Formalizing Anaesthesia: a case study in formal specification

Rix Groenboom[1], Erik Saaman[1], Ernest Rotterdam[2], and Gerard Renardel de Lavalette[1]

[1] Research Institute for Mathematics and Computing Science, University of Groningen, P.O. Box 800, NL 9700 AV Groningen, the Netherlands. E-mail: {rix,erik,grl}@cs.rug.nl

[2] University Hospital Groningen, Department of Medical Information Sciences, P.O. Box 30001, NL 9700 RB Groningen, the Netherlands. E-mail: ernest@fwi.uva.nl

Abstract. We report on the formalization of knowledge for a support system in the field of anaesthesiology. It is a case study in the use of the formal specification method we are developing. The method consists of guidelines (using concepts from object-oriented design methods), language (AFSL, Almost Formal Specification Language) and tools (type-checker, graphical representation of signatures).

Keywords: Case study, Development process, Linking formal and informal methods, Medical systems, Object-orientation.

1 Introduction

This paper reports on the project FAN (Formalization of ANaesthesia) which aims at formal specification of the domain knowledge that is needed to construct support systems for anaesthesiology. The specification is written in the formal specification language AFSL [26, 27], with modularization, parameterization, and sub-typing. The support systems to be based on FAN include diagnosis and monitoring systems.

This case study is part of a larger research effort concerning the use of formal specification languages (the Formal System Analysis-project). The FSA-project uses several cases to develop a method (consisting of guidelines, a language and tools) that is suited to formalize so-called open (i.e. ill-structured) knowledge domains. We will not compare several methods and languages, since we have only applied one method for our formalization.

1.1 Overview of paper

First, in section 2, we introduce our view on formalizing informal knowledge domains, in comparison with related work in this area. In section 3, we outline the problem: support of anaesthesia. We continue in section 4 by briefly explaining the language and method that are used for the FAN-project. Then we present the formal specification in section 5. Section 6 mentions the trajectory of the FAN-project including some of the design decisions. In the final section we draw some conclusions.

2 Formalizing the informal

2.1 Open versus closed knowledge domains

The FSA-project focuses on the formalization of *open* knowledge domains. Some characteristics of open knowledge domains are: ill-structured, unstable, and informal. Examples of open knowledge domains abound in e.g. cognitive science, linguistics, and medicine.

In contrast, *closed* problems are concerned with well documented, understood, and implemented problems. Examples are communication protocols and digital hardware. The literature on formal methods is often geared to closed problems: the proceedings of FME '93 [32] and FME '94 [19] contain hardly any case studies in open problems. Research is reported in railway applications, controllers and protocols. All these are fairly technical applications, involving systems that already have been implemented (and thus formalized in one way or another). For validation and proving correctness of implementations, the formalization of these domains is obligatory (cf. hardware and protocol verification).

In our opinion, the use of formal specification in ill-structured domains is an important field of research. The application of formal methods can be effective and help in the better understanding of the knowledge domain.

2.2 Related work

Knowledge-based systems An area that deals with formalization of open knowledge domains is the development of knowledge-based systems. Several methods for developing knowledge-based systems are available, examples are the KADS model of expertise [31] and the compositional method DESIRE [16].

The KADS-method gives guidelines for dividing problems into layers, and gives standard problem solving techniques for several kinds of problems. A drawback of this method is that it only prescribes the steps and layers that should be distinguished. The result is an informal description of the knowledge domain. Recently, formal specification languages have been developed for specifying and implementing complex reasoning systems which offer more possibilities for a formal specification of the knowledge domain (good overviews can be found in [5] and [30]).

Cognitive sciences Another source of open knowledge domains is cognitive science. An early case study of the FSA-project was the specification of SOAR. SOAR [15] is both a cognitive architecture based on Newell's theory of Human Problem Solving [20], and a computer program based on this architecture. An extensive formalization in Z is presented in [17], but this is in fact a specification of the program SOAR, not a high-level formalization of the theory behind SOAR. The specification in [2] comes closer to this. A more elaborate description of our view is in [11].

Linguistics In the field of linguistics much research is done on mathematization. With respect to formal specification of linguistic theories, we mention another case study of the FSA-project, which is concerned with the specification of a parser for natural language based on the Minimalist Program [1]. This work resembles the formalization in first-order logic and Prolog of Chomsky's theory of Government and Binding [29].

Medicine In the field of medicine much effort is invested in the production of implementations of medical systems. An example of this is [3], which gives many computer programs that perform clinical computations. More general work is done by Fieschi [6], this research focuses on expert system applications but is too restricted for our purposes.

3 Support in anaesthesia

Problem domain The context is: anaesthesia during thorax operations. At the Department of Anaesthesiology of the University Hospital Groningen a database system (Carola, documented in [8]) monitors the measurements performed during thorax operations. Based on this infrastructure, research is done to construct a system that can support anaesthetists during thorax operations [9].

Two essential ingredients are needed for the construction of a support system: computerized measurements and a formalization of medical knowledge. The measurements can be obtained from the Carola database and therefore we focus on the formalization of knowledge about these measurements.

We identify two types of anaesthesiological knowledge. The first is structural knowledge: knowledge about the concepts and their relations as present in anaesthesiology. Secondly, we identify medical facts as factual knowledge. In the model that we present, the factual knowledge is stored in a knowledge base. There is no ready-to-use model for anaesthesia available, and thus no formal model either. The research within FAN consists of defining a model for anaesthesia and formalizing it.

