MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A. I. Memo 461

February 1978

A GLIMPSE OF TRUTH MAINTENANCE

by

Jon Doyle*

Abstract:

Many procedurally-oriented problem solving systems can be viewed as performing a mixture of computation and deduction, with much of the computation serving to decide what deductions should be made. This results in bits and pieces of deductions being strewn throughout the program text and execution. This paper describes a problem solver subsystem called a truth maintenance system which collects and maintains these bits of deductions. Automatic functions of the truth maintenance system then use these pieces of "proofs" to consistently update a data base of program beliefs and to perform a powerful form of backtracking called dependency-directed backtracking.

* Fannie and John Hertz Foundation Fellow

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research was provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643 and in part by the National Science Foundation under grant MCS77-04828.

Acknowledgements

I wish to thank Gerald Jay Sussman, Richard M. Stallman, Guy L. Steele Jr., Johan de Kleer, Drew McDermott, David McAllester, Scott Fahlman, Howard Shrobe, Charles Rich, Marilyn Matz, Richard Brown, Richard Waters and Vaughan Pratt for ideas, comments and advice. I also thank the Fannie and John Hertz Foundation for supporting my research with a graduate fellowship.

Contents

Introduction 3
Representation of Knowledge about Belief 4
Default Assumptions 5
Sets of Alternatives 7
Dependency-Directed Backtracking 8
Truth Maintenance Mechanisms 11
Applications 12
References 13

Introduction

Many procedurally-oriented problem solving systems can be viewed as performing a mixture of computation and deduction, with much of the computation serving to decide what deductions should be made. This results in bits and pieces of deductions being strewn throughout the program text and execution. (I am indebted to Drew McDermott for this imagery.) This paper describes a problem solver subsystem called a truth maintenance system which collects and maintains these bits of deductions. Automatic functions of the truth maintenance system then use these pieces of "proofs" to consistently update a data base of program beliefs and to perform a powerful form of backtracking called dependency-directed backtracking.

Truth maintenance systems record and maintain proofs. The proofs are made up of justifications connecting data structures called nodes. Nodes will typically represent assertions, rules, or other program beliefs. Nodes may have several justifications, each of which represents a different method of deriving belief in the node. Some nodes may be designated to be hypotheses. For each node, the truth maintenance system computes whether or not belief in the node is justified by the existence of a non-circular proof from the basic hypotheses and the set of recorded justifications. The set of such non-circular proofs is recorded as the well-founded support of the believed nodes.

When the truth maintenance system is given a new justification to record, it checks to see if the new justification can be used to provide well-founded support for some currently unsupported node. If so, the node is marked as believed, and the new justification is attached to the node as its well-founded support. Previously existing justifications which connect the newly justified node to other nodes may now allow well-founded support for some of these other nodes to be derived. To do this, the process of truth maintenance is invoked. This consists of scanning from the newly justified node through the recorded justifications to check for any other nodes which can be supplied with proofs.

Because their knowledge is incomplete, problem solvers must frequently make assumptions for the sake of argument in order to proceed. Such assumptions take the form of non-monotonic justifications in the truth maintenance system. This type of justification is used to make a proof of a node which is based in part on the nonexistence of proofs for

some other node. The term "non-monotonic" means that new proofs can invalidate previous proofs. This is in contrast to the normal property of systems of mathematical logic in which the validity of a proof is not affected by the addition of new axioms. For example, non-monotonic justifications can be used in a way analogous to the use of the THNOT primitive of Micro-PLANNER by basing belief in a node representing a statement P on the lack of a proof for the node representing the statement ~P. If a proof of ~P is subsequently discovered, the process of truth maintenance will be invoked to undo the existing proof of P and of any nodes based on belief in P.

Representation of Knowledge about Belief

A node may have several justifications for belief. Each of these justification may be considered a predicate of other nodes. The node is believed if at least one of these justifications is valid. The conditions for validity of justifications are described below. We say that a node which is believed is *in*, and that a node without a valid justification is *out*. The distinction between *in* and *out* is not that of *true* and *false*. The former denote conditions of knowledge about reasons for belief; the latter, belief in a piece of knowledge or its negation.

Two basic forms of justifications suffice. The first is the support-list justification, which is of the form

(AND (IN <*in*list>) (OUT <*out*list>)).

