Formalisation for decision support in anaesthesiology

Renardel de Lavalette, Gerard R.; Groenboom, Rix; Rotterdam, Ernest; Harmelen, Frank van; Teije, Annette ten; Geus, Fred de

Published in:
Artificial Intelligence in Medicine

Publication date:
1997

ELSEVIER Artificial Intelligence in Medicine 11 (1997) 189–214

Artificial Intelligence in Medicine

Formalisation for decision support in anaesthesiology

Gerard R. Renardel de Lavalette [a,*], Rix Groenboom [a], Ernest Rotterdam [b], Frank van Harmelen [c], Annette ten Teije [c], Fred de Geus [d]

[a] *Department of Computing Science, University of Groningen, P.O. Box 800, 9700 AV Groningen, The Netherlands*
[b] *Syllogic, Houten, The Netherlands*
[c] *Department of Computer Science, Free University, Amsterdam, The Netherlands*
[d] *University Hospital, Groningen, The Netherlands*

Received 30 September 1996; received in revised form 30 December 1996; accepted 7 April 1997

Abstract

This paper reports on research for decision support for anaesthesiologists at the University Hospital in Groningen, the Netherlands. Based on CAROLA, an existing automated operation documentation system, we designed a support environment that will assist in real-time diagnosis. The core of the work presented here consists of a knowledge base (containing anaesthesiological knowledge) and a diagnosis system. The knowledge base is specified in the logic-based formal specification language AFSL. This leads to a powerful and precise treatment of knowledge structuring and data abstraction. © 1997 Elsevier Science B.V.

Keywords: Anaesthesiology; Formalisation of knowledge; Data abstraction; Diagnostic reasoning

* Corresponding author. Tel.: + 31 50 3633939; fax: + 31 50 3633800; e-mail: grl@cs.rug.nl

PII S0933-3657(97)00031-6

1. Introduction

We report in this paper on the research project FAN (formalisation of anaesthesiology). FAN aims to contribute to decision support for anaesthesiologists by formalisation of the knowledge required for this purpose. In order to realise this, FAN combines methods and techniques from Artificial Intelligence (e.g. diagnostic reasoning and knowledge elicitation) and Software Engineering (e.g. formal specification and prototyping).

1.1. Medical care and information technology

The idea of applying computers in medical care is quite obvious as it stands, but by no means an easy job. One general reason for this, is the inherent complexity of the design of information systems. A more specific reason lies in the human-centred character of medical care, where many processes, e.g. those involving personal contact and care, are not amenable to automatisation.

There are, however, abilities in which humans are outperformed by computers: real-time processing of huge amounts of data, vigilance and alertness during long periods of time. Such abilities are required in the operation room and intensive care, and it is not surprising that information technology is frequently applied here. The processing involved encompasses not only data collection and documentation, but also the more challenging task of diagnosis, therapy advice and prediction.

This paper reports on the FAN project, which is involved in this transition from more traditional activities of medical data collection and documentation to the more advanced tasks of diagnostic reasoning and therapy advice. As such, FAN is part of the long-term activities of the Department of Anaesthesiology of the University Hospital Groningen, which started in the beginning of the 1980s with the design and implementation of the documentation and data management system CAROLA (see Section 2).

1.2. The role of formal specification

The transition from a relatively traditional data management system like CAROLA to an extension with knowledge-intensive functionalities like diagnostic reasoning and therapy advice is in fact a transition from data processing to knowledge processing, from the structured and rather well-understood domain of physiological measurements to the less structured, open and sometimes ill-defined domain of physiological and medical knowledge.

To deal with this new situation, one has to handle these adverse properties. For this purpose, we chose to apply *formal specification*, a modern approach in Software Engineering especially geared towards disambiguation and structuring. Now the question is: how to formalise a weakly structured and open domain (in this case: the relevant medical knowledge)? Here we used the method developed in

the FSA project (which is related to FAN), consisting of guidelines, a formal language and tools. The guidelines divide the formalisation task in three subtasks: development of a dictionary, a signature, and an axiomatisation; see Section 5 for more information.

It is our experience that the use of a method is indispensible when creating a formal specification of a knowledge domain, especially when domain specialists without a formal/mathematical background are involved. The FSA method turned out to be effective in this respect: it divides the formalisation task in three subtasks and it supports the communication with domain specialists via the dictionary and a tool for creating signature diagrams. The hard part of the formalisation work lies in choosing adequate conceptualisations and abstractions. An example of a useful conceptualisation is the distinction of several modalities of courses-of-values (e.g. heart rate or body temperature during operation): actual, desired, and default (when no data are available); an example of useful abstraction is the notion *phenomenon*, encompassing illness, symptom, sign, syndrome, etc.

1.3. FAN: project description

FAN is an interdisciplinary project at the University Hospital Groningen and the Computing Science Department of the University of Groningen, with active participants from the Amsterdam Universities. It started in 1993, partly as the continuation of the CAROLA project at the University Hospital Groningen which resulted in the automated operation documentation system CAROLA. When CAROLA was realised, it was natural to think about extending its functionality: assist the medical staff during thorax operations in the *evaluation* of the state of the patient by not only providing the relevant data, but also suggesting possible conclusions, viz. diagnoses and therapy selection. These extended functionalities of CAROLA require the availability of medical knowledge in a form amenable to computer use. However, such medical knowledge is normally only available in written form (e.g. in textbooks) or not even directly accessible (e.g. knowledge and experience stored in the heads of medical specialists). As a consequence, it has to be made explicit (in the latter case) and to be formalised (i.e. written in a formal language, see Section 5).

Another starting point for FAN was the Formal System Analysis (FSA) project at the Computing Science Department of the University of Groningen, which aims at the development of a method and tools for developing formal specifications. For FSA, FAN is one of two major case studies, the other being the formalisation of Chomsky's Minimalist Program [2], a linguistic theory.

FAN serves the following goals:

- providing descriptions of anaesthesiological knowledge required for the design and development of decision support systems;
- testing a general method for creating such descriptions;
- in the long run: contributing to the improvement of medical care by making medical knowledge accessible for automated processing.

1.4. The rest of this paper

We start in Section 2 with a short overview of the operation documentation system CAROLA, which forms the starting point of the FAN project reported here. Section 3 contains a description of the tasks we aim to provide decision support for, and some remarks about the knowledge required for these tasks. In Section 4, we give a rather detailed treatment of one of these tasks, viz. value processing and abstraction. The formalisation method used for obtaining the formal specification of the knowledge base is presented in Section 5, together with a survey of the structure and the contents of the knowledge base, and with some remarks on the formalisation process. In Section 6, we present the diagnostic reasoning method used in FAN and compare it with a general framework for diagnostic reasoning. Section 7 is about a prototype that is being developed for diagnosis during perfusion. We end with a comparison with other work (Section 8) and some concluding remarks (Section 9).

2. The system CAROLA

CAROLA ([8,15]) is a documentation and data management system for anaesthesia during cardiovascular and other thorax surgery. (It is named after one of the students who assisted in the documentation tasks that are now performed by the system.) It has been operational at the University Hospital in Groningen (Netherlands) since 1983, and also at the Kerckhoff-Klinik in Bad Nauheim (Germany). Over 35 000 operations have been documented with CAROLA.