The factual knowledge stored in the knowledge base applies to the perfusion period. This is a critical period of a thorax operation. During the perfusion period a heart-lung machine takes over the functioning of heart and lungs.

Required functionality The aim of the FAN-project is to develop what is called a *support system* for anaesthetists. We do not call the envisioned system a decision support system since we aim at something different from what is commonly indicated by this term. As we perceive it, a typical decision support system computes which of a number of alternative options is best (according to particular criteria). The support system that we have in mind helps the anaesthetist by giving him better insight in the situation of the patient and by structuring the decisions that have to be taken.

The support we want to provide includes diagnosis (based on the measurements, the system infers possible diagnoses), simulation (the system enables the simulation of the application of a therapy), and treatment advice (the system advices the anaesthetist which therapy is applicable). A more detailed description of the functionality can be found in [24].

4 Formalization method and language

We outline the principles used in the FSA-project.

- Formal method: use of a formal specification language.

- Property oriented specification: the specification paradigm that is used is property based (as opposed to model based approaches).
- The formal specification language AFSL. It is inspired on the specification language COLD [4]. AFSL will be explained in somewhat more detail below. For FAN we use the static, declarative sub-language of AFSL.
- Graphical representation: to represent the signature of the specifications graphically we use a graphical notation based on the graph-visualization program da Vinci [7].
- Object-Oriented modeling technique: for the construction of the specification we use several ideas from the object oriented paradigm, viz. sub-typing and inheritance.

We discuss briefly the guidelines, language, and tools.

4.1 Guidelines

We present the guidelines used in the FAN project. The motivation is based on FAN and other case-studies; document [27] addresses these issues. Other sources of inspiration for the method were found in the literature on Object-Oriented Analysis, e.g. [25]. A specification is constructed in three parts: a dictionary, a signature, and an axiomatization. These three parts represent various aspects of a specification: intention, structure, and content.

Dictionary This is a list of concepts accompanied by *descriptions*. A description explains in informal terms what is covered by the concept. Additional (optional) parts are *examples*, *motivation* and *additional information* for each concept.

Signature The first step towards formalization is choosing the identifiers (names in the final specification) and their types for the concepts in the dictionary. Names can only refer to sorts, individual objects and functions. Individual objects have a sort name as type, and functions have the type $S_1 \times \ldots \times S_n \rightarrow S$ where $S_1, \ldots, S_n, S$ are sort names. The semantics of the names are (initially) given by informal axioms (stated in natural language). Diagrams play an important role in the construction of the signature.

Axiomatization The formalization is completed by adding axioms. The axioms must be such that the properties described in the dictionary and the corresponding informal axioms are satisfied.

4.2 Language

For presentation of the formal model, we use two types of representation. The first is a formal specification language, AFSL, with a formally defined syntax and semantics. The second way of presenting the model is the use of a graphical notation. The graphical notation is an additional representation, used to obtain a better overview over the specification. The meaning of this representation is explained in terms of the corresponding AFSL constructs. We discuss the two formats in turn.

AFSL: Almost Formal Specification Language The language AFSL is being developed as part of the FSA-project. It is an extension of first-order logic and is inspired on the specification language COLD [4]. AFSL is designed to support the guidelines that were mentioned in 4.1. Basically, AFSL is a language for writing first-order predicate logical theories (i.e. sets of axioms). To make predicate logic more appropriate for software specification, the language is extended by several constructs. A full account of AFSL can be found in [26]. We highlight some of the features of AFSL:

- AFSL is a typed first-order specification language, with equality and built-in sorts for boolean values, integer and real numbers, characters and strings. Functions may be partial. Predicates are regarded as boolean-valued functions. The language has a loose semantics, i.e. the semantics of a specification is not a single model but a class of models.
- Parameterized modules. Module definitions may contain parameters. Import of a module has the same effect as the textual substitution of the module text with the proper instantiations of the parameters.
- Sub-typing relation on sorts (denoted as <<<). When a function is defined on a type, it is also defined on its sub-types (inheritance). We allow multiple super-types.
- No overloading. Objects/functions with the same name should have the same semantics. To enforce this, the language does not allow for 'ad-hoc' overloading of object and function names. The only way to re-use a name is using sub-type polymorphism (a name is inherited from sorts higher in the sub sort hierarchy) or parameter polymorphism (a name is defined in a parameterized module, and imported via an instantiation of that module).
- Informal axioms. Axioms can be stated informally (as an intermediate step in the formalization process). Formal terms can be used in informal axioms by quotation; as such, they can be recognized and processed (e.g. type-checked) by software tools.

The syntax of AFSL is almost self explanatory. When necessary we explain constructs on the fly.

Graphical representation Diagrams are used to give overviews of larger parts of the signature definition. They contain information on object/function names, their types, and sub-typing relations. These diagrams, generated automatically from the specification text, proved to be an important instrument in the formal specification development process.

4.3 Tools

When working with large specifications the use of tools becomes inevitable. Within the FSA-project several tools have been developed, and for the FAN-project we used and experimented with the following tools:

- A parser and type-checker, implemented in the functional programming language SML [18] (the implementation is done by Indra Polak and the second author).
- The graph visualization system da Vinci of the University of Bremen (see [7]) for the construction of the diagrams.

