to appear in: *International Journal of Human-Computer Studies (IJHCS), special issue on Problem-Solving Methods.*

Inverse Verification of Problem-Solving Methods

Dieter Fensel[1] and Arno Schönegge[2]

[1]University of Karlsruhe, Institute AIFB, 76128 Karlsruhe, Germany, dfe@aifb.uni-karlsruhe.de
[2]University of Karlsruhe, Institute LDK, 76128 Karlsruhe, Germany, schoenegge@ira.uka.de

Abstract. Context dependency of knowledge models brings with it several problems: the unreliability of knowledge-based systems, maintenance costs, and limitations on sharing and reuse. Problem-solving methods are knowledge models of the reasoning process of knowledge-based systems. In this paper we present a method called *inverse verification* to deal with the context dependency of problem-solving methods. *Inverse verification* investigates the context dependency of a method by making underlying assumptions explicit. It uses failed proof attempts as a search method for assumptions and an analysis of these failures for constructing and refining assumptions.

1 Introduction

In the last few years, problem-solving methods have become widely used in describing the reasoning behavior of knowledge-based systems (compare the other contributions in this issue). Despite the strong agreement for the usefulness of problem-solving methods and the large number which have been documented there is still a lack of clear methodological support in developing problem-solving methods and in (re-)using them. Recent work in this area ([Akkermans et al., 1993], [Fensel, 1995a], [Wielinga et al., 1995], [Benjamins & Pierret-Golbreich, 1996], [Fensel et al., 1996], [O´Hara & Shadbolt, 1996], [Motta & Zdrahal, 1996], [Benjamins & Aben, 1997], [Breuker, 1997], [Fensel & Schönegge, 1997], [ten Teije, 1997], [Fensel & Straatman, 1998], and [Fensel & Benjamins, to appear]) provides an in-depth analysis of the essence and main rationales of some problem-solving methods. There seems to be a common agreement that the *assumptions* underlying a reasoning process are central in characterizing and developing problem-solving methods.

Problem-solving methods for knowledge-based systems need to make assumptions to provide effective and efficient problem solving: assumptions about the scope of the problem they should solve and assumptions about the domain knowledge they can use as a resource for their reasoning process [Fensel & Straatman, 1998], [Fensel & Benjamins, to appear]. If these assumptions are made explicit they can improve the reusability of problem-solving methods by guiding their refinement process for a given application and defining goals for the acquisition process of domain knowledge. However, making the underlying assumptions explicit is not an easy task. The goal of our paper is to provide methods for solving this problem. The main idea is to construct mathematical proofs and to analyze their failure as a systematic means for formulating assumptions. Tool support is provided by adapting the Karlsruhe Interactive Verifier (KIV) [Reif, 1995] for our purpose. KIV is an interactive theorem prover that returns with open goals if a proof could not be completed. These open goals can be used to derive the assumptions we are looking for.

The paper is organized as follows. In section 2 we discuss the problems which our approach aims to solve and sketch this solution. We will illustrate our approach by analyzing a local search method in section 3 and discuss the kind of assumptions that can be found when trying to perform abductive diagnosis with such a method in section 4. Section 5 sketches how more realistic assumptions reflecting the heuristic nature of problem-solving methods can be found. Related work is discussed in section 6 and section 7 deals with our conclusions and future work.

2 The Problem And its Solution

> "Science does not rest upon solid bedrock. The bold structure of its theories rises, as if were, above a swamp. It is like a building erected on piles. The piles are driven down from above into the swamp, but not down to any natural or 'given' base; and if we stop driving the piles deeper, it is not because we have reached firm ground. We simply stop when we are satisfied that the piles are firm enough to carry the structure, at least for the time being." [Popper, 1959]

2.1 Knowledge is Situated and Brittle

Knowledge is *situated* and its usefulness for different situations is limited [Menzies & Clancey, to appear]. Reuse of knowledge has to come to terms with its situatedness. Knowledge may be powerful in one context and useless or even dangerous in another. It may enable effective and efficient problem-solving if its implicit assumptions are fulfilled. Otherwise, it may produce incorrect results and inefficient search. This *brittleness* of knowledge creates serious problems when trying to reuse it outside the context in which it evolved. Problem-solving methods are a specific type of knowledge concerned with the problem-solving process. Therefore, they share this general problem of knowledge reuse. As a consequence, we will first situate our discussion in the general context of sharing and reusing knowledge and we will then provide specific solutions which deal with reusing problem-solving methods. Clearly, some of them may also be applicable for reusing different types of knowledge.

Cyc is a prominent and long-term research project aiming at a large knowledge base which would enable common-sense reasoning [Guha & Lenat, 1990]. To achieve this goal, large amounts of human common sense knowledge were encoded. A serious problem is encountered, however, when formalizing human knowledge. Knowledge can be understood as a model of reality which serves specific (probably implicit) purposes [Agnew et al., 1994]. Trying to represent this situated knowledge recursively creates the same problem. A representation of this knowledge, i.e. a model, reflects a point of view taken by the modeler [Clancey, 1993]. The point of view he takes reflects implicitly or explicitly the intended use of the model. [Guha, 1993] enumerates three aspects of a representation that reflect the situatedness of a knowledge model: the *vocabulary* used to formulate the model, the *granularity and accuracy* of the model, and the *assumptions* that underly the model.

[McCarthy, 1993] and [Guha, 1993] present *context logic* as a means to deal with this problem.[1] The notion of context is reified within first-order logic by introducing terms that denote a context, i.e. that provide context names. However, providing a notation for context dependency (i.e., a language that allows us to express situatedness) is only half of the work. The second problem is *how to become aware of the context dependency of a model*. That is, how can we provide methodological support for building a model of the context dependency of our knowledge model. Making the context of a (knowledge) model explicit is a tricky problem that is unsolvable in general, but (heuristically) solvable in practice. It is unsolvable in general because its solution would require us to solve a problem of infinite regress. Making a context explicit requires a perspective that is used as point of reference for this activity. Clearly this creates a new context dependency of the model. “As a consequence, there doesn’t seem to be any certain knowledge on which to stop and stand ... that doesn’t rely on unproven assumptions“ [Agnew et al., 1994]. However, this does not imply that there is no pragmatic solution at all. [Agnew et al., 1994] claim that a concept of purpose realized by a social selection process comparable to the effect of the evolution process in nature, would solve the problem in practice. The assumptions which coincide with the desired purpose and its efficient achievement remain and others are rejected, requiring a deeper search for a better (i.e., more suitable) foundation.

The context dependency of knowledge models is more than just a “philosophical“ problem. First, when developing a knowledge model for a single application the developer may have a good intuition about which assumptions can be made so as to deal with his problem adequately. In this case, hidden assumptions become apparent in cases where the system fails. The knowledge-based system may not be able to process a given input, or it returns a result that is not the solution as it is required. Using error situations and system breakdowns

---

1. See [Akman & Surav, 1996] for a survey of formalizations of contexts and [McCarthy & Buvac, 1997] for a more recent introduction to context logic.

is the most common (implicit) search method for assumptions. However, this “method“ may also cause significant damage. Reliable systems require the explication of their assumptions as an explicit part of their development process. Second, contexts have the problematic feature that they change over time. As a consequence, the knowledge-based system must be maintained [Menzies, to appear]. The notion of context can be used as a guideline for answering the questions as to whether it is necessary to change the system and how this can be done without loosing other necessary properties. Third, the problem of context-dependency is immediately present for knowledge models which are intended to be sharable and reusable. In this case they cannot be designed to intuitively fit to a given context because they must be applicable to a broad range of problems not known beforehand.

Making explicit the context of a knowledge model is an infinite process. We cannot expect to reach a solid ground upon which we can built our knowledge model. Therefore, the only things we can provide are shovels and dredgers that can be used to deepen the foundation and to rebuild the house if required. What can be provided and what is needed is active support in *context explication* and in *changing knowledge or its underlying context assumptions* in the case that we have to adapt it to a new context (i.e., we became aware of the fact that it does not fit well to a context).

