

Characterising Problem Solving Methods by gradual requirements: overcoming the yes/no distinction

Annette ten Teije^{1,2} and Frank van Harmelen²

¹ Imperial Cancer Research Fund, London WC2A 3PX, UK, annette@cs.vu.nl

² Vrije Universiteit Amsterdam, Dept. of Math and CS, Amsterdam, The Netherlands, frankh@cs.vu.nl

Abstract

This paper makes two contributions to the study of problem solving methods. First, we extend the current theory on how to characterise problem-solving methods. We show that the current theory is not strong enough, and does not allow the characterisation of many natural problem-solving methods.

Secondly, we show that characterisations of problem-solving methods should be stated in gradual terms, and not in terms of yes/no requirements. Such a gradual approach to characterising problem-solving methods allows methods to use domain knowledge that only partially fulfills the requirements, in which case the methods still produce useful, although suboptimal solutions. Problem-solving methods with such gradual behaviour can be applied and re-used in a wider set of circumstances than classically characterised methods.

1 Introduction and Motivation

Problem solving methods (PSMs) have become a central topic in Knowledge Engineering. Together with ontologies they are now the central notion around which the questions of Knowledge Engineering are discussed: construction of knowledge-based systems, and re-use of both knowledge contents and knowledge structure. Problem solving methods describe in a domain-independent way efficient reasoning strategies for specific problem classes. They require specific types of domain knowledge and introduce specific restrictions on the tasks that can be solved by them (Fensel & Straatman, 1996) (Fensel *et al.*, 1997).

Because of the central role that PSMs play in current Knowledge Engineering research, characterising their properties is of major importance. Such properties of PSMs are required for many of the potential uses that can be made of PSMs: *configuration of PSMs*, either by hand or automatically (tenTeije *et al.*, 1998); (Stroulia & Goel, 1997) *selection of PSMs from a library*, also known as indexing (Benjamins, 1994; Motta & Zdrahal, 1998); and *verification and validation of PSMs* (vanHarmelen & tenTeije, 1997; Cornelissen *et al.*, 1997; Marcos *et al.*, 1997)

This paper is concerned exactly with how to characterise PSMs. In the next section (§2), we will review and extend the most prominent existing approach to characterising PSMs. This review will expose an implicit assumption in this work, which we will call the *yes/no-distinction* on requirements. In section 3, we claim that this distinction unnecessarily restricts the characterisations of PSMs. In subsequent sections we will use a simple classification problem as an example to illustrate how the yes/no-distinction can be overcome by giving *gradual* characterisations of properties of a number of simple PSMs for this classification problem. For instance, we show that our gradual approach allows us to use the hierarchical classification method, which was intended for hierarchical knowledge represented in a perfectly balance tree, also in cases where the hierarchical knowledge

can be only represented in a less balanced tree. In practice, we will often find this kind of hierarchical knowledge. Such gradual characterisations of PSMs can be exploited in the configuration, selection and verification of problem solving methods.

2 Current work on characterising PSMs

In this section, we will review the work of Fensel and others on characterising PSMs, since we perceive this approach as the most prominent and promising in the literature so far. We will base ourselves on (Benjamins *et al.*, 1996) (to be called BFS96 from now on).

BFS96 first define the “ideal relation” between the goal-statement and the functionality achieved by a PSM. Secondly, they formulate how domain- and goal-related requirements can be used to bridge the gap between the ideal goal-statement and the actual functionality of a PSM. Thirdly, they describe how domain-related requirements can be exchanged for goal-related requirements (and vice versa).

We will follow the general line of their work, with the following differences: (i) we have a much stronger “ideal relation” between the goal-statement and the PSMs functionality; (ii) this results in an additional type of goal-related requirement. Without this additional type of requirement, many natural problem-solving methods cannot be properly characterised; (iii) we show that the exchange of requirements can be performed in a gradual way, allowing for a much finer-grained characterisation of PSMs.

Basic elements

Three elements are necessary to characterise a problem-solving method:

- The **ontological requirements** on the domain-knowledge and input of the PSM: $ont(D, I)$.
- The **implementation** of the PSM, which computes an output when applied to domain knowledge and input: $O := PSM_{op}(D, I)$.
- The **functional I/O-relation** achieved by the PSM: $PSM_{fun}(D, I, O)$.

The relation between these three elements has the following form in dynamic logic (Harel, 1984):

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

In words: if the domain- and input-requirements of the PSM hold, then the functional I/O-relation is guaranteed to hold after execution of the PSM.

A final concept is the goal relation that must be achieved with the PSM: $goal(D, I, O)$. The ideal situation would be that the functionality exactly coincides with the goal:

$$PSM_{fun}(D, I, O) \leftrightarrow goal(D, I, O). \quad (2)$$

This formula (2) differs from BFS96 where only soundness of PSM_{fun} is required (ie. the \rightarrow direction). Instead, we require both soundness and completeness of the PSM with respect to the ideal goal (in formula (2) above).