A support-list justification is valid if each node in its *in*list is *in*, and each node in its *out*list is *out*. A support-list justification can be used to represent several types of deductions. When both the *in*list and *out*list are empty, the justification forms a premise justification. A premise justification is always valid, and so the node it justifies will always be believed. Normal deductions are represented by support-list justifications with empty *out*lists. These represent monotonic deductions of the justified node from the belief in the nodes of the *in*list. Assumptions are nodes whose well-founded support is a support-list justification with a nonempty *out*list. These justifications can be interpreted by viewing the nodes of the *in*list as the reasons for making the assumption; the nodes of the *out*list represent the specific incompleteness of knowledge authorizing the assumption.

The second form of justification is the conditional-proof justification, which is of the form

(CP <consequent> <*in*hypotheses> <*out*hypotheses>).

A node justified by such a justification represents an implication, which is derived by a conditional proof of the consequent node from the hypothesis nodes. A justification of this form is valid if the consequent node is *in* whenever each node of the *in*hypotheses is *in* and each node of the *out*hypotheses is *out*. Except in a few esoteric uses, the set of *out*hypotheses is empty. Standard conditional proofs in natural deduction systems specify a single set of hypotheses, which correspond to our *in*hypotheses. The truth maintenance system requires that the set of hypotheses be divided into two disjoint subsets, since nodes may be derived both from some nodes being *in* and other nodes being *out*. Some natural deduction systems also allow a set of consequents in a conditional proof. For efficiency, conditional proofs are restricted in a truth maintenance system to a single consequent node.

Default Assumptions

Support-list and conditional proof justifications can be employed to represent more complex relationships between beliefs. The relationships presented below describe choice structures, which are useful in explicitly programming parts of the control structure of the problem solver into dependency relationships between control assertions. In these, the justifications are arranged to select one default or alternative from a set of alternatives. This choice is backtrackable. That is, if a contradiction is derived which depends on the choice, the dependency-directed backtracking mechanism will cause a new alternative to be chosen from the set of alternatives. Other choice structures (for example, equivalence class representative selectors) which are not backtrackable will not be described here. (See [Doyle 1978].)

One very common technique used in problem solving systems is to specify a default choice for the value of some quantity. This choice is made with the intent of overriding it if either a good reason is found for using some other value, or if making the default choice leads to an inconsistency. In the case of a binary choice, such a default assumption can be represented by believing a node if the node representing its negation is *out*. When the default is chosen from a set of alternatives, the following generalization of

the binary case is used. Let $\{F_1, \ldots, F_n\}$ be the set of the nodes which represent each of the possible values of the choice. Let G be the node which represents the reason for making the default assumption. Then F_i may be made the default choice by providing it with the justification

(AND (IN G) (OUT $F_1 \ldots F_{i-1} F_{i+1} \ldots F_n$)).

If no information about the choice exists, there will be no reasons for believing any of the alternatives except F_i. Thus F_i will be *in* and each of the other alternatives will be *out*. If some other alternative receives a valid justification from other sources, that alternative will become *in*. This will invalidate the support of F_i, and F_i will become *out*. If a contradiction is derived from F_i, the dependency-directed backtracking mechanism will recognize that F_i is an assumption by means of its dependence on the other alternatives being *out*. (See the section on dependency-directed backtracking for an explanation of this.) The backtracker may then justify one of the other alternatives at random, causing F_i to go *out*. In effect, backtracking will cause the removal of the default choice from the set of alternatives, and will set up a new default assumption structure from the remaining alternatives.

If the complete set of alternatives from which the default assumption is selected is not known *a priori*, but is to be discovered piecemeal, a slightly different structure is necessary. The following structure allows an extensible set of alternatives underlying the default assumption. Such extensibility is necessary, for example, when specifying a number as a default due to the large set of possible alternatives. For cases like this the following structure may be used instead. Retaining the above notation, let $\sim F_i$ be a new node which will represent the negation of F_i. We will arrange for F_i to be believed if $\sim F_i$ cannot be proven, and will set up justifications so that if F_j is distinct from F_i, F_j will imply $\sim F_i$. This is done by giving F_i the justification

(AND (IN G) (OUT $\sim F_i$)),

and by giving $\sim F_i$ a justification of the form

(AND (IN F_j) (OUT))

for each alternative F_j distinct from F_i. As before, F_i will be assumed if no reasons for using any other alternative exist. Furthermore, new alternatives can be added to the set simply by giving $\sim F_i$ a new justification corresponding to the new alternative. This structure for default assumptions will behave as did the fixed structure in the case of an unselected alternative receiving independent support. Backtracking, however, has a

different effect. If a contradiction is derived from the default assumption supported by this structure, $\sim F_i$ will be justified so as to make F_i become *out*. If this happens, no alternative will be selected to take the place of the default assumption. The extensible structure requires an external mechanism to construct a new default assumption whenever the default is ruled out.

