Revised Dependency-Directed Backtracking for Default Reasoning

Charles J. Petrie, Jr.
Microelectronics and Computer Technology Corporation
3500 West Balcones Center Drive
Austin, TX 78759

Abstract

Default reasoning is a useful inference technique which involves choosing a single context in which further inferences are to be made. If this choice is incorrect, the context may need to be switched. Dependency-directed backtracking provides a method for such context switching. Doyle's algorithm for dependency-directed backtracking is revised to allow context switching to be guided by the calling inference system using domain knowledge. This new backtracking mechanism has been implemented as part of software for developing expert systems.

I. Introduction

Doyle presented an algorithm for performing **contradiction resolution** by **dependency-directed backtracking**(DDB) in a Truth Maintenance System (TMS)[5, 6].[1] Doyle's algorithm performs an *abductive* inference [16]. It takes a special state, denoted by a set of conflicting beliefs, and finds some currently disbelieved assertion, belief in which would resolve the conflict. Doyle's algorithm provides a search method for finding such an assertion and for constructing a reason for its belief. This paper derives another algorithm more suitable as a general method for revising the results of default reasoning.

We perform default reasoning when we have a set of disjoint alternatives and heuristics allow us to make a choice without doing all of the computation necessary to ensure that that choice is correct. Commonsense reasoning as well as tasks which involve incremental construction, design [9], or decision making often require default reasoning. In contrast to tasks involving parallel computation in hypothetical worlds and comparison of the results, in default reasoning a single context is preferred.

A TMS maintains a single context and switches it when a conflict is signaled by the assertion of a contradiction. When such a switch is made, contradiction resolution constructs a reason for at least one new belief which then provides an explanation for the change. For example, a circuit designer may prefer to use flip-flops with totempole output and to base his design on that choice unless it later causes a conflict. If this default choice later must be rejected in favor of an alternative, the designer can always discover why he is using tristate output flip-flops instead of his original preference by inspecting the reasons for the current belief.

This paper proposes semantics for the justifications generated by contradiction resolution and revises Doyle's algorithm to conform to them. This technical revision has important consequences for default reasoning. Doyle's algorithm restricts the set of beliefs subject to revision through a domain-independent strategy of "minimal revision", but does not provide a general method of further specifying the revision. The dependency-directed backtracking method presented here eliminates domain-independent search constraints because they are insufficient to determine correct belief revisions for a given domain and they may even eliminate that revision from consideration. Instead, a syntax is presented for representing domain knowledge that can be used by the calling inference system to reason about belief revision and to generate new alternatives as needed to resolve the contradiction.

II. Contradiction Resolution

A. Justification Criteria

In [6], a network of assertions is maintained along with reasons for their belief or disbelief. Each node in the network has associated with it a set of **justifications**.[2] A justification is composed of two sets of nodes: an **IN-list** and an **OUT-list**. Each node also has associated with it a **support status**. A justification is **valid** if each node in its IN-list has a status of IN and each node in its OUT-list is similarly OUT. (A justification with empty IN-list and OUT-list is valid and called a **premise**.) An assignment of statuses to a TMS network is **consistent** when each node is assigned a status of IN iff it has at least one valid justification, and OUT otherwise. Status assignment algorithms for a TMS network attempt to find assignments in which the network is consistent and **well-founded**: no node is in its own **believed repercussions** [22]. Alternatives to Doyle's original algorithm, which did not always terminate,

[1]This was later renamed a Reason Maintenance System [7], but a TMS was defined to mean a class of algorithms by [13] and this historic usage is continued here.

[2]The first use of TMS technical terms will be printed in **boldface**. Some defined in [6] are not redefined here.

are given in [22, 11, 19]. Doyle also gave an algorithm for resolution of contradictions using **dependency-directed backtracking** [23].

In Doyle's TMS, **contradictions** have no logical import. They denote a user designated conflicting set of beliefs represented by a valid justification for the contradiction. Doyle's algorithm **resolves** contradictions by invalidating the reason for belief in an underlying **assumption**: a belief based upon the disbelief of some other node in the database of propositions. A belief is an assumption if its **supporters** include nodes which are OUT (disbelieved).

