Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern

SEKI - REPORT

(A)TMS in Expert Systems

Jürgen Paulokat (Ed.)

SEKI Report SR-92-06 (SFB)

Preface

This report contains a collection of abstracts of talks given during the workshop:

(A)TMS in Expert Systems.

The workshop was hold at University of Kaiserslautern in june, 1991.[1] It was the intention of the workshop to discuss the benefice and problems of integrating reason maintenance systems to support problem solving in expert systems.

Reason maintenance systems has been proposed more than a decade before. There is a lot of theoretical work on complexity, semantic, and efficiency presented on numerous conferences and workshops. Given the theoretical results reason maintenance seems to have a great potential for facilitating problem solving. But in practical work a lot of problems arise trying to integrate reason maintenance systems and problem solver. Some of these problems are described in this report and some solutions are presented to them.

Jürgen Paulokat (ed.)

[1]This workshop was partially founded by the Deutsche Forschungsgemeinschaft (DFG), SFB 314 "Artificial Intelligence — Knowledge Based Systems", project X9

Contents

Integration of a Reason Maintenance System into a Hybrid Computer Vision Tool

J. Dvorak, H. Bunke
Universität Bern
Institut für Informatik und angewandte Mathematik
Länggassstrasse 51, CH-3012 Bern, Switzerland
Email: dvorak@iam.unibe.ch , bunke@iam.unibe.ch

extended Abstract

Introduction

There are many classes of problems in computer vision where explicit a priori knowledge about the possible meaning of the images under consideration is required. Consequently, many knowledge based approaches to computer vision have been proposed in recent years. They mainly rely on the artificial intelligence based knowledge representation formalisms rules, frames, and predicate logic, and on other symbolic methods like structural prototype matching, discrete and continuous relaxation. Recently, reason maintenance has been proposed as suitable means for the interpretation task in computer vision [BSB89, Pro88, Pau88]. In the following we present a hybrid knowledge representation and reasoning tool that offers the possibility to combine rules, frames, relaxation, prototype graph matching and reason maintenance in a uniform and coherent style. This hybrid shell is particularly useful for, but not restricted to, applications from computer vision and pattern recognition. A prototype of the software tool without reason maintenance has been implemented in Common Lisp and the Common Lisp Object System CLOS [BDG+88, DB91, DB90]. The focus of this paper is on the ongoing integration of the reason maintenance system, with only a short overview of the other components of the tool in the next paragraph.

Overview of the Vision Tool

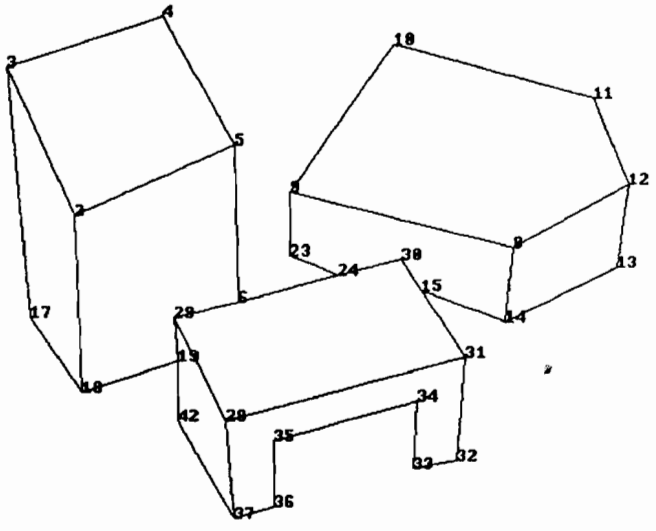
The tool is implemented in an object-oriented style with the frame system as an extension of the CLOS class/instance system, using the Metaobject Protocol (MOP) of CLOS [BDG+88]. It supports particularly hybrid approaches, where the knowledge is distributed among multiple representation methods. The components frames, rules (both forward and backward chaining), discrete relaxation and prototype matching are the knowledge representation and reasoning methods offered by our tool, and they can be instantiated multiple times in one application. To support hybrid applications, the components have a coherent user interface, and there are provivions for interactions among the components such that every instance of a component can interact with any other instance. To cope with a large variety of computer vision applications and with future needs, another important feature of the tool is its flexibility in terms of adaptability and extensibility.

This flexibility is achieved by the separation of a programmer and a metalevel interface, similar to the two interfaces of CLOS.

Integration of a RMS

The integration of a reason maintenance system with the vision tool consists of two basic tasks: the development of the reason maintenance system itself, in our case an ATMS [dK86a], and its integration into the existing four components. Two typical applications of the reason maintenance system are described below:

- Handling of multiple contexts, similar to the ART Viewpoints or KEE Worlds systems, for reasoning under a set of hypotheses. This application is illustrated by means of an example. A sample scene with three overlapping objects is shown in the figure below:



In order to identify the objects in the image a possible application might start with an attempt to separate the objects according to the types of the connections of lines. In particular, so-called T-connections, where one line meets two lines with approximately the same slope, are good candidates for object separation. The interpretation of a T-connection, e.g. point 15 in the above image, as a point where two objects meet is then an assumption, and combinations of these assumptions are the environments describing the contexts in which a prototype matching can be called for the identification of one or all separated objects.

Generalizing from this example, there are typically a few critical features in an image that form the hypotheses, and reasoning then proceeds with various processing steps, including calls to other components or Lisp functions, within the contexts formed by the assumptions.

- The second type of applications involves an ATMS problem solver in the sense of [dK86b, Jun89] for more direct interactions with the underlying ATMS. A typical example of this kind of use is the puppet interpretation system of Provan [Pro88]. In such an application, the tasks are completely solved within the reason maintenance system by consumers (rules) supplying justifications to the system. The image

interpretation proceeds in a purely deductive manner, without any call to another reasoning component.

There are a number of problems encountered with the integration of such a reason maintenance facility into our hybrid reasoning tool. With respect to the ATMS algorithm, there exist various extensions that one may need or not, providing negation, nonmonotonic justifications, default reasoning, and others (e.g. [Dre88]). In most cases the problem solver part as proposed by [dK86b] is not sufficient and has to be extended by attached expressions [Jun89]. But the most problems arise from the interaction of such a reason maintenance component with the other four components of the tool. This is due to the fact that these other components reason in a non-deductive way, with frequent modifications to the database. This kind of reasoning violates basic properties of the ATMS, and the ATMS has to be adapted to retain its reason maintenance capabilities. Various versions of similar multiple context handlers have been implemented (ART,KEE,Except [Jun89]). All these systems have certain limitations or drawbacks, based on extensions and adaptations or performance improvements that are traded in for restrictions in other parts of the reason maintenance formalism.

For our implementation, we have chosen a basic ATMS with a problem solver extended by attached expressions and a MCH similar to the KEE Worlds system as described in [MN86]. The problem solver part is a specialization of the forward chaining rule system in our tool, with restrictions on conditions and conclusions. The integration with the other tool components is based on the integration with the frame system. In our tool, all data is represented in frames, which are extensions of CLOS objects. Thus the task is to integrate CLOS objects with the ATMS. This is achieved with extensions to the slot definition and accessor protocol of CLOS. Although it can make sense to create instances based on an actual context, we have decided that slots are the only data elements that can be mapped to RMS nodes. The rules, relaxation, and prototype matching components are invoked with an optional environment parameter and operate in the context of this starting environment. Additionally, rule systems can be configured to operate in all extensions of the starting environment, performing an incremental descent in the context graph during rule execution.

The scope of this paper is too limited to give more details about the integration of the reason maintenance system. At the time of writing the integration of the reason maintenance facility into our hybrid computer vision tool is in its specification phase.

Conclusions

Although there are some existing applications in computer vision where reason maintenance has been successfully used, it is our impression that, regarding its potential, reason maintenance has not received adequate attention in image interpretation and similar domains. One of the reasons for this missing acceptance is certainly the high initial effort required to add reason maintenance to an application. Thus, our conclusion is basically a proposition for a framework and a toolkit for reason maintenance. A general framework for the integration of reason maintenance in tools like ours or in standalone applications in computer vision or similar artificial intelligence based domains would be very useful. Apart from hybrid tools, reason maintenance as a utility program should be avaible in toolkits or software libraries in a modular way such that it can be easily added to various applications.

References

[BDG+88] D.G. Bobrow, L.G. DeMichiel, R.P. Gabriel, S.E. Keene, G. Kiczales, and D.A. Moon. *Common Lisp Object System Specification.* X3J13 Document 88-002R, 1988.

[BSB89] R. Bodington, G.D. Sullivan, and K.D. Baker. Consistent labelling of image features using an assumption-based truth maintenace approach. *Image and Vision Computing*, 7(1):43–49, 1989.

[DB90] J. Dvorak and H. Bunke. Using CLOS to implement a hybrid computer vision tool. In A. Paepcke, editor, *Proceedings of the Third CLOS Users and Implementors Workshop*, pages 24–31, 1990.

[DB91] J. Dvorak and H. Bunke. A hybrid expert system shell for computer vision. In *To appear in: Proc. of the First World Congress on Expert Systems*, Orlando, Florida, 1991.

[dK86a] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

[dK86b] J. de Kleer. Problem solving with the ATMS. *Artificial Intelligence*, 28:197–224, 1986.

[Dre88] O. Dressler. An extended basic atms. In M. Reinfrank, J. de Kleer, M.L. Ginsberg, and E. Sandewall, editors, *Non-Monotonic Reasoning*, pages 143–163. 1988.

[Jun89] U. Junker. EXCEPT: A rule-based system for multiple contexts inconsistencies, and exceptions. Technical Report 371, Arbeitspapiere der GMD, 1989.

[MN86] P.H. Morris and R.A. Nado. Representing actions with an assumption-based truth maintenance system. In *AAAI-86*, pages 13–17, 1986.

[Pau88] L.F. Pau. Knowledge representation for three-dimensional sensor fusion with context truth maintenance. *NATO ASI Series, Vol. F42 Real-Time Object Measurement and Classification. Berlin Springer*, pages 391–404, 1988.

[Pro88] G.M. Provan. Model-based object recognition - a truth maintenance approach. *Proc 4th CAIA 88*, pages 230–235, 1988.

