

# Characterising approximate problem-solving by partially fulfilled pre- and postconditions

Frank van Harmelen<sup>1</sup> and Annette ten Teije<sup>2</sup>

**Abstract.** In Software Engineering, the functionality of a program is traditionally characterised by pre- and postconditions: if the preconditions are fulfilled then the postconditions are guaranteed to hold, but if the preconditions are not fulfilled, no postconditions are guaranteed at all. In this paper, we study how the functionality of a program is affected when the preconditions are only partially fulfilled. This is particularly important for heuristics AI methods which still function reasonably well (although perhaps suboptimally) under less than ideal preconditions. We introduce a framework for characterising partially fulfilled pre- and postconditions. We also present the proof obligations that must be met when using programs under partially fulfilled preconditions. We show that the classical characterisation of programs can be seen as a special case of our gradual characterisation. We illustrate our framework with two simple diagnostic algorithms which coincide in the classical approach, but which behave differently under gradually relaxed preconditions.

## 1 Motivation

Traditional Software Engineering characterises the functionality of a program by means of pre- and postconditions. As far back as 1969, Hoare [7] characterised a program as a triple  $\phi[\alpha]\psi$ , meaning that if preconditions  $\phi$  hold, then after executing program  $\alpha$ , the postconditions  $\psi$  are guaranteed to hold. In the past few years, research in Knowledge Engineering [6, 2, 1] has tried to apply these traditional ideas from Software Engineering to Knowledge-Base Systems (KBS), characterising the reasoning steps of a KBS through pre- and postconditions. In Knowledge Engineering such reasoning steps are often called a problem-solving method (PSM) and we will adopt this terminology.

Typical of all this work in both Software Engineering and Knowledge Engineering is that the pre- and postconditions are treated as binary yes/no conditions: if  $\phi$  holds, then after executing  $\alpha$ ,  $\psi$  will hold, but if  $\phi$  does not hold, we cannot say anything about either the applicability of  $\alpha$  or the conditions that will hold after  $\alpha$ .

This binary on/off-treatment of pre- and postconditions is problematic for characterising KBS (and AI methods general). In KBS (and in AI in general), we are typically dealing with intractable problems: inference, planning, diagnosis, scheduling, design and learning are all typical AI problems of which even the simple varieties are intractable. Since no efficient program exists that will satisfy the desired pre- and postconditions for such problems, in AI we exploit heuristics

to solve such problems. The use of heuristics turns our programs into methods that approximate the ideal algorithm: our heuristic methods will sometimes fail to find a solution (incompleteness), they may erroneously claim to have found a solution (unsoundness), or they may find only approximations of solutions. The goal of this paper is to try and adapt the traditional pre-postcondition framework to the approximate methods that we study in AI. In particular, we will allow the preconditions to be only *partially fulfilled*, and then try to characterise how much of the postconditions can still be guaranteed. As a result, we depart from the traditional binary yes/no pre-postconditions, and aim for a more gradual treatment.

This paper is quite distinct from work on approximate computational methods such as Horn compilation [10], approximate deduction [9, 4], etc. These are all techniques for specific approximate computational methods. In this paper, we do not study any such single technique, but instead we aim for a general framework in which applications of these techniques can be characterised.

The remainder of this paper is structured as follows: in section 2 we use one particular subfield of AI (model-based diagnosis) to illustrate the fact that such gradually fulfilled preconditions do actually occur in practice. In section 3 we present a simple formal framework to characterise the functionality of programs under such partially fulfilled preconditions. In section 4 we present a greatly simplified version of diagnostic problems that we use in section 5 to illustrate our approach. Section 6 concludes and looks at future work.

## 2 Justification for gradual conditions

The work reported in [5] analyses a number of assumptions that are implicit in the literature on diagnostic problem solving. All of the 27 different assumptions in [5] are presented as binary conditions, but almost all of them can be interpreted gradually, as being more true or less true. We illustrate this with the following examples, all taken from [5]:

**Existence of observations:** this precondition states that all observations required by the diagnostic system must indeed be available. The obvious gradual interpretation of this condition is to require that only some of the required observations are indeed provided, while others are allowed to be unknown.

