New Algorithms for Dependency-Directed Backtracking

Charles J. Petrie, Jr.

AI TR86-33 September 1986
New Algorithms for Dependency-Directed Backtracking

by

Charles J. Petrie, Jr., B.S.

THESIS
Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCES

THE UNIVERSITY OF TEXAS AT AUSTIN
August, 1986
Acknowledgments

I thank Jon Doyle, Elaine Rich, and Drew McDermott for their helpful discussions; Johan De Kleer for his criticisms and extensive review of an earlier version; David Russinoff for the suggestion about elimination of odd loop checking; Mohamed Gouda for his helpful review; Donald Steiner for his careful editing and many suggestions; and Bob Simmons for both his technical assistance and constant encouragement. This work was funded by Microelectronics and Computer Technology Corporation [24,25].
Abstract

The problem of providing dependency-directed backtracking in a Doyle-style Truth Maintenance System (TMS) is solved with three algorithms. The first modifies Doyle's basic algorithm by eliminating the necessity for conditional proof justifications. Unlike other attempts to do so, this method maintains the capability of the TMS to provide complete explanations from justifications. The second algorithm, which eliminates the generation of a maximal assumption set, provides a novel description of backtracking as a simple search. Finally, an extension of this algorithm allows contradiction resolution to be extended as an inference technique for expert systems. The dependency-directed backtracking algorithms presented are also the first to correctly ensure that an assertion is not justified unnecessarily and that unsatisfiable circularities are not created.
# Table of Contents

Acknowledgments iv

Abstract v

Table of Contents vi

List of Figures viii

1. Background 1
   1.1 Introduction 1
   1.2 Doyle-style TMS Contradiction Resolution 3
   1.3 The Charniak, et al. Fix 8
   1.4 Generating Explanations 11
   1.5 Multiple Belief Spaces 15
   1.6 Three Valued Logic 16

2. A Solution 18
   2.1 Elimination of Conditional Proof Justifications 18
   2.2 Odd Loop Checking 22
   2.3 The Contradiction Resolution Process 24

3. Reformulating Dependency-Directed Backtracking 26
   3.1 Characterization of DDB 26
   3.2 Maximal Assumption Set 27
3.3 "Lazy" DDB-based Contradiction Resolution .... 31

4. Extended Contradiction Resolution ...... 33
   4.1 Assumptions, Defaults, and Alternatives .... 33
   4.2 The Extended Algorithm .................. 35
   4.3 Extension Differences .................... 40
   4.4 An Example of Extended Resolution ........ 41

5. Conclusion .................. 45
   5.1 Future Work .......................... 45
   5.2 Summary ............................ 47

BIBLIOGRAPHY .......... 48

Vita
List of Figures

3.1 A Potential Odd Loop ........................................ 28
3.2 A Non-maximal Assumption ................................. 29
4.1 Default Circuit Design ....................................... 43
Chapter 1

Background

1.1 Introduction

Doyle's Truth Maintenance System [7,8], later named the Reason Maintenance System (RMS) [9], was the first implementation of what has become a general artificial intelligence technique, now generically known as a TMS. One of the major advantages of a TMS [28,29,30] is the capability to provide explanations for belief or disbelief in assertions through the means of attached justifications. Dependency-directed backtracking (DDB) contradiction resolution is another important capability of a TMS in nonmonotonic logic applications. Doyle's contradiction resolution process justifies assertions in a way that preserves the justification-based explanation capability of the TMS. In particular, the contradiction itself, and thus all of the beliefs that contributed to it, are traceable through the justifications constructed in order to resolve the contradiction. However, this involves a special "conditional proof" justification which Doyle [8] has noted is undesirable in some respects.

At least in part due to the difficulties associated with this kind of justification, subsequent truth maintenance systems provided different contradiction resolution mechanisms. Charniak [1] eliminated any direct reference to contradiction nodes in new justifications. Other systems [15,16,17,18,19,20]
gave up the independence of the TMS from the calling logic. Explicit contradiction nodes were completely eliminated in these systems. Such systems do not provide in the TMS justifications a complete record of the reasons for believing assertions justified by the contradiction resolution process.

At least two well-known expert system tools, DUCK [21,22] and MRS [13], which depend upon an independent TMS for explanations have simply omitted the contradiction resolution process largely because of the lack of an adequate way to handle contradictions. Here we present a simple method of eliminating the requirement for conditional proof justifications while providing justifications with direct references to explicit contradiction nodes. With such justifications, it is easy to generate an adequate explanation for assertions justified in order to resolve contradictions.

De Kleer [5,6] has identified a class of problems for which DDB is unnecessary. However, it remains an important problem solving technique for other problems, particularly those that are best solved by making a single choice whenever possible. In addition, DDB still seems to offer a better mechanism for providing explanations. Doyle [9,10,11] has criticized the "blind" DDB of RMS and has argued that more sophisticated techniques are required for reasoned retraction of assumptions. This presumes that DDB is a fully developed technique. We show that it can be improved, including the correction of two previously unnoticed problems with the DDB of [8] and [1].
1.2 Doyle-style TMS Contradiction Resolution

A wide variety of TMS mechanisms have been developed. Without going through the entire history as has been done better elsewhere [12,15], Doyle seems to have produced the first TMS with well defined mechanisms independent of any application for dealing with circularities and contradiction resolution (although [14] was published at the same time, London deals more with the application of dependency structures to general problem solving and explicitly renounces any claim to advancing the methodology of representing alternatives presented by Doyle's TMS). Doyle's TMS is the TMS most often referred to in the literature and deals extensively with explanations, unlike most other systems. This paper proposes a correction to one of the flaws of this system that motivated subsequent systems. To explain the problem and proposed solutions, it is necessary to take significant space to review Doyle's TMS and others.

We will illustrate Doyle's basic contradiction mechanism with the well known scheduling example. Some familiarity with the TMS is assumed. The first occurrence of technical terms will be placed in bold and the reader is directed to [8] for their precise meaning. The example is to schedule meeting M either in room 813 or in room 801 either at 10:00 or at some other time. That the meeting can be at 10:00 in room 813 is assumed as the initial situation. The TMS representation is:
<table>
<thead>
<tr>
<th>Node</th>
<th>Content</th>
<th>Justification</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-1</td>
<td>TIME(M) = 10:00</td>
<td>(SL () (N-2))</td>
<td>IN</td>
</tr>
<tr>
<td>N-2</td>
<td>TIME(M) ≠ 10:00</td>
<td></td>
<td>OUT</td>
</tr>
<tr>
<td>N-3</td>
<td>ROOM(M) = 813</td>
<td>(SL () (N-4))</td>
<td>IN</td>
</tr>
<tr>
<td>N-4</td>
<td>ROOM(M) = 801</td>
<td></td>
<td>OUT</td>
</tr>
</tbody>
</table>

Example 1

In a TMS, a node with assertional content is IN iff it has at least one valid justification, and OUT otherwise. The justifications above are all of the support list (SL) type consisting of an INlist and an OUTlist. Such a justification is valid iff the nodes in the INlist are all IN and the nodes in the OUTlist are all OUT. At the beginning, the TMS makes N-1 and N-3 IN and the other two nodes OUT (A node is OUT if it has no justification). If the external system supplies to the TMS the information that N-1 and N-3 conflict, then the TMS creates a special contradiction node, N-5, which represents this information and initiates dependency-directed backtracking which records its findings in a special nogood node, N-6, and selects a solution to the conflict. By tracing the foundations of the contradiction node, N-1 and N-3 are selected as the maximal assumption set since they are assumptions (the OUTlists of their supporting justifications are non-empty) and neither is in the foundations of any other assumption in the foundations of the contradiction. At this point, N-3 is arbitrarily selected to be the culprit
made OUT. This is done by justifying a node, called the \textit{elective} (new terminology) on N-3's OUTlist; the only one being N-4. The result is:

\begin{verbatim}
N-4 ROOM(M) = 801 (SL (N-6 N-1 ())) IN
N-5 CONTRADICTION (SL (N-1 N-3 ())) OUT
N-6 NOGOOD N-1 N-3 (CP N-5 (N-1 N-3 ())) IN
\end{verbatim}