However, we do not want to investigate knowledge models in general. Instead we focus our attention on a specific knowledge type: *problem-solving methods*. [Fensel, 1995a] attempted to specify the competence of the problem-solving method *propose & revise for parametric design* (cf. [Marcus et al., 1988], [Schreiber & Birmingham, 1996]) enabling its reuse for different tasks. However two significant problems appeared:

- There is not only one *propose & revise* method rather a large number of slightly different variants and there is no justification for selecting one of them as the gold standard.

- Each of these variants uses slightly different assumptions about what the precise problem is and about the strength of the domain knowledge that can be used for its reasoning process. In general, the competence of a problem-solving method cannot be described independent of these assumptions.

One could ask whether a solution would be to choose the variant of a problem-solving method which makes as few assumptions as possible. However, this fails to grasp the essence of problem-solving methods. “In order to think effectively, we must ´know´ a good deal about what not to think! Otherwise, we get bad ideas—and also, take too long.“ [Minsky, 1997] Efficient reasoning is only possible by introducing assumptions. These assumptions are necessary for reducing the complexity both of the reasoning task and of the process for developing the reasoning system. Therefore, *making assumptions for efficiency reasons* is an essential feature of problem-solving methods (cf. [Fensel & Straatman, 1998]). The more assumptions a problem-solving method makes, the more efficient reasoning power it can provide. There are two conclusions that have to be drawn for problem-solving methods. There is neither a gold standard for a problem-solving method nor can we expect to ever find a useful assumption-free method. Therefore, what needs to be provided are generic schemes of problem-solving methods and methodological support for adapting such schemes to problem-specific and domain-specific circumstances. Such support has to deal with two aspects (see Figure 1):

- A method has to be provided that can be used to make explicit context dependency.
- A method that supports the adaptation of a problem-solving method to a given context has to be found.

Context Explication

We already argued that making strong assumptions on available domain knowledge and problem restrictions is not a bug but a feature of problem-solving methods that should enable efficient reasoning. In [Fensel & Benjamins, to appear], we collected a large number of assumptions and related them to the different subtasks of diagnostic problem solving. This assumption list can be used to check given domain knowledge, to define goals for knowledge acquisition, to restrict the size of the problem that must be solved by the knowledge-based system, or to select a problem-solving method that fits to the given context as characterized by the established assumptions. However, this was a kind of post-analysis. These were the results of a common research project which has been carried out for ca. 20 years. Searching for hidden assumptions was a sort of implicit activity for a research community while developing problem-solvers for diagnosis. We will present a method called *inverse verification* that allows a pre-analysis for hidden assumptions. It provides support for an active and explicit search process for these assumptions.

Problem-solving Method Adaptation

The context problem of problem-solving methods was already encountered in the early

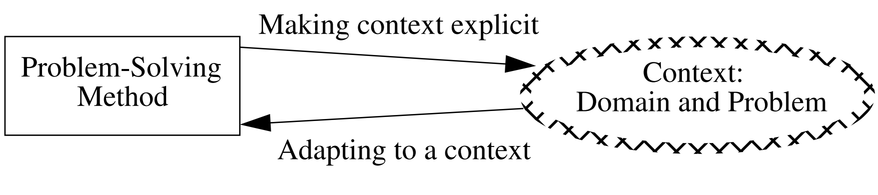

Fig. 1 The two directions of the context problem.

work on *role-limiting methods* [Marcus, 1988] and *generic tasks* [Chandrasekaran, 1986]. The implemented methods fit well for the applications they were developed for. However, the attempt at applying them to similar problems with slightly different goals and domain knowledge failed because of their brittleness. [Klinker et al., 1991] rephrased this as the usability-reusability trade-off of problem-solving methods. On the one hand, the more assumptions and commitments to a specific problem type that are made, the stronger the support of the method in developing a solution for this problem. On the other hand, as more commitments to the specifity of the problem are made, less reusability could be expected. Recently [Beys et al., 1996] have proposed that problem-solving methods should be described not only in a domain-independent but also in a completely task-independent manner so that they can be more broadly reusable. However, this decontextualization of problem-solving methods significantly reduces the usefulness that is usually provided with their task-specific vocabulary, assumptions, and granularity. In their approach, only abstract algorithmic schemas remain, without any relation to the problem. As a consequence, the mapping between such an algorithmic schema and a problem-, domain-, and application-specific problem-solver is very complex and little support in knowledge acquisition is provided.

All these discussions deal with the problem of adapting problem-solving methods to a new or modified context. [Fensel, 1997] describes methodological support for this process by providing a principled way of developing and adapting problem-solving methods. This is achieved mainly by using *adapters* [Fensel & Groenboom, 1997] as a means to express the adaptation of problem-solving methods. Refined versions of problem-solving methods are achieved by combining them with adapters. [Fensel & Motta, 1998] introduce a rational reconstruction of the Motta-library [Motta, 1997] for parametric design based on these principles. In the following we will discuss the aspect of context adaptation only as a side aspect and refer the reader to the quoted literature. We will devote our attention to the first aspect, i.e. we will present a method for context explication.

2.2 Inverse Verification of Problem-Solving Methods

Here we discuss how assumptions that are necessary to close gaps between different elements of a specification can be found using *failed* attempts to prove their proper relationship.

The method consists of two main steps: (1) Establishing the notion of competence of a method in dependence on its assumptions and (2) relating the competence of a method to a problem definition that introduces a general notion of the context in which the problem-solving method can be applied. Both steps are processes for searching for and constructing assumptions and both rely on the *same principle*. However they play *different conceptual roles* and require *different techniques,* as we will see later on.

We use failed proofs as a search method for assumptions and analysis of these failures for constructing and refining them. In other words, we attempt to prove that a problem-solving method achieves a goal and the assumptions appear during the proof as *missing pieces* in proving the correctness of the specification. A mathematical proof written down in a text book explains why a lemma is true under some preconditions (i.e., assumptions and other sublemmas). The proof establishes the lemma by using the preconditions and some deductive reasoning. Taking a look at the proof *process* we get a different picture. Usually, initial proof attempts fail when they run into improvable subgoals. These failed proof attempts point to necessary features that are not present from the beginning. Actually they make aware of further assumptions that have to be made in order to obtain a successful proof. Taking this perspective, a proof process can be viewed as a search and construction process for assumptions. Gaps that can be found in a failed proof provide initial characterizations of missing assumptions. They appear as sublemmas that were necessary to proceed with the proof. An assumption that implies such a sublemma which would close the gap in the proof is a possible candidate for which we are looking. That is, formulating this sublemma as an assumption is an initial step in finding and/or constructing assumptions that are necessary to ensure that a problem-solving method behaves well in a given context. Using an open goal of a proof directly as an assumption normally leads to very strong assumptions. That is, these assumptions are *sufficient* to guarantee the correctness of the proof, but they are often neither *necessary* for the proof nor *realistic* in the sense that application problems will fulfill them. Therefore, further work is necessary to find improved characterizations for these assumptions. This is achieved by a precise analysis of their role in the completed proof to retrace unnecessary properties.

Such proofs can be done semiformally in a textbook style as proposed by the *Abstract State Machines* community [Börger, 1995]. However, providing specification formalisms with a formal syntax and formal semantics allows (semi-)automated proof support. The great amount of details that arise when proving properties of software (and each problem-solving method eventually has to be implemented) indicates the necessity of such mechanization. Therefore, we provide a formal notion for problem-solving methods and semi-automatic proof support by using KIV [Reif, 1995]. KIV provides an *interactive* tactical theorem prover that makes it suitable for hunting hidden assumptions. We expect many proofs to fail. Using a theorem prover that returns with such a failed attempt adds nothing. Instead of returning with a failure KIV returns with open goals that could not be solved during its proof process. In addition, KIV provides support for the generation of counter examples. This is precisely the kind of support we are looking for when finding and constructing assumptions. As opposed to verification, here we do not start a proof with the goal of proving correctness. Instead, we start an impossible proof and view the proof process as a search and construction process for assumptions. Therefore we will call our method *inverse verification*.