**Completeness of fault-knowledge:** this precondition states that all possible fault-modes of all components are known to the diagnostic system. Again, the obvious gradual interpretation is that some components are allowed to have incompletely specified fault-modes.

**Completeness/soundness of the diagnosis:** An example of a postcondition is the type of explanation that is computed by a diagnostic method. Some methods compute all components that *must* be faulty, while other methods compute all components that *could* be faulty.

<sup>1</sup> Vrije Universiteit Amsterdam, Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands, frankh@cs.vu.nl

<sup>2</sup> Vrije Universiteit Amsterdam, Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands, annette@cs.vu.nl Supported by the Netherlands Computer Science Research Foundation with financial support from the Netherlands Organisation for Scientific Research (NWO)

Interesting gradual versions of these are to compute as many of the faulty components as possible (but not necessarily all of them), or to compute all components that might be faulty plus perhaps a few spurious ones.

In the next section we will present a simple formal framework for analysing such gradual conditions.

### 3 Formal Framework

In this section, we propose how to deal with partially fulfilled preconditions and their effect on the functionality of the problem solving method (PSM). We extend the standard characterisation of pre- and postconditions by a gradual characterisation.

In the classical pre-postconditions approach, the following elements are necessary to characterise a problem-solving method:

- The **preconditions** on the input of the PSM, which consists of domain-knowledge and data, for which we write  $pre(I)$ .
- The **implementation** of the PSM,<sup>3</sup> which computes an output when applied to the input:  $O := PSM_{op}(I)$ .
- The **functional I/O-relation** of the PSM:  $PSM_{fun}(I, O)$ .

The relation between these elements has the following form in Dynamic Logic [8], adopting the notation from [2]:

$$pre(I) \rightarrow [O := PSM_{op}(I)]PSM_{fun}(I, O) \quad (1)$$

In words: if the preconditions of the PSM hold, the postconditions are guaranteed to hold after execution of the PSM. The postconditions correspond exactly to the functional I/O-relation of the PSM, which is why we write them as  $PSM_{fun}(I, O)$ .

Our approach to a gradual interpretation of these elements is based on three types of knowledge: making the preconditions gradual (Sec. 3.1), making the postconditions gradual (Sec. 3.2), and establishing a relation between these two (Sec. 3.3). We discuss each type in turn and we discuss the proof-obligations that must be met for such a gradual characterisation.

#### 3.1 Gradual preconditions

In order to treat the preconditions  $pre$  as a gradual notion we need three steps:

*Step 1:* determine conceptually a gradual interpretation of the precondition;

*Step 2:* determine a metric for the precondition that expresses the gradual interpretation of the preconditions;

*Step 3:* determine an ordering on the metric, such that the minimal element of the ordering expresses the completely fulfilled precondition.<sup>4</sup> Higher elements in this ordering indicate preconditions that are less fulfilled.

Formally, we refine  $pre(I)$  into  $pre(I, M)$ , where  $M$  indicates the degree of partial fulfillment of the precondition. We define an ordering  $\prec_{pre}$ , and require (i) that for the minimal element  $m_0$  of the ordering the two preconditions are equivalent, and (ii) that higher elements in the ordering corresponds to less fulfilled preconditions. These are the first two proof obligation that a correct gradual characterisation of a PSM must satisfy:

<sup>3</sup> The implementation is also called operationalisation, which is why we write  $PSM_{op}$ .

<sup>4</sup> The minimal element corresponds to the strongest precondition, thus the maximally fulfilled precondition.

#### Proof Obligation 1 (Minimal element preserves preconditions)

$$pre(I, m_0) \leftrightarrow pre(I).$$

This proof obligation ensures that the classical preconditions are indeed a special case of the gradual preconditions.

#### Proof Obligation 2 (Ordering reflects strength of precond's)

$$M_1 \prec_{pre} M_2 \rightarrow (\forall I : pre(I, M_1) \rightarrow pre(I, M_2)).$$

This proof obligation ensures that the ordering on  $M$  correctly reflects the degree of fulfillment of the precondition.

### 3.2 Gradual functionality

We must also specify a gradual version of the functionality of the PSM, because we do not use the method only when the precondition is completely fulfilled, but also when the precondition is partially fulfilled. As indicated by the metric  $M$ . Formally, we refine  $PSM_{fun}(I, O)$  into  $PSM_{fun}(I, O, M)$ .

