

A GENERAL LABELING ALGORITHM FOR ASSUMPTION-BASED TRUTH MAINTENANCE¹

Johan de Kleer

Xerox Palo Alto Research Center

3333 Coyote Hill Road, Palo Alto CA 94304

Abstract

Assumption-based truth maintenance systems have become a powerful and widely used tool in Artificial Intelligence problem solvers. The basic ATMS is restricted to accepting only horn clause justifications. Although various generalizations have been made and proposed to allow an ATMS to handle more general clauses, they have all involved the addition of complex and difficult to integrate hyperresolution rules. This paper presents an alternative approach based on negated assumptions which integrates simply and cleanly into existing ATMS algorithms and which does not require the use of a hyperresolution rule to ensure label consistency.

1 Overview

Assumption-based truth maintenance systems [2] have become a powerful and widely used tool in Artificial Intelligence problem solvers. For example, they have been used for qualitative reasoning [9], diagnosis [4], interpretation of visual scenes [12], and evidential reasoning [1], as well as being incorporated into a variety of commercially available expert system shells [11; 8; 10]. The basic ATMS [2] is restricted to accepting only horn clause justifications. Although the various generalizations have been made and proposed [3; 13] to allow an ATMS to handle more general clauses, they have all involved the addition of complex and difficult to integrate hyperresolution rules. This paper presents an alternative approach based on negated assumptions which integrates simply and cleanly into existing ATMS algorithms and which does not require the use of a hyperresolution rule to maintain label consistency.

2 ATMS background

2.1 Basic propositional definitions

This paper takes the perspective of [13] viewing the ATMS as making inferences over propositions. A *literal* is a propositional symbol (called a *positive literal*) or the negation of a propositional symbol (called a *negative literal*). A *clause* is a finite disjunction $L_1 \vee \dots \vee L_n$ of literals, with no literal repeated. A *positive clause* consists only of positive literals. A *negative clause* consists only of negative literals. A *horn clause* is a clause with at most one positive literal. Note that the horn clause,

$$\neg x_1 \vee \dots \vee \neg x_m \vee n,$$

is equivalent to the material implication:

$$x_1 \wedge \dots \wedge x_m \rightarrow n,$$

and the horn clause,

$$\neg x_1 \vee \dots \vee \neg x_k,$$

is equivalent to the material implication:

$$x_1 \wedge \dots \wedge x_k \rightarrow \perp,$$

where \perp represents false.

2.2 Basic definitions for the ATMS

An ATMS-based problem solver consists of an inference engine coupled to an ATMS. Every datum which the inference engine reasons about is assigned an ATMS *node*. The inference engine designates a subset of the nodes to be *assumptions* — nodes which are presumed to be true unless there is evidence to the contrary. The distinguished node \perp designates false. Every important derivation made by the inference engine is recorded as a *justification*:

$$x_1, \dots, x_k \Rightarrow n.$$

Here x_1, \dots, x_k are the *antecedent* nodes and n is the *consequent* node (in this paper we make the simplifying pre-supposition that consequents can not be assumptions). An ATMS *environment* is a set of assumptions.

The ATMS does propositional reasoning over the nodes. Every node is a propositional symbol, and every justification is a horn clause. An environment is a conjunction of propositional symbols. Throughout this paper C refers to the set of clauses corresponding to the justifications which have been communicated to the ATMS.

A node n is said to hold in environment E if n can be propositionally derived from the union of E with C . An environment is inconsistent (called *nogood*) if the distinguished node \perp holds in it. A nogood is *minimal* if it contains no others as a subset.

The ATMS is incremental, receiving a constant stream of additional nodes, additional assumptions, additional justifications and various queries concerning the environments in which nodes hold. To facilitate answering these queries the ATMS maintains for each node n a set of environments $\{E_1, \dots, E_k\}$ (called the *label*) having the four properties:

1. [Soundness.] n holds in each E_i .
2. [Consistency.] E_i is not nogood.
3. [Completeness.] Every environment E in which n holds is a superset of some E_i .
4. [Minimality.] No E_i is a proper subset of any other.