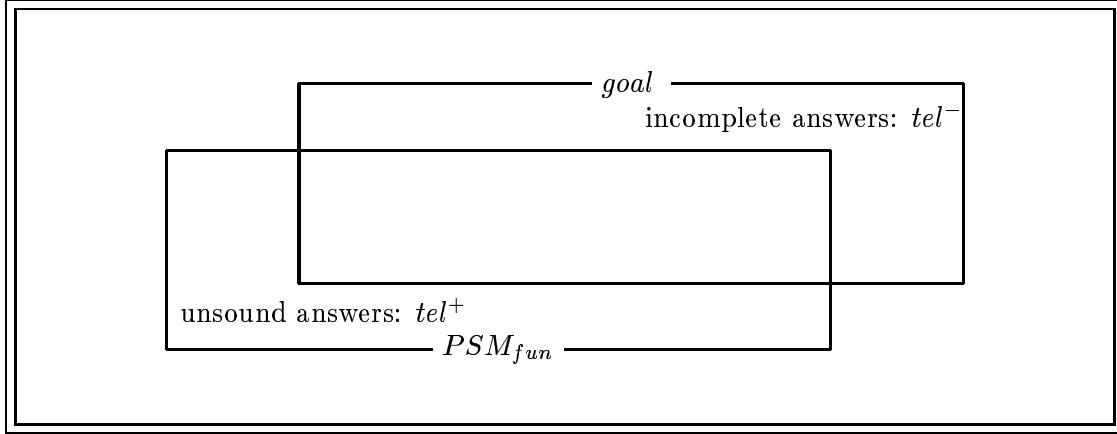


FIGURE 1: The relation between PSM_{fun} , $goal$, tel^+ and tel^-

Adjusting the goal for efficiency

In general Knowledge Engineering deals with goals that cannot be computed in reasonable time, or that cannot be computed at all. Therefore, the above equivalence between functionality and goal is not in general feasible. Instead, the situation is as in figure 1. When a PSM tries to efficiently compute goal, the 's functionality PSM_{fun} is sometimes “larger” than the goal (the PSM gives some unsound answers, tel^+) and sometimes “smaller” (the PSM misses out on others, tel^-). To characterise the relation between goal and functionality, we give names to the areas where they differ:

$$\begin{aligned} tel^-(D, I, O) &\stackrel{def}{=} goal(D, I, O) \wedge \neg PSM_{fun}(D, I, O) \\ tel^+(D, I, O) &\stackrel{def}{=} PSM_{fun}(D, I, O) \wedge \neg goal(D, I, O) \end{aligned} \quad (3)$$

Thus, tel^- represents the incomplete answers: tuples in $goal$ but not in PSM_{fun} , while tel^+ represents the unsound answers: the tuples in PSM_{fun} but not in $goal$. Notice that this definition directly implies that tel^+ and tel^- are disjoint: $\neg(tel^-(D, I, O) \wedge tel^+(D, I, O))$.

Instead of the ideal (2), the actual relation between $goal$ and PSM_{fun} must now be adjusted, and becomes:

$$PSM_{fun}(D, I, O) \leftrightarrow (goal(D, I, O) \wedge \neg tel^-(D, I, O)) \vee tel^+(D, I, O). \quad (4)$$

Rewriting the disjunction to an implication, and substituting this in (1) yields:

$$ont(D, I) \rightarrow [O := PSM_{op}(D, I)] \neg tel^+(D, I, O) \rightarrow (goal(D, I, O) \wedge \neg tel^-(D, I, O)). \quad (5)$$

In other words: after executing the PSM, if we have not computed something outside the goal (tel^+), then we have computed something in the intersection of goal and functionality (since $goal(D, I, O) \wedge \neg tel^-(D, I, O) \leftrightarrow goal(D, I, O) \wedge PSM_{fun}(D, I, O)$).

Both $tel^-(D, I, O)$ and $tel^+(D, I, O)$ characterise the errors that the PSM will make when calculating $goal(D, I, O)$: tel^- characterises the tuples inside goal that we will not compute, and tel^+ characterises the tuples outside goal that we will spuriously compute. Therefore, both tel^- and tel^+ represent concessions that we have to make to the ideal goal-relation when using PSM_{op} to compute it. As a result, both tel^- and tel^+ are called “teleological” (= goal related) requirements we place on the use of our PSM: the PSM is only an acceptable way of efficiently calculating $goal(D, I, O)$ if we are willing to accept the errors it makes, as characterised by tel^- and tel^+ .

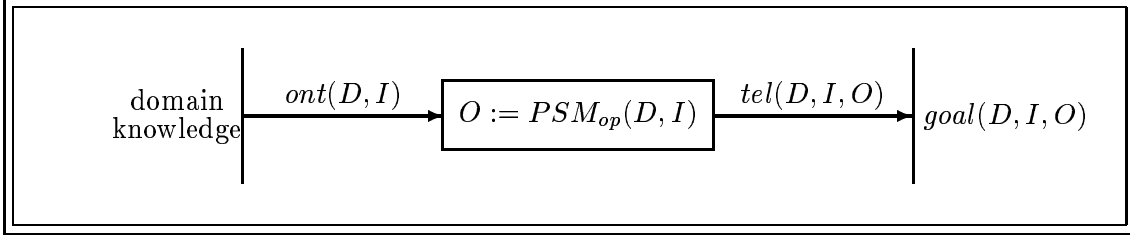


FIGURE 2: Framework for characterising PSMs (Benjamins *et al.*, 1996)

Our formula (5) differs from its counterpart (formula (6) in BFS96). Because of the disjointness of tel^+ and tel^- , our formula (4) is equivalent to

$$(goal(D, I, O) \vee tel^+(D, I, O)) \wedge \neg tel^-(D, I, O).$$

Substituting this in (1) yields an alternative formulation of (5):

$$\begin{aligned} ont(D, I) &\rightarrow [O := PSM_{op}(D, I)] \neg tel^+(D, I, O) \rightarrow (goal(D, I, O)) \\ ont(D, I) &\rightarrow [O := PSM_{op}(D, I)] \neg tel^-(D, I, O) \end{aligned} \quad (6)$$

The first part of this directly corresponds to the formula from BFS96: our $\neg tel^+$ corresponds to their *teleological assumptions*. This accounts for I/O-tuples which are computed by the PSM, but which do not satisfy the goal: any computed I/O-tuples outside tel^+ will satisfy the goal. It therefore deals with the unsoundness of the PSM. The difference with our approach lies in the second formula. This accounts for the part of the goal which is never computed by the PSM: no output of the PSM satisfies tel^- , and since $tel^- \subset goal$, these are parts of the goal that are never computed by the PSM. This characterises the incompleteness of the PSM.