The third proof obligation is similar to the first. It requires us to show that for the completely fulfilled preconditions ( $m_0$ ) the gradual functionality coincides with the classical functionality:

#### Proof Obligation 3 (Minimal element preserves functionality)

$$pre(I, m_0) \rightarrow (PSM_{fun}(I, O) \leftrightarrow PSM_{fun}(I, O, m_0)).$$

The fourth proof obligation will be to show that the implementation of the PSM does indeed compute this gradual functionality under partially fulfilled preconditions:

#### Proof Obligation 4 (Gradual program correctness)

$$pre(I, M) \rightarrow [O := PSM_{op}(I)]PSM_{fun}(I, O, M),$$

which is the gradual version of the classical proof obligation formula (1).

### 3.3 Relation between preconditions and functionality

Besides characterising a PSM's functionality under given partially fulfilled preconditions, it will be useful to characterise how this functionality changes when the preconditions become more fulfilled. The PSM might compute more output, or less, or completely different output altogether. We formalise this by a second ordering  $\prec_{post}$ . Smaller elements  $M$  under  $\prec_{post}$  correspond to a functionality  $PSM_{fun}(I, O, M)$  that is closer to the classical functionality  $PSM_{fun}(I, O)$ . The final proof obligation is to show that less fulfilled preconditions lead to a greater deviation from the classical functionality.

#### Proof Obligation 5 (Effect of preconditions on functionality)

$$M_1 \prec_{pre} M_2 \wedge pre(I_1, M_1) \wedge pre(I_2, M_2) \rightarrow M_1 \prec_{post} M_2$$

The ordering  $\prec_{post}$  must be defined for particular PSM's in terms of  $PSM_{fun}(I, O, M)$ . An example of this general schema would be

$$M_1 \prec_{post} M_2 \stackrel{def}{=} PSM_{fun}(I, O, M_1) \rightarrow PSM_{fun}(I, O, M_2) \\ M_1 \prec_{pre} M_2 \wedge pre(I, M_1) \wedge pre(I, M_2) \rightarrow M_1 \prec_{post} M_2$$

ie. if the precondition is less fulfilled then more  $(I, O)$  tuples might be computed, but not less. The specific instance of this general schema is the final proof obligation that a gradual characterisation of a method must satisfy.

## 4 A simple description of diagnostic problems

In a diagnostic problem, we are given a number of dependencies between causes and observables, plus a number of values for observables (observables can also be unknown), and we must determine which causes can explain these observed values.

We limit our examples to binary observables (ie. they are present or absent), but all our results can be easily extended to domains with observables that range over more than two possible values (e.g. *color* has value *red*, *white* or *blue*).

We will formalise causes as constants  $c_i$ , observables as constants  $o_j$ , and the fact that  $c_i$  causes  $o_j$  as the term  $cause(c_i, o_j)$ . A set of such terms is given as input:  $cause(c_i, o_j) \in I$ . Similarly, observations will be formalised as terms  $present(o_j)$ , which means that  $o_j$  has been observed as present, while  $absent(o_j)$  means that  $o_j$  has been observed as absent. The input data contains also such terms,  $present(o_j) \in I$ , or  $absent(o_j) \in I$ . If an observation has been done for  $o_j$ , and it did not return “unknown”, then either term is present in  $I$ , abbreviated as  $obs(o_j) \downarrow$ , ie:  $obs(o_j) \downarrow \stackrel{def}{=} (present(o_j) \in I \vee absent(o_j) \in I)$ . If an observation is unknown, we write  $obs(o_j) \uparrow$ , ie.  $obs(o_j) \uparrow \stackrel{def}{=} \neg(obs(o_j) \downarrow)$ . Finally, we will write OBS for the set of all observables  $o_j$  and CAUSES for the set of all possible causes  $c_i$ .

We limit ourselves to single cause solutions, where the observed behaviour is explained by a single cause (the so-called single-fault assumption). A set of such single-class solutions is interpreted as competing alternative solutions.