Given the label data structure the ATMS can efficiently answer the query whether n holds in environment E by checking whether E is a superset of some E_i . The ATMS also maintains a data-base of all minimal nogoods.

¹Corrected (as of September 30, 1988) version of the AAAI-88 paper.

3 The basic algorithm

The central task of the ATMS is to maintain node labels. The only operation which can change any node's label is the addition of a justification. When a new justification J is supplied to the ATMS **PROPAGATE**($J, \phi, \{\{\}\}$) is invoked. **PROPAGATE** takes a particular justification, an optional antecedent node a (absence is indicated by ϕ), and I a set of environments just added to the label of a . The intuition behind the algorithm is that it assumes that node labels are correct before the introduction of the new justification and therefore it only propagates the incremental changes caused by a new justification. Note that assumptions are created with labels containing the single environment containing itself and all other nodes are created with empty labels. The main work of the basic algorithm occurs in step 4 of **WEAVE**. (Viewed as propositional inference each iteration of **WEAVE** resolves a horn clause (i.e., a justification) in which h occurs negatively with another horn clause (i.e., h with its label) to eliminate h .)

ALGORITHM PROPAGATE(($x_1, \dots, x_k \Rightarrow n$), a, I)

1. [Compute the incremental label update.] $L = \mathbf{WEAVE}(a, I, \{x_1, \dots, x_k\})$. If L is empty, return.
2. [Update label and recur.] **UPDATE**(L, n).

ALGORITHM UPDATE(L, n)

1. [Detect nogoods.] If $n = \perp$, then call **NOGOOD**(E) on each $E \in L$ and return.
2. [Update n 's label ensuring minimality]
 - (a) Delete every environment from L which is a superset of some label environment of n .
 - (b) Delete every environment from the label of n which is a superset of some element of L .
 - (c) Add every remaining environment of L to the label of n .
3. For every justification J in which n as mentioned as an antecedent:
 - (a) [Propagate the incremental change to n 's label to its consequences.] Invoke **PROPAGATE**(J, n, L).
 - (b) [Remove subsumed and inconsistent environments from L .] Remove from L all environments no longer in n 's label.
 - (c) [Early termination.] If $L = \{\}$, return.

ALGORITHM WEAVE(a, I, X)

1. [Termination condition.] If X is empty, return I .
2. [Iterate over the antecedent nodes.] Let h be the first node of the list X , and R the rest.
3. [Avoid computing the full label.] If $h = a$, return **WEAVE**(ϕ, I, R).
4. [Incrementally construct the incremental label.] Let I' be the set of all environments formed by computing the union of an environment of I and an environment of h 's label.
5. [Ensure that I' is minimal and contains no known inconsistency.] Remove from I' all duplicates, nogoods, as well as any environment subsumed by any other.

6. Return **WEAVE**(a, I', R).

ALGORITHM NOGOOD(E)

1. Mark E as nogood.
2. Remove E and any superset from every node label.

This label propagation algorithm is easily shown to terminate and through careful choice of data structures can be made very efficient.

4 Hyperresolution

It is not possible to encode every possible propositional formula as a set of horn clauses. As many problem solvers wish to express arbitrary propositional formulas relating nodes and assumptions, [3] extends the basic ATMS to accept positive clauses of assumptions. Given the assumptions A_1, \dots, A_n , $choose\{A_1, \dots, A_n\}$, represents the positive clause $A_1 \vee \dots \vee A_n$. It is possible to express every propositional formula as a set of horn clauses and positive clauses of assumptions. The full ATMS accepts both justifications and chooses as input.