The first version of CAROLA was a stand-alone system based on non-standard hardware, a proprietary operating system and software, communication over serial lines, displaying its data during the operation on a plotter. In 1985 a database was added, used for archiving, administration and research. For the latter purpose, an easy-to-use query language was developed. In 1994, a second version of CAROLA was taken into deployment, based on a graphical workstation and with software running on top of HP-UX and X-Windows.

CAROLA automatically documents physiological variables measured by monitoring equipment (e.g. blood pressures, heart rate, temperatures) and by peripheral equipment (ventilator, heart lung machine); this yields 32 parameters, most of them measured once a minute. Information on the phase of the operation, drugs and fluids, and over 20 physiological variables that cannot be measured automatically, are documented manually by the anaesthesiologist. Identification data and other preoperative information are obtained from the Hospital Information System, which also communicates to CAROLA the results of laboratory tests performed during the operation (e.g. analysis of blood samples) and the data from shared equipment.

At the end of each operation, a paper document for the status and a copy for the paper archive are produced, and the record (with an average size of 80 Kb per operation, after conversion to the database format ca. 50 Kb) is added to the CAROLA database.

3. Desired functionality

The system to be developed on top of CAROLA should support the following functional tasks.

Value processing and abstraction: This is the transformation of quantitative measurements (the raw data from the CAROLA system) to qualitative phenomena (information that can be used for diagnosis). This task is elaborated in Section 4.

Diagnosis: This task is described in more detail in Section 6.

Therapy advice: This is the task that selects therapies that may cure the disease(s) of the (differential) diagnosis. To accurately assess and quantify the side effects of a particular therapy, simulation may be required (see below). Another aspect is therapy quantification, i.e. computation of the dose (to be used in a medicament administration), the infusion rate (for a particular infusion) and the concentration of medicaments in an infusion mixture. These quantitative aspects of therapy depend on parameters of the patient like sex, age, heart function, liver function, and lung function.

In this paper, we do not go deeply into the decision support for this task, but we include the specification of some relevant data types (such as medicaments, infusions and ways to administer these) in the knowledge base.

Simulation: This involves forecasting the further development of the condition of the patient, possibly assuming the application of a particular therapy. This is a subject for future research in FAN.

3.1. Knowledge involved in the functional tasks

It is relevant to distinguish qualitative and quantitative knowledge here. From this perspective, the knowledge flow is roughly as follows. Firstly, the support process starts quantitatively with value propagation. Then value abstraction translates this to the qualitative level, after which diagnosis, therapy selection and therapy advice take place. Subsequently, therapy quantification, prediction and simulation reconcretise to the quantitative level.

Quantitative relations include equations and, for continuous simulation, differential equations. The translation between quantitative and qualitative levels is done by means of inequality operators like $>$, $<$, $\geqslant$, $\leqslant$.

Qualitative relations used in FAN include nosological and causal relations. Nosological relations link physiological phenomena to less comprehensive phenomena that they consist of. For example a disease can be described in terms of the signs and symptom complexes by which it is characterised. Causal relations express cause-effect-relationships between phenomena. Causal relations can be characterised by the time-lapse between cause and associated effect.

Both nosological and causal relations can be characterised by a strength, which can have three values: facultative, obligatory and pathognomonic. The semantics of these strengths is discussed in Section 6.

4. Value processing and abstraction

The goal of the value abstraction task is to make the data more amenable for both the anaesthesiologist and the diagnostic component of our system. It achieves this goal by reducing the amount of detail per parameter by assigning qualitative judgements (e.g. too high, too low) to the values of these parameters. The value abstraction task does *not* reduce the number of parameters reported to the anaesthesiologist. We decompose the value abstraction task into the following subtasks: context acquisition, data filtering, time normalisation, SI-unit normalisation, value derivation and discretisation. We will discuss each of these subtasks in turn.

4.1. Context acquisition

This subtask comprises the acquisition and identification of possibly relevant context data concerning the patient (gender, weight, age, physical condition, possible allergies) and the operation (phases, periods and events). Typical examples for e.g. a bypass operation are the period of anaesthesia, begin and end of the operation itself, the perfusion period (when the blood circulation of the patient is taken over by the heart-lung machine), etc.

4.2. Data validation

This task consists of detecting and (whenever possible) correcting errors in the data. There are many possible causes of errors (often called artifacts) occurring in the data: typing errors and inaccuracies during manual input, artifacts appearing during initialisation and switch-off of equipment, etc. The CAROLA system performs several kinds of data validation when collecting the data (see [8]). For the purpose of the decision support functions described here, additional data validation is restricted to checking that specific data items lie within an appropriately chosen interval: if not, the data item is discarded.

4.3. Time normalisation

The data during an operation are measured at specific time points, and with a particular measuring frequency (see Section 2). These time points and frequencies vary between different parameters: e.g. for blood pressure, the frequency is once a minute, for data from laboratory tests the frequency is much lower (and not constant). However, various subtasks (e.g. value derivation and diagnosis) require the values of several parameters at a particular time point.

In order to cope with this problem, the time-normalisation subtask renormalises all time-dependent measurements by conversion to normalised time-points (e.g. every 10 s). This procedure has two potential disadvantages: firstly, two or more measurements for a particular variable may obtain the same time point; secondly, it may lead to significant changes in the derivative of a parameter (indicating the

degree of change, which is often a relevant notion), caused by rescaling. Both effects can be diminished effectively by increasing the normalisation frequency, but that also increases the amount of data.

There are other approaches to time normalisation: take the most recently available measurement at any point, or compute averages over intervals, etc. Experimentation with a prototype (see Section 7) is needed to assess the different options.

4.4. SI-unit normalisation

Again, to facilitate computation in subsequent tasks, this subtask normalises all quantities involved in the measurements to SI units. This involves mostly straightforward computations for conversion of units and normalisation of dimensions, e.g. in the following expressions:

$$\text{Haematocrite} = 28\% - (37°\text{C} - T_{\text{rect}}) \times 0.8\%°\text{C}$$

$$33\ \text{cm} = 330\ \text{mm}$$

4.5. Value computation

For many relevant variables, the values can be derived from others. To this end, the knowledge base of the system contains formulae describing relations between magnitudes. Some formulae are based on physical laws such as

$$\text{cardiac output} = \text{heart rate} \times \text{stroke volume},$$

others on medical experience, for example

$$\text{required blood flow} = \frac{\text{body surface area} \times \text{rectal temperature}}{30}$$

Observe that these formulae are not yet SI-normalised.

4.6. Discretisation

After the execution of the subtasks described above, the parameter data are in a consistent and reliable numerical format. However, two problems prevent these data from being directly useful to either the anaesthesiologist or the diagnostic component of a decision support system: (1) the volume of the data is far too large (volumes of up to 90 Kb in a period of a few hours, containing many thousands of measurements) and (2) the data are in a quantitative format, while the medical knowledge and experience of the anaesthesiologist is formulated mainly in qualitative terms.