In the TMS, the assumption selected for disbelief is known as the **culprit**. It is **retracted** by constructing a valid justification for the **elective**: the OUT supporter of the culprit which is chosen for belief in preference to belief in the culprit. We propose the following desiderata for the justification constructed for the elective:

1. The justification should be **sufficient**: it should allow a consistent and well-founded assignment of support statuses such that the contradiction is OUT.
2. The justification should be **safe**: it should not introduce an **unsatisfiable circularity** into the TMS data. (This has also been independently noted by Reinfrank [21].)
3. The justification should be **complete**: whenever the contradiction is OUT, either the elective is in any possible transitive closure of the supporters of the contradiction or the justification is invalid. (I.e., if it is possible for the contradiction to be resolved without the elective being IN, then the justification should become invalid.)

The justifications constructed by Doyle's contradiction resolution algorithm are sufficient but can be greatly simplified and are neither safe nor complete.

B. Revision of the Elective Justification

Doyle's algorithm for contradiction resolution determines a **maximal assumption set** (MAS): the largest subset of the set of all assumptions in the **foundations** of the contradiction such that no member of the subset is in the foundations of another subset member. If the MAS is A, an element A_i is chosen as the culprit, A_i has IN supporters I and OUT supporters D, then the chosen elective D_j will receive the justification with an IN-list composed of $I \bigcup \{NG\} \bigcup A - \{A_i\}$ and an OUT-list with $D - \{D_j\}$, where NG is a specially constructed **nogood** node with a **conditional-proof** (CP) justification, both of which we briefly describe below. We first note that previously published algorithms [6, 1] have failed to include the IN-list of the culprit in the justification of the elective. This minor error causes the justification to be incomplete.

The nogood node asserts that simultaneous belief in the members of the MAS is inconsistent. The justifications described so far are **support-list** justifications in contrast to the CP justification required for the nogood node. Rather than describe the CP justification, it is only necessary to note that it is a significant disadvantage of the TMS. The CP justifications are actually represented by equivalent support-list justifications requiring continued validity checks by the TMS. Not only is this emulation expensive, but also the nodes with CP justifications are IN only in a subset of cases in which they could be [6]. Other types of TMS's have been proposed that require no such justification [12, 13, 1]. In the case of [1], this means that the elective justification is incomplete. For the others, it means that the calling logic must be integrated into the TMS [17], which at the least alters the intent of the TMS. In the case of [13], nonmonotonic reasoning is not permitted. A different implementation of CP justifications has also been proposed in [11]. However, a slight alteration of the status assignment algorithm completely removes the necessity for CP justifications.

It is possible to create dependency networks in which there are at least two distinct consistent and well-founded assignments of support statuses. For instance, consider the network of nodes with single justifications: node A has a node B in its justification's OUT-list, node B has node C in its justification's OUT-list, and node C has node A in its justification's IN-list. Either nodes A and C are IN and B is OUT, or just the inverse. Either assignment of statuses is acceptable. We now require the status assignment mechanism to prefer the consistent and well-founded state in which all contradictions are OUT, if one exists. In this example, the status assignment mechanism would make A and C IN and B OUT, if node B were the contradiction and these were the only nodes and dependencies involved.

Now let each contradiction in the TMS be unique: each can have only one justification. Then the elective for a contradiction will have the same justification as described with two exceptions. The most important change in the justification is that we ***substitute the contradiction in the OUT-list for the nogood node in the IN-list*** and the latter is eliminated. This will always create an ambiguity in status assignment if the contradiction can be consistently labeled OUT. Then the status assignment mechanism will so label the contradiction. Now belief in the elective rests directly on lack of belief in the contradiction. The semantics of this are that we are assuming belief in the elective in preference to belief in the contradiction. The justification directly represents this explanation for belief in the elective.

The second change is that, to be complete, we must also include additional elements in the justification that were previously used in the CP justification. However, the work of finding these nodes can be eliminated, as well as the work of generating the MAS. We are led to this conclusion by noting that our elective justification, like Doyle's justification, is not safe.

C. Odd Loop Checking

If an element of the IN-list of the justification is in the believed repercussions of the elective, then obviously it will cause a problem. If some element of this set is a believed repercussion, then we must choose a different elective. If there are no more, we must choose a different assumption. Similarly, it is clear that no element of the OUT-list may be in the **repercussions** of the elective. Actually, the situation is worse than this. An unsatisfiable circularity may be created if any element of the justification of the elective is only in the transitive closure of its **consequences** (TCC). This is evident for the case when the elective occurs in the OUT-list of an already invalid justification of an element in the justification of the elective.