Sets of Alternatives

The default assumption structures allow a choice from a set of alternatives, but do not specify the order in which new alternatives are to be tried if the initial choice is wrong. Such advice can be embedded in a linear ordering on the set of alternatives. Linearly ordered sets of alternatives are useful whenever heuristic information is available for making a choice. One way such situations arise is by using recommendation lists in Micro-PLANNER. Another use is in heuristically choosing the value of some quantity, such as the state of a transistor or the day of the week for a meeting.

If it is certain that rejected alternatives are rejected permanently and will never again be believed, the linear ordering on the set of alternatives can be specified by a controlled sequence of default assumptions. This can be implemented in a ladder-like structure of justifications by justifying each F_i with

(AND (IN G $\sim F_{i-1}$) (OUT $\sim F_i$)),

where G is the reason for the set of alternatives. The first alternative F_1 will be selected initially. As alternatives are ruled out by their negations being justified, the next alternative in the list will be assumed.

If previously rejected alternatives can be independently rejustified, a more complicated structure is necessary. This type of set of alternatives can be described by the following justifications. For each alternative A_i, three new nodes should be created. These new nodes are PA_i (meaning "A_i is a possible alternative"), NSA_i (meaning "A_i is not the selected alternative"), and ROA_i (meaning "A_i is a ruled-out alternative"). Each PA_i should be justified with the reason for including A_i in the set of alternatives. Each ROA_i is left unjustified. Each A_i and NSA_i should be given justifications as follows:

A_i: (AND (IN PA_i NSA_1 ... NSA_{i-1}) (OUT ROA_i))
{or: (AND <is alternative> <no better is selected> <is not ruled out>)}

NSA_i: (AND (IN) (OUT PA_i)) , (AND (IN ROA_i) (OUT))
{or: (OR <is not a valid alternative> <is ruled out>)}

With this structure, processes can independently rule in or rule out an alternative by justifying the appropriate alternative node or ruled-out-alternative node.

This structure is also extensible. New alternatives may be added simply by constructing the appropriate justifications as above. These additions are restricted to appearing at the end of the order. That is, new alternatives cannot be spliced into the linear order between two previously inserted alternatives.

Dependency-Directed Backtracking

The truth maintenance system supports a powerful form of backtracking called dependency-directed backtracking. This method of backtracking is used to restore consistency of beliefs when assumptions based on incomplete knowledge lead to contradictions. Consistency is restored by using the contradiction to derive new knowledge. This new knowledge then fills in some of the incompletenesses which previously supported one or more assumed beliefs. This causes the truth maintenance system to retract belief in those assumptions.

To signal the existence of an inconsistency, nodes may be declared to be contradictions. Contradictions, as beliefs, have the semantics of *false*. During truth maintenance, nodes for which support is derived are checked to see if they are marked as contradictions. The derivation of belief in a contradiction indicates the inconsistency of the set of beliefs used in deriving the contradiction. To restore the (apparent) consistency of the set of beliefs, the truth maintenance system notifies the dependency-directed backtracker of the contradiction.

The backtracking process consists of tracing backwards through the well-founded support of the contradiction node to find the causes of the contradiction. The backtracker

presumes that all inconsistencies are due to the presence of assumptions based on incomplete knowledge. It therefore expects that all monotonically justified beliefs are correct, and searches only for the set of assumptions underlying the contradiction.

Belief in at least one of the assumptions underlying the contradiction must be retracted to remove the contradiction. This is accomplished by adding knowledge where knowledge was lacking before; that is, by providing a new justification for belief in one of the nodes that supported the assumption by being *out*. The justification used is that the assumption, when combined with the other assumptions, provides support for the contradiction. Since other beliefs besides the assumptions may have played a role in deriving the contradiction, the inconsistency of the set of assumptions is valid only under certain circumstances -- those in which the combination of the set of assumptions together with those other beliefs provides support for the contradiction. This is the statement of a conditional proof. That is, the justification for not believing a particular assumption is that the other assumptions are believed, and that if all the assumptions are believed, the contradiction follows. Thus the justification used to retract an assumption is the conditional proof of the contradiction from the complete set of assumptions, together with belief in the other assumptions.

