

A GLIMPSE OF TRUTH MAINTENANCE**

Jon Doyle
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
545 Technology Square
Cambridge, Massachusetts 02139

To choose their actions, reasoning programs must be able to draw conclusions from limited information and subsequently revise their beliefs when discoveries invalidate previous assumptions. In the *Truth Maintenance System* (TMS) I introduce a new notion of "reason for belief" which permits the mechanization of these abilities as a problem solver subsystem. The TMS records and maintains the reasons for program beliefs. These recorded reasons are useful in constructing explanations of program beliefs and actions, and in guiding the course of action of a problem solver. This paper outlines some of the structure and applications of the TMS.

1. INTRODUCTION

One important problem faced by reasoning programs is the need to make decisions based on limited information. This problem arises both in programs interacting with an external environment and in contemplative programs searching a data base for an answer to some question. There are two consequences of this need to predict; the program must have some way to make decisions based on limited information, and the program must have some way to revise its beliefs if these decisions are found to be in error. The first of these abilities is provided by utilizing epistemic classifications of possible program beliefs so that conclusions may be drawn from the lack of belief as well as from other beliefs. The second ability is in general a very complex problem for which no complete solutions are known. (See [3,7,8] for surveys of the problem.) However, the simpler problem of

revising beliefs based on limited information is solvable by recording the reasons for each program belief. These records can be used to find the set of extant beliefs by determining which beliefs have valid reasons. These recorded reasons also are useful in resolving conflicts that arise when limited knowledge gives rise to incompatible conclusions. This paper describes a mechanization of these abilities, embodied in a general-purpose problem solver subsystem called the *Truth Maintenance System* (TMS). The TMS is based on a new analysis of what constitute "reasons" for beliefs and "assumed" beliefs. In this analysis, all beliefs are derived from the existence of valid reasons for them, and assumptions are statements which are believed because some other statements are not believed.

The TMS records and maintains "proofs" of program beliefs. It manipulates two data structures; *nodes*, which represent beliefs, and *justifications*, which represent reasons for beliefs. The fundamental actions the TMS can be called upon to perform are the creation of a new node, to which the problem solving program can attach the statement of a belief, and the addition of a new justification to a node, to represent assertion of the belief associated with the node by some rule or procedure in the problem solver. The addition of new justifications may invoke the automatic procedure of *truth maintenance* to make any revisions necessary in the set of beliefs. The TMS revises beliefs by using the recorded justifications to compute non-circular proofs of beliefs from basic hypotheses. These proofs distinguish one or more justifications as the *well-founded support* for each believed node, and are used during truth maintenance to determine the set of beliefs to update by finding those nodes whose

* This research was conducted at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-75-C-0643, and in part by NSF grant MCS77-04828.

** My new paper [2] presents the foundations, mechanisms, and applications of the TMS in much greater depth, detail, and clarity than is possible here. That paper also mentions the many related works by others too numerous to cite in this short a paper.

well-founded support depends on changed beliefs. These proofs allow another process, *dependency-directed backtracking*, to resolve conflicts by tracing the well-founded supports of conflicting beliefs to remove one of the assumptions causing the conflict and to make a record used to prevent similar future conflicts.

Minsky [6] criticized traditional logics for their *monotonicity*, their lack of any ability to control inferences by the addition of new axioms. With this in mind the TMS employs a special type of justification, called a *non-monotonic justification*, to draw conclusions based on limited or incomplete knowledge. This type of justification allows belief in a node to be based not only on other beliefs, as occurs in the standard forms of deduction and reasoning, but also on lack of belief in certain nodes. For example, a node N-1 representing a statement P might be justified on the basis of a lack of belief in a node N-2 representing the belief $\sim P$. (Distinct nodes are used to represent P and $\sim P$.) In this case, the TMS would have N-1 believed as long as N-2 was not believed, and we would call N-1 an *assumption*. (More generally, by assumption we mean any node whose well-founded support is a non-monotonic justification.)