- To manipulate the large number of files (every module is a separate file) we used AWK programs and UNIX shell scripts for the extraction of LaTeX and AFSL files from a general file-format (containing formal specification and documentation).

5 Formalizing anaesthesia

As stated in section 3, the desired functionality of the support system includes different tasks. Within FAN we additionally we distinguished the following phases:

Phase 1	Phase 2	Phase 3	Phase 4
Formalizing domain ontology and measurement data	Formalizing causal and has-part relations	Formalizing inference mechanism	Formalizing tasks and applications

The first phase introduces the objects of the specification. The second and the third phase focus on the relations between objects and the inference of propositions about objects, respectively. In the fourth phase we concentrate on the several tasks that can be performed using the inference mechanism, examples of those tasks are diagnosis and therapy advice. Currently, the FAN-specification includes phase 1 and 2. It is recognized that later phases may require extension of the specification of earlier phases. Method, language, and tools allow for easy accommodation of such extensions.

The division is inspired on the different layers in KADS [31]. In the KADS-terminology, the result of the first phase is the specification of the domain-layer, the second and third phase specify the inference layer, and the task layer corresponds with the last phase. Besides this 'horizontal' division, we also made a 'vertical' division that was already described above: we identify a so-called 'structural model' and a 'knowledge base'. The model presents the structural knowledge and the knowledge base contains the factual knowledge (the actual interpretations and relations).

The following examples of factual knowledge illustrate the type of knowledge that has to be formalized:

Cardiac output is equal to heart rate times stroke volume ($CO = HR \times SV$).

During the perfusion period, disconnection is unacceptable and airway obstruction is undesirable.

A too-high Mean Arterial Blood-pressure can cause Edema.

In subsections 5.1 – 5.4, we discuss specification of a structure for representing this kind of factual knowledge. In subsection 5.5, we use this structure to formalize the given examples of factual knowledge.

5.1 Overall characteristics of the specification

The complete specification consists of about 80 modules, with over 3000 lines of specification. The library modules for AFSL consist of a group of general modules,

specifying algebraic structures and standard functions on the built-in sorts of AFSL (50 modules with 1100 lines). The FAN related modules consist of three groups. In the first we identify the domain ontology and measured data, a second group of modules specifies relations between measurements (in total 30 modules with 1600 lines) and the third is the knowledge base that contains the formalized knowledge (currently divided in 2 modules containing 1200 lines). The knowledge base will grow further since it currently contains only a sample of the knowledge that is actually needed. The general and structure parts of the specification are expected to keep their current size.

5.2 Algebraic structures

Modules for the definition of algebraic structures, like fields, groups and orderings, follow the definition in the Larch-library [13]. The absence of overloading forced us to select carefully the names and properties of operators which are fixed for the whole specification.

5.3 Domain ontology and measured data

When working in an ill-structured domain like anaesthesia, a precise definition of the ontology is of vital importance. The distinction between terms like disease, syndrome, sign, and symptom must be made explicit. Besides this, much of the data is obtained via measurement of physical quantities. Therefore, the structural model must incorporate both the domain ontology and measured data. Some preliminary design decisions include:

- The patient is not modeled explicitly: measurements are assumed to be from the same patient.
- Measurement data are assumed to be available at any moment in time (no sampling). We use a 'default' value when a signal is not known.
- For discretizations of measurements, we use five values: too-low, decreased, normal, increased, too-high.

To structure the domain ontology, we identify three classes of sorts: value sets (contain basic values), courses of value (model basic values that vary of time), and parameters (the collection of all known properties of a patient).

The three value sets are:

QuantityS	(physical quantities)
QualityS	(discrete values)
BoolS	(the two boolean values)

The next layer consists of courses of value (functions from time-points to a value set):

$$\begin{aligned}
\textbf{SignalS} &= \textbf{TimePointS} \rightarrow \textbf{QuantityS} \\
\textbf{DiscreteSignalS} &= \textbf{TimePointS} \rightarrow \textbf{QualityS} \\
\textbf{ConditionS} &= \textbf{TimePointS} \rightarrow \textbf{BoolS}
\end{aligned}$$

For parameters we distinguish three types: magnitudes, levels, and phenomena. *Magnitudes* represent parameters with numerical values (typically measurement data, for example blood-pressure). More discrete parameters, like anaesthesia-depth or ventilation, are modeled as *levels* with five discrete values. The 'yes/no-parameters', e.g. the presence of 'high-blood-pressure', are modeled by *phenomena*. Phenomena will form the basis for phase 2 and 3 of the specification, in particular the inference relations.

A second subdivision concerns different modalities or worlds-of-reasoning, we model these by introducing three different *worlds*. Besides the *actual* (current) value of a parameter we also want to refer to the *target* (medically ideal) value of a parameter and the *default* value for the case that the measurements are not available.

Parameters are modeled as clusters of three course-of-values (one for each world-of-reasoning):

MagnitudeS = **WorldS** → **SignalS**
LevelS = **WorldS** → **DiscreteSignalS**
PhenomenonS = **WorldS** → **ConditionS**

Later we will introduce some additional attributes for parameters. To specify courses and parameters, we use the parameterization and sub-typing mechanisms of AFSL. We will discuss the construction of the sort **ParameterS** from the built-in sorts and library modules.