In more detail, the first step of the backtracking process is the recognition of an inconsistency through derivation of well-founded support for a contradiction node. The well-founded support of the contradiction node is traced backwards to collect the set of assumptions supporting the contradiction. The third step of backtracking is the summarization of the inconsistency of the set of assumptions underlying the contradiction. Suppose that S = {A, B, ... , Z} is the set of inconsistent assumptions. The backtracker then creates a nogood, a new node signifying that S is inconsistent. The nogood represents the fact that

$$A \wedge \ldots \wedge Z \supset \mathit{false},$$

or alternatively, that

$$\sim (A \wedge \ldots \wedge Z).$$

S is called the nogood-set of the nogood. The summarization is accomplished by justifying the nogood with a conditional proof of the contradiction relative to the set of assumptions. In this way, the inconsistency of the set of assumptions is recorded as a node which will be believed even after the contradiction has been disposed of by the retraction of some

hypothesis.

The last step of backtracking uses the summarized cause of the contradiction, represented by the nogood, to both retract one of the inconsistent assumptions and to prevent future contradictions for the same reasons. This is accomplished by deriving new justifications for the *out* nodes underlying the inconsistent assumptions. The new justifications will cause one of these *out* facts to become *in*, thereby causing one of the offensive assumptions to become *out*. This step is reminiscent of the justification of results on the basis of the occurrence of contradictions in reasoning by *reductio ad absurdum*.

These new justifications are constructed as follows. Let the inconsistent assumptions be $A_1, \ldots, A_n$. Let $S_{i1}, \ldots, S_{ik}$ be the *out* nodes of the justification supporting belief in the assumption A_i. To effect the retraction of one of the assumptions, A_i, justify S_{i1} with the predicate

(AND (IN NG A_1 ... A_{i-1} A_{i+1} ... A_n) (OUT S_{i2} ... S_{ik})),

that is,

(AND (IN <nogood> <other assumptions involved>)
(OUT <other denials of this assumption>))

This will ensure that the justification supporting A_i by means of this set of *out* nodes will no longer be valid whenever the nogood (NG) and the other assumptions are believed. This process is repeated for each assumption in the inconsistent set. If the assumptions and the contradiction are still believed following this, the backtracking process is repeated. Backtracking halts when the contradiction becomes *out*, or when no assumptions can be found underlying the contradiction.

Dependency-directed backtracking improves on traditional backtracking mechanisms in two ways; irrelevant assumptions are ignored, since the set of inconsistent beliefs is determined by tracing dependencies; and the cause of the contradiction is summarized in terms of this set of inconsistent assumptions as a conditional proof which remains valid after the contradiction itself has been removed.

Truth Maintenance Mechanisms

Consider the situation in which the node *F* represents the assertion

"(= (+ X Y) 4)",

G represents

"(= X 1)",

and *H* represents

"(= Y 3)".

If both *F* and *G* are *in*, then belief in *H* can be justified by (AND (IN F G) (OUT)). This justification will cause *H* to become *in*. If *G* subsequently becomes *out* due to changing hypotheses, and if *H* becomes *in* by some other justification, then *G* can be justified by (AND (IN F H) (OUT)). Suppose the justification supporting belief in *H* then becomes invalid. If the decision to believe a node is based on a simple evaluation of each of the justifications of the node, then both *G* and *H* will be left *in*. This happens because the two justifications form circular proofs for *G* and *H* in terms of each other. These justifications are mutually satisfactory if *F*, *G* and *H* are *in*.

This example points out one of the major concerns in truth maintenance processing; the avoidance of using circular proofs to support beliefs. This is the reason why well-founded support is maintained.

Essentially three different kinds of circularities which can arise in purported proofs. The first and most common is a circularity in which all nodes involved can be considered *out* consistently with their justifications. Such circularities arise routinely through equivalences and simultaneous constraints. The above algebra example falls into this class of circularity.

The second type of circularity is one in which at least one of the nodes involved must be *in*. An example is that of two nodes *F* and *G*, such that *F* has an justification of the form (AND (IN) (OUT *G*)), and *G* has an justification of the form (AND (IN) (OUT *F*)). Here either *F* must be *in* and *G* *out*, or *G* must be *in* and *F* *out*. This type of circularity arises in defining some types of sets of alternatives. Other types of ordered alternative structures avoid such circularities.

