designer. In the course of preparing an Ontolingua formalisation of an earlier version of the Plinius ontology (see van der Vet, Speel & Mars [1994] for details), we found that Ontolingua is also heavily dependent on a top-down successive differentiation approach. We needed a lot of pure KIF code to achieve formalisation, thus decreasing portability. In addition, it proved impossible to provide Ontolingua counterparts for generalised definitions, as Ontolingua insists on treating them as higher-order constructs.

The logic underlying the ontology of pure substances is a traditional first-order predicate logic enriched to cope with complex arguments. Prolog can already handle complex arguments and thus is suggested as a suitable representation language. This is, in fact, the choice made here. Other advantages of Prolog are: it is a widely used language that is, moreover, rapidly becoming standardised (compare Sterling & Shapiro [1994]); and the program can be run to verify certain claims.

As an alternative to Prolog we may consider a typed feature logic (Carpenter [1992]) with a corresponding implementation in ALE (Carpenter & Penn [1994]). We in fact plan to do so in the near future. There are undoubtedly other possibilities.

## 7.2 The Prolog formalisation

### 7.2.1 Set-up

A Prolog program consists of assertions in a subset of first-order predicate logic, the Horn clause subset. Given the setup of the present ontology, it is straightforward to translate `predicate` into Prolog predicates and ontology concepts into Prolog terms (arguments to those predicates). Types as defined by the ontology definitions can be translated into unary Prolog predicates that take an arbitrary term as argument and evaluate to true if and only if the argument is of the intended type. Such predicates are called *type predicates* by the Prolog community.

The full set of type predicates for an ontology constitutes a complete formalisation in Prolog.<sup>1</sup> However, as we noted above, the need for a set of type predicates in a program is not evident. We have instead designed the Prolog formalisation as an executable demonstration of the claim that every well-formed complex concept can be assigned the correct type automatically. This has been done by writing the formalisation as a set of clauses:

```
type(Type, Concept) :- Body.
```

where `Body` collects the conditions under which `Concept` is of type `Type`. In handling the query

<sup>1</sup>In fact, the same approach is in principle possible in Common Lisp. There the idea is to declare new types by means of `declare` and `the forms`, see Steele [1990], chapters 4 and 9. Appropriate functions return true if they are given an argument of the intended type. The difference between an assertion that is true or false, on the one hand, and a function that returns a Boolean value, on the other, is not that big.

```
type(X, Concept).
```

with `Concept` some specified concept, Prolog unifies `Type` with the correct type and returns the result.

The full Prolog formalisation of the ontology presented in the present report is given in appendix B. Here, a number of accompanying notes are assembled.

### 7.2.2 Atomic concepts

The formalisation of atomic concepts is entirely straightforward. For numbers we have made use of the standard predicate `integer/1`. The functions `sample(n)` and `atom_kind(Z)` are written as functors.

### 7.2.3 Complex concepts

In the ontology, complex concepts are constructed as tuples and sets of tuples. An  $n$ -tuple is obviously formalised as a list of length  $n$ .

Many Prolog versions (among them the Quintus Prolog we use) come with a library of set predicates that can be used to formalise the sets of the ontology. However, the set library is not standard and set predicates may be implemented in different ways in the various Prologs that offer them. This has an adverse effect on portability. Therefore, it has been decided to formalise sets as lists of arbitrary but finite length and explicitly add the few predicates needed for set-theoretical operations.

It is useful to be able to distinguish between lists used as tuples and lists used as sets. We have therefore made use of the functors `set` and `tuple`, both of arity one, which take the list as argument. The Prolog term that corresponds to the ontology concept for hydroxylapatite thus becomes (compare example 5 in section 5.8):

```
set([tuple([set([tuple([atom_kind(20), 1]]), 5]),
          tuple([set([tuple([atom_kind(15), 1]),
                        tuple([atom_kind(8), 4]]), 3]),
                tuple([set([tuple([atom_kind(8), 1]),
                              tuple([atom_kind(1), 1]]), 1]])]), 1]]).
```

The type predicates for complex concepts are recursive because the sets they inspect do not have a fixed length. We take care of the constraint that rules out empty sets by choosing the base case to be a set of cardinality 1 rather than the empty set.

Another point is the so-called chemical constraint. It demands that the first member of any tuple does not occur in any other tuple in the set. The problem obviously does not arise for sets of cardinality 1. For the inspection of longer lists we introduce an auxiliary predicate `chem_correct/1` that takes the whole set as argument and evaluates to true if its argument satisfies the chemical constraint.

### 7.2.4 Other features

The formalisation includes the `constituent` and subset relations that define the hierarchies.

We included alternative versions of the `type` predicates for concepts of type group, ion, and pure substance based on constitutions and attribute combinations. For pure substances this is somewhat awkward because the concepts are relative constitutions for which we have to choose a scaling constant. Rational numbers that add up to one are obviously preferred, but this is at odds with the chemical custom of writing proportions in pure substance formulae as natural numbers. We have conformed to the chemical custom, because the choice of scaling constant is not the point to be demonstrated.

Since graphs are defined in terms of sets and tuples, the machinery for handling graphs is already available. A connection table is represented as a tuple `tuple([Atoms, Bonds])`, where `Atoms` is the set of nodes and `Bonds` the set of edges.

`Atoms` is a set of tuples, where each tuple has the syntax

```
tuple([Atom_id , atom_kind(N)])
```

`Atom_id` is the unique label of the node in the graph and `atom_kind(N)` identifies the chemical element.

`Bonds` is a set of sets, where each set has the syntax

```
set([Atom_id1, Atom_id2])
```

`Atom_id1` and `Atom_id2` are the labels of the nodes connected by (chemical) bonds.