Notice that our formula cannot be reduced by logical manipulations to the implication in BFS96, showing that it is a semantically different characterisation of the teleological requirements, and not simply notational difference. In later sections we will see simple examples where allowing for incompleteness is required to fill the gap between the goal and a PSM, which cannot be captured in the formula from BFS96.

Exchanging requirement

Formula (5) shows the possibility to “exchange” ontological commitments for teleological commitments, or vice versa. BFS96 gives some examples of how the same conceptual notion (for example: only dealing with single-fault diagnoses) can be expressed either as a teleological commitment (“no correct diagnoses need to be computed for multiple-fault cases”), or as an ontological requirement on the domain knowledge (“all multiple-fault combinations are encoded in the domain-knowledge as single-faults with combined symptoms”). Their framework for characterising PSMs is summarised in figure 2. In section 3 we will show how the mechanism proposed by BFS96 can be made much more flexible by gradually exchanging parts of assumptions, instead of exchanging entire assumptions at once.

Other work on characterising PSMs

The reviewed work is certainly not the only literature on characterising PSM. Similar ideas can be found in other work in the KE literature: (Akkermans *et al.*, 1993)’s “competence theory” T_0 , their

refinement theory T_1 and their operational theory T_2 correspond to $goal(D, I, O)$, $PSM_{fun}(D, I, O)$ and $PSM_{op}(I, D)$ respectively.

The work of (Stroulia & Goel, 1997), although less formal than BFS96, is concerned with selecting and configuring PSMs, through matching a PSM’s goal with requirements.

(Benjamins & Pierret-Goldbreich, 1997) identifies similar type of requirements of PSMs as discussed in BFS96, and provides a formalisation for a number of such requirements.

While most work on PSMs aim at a domain-independent description of PSMs (Beys & vanSomeren, 1997) also tries to give task-independent characterisations of PSMs.

The work in (Cornelissen *et al.*, 1997) is based on components of PSMs. A hierarchical compositional structure allows the derivation of requirements of the entire PSM on the basis of the requirements on the components.

Despite the variations among all these approaches, they share a common view on PSMs as characterisable by requirements. In the following section we will point out a common implicit assumption (which we will call the “yes/no distinction”) in all these approaches for which we propose an alternative.

3 Overcoming the yes/no-distinction

Irrespective of the technical differences between the various characterisations of PSMs, they all share a common implicit assumption, which we will call the yes/no-distinction on requirements. By this we mean that requirements are always treated as indivisible objects, which can either be applied in their entirety, or not at all, with no middle ground. This implicit assumption in the current literature on PSMs shows itself in two ways, namely in the application and in the exchange of requirements:

Application of requirements: When applying ontological and teleological requirements, they are considered as either true or false, with no intermediate value. When a given domain fulfills a given ontological requirement, this requirement can be used to enable application of a PSM. But when the domain does not *completely* fulfill the requirement, the PSM is not applicable in its entirety. In other words, there is no way of gradually applying an ontological requirement to a domain (such as: *most* of the knowledge fulfills the given requirement). This means that when a domain even slightly fails to fulfill an ontological requirement, nothing can be said about the resulting PSM’s -functionality. This is particularly unfortunate in a field like Knowledge Engineering, where we are often concerned with partial correctness due the use of heuristic knowledge. It is important to realise that it is in the nature of heuristics that we cannot specify in advance exactly when and how they will fail (if we could, we could also have written better heuristics). As a result, the heuristic nature of many requirements (both ontological and teleological) cannot be captured in the currently proposed frameworks. We will see an example of this in section 5.

Exchange of requirements: When “exchanging” a requirement from ontological to teleological or vice versa, the requirement is exchanged in its entirety, but never partially. In the single-fault example mentioned above, the requirement is either treated as a ontological requirement, or as a teleological requirement, but not as a mixture of both. In other words, there is no gradual transition

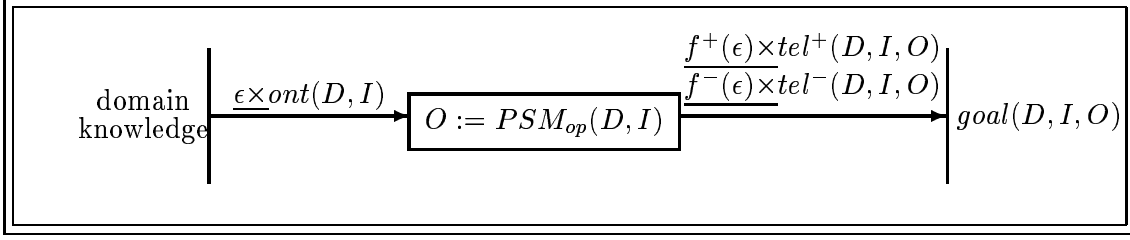


FIGURE 3: New framework for characterising PSMs

from ontological to teleological requirements. This is unfortunate, since when a domain partially fulfills an ontological requirement, we would like to use this partial fulfillment to make the PSM applicable where possible. The remaining cases could then be taken as a teleological assumption, thereby excusing the PSM from the obligation to compute the desired goal whenever the ontological requirement did not hold. Using again the single-fault example, this would roughly mean that the system could still deal with those multiple-faults for which a domain-encoding as a single-fault was present (the partial ontological requirement), and it would not need to compute the required functionality for any other multiple-fault cases (the remaining teleological requirement).