Discretisation is needed to bridge this gap between quantitative measurements and qualitative knowledge. In this final subtask of the value abstraction system, measurement data are translated into qualitative judgements. An example of such a translation:

`PartM=83` translates to: 'mean arterial blood pressure is increased'.

For discretisation, five qualitative categories are being used: *too low*, *decreased*, *normal*, *increased*, and *too high*. The correspondence between qualitative and quantitative values of a particular variable is given by a *discretiser*. Such discretisers are obtained via knowledge acquisition from expert anaesthesiologists or from textbook-values. Two types of discretisers are distinguished: absolute and relative. An absolute discretiser uses absolute values to delimit the five qualitative categories: for arterial blood pressure, these might be 45, 60, 80, and 100 mmHg. A relative discretiser uses percentages of the target value as delimiters, instead of absolute values. For example, relative cut-off points 85, 95, 105, and 115% imply e.g. that any value between 85 and 95% of the target value is considered decreased, and any value above 115% is too high.

The discretiser to be used for a particular variable may depend on other data. Two examples: a systolic blood pressure of 90 mmHg may be high for a 12-year-old while it is low for an adult; a body temperature of 34°C is too low at the beginning of an operation, but too high when the patient is supposed to be cooled down for cardiac surgery.

5. Formalisation

In this section, we present the application of a formalisation method from software engineering to the creation of a anaesthesiological knowledge base.

5.1. Formal descriptions

A formal description is a document written in an artificial language, called a formal language. A language is called formal if it is defined with mathematical precision. Usually this is done with help of a formal grammar, which indicates what symbols (letters, numbers, punctuation marks, names, keywords, mathematical symbols, etc.) are in the language and how these symbols can be combined to form larger expressions. The formal language then consists of all expressions generated by that grammar.

In order to be useful, a formal language should have meaning. Technically, this is provided by the *semantics* of the language, a systematical, precise and unambiguous definition of the meaning of all its possible expressions in mathematical terms. This is work for logicians and theoretical computer scientists; we do not go further into this here.

Just like natural languages (English, Dutch, etc.), formal languages can be used to make descriptions. One may rightfully ask: what is the advantage of a formal description when compared with an informal description in plain English? It is a lot of work to make it, and reading it is not all that easy. The answer is manifold:

- formal descriptions yield precision and unambiguity to a degree which is not attainable with natural languages;
- when properly used, they also yield clarity and systematisation;

- by their very nature, they are closer to computer programs (for programming languages are also formal) so they can be processed with help of software tools for consistency checks, prototyping and implementation.

Well-known examples of formal languages are programming languages. They are designed to write programs, i.e. descriptions of actions that can be performed by a computer. In software engineering, formal specification languages have been developed, with a firm base in logic and discrete mathematics. These languages are intended for the formal specification of a system to be constructed, describing what the system should do, not how it should be done (for that is what a program does). This complies with the sound engineering principle that a specification is indispensable for the successful design of any system of some complexity.

In the specification of a software system, one can distinguish two parts: the static part (specifying the data structures involved) and the dynamic part (specifying the processes that will perform the actions of the system, working on data from the data structures). Up to now, the formalisation effort in FAN has been put mainly in the static aspects of the system to be developed (i.e. the knowledge structure and the knowledge base). We intend to supplement this with a formal specification of the control structures involved in the system.

5.2. The formalisation method

Here we present the method of formalisation, developed by Erik Saaman (University of Groningen, Department of Computing Science). Besides on direct experience in FAN and other projects, the method is based on object-oriented analysis and design principles (e.g. [21]) such as sub-typing and inheritance. The method has three components: guidelines, language, and tools, which we briefly discuss now.

5.2.1. Guidelines

A specification is constructed in three parts: a dictionary, a signature, and an axiomatisation. These three parts represent various aspects of a specification: intention, structure, and content.

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

5.2.1.2. Signature. The first step towards formalisation 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. 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. An example is given below.

5.2.1.3. Axiomatisation. The formalisation is completed by adding formal axioms. The axioms must be such that the properties described in the dictionary and the corresponding informal axioms are satisfied.

5.2.2. Language

This is the formal specification language AFSL (see Section 5.3).

5.2.3. Tools

When working with large specifications (the FAN specification contains 1300 names in ca. 90 modules, in total more than 3000 lines of code), 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 [19].
- An interface with the graph visualisation system da Vinci developed at the University of Bremen (see [7]). It is used to generate diagrams from the signature part of the specification, displaying information on object and function names, their types, and sub-typing relations. Diagrams serve to give a comprehensive view of the ontology embodied in the signature.
- AWK programs and UNIX shell scripts were used to manipulate the large number of files (every module is a separate file), and for the extraction of (LA)TEX and AFSL files from a general file-format (containing both formal specification and documentation).

5.3. The formal specification language AFSL

AFSL (almost formal specification language; see [10]) is an extension of first-order predicate logic. Some of the additional ingredients are: a typing system with subtyping, inheritance and partial functions; a module mechanism with parametrisation; optional use of semi-formal terms. AFSL is designed by Erik Saaman and is inspired on the specification language COLD [4] and on object orientation ([21]).

The basic building blocks of an AFSL specification are parametrised modules. In a module, we can introduce sort, subsort, object and function names, using the keywords `SORT`, `SUBSORT`, `OBJ`, `FUNC`, respectively. These name introductions span the local signature of a module. Signature elements can also be imported from other modules. The signature elements can then be used in axioms which the elements must satisfy. Variables ranging over some sort can be introduced by declarations (keyword `DECL`). These variable are used in existential and universal quantification.

5.3.1. Example (from the knowledge base)

Suppose we have a domain of three sorts `MagnitudeS`, `TimePointS`, and `QuantityS`. We introduce one object in `MagnitudeS`, viz. `PArtM`. For `MagnitudeS` we have a partial function `Value` that yields an element of `QuantityS` for a given timepoint. The application of `Value` is only defined when a magnitude is

measured. To indicate this we have the boolean-valued function Measured. In AFSL. this is written down as follows:

```
SORT MagnitudeS
SORT TimePointS
SORT QuantityS

OBJ PArtM: MagnitudeS

FUNC Value: MagnitudeS, TimePointS- > PARTIAL ~ QuantityS

FUNC Measured: MagnitudeS- > BoolS
```

Subsorts are declared with the $< < <$ symbol. In the example, we could introduce the sort Parameters, of which MagnitudeS is a subsort:

```
SORT ParameterS
SORT MagnitudeS < < < ParameterS
```

We can declare variables and state an axiom:

```
DECL mag: MagnitudeS
DECL tp: TimePointS

AXIOM FORALL mag (EXISTS tp (Value(mag,tp) = / =Undef))
```

5.3.2. Modules and parametrisation

An AFSL construction often used in the FAN specification is the module mechanism with indexed names, which provides a kind of higher-order functions. We illustrate it with a module CourseM with parameter X. It imports the module TimeM, which introduces the sort TimePointS.

```
MODULE CourseM [X]

SORT X
SORT CourseS[X]

IMPORT TimeM

FUNC At: CourseS[X], TimePointS- > X

END MODULE
```