Example 1a

Now we introduce a new justification. A \textit{conditional proof} (CP) justification is valid if when the nodes in the \textit{INhypotheses} are IN and the nodes in the \textit{OUThypotheses} are OUT, the \textit{consequent} node must be IN. In the conditional proof justification for Node 6, N-5 is the consequent node and the \textit{INhypotheses} list contains N-1 and N-3; the \textit{OUThypotheses} list is empty. Node N-6 must be IN because the CP justification of that node is valid: node N-5 (the contradiction) will be IN when N-1 and N-3 are. In fact, since N-5 depends on N-1 and N-3, the justification for N-6 is always valid. In general, for a given contradiction for which $A$ is the maximal assumption set, if an element $A_i$ has been selected as the culprit with \textit{OUTlist} $D_i$ and element $D_j$ has been selected as the elective, then $D_j$ is justified with an \textit{INlist} containing the set of nodes $\{NG\} \cup A - \{A_i\}$ and an \textit{OUTlist} containing the set of nodes $D - \{D_j\}$, where $NG$ is the nogood node associated with the contradiction.

An important feature of this justification is that if later changes to the database enable the contradiction to be resolved regardless of the
validity of the elective justification, then it should be invalid. In example 1, the conflict is between N-1 and N-3. The sole justification for N-4 being IN is to resolve that conflict. Should it be resolved by another means, then there is no longer a good reason to believe N-4. If N-1 were to go OUT, this would resolve the conflict and we would want N-4 to go OUT as well. The presence of N-1 in the justification for N-4 ensures that this will happen. If there had been any other nodes in the OUTlist of N-3, they would have appeared in the OUTlist of N-4 as well.

The role of the nogood node in the elective justification is important. The nogood node is supported by a CP justification. Since the CP justification asserts an implication, it may remain valid even if its consequent is OUT. Thus the nogood node remains IN in this example. The CP justification is in turn implemented with an SL justification which includes those nodes necessary to support the validity of the CP justification. The CP justification thus provides a path back to all of the assumptions underlying the contradiction. This helps to provide a complete explanation for the elective and allows the other assumptions to be retracted if necessary.

A continuation of the scheduling example illustrates the further retraction of assumptions. Suppose that it is learned that room 801 is not available at all. When informed of this, the TMS creates a new contradiction node, N-7, with a support list justification with N-4 in its INlist, and a new nogood node, N-8, with a CP justification with N-7 as the consequent and N-1 in the INhypotheses because it was the maximal assumption underlying N-4. This CP is implemented with a SL consisting of N-6 in its INlist because that node must be IN in order for N-1 to imply N-7.
There is only one culprit to select to resolve this contradiction: N-1 must be made OUT. This is, of course, done by making N-2 IN. The justification of N-2 contains only the nogood node, N-8. That is, the contradiction involved only one node and there is no alternative to making N-2 IN to resolve the contradiction. Now, the final result looks like:

<table>
<thead>
<tr>
<th>Node</th>
<th>Content</th>
<th>Justification</th>
<th>Belief</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-1</td>
<td>TIME(M) = 10:00</td>
<td>(SL ( ) (N-2))</td>
<td>OUT</td>
</tr>
<tr>
<td>N-2</td>
<td>TIME(M) ≠10:00</td>
<td>(SL (N-8))</td>
<td>IN</td>
</tr>
<tr>
<td>N-3</td>
<td>ROOM(M) = 813</td>
<td>(SL ( ) (N-4))</td>
<td>IN</td>
</tr>
<tr>
<td>N-4</td>
<td>ROOM(M) = 801</td>
<td>(SL (N-6 N-1) ( ))</td>
<td>OUT</td>
</tr>
<tr>
<td>N-5</td>
<td>CONTRADICTION</td>
<td>(SL (N-1 N-3) ( ))</td>
<td>OUT</td>
</tr>
<tr>
<td>N-6</td>
<td>NOGOOD N-1 N-3</td>
<td>(CP N-5 (N-1 N-3) ( ))</td>
<td>IN</td>
</tr>
<tr>
<td>N-7</td>
<td>CONTRADICTION</td>
<td>(SL (N-4) ( ))</td>
<td>OUT</td>
</tr>
<tr>
<td>N-8</td>
<td>NOGOOD N-1</td>
<td>(CP N-7 (N-1) ( ))</td>
<td>IN</td>
</tr>
</tbody>
</table>

Example 1b

The nogood node provides a pointer back to all of the nodes that contributed to the justification of the current node. This is best seen in the case when the maximal assumption set is a singleton and that node has only a single element in its OUTlist. The only node available to be placed in the justification of the elective is the nogood node.
One alternative to using the nogood node is that of using a premise justification. But that would provide an insufficient explanation for the belief of the assertion represented by the node. We would lose the trace back to the contradiction and its causes. For instance, it would be impossible to determine that the reason for justifying the elective was to retract an assumption that caused a contradiction.

However, the CP justification is one of the major disadvantages of the TMS. The CP justifications are actually represented by equivalent SL justifications requiring continued validity checks by the TMS. Not only is this emulation expensive, the nodes with CP justifications are IN only in a subset of cases when they could be [8]. Other types of TMS's have been proposed that require no such justification.

1.3 The Charniak, et al. Fix

The first attempt to eliminate the conditional proof justification from a Doyle-style TMS appears to be by Charniak, Riesbeck, and McDermott in [1]. Although such an attempt is not explicitly stated, the text, code, and McDermott [23] suggest that the algorithms used for contradiction resolution are those proposed by Doyle with the modification of the elective justification to eliminate the requirement for conditional proofs. Doyle [8] also refers to [1] as a simplification of TMS. In the Charniak system, rules with premise justifications take the place of the nogood node for the rectifying justification. The example that Charniak gives illustrates this. We have liberally translated it here into a more standard TMS form rather than the graph format originally used. However, the support list notation will be
dropped as it is now the only type of justification. The example is a medical one: Fred has a pain in his side. We suspect appendicitis but it might be other diseases such as indigestion or colitis. The assumption of appendicitis is contradicted by the fact that Fred says that he has no appendix and we have no reason to believe that Fred is unreliable:

<table>
<thead>
<tr>
<th>Node</th>
<th>Content</th>
<th>Justification</th>
<th>Belief</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-1</td>
<td>(Has Fred Appendicitis)</td>
<td>(N-2)</td>
<td>(N-3 N-4) IN</td>
</tr>
<tr>
<td>N-2</td>
<td>(Pain Fred Side)</td>
<td>( )</td>
<td>( ) IN</td>
</tr>
<tr>
<td>N-3</td>
<td>(Has Fred Indigestion)</td>
<td></td>
<td>OUT</td>
</tr>
<tr>
<td>N-4</td>
<td>(Has Fred Colitis)</td>
<td></td>
<td>OUT</td>
</tr>
<tr>
<td>N-5</td>
<td>(Not (Has-a Fred Appendix))</td>
<td>(N-6)</td>
<td>(N-7) IN</td>
</tr>
<tr>
<td>N-6</td>
<td>(Says Fred</td>
<td></td>
<td>IN</td>
</tr>
<tr>
<td></td>
<td>(Not (Has-a Fred Appendix))</td>
<td>( )</td>
<td>( )</td>
</tr>
<tr>
<td>N-7</td>
<td>(Unreliable Fred)</td>
<td></td>
<td>OUT</td>
</tr>
<tr>
<td>N-8</td>
<td>(AND (Has ?X Appendicitis)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Not (Has-a ?X Appendix))</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>→ (Contradiction)</td>
<td>( )</td>
<td>( ) IN</td>
</tr>
<tr>
<td>***</td>
<td>(Contradiction)</td>
<td>(N-1 N-5 N-8)</td>
<td>( ) IN</td>
</tr>
</tbody>
</table>

Example 2

In the Charniak code, contradictions are stored within a special variable, so it is not represented by a node as above. (Charniak has “molecules”
rather than nodes.) In this example, the rule at Node 8 with a premise justification, allows the calling system to infer a contradiction from the assertions that are currently IN. This contradiction must be made OUT to be resolved. A maximal assumption set is created and a culprit selected, as in Doyle. An element of the OUTlist of the culprit is selected to be justified. It is given the same justification as in the Doyle system except that no nogood node is included in the INlist. Instead, Charniak uses the rule, N-8, that allowed the contradiction to be inferred. (Charniak's simplification is actually incorrect since it does not take into account all of the assumptions that Doyle's CP justifications do. Similarly, neither Doyle nor Charniak provide for inclusion of the INlist of the culprit, in this case N-2. This is covered in Chapter 2.)