We want to stress that this implicit assumption on the yes/no-nature of requirements in the existing literature is not simply an artifact of the use of classical two-valued logic in the formalisation. We claim that no conceptual notion of a gradually applicable requirements has been studied until now.

In the remaining sections, we will illustrate that it is not at all necessary to assume a yes/no-nature of requirements. We will show how a number of natural requirements on classification methods can be formulated in a gradual way, allowing both the gradual application and the gradual exchange of requirements. Furthermore, we will show how the degree to which a requirement is applicable can be made precise. This will allow us to present precise formula to characterise “how much” of a requirement is applicable. This precise “degree of applicability” can then be used to apply the ontological version of the requirement as much as possible (thereby strengthening the PSMs functionality as much as possible), while moving the “remaining” inapplicable portion of the requirement to the teleological category, (thereby weakening the required goal as little as necessary).

Our proposed extension of the existing framework is paraphrased graphically in figure 3, where ϵ symbolises the degree to which an ontological requirement is applicable, and $f^+(\epsilon)$ and $f^-(\epsilon)$ the degrees to which the corresponding teleological requirement must still be applied. The functions f^+ and f^- symbolise the precise relation that can be formulated on how much of a teleological requirement can be relaxed when an ontological requirement is strengthened (or vice versa).

4 A simple description of classification problems

In a classification problem, we are given a number of values of observable attributes of an individual, and we must determine to which class the individual belongs. In our example we will use the class-attribute relations from figure 4. The interpretation of the table is that a + at position (c_i, a_j) in the table indicates that any member of the class c_i has value “yes” for the attribute a_j . We allow attributes to be unknown. Such an unknown value for an attribute does not rule out the corresponding class.

We limit our examples to binary attributes (using values “yes” and “no”), but all our results also

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}
c_1	+		+				+												+
c_2	+		+					+							+				+
c_3	+			+					+						+	+			+
c_4	+			+						+					+	+	+		
c_5		+			+						+				+	+	+		
c_6		+			+							+			+	+	+	+	
c_7		+					+						+		+	+	+	+	
c_8		+					+							+	+	+	+	+	

FIGURE 4: Example relation between attributes and classes

hold for domains with attributes that range over more than two possible values (e.g attribute *color* has value *red*, *white* or *blue*).

We will formalise classes as constants c_i , and attributes as constants a_j . A + in figure 4 is formalised as the formula $val(c_i, a_j) = +$. In other words, the domain knowledge is the conjunction of all of these equalities: $\bigwedge_{i,j} val(c_i, a_j) = X_{ij}$. When the table entry (c_i, a_j) is empty, no equality is present. Observations will be formalised as equalities: $obs(a_j) = +$ means that a_j has been observed as present; $obs(a_j) = -$ means that a_j has been observed as absent (ie, input data are a conjunction of such equalities). If no observation has been done for a_j , or when an observation returned “unknown”, then the equality is not present in the input.

The classification of an individual as belonging to class c_i is formalised by the predicate $solution(c_i)$. Formalisation of assigning the individual to a class by the predicate $solution(c_i)$. We limit ourselves to single-class solutions, where an individual is assigned to at most one class. (Composite solutions would be written as a conjunction of $solution(c_i)$). Notice that it is still possible to have multiple single-class solutions as competing alternative solutions (written as a disjunction of $solution(c_i)$'s).

We will deal with two variations of classification:

Definition 1 (Strong classification)

A class c_i is a solution (i.e. $solution(c_i)$ holds) iff for all attributes a_j : if $val(c_i, a_j) = X_{ij}$ then $obs(a_j) = X_{ij}$ has been observed.

This definition demands that all required attributes must have been observed. This means that the value X_{ij} for attribute a_j is *required* for the class c_i . If a_j is known but has the wrong value, or a_j is unknown, then c_i is not a solution.

Definition 2 (Weak classification)

A class c_i is a solution (i.e. $solution(c_i)$ holds) iff for all observed attributes a_j : if $val(c_i, a_j) = X_{ij}$ then $obs(a_j) = X_{ij}$ has been observed.

This requires only that all observed attributes have the right value, which means that an unknown value for an attribute a_j is allowed, while c_i is still a possible solution.

Notice that (2) is weaker than (1) because (2) limits its demands to known attribute-values.

5 Example 1: From weak to strong classification

We will show how a method can change its functionality gradually from strong classification (1) to weak classification (2), by making ever stronger domain requirements.

We assume two kinds of table-entries: those table-entries that are *necessary* for a class, and those that are *possible* for a class. We will call these POS and NEC attributes respectively. These two types of table-entries will be used in the following way: A class is a solution if all its NEC table-entries are known and correct (ie. these entries are *necessary* before a class can become a solution), while the POS table-entries can be unknown, but if they are known, their value must be correct (ie. these entries are *possible* for a class, but not necessary).