The proposed justification will not introduce an unsatisfiable circularity if it does not introduce any **odd loops** [1]. If an element of the justification of the elective is contained in the TCC of the elective and there are only "even loops" containing that element and the elective, the elective is still "safe". Determining this adds negligible computation to that involved in generating the TCC. Also, we only need to do a partial closure in that we need not consider the consequences of contradiction nodes. But the requirement for this determination leads to elimination of the creation of the MAS.

For a contradiction C, the MAS contains those assumptions which are nearest to C in its foundations and none of which are in the foundations of another element of the set. This set implements the strategy of "minimal revision": retract the assumption that causes the least change in the database. Retraction of a maximal assumption is guaranteed to cause fewer changes than retraction of a nonmaximal assumption. We must check the TCC of the elective even when we have a maximal assumption set, but this check also allows us to avoid choosing an assumption which is not maximal. Suppose that we pick some set T of assumptions, not necessarily maximal, in the foundations of the contradiction. If we pick a culprit A and an elective E, we can easily determine if A is in the foundations of any other element of T by examining the TCC of E.

Generating the MAS requires that we completely examine the foundations of the contradiction. We must also examine the transitive closure of the consequences of the elective to be safe. Thus, the foundation examination represents unnecessary work. We now present an algorithm which eliminates such work, avoids deriving the nodes corresponding to the CP justification, still permits minimal revision, and generates a safe and complete justification.

D. Revised DDB-based Contradiction Resolution

Let set S be the supporters of contradiction C. (Assume for now that contradiction nodes have empty OUT-lists.) The contradiction is resolved iff function MAKEOUT(S,nil,{C}) returns a justification for an elective. If so the justification will satisfy the three properties of section II.A.

Define **MAKEOUT** (A, IJ, OJ):

1. If A is null, return "false".
2. Pick $A_i \in$ A. Construct the TCC of A_i and save it. If some element of A - A_i is in the repercussions of A_i, then go to step 4. Otherwise, let J_i be the **supporting justification** of A_i. If MAKEIN(OUT-list of J_i, IJ $\bigcup$ IN-list of J_i $\bigcup$ A - $\{A_i\}$, OJ), then return it.
3. If MAKEOUT(A - $\{A_i\}$, IJ $\bigcup$ $\{A_i\}$, OJ) then return it.
4. Return MAKEOUT(IN-list of J_i, IJ $\bigcup$ A - $\{A_i\}$, OJ $\bigcup$ OUT-list of J_i).

Define **MAKEIN** (A, IJ, OJ):

1. If A is null, return 'false'.
2. Pick $A_i \in$ A. If A_i is a contradiction, return MAKEIN(A-$\{A_i\}$, IJ, OJ $\bigcup$ $\{A_i\}$).
3. Construct justification J_e for A_i with an IN-list consisting of IJ, and an OUT-list consisting of OJ together with A - $\{A_i\}$. Construct the TCC of A_i making use of the TCCs of elements of IJ already constructed so far. If the new justification will not create any odd loops, return it and the elective.
4. Return MAKEIN(A-$\{A_i\}$, IJ, OJ $\bigcup$ $\{A_i\}$).

In this algorithm, we start by trying to make OUT some IN supporter of the contradiction using MAKEOUT. To make a node OUT with this function, we first try to make some element of the OUT-list of the supporting justification IN using MAKEIN. If that fails, we first try to make another element of the first argument of MAKEOUT OUT. As a last resort, we try to make some element of the IN-list of the supporting justification OUT using MAKEOUT recursively. A contradiction node cannot be made IN. Any other node can be made IN by giving it the constructed justification.

Sufficiency is acheived by an OUT-list-first search followed by a depth-first seach of the contradiction's foundations. Safety is guaranteed by step 3 of MAKEIN. The actual check for odd loops by examination of the TCC can be implemented cheaply. Completeness is ensured by the additions to IJ and OJ in recursive calls to MAKEOUT and MAKEIN. The algorithm also collects other relevant contradictions to be placed in the OUT-list of the justification of the elective.

III. Knowledge-Based Search

Because of the elimination of the MAS in the above algorithm, the search space can be extended to include the entire foundations of the contradiction rather than being restricted to the maximal assumptions. In an experimental expert application development system called "Proteus" [20][3], the above algorithm has been extended in the following ways.