Value sets The sort **BoolS** is an AFSL primitive. Qualities are defined as an enumerated sort **QualityS** with 5 elements:

```
MODULE QualityM

SORT   QualityS

OBJ    TooHigh    : QualityS
OBJ    Increased  : QualityS
OBJ    Normal     : QualityS
OBJ    Decreased  : QualityS
OBJ    TooLow     : QualityS

IMPORT Enum5M[QualityS, TooLow, Decreased, Normal,
                        Increased, TooHigh]

END MODULE
```

The import of **Enum5M** forces that the five objects are distinct and introduces a linear ordering on the elements.

We specified a model for quantities including but not limited to the SI-quantities (Système International). The medical field uses many units other the those of the SI. Therefore conversion relations between different units must be specified. We model physical dimensions, units and quantities as follows:

```
MODULE QuantityM
    SORT    DimensionS
    OBJ     Time      : DimensionS
    OBJ     Length    : DimensionS
    (...)

    SORT    UnitS
    OBJ     Second    : UnitS
    OBJ     Meter     : UnitS
    (...)

    SORT    QuantityS

    FUNC    Quant: RealS, UnitS -> QuantityS
    FUNC    Dim  : QuantityS -> DimensionS
    (...)

    END MODULE
```

So, **Quant(10,Second)** is a quantity of 10 seconds and **Dim Quant(10,Second) = Time**. The full specification, reported in [21], contains also arithmetical operations on dimensions and units, and scaling functions on units. SI is also formalized in Z [14], where the typing mechanism of Z is extended to allow for type checking on physical units.

An interesting part of the SI specification is concerns the modeling of time. Since AFSL has no standard language construct for temporal expressions, time must be modeled explicitly (as in the other case studies [12] and [28]). Elements of **TimeS** represent a quantity of time. Elements of **TimePointS** denote a particular point in time and **TimePeriodS** models time-periods.

```
    MODULE TimeM

    SORT    TimeS <<< QuantityS
    DECL    time   : TimeS
    AXIOM   Dim(time) = Time

    SORT    TimePointS
    SORT    TimePeriodS
    DECL    tp     : TimePointS
    DECL    tper   : TimePeriodS

    FUNC    Begin : TimePeriodS -> TimePointS
    FUNC    End   : TimePeriodS -> TimePointS
    AXIOM   Begin(tper) =< End(tper)
    (...)

    END MODULE
```

Using this specification, we can introduce several derived concepts, e.g.:

```
    FUNC    Duration : TimePeriodS -> TimeS
    FUNC    Contains : TimePeriodS,TimePointS -> BoolS
```

Course of values An important notion is time-dependent value, modeled as a function from time points to a value set. We collect them in the sort **CourseS** and distinguish three sub-sorts, depending on the value set involved.

Since AFSL is a first order language, we cannot define function types directly. So, we introduce the abstract sort **FunctionS[ArgS,ResultS]** of functions from **ArgS** to **ResultS**. Associated with these functions is a (partial) operation **.** which models function application.

```
MODULE FunctionM[ ArgS, ResultS ]

SORT ArgS
SORT ResultS
SORT FunctionS[ArgS,ResultS]

DECL arg  : ArgS
DECL func : FunctionS[ArgS,ResultS]

FUNC .    : FunctionS[ArgS,ResultS], ArgS -> ResultS PARTIAL

END MODULE
```

The sorts **ArgS** and **ResultS** are the parameters of the module. Observe that AFSL supports implicit quantification in axioms (here for **func1** and **func2**).

The module **CourseM[XS]** introduces the parameterized sort **CourseS[XS]** as a subset of the functions of **TimePointS** to **XS** to the arbitrary sort **XS**.

```
MODULE CourseM[XS]

SORT   XS

IMPORT TimeM
IMPORT FunctionM[TimePointS,XS]

SORT   CourseS[XS] <<< FunctionS[TimePointS,XS]
(...)

END MODULE
```

We use **CourseM** for defining **SignalS** (with **QuantityS** substituted for **XS**), for defining **DiscreteSignalS** (substituting **QualityS** for **XS**), and for defining **ConditionS** (with **BoolS** substituted for **XS**). Note that the definition of **CourseM** is more general and in the next paragraph we will re-use this module to define a fourth time-dependent sort.

Parameters With every parameter three courses of values are associated: the default course, the target course and the actual course. To select the different courses, we introduce the sort **WorldS** containing **Actual**, **Target**, and **Default**.

```
MODULE ParameterM[ XS ]

SORT   XS
IMPORT CourseM[XS]

SORT   WorldS

OBJ    Actual  : WorldS
OBJ    Target  : WorldS
OBJ    Default : WorldS

SORT   AnyParameterS

FUNC   Description : AnyParameterS -> StringS
FUNC   Observable  : AnyParameterS -> BoolS

SORT   ParameterS[XS] <<< AnyParameterS

FUNC   Course : ParameterS[XS], WorldS -> CourseS[XS]

END MODULE
```

The function **Description** gives the full medical name of a parameter. Observable parameters can be observed by the physician possibly with the aid of measurement equipment, e.g. the body-temperature is observable.