Now suppose that the culprit selected is N-1 and the default assertion justified is N-3. Then N-3 would receive the justification:

\[
\begin{align*}
N-3 & \quad \text{(Has Fred Indigestion)} & (N-5 & \quad N-8) & (N-4) & \quad \text{IN} \\
\end{align*}
\]

This method depends on using rules to generate contradictions in order to provide explanations. To see this, suppose that we handle the negation of Fred's indigestion as Doyle handled the negation of the assumption that the meeting could be held in room 801. In that case, instead of adding a node asserting that it is not the case that Fred has indigestion, we would simply have a contradiction with only N-3 in its INlist:

\[
\begin{align*}
*** & \quad \text{(Contradiction)} & (N-3) & \quad (\_\_\_) & \quad \text{IN} \\
\end{align*}
\]
When trying to resolve this contradiction, we find that N-3 is an assumption with exactly one element, N-4, in its OUTlist. Now we try to justify N-4. But there are no candidates to put in the justification. Charniak avoids using only a premise justification by always using a rule to derive a contradiction in his examples. Then the rule is available to provide some explanation. However, this is still inadequate, as discussed in the next section.

1.4 Generating Explanations

Justifications can be used directly to provide simple explanations for the belief or disbelief in a given assertion. Typically, as in DUCK or MRS, a “WHY” command for a believed assertion prints the supporting justification. If the assertion has been derived, then the justification gives the first branches in the proof tree for that assertion. The listing can give both the INlist and the OUTlist. The user can then ask “WHY” of any of the assertions on the INlist and receive a similar listing. If he asks “WHY”, or more appropriately, “WHY NOT” of an assertion on the OUTlist, a list of supporting nodes arranged in INlist and OUTlist format can be given. Thus it is possible to fully trace through the reasons for the status of the current assertion by looking at justifications. This trace terminates at premise justifications or when there is no justification at all.

Doyle does not explicitly discuss explanation generation. A possible problem with the Doyle system in providing explanations is the indirection of CP justifications. Given the query “(WHY N-2)” what sort of response should the user get? One simple possibility is:

(WHY N-2)
Because

Contradiction (N-7) if

\[ \text{TIME(M) = 10:00} \text{ (N-1) when NOGOOD N-1 N-3 (N-6)} \]

This hypothetical explanation mechanism went through the foundations of N-2 and found nogood node N-8 supported by a CP justification implemented by nogood node N-6. Eventually, it can be discovered that the proper explanation for N-2 being IN is because of two contradictions, N-5 and N-7. In other words, we would like the justification for the assertion that the meeting is not at 10:00 to reflect first that the meeting couldn’t be in room 813 at 10:00 and second that the only alternative to room 813, room 801, turned out to not be available. It requires tracing through a few levels of justifications, even in this simple example, to find this information. But it is possible to eventually trace back to all of the relevant assertions, including the contradiction nodes. This is not the case for the Charniak system.

To see that the Charniak system of contradiction resolution provides inadequate explanations, let us contradict N-3 as discussed in example 2:

\[
\begin{align*}
\text{N-3} & \quad \text{(Has Fred Indigestion)} \quad \text{(N-5 N-8)} \quad \text{(N-4)} \quad \text{OUT} \\
\text{N-4} & \quad \text{(Has Fred Colitis)} \quad \text{(N-9 N-10)} \quad \text{( )} \quad \text{IN} \\
\text{N-9} & \quad \text{(Not (Has Fred Indigestion))} \quad \text{( )} \quad \text{( )} \quad \text{IN} \\
\text{N-10} & \quad \text{(AND (?X) (Not (?x)))} \\
& \quad \rightarrow \text{(Contradiction)} \quad \text{( )} \quad \text{( )} \quad \text{IN}
\end{align*}
\]
Example 2a

In this example, if we ask why Fred now has colitis, then the immediate answer is that Fred does not have indigestion, along with the fact that a contradiction arises when both an assertion and its negation are IN. This is not a very good explanation for why Fred should have colitis. Tracing through the justifications of these assertions gives us no more information.

The explanation we desire is that the assertion that Fred had indigestion led to a contradiction and colitis was an alternative diagnosis. To create this explanation, starting from belief in colitis, we must reunify the rule and assertions in the justification of colitis to construct the missing link: N-3. This sort of inference is necessary because there is neither a reference to N-3 nor to any contradiction node in the relevant justifications. Our hypothetical WHY function illustrates the need for this inference:

(HAS FRED ?X)

?X = COLITIS

(WHY)

Because Has Fred Colitis if

# And

1 (Not (Has Fred Indigestion))

2 (AND (?X) (Not (?X)))

\rightarrow (Contradiction)

(WHY 1)

Because Not (Has Fred Indigestion) is a premise.
The last WHY question didn’t help, but putting 1 and 2 together, we construct a new query which leads to a WHY-NOT:

(? (HAS FRED INDIGESTION))

NIL

(WHY-NOT)

Because Has Fred Indigestion if

# And

1 (Not (Has-a Fred Appendix))

2 (AND (Has ?X Appendicitis)

(Not (Has ?x Appendix)))

→ (Contradiction)

But (Has Fred Colitis) is IN.

Obviously, a complete explanation for node N-4 must refer to node N-3. It is important to realize, that to do so, an explanation mechanism would have to refer to the assertional content of nodes N-9 and N-10. It would then have to make the proper substitutions in rule N-10 to construct and determine the omitted node. Only then could it conclude to search for the node that asserts “(Has Fred Indigestion)”. This is all because node N-3 was not traceable by the available justifications. The same process is necessary to determine why it was originally believed that Fred had indigestion.

With the Charniak system, we lose information. We must infer that an assertion was made to resolve a contradiction by examining each of the supporting nodes to see if one might be a rule which concludes a contradiction. Certainly, a user, or a smart explanation mechanism, could infer the presence
of the missing node, construct it, and posit a contradiction node. But the point is that the justifications have lost their direct reflection of the complete reasons for belief or disbelief of an assertion, and thus the TMS has had its ability to provide explanations crippled.

1.5 Multiple Belief Spaces

Other approaches to the problem of belief revision have integrated the calling logic and the TMS to the extent that they are the same system. In such systems, contradiction detection rather than just resolution is typically performed. Such systems exacerbate the Charniak explanation problem because they no longer give access to the rules used to derive the explanation. One such system is the SWM system described in [15,16,17]. The Doyle scheduling problem is treated in [16]. One outstanding feature of this system is its capability for representing multiple conflicting situations. But the system does not represent contradictions explicitly. Rather, restrictions on individual hypotheses are updated to reflect conflicts and a current context is maintained that does not depend on conflicting hypotheses. In the case of conflicts, the user is allowed to choose among the conflicting hypotheses. This scheme does not seem to allow an explanation to be generated that reflects why one hypothesis is currently believed and another not. But rather than explore in detail this apparent deficiency, we think it is more important to point out that this is a significantly different sort of system than the TMS proposed by Doyle or Charniak.