When a method interprets all the table entries as NEC, and none as POS), it will achieve formula (1). In order to achieve (2) it would have to interpret all table entries as POS. Any position in between (1) and (2) is possible by interpreting some attribute-value pairs as POS and others as NEC. The corresponding functionality consists of a combination of formula (2) for all POS table-entries and formula (1) for all NEC table-entries. This gradual transition from (1) to (2) will lead to an increase in possible answers for any given set of observations.

A practical use of such a mixed functionality would be to interpret the class-specific attributes $a_7 - a_{14}$ as NEC for their corresponding classes, while interpreting all other table-entries as POS.

We can relate this example to the notions discussed in the previous sections as follows:

- **goal:** find classes that satisfy weak classification (Def. (2)).
- **PSM_{fun} :** find classes that satisfy (1) for all NEC table-entries, and (2) for all POS entries.
- **ontological requirement:** make the set of the POS entries in the table as large as possible
- **teleological requirement:** accept the loss of some classes in the answer-set because for these classes the (1) is applied instead of the required (and weaker) (2). (Such an answer lies inside the goal-statement but outside PSM_{fun} , making this teleological requirement of type tel^-)
- **exchange rate:** Strengthening the ontological requirement (by increasing the set of POS entries in the table) allows us to weaken the teleological requirement.

Since this example deals with reducing tel^- , it aims at reducing the incomplete answers produced by the PSM. (ie. such an answer lies inside the goal-statement but outside PSM_{fun}) Therefore this simple example already cannot be captured in the framework of BFS96.

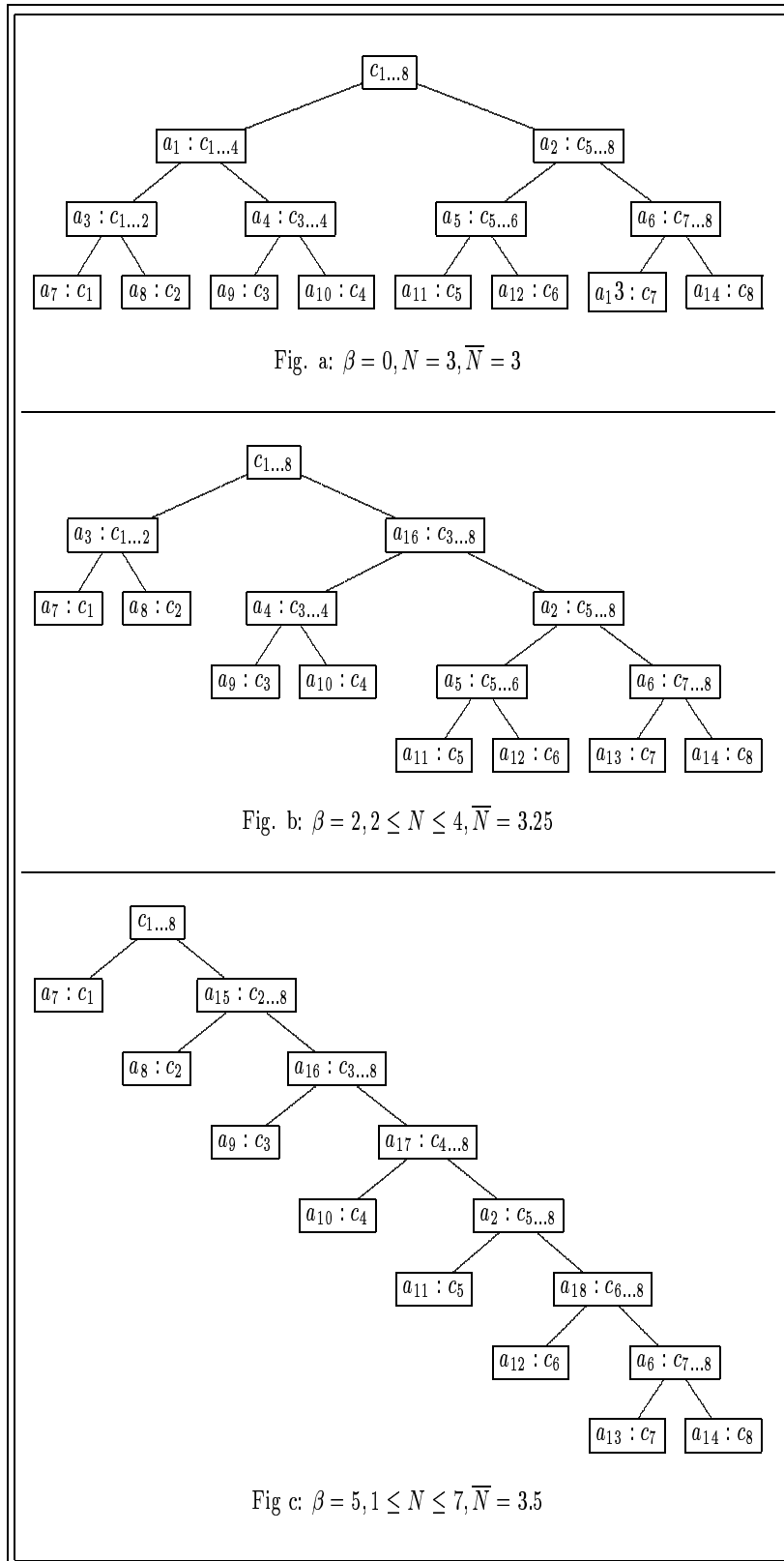


FIGURE 5: Differently balanced trees for the classification problem from figure 4

6 Example 2: Minimal number of required observations

In this example we will use a method that exploits the hierarchical structure of the classes in table 4. This hierarchy is depicted in figure 5-a. In this figure, the notation $c_{k...l}$ stands for the disjunction of classes $c_k \vee \dots \vee c_l$, indicating that these are classes sharing a set of attributes. In general, such a hierarchical structure among the classes is domain specific, and will be uncovered as a result of modelling the corresponding domain knowledge. Obviously, the hierarchical relations among $c_1 - c_8$ are a stylised example, but our results also holds for less idealised hierarchies.