Assumption Based Truth Maintenance on Partially Ordered Data

Albert Haag
Battelle Institut
Am Römerhof 35, D-6000 Frankfurt a.M. 90
West Germany
tel. (++49-69-7908-2559)

1 Summary

We describe an ATMS[1]-based problem solving architecture dedicated to support certain classes of configuration and planning applications. These applications largely involve solving constraint satisfaction problems, with an added optimization aspect introduced implicitly in the form of weak constraints. Problem solving is a sort of best-first search, mainly controlled (interactively) by the user. Our architecture supports this setting with several specific features

- The backbone of the system is the ATMS*, an Assumption Based Truth Maintenance System (DeKleer [1]), that was extended by us to respect so-called specialization relations. By specialization relations we mean partial orders on the domain data that express specificity of meaning of the data (not preference relations). (See section four, below.)
- The ATMS* is integrated with a rule based inference engine that provides a pattern matching capability over the facts the ATMS* manages. This can be seen as special instance of DeKleer's class consumer architecture. Both the ATMS* and the rule interpreter are integrated with an object oriented programming system[2] (OOP).
- Constraints are implemented as pattern matching rules. The rules allow formulating both hard and soft constraints. It is possible to declare problem variables, and to reason about domain restrictions for these variables.
- The role of problem solver (PS) assumptions is clarified. Justifications derived from soft constraints are in a one to one correspondence with the ATMS* assumptions. Or, conversely, assumptions are generally interpreted as soft justifications.
- The ATMS* can be queried interactively. To support such queries, we introduce the concept of an ATMS*-universe. An ATMS*- universe is here taken to be the set of all facts consistent with a given ATMS*-label.

A central result of our work concerns extending the a specialization relation on the domain data to different sets of logical formula that arise in connection with the ATMS*, particularly to justifications and conjunctions of facts. We can only briefly outline our approach here, a more complete account is given in [2].

---

1 We assume familiarity with basic concepts connected with an ATMS (Assumption Based Truth Maintenance System) as introduced by DeKleer [1] such as label, environment, assumption, etc.

2 Our system, called μPLAKON, is currently implemented in Common Lisp; the OOP chosen was CLOS [7].

2 Problem Setting

From 1986-1990 we were engaged in developing and applying expert system tools for solving real-world configuration and planning problems within the TEX–K project[3]. Of all the applications considered there, we shall mention two, which we deem to be typical: the derivation of production plans for the machining of metal parts (c.f. [3]), and the configuration of micro-computer hardware (c.f. [2], [4]). It turned out that the basic problems that needed to be solved in these applications are constraint satisfaction problems (CSPs) with an added optimization aspect [3].

The problem solving process we envision in this setting consists of a sequence of decisions that successively restrict the domains of the problem variables, until a satisfactory solution has been found. At each step of this process the problem solver (PS) usually has a choice of several different decisions about domain restrictions that can be made, leading to potentially different solutions that also vary in their degree of optimality. If the PS has a heuristic estimate of the utility (or cost) that can be attained based on decisions already made, this can be used to guide the solution finding process in a best-first manner. Unfortunately, in our experience, it is sometimes difficult or impossible to explicitly formulate such a utility (or cost) function. Instead, it turns out that optimality of solutions can also be expressed implicitly, via so-called *soft constraints*. A soft constraint states a desirable state of affairs (because good solutions often fulfill these constraints), but it can be relaxed when in conflict with other constraints. Consider the following two rules taken from the machining application referred to above.

R1: *"For any feature that must be both rough-machined and fine-machined, rough-machining always precedes fine-machining"*

R2: *"Any rough-machining should always precede any fine-machining"*

R1 translates into a hard constraint; there are technical reasons why it is not plausible to do rough operations (such as milling) after fine ones (such as polishing). *R2* corresponds to a soft constraint. It often characterizes good process plans, because it reduces the number of necessary retoolings, but there is no technical reason why it could not be violated.

Thus, not only must a CSP be solved, but the right set of constraints must be chosen from a potentially very large set. For lack of a better term at present, we shall call such problems *extended CSPs* (ECSPs). Whereas the complexity of ECSPs can be exponential, we found that one means of achieving successful (commercial) support, here, lies in providing a sort of 'truth maintenance' functionality. The user (or external PS) guides the search for a satisfactory solution, and the system helps juggle alternatives while ensuring the local satisfaction of the hard constraints.

3 Local Satisfaction of Constraints

The approach to CSPs in connection with ATMSs taken most frequently is to encode each assignment of a value to a problem variable as a proposition recorded directly by an ATMS-node (usually as an assumption) (c.f. [5]). This entails reasoning about disjunctions of such nodes,

[3] TEX–K was a joint venture sponsored in part by the German Ministry for Research and Technology (BMFT). The results referred to in this paper reflect only a part of the total effort of the project. (C.f. [10])

which in turn necessitates using an extended ATMS with hyperresolution. We tried a different approach that avoids hyperresolution. The main idea was to represent domain restrictions for the problem variables as ATMS-nodes and to ensure the local satisfaction of the constraints. A domain restriction associates a problem variable x with a set of values S that are candidates for assignment and will be denoted as $\{x \in S\}$. (Note that assigning a value to a problem variable is a special case of restricting the variables domain.) If the PS knows that both $\{x \in S\}$ and $\{x \in T\}$ hold in a given state, and $S \subset T$, then $\{x \in T\}$ is obviously uninteresting. We call $\{x \in S\}$ a *specialization* of $\{x \in T\}$. These specializations induce a partial ordering on the domain data, the so-called *specialization relation*. More generally, given two formulas φ and ψ, we consider φ to be more special than ψ if the PS will never be interested in ψ in the presence of φ. We feel specialization relations occur naturally in practical problem solving and provide an important means for minimizing the problem solving process (c.f. [2]).

Given a domain restriction for each of the problem variables we can apply a *local propagation scheme* [7] to achieve *local satisfaction* of the constraints. In this scheme individual constraints are used to filter out impossible values from the domain restrictions of the variables constrained by the constraint (thus further restricting the domain of these variables). Any newly found restrictions can be used as input for further filtering at other constraints. Although it can be shown that local propagation is a comparatively weak problem solving method, it helps to prune the search space, providing firmer ground for applying the soft constraints (heuristics).

Local propagation can be implemented using inference rules in a straightforward manner. Consider a constraint c that imposes restrictions on the simultaneous value assignments of, say, the variables x, y, and z. Let f_x, f_y, and f_z be the local filtering functions induced by c. Then the following inferences are justified by c.

$$\{x \in S_x\}, \{y \in S_y\}, \{z \in S_z\} \rightarrow \{x \in f_x(S_x,S_y,S_z)\}$$

$$\{x \in S_x\}, \{y \in S_y\}, \{z \in S_z\} \rightarrow \{y \in f_y(S_x,S_y,S_z)\}$$

$$\{x \in S_x\}, \{y \in S_y\}, \{z \in S_z\} \rightarrow \{z \in f_z(S_x,S_y,S_z)\}$$

Also, we can formulate the following inferences, independent of the problem domain:

$$\{x \in S\}, \{x \in T\} \rightarrow \{x \in S \cap T\}$$

$$\{x \in \varnothing\} \rightarrow \textit{contradiction}.$$

4 ATMS* – An ATMS that Handles Specialization

ATMSs were introduced by DeKleer [1] to help the PS manage dependencies of data on assumptions made during the of problem solving process by focusing attention on the data relevant to a given situation. An ATMS can be characterized by three sets, namely the set of domain data (denoted by D), the set of assumptions (denoted by A), and the set of justifications (denoted by J). We require the D to contain a special datum $\perp$ that represents falsity. As noted above, in the ATMS* assumptions always directly refer to soft constraints. We feel this greatly clarifies and

eases the use of assumptions in problem solving; thus, the sets D and A are considered disjoint. We interpret a justification as a propositional implication $\xi \rightarrow d$, where ξ is a (possibly empty) conjunction of domain data and d is a domain datum. Given this interpretation "*derivability w.r.t. J*" is defined, which we denote as usual by $\vdash_J$. A set of assumptions $E \in 2^A$ is called an environment. We interpret an environment as the conjunction of its assumptions.

The major computational task of an ATMS is the computation of a *label* for each domain datum $d \in D$. The label $\lambda_{d,J}$ of a datum $d \in D$ will contain exactly the minimal and consistent environments that allow deriving d, and is defined as follows. (Note that set inclusion defines a natural partial order on 2^A.)

$$\bar{\nu}_J := \{E \in 2^A \mid E \vdash_J \bot\}$$

$$\bar{\lambda}_{d,J} := \{E \in 2^A \mid E \vdash_J d\} \setminus \bar{\nu}_J$$

$$\lambda_{d,J} := min\ \bar{\lambda}_{d,J}$$

It was noted above that clients of the ATMS* (e.g. the user, the PS, and the class consumer architecture) are always interested in the most specific applicable information. This leads to a modified labelling that accounts for specialization relations. Let $(D, \leq)$ be a specialization relation on the set of facts. For $d \leq e$ the PS is not interested in e in the presence of d. Thus, if for any $E \in 2^A$ we have both $E \vdash_J d$ and $E \vdash_J e$, the latter will no longer be of interest, and the label of e can be changed to reflect this, by removing E or any superset. We now define the ATMS* label $\lambda^*_{d,J}$ incorporating a specialization relation. We take $\bot$ to be the most special datum: $\forall d \in D \ \ \bot \leq d$.

$$\nu^*_{d,J} := \bigcup_{e<d} \{E \in 2^A \mid E \vdash_J e\}$$

$$\bar{\lambda}^*_{d,J} := \{E \in 2^A \mid E \vdash_J d\} \setminus \nu^*_{d,J}$$

$$\lambda^*_{d,J} := min\ \bar{\lambda}^*_{d,J}$$

Note that for the trivial partial order defined by

$$d \leq e \ \Leftrightarrow \ (e = \bot \vee d = e)$$

the ATMS and ATMS* labels coincide, apart from the technical difference that $\lambda_{\bot,J}$ is empty, whereas $\lambda^*_{\bot,J} = min\ \bar{\nu}_J =: \nu_J$ which is just the set of *nogoods* in the terminology of DeKleer.

References

[1] DeKleer, J.: "An Assumption Based TMS", Artificial Intelligence 28 (1986), pp. 127–162

[2] Haag, A.: "Der regelorientierte Kontrollansatz in PLAKON: Konzepte zur praktischen Handhabbarkeit einer ATMS-basierten Problemlösung" TEX-K Report No. 26 (1990)[4].

[3] Haag A., Zetzsche F., Zinser G. "Die Behandlung von Alternativen in der Planung: Erfahrungen mit ATMS-basierten Expertensystemarchitekturen" in Hertzberg, Günther (Hrsg):

[4] An abbreviated version of this report is contained in Cunis, Günter, Strecker (Hrsg.) "Das PLAKON Buch", Informatik Fachberichte Band 266, Springer Verlag, 1991, pp. 212-237.

Proc. II. Workshop 'Planen und Konfigurieren', GMD Arbeitspapiere 1988

[4] Baginsky W., Philipp L. "MMC-Kon: Ein wissensbasiertes CAE-Werkzeug zur Projektierung verteilter Leitsysteme" in Cunis, Günter, Strecker (Hrsg.) "Das PLAKON Buch", Informatik Fachberichte Band 266, Springer Verlag, 1991, pp. 197-211.