In the SWM, one inherits a calling logic. One asserts neither contradictions nor rules which explicitly define them. Contradictions in the SWM
are automatically derived from assertions by internal mechanisms. This is not inherently undesirable, but such systems have essentially different characteristics than a “standard” TMS which performs its functions regardless of how the contradiction was produced. Therefore, we will not dwell on the explanation capabilities of such systems in detail except to the following extent.

When a calling logic is “built-in” the TMS with contradiction detection as an integral part of the system, the rules are implicit in the system rather than explicit. Thus they cannot be included in justifications. Explanations based solely on such justifications must be incomplete, even though the rules used are intuitive. This is well illustrated by the following system.

1.6 Three Valued Logic

McAllester’s Reasoning Utility Package (RUP) [18,19,20] is more similar to the TMS’s of Doyle and Charniak than is SWM. Its essential differences are that assertions have statuses of TRUE, FALSE, and UNKNOWN rather than IN and OUT; and that a calling logic is integrated with the TMS, although one much more simple than that of SWM. The following protocol from RUP shows the point made above about explanations:

(assert '(-> s t)
(-> S T)
(assert '(not t))
(NOT T)
(why 's)
((S IS FALSE FROM)
(1 (-> S T) IS TRUE)
(2 (T IS FALSE))

Notice that there is no mention of the implicit rule that causes RUP to conclude that S is false. It is a fairly obvious rule and the explanation is adequate, but the adequacy is dependent on the clarity of the rules of the system. In any case, we want to argue that SWM and RUP are at least a different class of a TMS than that of Doyle and Charniak. The former institutionalize incompleteness of justifications: not all of the dependency links between nodes are represented in their justifications. We ought to be able to cure the ills of the latter sort of a TMS while keeping the independence of the TMS from the calling logic. That is, we ought to be able to provide a contradiction resolution mechanism which produces justifications adequate for explanations without requiring special kinds of justifications such as the conditional proof, without reference to the assertional content of the nodes, and without recourse to pointers outside of the TMS justifications.
Chapter 2

A Solution

2.1 Elimination of Conditional Proof Justifications

We propose to introduce the contradiction into the OUTlist of the justification of the elective, omit the nogood node, and restrict contradictions to single justifications. For contradiction $C_x$, given culprit $A_i$ in maximal assumption set $A$, with OUTlist $D$ of the supporting justification of $A_i$, and elective $D_j \in D$, the INlist of the elective is $A - \{ A_i \} \cup I \cup BI$, and the OUTlist is $\{ C_x \} \cup BO \cup D - \{ D_j \}$, where $I$ is the INlist of $A_i$ and $BI$ and $BO$ are respectively the IN and OUT nodes required by Doyle [8] for the SL justification to implement the CP justification.

These last three sets of nodes are necessary to insure that the retraction of all non-maximal assumptions will invalidate this justification. In particular, the addition of the INlist in the justification is a correction to the algorithms of Doyle and Charniak. De Kleer seems to have been the first to publicly include all of the non-maximal assumptions in the justification of the elective [4]. He did so by definition. This is the first occurrence of an explicit correction to earlier algorithms.

This justification allows two possible status assignment states, one
of which we desire: namely that $C_x$ be OUT and $D_j$ be IN. Part of the solution is to modify the status assignment mechanism (SAM) to prefer a state in which all contradiction nodes are OUT, if there is such a consistent well-founded state. There will be, if the contradiction resolution mechanism (CRM) does not introduce any unsatisfiable circularities with the new justification. In addition, contradictions need not be resolved. If there is no consistent alternative, the SAM will allow contradictions to be IN.

Placing the contradiction in the OUTlist necessitates a revision of the definition of "assumption". Formerly, an assumption was any node with a supporting justification with a non-empty OUTlist. Now an assumption must be a node with a supporting justification with an OUTlist containing at least one element which is not a contradiction. When the maximal assumption set is created, the OUTlists of candidates for that set must be examined for contradictions. If the contradictions found are added to the OUTlist of the elective justification along with $C_x$, we gain two advantages. Firstly, our possible explanation for the status of the elective is improved: it was justified in order to resolve a set of contradictions. This point is related to the second. If any member of this set were to somehow lose its identity as a contradiction, it might be possible for the SAM to assign it a status of IN. In which case, $C_x$ would be resolved without the necessity of the elective having a valid justification. Thus the OUTlist of the justification of the elective should be $C \cup BO \cup D - \{D_j\}$ where $C$ initially contains $C_x$ and is constructed by adding new contradictions found during generation of the maximal assumption set.

This general idea is illustrated by example 1. Given the contradic-
tion at N-5, the new result would be:

<table>
<thead>
<tr>
<th>Node</th>
<th>Content</th>
<th>Justification</th>
<th>Belief</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-1</td>
<td>TIME(M) = 10:00</td>
<td>( )</td>
<td>(N-2)</td>
<td>IN</td>
</tr>
<tr>
<td>N-2</td>
<td>TIME(M) ≠ 10:00</td>
<td>( )</td>
<td>(N-4)</td>
<td>OUT</td>
</tr>
<tr>
<td>N-3</td>
<td>ROOM(M) = 813</td>
<td>( )</td>
<td>(N-1)</td>
<td>IN</td>
</tr>
<tr>
<td>N-4</td>
<td>ROOM(M) = 801</td>
<td>(N-1)</td>
<td>(N-5)</td>
<td>OUT</td>
</tr>
<tr>
<td>N-5</td>
<td>CONTRADICTION</td>
<td>(N-1 N-3)</td>
<td>( )</td>
<td>OUT</td>
</tr>
</tbody>
</table>

Example 3

Now, suppose we contradict N-4. While constructing the maximal assumption set N-1, we find contradiction N-5 and include it in the justification of the elective N-2 which resolves new contradiction N-6.

<table>
<thead>
<tr>
<th>Node</th>
<th>Content</th>
<th>Justification</th>
<th>Belief</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>N-1</td>
<td>TIME(M) = 10:00</td>
<td>( )</td>
<td>(N-2)</td>
<td>OUT</td>
</tr>
<tr>
<td>N-2</td>
<td>TIME(M) ≠ 10:00</td>
<td>( )</td>
<td>(N-5 N-6)</td>
<td>IN</td>
</tr>
<tr>
<td>N-3</td>
<td>ROOM(M) = 813</td>
<td>( )</td>
<td>(N-4)</td>
<td>IN</td>
</tr>
<tr>
<td>N-4</td>
<td>ROOM(M) = 801</td>
<td>(N-1)</td>
<td>(N-5)</td>
<td>OUT</td>
</tr>
<tr>
<td>N-5</td>
<td>CONTRADICTION</td>
<td>(N-1 N-3)</td>
<td>( )</td>
<td>OUT</td>
</tr>
<tr>
<td>N-6</td>
<td>CONTRADICTION</td>
<td>(N-4)</td>
<td>( )</td>
<td>OUT</td>
</tr>
</tbody>
</table>

Example 3a

This gives us the desired justification and explanation for N-2: the time of the meeting is not 10:00 because the meeting couldn’t be held at 10:00
in room 813 (N-5), room 801 wasn’t available (N-6), and there weren’t any alternatives (or they would have been included in the justification). The trick of adding N-5 in the justification of N-2 is a bonus. Even if it hadn’t been added, the justifications would still enable the user to directly trace back to all of the information required.