As a small example, suppose an office scheduling program is considering holding a meeting M on Wednesday. To do this, the program assumes that the meeting is on Wednesday. The data base of the program includes a rule which draws the conclusion that due to regular commitments, any meeting on Wednesday must occur at 1:00 PM. However, the fragment of the schedule for the week constructed so far has something else scheduled for that time already, and so another rule in the data base concludes that the day for the meeting cannot be Wednesday. These beliefs might be notated as follows:

Node	Statement	Justification
N-1	DAY (M) = WEDNESDAY	(SL () (N-2))
N-2	DAY (M) $\neq$ WEDNESDAY	
N-3	TIME (M) = 13:00	(SL (R-37 N-1) ())

As seen in the above notation for justifications, each justification consists of two lists. The meaning of the notation is that the statement depends on each of the nodes in the first list being believed, and on each of the nodes in the second list not being believed. Since there is no known justification for N-2, it is not believed. The justification for N-1 specifies that it depends on the lack of belief in N-2, and so N-1 is believed. The justification for N-3 shows that it is believed due to rule R-37 acting on N-1. When the assumption N-1 is rejected by some rule,

N-2	DAY (M) $\neq$ WEDNESDAY	(SL (R-9 N-7 N-8) ())
-----	--------------------------	-------------------------

where N-7 and N-8 represent the day and time of some other engagement, the TMS will revise the beliefs so that N-1 and N-3 are not believed.

2. REPRESENTATION OF KNOWLEDGE ABOUT BELIEF

A node may have several justifications, each of which represents a different reason for belief in the node. 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 has a valid justification 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 classification denotes conditions of knowledge about reasons for belief; the existence or non-existence of valid reasons. *True* and *false*, on the other hand, classify statements according to truth value independent of any reasons for belief. In this way, there can be four states of knowledge about a proposition P, corresponding to the node representing P being *in* or *out* and the node representing $\sim P$ being *in* or *out*.

There are two basic forms of justifications. These are inspired by the typical forms of arguments in natural deduction inference systems. A sample proof in such a system might be as follows:

Line	Statement	Justification	Dependencies
1.	A $\supset$ B	Premise	(1)
2.	B $\supset$ C	Premise	(2)
3.	A	Hypothesis	(3)
4.	B	MP 1,3	(1,3)
5.	C	MP 2,4	(1,2,3)
6.	A $\supset$ C	Discharge 3,5	(1,2)

Each step of the proof has a line number, a statement, a justification, and the set of line numbers the statement depends on. Premises and hypotheses depend on themselves, and other lines depend on the set of premises and hypotheses derived from their justifications. The above proof proves A $\supset$ C from the premises A $\supset$ B and B $\supset$ C by hypothesizing A and concluding C. The assumption A is then discharged to provide the proof of A $\supset$ C. There are two effects that justifications can have on the set of dependencies in natural deduction systems; either the justifications can sum the dependencies of the referenced lines (as in line 4), or they can subtract the dependencies of some lines from those of other lines (as in line 6). The two types of justifications used in a TMS account for these effects on dependencies. A *support-list (SL) justification* says that the justified node depends on a set of other nodes, and thus in effect sums the dependencies of the referenced nodes. A *conditional-proof*

(CP) justification says that the node it justifies depends on the validity of a certain hypothetical argument, and as in the example above, subtracts the dependencies of some nodes (the hypotheses of the hypothetical argument) from the dependencies of others (the conclusion of the hypothetical argument). These two types of justifications can be used to construct a variety of forms of dependency relationships. For example, we might rewrite the example in terms of TMS justifications as follows (here ignoring the difference between premises and hypotheses, and ignoring the inference rule MP):

N-1	A>B	(SL () ())
N-2	B>C	(SL () ())
N-3	A	(SL () ())
N-4	B	(SL (N-1 N-3) ())
N-5	C	(SL (N-2 N-4) ())
N-6	A>C	(CP N-5 (N-3) ())

The support-list justification is of the form

(SL <inlist> <outlist>).

A SL-justification is valid if each node in its *inlist* is *in*, and each node in its *outlist* is *out*. A SL-justification can be used to represent several types of deductions. When both the *inlist* and *outlist* are empty, we say 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 *outlists*. These represent monotonic deductions of the justified node from belief in the nodes of the *inlist*. *Assumptions* are nodes whose well-founded support is a support-list justification with a nonempty *outlist*. These justifications can be interpreted by viewing the nodes of the *inlist* as comprising the reasons for making the assumption; the nodes of the *outlist* represent the specific incompletenesses of knowledge authorizing the assumption.