In contrast to the previous example, in this example we will exploit particular properties of a specific method, while the example from the previous section was only concerned with the functional characterisation, and not with the operational description. The following sketches a simple classification method that exploits the hierarchical structures in figure 5 (the notation $a_j : c_k$ in figure 5 means that observing attribute a_j is required at node c_k):

```

P := top
repeat
  C := child of P
  observe attribute at C
  if observation = table-entry
    then P := C
    else P := sibling(C)
until P is a leaf
return P

```

(Notice that this method assumes definite answers to any observation. We assume that the answer unknown is never given). The following trace illustrates this method in figure 5-a.

```

0: P= $c_{1...8}$ ;
1: C= $c_{1...4}$ ; observe attribute  $a_1$ ; (observed value = +); P= $c_{1...4}$ ;
2: C= $c_{1...2}$ ; observe attribute  $a_3$ ; (observed value = -); P= $c_{3...4}$ ;
3: C= $c_3$ ; observe  $a_9$ ; (observed value = +); P= $c_3$ ; return  $c_3$ 

```

This method is only correct under the following domain requirements.

1. Subsumption of children by parents:
 $solution(c_{k...l}) \vee solution(c_{l+1}...m) \rightarrow solution(c_{k...m})$.
 This justifies not further investigating the children when the attribute at the parent is incorrect. In line 2 of the above trace, when $c_{1...2}$ is ruled out, avoid investigating $c_{1...2}$'s children.
2. Exhaustivity of siblings:
 $solution(c_{k...m}) \rightarrow solution(c_{k...l}) \vee solution(c_{l+1}...m)$.
 This justifies not asking the attribute at a node when the attribute at the sibling was incorrect. Again in line 2, when $c_{1...2}$ is ruled out, immediately investigate the children of $c_{3...4}$ without asking $c_{3...4}$'s attribute.
3. Exclusivity of siblings:
 $\neg(solution(c_{k...l}) \wedge solution(c_{l+1}...m))$.
 This justifies investigating at most one of every two siblings. In line 3, when c_3 has been found as a solution, do not investigate alternatives.

Together, these requirements justify a reduction in the number of questions that asked by the method. Incidentally, they also imply the existence of precisely one solution.

The number of questions that a method needs to ask during the classification process can be incorporated as part of the goal-statement, e.g. that it should be less than a given number, or that it should be as low as possible, or that it should be as low as possible on the average, etc. The degree to which such a goal is obtained depends on the quality of the domain knowledge. In particular it depends on how well balanced the tree is. The balance of the tree is a quality measure of the domain knowledge. The idea of a balanced tree is that the children are equally distributed in the left and the right subtree. A tree is balanced if the right and the left subtree differ at most 1 in the number of their children. A tree becomes more unbalanced when the differences between the number of children in the left and right subtrees increases. The following function β is a measure of how unbalanced a tree is (ie. a higher number indicates less ideal domain knowledge):

$$unbalance(node) = \begin{cases} 1 & \text{if there exist children } c_1, c_2 \text{ of } node \text{ with} \\ & leaves(c_1) - leaves(c_2) > 1 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

$$\beta(tree) = \sum_{node \in tree} unbalance(node)$$

Figure 5 shows three trees with increasing degrees of unbalance, where the optimally balanced three has value $\beta = 0$. This measure indicates the strength of the domain requirement that a tree must be balanced. The relation between the strength of this domain requirement and the actual number of questions required is as follows (n is number of classes (in our example $n = 8$), b is the branching factor of the tree (ie. the number of values per attribute, in our example $b = 2$), N is the number of questions required, and \bar{N} is the average number of questions required, assuming equal a priori probability of all classes).

	β (decreases)	minimum N (increases)	maximum N (decreases)	\bar{N} (decreases)
max. unbalanced, b=2:	$n - 3$	1	n-1	$1/2(n - 1)$
optimally balanced:	0	${}^b\log(n)$	${}^b\log(n)$	${}^b\log(n)$

The relation between the theoretical notions from the section 2 and this example is as follows:

- **goal:** find classes that satisfy strong classification (Def. (1)) with the lowest upper-bound on the required number of observations (ie. achieve the lowest maximum of N).
- **ontological requirements:** a classification tree with the lowest value for the β -measure that can be achieved in the domain.
- **teleological requirement:** accept a larger number of observations than the theoretically required lowest upper-bound (${}^b\log(n)$).
- **exchange rate:** Strengthening the ontological requirement (by making the tree more balanced (ie. decreasing the β -measure) will increase the number of classes for which the goal can be obtained (namely classification with at most ${}^b\log(n)$ observations). This will result in a reduction of both tel^+ and tel^- . For example, in the unbalanced tree in figure 5-b, the functionality “establish c_5 in 4 questions” is inside PSM_{fun} but outside the goal (since $4 > {}^b\log(n) = 3$), and therefore it is in tel^+). Correspondingly, the goal statement “establish c_5 in 3 questions” is outside PSM_{fun} , and therefore in tel^- . Increasing the ontological

requirement moves both the first statement outside PSM_{fun} and the second statement into PSM_{fun} , thereby increasing the intersection of PSM_{fun} and $goal$.