Since we now have direct reference to contradictions, the explanations would be better if an explanation mechanism treated nodes with contradictions in their justifications as special cases. This can be done by motivating the justification with the contradictions and listing the other supporting nodes as conditions. A “HOW” verb finds the supporting node for contradictions:

\[(\text{TIME}(M) = ?X)\]
\[?X = \lnot10:00\]

( Why )

Because \(\text{TIME}(M) = \lnot10:00\) resolves

# Contradiction set(s)

1 (Time = 10:00) (Room = 813)

2 (Room = 801)

( How 1 )

Because Time = 10:00 if

# And

1

But Time = \(\lnot10:00\) is IN.

If we had asked ( HOW 2 ), we would have gotten the answer:
Because Room(M) = 801 resolves
# Contradiction set(s)
1 (Time = 10:00) (Room = 813)

as long as

# And

1 Time = 10:00

But Time = 10:00 is OUT.

In no case is the explanation mechanism required to make inferences or postulate nodes not explicitly referred to in the TMS justifications.

2.2 Odd Loop Checking

No previous DDB algorithms have addressed another necessary property of the elective justification: that it not introduce an unsatisfiable circularity into the data base. If an element of the INlist 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 OUTlist 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. This is evident for the case when the elective occurs in the OUTlist of an already
invalid justification of an element in the justification of the elective.

<table>
<thead>
<tr>
<th>Justification</th>
<th>Belief</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>INlist</td>
</tr>
<tr>
<td>A</td>
<td>(D)</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>(E)</td>
</tr>
<tr>
<td>(F)</td>
<td>(B)</td>
</tr>
<tr>
<td>E</td>
<td>unspecified</td>
</tr>
<tr>
<td>F</td>
<td>unspecified</td>
</tr>
<tr>
<td>G</td>
<td>(A)</td>
</tr>
</tbody>
</table>

Example 4

Suppose in this example, that node G is a contradiction and that node B is chosen as an elective to resolve it. Then node D should be in the INlist of the justification for B. The believed repercussions of B do not contain D. However, suppose that at some future time, node E goes OUT and node F comes IN. Then we have an unsatisfiable circularity involving B and D. In general, we must check the transitive closure of the consequences (TCC) of the elective.

While we have not completely characterized this situation, we note the following. 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, the elective is still “safe” if there are only “even loops” containing that element and the elective. 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.

2.3 The Contradiction Resolution Process

The steps of the new contradiction resolution process can thus be described as follows:

Step 1: Given \( C_x \) to resolve, initialize list \( C \) to \( \{C_x\} \).

Step 2: For contradiction \( C_x \), find the maximal assumption set \( A = \{A_i\} \) as defined in [8] where an assumption must contain at least one node in its OUTlist which is not a contradiction. Let \( B \) be the set of nodes in the foundations of \( C_x \), not in \( A \) or in the repercussions of an element of \( A \), and either in an antecedent of \( C_x \) or of some node in the repercussions of an element of \( A \). Let \( BI \) be the IN nodes of \( B \) and \( BO \) the OUT nodes.

Step 3: Add to \( C \) new contradictions in the supporting nodes of the intersection of the foundations of \( C_x \) and the repercussions of the elements of \( A \).

Step 4: Choose the next \( A_i \) in the ordering of \( A \), starting with \( A_1 \). If all have been chosen before, the contradiction is not resolvable. Otherwise, let \( J \) be the supporting justification of \( A_i \). Then define \( I = \{I_k\} \) to be the set of nodes on the INlist of \( J \), and \( D = \{D_j\} \) to be the set of nodes on the
OUTlist of \( J \).

Step 5: For each element \( D_j \) of \( D \), if \( D_j \) is a contradiction node, then let \( C' = C \cup \{ D_j \} \) and \( D' = D - \{ D_j \} \). \( D' \) will be non-empty.

Step 6: Choose the next \( D_j \) in the ordering of \( D' \), starting with \( D_1 \). If all have been chosen before, go to step 4.

Step 7: If any element of \( A - \{ A_i \}, BI, I, BO, \) or \( D' \) is in the transitive closure of the consequences of \( D_j \) via an odd loop, go to step 6.

Step 8: Add a justification for \( D_j \) that consists of an INlist with the nodes in the set \( I \cup A - \{ A_i \} \cup BI \) and an OUTlist with the nodes in the set \( C' \cup BO \cup D' - \{ D_j \} \).

This algorithm is similar to that of [8]. The exceptions are that the CP justification is collapsed into sets \( BI \) and \( BO \), additional contradictions are gathered into set \( C \), the INlist of the culprit is included in the elective justification, and there is checking for odd loops in step 6.
Chapter 3

Reformulating Dependency-Directed Backtracking

3.1 Characterization of DDB

Elimination of the CP justification means collapse of a hierarchy of assumptions implicit in that justification. In Doyle the maximal assumptions directly supported the elective while some non-maximal assumptions underlay the SL justification that implemented the CP justification that supported the nogood node. (Actually, this hierarchy was only partial if the INlist of the culprit is correctly included in the justification of the elective.) If we no longer maintain this hierarchy, there may be no good reason to go to the trouble to generate the maximal assumption set.

However, Doyle correctly intended to make some attempt at “minimally revising” the database. Retracting a maximal assumption is guaranteed to cause no more changes to the database than retracting a non-maximal assumption. Any algorithm which eliminates the maximal assumption set should also select a culprit with such a guarantee. We will show this can be done as a part of examining the transitive closure of the consequences which represents work that must be done in any case.

In general, we wish to resolve a contradiction by retracting an as-
sumption in its foundations. One assumption is chosen as the culprit. It is retracted by justifying an elective. This justification should have three properties:

1. The justification should allow a consistent assignment of support statuses such that all valid justifications are well founded [27] and 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 in [26].)

3. The justification is "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 will become invalid.)

The DDB algorithm described in Section 2.3 had these properties. It also allowed for the possibility of contradiction retraction. If the datum which had been declared to be a contradiction were to lose that tag and subsequently be assigned a status of IN, then the justification of the elective would be invalid. However, this algorithm, as did Doyle's, generated a maximal assumption set which has disadvantages.

3.2 Maximal Assumption Set

For a contradiction C, the maximal assumption set contains those assumptions which are nearest to C in its foundations and none of which are
<table>
<thead>
<tr>
<th>Node</th>
<th>Justification</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contradiction</td>
<td>$(A_1, A_2)$</td>
<td>IN</td>
</tr>
<tr>
<td>$A_1$</td>
<td>()</td>
<td>IN</td>
</tr>
<tr>
<td>$A_2$</td>
<td>()</td>
<td>IN</td>
</tr>
<tr>
<td>$E_1$</td>
<td>$(E_1)$</td>
<td>OUT</td>
</tr>
<tr>
<td>$E_2$</td>
<td>$(E_1)$</td>
<td>OUT</td>
</tr>
</tbody>
</table>

Figure 3.1: A Potential Odd Loop

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 data base. Retraction of a maximal assumption is guaranteed to cause fewer changes than retraction of a non-maximal assumption.

The set is included in the elective justification to ensure that if some other assumption than the culprit loses its support status of IN, then the elective will no longer be justified. The definition of the maximal assumption set also provides partial protection against unsatisfiable circularities. It is difficult to characterize exactly when such circularities occur. However, they will occur only if there is an odd loop. One way to introduce an odd loop during contradiction resolution is to include in the INlist of the justification of the elective some element which is a believed repercussion of the culprit. By definition, this can not happen with maximal assumptions.

Nevertheless, an odd loop may still result from the inclusion of maximal assumptions in the INlist of the elective justified to retract the culprit, as illustrated by figure 1. In this example, \{A_1, A_2\} is the maximal assumption
set for the contradiction. Were $A_1$ to be selected as the culprit, $A_2$ would
be included in the justification of $E_1$. Obviously, if $E_1$ is IN, then $E_2$ is IN
and thus $A_2$ is OUT. But, $A_2$ is in the INlist of $E_1$, so it must be OUT as
well. The choice of $A_1$ as the culprit has introduced an odd loop. As was
pointed out previously, it is necessary to examine the transitive closure of
the consequences (TCC) of the elective, stopping at the contradiction, to be
certain no odd loop results from the justification of the elective.