According to this syntax, the representation of the connection table for the phosphate group becomes (compare also figure 5.1):

```
tuple([set([tuple([1, atom_kind(8)]),
             tuple([2, atom_kind(8)]),
             tuple([3, atom_kind(8)]),
             tuple([4, atom_kind(8)]),
             tuple([5, atom_kind(15)])]),
       set([set([1, 5]),
            set([2, 5]),
            set([3, 5]),
            set([4, 5])])])
```

Just to give a flavour of connection tables in Prolog, we have added a predicate `cons_corr_conf/1` that evaluates to true if the constitution corresponds to the configuration (hence the name). From a procedural point of view it can be used to construct a constitution from a connection table, demonstrating the point that the constitution is redundant if a connection table is available.

### 7.2.5 Toward implementation

In a real implementation we have to consider efficiency. We have included a few efficiency enhancing features, but much is left undone. For instance, the present formalisation does not include any cuts although there are many places where a ‘green cut’ will help.<sup>2</sup> Also, attention can be paid to the ordering of predicates and of clauses in the bodies of various rules.

Another measure does not primarily aim to enhance computational efficiency but to enhance legibility. The notation that only uses the functors `set` and `tuple` may be formally correct, it leads to uninterpretable code. One solution is to explicitly define abbreviations in a systematic way, keeping track of them in a separate database of `abbreviation/2` predicates, like in

```
abbreviation(hydroxylapatite,
             set([tuple([set([tuple([atom_kind(20), 1])]), 5]),
                  tuple([set([tuple([atom_kind(15), 1]),
                                tuple([atom_kind(8), 4])]), 3]),
                  tuple([set([tuple([atom_kind(8), 1]),
                                tuple([atom_kind(1), 1])]), 1])])).
```

We did so when testing the Prolog code.

It might alternatively be considered to introduce functors like `pure_substance` to convey intended meaning. Against this, the clarity of the `constituent/2` and `subset/2` predicates is lost because they will have to be multiplied to obtain versions for the different functors.

<sup>2</sup>A ‘green cut’ is defined by Sterling & Shapiro [1994] (chapter 11) as a cut whose addition does not change the meaning of the program. Adding a ‘red cut’ does change the meaning; ‘red cuts’ are to be avoided.

## Chapter 8

# Concluding remarks

Chapters 5 and 6 give examples in which the bottom-up approach leads to useful ontologies. We do not claim, however, that the approach is sensible or even feasible in other situations. We are not aware of a criterion that would enable us to assess at a relatively early stage whether a bottom-up approach is a good idea. One reason is lack of practical experience in many domains. Another reason is that the ease with which the approach can be followed and the quality of its result can be influenced along many dimensions, among them the choice of smallest parts and the decision to include configuration information or not. We can illustrate this by some further examples.

A bottom-up approach can become unwieldy or over-specific even when it can be applied in principle. Consider, for example, the water glass on our secretary's desk. Since any object is composed of molecules, a bottom-up approach can be used to model the glass and its contents such that molecules are the smallest parts. That would be very awkward if we only want to reason about the shape of the glass. Even then we might attempt to construct a bottom-up ontology of glass shapes in terms of elementary geometrical constituents. That is certainly more work than a taxonomic approach which successively differentiates according to whether the glass is a circle when seen from above or not, whether it has a foot, and so on. But less work is not a generally valid argument for preferring one approach over another, of course, because excess work may be earned back later. In short, without further knowledge about the nature of the system of which the knowledge base is to be a part, no approach is better than any other.

An ontology of materials is obviously important for Plinius. It is straightforward to specify materials down to the atomic-molecular level because that is the way materials scientists do it. The current ontology of materials for Plinius is wholly couched in terms of absolute and relative constitutions. An addition to include configuration information is only foreseen for groups (as, in fact, outlined in section 5.13 above).

In preparing other ontologies for Plinius we have not chosen a bottom-up approach. Consider an ontology of cracks. Cracks are flaws in ceramic products that may propagate through the entire product, causing it to break. It is not obvious how a crack can be fruitfully described as being constituted of a finite number of smallest units. The stress here is on 'fruitfully' because we know what the parts are. At the level typically addressed by the Plinius source texts, a crack consists of two surfaces and one or two tips. The current ontology of cracks, prepared by Wilco G. ter Stal, even contains the full list of those parts. A bottom-up approach may model the parts as atomic concepts and define cracks as constitutions of those parts. For laying down the meaning of the parts we have to rely on a natural-language account to anchor their meaning into consensus domain knowledge. However, quite apart from the fact that these concepts are far more ambiguous than the atomic concepts we use now (as compared, for instance, with those for chemical elements), it would smuggle the concept of crack into the definition through the informal account. A tip obviously is a tip of something. There is another way to model cracks in a bottom-up fashion, namely by going down to the level of atoms and bonds. That is simply too much work, if only because no further use is made of that information.

To sum up: there is limited practical experience with ontologies, both in the literature and

in our own work. It would therefore be premature to search for a rule able to tell us beforehand whether a bottom-up ontology is possible and useful. What we can do for now is to enlarge our experience through further research.

# Appendix A

## List of chemical elements

The table of elements given below lists all chemical elements as they are currently known and given official names by the IUPAC. This includes all elements up to and including atomic number 109. We have sorted the elements according to symbol rather than atomic number, because the symbol is more widely known.

Included are: the official symbol, atomic number  $Z$ , atomic weight  $A_r$  scaled to  $A_r(^{12}\text{C}) = 12$ , and official IUPAC name in English. The data have been taken from the most recent IUPAC recommendations, those for 1993 (published in December 1994, IUPAC [1994b]). Instead of giving the most precise value for atomic weight, we have included the data of the table entitled “Standard atomic weights abridged to five significant figures”, as they are less subject to change and sufficiently precise for the purpose at hand. We have, however, excluded data that refer to a single isotope of an element. The values are considered reliable to  $\pm 1$  in the last significant figure, unless a larger value is given in parentheses. Even these five-figure values are only reliable for samples that have not been subjected to any process that alters isotopic composition. The names and symbols of the elements 101 through 109 have been taken from the IUPAC recommendation 1994 (IUPAC [1994a]); they are, however, disputed (Seaborg [1994]).