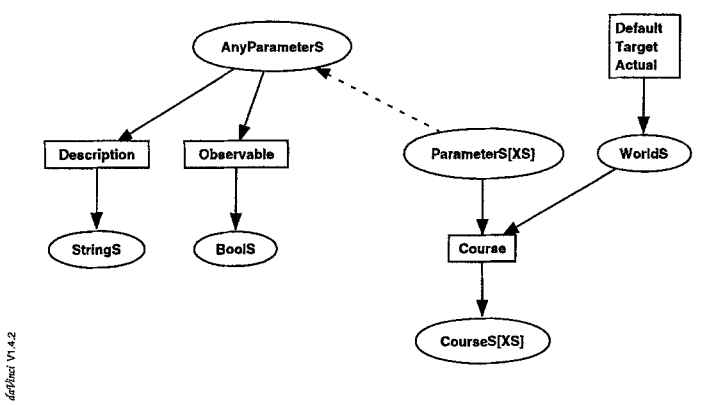

Fig. 1. Diagram of the module **ParameterM**.

A graphical representation of this module is given in figure 1. The abstract sort **ParameterS[XS]** is a subset of **AnyParameterS** (indicated with the dotted arrow). **ParameterS[XS]** therefore inherits the functions **Description** and **Observable** as defined on **AnyParameterS**. On **ParameterS[XS]** we define the function **Course** with a two arguments: a parameter and a world.

We obtain the three parameter sorts by importing this module with the correct instantiation for **XS**. We introduce the names **MagnitudeS**, **LevelS**, and **PhenomenonS** as abbreviations (using the keyword **ALIAS**):

```
MODULE FANParameterM

IMPORT QualityM
IMPORT QuantityM

IMPORT ParameterM[QuantityS]
IMPORT ParameterM[QualityS]
IMPORT ParameterM[BoolS]

ALIAS  MagnitudeS  === ParameterS[QuantityS]
ALIAS  LevelS      === ParameterS[QualityS]
ALIAS  PhenomenonS === ParameterS[BoolS]

FUNC   Discrete : MagnitudeS -> LevelS PARTIAL
(...)

END MODULE
```

From the module **FANParameterM** we omitted some additional axioms concerning the dimension and units on magnitudes and the corresponding quantities. The partial function **Discrete** gives the level corresponding with a magnitude according to a discretization.

An interesting point is the valuation function defined on **PhenomenonS**. We mentioned *disconnection is unacceptable* as an example of factual knowledge that the structure must be able to capture. 'Unacceptable' is modeled as a **Valuation** of phenomena:

```
SORT   ValuationS

OBJ    Necessary    : ValuationS
OBJ    Desirable    : ValuationS
OBJ    Undesirable  : ValuationS
OBJ    Unacceptable : ValuationS

IMPORT Enum4M[ValuationS, Necessary, Desirable,
                          Undesirable, Unacceptable]

IMPORT CourseM[ValuationS]
FUNC   Valuation: PhenomenonS -> CourseS[ValuationS]
```

It may be surprising that the result of the valuation function is a time dependent value. In general, the valuation of a certain phenomenon is not constant during an operation. Consider the phenomenon "low body temperature" (e.g. body temperature is below 25°C); under normal conditions this is unacceptable, while during the perfusion period it is necessary. By making the valuation a time-dependent function we can specify this as:

```
DECL    tp : TimePointS
OBJ     LowBodyTemp: PhenomenonS

AXIOM   PerfusionPeriod Contains tp        ==>
        Valuation(LowBodyTemp).tp = Necessary

AXIOM   Not (PerfusionPeriod Contains tp) ==>
        Valuation(LowBodyTemp).tp = Unacceptable
```

Operations on parameters Given the parameters as a data structure for our model, we define operations on parameters. First we define operations on value sets. Then we define a general lifting scheme to introduce operators on course-of-values and parameters based on those on the value sets (e.g. logical and numerical operators).

On **QuantityS** we define the arithmetical operations and a linear order, imposing some restrictions on the dimensions of the operands. On elements of **QualityS** we define the (linear) ordering **=<**. The operators on **BoolS** are as usual.

The second type of operation is the discretization of quantities to qualities. We define the function **Discretize** to perform the discretization of a quantity according to four bounding quantities:

```
FUNC Discretize : QuantityS, QuantityS, QuantityS, QuantityS,
                  QuantityS -> QualityS
```

The axiomatization of this function is straightforward: the first four arguments indicate the bounds for the five discretizations, the last quantity is to be discretized. Using this discretization, we also defined an relative discretizer which discretizes a quantity using four scaling factors and a reference quantity as bounds.

Now we want to lift these functions from the level of value sets to the level of course of value and parameters. An ad-hoc way is via overloading, but that is not allowed in AFSL. We prefer a solution based on parameterization, illustrated below for binary functions.

```
MODULE Course2LiftM[XS,YS,ZS,Oper]

SORT    XS
SORT    YS
SORT    ZS

IMPORT CourseM[XS]
IMPORT CourseM[YS]
IMPORT CourseM[ZS]

DECL covx : CourseS[XS]
DECL covy : CourseS[YS]
DECL tp : TimePointS

FUNC   Oper : XS, YS -> ZS
FUNC   Course2[Oper] : CourseS[XS], CourseS[YS] -> CourseS[ZS]
```

```
AXIOM (covx Course2[Oper] covy).tp = (covx.tp) Oper (covy.tp)

END MODULE
```

This module introduces the binary function **Course2[Oper]** (given a function **Oper** from **XS**, **YS** to **ZS**) and defines it point-wise. An example of the use of this scheme is the definition of multiplication on signals, given the multiplication on quantities (**Course2[*]**):

```
IMPORT Course2LiftM[QuantityS,QuantityS,QuantityS,*]
```

The scheme for lifting operators to **ParameterS** is similar.