Now the instantiation with QuantityS and BoolS results in the introduction of *two* sorts, each with a function At:

```
SORT CourseS[QuantityS]
SORT CourseS[BoolS]

FUNC At: CourseS[QuantityS], TimePointS- > QuantityS
FUNC At: CourseS[BoolS], TimePointS- > BoolS
```

5.4. Dictionary

The dictionary of FAN is given by Rotterdam in [10], Ch. 6. It provides informal but systematic descriptions of concepts in FAN. An example:

> *Magnitude*
>
> *Description.* A magnitude is a parameter (see parameter) of the patient or of equipment used in the operation that can have a quantitative value. Like with any kind of parameter, courses-of-value (see course-of-value) are associated with a magnitude for each world-of-reasoning (see world-of-reasoning). In the case of magnitudes, these courses are signals (see signal), that is the values of the course are quantities (see quantity).
>
> *Examples.* Examples of magnitudes applying to the patient are heart rate and systolic-bloodpressure. An example of a magnitude applying to the heart-lung-machine (equipment) is the line-pressure; the pressure by which the blood is pumped into the patient.

The dictionary serves as a point of reference for the construction of the knowledge base.

5.5. The structure of the knowledge base

We distinguish three parts of the knowledge base: data-types (representing medical parameters), operations (for arithmetics and abstraction) and relations (representation of causal and has-part links).

5.5.1. Data types

To structure the domain ontology, we identify three classes of sorts: value sets (contain basic values), COVs (courses of value, modeling basic values as a function of time), and parameters (the relevant properties of a patient).

The three value sets are:

```
SORT QuantityS
SORT QualityS
SORT BoolS
```

They yield three COVs, defined as sets of functions from timepoints to some value set:

```
SORT SignalS          === FunctionS[TimePointS, QuantityS]
SORT DiscreteSignalS  === FunctionS[TimePointS, QualityS]
SORT ConditionS       === FunctionS[TimePointS, BoolS]
```

A parameter is a COV with some modality, indicating its status: actual, target (indicating the medically ideal value) or default (indicating a value that can be used if no measurement data is available). `ModalityS` is the sort containing these three modalities.

```
SORT MagnitudeS   === FunctionS[ModalityS, SignalS]
SORT LevelS       === FunctionS[ModalityS, DiscreteSignalS]
SORT PhenomenonS  === FunctionS[ModalityS, ConditionS]
```

So magnitudes represent parameters with numerical values (typically measurement data, for example blood-pressure); levels are used to model discrete parameters (like anaesthesia-depth or ventilation); phenomena model 'yes/no-parameters' (e.g. the presence of high blood pressure). Phenomena form the basis for diagnostic reasoning.

A phenomenon is a condition which can be present or absent for the patient under treatment (e.g. gender, or the presence of a pupil reflex). Several phenomena can be defined in terms of the quantitative or qualitative value of a magnitude. Phenomena applying to the values of multiple magnitudes can be defined with has-part relations by means of which simple phenomena can be combined into more comprehensive ones.

5.5.2. Discretisation

The usual arithmetical operations and order relations are extended in a straightforward way to quantities, signals and magnitudes, with obvious restrictions: e.g. only quantities of the same dimension can be added or compared. Likewise, the logical operations are extended to conditions and phenomena. For qualities, we introduce the following total order:

```
TooLow < Decreased < Normal < Increased < TooHigh
```

This order is extended to discrete signals and levels.

The second type of operations is the discretisation of quantities to qualities. We define the function `Discretise` to perform the discretisation of a quantity according to four quantities that act as bounds:

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

The axiomatisation of this function is straightforward: the first four arguments (which are assumed to be in increasing order) determine five intervals corresponding with the five discrete values, the last argument is the quantity to be discretised. Using `Discretise`, we also defined a relative discretiser `RelDiscretise` which uses a reference quantity and four scaling factors determining the bounds.

`Discretise` and `RelDiscretise` are extended to signals and magnitudes. Observe that this also applies to the first four arguments, so we can apply time-dependent discretisation to signals. This is useful e.g. when discretising measurements of body temperature during heart operations, where artificial lowering of body temperature is applied: a value of 35°C will be considered `Decreased` when the operation starts, but it is far `TooHigh` when cooling has started.

5.5.3. Diagnostic reasoning

Diagnostic knowledge is mainly formulated in terms of relations between phenomena. Two kinds of relations play a role here: causal relations (between causes and effects), and has-part relations (between syndromes or symptom complexes and their constituents). The distinction between these two kinds corresponds with a difference in medical knowledge: in case of a has-part relation, one has to do with a collection of symptoms which is supposed to have a common cause that has not been discovered yet; in case of a causal relation, the underlying disease is known as a specific entity, not merely the collection of symptoms it causes.

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 formalisation of relations there are two options. We can model relations directly as boolean-valued functions, or we can objectify them by introducing special sorts for links. We have chosen for the last option for the following reasons:

- Anaesthesiologists 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 rules.

To model this, we introduce two link sorts as subsort of `PhenomenonS`, the sort `LinkStrengthS` containing three values, and two functions to create links:

```
SORT LinkStrengthS

OBJ Facultative    : LinkStrengthS
OBJ Obligatory     : LinkStrengthS
OBJ Pathognomonic  : LinkStrengthS

IMPORT TotalOrdered[LinkStrengthS]

AXIOM Facultative  < Obligatory
AXIOM Obligatory   < Pathognomonic

SORT CausLinkS     < < <LinkS
SORT NasPartLinkS  < < <LinkS

FUNC Causes: PhenomenonS, PhenomenonS, LinkStrengthS
             -> PARTIAL ~ CausLinkS

FUNC HasPart: PhenomenonS, PhenomenonS, LinkStrengthS
              -> PARTIAL ~ HasPartLinkS
```

Fig. 1 visualises the structure for diagnostic reasoning. The dotted lines indicate the subsort relations between links.

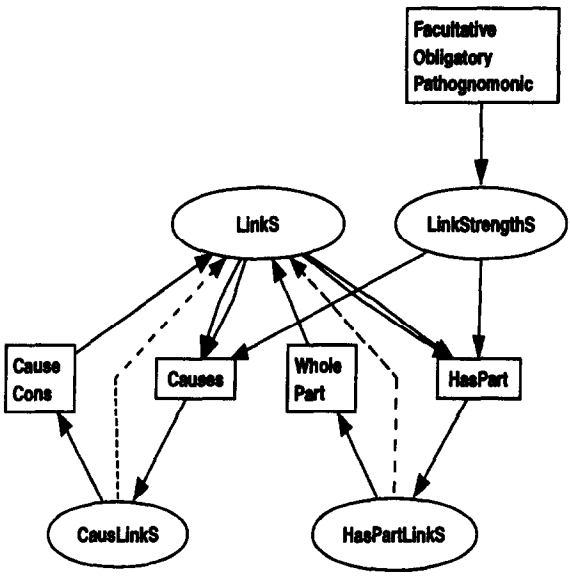

Fig. 1. Signature diagram for diagnostic reasoning.