[5] DeKleer, J.: "A Comparison of ATMS and CSP Techniques", Proc. IJCAI-89, pp. 290–296

[6] Keene S. "Object-Oriented Programming in COMMON LISP – A Programmer's Guide to CLOS" Addison-Wesley 1989

[7] Güsgen H. W. "CONSAT — A System for Constraint Satisfaction" Dissertation Universität Kaiserslautern 1988

Distributed Truth Maintenance[1]

EXTENDED ABSTRACT

Thilo Ç. Horstmann

German Research Center for AI (DFKI)
Project KIK
P.O. Box 2080
W-6750 Kaiserslautern
Germany

e-mail: *horstman@dfki.uni-kl.de*

May 1991

1 Introduction

Recent research in the field of Distributed Artificial Intelligence (DAI) has led to a broad variety of applications characterized by autonomous, loosely connected problem solving nodes. Each single node, or *agent*, is capable of individual task processing and able to coordinate its actions in combination with those of other agents in the net. DAI applications span a large field ranging from cooperating expert systems, distributed planning and control, to human computer cooperative work. In order to establish a domain independent theory of interacting autonomous agents, current DAI research focuses on defining an abstract agent model, which allows formalizing of cooperation strategies and multi agent reasoning mechanisms.

The requirements of multi agent reasoning algorithms are manifold. In most cases, it is *not* desirable to constrain the autonomy of agents by building a 'superstrate reasoner' managing all inferences or rules of a set of different agents. The reasons are discussed fully in [DLC89]. Instead, we want the agents to be able to reason autonomously; in particular, a single agent must deal with beliefs, which have probably been created in a complex cooperation process.

This requirement is best fulfilled by providing an agent with a Distributed Truth Maintenance System (DTMS)[2]. Based on classical TMS theories, distributed truth maintenance extends the conventional case to make reason maintenance suitable for multi agent scenarios. A DTMS has to represent and manage inferences and rules of interacting agents in a way that ensures a specified degree of consistency. The various degrees of consistency will be defined in this paper. Furthermore, other modules of an agent should be able to use information stored by the DTMS. For instance, a problem solving unit may avoid recomputation or a cooperation process might be based on the current context of the consistent knowledge base.

---

[1]This work was done in Project KIK at the German Research Center for Artificial Intelligence (DFKI). Project KIK is a collaborative effort between the DFKI and Siemens AG.

[2]We use the term Truth Maintenance System instead of the perhaps more appropriate term Reason Maintenance System or Belief Revision System. This is done for historical reasons.

We start off by presenting a Truth Maintenance System designed for backward reasoning systems. We shall show that the properties offered by a TMS for forward reasoning systems, can also be used by backward reasoning systems. Further, the amalgamation of the *incrementality* and *selectivity* of a justification based TMS with the properties of backward reasoning allows elegant and efficient programming techniques in a first-order logic representation.

The basic key features of the DTMS are summarized below:

- maintenance of a consistent state of beliefs. Because we record data dependencies, checking consistency involves little recomputation when the knowledge base is modified.
- data dependencies are recorded in the Horn subset of first-order predicate logic instead of propositional logic.
- explicit representation of proofs allows for easier generation of explanations.
- interface for exchanging beliefs, data and proofs among agents.
- meta level predicates aiding the design of clearly specified autonomous agents. In addition, it simplifies the classification of goals into those upon which reason maintenance should be performed and those which remain static.
- the DTMS is designed as a generalization of a TMS. As a result, the application domain is not restricted to the field of DAI.

In these terms, the results of this work may be divided into two main sections. Section 2 discusses TMS techniques tailored for backward chaining resulting in a *Backward Reasoning Truth Maintenance System (BRTMS)*. This section is not specific to the area of DAI. In Section 3, the BRTMS of Section 2 is extended to the distributed case. We define central terms concerning multi agent reasoning and illustrate the DTMS algorithm.

2 A JTMS for Backward Reasoning Systems

Former TMSs have been designed for use with incremental forward reasoning systems. In a forward reasoning system, each inference step produces new conclusions from antecedent data, which can be passed to the TMS. In contrast, in a backward reasoning system each inference step does not produce new conclusions, rather new conditions for the goal[3]. To make conclusions, we have to wait until the reasoning process is complete. In these terms, the problem solver would have to keep track its inferences, in order to transmit appropriate data to a classical TMS.

However, the designer of the problem solver should not have to think about how to represent inferences. Our system relieves the system designer from this task, all inference control is done by meta logic predicates in the BRTMS. This implies a different BRTMS architecture: The BRTMS *Meta Level* embeds the lower level *Kernel*. The BRTMS Meta Level includes meta logic predicates, user defined justifications, the current state of beliefs, the BRTMS Kernel system predicates and user defined static predicates. The meta level controls the evaluation of all goals, performs the bookkeeping of results and defines the BRTMS interface while the kernel defines low level predicates: predicates, which might be evaluated through a meta call, but whose proof is not significant for the BRTMS bookkeeping mechanism. The meta logic predicate `dtms_solve/5` plays a central role in the meta level. The definition of `dtms_solve/5` realizes a modification of a standard Prolog meta

[3]We use the terminology of logic programming as introduced in [Llo84].

interpreter. At first sight, this interpreter takes as an argument a Prolog query q and tries to find a proof for q in accordance with clauses of the kernel, the meta level and with the current set of beliefs. In the course of doing this, all data dependencies are stored or updated as necessary. One important feature concerning this meta concept should be mentioned at this point: The designer of BRTMS applications is freed from creating data dependencies, all dependencies are implicitly defined by justifications.

Justifications are defined in the meta level. These are dynamic program clauses defining the atomic formulas (or *atoms*), on which truth maintenance will be performed. In former JTMSs, justifications are –in a different form– the only kind of rules. But we will see when considering BRTMS applications that the combination of a TMS with a Prolog problem solver increases the TMS functionality by allowing for system predicates. As mentioned before, these predicates are also evaluated by `dtms_solve/5`. Furthermore, we will see that there is a whole class of predicates that should be evaluated in the same manner as system predicates. These are predicates that are never be altered such as `member/3` or `append/3`. Obviously, there is no point in performing truth maintenance on those predicates. Because of these reasons, we define the BRTMS Kernel. In the kernel all predicates of the meta level are invisible, but the meta level can evaluate predicates defined here. The proof tree of the result of a kernel call will not be stored. In other words, you may regard the kernel may be regarded as the 'static true world' and the meta level as the 'dynamic changing world'.

3 Extension of the BRTMS to Distributed Truth Maintenance

We present a way to establish reasoning among interacting autonomous agents. We define an abstract terminology that clearly specifies basic terms of multi agent reasoning. The central term is that of *proof consistency*. In contrast to previous attempts, this definition of consistency in multi agent scenarios is characterized by exchanging beliefs as well as exchanging reasons for the beliefs. We can see this in everyday life: When debating an issue, we do not want to know *what* somebody claims, but, in addition *why* he claims it. Furthermore, we usually agree with somebody only if we agree with him in his conclusion *and* in the foundations of his conclusion.

In these terms, interacting agents, which exchange beliefs along with the corresponding foundations, reason much more flexibly than agents which only transmit the results of inferences. If an agent later invalidates the foundation of an acquired belief, it might reconsult the agent from which the belief was originally acquired.

But, we do not want to overwhelm an agent with too much information by transmitting complex traces of inferences between agents. We will show that it is sufficient to transmit only a special representation of proofs and not the whole proof structure. In addition, the designer of multi agent scenarios can specify a *level of consensus*, which defines the knowledge upon which the agents will agree all the time. This feature can greatly improve efficiency in multi agent reasoning.

We do not force the agents to agree on all information - our notion of a proof consistent state allows agents to be partially inconsistent with one another. That is, agents might have different viewpoints of certain beliefs. If two agents reason together to solve the query "Can Tweety fly?" it is irrelevant if the agents disagree about matters which have no bearing on this question. Allowing certain inconsistencies can keep the information exchange between agents to a minimum with respect to the current task.

In order to formalize a multi agent reasoning process, we introduce the following terms. In that, an agent is defined by a set of program clauses $\mathcal{P}$, a set of beliefs $\mathcal{B}$ and the labeling of beliefs, called the *state* Ψ.

Let $\mathcal{A} = \{\alpha_1 = (\mathcal{P}_1, \mathcal{B}_1, \Psi_1), \cdots, \alpha_n = (\mathcal{P}_n, \mathcal{B}_n, \Psi_n)\}$ be a set of agents. We say a belief $b_i \in \mathcal{B}_i$, denoting the atom l_i, is

(i) *private* to α_i, if there is no belief b_j in $\mathcal{B}_j$, such that l_i can be unified with l_j, $(i \neq j)$.

(ii) *common* to α_i and α_j, if there is a belief b_j in $\mathcal{B}_j$, such that l_i can be unified with l_j, $(i \neq j)$. The status of b_i might be different from the status of b_j.

(iii) *transmitted* to agent α_j, if $\mathcal{P}_j$ contains either a positive agent rule of the form $l_i \leftarrow \alpha_i$ or a negative agent rule of the form $l_i \leftarrow \neg\alpha_i$ $(i \neq j)$.

(iv) *acquired* from agent α_j, if $\mathcal{P}_i$ contains either a positive agent rule of the form $l_i \leftarrow \alpha_j$ or a negative agent rule of the form $l_i \leftarrow \neg\alpha_j$, $(i \neq j)$.

(v) *mutual* to α_i and α_j, if b_i is transmitted to α_j.

Note, that we distinguish *transmitted* and *acquired* beliefs. *Shared beliefs* as introduced in [BH90], are the union of transmitted and acquired beliefs.

Informally[4], the state of beliefs in a multi agent scenario is *proof consistent*, if

1. each agent is locally consistent.
2. if a belief is transmitted to another agent, then the foundations of this belief are also transmitted. A transmitted belief and the acquired counterpart is either `in` or `out`.
3. An agent cannot transmit a belief that it has already acquired.
4. the set of mutual beliefs is *well founded.*

This concept clearly defines a state of consistency of mutually dependent beliefs across different agents. This state is characterized by exchanging inferences *and* their foundations. We showed that in contrast to previous approaches, our definition of consistency allows agents to reason in a more complex way. Information, lost in former approaches, will now be propagated to all relevant agents: Because one agent knows the foundations of an acquired inference of another agent, it can inform this agent when a foundation becomes invalid. We showed the agents will not be overwhelmed with information. It is sufficient only to exchange a special, minimal representation of inferences between agents.