5.4 Relations between phenomena

The previous subsection was concerned with the definition of parameters. They are used to model properties of a patient and to classify measured values. A different type of knowledge has to do with relations between parameters. Physiological knowledge often concerns quantitative relations between magnitudes: the $CO = HR \times SV$ example mentioned above. Diagnostic knowledge mainly concerns qualitative relations between phenomena. The discretization functions discussed above bridge the quantitative and the qualitative values. For modeling qualitative relations we distinguish:

- Causal relations. For example "A Too-high Mean Arterial Blood-pressure can cause Edema".
- Has-part relations. If a disease has several symptoms and signs, then these symptoms and signs have a has-part relation with the disease. An example: "Symptoms of an increased or higher level of Stress are increased or higher levels of Mean Arterial Blood-pressure and Heart Rate".

Associated with each relation is a 'strength'. The possible strengths are facultative, obligatory, and pathognomonic. Facultative denotes a positive correlation between two phenomena, obligatory indicates an implication, and pathognomonic is used to indicate a bi-implication between two phenomena.

For the formalization of relations there are two options. We can model relations directly as boolean functions (stated as axioms using the logical operations) or we can objectify them by introducing special sorts for links. We have chosen for the last option for the following reasons:

- Anaesthetists tend to reason about relations as objects. The presence or absence of a causal relation (for example some kind of reflex) is an information item itself.
- When relations are objects, it is easier to classify them, for example by introducing an attribute for strength.
- Reasoning with the relations can now be described on a higher level of abstraction. The relations are then parameters of the rules.

To model this, we introduce the sorts **CausLinkS** and **HasPartLinkS**. Additionally we decided to regard the objects denoting links to be of type **PhenomenonS**:

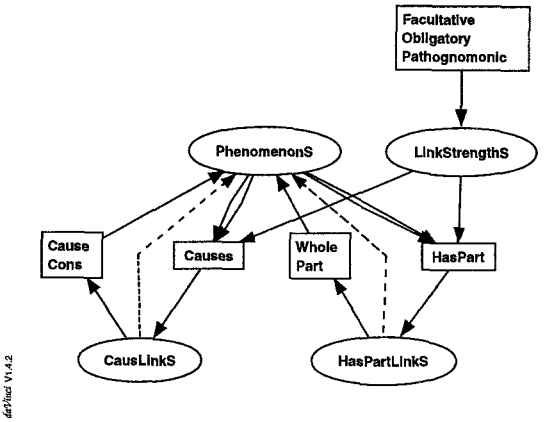

Fig. 2. Diagram of the link-sorts.

```
SORT   CausLinkS     <<< PhenomenonS
SORT   HasPartLinkS  <<< PhenomenonS
```

Objects of both **CausLinkS** and **HasPartLinkS** are constructed from two phenomena and a **LinkStrenghtS** (notice that Caus-links and Has-part-links are themselves phenomena):

```
FUNC Causes   : PhenomenonS, PhenomenonS, LinkStrengthS
                -> CausLinkS     PARTIAL
FUNC HasPart  : PhenomenonS, PhenomenonS, LinkStrengthS
                -> HasPartLinkS PARTIAL
```

Besides the constructor functions, we also have for each link-type two destructor functions (**Cause/Cons** and **Whole/Part**, see figure 2).

5.5 Knowledge base

The knowledge base contains the formalization of knowledge obtained from interviews and knowledge from physiological handbooks. The knowledge items are expressed in the signature provided by the model. Most of the expressions are straightforward, like our example:

> During the perfusion period, disconnection is unacceptable (...) and airway-obstruction is undesirable (...)

This is represented in the knowledge base as:

```
OBJ    Disconnection      : PhenomenonS
OBJ    AirwayObstruction  : PhenomenonS

AXIOM  Contains(PerfusionPeriod,tp)
```

```
        ==>  (Valuation(Disconnection).tp     = Unacceptable)
AXIOM   Contains(PerfusionPeriod,tp)
        ==> (Valuation(AirwayObstruction).tp = Undesirable)
```

Furthermore, the physiological constraints are represented in the knowledge base. One equation is $CO = HR \times SV$, which expresses that cardiac output equals heart rate times stroke volume. In the knowledge base this is formulated as:

```
OBJ   CO : MagnitudeS
OBJ   HR : MagnitudeS
OBJ   SV : MagnitudeS
DECL  w  : WorldS
DECL  tp : TimePointS

AXIOM Course((CO Param2[=] (HR Param2[*] SV)),w).tp
```

Note that we use the lifted, point-wise defined versions of = and *.

As an example of the specification of causal relations, we can formalize "A Too-high Mean Arterial Blood-pressure can cause Edema" as:

```
OBJ   PArtM  : MagnitudeS
OBJ   Edema  : PhenomenonS
AXIOM Course(
       Causes( (Discrete(PArtM) Param2[=] Param0[TooHigh])
             , Edema
             , Facultative),w).tp
```

The has-part relation "Symptoms of an increased or higher level of Stress are increased or higher levels of Mean Arterial Blood-pressure and Heart Rate" can be formalized as:

```
OBJ   Stress : LevelS
AXIOM Course(
       HasPart( Stress Param2[>=] Param0[Increased]
              , (Discrete(PArtM) Param2[>=] Param0[Increased])
                Param2[Or]
                (Discrete(HR)    Param2[>=] Param0[Increased])
              , Obligatory),w).tp
```

For a concise definition of the knowledge base we use abbreviations, for example:

```
FUNC EveryWhere : PhenomenonS -> BoolS
AXIOM EveryWhere phen
      <=>
      FORALL w,tp (Course(phen,w).tp)
```

The law $CO = HR * SV$ can now be re-formulated as:

```
AXIOM EveryWhere (CO Param2[=] (HR Param2[*] SV))
```

6 The FAN-project

The FAN-project went through several stages which we briefly discuss.