The conditional-proof justification takes the form

(CP <consequent> <inhypotheses> <outhypotheses>).

A node justified by such a justification represents an implication, whose support 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 (a) each node of the *inhypotheses* is *in* and (b) each node of the *outhypotheses* is *out*. Except in a few esoteric uses, the set of *outhypotheses* is empty. Standard conditional proofs in natural deduction systems specify a single set of hypotheses, which correspond to our *inhypotheses*. 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*.

3. DEFAULT ASSUMPTIONS

One very common technique used in problem solving systems is to specify a default choice for the value of some quantity. (See, for example, [8].) 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. The assumption of the day of the week for a meeting in the first example above is such a default assumption.

In the case of a binary choice, 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 $\{F_1, \dots, F_n\}$ be the set of the nodes which represent each of the possible values of the choice. Let G be a 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

(SL (G) ($F_1 \dots F_{i-1} F_{i+1} \dots F_n$)).

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 result of backtracking may be to justify one of the other alternatives, say F_j , causing F_i to go *out*. The justification for F_j will be of the form

(SL <various things> <remainders>)

where the remainders are the F_k 's remaining after F_i and F_j are taken away. In effect, backtracking will cause the removal of the default choice with the set of alternatives, and will set up a new default assumption structure from the remaining alternatives. As a concrete example, our scheduling program might default a meeting day as follows:

N-1 DAY (M) = MONDAY

N-2 DAY (M) = WEDNESDAY (SL () (N-1 N-3))
 N-3 DAY (M) = FRIDAY

In this example, Wednesday is assumed to be the day of the meeting M, with Monday and Friday being the alternatives. Wednesday will be the default choice until a valid reason is supplied for either Monday or Friday.

If the complete set of alternatives from which the default assumption is to be chosen cannot be known in advance but must be discovered piecemeal, a slightly different structure is necessary. This ability to extend the set of alternatives is necessary, for example, when the default is a number, 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

(SL (G) ($\sim F_i$)),

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

(SL (F_j) (1))

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 the extensible 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 current default is ruled out. For example, a census program might make assumptions about the number of children in a family as follows:

N-1 #-CHILDREN (F) = 2 (SL () (N-2))
 N-2 #-CHILDREN (F) = 2 (SL (N-3) (1))
 (SL (N-4) (1))
 N-3 #-CHILDREN (F) = 0
 N-4 #-CHILDREN (F) = 1

With this system of justifications, N-1 would be believed

because no different number of children is known. If it turns out that the family has, for example, 5 children a new statement would have to be made, along with a new justification of N-2 in terms of this new statement.

4. DEPENDENCY-DIRECTED BACKTRACKING

Making assumptions admits the possibility of making errors. When a contradiction or other inconsistent state of the data base occurs, the TMS employs a process called dependency-directed backtracking to find and remove incorrect assumptions so as to restore consistency. There are several steps involved in dependency-directed backtracking, but first the inconsistency must somehow be signalled to the TMS, as there is no built-in notion of inconsistency. This signalling consists of informing the TMS that a node represents an inconsistency. With this knowledge, the TMS will try to restore consistency whenever the node comes *in* by rejecting enough assumptions to force the node *out*. Any node may be marked for such treatment. A node so marked is called a *contradiction*.

The steps of dependency-directed backtracking are as follows. First, the well-founded support of the contradiction node is traced backwards to find the set of assumptions (nodes with a non-monotonic justification as their well-founded support) underlying the contradiction. Belief in at least one of these assumptions must be retracted to remove the contradiction. This is done by creating a new justification for one of the *out* nodes underlying one of the assumptions. Since the backtracker may be mistaken in its assignment of blame to that assumption, the justification used to retract the assumption must indicate the alternatives that were available but not utilized by the backtracker. Thus the new justification includes (a) the reason why the contradiction occurred and (b) the other assumptions involved. Thus the second step of the backtracking process is to construct a node recording the reason why the contradiction occurred, and the third step is to use this node and the other assumptions in justifying an *out* node supporting the assumption selected for removal.