### Definition 1 (Diagnosis)

A cause  $c_i$  is a diagnosis iff for all observations  $o_j$ : if  $c_i$  is a cause of  $o_j$ , ie.  $cause(c_i, o_j) \in I$  then  $o_j$  has been observed, ie.  $present(o_j) \in I$ .

This definition says that  $c_i$  is only a diagnosis if *all* its relevant observables have been observed, ie. the presence of observations  $o_j$  is *required* for the cause  $c_i$ . The single-fault assumption guarantees that such a  $c_i$  also explains all observed behaviour.

## 5 An Example

In this section we give a concrete example in the context of diagnosis of (1) characterising a gradual precondition of a method, (2) characterising the gradual functionality of that method and (3) proving the correspondence between the graduality of the preconditions and the graduality of the functionality. We first give the traditional (non-gradual) characterisation of two methods, then we give a gradual characterisation and show that all the proof-obligations for this characterisation are met.

### 5.1 Examples of traditional methods

A traditional characterisation of a method is by means of formula (1). We will give such a characterisation for two diagnostic methods: *sceptical* and *credulous*. Both methods have the same functionality  $PSM_{fun}$  and precondition  $pre$ , but of course different implementations  $PSM_{op}$ . Both methods compute all causes that satisfy definition 1:

#### Definition 2 (Functionality of *sceptical* and *credulous*)

$$PSM_{fun}(I, O) \stackrel{def}{=} \\ O = \{c | \forall o : cause(c, o) \in I \rightarrow present(o) \in I\}$$

The precondition for this functionality of both methods is that all the observables  $o$  have an observed value (ie.  $present(o)$  or  $absent(o)$ ):

#### Definition 3 (precondition of *sceptical* and *credulous*)

$$pre(I) \stackrel{def}{=} \forall o \in OBS : present(o) \in I \vee absent(o) \in I$$

The *sceptical*-method starts with an empty list of possible diagnoses, and adds a cause to this list when every relevant observable has the right value. The following defines  $PSM_{op}$  for the *sceptical* method:

#### Definition 4 (Implementation of *sceptical*, $PSM_{op}^{scep}$ )

```

1. O := []
2. for c in CAUSES
3. do solution:=true
4.   for o in OBS
5.     do if cause(c,o) in I and
6.         not present(o) in I
7.         then solution:=false
8.   if solution=true then O:= [c|O]
9. return O

```

The *credulous*-method on the other hand adds a cause to the list of possible diagnoses when no relevant observable is absent.

#### Definition 5 (Implementation of *credulous*, $PSM_{op}^{cred}$ )

The *credulous*-method differs from the *sceptical*-method only in line (6):

```

6.         absent(o) in I

```

Notice that under assumption of the precondition from Def. 3, the conditions used in the methods are equivalent.

## 5.2 Examples of gradual methods

We will now give gradual version of the above definitions.

### Gradual precondition:

The three steps that are needed to define gradual preconditions from section 3.1 are as follows:

*Step 1:* A more gradual interpretation of the precondition that “all observables must have an observed value” (Def. 3) is that we allow some observables to be unknown. For example, we might require that only the observables that are easy to obtain must have a value and the other observables are allowed to be unknown.

*Step 2:* A metric for this gradual precondition is the set of observables for which unknowns are allowed. If we write  $U$  for this set, then the gradual precondition becomes:

#### Definition 6 (Gradual precondition)

$$pre(I, U) \stackrel{def}{=} \forall o \in OBS : obs(o_j) \downarrow \vee o \in U$$

*step 3:* An obvious ordering on the metric  $U$  is:

#### Definition 7 (Ordering)

$$U_1 \prec_{pre} U_2 \stackrel{def}{=} U_1 \subset U_2$$

The minimal element of this ordering is the empty set.

**Proof Obligation 1:**  $\prec_{pre}$  from Def. 7 satisfies the proof obligation that for  $U = \emptyset$  the non-gradual and the gradual preconditions must correspond.

*Proof:* The gradual definition differs from the non-gradual definition (Def. 3) only in the additional disjunct  $o \in U$  which is never true when  $U = \emptyset$ .