Radioactive elements occurring in nature are marked \*. Synthetic elements are marked †; they are all radioactive.

Symbol	$Z$	$A_r(\text{E})$	Name
Ac	89	.	actinium*
Ag	47	107.87	silver
Al	13	26.982	aluminium
Am	95	.	americium†
Ar	18	39.948	argon
As	33	74.922	arsenic
At	85	.	astatine*
Au	79	196.97	gold
B	5	10.811(5)	boron
Ba	56	137.33	barium
Be	4	9.0122	beryllium
Bh	107	.	bohrium†
Bi	83	208.98	bismuth
Bk	97	.	berkelium†
Br	35	79.904	bromine
C	6	12.011	carbon

(continued ...)

Symbol	Z	$A_r(E)$	Name
Ca	20	40.078(4)	calcium
Cd	48	112.41	cadmium
Ce	58	140.12	cerium
Cf	98	.	californium†
Cl	17	35.453	chlorine
Cm	96	.	curium†
Co	27	58.933	cobalt
Cr	24	51.996	chromium
Cs	55	132.91	caesium (cesium)
Cu	29	63.546(3)	copper
Db	104	.	dubnium†
Dy	66	162.50(3)	dysprosium
Er	68	167.26(3)	erbium
Es	99	.	einsteinium†
Eu	63	151.96	europium
F	9	18.998	fluorine
Fe	26	55.845(2)	iron
Fm	100	.	fermium†
Fr	87	.	francium*
Ga	31	69.723	gallium
Gd	64	157.25(3)	gadolinium
Ge	32	72.61(2)	germanium
H	1	1.0079	hydrogen
He	2	4.0026	helium
Hf	72	178.49(2)	hafnium
Hg	80	200.59(2)	mercury
Hn	108	.	hahnium†
Ho	67	164.93	holmium
I	53	126.90	iodine
In	49	114.82	indium
Ir	77	192.22	iridium
Jl	105	.	joliotium†
K	19	39.098	potassium (kalium)
Kr	36	83.80	krypton
La	57	138.91	lanthanum
Li	3	6.941(2)	lithium
Lr	103	.	lawrencium†
Lu	71	174.97	lutetium
Md	101	.	mendelevium†
Mg	12	24.305	magnesium
Mn	25	54.938	manganese
Mo	42	95.94	molybdenum
Mt	109	.	meitnerium†
N	7	14.007	nitrogen
Na	11	22.990	sodium (natrium)
Nb	41	92.906	niobium

(continued ...)

Symbol	Z	$A_r(E)$	Name
Nd	60	144.24(3)	neodymium
Ne	10	20.180	neon
Ni	28	58.693	nickel
No	102	.	nobelium†
Np	93	.	neptunium†
O	8	15.999	oxygen
Os	76	190.23(3)	osmium
P	15	30.974	phosphorus
Pa	91	.	protactinium*
Pb	82	207.2	lead
Pd	46	106.42	palladium
Pm	61	.	promethium†
Po	84	.	polonium*
Pr	59	140.91	praseodymium
Pt	78	195.08(3)	platinum
Pu	94	.	plutonium†
Ra	88	.	radium*
Rb	37	85.468	rubidium
Re	75	186.21	rhenium
Rf	106	.	rutherfordium†
Rh	45	102.91	rhodium
Rn	86	.	radon*
Ru	44	101.07(2)	ruthenium
S	16	32.066(6)	sulfur
Sb	51	121.76	antimony (stibium)
Sc	21	44.956	scandium
Se	34	78.96(3)	selenium
Si	14	28.086	silicon
Sm	62	150.36(3)	samarium
Sn	50	118.71	tin
Sr	38	87.62	strontium
Ta	73	180.95	tantalum
Tb	65	158.93	terbium
Tc	43	.	technetium†
Te	52	127.60(3)	tellurium
Th	90	232.04	thorium*
Ti	22	47.867	titanium
Tl	81	204.38	thallium
Tm	69	168.93	thulium
U	92	238.03	uranium*
V	23	50.942	vanadium
W	74	183.84	tungsten (wolfram)
Xe	54	131.29(2)	xenon
Y	39	88.906	yttrium
Yb	70	173.04(3)	ytterbium
Zn	30	65.39(2)	zinc
Zr	40	91.224(2)	zirconium

(complete)

# Appendix B

## Prolog implementation

### B.1 Introduction

This appendix lists the set of Prolog formalisations of the ontology of pure substances. The ontology itself is specified in chapter 5 while the decisions underlying the present formalisation are discussed in chapter 7. We rehearse here that the formalisation defines representations of ontology concepts in an indirect way. It essentially is a procedure consisting of `type/2` predicates, one for each concept. We will drop the specification of arity and write “`type predicate`” from now on. We will also be somewhat sloppy and write, for instance, “samples” when we mean “concepts of type sample”.

For legibility we have rendered comments in normal typeface and not in standard Prolog comment lines preceded by `%`.

The programs listed here are exercises in pure logic programming. Efficiency has not been considered. (To be sure: the code listed here *can* be run on any Prolog.) In the comments, we have added indications about efficiency enhancement as would be necessary in any real program. This also illustrates our views about the difference between representation and implementation.

The programs listed here are copyrighted by the Knowledge-Based Systems Group, University of Twente, the Netherlands, 1995. They were written by Paul E. van der Vet.

We will list the following programs:

`atomic.pl` Predicates for atomic concepts.