5.6. *The contents of the knowledge base*

The knowledge base contains the formalisation of knowledge obtained from interviews and knowledge from physiological handbooks. The knowledge is divided in three types: general knowledge, knowledge for value abstraction, and knowledge for diagnosis.

5.6.1. *General knowledge*

The general knowledge is shared by all tasks, i.e. it includes the introduction of all the measured parameters. Besides these, generally applicable rules are formalised, e.g. '*during the perfusion period, disconnection is unacceptable and airway obstruction is undesirable*'. This is represented in the knowledge base as follows:

```
OBJ Disconnection: PhenomenonS
OBJ Airway Obstruction: PhenomenonS

AXIOM Contains(PerfusionPeriod, tp)
  ==> (Valuation Disconnection) At tp = Unacceptable

AXIOM Contains(PerfusionPeriod,tp)
  ==> (Valuation AirwayObstruction) At tp = Undesirable
```

5.6.2. *Discretisation*

For value abstraction, we need to perform operations on magnitudes. They include operations for discretisation and physiological constraints. So the knowledge item '*cardiac output = heart rate × stroke volume*' is formulated in the knowledge base as:

```
OBJ CO: MagnitudeS[VolumeTime]
OBJ HR: MagnitudeS[Frequency]
OBJ SV: MagnitudeS[Volume]

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

5.6.3. Diagnostic reasoning

The diagnostic task needs a specification of a causal network. This network is given by providing a number of causal and has-part links. As an example of the specification of causal relations, we present the formalisation of '*too high mean arterial blood pressure can cause edema*':

```
OBJ PArtM: MagnitudeS[Pressure]
OBJ Edema: PhenomenonS

AXIOM EveryWhere Causes((Discrete(PArtM)==TooHigh)
                        , Edema
                        , Facultative)
```

The knowledge item '*increased mean arterial blood pressure is a symptom of increased stress*' can be formalised as:

```
OBJ Stress: LevelS

AXIOM EveryWhere HasPart(Stress >=Increased
                         , (Discrete(PArtM) >=Increased)
                         , Obligatory)
```

5.7. Formalisation experience in FAN

After the survey of the products of the formalisation process, we give a sketch of the formalisation process itself. This sketch takes the form of a rational reconstruction of the process, along the lines of our formalisation method. We have to admit that this is somewhat misleading in the sense that the actual formalisation did not always proceed as rationally as the method suggests; in fact, the method has been shaped after the experiences in FAN and other projects.

The main sources for anaesthesiological knowledge are interviews and discussions with anaesthesiologists, and the existing CAROLA system. Other relevant sources are e.g. the Système International (physical dimensions and units [14]). These sources were the starting point for formalisation. A first attempt (by Rotterdam) used a special-purpose formal representation language, geared to the anaesthesiological domain. Later on, it was decided to work with the general-purpose formal specification language AFSL, in order to separate language design and language use. AFSL was under construction at that time, a process that was strongly influenced by the experiences in using it in FAN and another formalisation project.

Much effort was consumed by the search for an appropriate ontology for FAN. With the term ontology we refer to the structure of basic concepts with which all relevant objects in a knowledge domain can be described. Examples of basic concepts for FAN are: physical dimensions, units and quantities, time, parameters (relevant medical concepts like blood pressure, temperature), signals, conditions, diagnoses, therapies, etc. In a formal specification, an ontology roughly corresponds with what is called a signature (see Section 5.3).

In the search for a good ontology, this signature has been restructured several times during the project. To have a somewhat stable starting point for the construction of the signature, a *dictionary* has been created, containing informal descriptions of the items in the ontology (this idea was adopted from the complementary formalisation project involving Chomsky's Minimalist Program). In order to visualise the signature (and hence the structure of the ontology), an interface between AFSL and the graph visualisation daVinci [7] has been developed.

The separate presentation of the signature appeared useful during the development of the formal model. In later stages of the development, the signature definition and axiomatisation were combined in the presentation of the formal specification.

The specification of the knowledge base is incorporated in the FAN report [10], which is available via World Wide Web. A survey of FAN from a software engineering perspective is in [9].

5.8. Other approaches in knowledge modelling

Many of the existing knowledge modelling approaches (see [13] for a survey) make a distinction between the static knowledge to be modelled and the reasoning process that is to be performed using that static knowledge. For instance, in the KADS method [22,26], the domain layer is concerned with modelling static knowledge, while the inference and task layer deal with the reasoning process, by specifying respectively the legal inference steps that can be made using the knowledge and the control enforced over these inference steps. Similar distinctions exist in other knowledge modelling methods.

Using the KADS terminology, AFSL is aimed at describing domain knowledge: it does not offer language elements to describe the reasoning process to be performed with this domain knowledge. In this section AFSL was used to specify the domain knowledge formally, while in Section 6 the diagnostic reasoning is described in semiformal mathematical and logical terms. These could be squeezed into an AFSL specification; however, a straightforward and more natural formalisation requires an extension of AFSL with additional language constructs to describe the dynamics of the reasoning process. Steps in this direction are the languages KARL [5], ML^2 [11] and MLPM [6], all based on dynamic logic. (See [12] for a survey.) Compared with AFSL, these languages (with the exception of KARL) provide fewer modeling primitives for domain knowledge, in general only first-order logic.

6. Diagnosis

A conventional definition of a diagnostic problem is a discrepancy between observed and expected behaviour of a system (or e.g. in the case of FAN, a patient). The diagnostic task is to find an explanation of this discrepancy. Such an explanation is a possible diagnosis; a *differential diagnosis* is a collection of possible diagnoses.

In this section, we firstly formulate the diagnostic method that has been developed in FAN, based on interviews with anaesthesiologists and on some of the literature on this subject (e.g. [20]). Then we present a general framework for diagnostic methods (see also [24]), which enables us to compare the method used here to other diagnostic methods described in the literature, which can be seen as instances of this framework.

6.1. FAN diagnosis

The following concepts of the knowledge structure discussed in Section 5 play a role in diagnosis.

Phenomena, a sort containing possible observations (symptoms and signs) and explanations (diseases, syndromes and symptom complexes).

Causal and has-part links between phenomena. These links are pairs of phenomena: the head in the role of cause (or whole, in the case of a has-part link), the tail in the role of effect (or part). Although the two kinds of links are different in nature (causal links have explanatory power, has-part links only have summarising power by relating a syndrome or a symptom complex to its constituting parts), we shall not distinguish between them in our diagnostic method, and refer to them as *links*.

Strength levels of these links, viz. facultative, obligatory, and pathognomonic (ordered from weak to strong); their meaning will become clear below.We add some mathematical notation:

PH (the collection of phenomena)

$PHC = \wp(PH)$ (the collection of sets of phenomena)

$FL \subseteq PH^2$ (the collection of facultative links)

$OL \subseteq PH^2$ (the collection of obligatory links)

$PL \subseteq PH^2$ (the collection of pathognomonic links)

PH is assumed to be closed under negation: $\forall p \in PH(\neg p \in PH)$. We write $dFLo$ for $(d,o) \in FL$, expressing that there is a facultative link between d and o. Analogously for OL and PL.

A reasoning chain is a chain of causal and/or has-part links. Chains of links are lists of links with pairwise corresponding head and tail. In the sequel, we are only interested in the first and last element in a chain. This leads to the transitive closure of a relation, defined as follows for an arbitrary relation R:

$$R^* = \{(x,y) \mid \exists n \geq 1 \exists a_1 \ldots a_n (x = a_1 \wedge y = a_n \wedge a_1 R a_2 R \ldots R a_n)\}$$

We define two operations involving relations: applying a relation to a set

$$R(X) = \{y \mid \exists x \in X(xRy)\}$$

and inverting a relation

$$R^{-1} = \{(y, x) \mid xRy\}$$

The collections of (obligatory/pathognomonic) link chains are defined by:

$$LC := (FL \cup OL \cup PL)^*$$

$$OLC := (OL \cup PL)^*$$

$$PLC := PL^*$$

These definitions reflect the idea that the strength of a chain is the strength of its weakest link.

We describe the diagnostic reasoning method used for FAN in two steps: first we define when a phenomenon complex D is a diagnosis for an observed phenomenon complex O, then we define which of the diagnoses are put into the differential diagnosis. We put

$$\begin{array}{lll} \texttt{diag}: \mathrm{PHC} \times \mathrm{PHC} & & \\ \texttt{diag}\ (D,O) := O \subseteq LC(D) & \& & \text{(completeness)} \\ \qquad\qquad\quad OLC(D) \subseteq O & \& & \text{(O-correctness)} \\ \qquad\qquad\quad PLC^{-1}(O) \subseteq D & & \text{(P-correctness)} \end{array}$$

We paraphrase the three components of this definition.

Completeness: every observation is covered by the diagnosis, i.e. for each observation, there is a link chain from some element of the diagnosis to that observation.

O-correct: all obligatory consequences of any phenomenon in the diagnosis are among the observations.

P-correct: all phenomena that have a pathognomonic link chain towards an observation are in the diagnosis. Now we define the differential diagnosis, using two auxiliary functions.

$$\begin{array}{l} \texttt{difdiag}: PHC \to \wp(PHC) \\ \texttt{difdiag(O)} = \texttt{mincons}\ (\texttt{min}(\texttt{diag}^{-1}\texttt{(D,O)})) \\ \texttt{min, mincons}: \wp(PHC) \to \wp(PHC) \\ \texttt{min}(X) = \{D \in X \mid \forall D' \in X(\|D\| \leq \|D'\|)\} \\ \texttt{mincons}(X) = \{D \in X \mid \forall D' \in X(LC(D) \subseteq LC(D'))\} \end{array}$$