The specification of the full ATMS follows section 2.2 except that the initial clause set \mathcal{C} contains a positive clause corresponding to each choose. Unfortunately, given the expanded clause set, the label algorithm outlined in the previous section no longer ensures label consistency or completeness. By far the most serious problem is the loss of label consistency — namely there are nogoods logically entailed by the clauses in \mathcal{C} which the algorithm does not find. Consider the following set of chooses and justifications (upper-case letters in ATMS input always refer to assumptions):

$$choose\{A, B\},$$

$$A, C \Rightarrow \perp,$$

$$B, C \Rightarrow \perp.$$

The basic algorithm will not discover $\{C\}$ is nogood as it can not consider the choose. To discover all logically entailed nogoods, the full ATMS incorporates a complex hyperresolution rule: Given a set of sets of inconsistent assumptions α_i (i.e., ATMS nogoods), and a positive clause (i.e., an ATMS choose):

$$choose\{A_1, \dots, A_k\}$$

$$nogood\ \alpha_i\ \text{where}\ A_i \in \alpha_i\ \text{and}\ A_{j \neq i} \notin \alpha_i,\ \text{for all}\ 1 \leq i, j \leq k$$

$$nogood\ \bigcup_i [\alpha_i - \{A_i\}]$$

An instance of this rule is:

$$choose\{A, B\}$$

$$nogood\{A, C\}$$

$$nogood\{B, C\}$$

$$nogood\{C\}$$

which in propositional form is:

$$A \vee B$$

$$\neg A \vee \neg C$$

$$\neg B \vee \neg C$$

$$\neg C$$

5 Negated assumptions

An earlier paper [3] demonstrates how any clause can be encoded as a set of ATMS inputs. However, those encodings require the introduction of extraneous assumptions, needless additional justifications, and the many computational and implementational complications introduced by the use of the hyperresolution rule. We present an extended ATMS (NATMS — Negated assumption ATMS) which achieves label consistency without the use of the hyperresolution rule, integrates well with the basic algorithm, produces more complete node labels, and, when needed, allows arbitrary clauses to be encoded more efficiently and parsimoniously. The NATMS allows negated assumptions to appear directly in justification antecedents. The negation of assumption A is a *non*-assumption node and is referred to as $\neg A$.

Every choose is easily encoded as an NATMS justification. For example, the choose,

$$\text{choose}\{A, B, C\}$$

is expressed by the NATMS justification,

$$\neg A, \neg B, \neg C \Rightarrow \perp.$$

Therefore, any full ATMS problem can be trivially translated into an NATMS problem.

The specification of the NATMS follows section 2.2 extended to allow antecedents to justifications to include negations of assumptions but negated assumptions may not appear in justification consequents². The input clause set C is produced by treating justifications as material implications and translating them to clausal form. For example, the justification: $a, \neg B \Rightarrow c$ corresponds to the clause $\neg a \vee B \vee c$ (lower-case letters in ATMS input always refers to non-assumption nodes). This example illustrates the NATMS is more general than the full ATMS which can only directly represent positive clauses of assumptions (i.e., chooses).

6 The extended algorithm

The extended labeling algorithm is based on the observation that any negative clause of size k is logically equivalent to any of the k implications with one literal on its right-hand side. For example,

$$\neg A \vee \neg B \vee \neg C,$$

can be equivalently expressed in the propositional calculus as any of:

$$A \wedge B \rightarrow \neg C,$$

$$A \wedge C \rightarrow \neg B,$$

$$B \wedge C \rightarrow \neg A.$$

The basic ATMS algorithm is based on the idea that *all* label updates can be determined by propagating environments forward through justifications. Remaining with propagation as the basic technique, it is easy to see that it is necessary to encode all of these material implications

²The current implementations of all three forms of the ATMS place no restrictions on the consequents of justifications. However, greater care must be taken in the specifications and algorithms than space allows here.

as justifications. Fortunately, it is also sufficient to ensure label consistency.