**Proof Obligation 2:**  $\prec_{pre}$  from Def. 7 correctly reflects the degree of fulfillment of the preconditions.

*Proof:* If  $pre(I, U_1)$  holds by the first disjunct of Def. 6 then  $pre(I, U_2)$  holds because the same disjunct is true there. If  $pre(I, U_1)$  holds because  $o \in U_1$  then  $o \in U_2$  (because  $U_1 \subset U_2$ ), fulfilling the second disjunct of  $pre(I, U_2)$ .

## Gradual functionality:

In the gradual formulation of the functionality we need to characterise what happens with unknown observables. For this we distinguish two cases: observables inside  $U$  (which are potentially unknown) and observables outside  $U$ . For observables outside  $U$  the functionality remains unchanged, while for observables in  $U$ , we must change the functionality.

In the non-gradual case, all observables had to be known. In the sceptical method, observables in  $U$  are allowed to be unknown, but only if they are not relevant to a solution. In other words, if an observable  $o$  is relevant to a solution  $c$  (ie.  $cause(c, o) \in I$ ), it must be known, even if it is in  $U$ . Of course, it must also be present (as demanded by the definition of diagnosis):

**Definition 8 (Gradual functionality for sceptical)**

$$PSM_{fun}^{scep}(I, O, U) \stackrel{def}{=} \\ O = \{c \mid \forall o : cause(c, o) \in I \rightarrow \\ (o \in U \rightarrow obs(o) \downarrow) \wedge present(o)\}.$$

**Proof Obligation 3:** Definition 8 satisfies the proof-obligation that for  $U = \emptyset$  the gradual functionality coincides with the non-gradual functionality.

*Proof:* The gradual definition differs from the non-gradual definition (Def. 2) only in the additional conjunct concerning observables in  $U$ , which is trivially true when  $U = \emptyset$ .

**Proof Obligation 4:** Definition 8 is a correct characterisation of the algorithm sceptical.

*Proof:* For any  $c \in O$  we must show two things: (i) that any relevant  $o$  is present, and (ii) and that any relevant  $o$  with  $o \in U$  is defined. Point (i) follows because the algorithm uses the tests in lines 5,6 to discard candidate solutions. In other words every solution satisfies the negation of lines 5 to 6, which is equivalent to the required condition that all relevant  $o$  are present. Point (ii) follows because the algorithm demands that  $present(o)$  in  $I$  for all relevant  $o$  (implying  $obs(o) \downarrow$ ), so in particular for those  $o \in U$ .

For the credulous method on the other hand, we do allow observables in  $U$  to be unknown, but if they are known, they must have the right value (ie be present):

**Definition 9 (PSM gradual functionality for credulous)**

$$PSM_{fun}^{cred}(I, O, U) \stackrel{def}{=} \\ O = \{c \mid \forall o : cause(c, o) \in I \rightarrow \\ (o \in U \wedge obs(o) \uparrow) \vee present(o)\}$$

The treatment of unknown observations in  $PSM_{fun}^{cred}$  does indeed differ from  $PSM_{fun}^{scep}$ : the additional condition for unknown observables in  $PSM_{fun}^{cred}$  is exactly the negation of the corresponding condition in  $PSM_{fun}^{scep}$ :  $PSM_{fun}^{cred}$  treats unknown observables as if they were present (their causes can be part of a diagnosis), while  $PSM_{fun}^{scep}$  treats unknown observations as if they were absent (their causes cannot be part of a diagnosis).

By weakening the precondition of  $PSM_{fun}$  (that all observables are known), the sceptical and credulous methods are more often applicable (namely also when some observables are unknown). In such cases, credulous returns those causes that can still fulfill Def. 1 when more observations become known, while sceptical only returns those causes which already fulfill Def. 1, even with the limited set of known observations. As a result, the set of solutions characterised by  $PSM_{fun}^{scep}$  will grow when more observations become known, and will gradually approximate  $PSM_{fun}$ . Similarly, the set of solutions characterised by  $PSM_{fun}^{cred}$  will shrink when more observations become known, again gradually approximating  $PSM_{fun}$ . It is exactly such gradual behaviour of methods that we are interested in capturing in our framework. The final proof obligation (proof obligation 5) exactly captures this behaviour. However, before turning to this final proof obligation, we will first state the other proof obligations for the credulous method:

**Proof Obligation 3:**  $PSM_{fun}^{cred}(I, O, U)$  is equivalent to  $PSM_{fun}^{scep}(I, O)$  when  $U = \emptyset$

*Proof:* The two definitions differ in a condition which is vacuous for the case  $U = \emptyset$ .