`complex.pl` Predicates for complex concepts, simple version (that is, not in the form of constitutions or attribute combinations).

`constitution.pl` Predicates for complex concepts written as specialisations of (absolute or relative) constitutions.

`constituent.pl` The Prolog version of the `constituent` relation.

`subconcept.pl` Predicates for the subconcept relation.

`graph.pl` A predicate for graphs.

`auxiliary.pl` Auxiliary predicates needed in the programs listed above.

We have written the code using the conventions for standard Prolog as given by Sterling & Shapiro [1994]. This is not the form in which the programs were tested, because our Quintus Prolog deviates from the conventions. In particular, the standard predicate `not` is written as `\+` in Quintus. The predicate for non-identity is given as `≠` by Sterling & Shapiro [1994] but as `=\=` in Quintus.

## B.2 Program `atomic.pl`

First follow predicates for numbers. A `type` predicate as formalisation of definition 1 (concepts of type whole number) is omitted for two reasons. First, whole numbers correspond with the standard Prolog predicate `integer/1`. Second, whole numbers do not occur in definitions of complex concepts. A `type` predicate as formalisation of definition 4 (concepts of type strictly positive rational number) is omitted because in the present formalisation we use natural rather than rational numbers to specify proportions in relative configurations.

```
type(natural_number, N) :-
    integer(N), N > 0.
```

```
type(whole_number_uneq_zero, Z) :-
    integer(Z), Z /= 0.
```

The `type` predicates for the representations of samples and chemical elements are equally obvious. In a running system, new representations of samples can be introduced by keeping track of the highest number assigned so far and raising it by one for each new concept. In the `type` predicate for the representations of chemical elements, the number `N` stands for an atomic number.

```
type(sample, sample(N)) :-
    type(natural_number, N).
```

```
type(chemical_element, atom_kind(N)) :-
    type(natural_number, N).
```

## B.3 Program `complex.pl`

The pattern is set by the `type` predicate for the representations of groups:

```
type(group, set([ tuple([E, N]) ] ) ) :-
    type(chemical_element, E),
    type(natural_number, N).
type(group, set([ tuple([E, N]) | Tuples ] ) ) :-
    type(chemical_element, E),
    type(natural_number, N),
    type(group, set(Tuples)),
    chem_correct(set([ tuple([E, N]) | Tuples ])).
```

The second clause is recursive because groups can be expressed as sets of arbitrary (but finite) cardinality. The base case is handled in the first clause. It is a set that consists of a single tuple (thus empty sets are ruled out), where the first member has to be a representation of a chemical element and the second a representation of a natural number.

Returning to the second clause, a set of cardinality  $> 1$  is a representation of a group if the first tuple fulfills the requirements imposed by the first clause, and the other tuples form a set that is a representation of a group. In sets of cardinality  $> 1$  we also have to impose the chemical constraint (compare the discussion in section 5.5). This is done by the auxiliary predicate `chem_correct/1`. `chem_correct(Set)`, where `Set` is a set of ordered pairs, is true if the first member of every pair does not occur in any other pair.

Incorporation of `chem_correct` in the recursive clause is computationally inefficient. Prolog tries to satisfy the `chem_correct` goal for each recursive call while the check has to be performed only once. An efficient implementation would first recursively check correctness of tuples; if in order, the whole expression is subsequently checked for the chemical constraint.

The structure of the `type` predicate for the representation of concepts of type pure substance is similar to that of the `type` predicate corresponding to groups. So we deviate from the order of exposition of chapter 5:

```

type(pure_substance, set([ tuple([GR, N]) ]) ) :-
    type(group, GR),
    type(natural_number, N).
type(pure_substance, set([ tuple([GR, N]) | Tuples ]) ) :-
    type(group, GR),
    type(natural_number, N),
    type(pure_substance, set(Tuples)),
    chem_correct(set([ tuple([GR, N]) | Tuples ])),
    range(Range, set([ tuple([GR, N]) | Tuples ])),
    gcd_of_list(1, Range).

```

Compared with the recursive clause for the representation of groups, the recursive clause here has two extra goals to accommodate the simplest proportions demand (compare the discussion in section 5.7). `range/2` and `gcd_of_list/2` are auxiliary predicates. `range(Range, Set)`, where `Set` is a set of ordered pairs, is true if `Range` is the list of all second members of the pairs. `gcd_of_list(N, List)`, where `List` is a list of integers, is true if `N` is the greatest common divisor of all integers in `List`.

The efficiency remarks concerning the execution of the goal `chem_correct/1` in recursive clauses can be repeated here. The goals `range/2` and `gcd_of_list/2` also need to be executed only once.

We close with the type predicate for the representations of ions.

```

type(ion, tuple([GR, Z])) :-
    type(group, GR),
    type(whole_number_uneq_zero, Z).

```

## B.4 Program constitution.pl

The definitions of absolute and relative constitutions (generalised definitions 1 and 3, respectively) are quite similar to those for groups and pure substances, except that the constituents are left unspecified. The concepts are straightforwardly represented as Prolog functors of arity one, where the argument specifies the type of constituents. The argument is a variable.

In relative constitutions we may also choose the scaling constant. Here, we adopt the scaling used for pure substances, so that the proportions are natural numbers subject to the constraint that their greatest common divisor is one.

We obtain:

```

type(absolute_constitution(X), set([ tuple([Y, N]) ]) ) :-
    type(X, Y),
    type(natural_number, N).
type(absolute_constitution(X), set([ tuple([Y, N]) | Tuples ]) ) :-
    type(X, Y),
    type(natural_number, N),
    type(absolute_constitution(X), set(Tuples)),
    chem_correct(set([ tuple([Y, N]) | Tuples ])).

type(relative_constitution(X), set([ tuple([Y, N]) ]) ) :-
    type(X, Y),
    type(natural_number, N).
type(relative_constitution(X), set([ tuple([GR, N]) | Tuples ]) ) :-
    type(X, Y),
    type(natural_number, N),
    type(relative_constitution(X), set(Tuples)),
    chem_correct(set([ tuple([Y, N]) | Tuples ])),

```

```

range(set([ tuple([Y, N]) | Tuples ]), Range),
gcd_of_list(Range, 1).