De Geus and Rotterdam [9] developed the Carola-database system and investigated the use of constraint satisfaction techniques for diagnosis. This thesis also contains the first attempts at formalizing anaesthesia for decision support. Two types of knowledge are identified: arithmetical constraints (mainly on physiological variables) and qualitative knowledge. A small implementation in a rule-based language RL was presented for arithmetic constraints [10]. The second type of knowledge is concerned with qualitative knowledge, and in [9, pp. 229] it is stated that:

> It seems most appropriate to represent knowledge with a language that can capture the different aspects of knowledge uniformly.

A first attempt to develop such a language is given in [22]. This report presents an ad-hoc formalism, baptized LaFAn (Language for Formalization Anaesthesia). LaFAn was used to give a first formalization of interviews of anaesthetists. It was hard to give a good definition of LaFAn since it was not stable. The main reason for this is that one first needs a proper theory of anaesthesia before being able to define a language to reason about it. To overcome these problems, we decided to take a new path: a general purpose formal specification language to formalize the concepts of anaesthesia. The formalization was carried out as part of the FSA-project. We started to work on the formalization of the model using the language and method proposed in the FSA-project. The two main tasks, viz. to define a model and to formalize it, were dealt with simultaneously.

The first result of the FAN-project was a specification written in the non-modular version of AFSL. One major drawback of this specification was that it did not provide a clear view on the underlying model. To improve this, a graphical representation technique was introduced.

Report [23] contains a reformulation of the model. The modular version of AFSL and a graphical notation are used to present the specification in a modular way. The overwhelming number of identifiers (function and sort names), all necessary to describe the model in full detail, forced us to use the parameter mechanisms of AFSL extensively. To enforce the use of these possibilities, a radical decision was taken: *No overloading allowed.* Other restrictions were to remove the parts that had to do with therapy advice and prediction/simulation. This way the formalization became more rigorous and less complex.

The final report of the FAN-project [21] contains the type-checked version of the FAN-specification, with automatically generated diagrams. The knowledge with respect to diagnosis is separated from the knowledge that is needed to interpret measurements. Furthermore, the aspect of data sampling is not treated.

Still the project needs further development. Especially the validation of the formal model is important. The FAN-project was mainly devoted to the development of the model and the formalization. Given these descriptions, we need to validate the knowledge that we have formalized.

6.1 Other phases

The report on FAN [21] contains the formal specification, structural model and knowledge base, of the first two phases. The next step is the formalization of the inference mechanisms, in particular to infer diagnoses based on the measured data. Further steps may consist of:

- The formalization of more factual knowledge in terms of the developed model. This includes the analysis of more interviews and the incorporation of physiological models from literature.
- The validation of the specification using a prototype. The functionality of this prototype is to infer simple diagnoses from the measured data. In view of the declarative nature of the specification, a declarative programming language is best suited to implement a prototype.

We think that the construction of a prototype based on the first three phases is important for the validation of the specified knowledge. After validation the fourth phase (design of required functionality of the support system) is feasible.

7 Conclusions

We have presented an overview of the specification process of a model for anaesthesiology. This model is intended as a basis for the specification and construction of a support system.

With the specification we are able to express arithmetical constraints as well as other declarative knowledge. Furthermore the specification gives a structured presentation of the concepts that we found important in the knowledge domain. The final version of the specification of the model and the knowledge base is available in [21].

We summarize some of the lessons that we have learned from this formalization case:

- The use of a special purpose language, like LaFAn, is not very productive in a large and not fully explored domain like anaesthesiology. In such a domain, first a proper formal model has to be constructed. A general purpose language, like AFSL, is suited for the description of such a model. Only when that model is stable, one can think of designing a special purpose language.
- The use of a first order language, with sub-typing and parameterized modularization (offering a poor man's second order facility) and no overloading, like AFSL, leads to a dilemma. On the one hand, overloading was not added to AFSL in order to improve the clarity of specifications by avoiding semantically different functions with identical names. It is our impression that this design decision on AFSL definitely had a beneficial effect on the overall structure of the specification. On the other hand, there are situations where overloading is common and considered to be natural. E.g. in the case of lifting an operation on some sort S to functions of type $S' \rightarrow S$. In AFSL, lifting of an operation can be accomplished via a parameterized module, see e.g. **`Course2LiftM`**.

- Restructuring of the specification was mainly performed during the the definition of the signature of the specification. This led to the guideline to restructure at the signature level, before full axiomatization takes place. The use of a graphical notation is useful, especially when defining the signature.
- A property-oriented way of specification seems a proper way of formalizing an open knowledge domain. Additional attributes can easily be given to objects, without conflicting with earlier introduction.
- Subtype and parameter polymorphism help to reduce the number of similar concepts. In earlier versions, magnitudes, levels and phenomena were not subsets of a common sort. The introduction of **`ParameterS[XS]`** allows a uniform treatment of parameters. Furthermore, such abstractions make it possible to re-use specifications. An example is the specification of the mathematical structures and course of value.