References

[BH90] D. M. Bridgeland and M. N. Huhns. Distributed truth maintenance. In *Proc. of AAAI-90*, pages 72–77, Boston, MA, 1990.

[DLC89] Edmund H. Durfee, Victor R. Lesser, and Daniel D. Corkill. Trends in cooperative distributed problem solving. In *Transactions on Knowledge and Data Engineering*, 1989.

[Hor91] Thilo C. Horstmann. Distributed truth maintenance. Master's thesis, University of Kaiserslautern, 1991. DFKI-Document D-91-11.

[Llo84] John W. Lloyd. *Foundations of Logic Programming.* Springer, 1984.

[4] A more formal definition is presented in [Hor91]

Generating Diagnoses by Prioritized Defaults*

Ulrich Junker
GMD
P O Box 1240
5205 St. Augustin 1
Fed. Rep. of Germany
++49 - 2241/142671
junker@gmdzi.gmd.de

Abstract

Preferences between diagnostic assumptions allow to control what kinds of assumptions are retracted in presence of a conflict and help to focus the diagnosis process to candidates that should be considered first. For example, we want to prefer the correct behaviour to the failure modes and mechanical faults to electrical ones. We use prioritized default theories to define preferred diagnoses based on these preferences on assumptions. Preferred diagnoses can easily be constructed using TMS-networks for prioritized default theories. Hence, generation of all candidates is avoided.

1 The Need for Preferences

Diagnosis is the process of finding malfunctioning parts or diseases that cause some observed symptoms. Normally, we obtain several candidates and must acquire further observations to discriminate between them. Hence, diagnosis consists of a cycle of observation, candidate generation, and selection of further observables.

To generate candidates, different kinds of methods have been developed including model-based diagnosis, abductive approaches, and heuristic approaches. *Integrating* them properly is currently an interesting research topic. A second topic is *focussing* the diagnosis process to candidates that should be considered first according to probability, danger, or measurement costs. In this abstract, we show how preferences between diagnostic assumptions are helpful for the first problems:

control retraction: Preferences allow to control the selection of diagnostic assumptions. They specify what assumptions may be retracted in presence of conflicts. For example, we want to avoid that correct modes are retracted by failure modes and that basic diagnostic principles such as the single-fault hypotheses are retracted by a double fault even when there is also a single fault.

In this sequel, we discuss how priorities on assumptions help to control the arbitrary interaction of assumptions in diagnosis. First, we introduce a simple framework for diagnosis.

2 Framework for Diagnosis

We formulate our diagnostic problems and theories in a first-order language $\mathcal{L}$ containing an unsatisfiable constant $\perp$ and predicates $=, +, -, *$ for equality and arithmetics.

*A long version of this paper has been presented at the Second International Workshop on Principles of Diagnosis, Milano, 1991.

A system for generating diagnoses is supplied with observations and should find possible faults. Hence, a diagnostic problem consists of:

1. a set $\mathcal{O} \subseteq \mathcal{L}$ of observations. It may contain atomic formulas if a single test is performed or disjunctions of several test results. Sometimes, the observations are divided into those that are asserted as premises and must not be refuted and those that must explicitly be derived. Hence, $\mathcal{O}$ is split into two sets $\mathcal{O}^-$ and $\mathcal{O}^+$. This distinction due to Console and Torasso [Console and Torasso, 1990].

2. a set $\mathcal{F}$ of ground literals from $\mathcal{L}$ representing the possible faults. It can contain diseases or abnormalities of components.

We define a *diagnostic problem* DP by a tuple $(\mathcal{O}^-, \mathcal{O}^+, \mathcal{F})$. To solve it, different methods including model-based and heuristic diagnosis, as well as consistency-based and abductive approaches can be used. In all cases, we can distinguish safe and hypothetical knowledge. The first one is represented by a set W of first-order formulas that are asserted as premises. Examples are the system descriptions and general laws of behaviour. Hypothetical knowledge is represented by assumptions, which are ground atomic formulas, and their definitions that are also asserted as premises. Examples for assumptions are:

- applicability of heuristic rules. For example $r1(car1)$ in a heuristic rule $\forall x.r1(x) \wedge car(x) \wedge \neg starts(x) \supset ab(battery(x))$ saying that the battery of a car is probably defect if it does not start.

- correctness of the normal behaviour. For example, $ok(A1)$ expresses that adder $A1$ works correctly: $\forall x.adder(x) \wedge ok(x) \supset in1(x) + in2(x) = out(x)$.

- failure modes. For example, $shorted(i1)$ in $\forall x.inverter(x) \wedge shorted(x) \supset in(x) = out(x)$.

- assumptions for subcomponents: e.g. $ab(gate2(A1))$ should only be considered if $ab(A1)$ has been selected.

- correctness of observations: e.g. $out(i1) = 1$.

- basic diagnostic principles. Examples are the single-fault hypothesis[1]

$$single\text{-}fault \equiv \forall x, y.\neg ok(x) \wedge x \neq y \supset ok(y) \tag{1}$$

  and completeness assumptions for the behavioural modes of a component, e.g. $complete(i1)$ for the behavioural modes of inverter $i1$:

$$\forall x.complete(x) \wedge inverter(x) \supset ok(x) \vee shorted(x) \vee stuck\text{-}at\text{-}0(x) \tag{2}$$

Making these assumptions explicit allows to find relationships between them and to control their interaction and selection.

3 Conflicts

Integrating different diagnostic strategies leads to systems where several or all of these kinds of assumptions are used. If no special mechanism is used (as e.g. in Poole's Theorist [Poole, 1988]) these assumptions interact arbitrarily if there are conflicts between them:

[1] This explicit representation of the single-fault hypothesis has been suggested to me by Gerd Brewka.

	minimal conflicts
in the original formulation:	{*ok*(A1), *ok*(M1), *ok*(M2)} {*ok*(A1), *ok*(A2), *ok*(M1), *ok*(M3)}
+ single-fault hypothesis as a premise:	{*ok*(A1), *ok*(M1)}
+ single-fault hypothesis as an assumption:	{*ok*(A1), *ok*(M1), *ok*(M2)} {*ok*(A1), *ok*(A2), *ok*(M1), *ok*(M3)} {*single-fault*, *ok*(A1), *ok*(M1)}

Figure 1: conflicts in the adder-multiplier example

Example 3.1 *We assume that the reader is familiar with the adder-multiplier example of Davis (cf. [De Kleer and Williams, 1987]). Below, we give a brief formulation in predicate logic:*

$$\text{adder}(A1)\quad \text{adder}(A2)\quad \text{multiplier}(M1)\quad \text{multiplier}(M2)\quad \text{multiplier}(M3) \tag{3}$$

where $A1 \neq A2$ *etc. These components are connected as follows:*

$$\begin{array}{lllll} \text{in1}(M1) = A & \text{in1}(M2) = B & \text{in1}(M3) = C & \text{in1}(A1) = X & \text{in1}(A2) = Y \\ \text{in2}(M1) = C & \text{in2}(M2) = D & \text{in2}(M3) = E & \text{in2}(A1) = Y & \text{in2}(A2) = Z \\ \text{out}(M1) = X & \text{out}(M2) = Y & \text{out}(M3) = Z & \text{out}(A1) = F & \text{out}(A2) = G \end{array} \tag{4}$$

The correct behaviour is described by a predicate ok *and:*

$$\begin{array}{l} \forall x.\text{adder}(x) \wedge \text{ok}(x) \supset \text{in1}(x) + \text{in2}(x) = \text{out}(x) \\ \forall x.\text{multiplier}(x) \wedge \text{ok}(x) \supset \text{in1}(x) * \text{in2}(x) = \text{out}(x) \end{array} \tag{5}$$

The supplied input and observed output values are:

$$A = 2 \quad B = 2 \quad C = 3 \quad D = 3 \quad E = 2 \quad F = 10 \quad G = 12 \tag{6}$$

Originally there are two minimal conflict sets (i.e. assumption sets that are inconsistent with the premises) consisting of *ok*-assumptions. Introducing the single-fault hypothesis 1 as an assumption leads to an additional conflict (cf. figure 1):

$$W \cup \{\textit{single-fault}, \textit{ok}(A1), \textit{ok}(M1)\} \models \bot \tag{7}$$

This conflict can e.g. be avoided by retracting the single-fault hypothesis itself. In this case, it is ineffective and double faults can be detected and returned as well.

Example 3.2 *Consider a chain of two inverters* $\text{inverter}(i1)$ *and* $\text{inverter}(i2)$ *s.t.* $\text{out}(i1) = \text{in}(i2)$. *Both inverters can work correctly, be shorted, or their outputs are stuck at zero:*

$$\begin{array}{l} \forall x.\text{inverter}(x) \wedge \text{ok}(x) \supset \text{out}(x) = 1 - \text{in}(x) \\ \forall x.\text{inverter}(x) \wedge \text{shorted}(x) \supset \text{out}(x) = \text{in}(x) \\ \forall x.\text{inverter}(x) \wedge \text{stuck-at-0}(x) \supset \text{out}(x) = 0 \end{array} \tag{8}$$

Consider the observations $in(i1) = 0$ and $out(i2) = 0$. Then things seem to be okay. However, we detect some conflicts if we use assumptions for failure modes in addition to correctness assumptions: It cannot be the case that one inverter is shorted and the other works correctly:

$$\{\textit{shorted}(i1), \textit{ok}(i2)\} \quad \{\textit{shorted}(i2), \textit{ok}(i1)\} \tag{9}$$

Due to these conflict sets we can also retract both correctness assumptions and conclude that both inverters are shorted.

These examples show that we cannot apply all assumptions and must retract some. A big problem is the selection of the culprits, i.e. the elements of conflicts that are retracted. If every assumption may be retracted then some of the diagnostic assumptions, e.g. the single-fault hypothesis, loose their value and others, e.g. failure modes, can be selected even if a correct mode may be considered as well.

Hence, *preferences* between diagnostic assumptions are needed to handle them properly. For example, we want to assign a higher priority to the single-fault assumption in order to avoid its immediate retraction. The purpose of the single fault assumption is to get rid of multiple faults in case of conflicts. It should be retracted only if there is no single fault. Furthermore, we want to prefer the correct behaviour to the faulty behaviour as in [Dressler and Struss, 1990].

4 Priorities on Assumptions

Prioritized defaults give a satisfactory answer to these problems. They are related to prioritized circumscription and have been introduced by Brewka [Brewka, 1989] to extend the simple THEORIST framework of Poole [Poole, 1988]. A prioritized default theory consists of a set W of premises, a set $\mathcal{D}$ of defaults that are represented as assumptions, and a strict partial order $<\subseteq \mathcal{D} \times \mathcal{D}$ on defaults. Hence, it matches our framework above.