```

The remarks about computational efficiency can be repeated.

The alternative type predicates for the representations of groups and pure substances trivially become:

```

type(group, X) :-
    type(absolute_composition(chemical_element), X).
type(pure_substance, X) :-
    type(relative_composition(group), X).

```

An equally satisfying formalisation of generalised definition 2 (attribute combination) is not possible in a first-order framework because the tuples are allowed to have arbitrary (but finite) length. We omit a second-order definition here, although Prolog supports second-order constructions to some extent. (They are, however, seldom used because they are computationally inefficient.)

A practical first-order workaround is to employ `attr_comb` functors of variable arity and specify a type predicate for each arity actually used. Prolog regards functors of different arity as different even if their labels are identical. For arity two and three we would then obtain:

```

type(attr_comb(X, Y), tuple([P, Q])) :-
    type(X, P), type(Y, Q).
type(attr_comb(X, Y, Z), tuple([P, Q, R])) :-
    type(X, P), type(Y, Q), type(Z, R).

```

Then the alternative type predicate for the representations of ions becomes:

```

type(ion, X) :-
    type(attr_comb(group, whole_number_uneq_zero), X).

```

## B.5 Program constituent.pl

The formalisation of the `constituent` relation defined by generalised definition 4 is, in fact, the definition itself written in Prolog syntax:

```

constituent(Part, Whole) :-
    domain(Domain, Whole),
    listmember(Part, Domain).
constituent(Part, Whole) :-
    domain(Domain, Whole),
    listmember(Subwhole, Domain),
    constituent(Part, Subwhole).

```

The predicates `domain/2` and `listmember/2` are auxiliary predicates. `domain(Domain, Set)`, where `Set` is a set of ordered pairs, is true if `Domain` is the list of all first members of the pairs. `listmember(X, List)` is true if `X` occurs in `List`.

Type restrictions on the arguments of the `constituent/2` predicate are not imposed. In other words, the representation does not distinguish between ill-formed and false expressions. If ill-formed cases have to be recognised, a separate recognition predicate has to be evaluated before the `constituent/2` predicate itself is evaluated.

As explained in section 5.11, determining the constituent relation for ions has to proceed in a different manner:

```

constituent_of_ion(Part, tuple([GR, X])) :-
    constituent(Part, GR).

```

## B.6 Program `subconcept.pl`

The definitions given in section 5.12 are first order and can be translated into Prolog syntax with relative ease. The construct `arbitrary(X)` is written as a functor of arity one, with a type (rather than a set) as argument. The simple cases are easy:

```
subconcept(X, X).
subconcept(X, arbitrary(Type)) :-
    type(Type, X).
```

In comparing two complex concepts expressed as sets of tuples, we have to be careful. In Prolog, sets are represented as lists and the corresponding members cannot be expected to occur in the correct order. Therefore we first inspect the sets for corresponding members, one in each. If this succeeds, the sets obtained by removing the inspected members from the originals are recursively checked.

```
subconcept(set([X]), set([Y])) :-
    subtuple(X, Y).
subconcept(set([X | Xs]), set(Ys)) :-
    setmember(Y, Ys),
    subtuple(X, Y),
    one_member_less(set(Ys), set(Ys), Y),
    subconcept(set(Xs), set(Ys)).
```

The `subtuple/2` predicate is defined below.

`one_member_less/3` is an auxiliary predicate. `one_member_less(Set1, Set2, X)` is true if

$$\text{Set1} = \text{Set2} - \{X\}$$

where “ $-$ ” stands for asymmetric set difference.

Subconcept relations between concepts expressed as tuples are simple.

```
subconcept(tuple(X), tuple(Y)) :-
    subtuple(tuple(X), tuple(Y)).
```

The formalisation of the `subtuple` relation is:

```
subtuple(X, X).
subtuple(tuple([X]), tuple([Y])) :-
    subconcept(X, Y).
subtuple(tuple([X | Xs]), tuple([Y | Ys])) :-
    subconcept(X, Y),
    subtuple(tuple(Xs), tuple(Ys)).
```

Finally, subconcept relations based on shared constitution can be represented by adding another clause to the `subconcept` procedure:

```
subconcept(X, arb_const(Type, Part)) :-
    type(Type, X),
    constituent(Part, X).
```

Establishing a subconcept relation may become computationally expensive because the program will have to work its way through the layered expressions. If the surrounding program often calls `subconcept` irrespective of the types of the concepts, it makes sense to check for type equality first. Then it is more efficient to check for type *inequality*. For instance, if one concept is represented as a `set(...)` and the other as a `tuple(...)`, inequality follows immediately.