Therefore the NATMS algorithm incorporates the equivalent to the following inference rule for minimal nogoods:

$$\frac{\text{nogood}\{A, A_1, \dots, A_k\}}{A_1, \dots, A_k \Rightarrow \neg A}$$

For example, given initially empty labels, on discovering the new nogood:

$$\text{nogood}\{A, B, C\},$$

the NATMS produces the following (where $\langle x, L \rangle$ indicates L is the label for node representing x):

$$\langle \neg A, \{\{B, C\}\} \rangle,$$

$$\langle \neg B, \{\{A, C\}\} \rangle,$$

$$\langle \neg C, \{\{A, B\}\} \rangle,$$

which has the same effect as having installed the following justifications:

$$A, B \Rightarrow \neg C,$$

$$A, C \Rightarrow \neg B,$$

$$B, C \Rightarrow \neg A.$$

The resolution example of section 4 is encoded as follows:

$$\neg A, \neg B \Rightarrow \perp,$$

$$A, C \Rightarrow \perp,$$

$$B, C \Rightarrow \perp.$$

Communicating these to the ATMS immediately produces the two nogoods $\{A, C\}$ and $\{B, C\}$ to which the minimal nogood rule applies 4 times producing:

$$\langle \neg A, \{\{C\}\} \rangle,$$

$$\langle \neg B, \{\{C\}\} \rangle,$$

$$\langle \neg C, \{\{A\}, \{B\}\} \rangle,$$

which when propagated to the justification,

$$\neg A, \neg B \Rightarrow \perp,$$

results in the discovery of the new nogood $\{C\}$ as desired.

Note that it is unnecessary to compute the label for $\neg A$ unless that node appears as an antecedent to some justification. Therefore, if there are no justifications referring to a negation of an assumption, the NATMS is identical to the basic ATMS. Only one step has to be added to the function **NOGOOD** to achieve label consistency.

ALGORITHM NOGOOD'(E)

- [Handle negated assumptions.] For every $A \in E$ for which $\neg A$ appears in some justification call **UPDATE**($\{E - \{A\}\}, \neg A$).

If assumption A has appeared in nogoods before $\neg A$ is used in some antecedent, then the node $\neg A$ must be created with the initial label **NOGOOD'** would have created for it had $\neg A$ appeared in some justification before any nogood was discovered.

If all the NATMS justifications represent either basic ATMS justifications (i.e., horn clauses) or chooses (i.e., positive clauses of assumptions), then the computational

complexity of this algorithm is no different than the one employing the explicit hyperresolution rule. They are essentially doing the same work; this algorithm replaces a single hyperresolution step involving a choose of size k and k nogoods with k extended label updates and one conventional label propagation. Consider the consequent of hyperresolution rule:

$$\text{nogood} \bigcup_i [\alpha_i - \{A_i\}].$$

Removing each A_i from each nogood α_i is equivalent to constructing the label for $\neg A_i$, the iterated union \bigcup_i over each nogood is equivalent to constructing the label for the consequent of the justification $\neg A_1, \dots, \neg A_k \Rightarrow \perp$, and the outer nogood assertion is equivalent to marking the computed label for \perp nogood.

It is important to note that the use of negated assumptions, as the use of chooses, needs to be used with some care as it is easy to introduce an exponential blowup in ATMS computations. The addition of any justification can result in the discovery of any number of nogoods, and every discovery of a new nogood of size k can cause k new label propagations each of which can result in the discovery of any number of nogoods. This blowup in ATMS computation is purely a consequence of the fact that a problem may have an exponential number of nogoods that need to be discovered. For many problems the expense of ensuring label consistency is too high and a more pragmatic approach is to only apply step 3 of **NOGOOD'** to nogoods below a certain size. This is discussed in [5].

7 Label completeness

Although the extended labeling algorithm ensures label soundness, consistency and minimality, it does not ensure label completeness. Consider the following example

$$\begin{aligned} A &\Rightarrow b, \\ \neg A &\Rightarrow b. \end{aligned}$$

If there are no other justifications, the NATMS will compute the label $\langle b, \{\{A\}\} \rangle$ which is incomplete as b holds universally.

For most problem-solving tasks label consistency is far more important than label completeness [3] because it is extremely important to avoid useless work on inconsistent environments. As ensuring label completeness is so computationally expensive, the ATMS computes the complete label for a node only upon request.

Foregoing label completeness renders the NATMS potentially uninteresting because the trivial algorithm which leaves all node labels empty obeys the three remaining properties of soundness, consistency and minimality. Fortunately, it is possible to clarify how close the algorithm approximates full label consistency.