In more detail, the first step of the backtracking process traces backwards through the well-founded support of the contradiction node to collect the set of "maximal" assumptions supporting the contradiction. Not all assumptions found by tracing the well-founded support are used; instead, only those assumptions which do not support other assumptions underlying the contradiction as well. That is, the well-founded support relationships induce a natural partial-ordering on nodes, where one node is said to be "lower" than a second node if the first occurs in the second's well-founded support. The maximal assumptions are then

those assumptions which are maximal in this partial order. Only this "front line" of assumptions is used because if the reason for revoking a lower-level assumption involves a higher-level assumption, then the removal of the lower-level assumption would cause truth maintenance to remove the higher-level assumption that it supports, so the reason for removing the lower-level assumption would not hold up. This reflects the fact that there may not be enough information to definitely rule out a lower-level assumption.

The second step summarizes the reason for the contradiction in terms of the set of selected assumptions. Let $S = \{A_1, \dots, A_n\}$ indicate the set of inconsistent assumptions. The backtracker then creates a node called a *nogood*, a new node signifying that S is inconsistent. Since contradiction nodes really represent the false statement, the *nogood* node can be taken to represent

$$A_1 \wedge \dots \wedge A_n \supset \text{false},$$

or alternatively,

$$(1) \quad \sim (A_1 \wedge \dots \wedge A_n).$$

S is recorded as the *nogood-set* of the *nogood*. This meaning for the *nogood* node is produced by justifying it with the conditional proof of the contradiction node relative to the assumption nodes, that is, with the justification

$$(2) \quad (\text{CP } \langle \text{contradiction node} \rangle S \ ()).$$

In this way, the inconsistency of the set of assumptions will be remembered even after the contradiction has been resolved by the retraction of some hypothesis.

The final step is selecting an assumption A_i (the "culprit") from S and justifying one of the *out* nodes listed in its well-founded supporting justification. (If these underlying *out* nodes are thought of as "denials" of the assumption, then this step is much like reasoning by *reductio ad absurdum*.) Let NG be the *nogood*, and let the inconsistent assumptions be $A_1, \dots, A_n$. Let $D_1, \dots, D_k$ be the *out* nodes appearing in the justification which supports belief in the assumption A_i . This justification for the assumption can be invalidated by justifying D_1 with the justification

$$(3) \quad (\text{SL } (NG \ A_1 \dots A_{i-1} \ A_{i+1} \dots A_n) \ (D_2 \dots D_k)).$$

This justification is valid whenever the *nogood* and other assumptions are believed and the other "denials" of the culprit are not believed. If the choice of culprit was in error, then another contradiction will occur in the future involving D_1 , and by this justification will be led to suspect the

remaining assumptions, as well as D_1 if there are any other *out* nodes listed in its justification. If, by means of other previously existing justifications, the current contradiction is still *in* following the addition of this justification, backtracking is repeated. Presumably the new invocation of the backtracking process will find that the previous culprit is no longer an assumption. Backtracking halts when the contradiction becomes *out*, or when no assumptions can be found underlying the contradiction.

As an example, consider a program scheduling a meeting, preferably at 10 AM in either room 813 or 801. This might be represented as:

N-1	TIME(M) = 1000	(SL () (N-2))
N-2	TIME(M) = 1000	
N-3	ROOM(M) = 813	(SL () (N-4))
N-4	ROOM(M) = 801	

With these justifications, N-1 and N-3 are *in*, and the other two nodes are *out*. If some previously scheduled meeting exists, it might cause this combination of time and room for the meeting to be ruled out by means of a contradiction.

N-5	CONTRADICTION	(SL (N-1 N-3) ())
-----	---------------	--------------------

The dependency-directed backtracking system then traces the well-founded support of the contradiction to find that it depends on two assumptions, N-1 and N-3, both of which are maximal.