[3]There are two commercial design applications [25, 24] using Proteus. Other design applications are experimental and internal to MCC.

A. Extended Backtracking

The elective may already have at least one justification, although it has no valid one. Adding a new justification may implicitly violate domain rules. For instance, the reason a particular elective is OUT is because some exception is IN. Simply adding another justification, as dictated by the strategy of minimal revision, ignores this exception. In the current implementation, a proof attempt is first made on each candidate elective. If it is not successful, then either domain knowledge or the user must indicate that it is permissible to construct a new justification for the elective. If this is not the case, an attempt is made to make one of the existing justifications, if any, valid. This is accomplished by recursively calling MAKEOUT on the elements of the OUT-list and MAKEIN on the elements of the IN-list. This may result in justifications being constructed for more than a single elective. Contradictions resolved earlier are prevented from coming IN again by recursively calling MAKEOUT on those found in the TCC of the elective. If the current contradiction cannot be resolved, or cannot be resolved without bringing IN some previously resolved contradiction, it is left unresolved and marked as such. An extended algorithm for this is given in [18].

B. Depth-First Guidance

The extended algorithm is essentially a depth-first search biased toward the OUT-lists. It is easily modified to use domain knowledge to guide the search for electives. In step two of the algorithm for MAKEOUT, some element of the set of candidate culprits, which is the first argument to MAKEOUT, is chosen. An attempt will be made to find a reason to disbelieve the chosen culprit. It would be a less than optimum choice to select a candidate which is more strongly believed than some other candidate. It is desirable, then, to provide a general mechanism for representing domain knowledge about the relative rankings of beliefs in the various assertions under different circumstances and be able to use the same rule system to reason about this knowledge as is used in the rest of the application.

A two argument predicate *PREFER* has been implemented which states that belief in the assertion of its first argument is preferred to belief in that of the second for the purposes of belief revision. When choosing a candidate culprit from the MAKEOUT set, an attempt is made to prove that this culprit is preferred to some other element of the set. If a less preferred element is found, then it becomes the candidate culprit and the process recurses. PREFER thus enforces a partial ordering on the candidate culprits such that none of them will be chosen before others which are more suspect. Similarly, in MAKEIN, PREFER is used to ensure that no candidate elective is chosen before some other for which belief is preferred.

Assertions using the PREFER predicate, called preferences, can be based on object types and can be concluded by Proteus rules with variables instantiated at run time. Since these rules may have antecedents which are satisfied only in certain contexts, the selection of culprits and electives can be controlled dynamically. The preferences may be based on numbers, lists, or arbitrary domain reasoning. We also provide for interactive control of the selection of culprits and electives if desired. The ability to add this domain knowledge overcomes the disadvantage of "blind" dependency-directed backtracking.[14]

PREFER is also used to define a terminal node in the search. As described above, a proof attempt is made on each candidate elective. If such a proof is not possible using rules, but the application allows the truth of that particular elective to be asked of the user, one of three possible answers is allowed: "yes", "no", and "maybe". The first causes the elective to be provided with a premise justification. The second disqualifies the elective from further consideration. The third gives permission for a new justification to be constructed, thus defining a leaf of the network search. Instead of such a user query, PREFER can be used for the same purpose. If CONTRADICTION is the second argument of a preference, then the first argument is eligible to receive a constructed justification: the semantics are that belief in the elective may be assumed in preference to that in the contradiction. If CONTRADICTION is the first argument, the inverse is true and the candidate also defines a terminal but unsuccessful node in the network. In no case will Proteus arbitrarily justify an elective in order to resolve a contradiction.

C. Finding New Alternatives to Defaults

The capability for dynamic generation of elective candidates is provided with a predicate, *DEFEAT*, which takes three arguments: the candidate culprit, some element of the IN-list of a justification of the culprit, and some new elective. If a DEFEAT assertion can be proven, the elective may be added to the OUT-list of any justification of the culprit identified by the IN-list element. The semantics of DEFEAT are that some reason for believing the culprit can be defeated by belief in some new exception or alternative. In step two of the algorithm for MAKEOUT, if no member of the existing OUT-list can be made IN, then an attempt is made to prove a DEFEAT for the culprit to add to the OUT-list. DEFEATs are general in that they can be concluded by rules, like preferences. Thus, they can be used to generate arbitrary new alternatives to belief in the culprit. Proteus also allows for the interactive acquisition of new alternatives and even DEFEATS.