A special case is obtained if the order $<$ satisfies the following condition: If $d_1 \not< d_2$, $d_2 \not< d_1$, $d < d_1$ implies $d < d_2$ for all defaults $d, d_1, d_2 \in \mathcal{D}$ then uncomparable defaults are put into the same level. In this important special case, defaults are divided into different levels L_i of reliability where elements d_1 of L_i have higher priority than defaults d_2 in L_{i+k} (i.e. $d_1 < d_2$). We mainly consider these *level-based default theories* in this paper.

Definition 4.1 *A* level-based default theory $\Delta := (W, L)$ *consists of a set* $W \subseteq \mathcal{L}$ *of classical premises and a tuple* $L := (L_1, \ldots, L_n)$ *of disjoint levels* L_i *containing ground atomic formulas from* $\mathcal{L}$.

For example, we can add the single-fault hypothesis into level L_1, the correctness assumptions into level L_2, and failure modes to L_3.

To find extensions of these default theories, i.e. possible selections of defaults, we start with the first level L_1, select elements as long as consistency is kept, and then repeat this for the next levels:

Definition 4.2 *Let* $\Delta = (W, (L_1, \ldots, L_n))$ *be a level-based default theory.* T *is a preferred subtheory of level* i *iff*

1. $T = \emptyset$, *for* $i = 0$
2. T *is the union of a preferred subtheory* T' *of level* $i - 1$ *and a maximal subset of* L_i *s.t.* $T \cup W \not\models \bot$, *for* $i = 1, \ldots, n$.

The preferred subtheories of Δ *are the preferred subtheories of level* k.

We can extract candidates from level-based default theories as follows:

Definition 4.3 *Let $DP = (\mathcal{O}^-, \mathcal{O}^+, \mathcal{F})$ be a diagnostic problem and $\Delta = (W \cup \mathcal{O}^-, (L_1, \ldots, L_n))$ be a level-based default theory to solve it. C is a* diagnosis *iff there exists a preferred subtheory T of Δ s.t. $T \cup W \models o$ for all $o \in \mathcal{O}^+$ and $C = \{f \in \mathcal{F} \mid T \cup W \models f\}$.*

If the single-fault hypothesis is consistent for its own it will be selected in level L_1 and cannot be retracted due to later conflicts. Hence, priorities on defaults give an answer to our culprit selection problem: If there is a conflict $W \cup \{d_1, \ldots, d_k\} \models \bot$ we can retract one of the defaults having the smallest priority (i.e. are in the highest level). However, such a culprit is not retracted in every case. If elements d_i with higher priority are retracted due to other conflicts then our conflict is ineffective anyway and our culprit may be selected.

The exact behaviour is achieved by the implementation of level-based default theories [Junker, 1991] using Doyle's TMS [Doyle, 1979]. Defaults d are handled by a non-monotonic justification: If their negation cannot be derived they are selected:

$$\langle out(\neg d) \rightarrow d \rangle \quad \text{for all } d \in \mathcal{D} := L_1 \cup \ldots \cup L_n \tag{10}$$

Conflicts give rise to counterarguments for those of their defaults having the smallest priority. If $W \cup \{d_1, \ldots, d_k\} \models \bot$ then for all $i = 1, \ldots, k$ we add

$$\langle in(\{d_1, \ldots, d_k\} - \{d_i\}) \rightarrow \neg d_i \rangle \quad \text{if there is no } j \in \{1, \ldots, k\} \text{ s.t. } d_i < d_j \tag{11}$$

We can incrementally construct a TMS-network consisting of those justifications. First of all, we add justification 10 for all $d \in \mathcal{D}$. Labelling makes all defaults IN. Then we repeat the following procedure. (1) Check whether the set of IN-labelled defaults is consistent in conjunction with W. This can be achieved by executing rules or by invoking a classical prover. (2) If an inconsistency is detected a conflict involving IN-labelled defaults is computed and used to add justifications according to 11. (3) After that, the network is relabelled as usual [Doyle, 1979]. This procedure is repeated until no further inconsistency is detected. Then a first preferred subtheory has been detected. Further preferred subtheories can be computed by choosing another labeling of the already generated network and repeating the procedure above. Thus, we can every preferred subtheory straightforwardly.

Indeed, the system in [Petrie, 1987] already uses a kind of priorities to control the culprit selection of dependency-directed backtracking as sketched above. Hence, it should be possible to use this system for computing preferred substheories.

5 Conclusion

Prioritized defaults [Brewka, 1989] are well-suited to control the retraction of different kinds of assumptions including basic diagnostic principles, correctness assumptions, and fault modes. Furthermore, prioritized defaults specify a preference relation on candidates and thus help to focus diagnosis. This focusing method is less flexible than statistical assessments of candidates, but needs less search effort. Our preferred diagnoses can straightforwardly be computed by a TMS-based system [Junker, 1991]. First proposals how to handle partial orders between assumptions have been developed in [Junker and Brewka, 1991]. Currently, we investigate how to handle context-dependent priorities of the form $ok(c_1) < ok(c_2)$ properly. These can be derived from the diagnostic theory and and allow more flexibility in specifying priorities.

Acknowledgements

Gerd Brewka, Gerhard Friedrich, Peter Struss, and Hans Voss gave substantial comments to this work. Markus Junker helped to clarify the problems with de Kleer's lexicographical ordering.

References

[Brewka, 1989] G. Brewka. Preferred subtheories: An extended logical framework for default reasoning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 1043–1048, Detroit, MI, 1989.

[Console and Torasso, 1990] L. Console and P. Torasso. Integrating models of the correct behavior into abductive diagnosis. In *Proceedings of the European Conference on Artificial Intelligence*, pages 160–166, Stockholm, 1990.

[De Kleer and Williams, 1987] J. De Kleer and B.C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.

[Doyle, 1979] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[Dressler and Struss, 1990] O. Dressler and P. Struss. Back to defaults: Computing diagnoses as coherent assumption sets. Technical report, Siemens AG, 1990.

[Junker and Brewka, 1991] U. Junker and G. Brewka. Handling partially ordered defaults in TMS. In R. Kruse and P. Siegel, editors, *Symbolic and Quantitative Aspects for Uncertainty. Proceedings of the European Conference ECSQAU*, pages 211–218. Springer, LNCS 548, Berlin, 1991.

[Junker, 1991] U. Junker. Prioritized defaults: Implementation by TMS and application to diagnosis. In *Proceedings of the Twelth International Joint Conference on Artificial Intelligence*, pages 310–315, Sydney, Australia, 1991.

[Petrie, 1987] C. Petrie. Revised dependency-directed backtracking for default reasoning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI)*, pages 167–172, 1987.

[Poole, 1988] D. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36:27–47, 1988.

A Truth Maintenance System for Medical Knowledge-Based Assistance Systems

Hauke Kindler,
University of Ulm, Institute for Occupational and Social Medicine,
FAW, Postfach 2060, D-7900 Ulm

Many medical expert systems have already been developed. In contrast to expert systems in industrial applications they have rarely been used in routine. Legal, ergonomic, and ethical charges imply special requirements, which have for the most part not been fulfilled, yet. Physicians are personally responsible for a patient's treatment even when applying complex technical aids. Thus, using a knowledge-based system the physician has to take the full responsibility for its application. Therefore, technical decision aids have to be transparent, fully understandable, explicable, and justifiable for them. We use the term knowledge-based assistance system instead of expert system due to the role of a system which respects the above mentioned constraints being a cooperative assistant, who has a staff function, providing advice like a consultant, and is able to explain and justify its advice.

In accordance to the hereinbefore named conditions the following design goals for knowledge-based assistance systems can be defined:

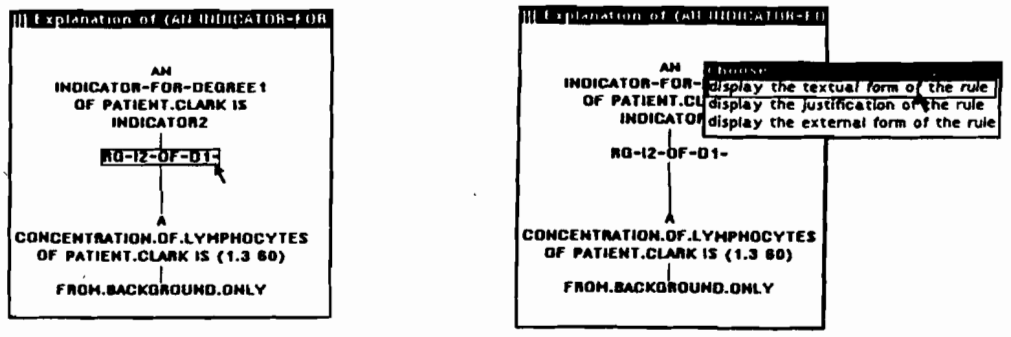
- explicability consisting of the possibility to give "why"-explanations and "how"-explanations,
- justifiability meaning that the correctness of the applied knowledge can be justified,
- the possibility for the user to override the conclusions of the system, and
- the possibility to correct wrong input.

A "How"-explanation is the tree of deductions whose father is a certain value of a patient's property. A TMS [2] will be necessary to give "How"-explanations if the inferences are performed with rules in first-order predicate calculus due to instantiable knowledge chunks [3]. The first-order predicate calculus is an essential prerequisite in complex medical domains, e. g., electromyography [3,4] and acute radiation syndrome [5].

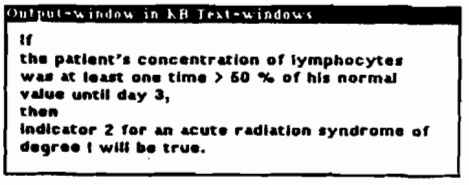
For the structuring of the problem solving a cognitive problem solving model is used [4, 5]. The problem solving task is decomposed hierarchically into subtasks until means are found to solve the subtasks. This is done by explicitly represented strategic knowledge. The task hierarchy and its creation by strategic rules are recorded by a TMS. "Why"-explanations to a question like "Why is a certain subtask executed?" and "Why is a certain question asked?" can be given by a TMS as the tree of deductions which invoked the action.

Due to the TMS the repetition of all inferences will be avoided if deductions of the knowledge-based assistance system are overridden by the user or the user corrects his input.

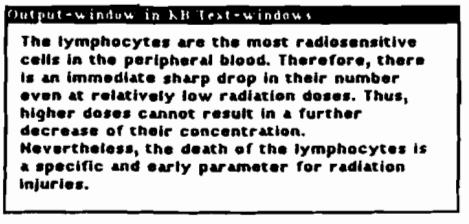
Below the answer to the question "How was deduced that the indicator 2 indicates the degree 1 of the acute radiation syndrome?" is depicted. Followed by the response to the question "Why was the task medical measures of day 4 instantiated?". The examples have been implemented on KEE™ using its ATMS [1] and rule-based inference engine. The user interface has been extended.