A complete introduction to KIV is beyond the scope of the paper. However, we will mention some of its main features and the main rationales for choosing it for our purpose.

KIV supports the software development process starting from formal specifications (algebraic full first-order logic with loose semantics) and ending with verified code (Pascal-like procedures grouped into modules). It has been successfully applied in case-studies up to a size of several thousand lines of code and specification (see e.g. [Fuchß et al., 1995]). Its specification language is based on abstract data types for the functional specification of components and dynamic logic for the algorithmic specification. It provides an interactive theorem prover integrated into a sophisticated tool environment supporting aspects like the automatic generation of proof obligations, generation of counter examples, proof management, proof reuse etc. Such a support is essential for making feasible the verification of complex specifications.

The use of the KIV system for the verification of KBSs is quite attractive. KIV supports dynamic logic (cf. [Goldblatt, 1982], [Harel, 1984]) which has been proved useful in the specification of KBSs (cf. KARL [Fensel, 1995b], $(ML)^2$ [van Harmelen & Balder, 1992], MLPM and MCL [Fensel et al., 1998]). Dynamic logic has two main advantages (especially if compared to first-order predicate logic). First, dynamic logic is quite expressive, e.g. we can formalize and prove the termination or equivalence of programs or the generatedness of data types. Second, in dynamic logic, programs are explicitly represented as part of the formulas. Thus (especially if compared to the verification condition generator approach) formulas and proofs are more readable and provide more structural information which can be employed by proof heuristics.

KIV allows the structuring of specifications and the modularization of software systems. Therefore, the conceptual model of our specifications (see [Fensel & Groenboom, 1997] for more details) can be realized by the modular structure of a specification in KIV. Finally, the KIV system offers well-developed proof engineering facilities: Proof obligations are generated automatically. proof trees are visualized and can be manipulated with the help of a graphical user interface. Even complicated proofs can be constructed with the interactive theorem prover. A high degree of automation can be achieved by a number of implemented heuristics. An elaborated correctness management keeps track of lemma dependencies (and their modifications) and the automatic reuse of proofs allows an incremental verification of corrected versions of programs and lemmas (see [Reif & Stenzel, 1993]). Both aspects are essential to make verification feasible given the fact that system development is a process of steady modification and revision.

The main aspect in choosing KIV as the tool environment for our approach however stems from the fact that KIV applies an *interactive* theorem prover. We already expect our proofs to fail and such a return from an automatic theorem prover would not add much. However an open goal, as it results from an impossible proof, provides an assumption or at least a departure point for constructing an assumption via refining the open goal.

We will illustrate inverse verification by discussing small examples that present our ideas clearly and in a form which is easy to understand. First, we present the search method *hill-climbing* and connect the competence of this local search method with a problem of finding a global optimum. Second, we discuss a version of abductive diagnosis and the kind of assumptions that can be found when trying to solve this problem with a local search method. The reader may argue that we do not discuss problem-solving methods for knowledge-based system but rather simple search methods. However, we would like to mention four arguments:

- The algorithmic core of problem-solving methods consists of simple search methods. Take *propose & revise* ([Marcus et al., 1988], [Schreiber & Birmingham, 1996]) as an example. It is a simple local search method with two different modes: proposing extensions of a state and revising a state if constraint violations occur. Therefore, our results can be immediately applied to this type of method.

- Problem-solving methods do not achieve their "intelligence" through the use of complex algorithmic strategies. Instead, they use strong domain heuristics for the search process, restrict the size of the problem they can deal with properly, and make ontological commitments so as to be immediately applicable to a specific problem type.

- We will explain in section 4 how the search methods that we discuss can be adapted to richer contexts constituting task-specific (or problem-type specific) problem-solving methods.

- In section 5 we will show how more realistic assumptions reflecting the heuristic nature of problem-solving methods can be found.

3 First Example: A Local Search Method

We will illustrate the two main activities: (1) Establishing a notion of the competence of a method in dependence on assumptions and (2) relating the competence of a method with a problem definition by introducing further assumptions. Establishing the competence of an operational algorithm specification requires proof techniques of dynamic logic. We have to relate a procedural specification with a first-order specification of its assumptions and competence. For the second step two declarative specifications must be related. The gap between a competence specification and a problem definition has to be closed via assumptions. The specifications and proofs remain within first-order logic by proving implication between first-order formulas.

3.1 Establishing the Competence of a Problem-Solving Method in

Dependence on Assumptions

Hill-climbing is a local search algorithm that stops when it has found a local optimum. The control flow is defined in Figure 2. The method works as follows: First, we select a start object. Then we recursively generate the successors of the current object and select a successor if we find a better one. Otherwise we terminate and return the current object that does not have better successors. The main requirements on *domain knowledge* that are introduced by *hill-climbing* (and by other local search methods) are the existence of a

operational specification *hill-climbing*

output := *hill-climbing*(*input*)

hill-climbing(*X*)
begin
current := *select-start*(*X*);
output := *recursion*(*current*)
end

recursion(*X*)
begin
successors := *generate*(*X*);
new := *select-a-best*(*X*,*successors*)
if *X* = *new*
then *output* := *X*
else *recursion*(*new*)
endif
end

/* *select-start* must select an element of input and uses a selection criterion. */
$\mathit{select\text{-}start}(x) \in x \wedge \mathit{select\text{-}start}(x) \in \mathit{select\text{-}criterion}(x)$

/* *generate* selects input elements that are in a successor relation with the current object.*/
$x \in \mathit{generate}(y) \leftrightarrow x \in \mathit{input} \wedge \mathit{successor}(y,x)$

/* *select-a-best* selects the current object if no better successors exist or a successor if a better successor exists. In the latter case the selected successor must be better than the current object and there must not be another successor that is better than the selected successor. */
$\neg\exists z\,.\,(z \in \{y\} \cup y' \wedge \mathit{select\text{-}a\text{-}best}(y,y') < z)$
$\neg\exists z\,.\,(z \in y' \wedge y < z) \rightarrow \mathit{select\text{-}a\text{-}best}(y,y') = y$
$\exists z\,.\,(z \in y' \wedge y < z) \rightarrow \mathit{select\text{-}a\text{-}best}(y,y') \in y' \wedge y < \mathit{select\text{-}a\text{-}best}(y,y'))$

endoperational spec

Fig. 2 The operational specification of *hill-climbing*.

preference relationship and a *successor* relationship between the objects. The former is used for selection and the latter is used to enable the local search process. A third requirement is a selection criterion for the start object of the search process. The performance and competence of the method depend on the properties of these three relations (cf. [Graham & Bailor, 1996]).

We tried to prove that *hill-climbing* always terminates and that it has the competence to find a local optimum (see Figure 3). KIV automatically generates all proof obligations in dynamic logic that are necessary to ensure termination and competence of an algorithmic specification. In our case it generates the following proof obligations:

$\vdash \langle \textit{hill-climbing}(\textit{input}) \rangle$ true, i.e. termination

$\vdash \langle \textit{hill-climbing}(\textit{input}) \rangle\ \textit{output} \in \textit{input}$

$\vdash \langle \textit{hill-climbing}(\textit{input}) \rangle\ \neg\exists x\ .\ (\textit{successor}(\textit{output},x) \wedge x \in \textit{input} \wedge \textit{output} < x)$.

Tool support is provided in unfolding these proof obligations and in applying tactics. In general the user has to select proof tactics and heuristics from a menu. In our case, user interaction is needed to select axioms of a specification as supporting lemmas for the proof and to select the kind of induction. We run into a number of open goals during the proofs:

- We had to ensure that *select-criterion* retrieves an object for each possible input. Otherwise it cannot be guaranteed that *hill-climbing* will provide an output.