Not only must we check the TCC of the elective even when we have
a maximal assumption set, this check also allows us to safely work with a set
of assumptions which are 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 $E$.

Figure 2 gives an example of such a situation. In this somewhat
careless example of knowledge representation, there are two assumptions in
the foundations of the contradiction: nodes 1 and 2. However, node 2 is in the foundations of node 1 and thus must be excluded from the maximal assumption set which consists only of node 1. But suppose the maximal assumption set was not generated and we had chosen node 2 as the culprit. Although this selection might make more sense intuitively, it would have introduced an unsatisfiable circularity into the data base due to the necessity of including node 1 in the INlist of the justification of node 4, the elective. This would have been detected by examination of the TCC of node 4, node 2 would have been rejected as a culprit, and node 1 chosen just as in the case of using the maximal assumption set. (Node 3 will be the elective in this case. If this is rejected by a subsequent contradiction, node 2 will be selected as the culprit and node 4 safely justified.)

Generating the maximal assumption set requires that we completely examine the foundations of the contradiction. But we must also examine the transitive closure of the consequences of the elective to be safe. Thus, for this purpose, the foundation examination represents unnecessary work. Further work is also required by the algorithms of [8] and Section 2.3 to find the non-maximal assumptions to ensure completeness. We now present an algorithm which eliminates all of this work except that of examining the TCC of the elective, still permits minimal revision, and generates a safe and complete justification.
3.3 "Lazy" DDB-based Contradiction Resolution

Let set S be the supporters of contradiction C. (Assume for now that contradiction nodes have empty OUTlists.) The contradiction is resolved iff predicate MAKEOUT(S, nil, {C}) returns a justification associated with an elective satisfying the three properties of Section 3.1. The contradiction is still included in the OUTlist of the justification as described in Chapter 2. In the definition of the required predicates below, if an "else" clause is not specified for an "if" statement, then assume one pointing to the next step.

Define predicate 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(OUTlist of \( J_i \), IJ \( \cup \) INlist of \( J_i \cup A - \{A_i\}, OJ \)), then return it.
3. If MAKEOUT(A - \{A_i\}, IJ \( \cup \) \{A_i\}, OJ) then return it.
4. Return MAKEOUT(INlist of \( J_i \), IJ \( \cup \) A - \{A_i\}, OJ \( \cup \) OUTlist of \( J_i \)).

Define predicate 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 \( \cup \) \{A_i\}).
3. Construct justification $J_e$ for $A_i$ with an INlist consisting of IJ, and an OUTlist 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 $\cup \{A_i\}$).

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 predicate, we first try to make some element of the OUTlist 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 INlist 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. The examination of TCCs of candidate culprits is optional: it is included above because it is typically more efficient in some applications to terminate the search early rather than later. However, this is not always the case.

This simple search will create a safe and complete justification to resolve the contradiction. 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 those contradictions to be placed in the OUTlist of the justification of the elective that were previously collected during generation of the maximal assumption set in the algorithm of Section 2.3. This method is improved and extended below.
Chapter 4

Extended Contradiction Resolution

4.1 Assumptions, Defaults, and Alternatives

A TMS is useful for expert systems. It provides a uniform method for generating explanations and allows nonmonotonic reasoning. Contradiction resolution adds to the inferencing and knowledge representation capabilities of an expert system. More importantly, it allows the user to designate undesirable situations and to disagree with system results obtained by default reasoning. However, the methods of contradiction resolution described by [8] and above have an undesirable effect in this environment.

In a TMS, an assumption is a node with an OUT supporter other than a contradiction. Informally, this represents belief in the assertion associated with the node which assumes that there is no reason to believe any of the assertions associated with the OUT supporters. For the purposes of knowledge representation, it is convenient to think of the assumed assertion as the “default” and the assertions of the OUT supporters as “alternatives” which are a priori either less likely or less preferred.

When an alternative has no justification, the semantics of contradiction resolution are clear. Prior to the contradiction, there was no reason to prefer an alternative to the default. Once the default has been involved in
a contradiction, we have such a reason. The selected alternative is given the justification that the contradiction caused us to doubt the default in some set of circumstances. (This is an interpretation of the justification given in Section 2.3.)

When the alternative has an existing but invalid justification, we have a potential reason for believing the alternative in preference to the default. To make a subtle distinction, it is not that there is no reason at all: there is at least one but it is currently not believed. In the previous case, our default rested on the fact that the alternative had no justification at all. Perhaps the invalidity of one of the alternative's justifications is similarly founded on the lack of justifications for some set of nodes. We propose that these are the nodes, rather than the alternative itself, that should be justified by contradiction resolution.

Suppose we have an expert system for diagnosing diseases of some sort and a particular disease is the alternative to some default diagnosis which has been contradicted by new data. Suppose further, that a knowledge engineer has specified that this alternative disease is to be diagnosed if the patient displays a certain symptom which is represented as a justification for the disease. Then believing that the patient has the disease solely because the default was contradicted ignores the design of the knowledge engineer. The disease should be diagnosed just when the symptom is present. We do this if we justify the symptom, instead of the disease.

Further, there may be a rule or justification which specifies that the alternative disease is not to be diagnosed under some set of circumstances. Not only do we wish to ensure that we do not make such an improper diag-
nosis, but that if the specified set of circumstances is ever asserted, then we
no longer believe the diagnosis of that alternative disease. To be able to do
so is one of the primary benefits of a TMS for an expert system. If we only
add a new justification and ignore existing ones, then this benefit is lost.

4.2 The Extended Algorithm

We now extend the method of lazy contradiction resolution of Sec-
tion 3.3. This contradiction resolution mechanism (CRM) works by first try-
ing to make IN some element of the OUTlist of the contradiction justification.
If that fails, the CRM attempts to make some element of the INlist OUT. A
node is made IN by either generating a new justification for it, or by making
an existing one valid. A justification is made valid if all of the nodes in its
OUTlist are made OUT and all in its INlist made IN. A node is made OUT
by invalidating each of its justifications by making some element either of its
OUTlist IN or of its INlist OUT. A node is also considered to be made OUT if
it has no justification unless it is planned to be made IN, and it is made IN if
it has a premise justification. A node will not receive a justification to
resolve a contradiction if it already has one, unless the node is justified
solely to resolve contradictions. For a given set of nodes, one of which must
be made IN, the CRM will first select one with no justification, if any.

In this extended CRM, MAKEIN and MAKEOUT are called by
CHECKIN and CHECKOUT to first determine if the desired status of one of
the nodes in question is not already guaranteed. MAKE-ALL-IN and MAKE-
ALL-OUT simply require that the desired status be obtained for every node
in their first argument, rather than only one. MAKE-INVALID takes as its
first argument a set of justifications and attempts to invalidate some member of that set. These predicates are defined in this section and return either "false" or an ordered set of values described below:

**INC:** If a node is in the set INC, the predicate using this set cannot succeed by making that node OUT. The predicate can succeed by assuming that the node has been made IN only if the node is not also an element of the set F.

**OUTC:** Statements analogous to those for INC are true for OUTC.

**NEWJ:** This is a set of pairs. The first element of each pair is a node and the second is a justification to be added to that node.

**F:** This is a set of nodes which must be made IN or made OUT for the predicate using this set to succeed.

We will let L stand for the vector of these items. Thus, for such a predicate P, P(X, L) is shorthand for P(X, INC, OUTC, NEW-J, F). After a call to one of these predicates, L will be assumed to be automatically assigned the result of the call. Thus L = P(X, L). Also, if we reassign the value of one of the elements of L, then future use of L assumes the new value. We will also use the parameters IJ and OJ as in the previous algorithm. We use K as shorthand for <IJ, OJ, L>, in the same way that we use L.