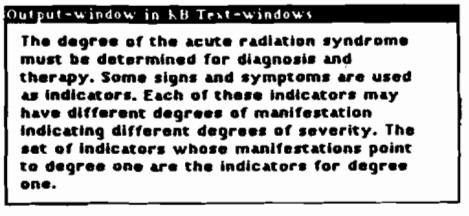
After the "How"-question an explanation window opens and shows the deduction graph computed by the ATMS. The fact "indicator 2 indicates the degree 1 of the acute radiation syndrome" was deduced by the rule "rg-I2-of-D1-" which was instantiated with the fact "the concentration of lymphocytes of patient was 60% of his normal value at the day 1.3". The rectangle around "rg-I2-of-D1-" denotes that supplementary information is available. On the right side a menu lists the different types of information.



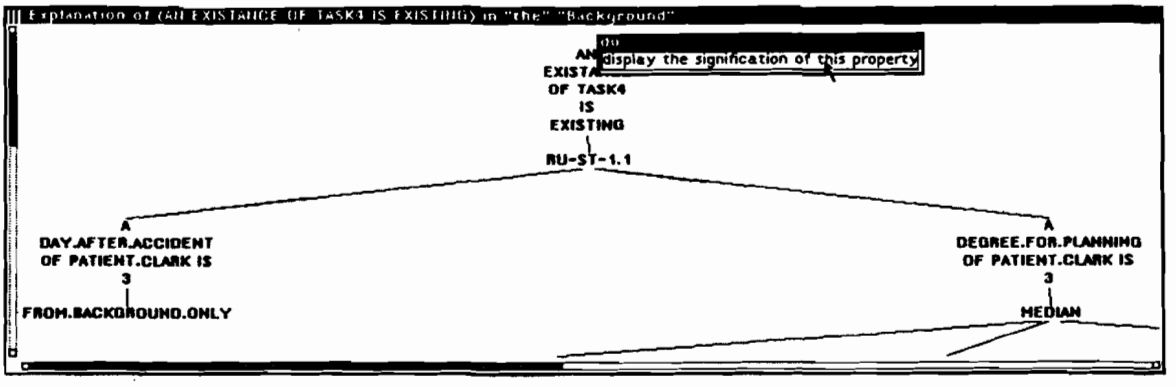
The user can look at the textual form of the rule.



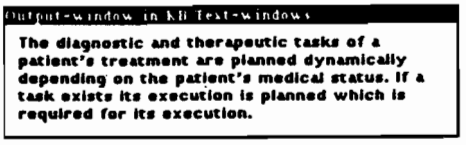
He can get a text which justifies the rule.



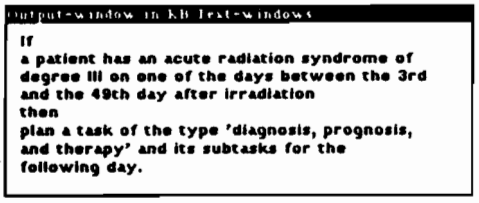
A description of the signification of "indicator for degree 1" can be given to the user.



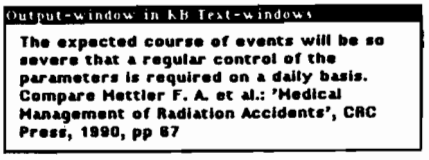
After the "Why"-question an explanation window opens and shows the deduction graph computed by the ATMS. The fact "task 4 is existing" was deduce by the rule "rg-st-1.1" which was instantiated with the facts "today is the third day after the accident" and "the estimated degree of the acute radiation syndrome (for planning) is 3".



A description of the signification of "existance of task 4" can be given to the user.



The user can look at the textual form of the strategic rule.



He can get a text which justifies the rule by referencing to literature.

The texts for the justifications, textual forms and significations are attached to the strategic rules and patient's properties due to object-oriented methodology of KEE™.
Apart from the adaption of the user interface it has been possible to apply the ATMS of KEE™ and its inference engine without modifications. The deductive knowledge depicted above can be represented in the form of deduction rules, which are a special type of rules with some limitations compared to normal rules. But, in the domain of application which is the medical management of irradiated persons both nonmonotonic reasoning and reasoning with the universal quantifier is required. The knowledge to express this, cannot be represented with deduction rules, because only the premisses part of deduction rules can be extended by Lisp code.

Normally the rule system of KEE™ provides no possibility for nonmonotonic reasoning and the use of the universal quantifier. Nevertheless, these two effects can be achieved by integrating Lisp code into the rules. Some of this Lisp code has to be integrated into the conclusion part of the rules. The rule syntax becomes complex, but the advantage is that every rule is still an autonomous chunk of knowledge, which can be used normally in forward and backward chaining by the normal inference engine profiting from a Rete network and providing nonmonotonic reasoning and the use of the universal quantifier.
Nonmonotonic deductions are provided by retracting the values of the object-property-value triples at the beginning of the conclusion part of the rule.
Reasoning with the universal quantifier is invoked when one supplementary object-property-value triple is asserted or retracted. After a get.values command all values of the property of the object can be processed at the same time. This is done in the premisses part of the rule.

Within the KEE™ deduction rules, which are the rules which trigger the ATMS automatically, it is not possible to use Lisp code in the conclusion part. Thus, it is not possible to enhance the deduction rules like normal rules in KEE™ to integrate nonmonotonic reasoning and the use of the universal quantifier. The possibility to create ATMS justifications by Lisp code is offered by KEE™. Using the normal rules with Lisp code and ATMS operators like "remove.justification" and "create.justification" nonmonotonic reasoning and reasoning with the universal quantifier is provided. Two examples are depicted below. A mixing of nonmonotonic reasoning and the use of the universal quantifier would be possible, too.

```
((IF
  (THE SLOT2 OF ?OBJECT IS ?VALUE)
  THEN
  (LISP
   (IF
    (CAR (GET.SUPPORTING.JUSTIFICATIONS (CAR (GET.SLOT.PROPOSITIONS ?OBJECT 'SLOT1))))
    (REMOVE.JUSTIFICATION
     (CAR (GET.SUPPORTING.JUSTIFICATIONS (CAR (GET.SLOT.PROPOSITIONS ?OBJECT 'SLOT1)))))
    T))
  (LISP (CREATE.JUSTIFICATION (LIST 'THE 'SLOT1 'OF ?OBJECT 'IS ?VALUE)
                              (LIST (LIST 'THE 'SLOT2 'OF ?OBJECT 'IS ?VALUE))
                              "RUNEW12"))
  (THE SLOT1 OF ?OBJECT IS ?VALUE)
  (LISP (RETRACT (LIST 'THE 'SLOT1 'OF ?OBJECT 'IS ?VALUE)))))
```

Nonmonotonic deductions and the creation of deductions are provided by the following actions in the premisses part:

- removing the old justification of the object-property-value triple in the beginning of the conclusion part of the rule if an old justification exists,
- creating the new justification for the property of the object,
- asserting conventionally the new value to the property of the object enabling normal triggering of the rule, and
- retracting this value for having only available the object-property-value of the ATMS.

```
(IF (THE SLOT4 OF OBJECT1 IS ?)
    (?VALUE = (LISP (EVAL (CONS '+ (GET.VALUES 'OBJECT1 'SLOT4)))))
    THEN
    (LISP (CREATE.JUSTIFICATION (LIST 'THE 'SLOT3 'OF 'OBJECT1 'IS ?VALUE)
                                (DO ((HL (GET.VALUES 'OBJECT1 'SLOT4) (CDR HL))
                                     (NL NIL))
                                    ((NULL HL) NL)
                                  (SETQ NL (CONS (LIST 'THE
                                                       'SLOT4
                                                       'OF
                                                       'OBJECT1
                                                       'IS
                                                       (CAR HL))
                                                 NL)))
                                "RGLE3"))
    (THE SLOT3 OF OBJECT1 IS ?VALUE)
    (LISP (REMOVE.VALUE 'OBJECT1 'SLOT3 ?VALUE)))
)
```

Reasoning with the universal quantifier is invoked when one supplementary object-property-value triple is asserted or retracted. After a "get.values" command all values of the property of the object can be processed at the same time. This is done in the premisses part of the rule. In the conclusion part the justification is created. As in the example for the nonmonotonic deductions the value is asserted normally to enable correct triggering and retracted thereafter. Thus, only the values justified by the ATMS remain.

```
(Output) The OBJECT1 Unit in ATMS-TEST Knowledge B
Unit: OBJECT1 in knowledge base ATMS-TEST
Created by hauke on 7-12-91 12:44:20
Modified by hauke on 7-24-91 11:07:51
  Member Of: OBJECT

Own slot: SLOT1 from OBJECT1
  Inheritance: OVERRIDE.VALUES
  Values: UNKNOWN

Own slot: SLOT2 from OBJECT1
  Inheritance: OVERRIDE.VALUES
  Values: 1, 2

Own slot: SLOT3 from OBJECT1
  Inheritance: OVERRIDE.VALUES
  Values: UNKNOWN

Own slot: SLOT4 from OBJECT1
  Inheritance: OVERRIDE.VALUES
  Values: 4, 1, 2, 3
```

```
(Output) The OBJECT1 Unit in ATMS-TEST Knowledge B
Unit: OBJECT1 in knowledge base ATMS-TEST
Created by hauke on 7-12-91 12:44:20
Modified by hauke on 7-24-91 11:25:18
  Member Of: OBJECT

Own slot: SLOT1 from OBJECT1
  Inheritance: OVERRIDE.VALUES
  Values: 1

Own slot: SLOT2 from OBJECT1
  Inheritance: OVERRIDE.VALUES
  Values: 1, 2

Own slot: SLOT3 from OBJECT1
  Inheritance: OVERRIDE.VALUES
  Values: 10

Own slot: SLOT4 from OBJECT1
  Inheritance: OVERRIDE.VALUES
  Values: 4, 1, 2, 3
```

The nonmonotonic rule is instantiated two times with the values 1 and 2 of slot2 of object 1. First the value 2 for slot1 of object 1 is deduced, which is then replaced by the value 1. The universal quantifier rule is instantiated with the values of slot 4 of object 1 to deduce the sum of them, which is written into slot 3. Explanations for the values of slot 1 and slot 3 can be provided by the ATMS.