- We have to ensure that the preference we use ($<$) is a *partial order*. We have to be sure of irreflexivity and transitivity to ensure that *hill-climbing* cannot be caught up in circles (imagine $a < b$ and $b < a$ and $a \in \textit{successor}(b)$ and $b \in \textit{successor}(a)$). In

competence *hill-climbing*
  axioms
    $\textit{output} \in \textit{input}$
    $\neg\exists x\ .\ (\textit{successor}(\textit{output},x) \wedge x \in \textit{input} \wedge \textit{output} < x)$
endcompetence

requirements
  axioms
    $\exists x.\ (x \in \textit{input} \wedge x \in \textit{select-criterion}(\textit{input}))$
    $\neg(x < x)$
    $x < y \wedge y < z \rightarrow x < z$
endrequirements

Fig. 3 Competence and requirements of *hill-climbing*.

addition, the finiteness of the input set has to be ensured.

Based on these assumptions (shown as requirements in Figure 3) we could establish that *hill-climbing* finds a *local* optimum of the input.

3.2 Connecting the Competence of a Problem-Solving Method to a Task

A *task definition* specifies the *goals* that should be achieved by the knowledge-based system. It establishes an explicit notion of the context in which the problem-solving method is applied. A general context where *hill-climbing* can be applied is in searching for a (global) optimum. Figure 4 provides the definition for our running example. The goal describes what an optimum must fulfill. However, the problem-solving method *hill-climbing* only has the competence to find a local optimum in a graph. We start the interactive proof process again knowing that it will lead us to further assumptions because *hill-climbing* in general does not have the competence to find a global optimum. Two main problems arise during the proof:

1) We would have to prove that the selected start object is always connected with a global optimum (several global optima may exist because we do not require a total order). Otherwise, the global optimum is not attainable by the recursive search of *hill-climbing*.

2) Even if we could prove (1) we may get stuck at the case distinction
   if $X = new$
   where *hill-climbing* stops due to a local optimum.

A trivial assumption that closes both gaps in the proof is to require that each object be directly connected with each other object.

totally-connected assumption: *successor*(x,y)

task *global optimum*
  goal
    global-optimum $\in$ *input*
    $\neg\exists x \,.\, (x \in input \wedge global\text{-}optimum < x)$
endtask

Fig. 4 The task definition *global optimum*.

However this is a very useless assumption. In this case, *hill-climbing* collapses to a complete search in one step because all objects are successors of each possible start object. A less drastic assumption is to require that each object (except a global optimum) has a successor with a higher preference.

better-successor assumption:

$$x \in \mathit{input} \rightarrow (\exists y \,.\, (y \in \mathit{input} \wedge \mathit{successor}(x,y) \wedge x < y) \vee \neg \exists z \,.\, (z \in \mathit{input} \wedge x < z))$$

This assumption is derived to close the gap in the case distinction of the recursion. If the recursion stops we have found a global optimum. We already know from the termination proof of *hill-climbing* that the problem-solving method always terminates.

The question remains as to whether the assumptions are *minimal*. Here, *minimality* means that the assumptions are not only sufficient but also *necessary* to guarantee that the competence of the problem-solving method implies the problem definition, formally:

$$(\mathit{Problem\text{-}solving\ method}_{competence} \rightarrow \mathit{Task\ Definition}) \rightarrow \mathit{Assumption.}$$

An assumption that is *minimal in the logical sense* (i.e., necessary) has the clear advantage that it maximizes the circumstances under which it holds true. It does not require anything more than what is precisely necessary to close the gap between the competence and the problem. In fact, we have proven with KIV that the *better-successor assumption* is a minimal assumption in the logical sense. Actually this proof process leads to several refinements of the original assumption as we encountered several holes during the proof process. In general, minimizing (i.e., weakening) assumptions can be achieved by analyzing their sufficiency proof with KIV and eliminating aspects that are not necessary for continuing the proof. However, besides logical minimality other aspects like cognitive minimality (effort in understanding an assumption) or computational minimality (effort in proving an assumption) may also influence the choice of assumptions. We will illustrate this in the following section.

The two assumptions that we have introduced are rather trivial. This is a consequence of the simplistic example. In the following section we will define a more complex task and problem-solving method which will lead to more interesting assumptions.

4 Second Example: Finding an Abductive Explanation

In the following, we use a richer task definition as example. The task *abductive diagnosis* receives a set of observations as input and delivers a complete and parsimonious explanation (see e.g. [Bylander et al., 1991]). An explanation is a set of hypotheses. A *complete explanation* must explain all input data (i.e., *observations*) and a *parsimonious* explanation must be minimal (that is, no subset of hypotheses explains all *observations*). Figure 5 provides the task definition for our example. Any explanation that fulfills the *goal* must be *complete* and *parsimonious*. The task introduces requirements on domain knowledge: It must provide sets to interpret the sorts *datum* and *hypothesis* and an explanation function to interpret *explain*. The (axiomatic) input requirement ensures that there are *observations*.

Finding a complete and parsimonious explanation is intractable in the number of hypotheses [Bylander et al., 1991]. Therefore, we have to apply heuristic search strategies. In the following, we characterize a local search method which we call *set-minimizer*. The competence theory in Figure 6 defines the competence of *set-minimizer*: it is able to find a correct and locally minimal set. Local minimality means that there is no correct subset of the output that only has one less element. The method has one requirement: it must receive a correct initial set (cf. Figure 6). The competence and the requirement illustrate the additional aspects that are introduced by the problem-solving method:

task *complete and parsimonious explanation*
  sorts
    datum, data : **set of** *datum,*
    hypothesis, hypotheses : **set of** *hypothesis*
  functions
    explain: *hypotheses* $\rightarrow$ *data*
    observables: *data*
    goal : *hypotheses*
  predicates
    complete: *hypotheses*
    parsimonious: *hypotheses*
  variables
    x : *datum*
    H,H' : *hypotheses*
  axioms
    goal
      $complete(goal) \wedge parsimonious(goal)$
      $complete(H) \leftrightarrow explain(H) = observables$
      $parsimonious(H) \leftrightarrow \neg\exists H' \; (H' \subset H \wedge explain(H) \subseteq explain(H'))$
    requirement
      $\exists x \; (x \in observables)$
endtask

Fig. 5 The task definition for *abduction*.

- The task of finding a parsimonious set is reduced to local parsimonious sets.

- Constructing an initial correct set is beyond the scope of the method. It is assumed as being provided by the domain knowledge, by a human expert, or by another problem-solving method. The method only minimizes this correct set.

The applied search strategy is one-step look ahead. The *competence* states that *set-minimizer* is able to find a *local* minimal subset of the given set of objects. The axioms of its competence definition in Figure 6 state that it finds a subset that is correct, and that each set containing one element less is not a correct set. We skip all proofs that were necessary to establish this competence. Actually we could reuse the proofs that were done for *hill-climbing* based on our *adaptation method* (see [Fensel, 1997], [Fensel et al., 1997]) and the proof reuse facilities of KIV. Set-minimizer is a task-specific adaptation of hill-climbing. By providing a library of reusable problem-solving methods and support in adaptation, their proofs can be reused, too. Therefore, this type of proof that an operational specification of a problem-solving method has some competence only has to be done once when introducing the problem-solving method into the library.

The task and problem-solving method are connected by providing the set of all hypotheses as input and identifying correct sets with complete explanations, i.e.