Let S+ be the INlist of contradiction C, and S- its OUTlist. Initialize K to be the vector <nil, {C}, nil, {C}, nil, nil>. The CRM succeeds if either CHECKIN(S-, K) or CHECKOUT(S+, K) does. If either succeeds, inhibit calls to the CRM and add the justifications to the nodes in the set.
NEW-J which was returned, enabling calls to the CRM for the addition of
the last justification in the set. The necessary predicates are defined below.

Define \textbf{CHECKOUT}(A, K):

1. If an $A_i \in A$ is a contradiction, or is $\in$ OUTC and not $\in$ F, or
   not $\in$ INC and either has no justifications or is OUT and is not in the
   TCC of any elective already in NEWJ, then
   \[ \text{OUTC} := \text{OUTC} \cup \{A_i\} \text{ and } F := F - \{A_i\} \]. Return L.
   Else return \text{MAKEOUT}(A, K).

Define \textbf{MAKEOUT}(A, K):

1. If A is null, return "false".

2. Pick $A_i \in A$ and construct its TCC. If $A_i \in$ INC, $\in$ IJ, or has a premise
   justification, or if there is an odd path in its TCC from $A_i$ to $A - A_i$ or
   IJ, then $IJ := IJ \cup \{A_i\}$.
   Return \text{MAKEOUT}(A - \{A_i\}, K).

3. Let J be the set of justifications of $A_i$. If \text{MAKE-INVALID}(J,
   $IJ \cup A - \{A_i\}$, OJ, INC, OUTC $\cup \{A_i\}$, NEW-J, F $-$ \{A_i\}),
   then return it.

4. Return \text{MAKEOUT}(A - \{A_i\}, IJ $\cup \{A_i\}$, OJ, INC, OUTC, NEW-J,
   F).

Define \textbf{CHECKIN}(A, K):
1. If an $A_i \in A$ is $\in$ INC and not $\in$ F, or has a premise justification, or is not $\in$ OUTC and is IN and not in the TCC of any elective in NEWJ, then INC := INC $\cup$ \{A_i\} and F := F$-$ \{A_i\}. Return L.

Else return MAKEIN(A, K).

Define MAKEIN(A, K):

1. If A is null, return "false".

2. Pick $A_i \in A$. If $A_i \in$ OUTC, $\in$ OJ, or is a contradiction, then
   
   OJ := OJ $\cup$ \{A_i\}. Return MAKEIN(A - \{A_i\}, K).

3. Let J be the set of justifications of $A_i$. If J is not null, and if MAKEIN(A - \{A_i\}, IJ, OJ $\cup$ \{A_i\}, INC, OUTC, NEW-J, F), then return it.

4. If J is null, then let $J_e$ be constructed with an INList of IJ and an OUTList of OJ $\cup$ A - \{A_i\}. If $J_e$ will not result in an odd loop with its repercussions or INC or OUTC, then INC := INC $\cup$ \{A_i\}, NEWJ := NEW-J $\cup$ \{A_i\}, and F := F$-$ \{A_i\}. Save the TCC of $J_e$. If it contains no contradictions, return L. Otherwise, recurse the CRM on them and return the result.

5. Let OJ := OJ $\cup$ A - \{A_i\} and INC := INC $\cup$ \{A_i\}. If MAKE-VALID (J, K), return it.

6. Return MAKEIN(A - \{A_i\}, IJ, OJ $\cup$ \{A_i\}, INC, OUTC, NEW-J, F).

Define MAKE-VALID(J, K):

1. If J is null, return "false".
2. Pick $J_k \in J$. Let $D$ be the OUTlist and $E$ the INlist of $J_k$. Let $B$ be a new node that asserts that $A_i$ is justified other than by resolution of the contradiction. Let $NJ$ be the set of vectors $<B, J_j>$ where $J_j \in J - \{J_k\}$. Let $L'$ be $\text{MAKE-ALL-OUT}(D, IJ, OJ \cup \{B\}, \text{INC} \cup E, \text{OUTC} \cup D \cup \{B\}, \text{NEW-J} \cup \text{NJ}, (F \cup (D - \text{OUTC})) \cup (E - \text{INC})) - \{A_i\})$. If $L'$ is not "false", return $\text{MAKE-ALL-IN}(E, IJ, OJ \cup \{B\}, L')$. Else return $\text{MAKE-VALID} (J - J_k, K)$.

Define $\text{MAKE-INVALID}(J, K)$:

1. If $J$ is null, return $L$.

2. Pick $J_k \in J$. Let $IJ' := IJ \cup \text{INlist of } J_k$. Let $L'$ be $\text{CHECKIN}(\text{OUTlist of } J_k, IJ', OJ, L)$. If $L'$, return $\text{MAKE-INVALID}(J - \{J_k\}, IJ', OJ, L')$. Otherwise, let $OJ := OJ \cup \text{OUTlist of } J_k$. Let $L$ be $\text{CHECKOUT}(\text{INlist of } J_k, K)$. If $L$, return $\text{MAKE-INVALID}(J - \{J_k\}, K)$. Else return "false".

Define $\text{MAKE-ALL-IN}(A, K)$:

1. If $A$ is null, return $L$.

2. Pick $A_i \in A$. Let $L$ be $\text{CHECKIN}(\{A_i\}, K)$. If $L$, then return $\text{MAKE-ALL-IN}(A - \{A_i\}, IJ, OJ, L)$. Else return "false".

Define $\text{MAKE-ALL-OUT}(A, K)$:

1. If $A$ is null, return $L$.

2. Pick $A_i \in A$. Let $L$ be $\text{CHECKOUT}(\{A_i\}, K)$. If $L$, then return $\text{MAKE-ALL-OUT}(A - \{A_i\}, IJ, OJ, L)$. Else return "false".
4.3 Extension Differences

The previous algorithm worked by recursively making some node OUT by either making some member of the OUTlist of the supporting justification of the node IN, or by making some member of the INlist OUT by the same method. In this algorithm, which has been implemented as part of an expert system, we examine all of the justifications of the node to be made OUT, rather than just its supporting justification. This is because previously, the only way to make a node IN was to give it a contradiction justification. The extended algorithm allows a node to be made IN by validating one of its justifications (see step 6 in MAKEIN) rather than simply adding a new justification.

The new parameters INC and OUTC provide a record, similar to a TMS justification, of nodes and desired statuses which have been made dependent on the current predicate call with the exception of those nodes in F. The statuses of the latter are desired but not determined. These parameters are now required to record nodes traversed, or intended for traversal, through a graph.

Since this algorithm validates justifications, in order to preserve completeness, it is necessary to take into account existing invalid justifications. Suppose that some node X is to be made IN to resolve some contradiction. Node X has two existing invalid justifications J1 and J2. Justification J1 is made valid by justifying some elective E. Later, if J2 became valid, it would not be necessary to believe E in order to believe X, which resolves the contradiction. In general, we need to ensure that the later validity of previously invalid justifications does not resolve a contradiction while leaving an
elective IN.

In MAKE-VALID, we construct a new node B which asserts that the node to be made IN should be believed independently of resolution of the contradiction. The node B receives all the justifications of the original node except the one currently being validated. The OUTlist of the justification of the relevant elective will include B. Thus, the elective is guaranteed to go OUT should one of the related previously invalid justifications become valid. This does not account for new valid justifications and this is a source of incompleteness for the current algorithm.

Finally, the above algorithm can be made more efficient by having MAKE-INVALID first check to see if some element of the OUTlist of a justification is already IN, or of the INlist OUT, by use of the methods described in CHECKIN and CHECKOUT respectively. However, doing so here would add unnecessary functions to the exposition.

A version of this algorithm has been implemented in an experimental expert system support program called “Proteus” at Microelectronics and Computer Technology Corporation and is being used in several applications.