For the purposes of the application mentioned above it would be sufficient to only use a JTMS [2]. The recalculation of the labels and nodes of the ATMS when asserting or retracting justifications while reasoning nonmonotonically becomes very timeconsuming in compared to a JTMS.
In the above mentioned application planning is performed. A plan, which is a hierarchy of tasks, is created. Every task has states of execution, like "instantiated", "active", "executed". Deducing from the instantiation of a task and some other premisses that it is "active" is nonmonotonic because its state changes from "instantiated" to "active" and for the TMS a hard loop occurs. The hard loop means that the value of a property of the object would have to be justified by the not any more existing previous value of the same property of the object. This creates a problem for the recording by a TMS for which two solutions exist.
The first solution is to have the three properties "instantiation", "activation", and "execution" for each task, whose value types are Boolean. Thus, the value "yes" of the property activation of an object would be justified by the value "yes" of the property instantiation of the same object by the TMS. This solution is only applicable if the set of possible values of the property is finite. The implementation is easy.
The second solution is having a stack of values for one property of the object in chronological order with only the top value readable. When reasoning nonmonotonically a value is not physically retractet but only overwritten and still recorded in the stack. The value "active" of a property of the object would then be justified by its former value "instantiated", which is not readable but still existing in the stack one element below. This solution would be applicable for infinite value sets of properties. Another advantage of this solution would be that an overridden value remains recorded. But the implementation would require modifications of the trigger-mechanism for the inference engine and the introduction of a special type of single value property.

[1] DeKleer J.: "An Assumption Based TMS", Artificial Intelligence, Bd. 28, 1986

[2] Doyle : "A Truth Maintenance System", Artificial Intelligence Journal, Bd. 12, North Holland, Amsterdam, 1979

[3] Kindler H.: "Wissensmodellierung als Grundlage eines intelligenten Tutors in der Elektromyographie", in Reuter A.: "20. GI-Jahrestagung", vol. 2, Springer, Heidelberg, 1990

[4] Kindler H.: "Controlling Qualitative Reasoning by a Cognitive Problem-Solving Model for Decision Making in Electromyography", Proceedings of the Workshop of Qualitative Reasoning and Decision Support Systems 1991, Toulouse, North Holland, Amsterdam, to appear 1991

[5] Kindler H., Densow. D., Fliedner T. M.: "RADES - Medical Assistance System for the Management of Irradiated Persons", 2nd International Conference on Database and Expert Systems Applications, 1991, Berlin, Springer, Vienna, to appear 1991

The Integration of an ATMS and a Constraint–System in IDA*

Jürgen Paulokat
Universität Kaiserslautern
P O Box 3049
W–6750 Kaiserslautern
Fed. Rep. of Germany
paulokat@informatik.uni-kl.de

Holger Wache
Universität Dortmund
P O Box 500 500
W–4600 Dortmund 50
Fed. Rep. of Germany
wache@jake.informatik.uni-dortmund.de

Extended Abstract

1 Introduction

We describe the integration of an ATMS and a Constraint–System in IDA[1] [MPS+90, Kra91] a shell for the development of expert systems for conception and configuration tasks in technical domains. The IDA system implements a model of design which provides means to describe physical components (e.g. a hydraulic element) and technical processes (e.g. welding), their possible use in aggregates, and their functional specification on different levels of abstraction.[2] Functional specifications, called functions, are used to specify the task or subtasks of an aggregate to be constructed by the expert system. During the design process a function can be divided into more specialized subfunctions or realized by a technical realization (i.e. a physical component or a technical process). Compatibility conditions between the components of aggregates can be expressed by constraints.

A solution of a design task is an aggregate where every function has been refined by more specialized subfunctions or realized by a technical realization. If some of these steps remain to be done we have a partial solution. During the process of refinement and realization we must fulfil all compatibility constraints. Further a more specialized function must fulfil all requirements of the function detailed by it defining new specifications on a lower level of abstraction. If one or more of these consistency conditions can not be met this partial solution is inconsistent. In general tasks can be done in alternative ways. Every alternative found during search defines a (partial) solution of its own. The inference component can change arbitraryly the alternatives and selects one at a time for further computations. In the following we will not always distinguish functions and subfunctions or a solution and partial solutions because most of the given propositons are independent of these differentiations.

In IDA an ATMS is used to represent the alternatives found during search and the uni–directional presuppositions beween a function and its technical realization or subfunctions. In contrast to the ATMS constraints describe multi–directional dependencies. They are used to express that combinations of n components are inconsistent. So, the selection of $(n-1)$ components out of such an inconsistent combination results in the exclusion of the others from further considerations of the inference component. If there is an unsolved task, but all alternatives solving it are ruled out by constraints this partial solution is inconsistent. In this case the system must draw back some decisions to resolve the conflict (dependency directed backtracking). This problem may even be more complicated because constraints can express preferences and can be relaxed or retracted. The relaxation of constraints is described in [Pau90, Pau91]. In the following we describe only the modification of an ATMS to fit our computational model underlying the IDA system and its integration with the constraint system.

*The work presented herein was partially founded by the Deutsche Forschungsgemeinschaft (DFG), SFB 314 "Artificial Intelligence — Knowledge Based Systems", projects X7 and X9

[1] IDA is an abbrreviation of Intelligent Design Assistant

[2] The examples are taken from an expert system for the construction of fixing elements realized using IDA.

2 Representation of a (Partial) Solution in the ATMS

In general the search space of a design problem is very large. In most cases however we are only interested in one "good" solution. This implies that we must focus search. This behavior is supported by the concept of a control environment for the ATMS introduced by de Kleer [dK86a, dK86b], which we extend to the global environment. A global environment contains all assumptions currently believed in and defines a global context which contains all nodes which can be derived from the assumptions in the global environment. Beside of that we use two special kinds of nodes in our ATMS: an alternative node for every alternative found during search and a group node representing a disjunction of alternatives. Further, group nodes serve as connection between the ATMS and the constraint system. The constraint system interprets a group node as a variable whose possible values are the alternatives of the group. Values of a variable (i.e. some alternatives of a group) can be removed by constraint propagation and are excluded from further inferences of the problem solver controlling the ATMS. Allowing a dynamical change of the contents of a group, i.e. the addition and removal of ATMS nodes by the constraint system we prevent a redundant representation of incompatibility conditions by both constraints and nogoods of the ATMS. The use of a constraint system is necessary for propagation and relaxation of multi-directional dependencies which cannot be handled directly by the justifications of an ATMS.

Given that, the features of our design model lead to three conditions for a consistent (partial) solution which can be tested on the implementation level of the ATMS. 1.) Every alternative must belong to exactly one group, 2.) an alternative can only belong to the global context if its group belongs to it, 3.) if a group is in a global context then there must be one alternative belonging to this group which is contained in the global context or can be consistently added (i.e. there are no nogoods that deny the addition).

In [dK86a, dK86b] de Kleer describes the problem that all alternatives of a group are ruled out by nogoods, which we call the exclusion problem. He generates new nogoods preventing this kind of inconsistency. In our case the problem is more complex because the set of alternatives can change dynamically.

3 Exclusion Problem

In our computational model of configuration a partial solution is inconsistent if a group $G = \{A_1, ..., A_n\}$ is contained in the global context whose alternatives A_i are ruled out completely by nogoods. In this case for every A_i exists an assumption a_i and every a_i is contained in a nogood $N_i = N_i' \cup \{a_i\}$. If A_i is ruled out by the nogood N_i then N_i is active with respect to the global environment, i.e. every assumption $n_j \in N_i$ is an element of the global environment.

3.1 The Hyperresolution

Such an inconsistency can only be resolved if we remove at least one assumption $n_k \in N_i'$ out of the global environment. After this removal A_i is no more ruled out by the nogood N_i and can be added to the global environment if there is no other nogood ruling out A_i, again. But this can be solved by repeatedly removing assumptions of the offending nogoods out of the global environment. To prevent the encounter with this inconsistency again in future inference steps we generate a new nogood $N = \bigcup_{i=1}^{n} N_i' = \bigcup_{i=1}^{n} N_i \setminus \{a_i\} = (\bigcup_{i=1}^{n} N_i) \setminus G$. In [dK86a, dK86b] this computation rule is called hyperresolution and noted as:

$$\frac{\begin{array}{l}\textsc{Gruppe}\ G = \{A_1, ..., A_k\}\ \text{mit}\ A(G) = \{a_1, \ldots, a_n\} \\ \exists\ \textsc{Nogood}\ N_i = (n_{i1}, \ldots, n_{il}) \quad \text{mit}\ a_i \in N_i\ \text{und}\ \forall j \neq i : a_j \notin N_i\end{array}}{\textsc{Nogood}\ N_{neu} := \bigcup N_i \setminus A(G)}$$

In this rule $A(G)$ represents a set of assumptions a_i for every alternative $A_i \in G$. To select the alternative A_i the assumption a_i must be added to the global environment. The nogood N_{neu} guarantees that at least one A_i of G can be selected with respect to the nogoods N_i.

3.2 Combining Dynamic Groups and Hyperresolution

The hyperresolution described so far guarantees the selection of an alternative for every group if the sets of alternatives doesn't change. But if we allow the removal and additon of alternatives (e.g. by the constraint system) the nogoods generated by the hyperresolution may become incorrect and must be deactivated if we want guarantee the completeness of search. This causes further problems if we have removed nogoods subsumed by a deactivated nogood out of the nogood database. To overcome the first problem we modify the hyperresolution rule to produce conditioned nogoods. A conditioned nogood is of the form $N_{Cond} = Cond \rightarrow nogood\{n_1, ..., n_m\}$. Using a condition a nogood becomes dependent of the global environment. It specifies a conjunction of sets of alternatives which exactly must be contained in the referenced groups so that this nogood is correct. The form of a condition is $Cond = \bigwedge_{i=1}^{n}[A(G_i) = \{a_{i1}, ..., a_{im}\}]$. I.e. $Cond$ reflects the contents of the groups G_i at the moment of resolution. If $n = 0$ the condition is the empty conjunction which is interpreted to be always true. So, we can express unconditioned nogoods by an empty condition. To prevent the second problem we exclude conditioned nogoods from subsumption.

3.3 Modified Hyperresolution

Using conditioned nogoods the hyperresolution can be modified in the following way:

$$\frac{\begin{array}{l}\textsc{Gruppe}\ G = \{A_1, ..., A_k\}\ \text{mit}\ A(G) = \{a_1, \ldots, a_k\} \\ \exists\ \textsc{ConditionedNogood}\ N_i = B_i \rightarrow (n_{i1}, \ldots, n_{il}) \quad \text{mit}\ a_i \in N_i\ \text{und}\ \forall j \neq i : a_j \notin N_i\end{array}}{\textsc{ConditionedNogood}\ N_{neu} := (\bigwedge B_i) \wedge [A(G) = \{a_1, \ldots, a_k\}] \rightarrow (\bigcup N_i \setminus A(G))}$$

Conditions used in the precondition of the nogoods N_i can be empty. Initially, empty preconditons are generated by the problem solver. Not empty preconditions can only be generated by the hyperresolution. If nogoods with a not empty precondition are used in the hyperresolution the conjunction of all of these preconditions and the expression $A(G) = \{a_1, \ldots, a_k\}$ are used to build up the precondition of the resolved nogood. $A(G)$ describes the set of alternatives of the group causing the hyperresolution at the moment of resolution.