- *input* := $\{x \mid x$ is a *hypothesis*$\}$ and

- *correct*(x) $\leftrightarrow$ *complete*(x).

competence *set-minimizer*
    import requirements *set-minimizer*
    variables *x* : *object*
    constants *Output* : *objects*
    axioms
        correct(*Output*)
        $x \in Output \rightarrow \neg correct(Output \setminus \{x\})$
endcompetence

requirements *set-minimizer*
    sorts *object, objects* **set of** *object*
    predicates *correct* : *objects*
    constants *Input* : *objects*
    axioms
        correct(*Input*)
endrequirements

Fig. 6 The competence and requirements of the problem-solving method *set-minimizer.*

The circumstances that ensure that the problem-solving method achieves the goal as introduced by the problem definition have to be investigated again. The same procedure can be used again. We try to prove:

- *complete*(*output*)
- *parsimonious*(*output*)

Completeness. The completeness of output follows directly from the competence of the problem-solving method. However, it is based on the input requirement of the method. The problem definition of abduction has to be strengthened. The set of all hypothesis (i.e., the input) must be a complete explanation. This puts a strong restriction on the abductive problem: The function *explain* has to be defined in a way that adding hypotheses to a set of hypotheses does not destroy the explanatory power of the set (cf. [Bylander et al., 1991]).[2]

Parsimony. The competence of our method ensures local parsimony. Our method *set-minimizer* finds a local-minimal set that is parsimonious in the sense that each subset that contains one element less is not a complete explanation. However, it cannot be guaranteed that it is parsimonious in general. Smaller subsets may exist that are complete explanations. The reader may already have realized the similarity with the problem of *hill-climbing* that only finds local optimal elements. However, with the *better-successor assumption,* global optima can be found. A natural way to close our gap is therefore to reformulate the *better-successor assumption* and *global optimum* in terms of the new problem. Instantiating the *better-successor assumption* requires the definition of a preference and a successor relation:

$$x < y :\leftrightarrow explain(x) \subseteq explain(y)$$

$$successor(x,y) :\leftrightarrow \exists z . (z \in x \wedge y = x \setminus \{z\})$$

The *better-successor assumption* can now be reformulated based on these two definitions and the definition of the input above:

$$\begin{aligned} x \subseteq hypotheses \rightarrow \\ &(\exists y . (y \in x \wedge explain(x) \subseteq explain(x \setminus \{y\})) \\ &\vee \neg \exists z . (z \subseteq hypotheses \wedge explain(x) \subseteq explain(z)) \end{aligned}$$

However, this is a minimal but not very intuitive assumption. Again we apply our failed-proof technique. We use it now to *generalize* an assumption. The idea is to prove that the *better-successor assumption* holds based on the problem definition, the input requirements and the competence of the *set-minimizer* method, as well as the way the preference was defined. KIV returns with open goals that would be necessary to complete the proof. These

2. In fact, this property will be established as a consequence during the proof of *parsimony.*

open goals are generalizations of the *better-successor assumption* because they imply its truth. This proof attempt with KIV is straightforward and gets stuck in the following subgoal:

$$y \subset x \wedge explain(x) \subseteq explain(y) \rightarrow \exists z \,.\, (z \in x \wedge explain(x) \subseteq explain(x \setminus \{z\})$$

This assumptions requires that if there are smaller subsets of x with greater explanatory power then there must be another subset that differs only in one element from x and also has greater explanatory power. A simple generalization of this implication is to negate its premise. That is, we select the strengthening tactic:

$$\neg a \vdash a \rightarrow b$$

This tactics leads to

$$\neg(y \subset x \wedge explain(x) \subseteq explain(y)),$$

i.e.,

$$y \subset x \rightarrow \neg\, explain(x) \subseteq explain(y)$$

This assumption requires that for any set of hypotheses no subset exists that has a greater explanatory power. Deleting a hypothesis from the set of hypotheses may never lead to a superset of explained observations. Actually, this assumption plays a prominent role in the literature on abductive reasoning and model-based diagnosis. A strengthened version of this assumption is used by [Bylander et al., 1991] to define polynomial subclasses of abduction. In general, abduction is NP-hard in the number of hypotheses. However, with the following monotonic abduction assumption

$$y \subset x \rightarrow explain(y) \subseteq explain(x)$$

[Bylander et al., 1991] prove that it is possible to find a complete and parsimonious explanation in polynomial time. The assumption they use requires that a superset of hypotheses also explains a superset of observations. The assumption is used to restrict the worst-case effort of a method. [de Kleer et al., 1992] examine its role in model-based diagnosis. The assumption holds for applications where no knowledge that constrains the fault behavior of devices is provided or where this knowledge respects the *limited-knowledge-of-abnormal behavior assumption*. This is used by [de Kleer & Williams, 1987] as the *minimal diagnosis hypothesis* to reduce the *average-case effort* of finding all parsimonious and complete explanations with GDE. A syntactical way to ensure this assumption (i.e., to formulate it as a requirement on the domain knowledge) is to restrict the domain theory to Horn clauses constraining only the correct behavior of devices, cf. [de Kleer et al., 1992]. It is interesting to see how the very generic *better-successor assumption* transforms into such intuitive and broadly used task-specific assumptions.

5 Heuristic Assumptions

The assumptions introduced so far ensure that a local method solves a global problem. They ensure that a local search method has the same competence as a global search method. However, that is not what we are usually looking for. Often these assumptions are too strong. In addition, we mentioned that GDE uses the monotonic-abduction problem to reduce the average-case behavior of the problem-solver. However, we have not yet provided measurements nor proof of these aspects. This section discusses how both can be integrated.

5.1 Domain-specific Reformulation and Weakening of Assumptions

[Marcus et al., 1988] introduced the problem-solving method *propose & revise*, which was also recently used by several research groups in a comparable case study (cf. [Schreiber & Birmingham, 1996]). It is a local search method consisting of two substeps: *Propose* extends a current state and *revise* modifies the state if constraint violations occur. Both activities are iterated until a complete and correct state is reached. In general, this cannot be guaranteed because *propose & revise* does not include a backtracking mechanism that would allow us to escape a dead-end in the solution process. Completeness of the search method can only be guaranteed if we introduce strong assumptions about the domain knowledge that is used by the *propose* and *revise* steps (see Figure 7).

[Zdrahal & Motta, 1995] provide an interesting and in-depth analysis of the problem-solving method *propose & revise* and its application to the vertical transportation (VT) domain ([Marcus et al., 1988], [Schreiber & Birmingham, 1996]). The goal is to design an elevator that meets several requirements and constraints. [Zdrahal & Motta, 1995] identify two key parameters in this domain that influence the difficulty of the problem: the *capacity* and the required *speed* of the elevator. The larger the values of these two parameters the more difficult it is to find a solution. They investigate how *propose & revise* behaves based on the available domain knowledge for different combinations of these two parameter values. The instantiated method failed for some of the simple cases and, not surprisingly, for many of the difficult ones. Some of the difficult ones could be solved by a complete search method, however it required large amount of storage size and computation time. Failures of *propose & revise* in these cases can be accepted because we are looking for a heuristic problem-solver that gains efficiency by restriction to the simple cases. Aiming for a complete and efficient problem-solver for the general case would be an unsolvable problem. However, *propose & revise* also fails for some of the simple cases and this must be viewed as being due to gaps in the domain knowledge provided. The domain knowledge should be strong enough to enable *propose & revise* to find a solution for these cases.

Based on this domain analysis we could weaken our assumptions of Figure 7. Instead of requiring that we always have a *propose* and a *revise* step available that find an optimal successor state we could restrict these requirements to the more simple cases. We could define boundaries for the values of the key parameters in the assumptions. Completeness or optimality may only be guaranteed for domain cases having small values for these key parameters. For difficult cases we either have to use a more complex search method with higher demands on storage and time or we have to ask the human expert to solve them. Such an approach for classification methods is described in [van Harmelen & ten Teije, 1998].

5.2 Specification and Verification including Thresholds for the

competence *propose & revise*
axioms
/* The output is a complete, correct and optimal state. */
$Complete(output) \wedge Correct(output) \wedge$
$\neg\exists\, s\, .\, (Complete(s) \wedge Correct(s) \wedge output < s)$
endcompetence

requirements *propose & revise*
axioms
/* The propose knowledge never fails and monotonically extends the state. */
$\neg\, Complete(s) \rightarrow Partial\ completeness(s) < Partial\ completeness(propose(s))$
/* The application of a propose leads to an optimal state. */
$\neg\, Complete(s) \rightarrow$
$\neg\exists\, s' (Correct(s') \wedge propose(s) < s' \wedge$
$Partial\ completeness(s') = Partial\ completeness(propose(s))$
/* The revise knowledge never fails. */
$\neg\, Correct(s) \rightarrow Correct(revise(s))$
/* The application of revise does not change the completeness of a state. */
$Partial\ completeness(revise(s)) = Partial\ completeness(s)$
/* The application of revise leads to an optimal state. */
$\neg\, Correct(s) \rightarrow$
$\neg\,\exists\, s' (Correct(s') \wedge revise(s) < s' \wedge$
$Partial\ completeness(revise(s)) = Partial\ completeness(s'))$
endrequirements

Fig. 7 The competence of *propose & revise* (see [Fensel et al., 1997]).