4.4 An Example of Extended Resolution

Suppose that we have built an expert system for constructing electronic circuits with the following rules:

1. If a counter is desired, construct it with a flip-flop of type 1 rather than one of type 2 unless there is specific reason to do so.
2. The output of the flip-flop used in 1 is equivalent to register-bit 1 of the counter.

3. A flip-flop of type 2 must have an output-enable line.

4. One can use a flip-flop of type 2 only when its output-enable line is not grounded low and just when one is either using CMOS or a bus circuit.

5. The output of a flip-flop of type 1 is tristate.

6. The output of a flip-flop of type 2 is totempole.

7. If two things are equivalent and one is totempole, then so is the other.

8. Register-bit 3 of any memory is totempole.

9. One may not connect tristate and totempole (outputs).

In such a circuit, when a counter is desired, it is constructed using a flip-flop of type 1 by preference, unless it is known that CMOS or a bus circuit is being used (these rules are not particularly meaningful). Later, the circuit designer may want to connect the first bit of the register of the counter to the third bit of a memory. But this causes a contradiction.

This contradiction can be resolved by assuming that a flip-flop of type 2 should be used instead of type 1. Rule 3 entails that we believe that either CMOS or a bus circuit is used. Since we don’t know which one, we might follow a path of least commitment and assume neither. De Kleer [5,6] has proposed a system of contradiction resolution that involves no backtracking at all. Such a system allows all possible (consistent) worlds until an interpretation is finally chosen.

However, we know that we desire one of these possibilities to be the case. We may have an a priori preference. And each possibility may have
<table>
<thead>
<tr>
<th>Node</th>
<th>Proposition</th>
<th>Justification</th>
<th>Support Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(Counter A)</td>
<td>(1)</td>
<td>IN</td>
</tr>
<tr>
<td>2</td>
<td>(F/F-Type 1 A)</td>
<td>(1) (3)</td>
<td>IN</td>
</tr>
<tr>
<td>3</td>
<td>(F/F-Type 2 A)</td>
<td>(4) (8)</td>
<td>OUT</td>
</tr>
<tr>
<td>4</td>
<td>(Not (F/F-Type 2 A))</td>
<td>(5) (6)</td>
<td>IN</td>
</tr>
<tr>
<td>5</td>
<td>(CMOS used)</td>
<td></td>
<td>OUT</td>
</tr>
<tr>
<td>6</td>
<td>(Bus circuit used)</td>
<td></td>
<td>OUT</td>
</tr>
<tr>
<td>7</td>
<td>(Has A output-enable)</td>
<td>(3)</td>
<td>OUT</td>
</tr>
<tr>
<td>8</td>
<td>(Low output-enable A)</td>
<td></td>
<td>OUT</td>
</tr>
<tr>
<td>9</td>
<td>(Totempole (Reg-bit 1 A))</td>
<td>(1) (2)</td>
<td>IN</td>
</tr>
<tr>
<td>10</td>
<td>(Tristate (Reg-bit 1 A))</td>
<td>(1) (3)</td>
<td>OUT</td>
</tr>
<tr>
<td>11</td>
<td>(Memory B)</td>
<td>(5)</td>
<td>IN</td>
</tr>
<tr>
<td>12</td>
<td>(Tristate (Reg-bit 3 B))</td>
<td>(1) (11)</td>
<td>IN</td>
</tr>
<tr>
<td>13</td>
<td>(Connect (Reg-bit 1 A)</td>
<td>(9) (12) (13)</td>
<td>IN</td>
</tr>
<tr>
<td>14</td>
<td>(Contradiction)</td>
<td></td>
<td>IN</td>
</tr>
</tbody>
</table>

**Figure 4.1: Default Circuit Design**

Different consequences that constrain the design. These consequences may suggest to us how to proceed with the circuit design. Therefore, at least in some cases, it is reasonable to make such a choice earlier rather than later.

With respect to DDB-based contradiction resolution, the important point is not to miss the existing justifications. In the present example, regardless of our choice of elective, the existing justifications tell us that the flip-flop of type 2 has an output enable line and that if it is ever grounded, then this solution won’t work.

Figure 3 gives a simplified TMS representation of a particular situation that might be produced by such rules, ignoring an explicit representation
of the rules. A contradiction results from the construction of two devices, A and B, and their subsequent connection. The first algorithm presented here would justify node 3 to resolve the algorithm, leaving node 4 IN and disregarding the current or future status of node 8. The second algorithm would justify node 5 with nodes 12 and 13 in the INlist and nodes 14, 8 and 6 in the OUTlist. If the user later wants to know why he is using CMOS, the answer is available from the justification.

If the user or subsequent circumstances caused an objection to the assumption of node 5, node 6 would be justified. (For instance, in one implementation of this algorithm, the user would be informed of the resolution of the contradiction by the assumption of node 5 and can object simply by typing "WRONG".) If node 6 was then later contradicted, the design would contain an unresolved contradiction. Even if either 5 or 6 remains IN, if node 8 is justified (through inadvertent grounding of the output enable line), then the user will discover an unresolvable contradiction.
Chapter 5

Conclusion

5.1 Future Work

The extended CRM implements the very weakest sort of abductive inference and requires a guidance mechanism to be useful in an expert system. However, the CRM provides the requisite foundation for intelligent backtracking. In general, the reasoned retraction of assumptions and the guidance of proofs are major research topics in artificial intelligence [9,10,11,3,2,13]. In our initial implementation, we provide interactive guidance.

Knowledge-based guidance may allow us to eventually ignore the principle of minimal revision in some cases. If so, then we may also want to ignore odd loop checking. Although this could introduce unsatisfiable circularities into the data base, we may be able to eliminate the burden of checking for such by the CRM. A status assignment mechanism such as that of [27] will avoid unsatisfiable circularities if possible. Our method of including the contradiction in the OUTlist of the elective makes such avoidance possible. The immediate result of an odd loop produced by an attempt at contradiction resolution is only that the contradiction remains IN. If the CRM can be made sufficiently intelligent to recognize this presumably uncommon occurrence, then to the extent that we can ignore minimal revision, we can obtain
relatively cheap problem solving by backtracking.

The extended CRM does not handle cycles. For example, if the elective supported itself, the CRM would not terminate. This is similar to attempting a backward chaining proof using a rule where the antecedent and consequent were identical. We think that this problem can be handled by adding another parameter, but this has not been implemented.

The CRM will justify an elective of the form “There exists X such that P(X)” with the standard justification when it may be perfectly obvious that P(X) for a particular X should be justified instead. We need at least to allow the user to be able to write rules to represent such knowledge.

Since contradictions only represent undesirable situations, we should like to give the user the option to declare that some situations are less desirable than others. This is particularly important if not all contradictions can be resolved. Such a ranking of contradictions should also contribute to the preference factors previously discussed.

Dependency-directed backtracking contradiction resolution also offers a chance for some degree of debugging/knowledge acquisition. If a contradiction cannot be resolved, than it must be that some assumption is missing from the data base. By examining the foundations of the contradiction, the user can be led to the logical site of the missing assumption. Since assumptions are caused by “unless” clauses in rules, the process can be inverted to further qualify the rule responsible for the incomplete justification.
5.2 Summary

We have presented three dependency-directed backtracking contradiction resolution algorithms with advantages over previous ones. The first eliminated the need for conditional justifications, avoided introducing unsatisfiable circularities, and corrected errors in previously published algorithms. The second requires less work to provide a correct solution and provides a basic, simple and general algorithm for dependency-directed backtracking. It is easily extended to a contradiction resolution algorithm which provides an abductive inference method using the justifications of a TMS in an expert system. Although inference by contradiction resolution requires guidance to be useful, the algorithm presented provides some basic logical constraints on the choice of electives. Further, it ensures that assumptions made in order to resolve a contradiction will become invalid when no longer required.
BIBLIOGRAPHY