## B.7 Program graph.pl

This program consists of a procedure for a single predicate that determines whether a constitution is the absolute constitution that corresponds to a particular graph (connection table). Its general syntax is

```
cons_corr_conf(AbsoluteConstitution, Graph)
```

The procedure is:

```
cons_corr_conf(set([tuple([AtomKind, 1])]),
               tuple([set([tuple([Label, AtomKind])]), Bonds])).
cons_corr_conf(set([tuple([AtomKind, M]) | Tuples]),
               tuple([set([tuple([Label, AtomKind]) | Atoms]), Bonds])) :-
    M is N+1,
    cons_corr_conf(set([tuple([AtomKind, N]) | Tuples]),
                  tuple([set(Atoms) | Bonds])).
cons_corr_conf(set([tuple([AtomKind, 1]) | Tuples]),
               tuple([set([tuple([Label, AtomKind]) | Atoms]), Bonds])) :-
    cons_corr_conf(set(Tuples), tuple([set(Atoms), Bonds])).
```

## B.8 Program auxiliary.pl

We close by listing the auxiliary predicates needed to establish the meaning of the programs listed thus far. The explanations of the predicates have been given above.

```
listmember(X, [X | Xs]).
listmember(X, [_ | Xs]) :-
    listmember(X, Xs).
```

For computational purposes it is far more efficient to write the second clause as:

```
listmember(X, [Y | Xs]) :-
    X /= Y,
    listmember(X, Xs).
```

Predicates for handling sets follow.

```
domain([], set([])).
domain([X | Xs], set([tuple([X, Y]) | Tuples])) :-
    domain(Xs, set(Tuples)).
```

```
range([], set([])).
range([Y | Ys], set([tuple([X, Y]) | Tuples])) :-
    range(Ys, set(Tuples)).
```

```
one_member_less(set([]), set([]), X).
one_member_less(set(Xs), set([X | Xs]), X).
one_member_less(set([Y | YYs]), set([Y | Ys]), X) :-
    X /= Y,
    one_member_less(set(YYs), set(Ys), X).
```

The non-identity check in the last clause is essential to fix the meaning of the program.

The predicate `chem_correct/1` is defined in terms of another and self-explanatory predicate called `no_doubles/1`.

```
chem_correct(Concept) :-
    domain(Domain, Concept),
    no_doubles(Domain).

no_doubles([X]).
no_doubles([X | Xs]) :-
    not listmember(X, Xs),
    no_doubles(Xs).
```

We close with the predicate `gcd_of_list/2` that defines the greatest common divisor (hence: GCD) of all integers in a list. It makes use of a predicate `gcd/3` that defines the GCD of two numbers. `gcd/3` is based on the well-known Euclidean algorithm:

```
gcd(N, 0, N).
gcd(Gcd, N, M) :-
    M > 0,
    R is N mod M,
    gcd(Gcd, M, R).
```

`gcd_of_list/2` is based on the observation, that the GCD of all integers in a list is the GCD of another list obtained by taking out two arbitrary numbers from the original list and put their GCD in their place.

```
gcd_of_list(N, [N]).
gcd_of_list(Gcd, [N, M]) :-
    gcd(Gcd, N, M).
gcd_of_list(Gcd, [N, M | Ns]) :-
    gcd(IntGcd, N, M),
    gcd_of_list(Gcd, [IntGcd | Ns]).
```

Efficiency can be enhanced by adding a clause to handle cases where the number 1 occurs in the list, because then trivially the GCD of the whole list is 1.

# Appendix C

## Bibliography

- Jim H. Adams & Mark A. Dahl [1994], "Using knowledge-based systems to define materials technology in the aircraft design/build process," in *Knowledge-based applications in materials science and engineering*, James K. McDowell & Kenneth J. Meltsner, eds., The Minerals, Metals & Materials Society, Warrendale PA, 67–74.
- Alexandru T. Balaban [1995], "Chemical graphs: looking back and glimpsing ahead," *Journal of Chemical Information and Computer Sciences* 35, 339–350.
- Ronald J. Brachman [1977], "What's in a concept: structural foundations for semantic networks," *International Journal of Man-Machine Studies* 9, 127–152.
- Ronald J. Brachman [1985], "'I lied about the trees', or defaults and definitions in knowledge representation," *AI Magazine* 6, 80–93.
- Ronald J. Brachman & Hector J. Levesque [1982], "Competence in knowledge representation," in *Proceedings National Conference on Artificial Intelligence, Pittsburgh, PE, 18–20 August 1982*, American Association for Artificial Intelligence, 189–192.
- Ronald J. Brachman & James G. Schmolze [1985], "An overview of the KL-ONE knowledge representation system," *Cognitive Science* 9, 171–216.
- S.R. Bradley, A.M. Agogino & W.R. Wood [1994], "Intelligent engineering component catalogs," in *Artificial Intelligence in Design '94*, J.S. Gero & F. Sudweeks, eds., Kluwer Academic, Dordrecht, 641–658.
- Harold Brown, Larry Hjelmeland & Larry Masinter [1974], "Constructive graph labeling using double cosets," *Discrete Mathematics* 7, 1–30.
- Harold Brown & Larry Masinter [1974], "An algorithm for the construction of the graphs of organic molecules," *Discrete Mathematics* 8, 227–244.
- Mario Bunge [1977], *Treatise on basic philosophy. Volume 3. Ontology I: The furniture of the world*, D. Reidel, Dordrecht, the Netherlands.
- Raymond E. Carhart, Dennis H. Smith, Harold Brown & Carl Djerassi [1975], "An approach to computer-assisted elucidation of molecular structure," *Journal of the American Chemical Society* 97, 5755–5762.
- Raymond E. Carhart, Dennis H. Smith, Neil A.B. Gray, James G. Nourse & Carl Djerassi [1981], "GENOA: a computer program for structure elucidation utilizing overlapping and alternative substructures," *Journal of Organic Chemistry* 46, 1708–1718.
- Bob Carpenter [1992], *The logic of typed feature structures*, Cambridge University Press, Cambridge.
- Bob Carpenter & Gerald Penn [1994], "ALE: the Attribute Logic Engine, Version 2.0. User's guide," Computational Linguistics Program, Philosophy Department, Carnegie Mellon University, Pittsburgh PA.
- Nancy Cartwright [1983], *How the laws of physics lie*, Clarendon, Oxford.
- J.P. Coutures & M.H. Rand [1989], "Melting temperatures of refractory oxides: Part II. Lanthanoid sesquioxides (IUPAC recommendations)," *Pure and Applied Chemistry* 61, 1461–1482.
- R.W. Davidge [1979], *Mechanical behaviour of ceramics*, Cambridge University Press, Cambridge.