Computational Effort

The assumptions of problem-solving methods are motivated by the goal of improving the efficiency of the assumption-based reasoning process compared to a reasoning process using less assumptions. However, we have not yet provided at the means to specify and verify the efficiency of problem-solving methods. [Shaw, 1989] and [Straatman & Beys, 1995] described elements that can be integrated into our framework. [Shaw, 1989] includes counters and boundaries for the values of these counters in the specification of real-time software. Precisely the same can be done for problem-solving methods. In the case of *hill-climbing* we can add a counter for the number of successors that are derived and compared in one step and a second counter for the number of steps. Then, we can formulate boundaries for their combined value and formulate this in terms of assumptions that either introduce requirements on the domain knowledge or restrict the set of problems that are solved by *hill-climbing* (i.e., *hill-climbing* terminates after it has consumed its computational time). Using this direction, work on *anytime* algorithms (cf. [Zilberstein, 1996]) could be integrated into the work on problem-solving methods.

In general, such refinements of specifications by using measurements allow more refined assumptions on the graph that is used by the search methods. [Stefik, 1995] provides informal examples of such refined assumptions for different variants of different problem-solving methods in what he calls the symbol-level analysis of problem-solving methods. For example, he discusses the circumstances under which data-directed searches, solution-directed searches, and opportunistic searches are preferable for classification methods. Each search type requires some knowledge types with specific properties in order to perform well (i.e., effectively *and* efficiently).

6 Related Work

> "The problem of performing deduction of new facts from a set of axioms is well-studied and understood. An equally important but far less explored problem is the derivation of hypotheses to explain observed events. In formal terms this involves finding an *assumption* that, together with some axioms, implies a given formula." [Cox & Pietrzykowski, 1986]

[de Kleer, 1986] describes a truth-maintenance system (ATMS) that could in principle be applied to our problem. Actually most of the approaches to model-based diagnosis we discussed use adaptations of this technique. However, applying this technique introduces two strong (meta-)assumptions:

- All the assumptions required to solve the gap between the goals of the task and the competence of the problem-solving method must already be known and provided to the system.
- The system needs to know the impacts of the assumptions, i.e. their influence on the truth of the formulas describing the competence of the problem-solving method and the goals of the task.

If this complete knowledge is available, establishing the proper set of assumptions boils down to selecting a minimal set of assumptions, and a bookkeeping mechanism like ATMS can process this task (cf. [de Kleer & Williams, 1987]). When such a complete set of assumptions does not exist, finding assumptions is rather a constructive activity.

Constructive approaches to derive such assumptions can be found in approaches to abduction ([Cox & Pietrzykowski, 1986]), in program debugging with inductive techniques (cf. [Shapiro, 1982], [Lloyd, 1987]), in explanation-based learning (cf. [Minton et al., 1989], [Minton, 1995]) or more generally in inductive logic programming ([Muggleton & Buntine, 1988], [Muggleton & De Raedt, 1994]). However, these approaches achieve automation by making strong (meta-)assumptions about the syntactical structure of the representation formalisms of the components, the representations of the "error", and the way an error can be fixed. Usually, Prolog or Horn logic is the assumed representation formalism, and errors or counter-examples are represented by a set of input-output tuples or a finite set of ground literals. Modification is done by changing the internal specification of a component. In this scenario, error detection boils down to backtracking a resolution-based derivation tree for a "wrong" literal. However, we have to aim for new formulas (i.e., an assumption may be represented by a complex first-order formula) and our "counter-examples" are not represented by a finite set of ground literals but by a complex first-order specification. Also most of the approaches mentioned do not regard architectural descriptions of the entire reasoning system.[3]

We distinguished two roles of assumptions. First, they are necessary to ensure that a problem-solving method has a specific competence. For example, without the assumption that the preference relation is a partial order, the termination proof of *hill-climbing* would not be possible. Second, they are necessary to ensure that the competence of a problem-solving method is able to achieve a goal as introduced by a problem definition. For example, with the *better-successor assumption* we can prove that the competence of *hill-climbing* is strong enough to find a global optimum. Both roles differ conceptually and technically.

The first aspect has to be examined when establishing a problem-solving method in a library of reusable elements. Such elements must be reliable in the sense that they

3. An exception are the approaches to explanation-based learning that use explicit architecture axioms [Minton, 1995].

guarantee some competence and the conditions necessary to provide this competence. Technically, it is a proof that concerns the algorithmic structure of the method. Therefore, dynamic logic is used to specify the algorithm and the proof obligations are formulas in dynamic logic.

(1) Assumptions $\rightarrow$ [Operational Specification] Competence

Finding the weakest preconditions for an algorithm has a long tradition in software engineering (cf. the wp calculus [Dijkstra, 1975] and predicate transformers [Chandy & Sanders, 1995]). Since the dynamic logic we use can be regarded as a generalization of Hoare-triples we can employ methods and techniques developed in this area (e.g. the B-Toolkit [Wordsworth, 1996] or Z/EVES [Meisels & Saaltink, 1996]). A very interesting approach in this area is KIDS/SPECWARE [Smith, 1996] that supports the semiautomatic derivation of efficient programs. [Smith, 1985] provides a generalized notion of deduction that includes the derivation of weakest pre-conditions for input to ensure the correctness of a synthesized program. The main distinction between this work in software engineering and our method for knowledge-based systems stems from the specifity of the class of systems we are referring to. In software engineering the derived pre-conditions are rather simple safe guards that exclude some input values, for example that prevent division by zero or require a non-empty list for sorting. In our case, the entire task is decomposed into assumptions on domain knowledge, into assumptions that reduce the complexity of the task, and into the competence of a problem-solving method. The algorithmic part of the problem-solving method is only a minor part in problem-solving. Most of the problem is assumed to be solved by the domain knowledge[4] and the problem-solving method basically defines the right order in which this knowledge is to be applied.

In consequence these assumptions have a different purpose and require different techniques. We are also looking for assumptions that ensure that the competence of the method implies the problem definition, for example

(2) Assumptions $\rightarrow$ (Competence $\rightarrow$ Problem Definition)

That is, we relate two declarative specifications, the functionality of the system and the specification of the required functionality. Assumptions are used to split the required functionality into two parts. One part that is solved by the competence of the problem-solving method (in the case that the assumptions hold) and one part that is only assumed to be solved. That is, this part is either solved by the domain knowledge, by a possible external human agent, or it must be viewed as a restriction on the class of solvable problems. We argued that our problem-solving methods can only provide a limited fragment of the entire functionality because of the intractability of typical problems they are applied to. In software engineering, a different point of view is usually taken. One

4. *nomen es omen*

assumes that the specification functionality must be completely provided by the system. In this setting, our assumption hunting method makes no sense because the distinction between the two different specifications and their relationship does not exist at all.

7 Conclusions

Like every other knowledge model problem-solving methods are connected with a context. This situatedness may appear in three types of problems: (1) The context which a problem-solving method is developed for may cause an error because the developers were not aware of an implicit assumption that does not hold in this context. (2) The initial context of a problem-solving method may change over time and this causes the typical maintenance problems. (3) A problem-solving method is to be shared and reused in a new context and it is hard to decide whether it fits into the new context and whether and how it has to be adapted.

In [Fensel & Straatman, 1998] we proposed the idea of characterizing and developing problem-solving methods by using their underlying assumptions. These assumptions make the context dependency of a problem-solving method explicit. However, the problem of how to find such assumptions arises. We presented the *inverse verification* method that uses failed proof attempts and an interactive theorem prover for this purpose. KIV has been shown to be an excellent tool for our purpose. Its concepts of proof modularity and proof reuse made the highly iterative and reversive development and adaptation process of problem-solving methods tractable. The interactive theorem prover could be used to identify assumptions as open goals in partial proofs.