**Proof Obligation 4:**  $PSM_{fun}^{cred}(I, O, U)$  is correctly implemented by algorithm credulous.

*Proof:* The credulous algorithm does not consider a cause  $c$  as a solution if one of its observations is absent. If all observations that are relevant for  $c$  are either present or undefined (ie not absent),  $c$  is added to the solution-set, as demanded by  $PSM_{fun}^{cred}(I, O, U)$ .

## Relation between gradual preconditions and functionality

The most important of all our proof obligations is obligation 5, since it states how the functionality of a method gradually changes depending on the gradual fulfillment of the preconditions. For this proof obligation, we must give specific instances of the general schema of formula (5) for  $PSM_{fun}^{scep}$  and  $PSM_{fun}^{cred}$ . We will need an auxiliary predicate  $less-informed(I_1, I_2)$ , which states that  $I_1$  differs from  $I_2$  only by having more unknown observations, ie:  $I_1$  is smaller than  $I_2$ , but they only differ on their observations, not on their causal relations.

$$less-informed(I_1, I_2) \stackrel{def}{=} \\ I_1 \subseteq I_2 \wedge \\ (cause(c, o) \in I_1 \leftrightarrow cause(c, o) \in I_2)$$

We use the following definition of  $\prec_{post}$ :

$$U_1 \prec_{post}^{cred} U_2 \stackrel{def}{=} \rightarrow ( less-informed(I_2, I_1) \wedge \\ PSM_{fun}^{cred}(I_1, C_1, U_1) \wedge \\ PSM_{fun}^{cred}(I_2, C_2, U_2)) \rightarrow C_1 \subseteq C_2$$

This states that higher elements in the  $\prec_{post}^{cred}$  ordering indicate larger solution sets when less observables are actually known. It allows us to prove the following:

**Proof Obligation 5 for  $PSM_{fun}^{cred}$ :**

$$U_1 \prec_{pre} U_2 \wedge pre(I_1, U_1) \wedge pre(I_2, U_2) \rightarrow U_1 \prec_{post}^{cred} U_2$$

*Proof:* Omitted for reasons of space.

The definition of  $\prec_{post}^{scep}$  for  $PSM_{fun}^{scep}$  is exactly the dual of the version for the *credulous*-method: this time, less informed input leads to a *smaller* set of solutions:

$$U_1 \prec_{post}^{scep} U_2 \stackrel{def}{=} ( \text{less-informed}(I_2, I_1) \wedge \\ PSM_{fun}^{scep}(I_1, C_1, U_1) \wedge \\ PSM_{fun}^{scep}(I_2, C_2, U_2) ) \rightarrow C_2 \subseteq C_1$$

This allows us to prove the similar proof obligation for  $PSM_{fun}^{scep}$ :

**Proof Obligation 5 for  $PSM_{fun}^{scep}$ :**

$$U_1 \prec_{pre} U_2 \wedge pre(I_1, U_1) \wedge pre(I_2, U_2) \rightarrow U_1 \prec_{post}^{scep} U_2$$

*Proof:* Omitted for reasons of space.

These two statements show that the two algorithms do indeed have interesting behaviour under gradual preconditions. In particular, an anytime algorithm could be constructed that obtains gradually more observations (ie  $U$  decreases, and the preconditions become stronger). This will cause the *sceptical* algorithm to approximate the final set of solutions (when all observables are known) “from below”, ie by ever larger subsets of this final solution, while *credulous* does the converse “from above”.

## 6 Summary, conclusions and future work

**Summary** In this paper, we have argued that it is useful to study how the functionality of problem-solving methods is affected when the preconditions under which the methods are applied are gradually relaxed.

We introduced a formal framework in which it is possible to characterise both preconditions and functionality of a method as gradual notions, as well as the relation between these two. As part of our framework, we formulated five proof obligations. These obligations ensure that gradual preconditions and gradual functionality converge to the classical notions in the borderline case, they contain a gradual version of the classical correctness-proof, and they capture the relation between changing the preconditions and the corresponding change in functionality. Finally, we illustrated our framework through two simple diagnostic algorithms which coincide in the classical case, but which behave differently under gradually relaxed preconditions concerning the availability of observations.