N-6	NOGOOD N-1 N-3	(CP N-5 (N-1 N-3) ())
N-4	ROOM(M) = 801	(SL (N-6 N-1) ())

A *nogood* node is created which means, in accordance with form (1) above,

$$\sim (\text{TIME(M) = 1000} \wedge \text{ROOM(M) = 813})$$

and this *nogood* is given a justification corresponding to form (2) above. The assumption N-3 is selected arbitrarily as the culprit, and is rejected by providing its only *out* supporting node, N-4, with a justification of the form (3) above. Following this, N-1, N-4, and N-6 are *in*, and N-2, N-3, and N-5 are *out*. N-6, the *nogood* node, has an always-valid CP-justification since the contradiction node N-5 depends directly on the two assumptions N-1 and N-3 without any additional beliefs intervening. If some further consideration determines that room 801 cannot be used after all, another contradiction node could be created to force a different choice.

N-7	CONTRADICTION	(SL (N-4) ())
-----	---------------	----------------

N-8	NOGOOD N-1	(CP N-7 (N-1) ())
N-2	TIME (M) ≈ 1000	(SL (N-8) ())

Tracing backwards from N-7 through N-4, N-6, and N-1, the backtracker finds that the contradiction depends on only one assumption, N-1. The nogood node N-8 is created and justified with a CP-justification which in effect is equivalent to the SL-justification

(SL (N-6) ()),

since the nogood N-6 contributes to the contradiction but does not itself depend on the assumption N-1. The revocation of the assumption N-1 removes N-5, the previous objection to the choice of room, so at the close of this bit of decision making N-2, N-3, N-6, and N-8 are *in*, and N-1, N-4, N-5, and N-7 are *out*.

There are a number of variations on this particular scheme for dependency-directed backtracking. All of these variations are considerable improvements over the chronological backtracking systems used in classical systems like MICRO-PLANNER and many early theorem provers. The improvements stem from the non-chronological nature of dependency-directed backtracking, in which the support relationships rather than the temporal orderings determine the choices responsible for an error. Another improvement is that the cause of the contradiction is summarized via a nogood node. This summarization keeps the system from making the same mistake in the future. Stallman and Sussman [10] indicate that these two improvements lead to sizable gains in efficiency.

5. DISCUSSION

The TMS solves part of the problems of belief revision and limited information by introducing new conceptions of justified belief, justification, and assumption. It lends itself to several other uses as well as belief revision and making assumptions: for example, providing explanations of program beliefs and actions (see [2]), and controlling the problem solver's actions (see [1,2,10]). London [4] explores in detail the use of dependency networks in updating a problem solver world model in the face of actions. For the mathematically oriented, Drew McDermott and I try to analyze the logic and semantics underlying the TMS in [5]. *Non-monotonic logic*, as we call it, has several novel properties.

ACKNOWLEDGEMENTS

I 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, Beth Levin, Jim Stansfield, Mitchell Marcus, and Richard Brown for ideas, comments and advice, and the Fannie and John Hertz Foundation for supporting my research with a graduate fellowship.

REFERENCES

- [1] de Kleer, J., J. Doyle, G. L. Steele Jr., and G. J. Sussman, "Explicit Control of Reasoning," MIT AI Lab, Memo 427, 1977.
- [2] Doyle, J., "A Truth Maintenance System," MIT AI Lab, Memo 521, 1979.
- [3] Hayes, P. J., "The Frame Problem and Related Problems in Artificial Intelligence," in A. Elithorn and D. Jones, editors, *Artificial and Human Thinking*, San Francisco: Josey-Bass, 1973.
- [4] London, P. E., "Dependency Networks as a Representation for Modelling in General Problem Solvers," Computer Science Department TR-589, University of Maryland, 1978.
- [5] McDermott, D. and J. Doyle, "Non-Monotonic Logic I", MIT AI Lab, Memo 486, 1978, to appear in *Artificial Intelligence*.
- [6] Minsky, M., "A Framework for Representing Knowledge," MIT AI Lab, Memo 306, 1974.
- [7] Quine, W. V. and J. S. Ullian, *The Web of Belief*, second edition, New York: Random House, 1978.
- [8] Reiter, R., "On Reasoning by Default," *Proc. Second Symp. on Theoretical Issues in Natural Language Processing*, Urbana, Illinois, 1978.
- [9] Rescher, N., *Hypothetical Reasoning*, Amsterdam: North Holland 1964.
- [10] Stallman, R. M., and G. J. 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.