To minimize computational efforts conditioned and uncondioned nogoods are kept in different nogood databases. The database of the conditioned nogoods must be updated after every change of the global environment. Here a nogood can only be subsumed by another one if their conditions are identical.

3.4 Reduction of Work

Given a design problem there are many nogoods which can be resolved but the situation in which the resolved nogoods are active are never reached. To reduce the amount of work the hyperresolution is only activated if there is a group belonging to the global context whose alternatives are ruled out completely, i.e. an inconsistency has been discovered.

4 Backtracking

Using a global environment to focus on one (partial) solution at a time we must backtrack if an inconsistency has been detected. Backtracking is started with the nogood causing the inconsistency and removes at least one assumtion out of the global environment. If the nogood is unconditioned backtracking works as described in [dK86a]. If backtracking is started with a conditioned nogood the justification for the elective must be conditioned, too, i.e. this justification must be removed, when the condition of the nogood becomes false. This removal of an conditionend justification can be handeled by standard backtracking and guarantees completeness of search.

5 Discussion

We described the integration of an ATMS and a constraint system. A typical aspect of integration is the reduction of redundant information. In our case we don't translate constraints into nogoods and use the nodes of the ATMS as variables and their possible values in the constraint system. This leads to the problem that the node data base of the ATMS is modified by the constraint system from outside of the ATMS. This extended functionality of the ATMS is supported by a modification of the hyperresolution. The explicit representation of constraints offers a means to relax the requirements if the problem is overconstrained, i.e. there is no consistent solution.

References

[dK86a] J. de Kleer. Back to backtracking: Controlling the atms. In *Proc. of AAAI-86*, pages 910–917, Philadelphia, PA, 1986.

[dK86b] Johan de Kleer. Extending the ATMS. *Artificial Intelligence*, 28:163–196, 1986.

[Kra91] Norbert Kratz. *Architektur eines wissensbasierten Systems zur Unterstützung der Konzeptionsphase in der Konstruktion.* PhD thesis, Universität Kaiserslautern, Kaiserslautern, Germany, 1991.

[MPS+90] Michael Mehl, Jürgen Paulokat, Eric Schaumlöffel, Peter Spieker, and Elmar Then. IDA — Ein Expertensystem zur Konzeption von Vorrichtungselementen. SEKI Working Paper: SWP–90–06, Universität Kaiserslautern, 1990.

[Pau90] Jürgen Paulokat. Ein System zur Verarbeitung und Relaxierung von Constraints. Master's thesis, Universität Kaiserslautern, 1990.

[Pau91] Jürgen Paulokat. Verarbeitung und Relaxierung von Constraints in MOLTKE–P/IDA–III. In *Proceedings 5. Workshop „Planen und Konfigurieren“*, pages 123–129, Hamburg, Germany, 1991.

Extended Abstract

Semantics for Reason Maintenance

Charles Petrie
MCC AI Lab
3500 West Balcones Center Drive
Austin, TX 78759
petrie@mcc.com
petrie@informatik.uni-kl.de

Workshop: "(A)TMS in Expertsystemen"
Universität Kaiserslautern
17. Juni, 1991

1 Introduction

Truth (or Reason) Maintenance is an AI technique developed originally by Jon Doyle more than a decade ago. Doyle envisioned that a Truth Maintenance System(TMS) would make possible planning and design applications in which the rationale for the reasoning would always be available and would support incremental decision making and revision.

Planning problems in particular require such support. Not only are there constraints among the planning decisions, but contingencies may require replanning. More traditional techniques, such as Constraint Satisfaction and Integer Programming, are inadequate to support such problems. Even the AI planning work fails to support replanning.

Truth maintenance has long been promising because it provides a general computational mechanism for tracking dependencies and using them in problem solving. But this potential has not been realized. Despite much interest in truth maintenance, few design or planning applications making significant use of a TMS have appeared in the literature.[13] Our hypothesis is that this is because there has not been a model of planning and design. The result is that the dependencies produced during problem solving are ad hoc and the application behavior is so difficult to control that large applications are not possible.

We identify crucial shortcomings of classical problem solver/TMS architectures and give a model, "Constrained Decision Problems" (CDPs), of the abstract computation that characterizes planning and design. This model introduces a novel search concept, "context maintenance", that justifies the dependency directed backtracking mechanism associated with a

TMS. From this model, we have developed and implemented a computational architecture, called REDUX, that uses a TMS to guide CDP solving. The CDP model provides semantics for TMS dependencies and removes the need for the application builder or user to understand a TMS.

The preliminary results indicate that at least some CDP applications are much easier to build with REDUX than with previous expert system shells using a TMS. The intent is to develop REDUX into a new class of decision support systems.

2 TMS Problems

Of the several varieties of TMSs, we focus on Doyle's[6] because planning and design applications typically work toward a single solution using default heuristics. Some other systems that are fundamentally oriented for reasoning within a single context are [5, 8, 7]. The Assumption-based TMS (ATMS)[2], best supports problems in which one only wants to know what set of contexts are consistent with the constraints [3, 10]. Modifying an ATMS to focus on a single context, perform backtracking, and maintain nonmonotonic justifications is problematic and the subject of current research. With Doyle's TMS, such functionality is free.

However, the single-context TMS also has problems shared with other TMSs. A major source of these problems comes from the mistaken principle of *inferential indifference*: there is a problem solver and a TMS that exchange clauses and labels and the content of the inferences doesn't matter. That this principle can cause problems is particularly evident if one compares control and domain inferences. The former are generally based upon the problem solving state which necessarily changes. Justifying control inferences can easily lead to unsatisfiable circularities, among other problems.

Another major problem is with dependency-directed backtracking. The conditional proof in [6] and the elective justification of [10] have similar behaviors. Unfortunately, the semantics for this backtracking mechanism are undefined. The behavior of the justifications generated by backtracking can generate problems similar to those of control inferences.

The third problem we mention is lack of differentiation among assumptions. Morris' principle of "wishful thinking"[9] points out that some assumptions can be retracted in order to solve a problem and some cannot. The latter are default expectations about the plan execution environment that can not be changed just as matter of convenience.

3 A Planning Model

The underlying reason for these problems is lack of a model of default reasoning in planning and design. Without such a model the dependencies produced by a problem solver are *ad hoc* and their labeling will be unpredictable.

Our CDP is a goal/operator model of planning. Its most important feature is that planning decisions have both validity and optimality. The latter characterizes the local goodness of the choice from among alternatives to satisfy a goal. Validity and optimality

have distinct TMS dependency networks. An example of validity is whether or not a flight has been canceled. If it is canceled, then the decision to include the flight in the plan becomes invalid. This change can be syntactically determined and left entirely to the TMS to manage.

The optimality of a flight might depend, say, on its cost relative to similar flights. If this relative choice changes, then the decision whether or not to continue to include the flight in the plan must be reasoned. For instance, we may be too far along in our planning to redo it to save a few dollars. This is a semantic decision. The TMS can be used to signal the problem solver that this decision should be made. But by separating it from validity, we prevent the TMS from changing the current solution context automatically.

Rejection of a decision when backtracking due to a constraint violation is also a decision with optimality. That optimality depends upon the validity of conflicting decisions. The elective justification of [10] captures this optimality, with a slight modification to distinguish validity and optimality. Collectively, the optimality of decisions and decision rejections constitutes the *context rationale*, which can be used effectively in problem solving.

Context rationale turns out to capture the economic notion of *pareto optimality*. A solution is pareto optimal if no part can be improved without decreasing the goodness of some other solution element. The goal satisfactions in a CDP are connected only by constraints. Thus the rejection optimalities reflect their connections. A rejection becomes suboptimal exactly when it could be improved without changing another decision. The optimality of decisions will not depend upon other decisions. Thus the context rationale tracks pareto optimality. This finally provides semantics for the DDB mechanisms in TMSs.

The CDP model is implemented in REDUX, derived from RAD[1] which was itself based on Proteus[11]. Initial application studies indicate that REDUX greatly facilitates application building. In particular, the professor/course assignment problem described in [12] and [4] is greatly simplified.

References

[1] Natraj Arni, David Murray Bridgeland, James Christian, Michael N. Huhns, Charles J. Petrie Jr., Elaine Rich, James Conrad Shea, and Munindar P. Singh, "Overview of RAD: A Hybrid and Distributed Reasoning Tool," MCC Technical Report Number ACT-RA-098-90, Microelectronics and Computer Technology Corporation, March 1990.

[2] de Kleer, J., "An Assumption-based TMS", *Artificial Intelligence* **28**, pp. 127-162, 1986.

[3] de Kleer, J., "Back to Backtracking: Controlling the ATMS," *Proc. of the Fifth National Conference on Artificial Intelligence*, AAAI, pp. 910-917, 1986.

[4] Dhar V. and Raganathan N., "An Experiment in Integer Programming," *Communications of the ACM*, March 1990. Also MCC TR ACT-AI-022-89 Revised.

[5] Dhar V. and Croker A., "A Knowledge Representation for Constraint Satisfaction Problems," Dept. of Info. Systems, NYU, Technical Report 90-9, January 1990.

[6] Doyle J., "A Truth Maintenance System," *Artificial Intelligence*, **12**, No. 3, pp. 231-272, 1979.

[7] Dressler O. and A. Farquhar, "Problem Solver Control over the ATMS," submitted to AAAI-89.

[8] McAllester D., "An Outlook on Truth Maintenance," A.I. Memo 551, Massachusetts Institute of Technology, AI Lab., 1980.

[9] Morris, P. H., Nado, R. A., "Representing Actions with an Assumption-Based Truth Maintenance System," *Proc. Fifth National Conference on Artificial Intelligence*, AAAI, pp. 13-17, 1986.

[10] Petrie, C., "Revised Dependency-Directed Backtracking for Default Reasoning," *Proc. AAAI-87*, pp. 167-172, 1987.

[11] Petrie, C., et al., "Proteus 2: System Description," Technical Report, Microelectronics and Computer Technology Corporation MCC TR AI-136-87, 1987.

[12] Petrie, C., R. Causey, D. Steiner, and V. Dhar, "A Planning Problem: Revisable Academic Course Scheduling," Technical Report, Microelectronics and Computer Technology Corporation TR AI-020-89, 1989.

[13] Petrie, C., "Reason Maintenance in Expert Systems," Kuenstliche Intelligence, June, 1989. Also, Microelectronics and Computer Technology Corporation TR ACA-AI-021-89.