The third form of circularity which can arise is the unsatisfiable circularity. In this type of circularity, no assignment of support-statuses to nodes is consistent with their justifications. An example of such a circularity is a node *F* with the justification (AND (IN) (OUT *F*)). This justification implies that *F* is *in* if and only if *F* is *out*. Unsatisfiable circularities are bugs, indicating a misorganization of the knowledge of the program using the truth maintenance system. Unsatisfiable circularities are violations of the semantics of *in* and *out*, which can be interpreted as meaning that the lack of reasons for belief in a node is equivalent to the existence of reasons for belief in the node. (It has been my experience that such circularities are most commonly caused by confusing the concepts of *in* and *out* with those of *true* and *false*. For instance, the above example could be produced by this misinterpretation as an attempt to assume belief in the node *F* by giving it the justification (AND (IN) (OUT *F*)).)

The details of the truth maintenance process will not be pursued here. Many details of this, and of several other processes such as the procedure for dealing with conditional proofs are discussed in [Doyle 1978]. David McAllester [1977] has developed an attractive alternate data structure for the proofs maintained by the truth maintenance system. This allows several algorithms to be combined into one simplified process.

Applications

There are several applications of truth maintenance systems in problem solving systems. The most immediate application is that of maintaining the consistency of a data base in the presence of assumptions based on incomplete knowledge. (See [Stallman and Sussman 1977].)

Truth maintenance systems also apply to systems which generate explanations. Problem solvers which record the reasons for their beliefs can use these records to justify their actions and beliefs to a human (or otherwise) user. (See [Sussman and Stallman 1975, Stallman and Sussman 1977, Doyle 1978].)

A crucial aspect of the problem of explanation is that levels of detail must be separated in the explanations produced by hierarchical systems. A truth maintenance

system can be used to automatically perform such a structuring of arguments. The method used for this is that of applying conditional proofs to factor unwanted low-level details from explanations. When such factoring is done at each level, a hierarchical structure emerges in explanations. (See [Doyle 1978] for more details.)

Another application of truth maintenance systems is in modelling. Most modelling systems specify the effects of actions only in terms of the primary effects of the actions. Many secondary or derived effects remain unspecified. By recording the reasons for derived knowledge, a modelling system can employ a truth maintenance system in updating the derived portions of its model. (See [Fikes 1975, Hayes 1975, McDermott 1977, London 1977].)

The final application we mention is that of control. A truth maintenance system supplies the powerful method of dependency-directed backtracking for use in controlling the actions taken by a problem solver. Another use is in separating the reasons for control decisions from the reasons for beliefs derived in response to those control decisions. (See [Stallman and Sussman 1977, Doyle 1978, de Kleer, Doyle, Steele and Sussman 1977, de Kleer, Doyle, Rich, Steele and Sussman 1978].)

References

[de Kleer, Doyle, Rich, Steele and Sussman 1978]
Johan de Kleer, Jon Doyle, Charles Rich, Guy L. Steele Jr., and Gerald Jay Sussman, "AMORD: A Deductive Procedure System," MIT AI Lab, Memo 435, January 1978.

[de Kleer, Doyle, Steele and Sussman 1977]
Johan de Kleer, Jon Doyle, Guy L. Steele Jr., and Gerald Jay Sussman, "Explicit Control of Reasoning," MIT AI Lab, Memo 427, June 1977.

[Doyle 1978]
Jon Doyle, "Truth Maintenance Systems for Problem Solving," MIT AI Lab, TR-419, January 1978.

[Fikes 1975]
Richard E. Fikes, "Deductive Retrieval Mechanisms for State Description Models," *IJCAI4*, September 1975, pp. 99-106.

[Hayes 1975]
Philip J. Hayes, "A Representation for Robot Plans," *IJCAI4*, September 1975, pp. 181-188.

[London 1977]
Phil London, "A Dependency-Based Modelling Mechanism for Problem Solving," Computer Science Department TR-589, University of Maryland, November 1977.

[McAllester 1977]
David A. McAllester, "Implementing Truth Maintenance Systems with Bidirectional Disjunctive Clauses," MIT EE&CS Bachelor's Thesis proposal, October 1977.

[McDermott 1977]
Drew V. McDermott, "Flexibility and Efficiency in a Computer Program for Designing Circuits," MIT AI Lab, TR-402, June 1977.

[Stallman and Sussman 1977]
Richard M. Stallman and Gerald Jay Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artificial Intelligence*, Vol. 9, No. 2, (October 1977), pp. 135-196.

[Sussman and Stallman 1975]
Gerald Jay Sussman and Richard Matthew Stallman, "Heuristic Techniques in Computer-Aided Circuit Analysis," *IEEE Transactions on Circuits and Systems*, Vol. CAS-22, No. 11, November 1975, pp. 857-865.