Anytime behaviour

This example can also be used to illustrate another gradual property of this hierarchical classification method, namely attractive anytime behaviour. An algorithm is called “anytime” if it can be interrupted at any time and then return a useful partial result, with ever later interrupts returning ever better results. In our example, the value of the variable P can be interpreted as the set of classes that are still candidates for a solution: $class_{i\dots j} = class_i \vee \dots \vee class_j$. During the descent of the abstraction hierarchy, this candidate set is gradually decreased, until a single candidate remains. When the abstraction hierarchy is better balanced (has a lower value for β), this reduction of the candidate class happens faster. The average reduction per observation is by a factor $1/b$ in a balanced tree (ie. exponential reduction of the candidate set by $1/b^k$ after k questions) while in the least balanced tree 5-c, the decrease of the candidate set is only linear (reduction by k candidates after k questions). In this case, the ontological requirement is again the balance of the abstraction-hierarchy, and the teleological requirement is the amount of spurious candidates still in the candidate set.

7 Summary and conclusions

In the ideal situation, we would like the functionality of a problem-solving method to be both sound and complete with respect to the intended goal (as well as of course providing an efficient operationalisation of this functionality). However, for most problems tackled in Knowledge Engineering, this ideal situation is not achievable because of their complexity properties. As a result, most problem-solving methods will be either unsound or incomplete with respect to the goal-statement. In section 2 we derived a formal characterisation of PSMs which is an extension of existing characterisations. Our characterisations models both possibilities: unsoundness (ie. incorrect answers) corresponds to a non-empty relation tel^+ , while incompleteness (ie. missing answers) corresponds to a non -empty tel^- in formula (5).

As pointed out in the literature, there are two ways of bridging the gap between the functionality of a PSM and the intended goal: either by increasing the functionality of the PSM by adding stronger domain-knowledge (corresponding to increasing the ontological assumptions ont), or stating that the unintended or unachieved answers are acceptable (corresponding to increasing the teleological assumptions tel^+ and tel^- respectively).

In the literature it has been pointed out that the ontological and teleological requirements can be interchanged, in the sense that strengthening one category allows weakening of the other, and vice versa.

We have pointed out that both ontological and teleological requirements can often be phrased in gradual terms instead of yes/no terms. We can often give a precise characterisation of the exchange relation between the different types of requirements. This allows us to deal with PSM that partially fulfill certain requirements. In other words, we are not restricted to reasoning about PSMs that do or don't fulfill a given requirement, but we can reason about the degree to which a PSM fulfill the requirement.

We have illustrated these ideas using simple classification PSMs whose functionality increased gradually depending on the gradual increase of the strength of the attribute-class relation, and on the gradual improvement of the quality of a classification hierarchy.

We claim that such a gradual characterisation of PSMs is useful for a number of different purposes:

Configuration, indexing and re-use of PSMs: Until now, existing libraries of PSMs have contained only small numbers of complete methods. However, when making libraries of PSM-components instead of entire ready-made PSMs ((Benjamins, 1994) (Motta & Zdrahal, 1998)), large numbers of PSMs can be constructed by combining such components in many ways. In such libraries, indexing (ie. the problem of finding a combination of components that satisfies a given set of properties) becomes a significant problem. Our gradual characterisations of PSMs allow much finer-grained indexing than the binary “yes/no” characterisations that have been provided so far. Such coarse “yes/no” characterisations 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 combination of components can be found that precisely satisfies the given goal). Our gradual characterisations of methods (which might also be applied to components of methods) can be used to reduce a large set of methods by gradually strengthening the demands on the requirements. 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, we need not only prove properties of PSM, but we also need repair strategies when our PSMs fail to have the required properties. Our precise characterisation of the exchange relation between requirements can be used exactly for this purpose. For example, when a classification method turns out to require too many data, section 6 tells us precisely how to strengthen the domain knowledge, and also the effects on 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 requirements have been fully met, whereas the traditional characterisations from Software Engineering (in the form of formula (1)) only tell us what happens when the preconditions hold completely.

Further work

Much future work can be derived from our proposal for more gradual characterisations of PSM. More examples from other families of methods should be identified, and more realistic knowledge from actual domains should be used, to see if practically useful properties can be captured in this way. Our work on gradual characterisations of diagnosis (tenTeije & vanHarmelen, 1997) already points in that direction. Another way to verify our proposal on granularity is to take a number of KBSs and to re-engineer these applications in the form of our proposal for gradual characterisations of PSM.

Secondly, following the trail blazed by (Fensel & Schoenegge, 1997) we should try to give machine assisted formal proofs of such gradual characterisations of PSMs. Thirdly, our characterisations should be tested in the context of large libraries of methods as described in (Motta & Zdrahal, 1998) and (Benjamins, 1994). A final, but much more theoretically difficult point is to find a good way to treat dynamic properties of PSMs. In section 6 we were characterising the number of observations that a PSM required during its problem solving, by encoding this dynamic property as

part of PSM_{fun} and the *goal* statement. For other dynamic properties (such as anytime behaviour of PSMs) such an encoding would be much more difficult, and a more explicit treatment of dynamic properties (as opposed to the functional I/O-relation) would be needed.