Acknowledgments. We would like to thank Richard Benjamins, Joost Breuker, Rix Groenboom, Tim Menzies, Enrico Motta, Annette ten Teije, Frank van Harmelen, and Bob Wielinga for inspiring discussions on issues related to this paper. Jeff Butler did a great job in improving the English.

References

[Agnew et al., 1994]
N. M. Agnew, K. M. Ford, and P. J. Hayes: Expertise in Context: Personally Constructed, Socially Selected, and Reality-Relevant?, *International Journal of Expert Systems*, 7(1), 1994.

[Akkermans et al., 1993]

J. M. Akkermans, B. Wielinga, and A. Th. Schreiber: Steps in Constructing Problem-Solving Methods. In N. Aussenac et al. (eds.), *Knowledge-Acquisition for Knowledge-Based Systems*, Lecture Notes in Artificial Intelligence (LNAI) 723, Springer-Verlag, Berlin, 1993.

[Akman & Surav, 1996]
V. Akman and M. Surav: Steps Toward Formalizing Context, *AI Magazine*, 17(3), 1996.

[Benjamins & Aben, 1997]
R. Benjamins and M. Aben: Structure-Preserving Knowledge-Based System Development through Reusable Libraries: a Case Study in Diagnosis, *International Journal on Human-Computer Studies (IJHCS)*, 47(2):223—258, 1997.

[Benjamins & Pierret-Golbreich, 1996]
R. Benjamins and C. Pierret-Golbreich: Assumptions of Problem-Solving Method. In N. Shadbolt et al. (eds.), *Advances in Knowledge Acquisition*, Lecture Notes in Artificial Intelligence (LNAI) 1076, Springer-Verlag, Berlin, 1996.

[Beys et al., 1996]
P. Beys, R. Benjamins, and G. van Heijst: Remedying the Reusability-Usability Trade-off for Problem-solving Methods. In *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´96)*, Banff, Canada, November 9-14, 1996.

[Börger, 1995]
E. Börger: Why Use Evolving Algebras for Hardware and Software Engineering. In M. Bartosek et al. (eds.), *SOFSEM'95: Theory and Practice of Informatics*, Lecture Notes in Computer Science (LNCS) 1012, Springer-Verlag, 1995.

[Breuker, 1997]
J. Breuker: Problems in Indexing Problem Solving Methods. In *Proceedings of the Workshop on Problem-Solving Methods for Knowledge-based Systems (W26)* during the *Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97),* NAGOYA, Japan, August 23-29, 1997.

[Buvac & Iwanska, 1997]
T. Buvac and L. Iwanska (eds.): Context in Knowledge Representation and Natural Language. In *Working Notes AAAI-97 Fall Symposium Series*, MIT Cambridge, Massachusetts, November 8-10, 1997.

[Bylander et al., 1991]
T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson: The Computational

Complexity of Abduction, *Artificial Intelligence*, 49:25—60, 1991.

[Chandrasekaran, 1986]
B. Chandrasekaran: Generic Tasks in Knowledge-based Reasoning: High-level Building Blocks for Expert System Design. *IEEE Expert*, 1(3): 23—30, 1986.

[Chandy & Sanders, 1995]
K. M. Chandy and B. A. Sanders: Predicate Transformers for Reasoning about Concurrent Programs, *Science of Computer Programming,* 24, 1995.

[Clancey, 1993]
W. J. Clancey: The Knowledge Level Reinterpreted: Modeling Socio-Technical Systems, *The International Journal of Intelligent Systems*, 8(2), 1993.

[Cox & Pietrzykowski, 1986]
P. T. Cox and T. Pietrzykowski: Causes of Events: Their Computation and Application. In *Proceedings of the 8th International Conference on Automated Deduction*, Oxford, England, July 27 - August 1, Lecture Notes in Computer Science (LNCS) 230, Springer-Verlag, 1986.

[de Kleer, 1986]
J. de Kleer: An Assumption-based TMS, *Artificial Intelligence*, 28, 1986.

[de Kleer et al., 1992]
J. de Kleer, A. K. Mackworth, and R. Reiter: Characterizing Diagnoses and Systems, *Artificial Intelligence*, 56, 1992.

[de Kleer & Williams, 1987]
J. de Kleer and B. C. Williams: Diagnosing Multiple Faults, *Artificial Intelligence*, 32:97—130, 1987.

[Dijkstra, 1975]
E. W. Dijkstra: Guarded Commands, Nondeterminancy, and Formal Derivation of Programs, *Communication of the ACM,* 18:453—457, 1975.

[Fensel, 1995a]
D. Fensel: Assumptions and Limitations of a Problem-Solving Method: A Case Study. In *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´95)*, Banff, Canada, January 26 - February 3, 1995.

[Fensel, 1995b]
D. Fensel: *The Knowledge Acquisition and Representation Language KARL*, Kluwer Academic Publ., Boston, 1995.

[Fensel, 1997]
D. Fensel: The Tower-of-Adapter Method for Developing and Reusing Problem-Solving Methods. In E. Plaza et al. (eds.), *Knowledge Acquisition, Modeling and Management*, Lecture Notes in Artificial Intelligence (LNAI) 1319, Springer-Verlag, 1997.

[Fensel & Benjamins, to appear] D. Fensel and R. Benjamins: The Role of Assumptions in Knowledge Engineering, to appear in *International Journal of Intelligent Systems (IJIS)*.

[Fensel et al., 1996]
D. Fensel, H. Eriksson, M. A. Musen, and R. Studer: Conceptual and Formal Specification of Problem-Solving Methods, *International Journal of Expert Systems*, 9(4), 1996.

[Fensel et al., 1997]
D. Fensel, E. Motta, S. Decker, and Z. Zdrahal: Using Ontologies For Defining Tasks, Problem-Solving Methods and Their Mappings. In E. Plaza et al. (eds.), *Knowledge Acquisition, Modeling and Management*, Lecture Notes in Artificial Intelligence (LNAI) 1319, Springer-Verlag, 1997.

[Fensel et al., 1998]
D. Fensel, R. Groenboom, and G. R. Renardel de Lavalette: Modal Change Logic (MCL): Specifying the Reasoning of Knowledge-based Systems, to appear in *Data and Knowledge Engineering (DKE)*, 1998.

[Fensel & Groenboom, 1997]
D. Fensel and R. Groenboom: Specifying Knowledge-Based Systems with Reusable Components. In *Proceedings of the 9th International Conference on Software Engineering & Knowledge Engineering (SEKE-97)*, Madrid, Spain, June 18-20, 1997.

[Fensel & Motta, 1998]
D. Fensel and E. Motta: Structured Development of Problem Solving Methods. In *Proceedings of the 11th Workshop on Knowledge Acquisition, Modeling and Management (KAW´98)*, Banff, Canada, April 1998.

[Fensel & Schönegge, 1997]
D. Fensel and A. Schönegge: Assumption Hunting as Developing Method for Problem-Solving Methods, In *Proceedings of the Workshop on Problem-Solving Methods for Knowledge-Based Systems during the 15th International Joint Conference on AI (IJCAI-97)*, Nagoya, Japan, August 23-30, 1997.

[Fensel & Straatman, 1998]

D. Fensel and R. Straatman: The Essence of Problem-Solving Methods: Making Assumptions to Gain Efficiency, *International Journal on Human-Computer Studies (IJHCS)*, to appear 1998.

[Fuchß et al., 1995]
Th. Fuchß, W. Reif, G. Schellhorn and K. Stenzel: Three Selected Case Studies in Verification. In M. Broy and S. Jähnichen (eds.): *Methods, Languages, and Tools for the Construction of Correct Software*, Lecture Notes in Computer Science (LNCS), no 1009, Springer-Verlag, 1995.