- Arthur A. Eggert, Anthony T. Jacob & Catherine H. Middlecamp [1992], "Converting chemical formulas to names: an expert strategy," *Journal of Chemical Information and Computer Sciences* 32, 227–233.
- Arthur A. Eggert, Anthony T. Jacob & Catherine H. Middlecamp [1993], "Converting chemical formulas to names: a second expert problem," *Journal of Chemical Information and Computer Sciences* 33, 458–465.
- Gerard Ellis [1993], "Efficient retrieval from hierarchies of objects using lattice operations," in *Conceptual Graphs for Knowledge Representation. Proceedings of the First International Conference on Conceptual Structures ICCS'93*, Guy W. Mineau, Bernard Moulin & John F. Sowa, eds., Springer, Berlin, 274–293.
- Andrea L. Fella, James G. Nourse & Dennis H. Smith [1983], "Conformation specification of chemical structures in computer programs," *Journal of Chemical Information and Computer Sciences* 23, 43–47.
- J. Figueras [1983], "Chemical symbol string parser," *Journal of Chemical Information and Computer Sciences* 23, 48–52.
- Thomas Fogle [1990], "Are genes units of inheritance?," *Biology and Philosophy* 5, 349–371.
- L.T.F. Gamut [1991], *Logic, language, and meaning. Volume 2. Intensional logic and logical grammar*, University of Chicago Press, Chicago.
- Herbert Gelernter, J. Royce Rose & Chyouhwa Chen [1990], "Building and refining a knowledge base for synthetic organic chemistry via the methodology of inductive and deductive machine learning," *Journal of Chemical Information and Computer Sciences* 30, 492–504.
- Michael R. Genesereth & Nils J. Nilsson [1987], *Logical foundations of artificial intelligence*, Morgan Kaufmann, Palo Alto CA.
- Thomas R. Gruber [1991], "The role of common ontology in achieving sharable, reusable knowledge bases," in *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, Cambridge, MA, April 22–25, 1991, James Allen, Richard Fikes & Erik Sandewall, eds., 601–602.
- Thomas R. Gruber [1993a], "A translation approach to portable ontology specifications," *Knowledge Acquisition* 5, 199–220.
- Thomas R. Gruber [1993b], "Toward principles for the design of ontologies used for knowledge sharing," Knowledge Systems Laboratory, Stanford University, Technical Report KSL 93–04, Palo Alto, CA, revised version, dated August 23, 1993, of a paper originally published in the Proceedings of the International Workshop on Formal Ontology in Conceptual Analysis and Knowledge Representation, Padova, Italy, March 17–19, 1993.
- Nicola Guarino & Pierdaniele Giaretta [1995], "Ontologies and knowledge bases: towards a terminological clarification," in *Towards very large knowledge bases. Knowledge Building and Knowledge Sharing 1995*, Nicolaas J.I. Mars, ed., IOS Press, Amsterdam, 25–32.
- Frank Harary [1972], *Graph theory*, Addison-Wesley, Reading, MA.
- Patrick J. Hayes [1985], "Naive physics I: ontology for liquids," in *Formal theories of the commonsense world*, Jerry R. Hobbs & Robert C. Moore, eds., Ablex, Norwood NJ, 71–107.
- Otthein Herzog & Claus-Rainer Rollinger [1991], *Text understanding in LILOG: integrating computational linguistics and artificial intelligence. Final report on the IBM Germany LILOG-Project*, Springer, Berlin.
- J. Hlaváč [1982], "Melting temperatures of refractory oxides: Part I (IUPAC recommendations)," *Pure and Applied Chemistry* 54, 681–688.
- Betsy L. Humphreys, Donald A.B. Lindberg & William T. Hole [1991], "Assessing and enhancing the value of the UMLS Knowledge Sources," in *Proceedings of the Fifteenth Annual Symposium on Computer Applications in Medical Care, a conference of the American Medical Informatics Association, Washington DC, November 17–20, 1991*, Paul D. Clayton, ed., McGraw-Hill, New York, 194–198.
- IUPAC [1994a], "Names and symbols of transfermium elements (IUPAC Recommendations 1994)," *Pure and Applied Chemistry* 66, 2419–2421, recommendations by the IUPAC Committee on Nomenclature of Inorganic Chemistry.