D. Focused Search

For a given domain and situation, the culprit and elective that should be considered first may lie several levels deep in the dependency-structure which supports the contradiction. A predicate FIX has been implemented which takes three arguments: the first is a candidate culprit, the second a possible ancestor, and the third a possible elective for that ancestor. The semantics of a FIX are that if the second argument is an ancestor of the first, for the current dependency network, and if the third can be an elective of the second (is in the OUT-list of a supporting justification or can be placed there by a DEFEAT), then an attempt should be made to believe the elective (and a call to MAKEIN is made on it.) Such FIXes are used to focus first on nodes deep within the foundations of a contradiction prior to performing the depth-first search described above. Fixes are most useful when the first argument unifies with a supporter of the contradiction, but may also be used during search. The choice between which of several FIXes to pursue first is determined by the preference order on their various electives. FIXes may also be concluded by rules.

E. Example

A doctor might conclude that patient Jane is dehydrated from observation of the appropriate symptoms. That leads to a conclusion that Jane has a low amount of water which in turn leads to a conclusion that Jane should have a high sodium concentration. However, the lab results indicate that Jane actually has a low sodium concentration. There were at least three default assumptions made in this example, retraction of which would resolve the contradiction. The doctor assumed that the symptoms were those of dehyration and not some other disease. He assumed that Jane's sodium level was normal. And there is an assumption that the lab test is correct.

In a typical TMS dependency network produced by this logic, a depth-first search would explore first a possible lab test error, then an abnormal sodium level, and finally a recheck of the symptoms. This does not correspond to actual practice. Typically, the doctor first rechecks the symptoms because it is easy to do. Then (or perhaps at the same time), he may ask for the lab work to be redone. He will be particularly suspicious of the lab work if a high glucose level is indicated because this interferes with normal calculations of sodium levels. If these simple "fixes" to the problem aren't sufficient to resolve it, he carefully reconsiders his thinking. The obvious alternative to a normal sodium level is either directly proven or assumed and the doctor is led to consideration of a cause.

In Proteus, this problem resolution could be represented by two FIXes, one conditional preference between them, and a DEFEAT. One FIX would say that if the contradiction is supported by a conclusion which is somehow supported by an observation of symptoms, then suspect a mistaken observation. The other FIX would simply say that any lab test supporting a contradiction should be redone and checked for error. The preference would be to check out the mistaken observation prior to redoing the lab test, unless the patient were unusually difficult to observe or it was known that the patient's glucose level was high and the lab test under consideration was for sodium concentration. The DEFEAT would defeat the conclusion of a high sodium concentration by generating the possibility of an abnormal sodium level. This would occur in the depth-first search if neither FIX were successful.

IV. Comparisons

McAllester's RUP [13] provides a general purpose method of premise control using "likelihood classes". However, this approach imposes a global ordering on the database which is more complete than is actually justified. Proteus uses the PREFER predicate to avoid this problem. Unlike Cohen [2], no predefined categories of preferences are defined. In DEBACLE, Forbus [10] also objects to likelihood classes and instead defines "closed-world assumptions" which are checked for invalidity before trying special purpose routines ordered in a stack.

In an extension to the ATMS [4], a simple list is used to order the selection of culprit candidates. In WATSON [15] , the calling inference system is allowed to attempt to prove that candidate culprits may be ordered by relevance to the story being parsed. In PLANET [3], special objects called "decision choices" are created which contain information about alternatives and their effects on constrained resources. Contradictions are caused by resource constraint violations and resolved by selecting alternatives which are not disadvantageous to the resource in question.

Proteus provides a more domain-independent and dynamic mechanism for controlling backtracking than these systems. Preferences may be concluded on an arbitrary basis and alternatives need not be enumerated prior to the occurance of the contradiction. Rather than select a set of predefined objects from the foundations of the contradiction, FIXes allow any assertion in the database to belong in the initial focus of candidate culprits and electives. If these are not successful, exhaustive search is used. Unlike previous algorithms, the extended search described in section III may result in more than one elective being justified.