Note that node labels will be correct with respect to the union of the subset of \mathcal{C} which is horn and the implicit justifications installed by the minimal nogood rule. This ensures the following useful approximation to completeness (following section 2.2):

3. [Weak completeness.] Every environment E in which n holds is a subset or superset of some E_i .

Most ATMS-based problem-solvers construct global solutions, or *interpretations*. An interpretation is a consistent environment to which no other assumption can be

added without the combination becoming nogood. It is absolutely crucial that the problem solver can determine whether a node holds or not in an interpretation. Fortunately, the following holds.

3. [Weak (interpretation) completeness.] Every interpretation I in which n holds is a superset of some E_i .

The two justifications for b at the beginning of this section can be used to illustrate how the algorithm ensures interpretation completeness. According to label completeness node b should have an empty environment in its label. Nevertheless interpretation completeness is ensured. Consider any interpretation I . If $A \in I$ then b holds trivially. If A is not in I , it can only be because it is inconsistent to add it. This means there is a minimal nogood say α consisting of A and other assumptions of I . But the minimal nogood inference rule would have assigned the consistent environment $\alpha - \{A\}$ to node $\neg A$ which propagated to b . As $\alpha - \{A\}$ is a subset of I , b holds in I . Thus b necessarily holds in every interpretation.

8 Encoding tricks

The NATMS negated technique allows it to encode negated non-assumption nodes in antecedents as well. For every node n which appears negatively in the antecedent of some justification define a new assumption A and add the following two justifications:

$$\begin{aligned} A &\Rightarrow n, \\ \neg A &\Rightarrow \neg n. \end{aligned}$$

For example, given,

$$\begin{aligned} \neg a, B &\Rightarrow c, \\ a, D &\Rightarrow \perp, \end{aligned}$$

using this encoding provides:

$$\langle c, \{\{B, D\}\} \rangle.$$

Note that this encoding has the inconvenience that the assumptions created purely for encoding purposes now appear in node's labels. These assumptions have no significance for the inference engine and should be ignored by it (see [3]). Note also if the negated nodes appear in consequents of inference-engine supplied justifications, then an additional assumption and justification set must be added to ensure total symmetry between n and $\neg n$.

In some cases a problem solver may want to force the negation of an assumption to be an assumption as well in order to have it appear in node labels. Such assumptions must be explicitly encoded as two justifications. The assumption which is the negation of A , i.e., $\sim A$ is created with the following two justifications connecting it to A :

$$\begin{aligned} A, \sim A &\Rightarrow \perp, \\ \neg A, \neg \sim A &\Rightarrow \perp. \end{aligned}$$

This has the result that there are two expressions for the negation of A : (1) $\neg A$ which is not an assumption, and (2) $\sim A$ which is an assumption. The basic difference is that the assumption $\sim A$ appears in labels while node $\neg A$ does not.

9 Backtracking

The label propagation technique presented in this paper is equivalent to installing a set of new ATMS-created justifications whenever a new nogood is discovered. This idea is similar to the dependency-directed backtracking (DDB) scheme described by Doyle [6]. The NATMS extended labeling algorithm is, in effect, performing dependency-directed backtracking in all contexts simultaneously.

For Doyle an assumption is a node supported by a non-monotonic justification. A non-monotonic justification includes an additional outlist of nodes which must all be absent from a context for the consequent to hold. The denials of the assumption are the members of the outlist. In many cases assumptions have only one denial — their own negation. A (justification-based) TMS idiom is to assume x by installing the justification:

$$\text{out } \neg x \Rightarrow x.$$

This states that x holds in a context until there is support for $\neg x$. In NATMS terms this non-monotonic justification is encoded by making x an assumption, $\neg x$ the negation of that assumption, and discarding the justification. Extremely simplified, Doyle's DDB algorithm, which operates within a single context, is as follows.

1. Find the maximal³ assumptions $S = \{A_1, \dots, A_n\}$ by tracing through the dependencies of the currently believed contradiction.
2. Select some A_i , the culprit from S . Force assumption A_i out by justifying some denial (intuitively $\neg A_i$) of the culprit with a justification whose antecedents are $\{A_1, \dots, A_{i-1}, A_{i+1}, \dots, A_n\}$.