- IUPAC [1994b], "Atomic weights of the elements 1993," *Pure and Applied Chemistry* 66, 2423–2444, recommendations by the IUPAC Committee on Atomic Weights and Isotopic Abundances.
- Harald Krautscheid, Dieter Fenske, Gerhard Baum & Marcus Semmelmann [1993], "A new copper selenide cluster with PPh<sub>3</sub> ligands: [Cu<sub>146</sub>Se<sub>73</sub>(PPh<sub>3</sub>)<sub>30</sub>]," *Angewandte Chemie, International Edition* 32, 1303–1305.
- G.J. Leigh, ed. [1990], *Nomenclature of inorganic chemistry. IUPAC recommendations 1990*, Blackwell Scientific Publications, Oxford.
- Douglas B. Lenat & R.V. Guha [1990], *Building large knowledge-based systems*, Addison Wesley, Reading, MA.
- Robert Levinson [1984], "A self-organizing retrieval system for graphs," in *Proceedings National Conference on Artificial Intelligence, Austin, TX, 6–10 August 1984*, William Kaufmann, Los Altos, CA, 203–206.
- Robert K. Lindsay, Bruce B. Buchanan, Edward A. Feigenbaum & Joshua Lederberg [1993], "DENDRAL: a case study of the first expert system for scientific hypothesis formation," *Artificial Intelligence* 61, 209–261.
- K.M. Mackay & R.A. Mackay [1981], *Introduction to modern inorganic chemistry*, International Textbook Company, London.
- Nicolaas J.I. Mars [1994], "The role of ontologies in structuring large knowledge bases," in *Knowledge Building and Knowledge Sharing*, Kazuhiro Fuchi & Toshio Yokoi, eds., Ohmsha, Tokyo, 240–248.
- Nicolaas J.I. Mars & A.T. Schreiber [1985], "Direct access to knowledge in bibliographic databases," in *Proceedings of the ARTINT Workshop on Artificial Intelligence and Information Retrieval, Luxembourg, 13 September 1985*, Commission of the European Communities, Luxembourg, 83–86.
- Nicolaas J.I. Mars, Piet-Hein Speel & Paul E. van der Vet [1991], "Eindrapportage Sapiens project, UT-gedeelte," Universiteit Twente, UT-KBS-90-02, Enschede, the Netherlands.
- Nicolaas J.I. Mars, Wilco G. ter Stal, Hidde de Jong, Paul E. van der Vet & Piet-Hein Speel [1994], "Semi-automatic knowledge acquisition in Plinius: an engineering approach," in *Proceedings of the Eighth Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, January 30 – February 4, 1994*, Brain Gaines & Mark Musen, eds., 4–1 – 4-15.
- Nicolaas J.I. Mars & Paul E. van der Vet [1990], "A semi-automatically generated knowledge base for direct answers to user questions," in *TKE'90: Terminology and Knowledge Engineering*, Hans Czap & Wolfgang Nedobity, eds., Indeks Verlag, Frankfurt a.M., 352–362.
- Ernst Meyer [1991], "Computer representation and handling of structures: retrospect and prospects," *Journal of Chemical Information and Computer Sciences* 31, 68–75.
- Jacob Joan Mulckhuysen [1960], "Molecules and models: investigations on the axiomatization of structure theory in chemistry," University of Amsterdam, Amsterdam, Ph.D. Thesis.
- Amedeo Napoli [1992], "Subsumption and classification-based reasoning in object-based representations," in *Proceedings Tenth European Conference on Artificial Intelligence, 3–7 August 1992, Vienna, Austria*, Bernd Neumann, ed., John Wiley, Chichester, UK, 425–429.
- Robert Neches, Richard Fikes, Tim Finin, Thomas Gruber, Ramesh Patil, Ted Senator & William R. Swartout [1991], "Enabling technology for knowledge sharing," *AI Magazine* 12, 36–56.
- Allen Newell [1982], "The knowledge level," *Artificial Intelligence* 18, 87–127.
- James G. Nourse [1977], "Generalised stereoisomerization modes," *Journal of the American Chemical Society* 99, 2063–2069.
- James G. Nourse, Raymond E. Carhart, Dennis H. Smith & Carl Djerassi [1979], "Exhaustive generation of stereoisomers for structure elucidation," *Journal of the American Chemical Society* 101, 1216–1223.
- James G. Nourse, Dennis H. Smith, Raymond E. Carhart & Carl Djerassi [1980], "Computer-assisted elucidation of molecular structure with stereochemistry," *Journal of the American Chemical Society* 102, 6289–6295.
- Gerard Parkin [1992], "Do bond-stretch isomers really exist?," *Accounts of Chemical Research* 25, 455–460.

- Dennis H. Rouvray, ed. [1990], *Computational chemical graph theory*, Nova Science Publishers, New York.
- Glenn T. Seaborg [1994], "Terminology of the transuranium elements," *Terminology* 1, 229–252.
- Piet-Hein Speel [1995], *Selecting knowledge representation systems*, University of Twente, Enschede, the Netherlands, Ph.D. thesis.
- Piet-Hein Speel, Nicolaas J.I. Mars & Paul E. van der Vet [1991], "A knowledge-based approach to semi-automatic indexing," in *Proceedings of the Workshop on Language & Information Processing, October 27, 1991, Washington, DC, held at the 54th ASIS Annual Meeting*, Alexa T. McCray, ed., 49–58.
- Guy L. Steele [1990], *Common Lisp: the language*, Digital Press, Bedford MA, second edition.
- Leon Sterling & Ehud Shapiro [1994], *The art of Prolog. Advanced programming techniques*, MIT Press, Cambridge MA, second edition.
- Raúl E. Valdés-Pérez [1995], "Machine discovery in chemistry: new results," *Artificial Intelligence* 74, 191–201.
- Paul E. van der Vet [1987], *The aborted takeover of chemistry by physics. A study of the relations between chemistry and physics in the present century*, University of Amsterdam, Amsterdam, Ph.D. Thesis.
- Paul E. van der Vet, Hidde de Jong, Nicolaas J.I. Mars, Piet-Hein Speel & Wilco G. ter Stal [1994], "Plinius intermediate report," University of Twente, Memoranda Informatica 94-35, Enschede, the Netherlands.
- Paul E. van der Vet & Nicolaas J.I. Mars [1993], "Structured system of concepts for storing, retrieving, and manipulating chemical information," *Journal of Chemical Information and Computer Sciences* 33, 564–568.
- Paul E. van der Vet & Nicolaas J.I. Mars [1994], "Concept systems as an aid for sharing and reuse of knowledge bases in materials science," in *Knowledge-based applications in materials science and engineering*, James K. McDowell & Kenneth J. Meltner, eds., The Minerals, Metals & Materials Society, Warrendale PA, 43–55.
- Paul E. van der Vet, Piet-Hein Speel & Nicolaas J.I. Mars [1994], "The Plinius ontology of ceramic materials," in *Comparison of implemented ontologies. Working notes of the ECAT'94 workshop, August 9, 1994, Amsterdam*, Nicolaas J.I. Mars, ed., 187–205.