So, `min` selects the diagnoses with minimal number of elements, then `mincons` takes only the diagnoses which are minimal w.r.t. inclusion of consequences.

As an illustration, Fig. 2 gives a schematic example of observations and a diagnosis that matches them according to the given links.

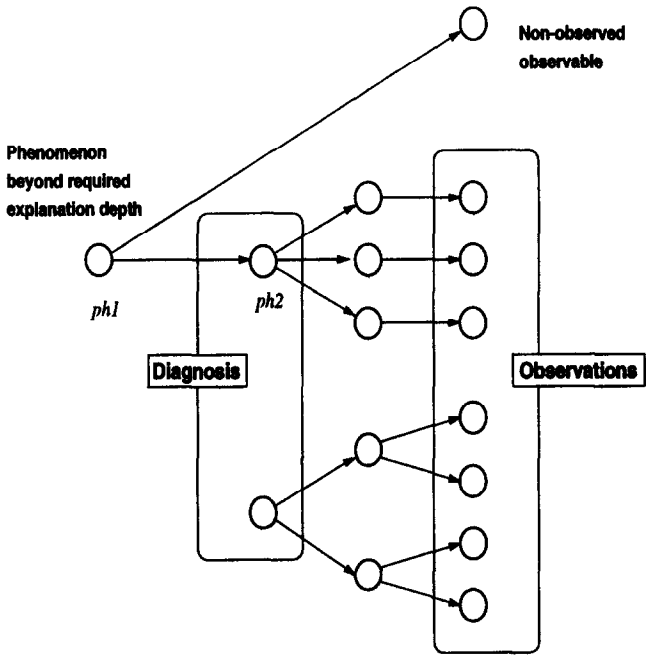

Fig. 2. Observations and a diagnosis that matches them according to the given links.

The diagnosis consists of two phenomena; together they explain the observations via a layer of intermediate phenomena. The five phenomena in the intermediate layer also fulfill the criteria above, so they form a diagnosis, too, which is eliminated by `min`. Replacing *ph2* by *ph1* also yields a diagnosis which will not be eliminated by `min`, but it does get rejected by `mincons`.

6.2. A general diagnostic framework

In [24] a general framework for diagnostic reasoning is proposed (in the sequel, we call it the *framework*). It is a generalisation of the spectrum of diagnosis definitions described by Console and Torasso [3], aiming to include a large variety of existing diagnostic reasoning methods. The framework describes diagnostic reasoning in terms of a number of parameters: a specific diagnostic reasoning method is obtained by instantiating each of the parameters.

We compare the diagnostic method of FAN described above with the framework. We start with a paraphrase of the framework, adapting the notation of [24] to the description in the previous subsection. We use again the sets *PH*, *PHC* and introduce two new sets:

KN (the collection of 'knowledge sets')
SOL (the collection of solutions)

Elements of *KN* are sets of knowledge relevant for the diagnosis, but which does not have to be explained (unlike the observations). Possible components of such a set of knowledge are: information concerning causal and other relations between phenomena, contextual observations that do not have to be explained (e.g. gender and age of the patient), the vocabulary of the explanations that are to be found (e.g. the collection of all possible diagnoses). SOL contains all possible solutions (i.e. presentations of explanations). Now the diagnostic method can be represented as a function:

$$\mathtt{dm}\colon KN \times PHC \to SOL$$
$$\mathtt{dm}(K, O) = \mathtt{sf}(\mathtt{sel}(\mathtt{es}(K, O)))$$

Here

$$\mathtt{sf}\colon \wp(PHC) \to SOL$$
$$\mathtt{sel}\colon \wp(PHC) \to \wp(PHC)$$

act as parameters of the framework. `sf` generates the solution form, when given a solution (we shall not use it here), and `sel` selects the solution, i.e. the subset of preferred explanations, form the set of all explanations. `es`, the function yielding the set of explanations, is defined by

$$\mathtt{es}\colon KN \times PHC \to \wp(PHC)$$
$$\mathtt{es}(K, O) = \{E \in PHC \mid K, E \vdash_1 \mathtt{cov}(O) \quad \& \tag{1}$$
$$K, E \nvdash_1 \bot \quad \& \tag{2}$$
$$K, E \cup \mathtt{con}(O) \nvdash_2 \bot \quad \& \tag{3}$$
$$E \subseteq \mathtt{voc}(K)\} \tag{4}$$

Here we have five other parameters:

$$\mathtt{cov}, \mathtt{con}, \mathtt{voc}\colon PHC \to PHC$$
$$\vdash_1, \vdash_2\colon KN \times PHC \times PHC$$

`cov` and `con` act on the set of observations, yielding phenomenon complexes that have to be *covered* by, respectively to be consistent with the explanations. `voc` yields the vocabulary of the explanations. $\vdash_1$, $\vdash_2$ are (possibly different) entailment relations, stating that the knowledge and phenomena at the left hand side imply the phenomena on the right hand side. $\bot$ is used for expressing contradiction, in the following sense:

$$K, E \vdash \bot \equiv \exists p \in PH(K, E \vdash \{p, \neg p\})$$

This finishes the definition of the diagnostic framework. We observe that it is parametrised by seven parameters:

$$\mathtt{cov}, \mathtt{con}, \vdash_1, \vdash_2, \mathtt{voc}, \mathtt{sel}, \mathtt{sf}$$

6.3. Comparing the diagnostic method with the framework

We compare the diagnostic method of FAN with the framework by trying to find correspondences between components of the two definitions. As a result, we observe the following.

1. `mincons` ∘ `min` (∘ denotes function composition) corresponds with `sel`.
2. `diag` corresponds roughly with `es`:
   (a) the completeness condition corresponds with Eq. (1);
   (b) the O-correctness condition corresponds with Eq. (3).

We consider the correspondence mentioned in 1 to be evident. In order to establish the other two, we need a few instantiations.

$$K := (FL, OL, PL)$$

$$\mathrm{cov}(O) := O$$

$$\mathrm{con}(O) := \{\neg p \mid p \in OBS - O\},$$

where $OBS = \{p, \neg p \mid \exists q\, qLp \wedge \neg\exists r\, pLr\}$

and $L = K_1 \cup K_2 \cup K_3$

$$K, D \vdash_1 Q := Q \subseteq L^*(D)$$

$$K, D \vdash_2 Q := Q \subseteq (K_2 \cup K_3)^*(D)$$

(K_n denotes the nth part of K.) In words: the knowledge relevant for the diagnosis consists of the facultative, obligatory and pathognomonic links; all observations have to be covered; every diagnosis has to be consistent with the negation of absent observations; $\vdash_1$ expresses derivability using all links, $\vdash_2$ idem with restriction to the obligatory and pathognomonic links.

One easily observes that the definition of $\vdash_1$ establishes the correspondence 2a. To see the correspondence of 2b, we argue as follows.

$K, D \cup \mathrm{con(O)} \vdash_2 \bot$

$\equiv$ {definition of K, OLC, con and $\vdash_2 \bot$}

$\neg\exists q \in PH\{q, \neg q\} \subseteq OLC^*(D \cup \{\neg p \mid p \in OBS - O\})$

$\equiv$ {OLC distributes over $\cup$ and leaves subsets of OBS unchanged}

$\neg\exists q \in PH\{q, \neg q\} \subseteq (OLC^*(D) \cup \{\neg p \mid p \in OBS - O\})$

$\equiv$ {$\neg\neg q = q$ and OBS is closed under $\neg$}

$\neg\exists q \in OBS(q \in OLC^*(D)\ \&\ q \notin O)$

$\equiv$ {predicate logic}

$\forall q \in OBS(q \in OLC^*(D) \rightarrow q \in O)$

$\equiv$ {assuming $O \subseteq OBS$}

$OLC^*(D) \subseteq O$

So we do have a partial correspondence between the method and the framework. Both might be extended in order to complete the correspondence. We discuss some possible extensions.

6.3.1. Extending the FAN diagnostic method

Eq. (2) in the definition of es states the consistency of the explanation under consideration. In the context of FAN, this comes down to the absence of contradicting consequences of a diagnosis. Since facultative links are included here, it is questionable whether such a requirement would be appropriate. When facultative links are excluded, consistency of the diagnosis follows from Eq. (2).

Eq. (4) in the definition of es restricts explanations to a specific vocabulary. This is a natural requirement, and we think it should (and will) be added to FAN. As a first try, we might take

$$dom(L) = \{p \in PH \mid \exists q\, pLq\}$$

i.e. the collection of heads of links, as a vocabulary.

sf in the framework definition transforms the solution in a specific presentation, and this is left unspecified in FAN. Case studies with system prototypes will be required to find out appropriate representations of differential diagnoses in FAN.

6.3.2. Extending the diagnostic framework

P-correctness specifies the minimal content of a diagnosis: it must contain all phenomena pathognomonically linked with any observation. There is no directly comparable condition in the framework, although it can be brought into Eq. (1) by adapting the definition of $\vdash_1$. This results in a generalisation of $\vdash_1$ transcending the usual derivability relation.

One might generalise the framework further by replacing the definition of es by:

$$\mathtt{es}(K, O) := \{E \mid R_1(K, O, E)\ \&\ R_2(K, O, E)\}$$

Here R_1, R_2: $KN \times PHC \times PHC$ are two relations satisfying:

$$R_1(K, O, E)\ \&\ E \subseteq E' \Rightarrow R_1(K, O, E')$$

$$R_2(K, O, E)\ \&\ E' \subseteq E \Rightarrow R_1(K, O, E')$$

So R_1 is *monotonic* in its last argument, and R_2 *antimonotonic*.

Eq. (1) is an example of a monotonic condition. Eq. (2) and Eq. (3) are antimonotonic conditions, and so is their conjunction. So the new definition of es above indeed generalises the diagnostic framework.

7. Prototype

Recently, the construction of a prototype for the data abstraction and diagnosis tasks of FAN has been undertaken by Nisaar Jaggoe as part of his Master's thesis work at the Free University in Amsterdam. The main purpose of building the