Acknowledgement

We are grateful to Dieter Fensel (Univ. of Karlsruhe) and Remco Straatman (Univ. of Amsterdam) for their critical, helpful and encouraging remarks. Annette ten Teije is supported by a TMR project financed by the European Commission. Project: ERBFMBICT961130.

References

- J. M. Akkermans, B. J. Wielinga, and A. Th. Schreiber. Steps in constructing problem-solving methods. In N. Aussenac, G. Boy, B. Gaines, M. Linster, J.-G. Ganascia, and Y. Kodratoff, editors, *Knowledge Acquisition for Knowledge-Based Systems. Proceedings of the 7th European Workshop EKAW'93, Toulouse and Caylus, France*, number 723 in Lecture Notes in Computer Science, pages 45–65, Berlin Heidelberg, Germany, September 1993. Springer-Verlag.
- R. Benjamins and C. Pierret-Goldbreich. Assumptions of problem-solving methods. In E. Plaza and R. Benjamins, editors, *10th European Knowledge Acquisition Workshop, EKAW-97*, number 1076 in Lecture Notes in Artificial Intelligence, pages 1–16, Berlin, Germany, 1997. Springer-Verlag.
- V. R. Benjamins. On a role of problem solving methods in knowledge acquisition – experiments with diagnostic strategies –. In L. Steels, A. T. Schreiber, and W. van de Velde, editors, *Lecture Notes in Artificial Intelligence, 867, 8th European Knowledge Acquisition Workshop, EKAW-94*, pages 137–157, Berlin, Germany, 1994. Springer-Verlag.
- V.R. Benjamins, D. Fensel, and R. Straatman. Assumptions of problem-solving methods and their role in knowledge engineering. In *Proceedings of the European Conference on Artificial Intelligence (ECAI-96)*, pages 408–412, Amsterdam, August 1996.
- P. Beys and M. van Someren. A systematic approach to the functionality of problem-solving methods. In E. Plaza and R. Benjamins, editors, *10th European Knowledge Acquisition Workshop, EKAW-97*, number 1319 in Lecture Notes in Artificial Intelligence, pages 17–32, Berlin, Germany, 1997. Springer-Verlag.
- F. Cornelissen, C. Jonker, and J. Treur. Compositional verification of knowledge-based systems: a case study for diagnostic reasoning. In E. Plaza and R. Benjamins, editors, *10th European Knowledge Acquisition Workshop, EKAW-97*, number 1319 in Lecture Notes in Artificial Intelligence, pages 65–80, Berlin, Germany, 1997. Springer-Verlag.
- D. Fensel and A. Schoenegge. Using kiv to specify and verify architectures of knowledge-based systems. In *Twelfth IEEE International Conference on Automated Software Engineering (ASEC'97)*, Nevada, November 1997.
- D. Fensel and R. Straatman. The essence of problem-solving methods: Making assumptions for efficiency reasons. In *9th European Knowledge Acquisition Workshop EKAW-96*, Lecture Notes in Artificial Intelligence (LNAI 1076), Nottingham, England, May 1996. Springer-Verlag.

- D. Fensel, E. Motta, S. Decker, and Z. Zdrahal. Using ontologies for defining tasks, problem-solving methods and their mapping. In E. Plaza and R. Benjamins, editors, *10th European Knowledge Acquisition Workshop, EKAW-97*, number 1319 in Lecture Notes in Artificial Intelligence, pages 113–128, Berlin, Germany, 1997. Springer-Verlag.
- D. Harel. Dynamic logic. In D. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic, Vol. II: extensions of Classical Logic*, pages 497–604. Reidel, Dordrecht, The Netherlands, 1984.
- M. Marcos, S. Moisan, and A. del Pobil. A model-based approach to the verification of program supervision systems. In *Proceedings of the Fourth European Symposium on the Validation and Verification of Knowledge Based Systems (EUROVAV'97)*, pages 231–241, Leuven, Belgium, June 1997. Katholieke Universiteit Leuven.
- E. Motta and Z. Zdrahal. An approach to the organization of a library of a library of problem solving methods which integrates the search paradigm with task and method ontologies. *International Journal of Human-Computer Studies, special issue on problem-solving methods*, 1998. Submitted.
- E. Stroulia and A. Goel. Redesigning a problem-solver's operators to improve solution quality. In M.E. Pollack, editor, *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, volume 1, pages 562–567, Nagoya, Japan, August 1997. Morgan Kaufmann.
- A. ten Teije and F. van Harmelen. Exploiting domain knowledge for approximate diagnosis. In M.E. Pollack, editor, *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, volume 1, pages 454–459, Nagoya, Japan, August 1997. Morgan Kaufmann.
- A. ten Teije, F. van Harmelen, A. Th. Schreiber, and B. J. Wielinga. Construction of problem-solving methods as parametric design. *International Journal of Human-Computer Studies, special issue on problem-solving methods*, 1998. Submitted.
- Frank van Harmelen and Annette ten Teije. Validation and verification of conceptual models of diagnosis. *Proceedings of the Fourth European Symposium on the Validation and Verification of Knowledge Based Systems (EUROVAV'97)*, pages 117–128, June 1997. Also in: Validation, Verification & Refinement if AI systems & Subsystems Workshop on IJCAI97.