V. Summary

Doyle's original algorithm for contradiction resolution is revised to conform to proposed semantics. A new algorithm for dependency-directed backtracking is derived which allows any node in the dependency network to be considered for inclusion in a set of assumptions to be retracted during contradiction resolution. This allows domain-independent search constraints to be rejected in favor of a general backtracking control method dependent on domain knowledge. Special predicates are defined which allow the application builder to represent domain knowledge about how to switch contexts when beliefs conflict. The selection of the new context can be reasoned on the basis of the current state of the database and new alternatives generated dynamically. This has been implemented as a general method for revising the results of default reasoning in expert systems.

References

[1] Charniak E., Riesbeck C., and McDermott D., "Data Dependencies," *Artificial Intelligence Programming*, Chap. 16, L. E. Erlbaum, Baltimore, 1979.

[2] Cohen, P. R., *Heuristic Reasoning about Uncertainty: An Artificial Intelligence Approach*, Pitman Publishing, Marshfield, MA, 1985.

[3] Dhar V. and Quayle C., "An Approach to Dependency Directed Backtracking using Domain Specific Knowledge," *Proc. IJCAI-85*, pp. 188-190, 1985.

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

[5] Doyle J., "Truth Maintenance Systems for Problem Solving," Technical Report AI-TR-419, Massachusetts Institute of Technology, AI Lab., 1978.

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

[7] Doyle J., "A Model for Deliberation, Action, and Introspection," AI-TR-581, Massachusetts Institute of Technology, AI Lab., 1980.

[8] Doyle J., "Some Theories of Reasoned Assumptions," CMU CS-83-125, Carnegie-Mellon University, Dept. of Comp. Sci., 1983.

[9] Feldman, Y. A., and Rich, C., "Reasoning With Simplifying Assumptions: A Methodology and Example," *Proc. of the Fifth National Conference on Artificial Intelligence*, AAAI, pp. 2-7, 1986.

[10] Forbus, K. D., "Qualitative Process Theory," Appendix, AI-TR-789, Massachusetts Institute of Technology, AI Lab., 1984.

[11] Goodwin, James W., "An Improved Algorithm for Non-monotonic Dependency Net Update," LiTH-MAT-R-82-23, Linkoping Institute of Technology, Sweden.

[12] Martins J., "Reasoning in Multiple Belief Spaces," TR-203, State University of New York at Buffalo, 1983.

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

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

[15] Orejel-Opisso, J. L., "Story Understanding with WATSON: A Computer Program Modeling Natural Language Inferences Using Nonmonotonic Dependencies," Masters Thesis, University of Illinois at Urbana-Champaign, 1984.

[16] Pierce, C. S., **Scientific Metaphysics**, Vol. VI, pp. 358.

[17] Petrie, C., "Using Explicit Contradictions to Provide Explanations in a TMS," Microelectronics and Computer Technology Corporation Technical Report MCC/AI/TR-0100-05, 1985.

[18] Petrie, C., "Extended Contradiction Resolution," Technical Report, Microelectronics and Computer Technology Corporation MCC TR AI-102-86, 1986.

[19] Petrie, C., "A Diffusing Computation for Truth Maintenance," *Proc. of the IEEE International Conf. on Parallel Processing*, August 1986, pp. 691-695.

[20] Petrie, C., Russinoff, R., and Steiner, D., "Proteus: A Default Reasoning Perspective," *Proc. 5th Generation Conf.*, Nat. Inst. for Software, October, 1986.

[21] Reinfrank, M., et al., "KAPRI - A Rule-Based Non-Monotonic Inference Engine with an Integrated Reason Maintenance System," Research Report Draft, University of Kaiserslautern, January 1986.

[22] Russinoff, D., "An Algorithm for Truth Maintenance," Microelectronics and Computer Technology Corporation Technical Report AI/TR-062-85, 1985.

[23] Stallman ,R. and Sussman, G., "Forward Reasoning and Dependency-Directed Backtracking," Memo 380, Massachusetts Institute of Technology, AI Lab., Sept. 1976.

[24] Steele, R., "An Expert System Application in Semicustom VLSI Design," *Proc. 24th IEEE/ACM Design Automation Conference*, Miami, 1987.

[25] Virdhagriswaran, S., et al., "PLEX: A Knowledge Based Placement Program for Printed Wire Boards," *Proc. 3rd IEEE AI Applications Conf.*, February, 1987.