prototype is the validation of the existing specifications in FAN, especially the adequacy of the knowledge structures and the diagnostic method. A secondary goal is the ability to show a project product that appeals more directly, e.g. to anaesthesiologists, than a collection of specification files. The prototype is restricted to the perfusion phase of the operation (during which the heart lung machine takes care of the blood circulation of the patient) and to the diagnosis of five specific phenomena: too low and too high blood balance, disconnection, centralisation, and stress.

The prototype is a Prolog program that works off-line on real patient data from the CAROLA system. The implementation of the value abstraction tasks makes use of predicate testing (directly available in Prolog) and term rewriting, which is implemented in Prolog for this purpose. For the diagnostic task, both forward and backward chaining are applied.

Although the functionality of the prototype is rather limited (due to the restricted availability of specific formalised anaesthesiological knowledge), the implementation process showed that the construction of a system based on the formal specification is feasible. Validating the output of the prototype against experts (i.e. anaesthesiologists) is in progress.

8. Related research

There is some resemblance between FAN and the YAQ approach [25]. YAQ aims at diagnosis, prediction and therapy management, based on an ontology for modelbased reasoning. Like FAN, it also uses a combination of qualitative and quantitative reasoning, and it implements associative (has-part) and model-based (causal-relations) diagnosis.

Some of the principles behind FAN are shared with [23] where it is conjectured that three major models of diagnosis exist: causal, probabilistic and case-based. The approach taken in FAN globally support the causal and probabilistic model. [23] also stresses that artificial intelligence systems should be regarded as a supporting instrument rather than as a decision making device. This is in accordance with the approach taken in FAN.

The role of first-order logic in medical reasoning is stressed in [17]. Our language AFSL is based on first-order logic, and its module mechanism is used for the definition of re-usable specifications. They can be used in different architectures for support systems [18]. Such reusable components are a prerequisite for widespread adoption of AI systems in medicine. [17] also advocates the use of theorem provers for providing decision support in medicine; the usage of Prolog for building of the FAN prototype demonstrates that this is indeed a viable option.

An aspect of FAN that needs some further study is temporal reasoning, e.g. using temporal models as studied in [1] and [16]. The FAN knowledge-structure deals with time explicitly, but this feature has not been used yet in diagnostic reasoning.

9. Concluding remarks

First experiences with the development of the prototype indicate that the structure of the specification of the knowledge is reasonably adequate. However, on some points (e.g. time normalisation), the specification is not yet complete. Moreover, developing the prototype has contributed to describing and structuring the relevant tasks and subtasks of the decision support system.

Generally speaking, the declarative character of the specification (due to the purely algebraic character of AFSL) leaves the control aspects of the system unspecified. So an extension of AFSL in which these control aspects can be formalised is desirable. Other remaining tasks are the formalisation of therapy advice, simulation and prediction. It is expected that the KADS method [22,26] will be appropriate for developing the task specifications required here.

Interesting research in another direction is the application of data mining techniques to the contents of the CAROLA database. It is reasonable to expect that the outcome of this research will be directly applicable in a decision support system for anaesthesiology.

Acknowledgements

The authors are indebted to Pim Hennis (University Hospital Groningen, Department of Anaesthesiology) for his inspiration, encouragement and support. We also want to thank the anaesthesiologists B. Ballast, G. Massée, J. de Jong and P. Hennis for the opportunity to interview them. Several people of the Department of Computing Science of the University of Groningen are gratefully acknowledged for their help and support: Erik Saaman for developing AFSL and adapting it to our needs; Indra Polak, Jan Jongejan and (again) Erik Saaman for implementing the AFSL tools; Arend van der Knaap for the development of the interface between AFSL and the graph visualisation system 'da Vinci'. Finally we thank Nisaar Jaggoe, student Computer Science at the Free University in Amsterdam, for designing and implementing the prototype system, which enabled us to validate the specification of the knowledge base.

References

[1] P. Barahona, A causal and temporal reasoning model and its use in drug therapy applications, Artif. Intell. Med. 6 (1994) 1–27.

[2] N. Chomsky, Minimalist Program, MIT Press, 1995.

[3] L. Console, P. Torasso, A spectrum of logical definitions of model-based diagnosis, Computational Intell. 7 (1991) 133–141.

[4] L.M.G. Feijs, H.B.M. Jonkers, Formal Specification and Design, Vol. 35, of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1992.

[5] D. Fensel, The Knowledge Acquisition and Representation Language KARL, Kluwer Academic Publisher, Boston, 1995.

[6] D. Fensel, R. Groenboom. MLPM: defining a semantics and axiomatization for specifying the reasoning process of knowledge-based sytems, in: W. Wahlster (Ed.), Proc. 12th European Conf. Artif. Intell. ECAI 96, Budapest, 1996, pp. 423–427.
[7] M. Frohlick, M. Werner. daVinci V1.4 user manual. Department of Computer Science, University of Bremen, 1995.
[8] F. de Geus, E. Rotterdam. Decision Support in Anaesthesia. Ph.D. thesis, Department of Medical Information Science, Groningen, The Netherlands, 1992.
[9] R. Groenboom, E. Saaman, E.P. Rotterdam, G.R. Renardel de Lavalette. Formalizing anaesthesia: a case study in formal specification, in: M.-C. Gaudel, J. Woodcock (Eds.), FME '96: Industrial Benefit and Advances in Formal Methods, Lecture Notes in Computer Science 1051, Springer-Verlag, Berlin, 1996, pp. 120–139.
[10] R. Groenboom, G.R. Renardel de Lavalette (Eds.), FAN: Formalizing ANaesthesia—Report on the FAN-project (preliminary version). Computing Science Notes, CSN 9601. University of Groningen, Department of Computing Science, 1996 [http://www.cs.rug.nl/users/rix/FSA/fanreport.html].
[11] F. van Harmelen, J.R. Balder, $(ML)^2$: a formal language for KADS models of expertise. Knowledge Acquisition 4 (1992).
[12] F. van Harmelen, R. Lopez de Mantaras, J. Malec, J. Treur, Comparing formal specification languages, in: J. Treur, Th. Wetter (Eds.), Formal Specification of Complex Reasoning Systems, Ellis Horwood, Workshop Series 1993, 257–282.
[13] F. van Harmelen, D. Fensel, Formal Methods in Knowledge Engineering, Knowledge Eng. Rev. 10 (1995) 345–360.
[14] Units of Measurements: handbook on international standards for units of measurements, International and Organisation and Standardization, ISO, Geneva, 1979.
[15] G.F. Karliczek, A.F. de Geus, G. Wiersma, S. Oosterhaven, I. Jenkins, CAROLA, a computer system for automatic documentation in anesthesia, Int. J. Clin. Monit. Comput. 4 (1987) 211–221.
[16] E.T. Keravnou, Temporal diagnotistic reasoning based on time-objects, Artif. Intell. Med. 8 (1996) 235–265.
[17] P. Lucas, The representation of medical reasoning models in resolution-based theorem provers, Artif. Intell. Med. 5 (1993) 395–414.
[18] M.A. Musen, A.Th. Schreiber, Architectures for intelligent systems based on reusable components, Artif. Intell. Med. 7 (1995) 189–199.
[19] C. Myers, C. Clack, E. Poon, Programming with Standard ML, Prentice Hall, Englewood Cliffs, 1993.
[20] Y. Peng, J.A. Reggia, Plausibility of diagnostic hypotheses: The nature of simplicity, in: Proc. AAAI-86, AAAI, Philadelphia, 1986, pp. 140–145.
[21] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object Oriented Modeling and Design, Prentice-Hall, Englewood Cliffs, 1991.
[22] G. Schreiber, B. Wielinga, R. de Hoog, H. Akkermans, W. Van de Velde, CommonKADS, a comprehensive methodology for KBS development. IEEE Expert 9, 1994.
[23] B. Spyropoulos, G. Papagnounos, A theoretical approach to artificial intelligence systems in medicine, Artif. Intell. Med. 7 (1995) 455–465.
[24] A. ten Teije, F. van Harmelen, An extended spectrum of logical definitions for diagnostic systems, in: Proc. of DX-94, the 5th Int. Workshop Principles of Diagnosis, New Paltz, New York, 1994.
[25] S. Uckun, B.M. Dawant, D.P. Lindstrom, Model-based diagnosis in intensive care monitoring: The YAQ approach, Artif. Intell. Med. 5 (1993) 31–48.
[26] B.J. Wielinga, A.Th. Schreiber, J.A. Breuker, KADS: A modelling approach to knowledge engineering. Knowledge Acquisition 4 (1992).