The effect of installing the justification in step 2 is to retract one of the current assumptions therefore removing the contradiction. Doyle calls this dependency-directed backtracking because his TMS analyzes the dependency structure underlying a contradiction to determine what assumption to retract. The NATMS approach is similar to Doyle's. Step 1 of DDB is handled automatically by the ATMS as it explicitly represents the assumptions underlying every node. Step 2 of DDB is equivalent to applying the extended label rule of section 6 once.

10 Non-monotonicity

The preceding section suggests there is a close connection between non-monotonic justifications and negated assumptions. The ATMS can only represent certain kinds of non-monotonic justifications. To encode general non-monotonic justifications in the NATMS requires more substantial extensions. There have been a variety of proposals to encode non-monotonic justifications in the ATMS, but most are either faulty [3; 7] or not general [11]. In an attempt to encode non-monotonic justifications Dressler [7] has independently extended the ATMS in a very similar way to that presented in this paper by introducing 'Out-assumptions.' An out-assumption corresponds to an

³Doyle's TMS allows assumptions to be supported by justifications and hence, ultimately, on other assumptions. An assumption is non-maximal if it only supports the contradiction through other assumptions.

NATMS assumption created to represent the negation of some node. Although the use of negated assumptions approximates some of the effects of non-monotonicity, they are insufficient because there is no convenient way to represent well-foundedness. Encoding non-monotonic justifications in an ATMS is an open research topic.

11 Proofs

This paper has made many claims, without proof, that the algorithm achieves its specifications. This is the subject of a forthcoming paper which shows that the basic, full and negated ATMS algorithms ensure the specified label properties.

12 Acknowledgments

Daniel G. Bobrow, David Chapman, Ken Forbus, John Lamping, Alan Mackworth, Vijay Saraswat, Jeff Shrager, and Ramin Zabih provided useful comments on early versions of this paper.

References

- [1] D'Ambrosio, B., A hybrid approach to uncertainty, *International Journal of Approximate Reasoning*, to appear.
- [2] de Kleer, J., An assumption-based truth maintenance system, *Artificial Intelligence* **28** (1986) 127-162. Also in *Readings in NonMonotonic Reasoning*, edited by Matthew L. Ginsberg, (Morgan Kaufman, 1987), 280-297.
- [3] de Kleer, J., Extending the ATMS, *Artificial Intelligence* **28** (1986) 163-196.
- [4] de Kleer, J. and Williams, B.C., Diagnosing multiple faults, *Artificial Intelligence* **32** (1987) 97-130. Also in *Readings in NonMonotonic Reasoning*, edited by Matthew L. Ginsberg, (Morgan Kaufman, 1987), 372-388.
- [5] de Kleer, Johan, Constraint satisfaction vs. assumption-based truth maintenance, submitted for publication.
- [6] Doyle, J., A truth maintenance system, *Artificial Intelligence* **12** (1979) 231-272.
- [7] Dressler, Oskar, Extending the basic ATMS, *Proceedings European Conference on Artificial Intelligence*, 1988.
- [8] Filman, R.E., Reasoning with worlds and truth maintenance in a knowledge based system shell, *Communications of the ACM* **21** (1988) 382-401.
- [9] Forbus, K.D., The qualitative process engine, University of Illinois Technical Report UIUCDCS-R-86-1288, 1986.
- [10] Morris, P., Curing Anomalous Extensions, *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA (July 1987), 437-442.
- [11] Morris, P.H. and R.A. Nado, Representing Actions with an Assumption-Based Truth Maintenance System, *Proceedings of the National Conference on Artificial Intelligence*, Philadelphia, PA (August 1986), 13-17.
- [12] Provan, Gregory M., Efficiency analysis of multiple-context TMSs in scene representation, *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA (July 1987), 173-177.
- [13] Reiter, R. and J. de Kleer, Foundations of Assumption-Based Truth Maintenance Systems: Preliminary Report, *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA (July, 1987), 183-188.