Acknowledgments

We acknowledge Indra Polak for implementing parts of the type checker. We thank B. Ballast, G. Massée, J. de Jong, and P. Hennis for the possibility to interview them. Three anonymous referees are acknowledged for their comments on an earlier version of this paper.

References

1. N. Chomsky. A minimalist program for linguistic theory. Technical report, MIT Occasional Papers in Linguistics, 1992.
2. R. Cooper, J. Farringdon, J. Fox and T. Shallice. Levels of description in specifying Soar. In: *Proceedings EuroSoar-5*, 1991.
3. D. John Doyle. *Computer Programs in Clinical and Laboratory Medicine.* Springer Verlag, 1989.
4. L.M.G. Feijs and H.B.M. Jonkers. *Formal Specification and Design.* Cambridge Tracts in Theoretical Computer Science 35. Cambridge University Press, 1994.
5. D. Fensel and F. van Harmelen: A Comparison Of Languages Which Operationalize And Formalize KADS Models of Expertise, *The Knowledge Engineering Review*, vol 9(2), 1994.
6. M. Fieschi. *Artificial Intelligence in Medicine.* Chapman and Hall, 1990. Original title: Intelligence Artificielle en Médecine des Systèmes experts (translated by D. Cramp).
7. M. Fröhlick and M. Werner. *daVinci V1.4 User Manual.* Department of Computer Science, University of Bremen, January 1995.
8. A.F. de Geus. *The Carola Database – User Manual.* Department of Anaesthesiology, University Hospital Groningen, 1990.
9. A.F. de Geus and E.P. Rotterdam. *Decision Support in Anaesthesia.* PhD thesis, Department of Anaesthesiology, University of Groningen, 1992.
10. F. de Geus, E. Rotterdam, S. van Denneheuvel, and P. van Emde Boas. Physiological modeling using RL. In M. Stefanelli, A. Hasman, M. Fieschi, and J. Talmon, editors, *Proceedings of AIME '91*, pages 198 – 210. Springer Verlag, 1991.
11. R. Groenboom and G.R. Renardel de Lavalette. Formal Specification and Soar: does cognitive science need formalisms. In: *Proceedings of workshop Euro-Soar6*, 1992.

12. R. Groenboom, R.M. Tol, and E. Saaman. Formal specification and design of a simple real time kernel. In A. Nieuwendijk, editor, *Proceedings Accolade '94*, pages 87 – 102. Dutch Gradute School in Logic, Amsterdam, 1995.
13. John V. Guttag and James J. Horning, with S.J. Garland, K.D. Jones, A. Modet, and J.M. Wing. *Larch: Languages and Tools for Formal Specification.* Springer-Verlag Texts and Monographs in Computer Science, 1993.
14. I.J. Hayes and B.P. Mahony. Using units of measurement in formal specifications. *Formal Aspects of Computing*, 7: 329 – 347, 1995.
15. J.E. Laird, A. Newell, and P.S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33(1): 1 – 64, 1987.
16. I. van Langevelde, A. Philipsen, and J. Treur. Formal specificartion of compositional architectures. In: *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92)*, Vienna, 1992.
17. B.G. Miles. *A specification of the Soar cognitive architecture in Z.* Technical report CMU-CS-92-169, Carnegie Mellon University Pittsburgh PA, 1992.
18. C. Myers, C. Clack, and E. Poon. *Programming with Standard ML.* Prentice Hall, 1993.
19. M. Naftalin, T. Denvir, and M. Bertran (eds). *FME '94: Industrial Benefit of Formal Methods.* LNCS 873, Springer Verlag, 1994.
20. A. Newell and H.A. Simon. *Human problem solving.* Prentice-Hall, 1972.
21. G.R. Renardel de Lavalette (ed). *Formalization of Anaesthesia. Report on the FAN-project.* Technical Report, Department of Computing Science, University of Groningen, (in preparation).
22. E.P. Rotterdam. *Anesthesiekennis in Lafan.* Technical report R93026, Department of Medical Information Science, University of Groningen, 1993. (In Dutch).
23. E.P. Rotterdam. FAN: Formalizing anaesthesiology in AFSL. Technical Report R9406, Department of Medical Information Science, University of Groningen, 1994.
24. E.P. Rotterdam. *Desired Functionality of a Support System.* In: [21].
25. J. Rumbaugh et. al. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.
26. E. Saaman. *User manual AFSL.* Department of Computing Science, University of Groningen, (in preparation).
27. E. Saaman and G.R. Renardel de Lavalette. *Object-Oriented Formalization*, Manuscript, Department of Computing Science, University of Groningen, 1995.
28. E. Saaman, P. Politiek, and K. Brookhuis. Specification and Design of InDeter-1. Technical report, Traffic Research Centre, University of Groningen, 1994. Deliverable 9 (321A).
29. E.P. Stabler. *The logical approach to syntax.* MIT Press, Cambridge, 1992.
30. J. Treur and T. Wetter (eds). *Formal Specification of Complex Reasoning Systems.* Ellis Horwood, 1993.
31. B.J. Wielinga, A. Th. Schreiber, and J.A. Breuker. KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1), March 1992.
32. J.C.P Woodcock and P.G. Larsen (eds). *FME '93: Industrial-Strength Formal Methods.* LNCS 670, Springer Verlag, 1993.