[Goldblatt, 1982]
R. Goldblatt: *Axiomatising the Logic of Computer Science*, LNCS 130, Springer-Verlag, Berlin, 1982.

[Graham & Bailor, 1996]
R. P. Graham, Jr. and P. D. Bailor: Synthesis of Local Search Algorithms by Algebraic Means. In *Proceedings of the 11th Knowledge-Based Software Engineering Conference (KBSE-96)*, 1996.

[Guha, 1993]
R. V. Guha: *Context Dependence of Representations in Cyc*, MCC Technical Report, CYC 066-93, 1993.

[Guha & Lenat, 1990]
R. V. Guha and D. B. Lenat: Cyc: A Midterm Report, *AI Magazine*, 11:32—59, 1990.

[Harel, 1984]
D. Harel: Dynamic Logic. In D. Gabby et al. (eds.), *Handbook of Philosophical Logic, vol. II, Extensions of Classical Logic*, D. Reidel Publishing Company, Dordrecht (NL), 1984.

[Klinker et al., 1991]
G. Klinker, C. Bhola, G. Dallemagne, D. Marques, and J. McDermott: Usable and Reusable Programming Constructs, *Knowledge Acquisition*, 3:117—136, 1991.

[Lloyd, 1987]
J. W. Lloyd: Declarative Error Diagnosis, *New Generation Computing,* 5:133—154, 1987.

[Marcus, 1988]
S. Marcus (ed.): *Automating Knowledge Acquisition for Experts Systems*, Kluwer Academic Publisher, Boston, 1988.

[Marcus et al., 1988]
S. Marcus, J. Stout, and J. McDermott VT: An Expert Elevator Designer That Uses Knowledge-based Backtracking, *AI Magazine*, 9(1):95—111, 1988.

[McCarthy, 1993]
J. McCarthy: Notes on Formalizing Context. In *Proceedings of the 13th International Conference on Artificial Intelligence (IJCAI-93)*, Chambery, France, 1993.

[McCarthy & Buvac, 1997]
J. McCarthy and S. Buvac: Formalizing Context (Expanded Notes). [Buvac & Iwanska, 1997].

[Meisels & Saaltink, 1996]
I. Meisels and M. Saaltink: The Z/EVES Reference Manual, ORA Canada, 1996. Available via http://www.ora.on.ca/z-eves/.

[Menzies, to appear] T. Menzies: *35 Kinds of Knowledge Maintenance*, to appear in *The Knowledge Engineering Review*.

[Menzies & Clancey, to appear]
T. Menzies and W. J. Clancey (eds.): *Special Issue on the Challenge of Situated Cognition for Symbolic Knowledge Based Systems*, *International Journal of Human-Computer Studies (IJHCS)*, to appear.

[Minsky, 1997] M. Minsky: Negative Expertise. In P. J. Feltovich et al. (eds.), *Expertise in Context*, AAAI Press, The MIT Press, 1997.

[Minton, 1995]
S. Minton: Quantitative Results Concerning the Utility of Explanation-Based Learning. In A. Ram and D. B. Leake (eds.): *Goal-Driven Learning*, The MIT Press, 1995.

[Minton et al., 1989]
S. Minton, S. Carbonell, C. Knoblock, D. R. Kuokka, O. Etzioni, and Y. Gil: Explanation-based Learning: A Problem Solving Perspective, *Artificial Intelligence*, 40:63—118, 1989.

[Motta, 1997]
E. Motta: *Reusable Components for Knowledge Modeling*, Ph.D. Thesis, Knowledge Media Institute, The Open University, UK, 1997.

[Motta & Zdrahal, 1996]
E. Motta and Z. Zdrahal: Parametric Design Problem Solving. In *Proceedings of the*

10h Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´96), Banff, Canada, November 9-15, 1996.

[Muggleton & Buntine, 1988]
S. Muggleton and W. Buntine: Machine Invention of First-Order Predicates by Inverting Resolution. In *Proceedings of the 5th International Conference on Machine Learning (ICML-88)*, Michigan, US, 1988.

[Muggleton & De Raedt, 1994]
S. Muggleton and L. De Raedt: Inductive Logic Programming: Theory and Methods, *Journal of Logic Programming*, 19/20:629—679, 1994.

[O´Hara & Shadbolt, 1996]
K. O'Hara and N. Shadbolt: The Thin End of the Wedge: Efficiency and the Generalized Directive Model Methodology. In N. Shadbolt (eds.), *Advances in Knowledge Acquisition,* Lecture Notes in Artificial Intelligence (LNAI) 1076, Springer-Verlag, Berlin, 1996.

[Popper, 1959]
K. R. Popper: *The Logic of Scientific Discovery*, London, 1992 (reprint of 1959).

[Reif, 1995]
W. Reif: The KIV Approach to Software Engineering. In M. Broy and S. Jähnichen (eds.): *Methods, Languages, and Tools for the Construction of Correct Software*, Lecture Notes in Computer Science (LNCS) 1009, Springer-Verlag, Berlin, 1995.

[Reif & Stenzel, 1993]
W. Reif and K. Stenzel: Reuse of Proofs in Software Verification. In Shyamasundar (ed.), *Foundation of Software Technology and Theoretical Computer Science*, LNCS 761, Springer-Verlag, 1993.

[Schreiber & Birmingham, 1996]
A. Th. Schreiber and B. Birmingham (eds.): *Special Issue on Sisyphus, International Journal of Human-Computer Studies (IJHCS)*, 44(3-4), 1996.

[Shapiro, 1982]
E. Y. Shapiro: *Algorithmic Program Debugging*, The MIT Press, 1982.

[Shaw, 1989]
A. Shaw: Reasoning About Time in Higher Level Language Software, *IEEE Transactions on Software Engineering*, 15(7):875—889, 1989.

[Smith, 1985]

D. R. Smith: Top-Down Synthesis of Divide-and-Conquer Algorithms, *Artificial Intelligence*, 27:43—96, 1985.

[Smith, 1996]
D. R. Smith: Towards a Classification Approach to Design. In *Proceedings of the 5th International Conference on Algebraic Methodology and Software Technology (AMAST-96)*, Munich, Germany, July 1-5, 1996.

[Stefik, 1995]
M. Stefik: *Introduction to Knowledge Systems*, Morgan Kaufman Publ., San Francisco, 1995.

[Straatman & Beys, 1995]
R. Straatman and P. Beys: A Performance Model for Knowledge-based Systems. In M. Ayel et al. (eds.): *EUROVAV-95 European Symposium on the Validation and Verification of Knowledge Based Systems*, University of Savoie, Chambery, June 26-28, 1995.

[ten Teije, 1997]
A. ten Teije: *Automated Configuration of Problem Solving Methods in Diagnosis*, Ph.D. thesis, University of Amsterdam, the Netherlands, 1997.

[van Harmelen & Balder, 1992]
F. van Harmelen and J. Balder: $(ML)^2$: A Formal Language for KADS Conceptual Models, *Knowledge Acquisition*, 4(1), 1992.

[van Harmelen & ten Teije, 1998]
F. van Harmelen and A. ten Teije: Characterizing Problem Solving Methods by Gradual Requirements: Overcoming the Yes/No Distinction. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´98)*, Banff, Canada, April 18-23, 1998.

[Wielinga et al., 1995]
B. Wielinga, J. M. Akkermans, and A. TH. Schreiber: A Formal Analysis of Parametric Design Problem Solving. In *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´95)*, Banff, Canada, January 26 - February 3, 1995.

[Wordsworth, 1996]
J. B. Wordsworth: *Software Engineering with B*, Addison-Wesley, 1996.

[Zdrahal & Motta, 1995]
Z. Zdrahal and E. Motta: An In-Depth Analysis of Propose & Revise Problem Solving

Methods. In *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´95)*, Banff, Canada, January 26 - February 3, 1995.

[Zilberstein, 1996]
S. Zilberstein: Using Anytime Algorithms in Intelligent Systems, *AI Magazine*, 17(3), 1996.