Although we have illustrated our proposal with examples from diagnostic reasoning, we claim that our results are equally valid for other forms of reasoning in AI, such as configuration, planning, learning, etc.

**Contributions** We claim that our gradual characterisation of PSMs is useful for a number of different purposes:

*Configuration, indexing and re-use of PSMs:* In libraries of problem solving methods, indexing (ie. the problem of finding a method that satisfies a given set of properties) is a significant problem. The classical “yes/no” characterisations of methods are in danger of either returning too large a set of methods (of which many will not actually perform well), or an empty set of methods (if no method precisely satisfies the given goal). Our gradual characterisations of methods can

be used to reduce a large set of methods by gradually strengthening the demands on the functionality. Similarly, they can be used to find a method as close as possible to the intended goal when no precisely satisfying method can be retrieved.

*Verification and Validation:* For purposes of verifying PSMs, it is not sufficient to only prove properties of PSMs, but we also need repair strategies when our PSMs fail to have the required properties. Our gradual characterisations of PSMs can be used for this purpose. A gradual characterisation of  $PSM_{fun}$  tells us precisely how to change the preconditions of the PSM in order to affect the behaviour of the method. A second benefit of our gradual characterisation is that it becomes possible to state properties of PSMs even when not all preconditions have been fully met, whereas the traditional characterisations only tell us what happens when the preconditions hold completely.

**Future Work** In this paper, we have presented only very simple examples to illustrate our approach. We must also illustrate that our approach works for more realistic applications. We believe that the methods for approximate diagnostic reasoning described in [11] is a source of such more realistic applications of gradual methods which can be captured in our framework.

We should investigate how different uniform methods of approximate reasoning (e.g. [4, 9, 10]) can be modelled in our framework. Such approximate methods generally use some parameter to characterise their “degree of precision” (the maximum length of a clause, or a subset of the alphabet to be used, etc). We expect that such parameters are special cases of the general metric on pre- and post-conditions that we proposed.

Experimental work is required to test the claims made above concerning the use of our framework for configuring, indexing and verifying real-life problem solving methods as used in actual Knowledge-Based Systems.

## REFERENCES

- [1] M. Aben, ‘Formally specifying re-usable knowledge model components’, *Knowledge Acquisition*, **5**, 119–141, (1993).
- [2] V.R. Benjamins, D. Fensel, and R. Straatman, ‘Assumptions of problem-solving methods and their role in knowledge engineering’, *ECAI’96*, pp. 408–412.
- [3] L. Console and P. Torasso, ‘A spectrum of logical definitions of model-based diagnosis’, in *Readings in Model-based Diagnosis*, eds., L. Console, J.H. de Kleer, and W.C. Hamscher, Morgan Kaufmann, (1992).
- [4] M. Dalal, ‘Semantics of anytime family of reasoners’, in *ECAI’96*, pp. 360–364.
- [5] D. Fensel and R. Benjamins, ‘Assumptions in model-based diagnosis’, *International Journal of Intelligence Systems*, (1998). To appear.
- [6] D. Fensel and R. Straatman, ‘The essence of problem-solving-methods: Making assumptions for gaining efficiency’, *Journal of Human Computer Studies*, (1998). (to appear).
- [7] C.A.R. Hoare, ‘The axiomatic basis of computer programming’, *Communications of the ACM*, **12**(10), 567–583.
- [8] V.R. Pratt, ‘Semantical considerations on floyd-hoare logic’, in *IEEE Symp. on Foundations of Computer Science*, pp. 109–121.
- [9] M. Schaerf and M. Cadoli, ‘Tractable reasoning via approximation’, *Artificial Intelligence*, **74**(2), 249–310, (April 1995).
- [10] B. Selman and H. Kautz, ‘Knowledge compilation using horn approximations’, *AAAI’91*, pp. 904–909.
- [11] A. ten Teije and F. van Harmelen, ‘Exploiting domain knowledge for approximate diagnosis’, *IJCAI’97*, pp. 454–459.