Variations on Backtracking for TMS

Ulrich Junker
GMD
P O Box 1240
5205 St. Augustin 1
Fed. Rep. of Germany

Abstract

We review the backtracking technique of Doyle's Truth Maintenance System (TMS). Backtracking in principle is to recover from a failure state by retracting some assumptions. We discuss three possible applications for the TMS: (1) Transforming a contradiction proof into a direct proof if a false-node is labelled with IN. (2) Selecting another extension if an odd loop failure arises. (3) Doing theory revision if no extension exists. In this paper, we elaborate the first and the second alternative. If the TMS is used in an autoepistemic or default prover (as in [15]) it needs sufficient justifications for first-order proofs. We show how backtracking allows to complete this set incrementally by applying a special case of the deduction theorem. For finding extensions, we describe Doyle's label methods as meta rules deriving statements of the form $\text{IN}(p)$ and $\neg\text{IN}(q)$. If a node q of an odd loop is labelled with IN and OUT we obtain a contradiction in this meta reasoning system. Hence, we can again apply backtracking to identify and retract the label assumptions being responsible for the failure. Thus, we obtain an efficient TMS that detects an extension whenever one exists.

1 Introduction

Reason maintenance systems may be used to record proofs, to support efficient search, and to do non-monotonic reasoning. We are mainly interested in using such a system as a basis for non-monotonic theorem provers. This facility has at best been elaborated in Doyle's TMS [8] because it provides non-monotonic justifications as a basic constituent. Other systems such as the multiple belief reasoner (MBR) of Martins and Shapiro [17], or de Kleer's ATMS [4] must be extended to get a restricted form of non-monotonic reasoning ([3], [11], [14], [12]). Very often, an extra search procedure is needed to find extensions which is itself similar to the TMS. The TMS also seems to be interesting because it can be used to find extensions of autoepistemic and default theories. This has been elaborated in a joint work with Kurt Konolige [15].

However, the TMS suffers from some problems. Most label algorithms are incomplete in the sense that they do not find an existing extension if a so called 'odd loop error' arises: In this case, Doyle's TMS does not terminate [8], whereas Goodwin's system stops without definite result [13]. Chronological backtracking helps, but decreases efficiency. It has been implicitly used by Russinoff [21] and is also considered in [15]. Doyle also implemented a technique called dependency-directed backtracking which misses a clear theoretical analysis and foundation. Our goal is to clarify backtracking and to elaborate well-founded applications in the context of the TMS. Thus, we will get rid of some obstacles towards an efficient non-monotonic theorem prover.

Backtracking in general is to recover from a failure state by retracting some assumptions. We distinguish three possibilities to make this more precise.

1. **Deductive Backtracking:** If the TMS is used in an autoepistemic or default prover it needs sufficient justifications for first-order proofs. We show how backtracking allows to complete this set incrementally by transforming a contradiction proof into a justification. Thus, we can use the monotonic part of the TMS as an efficient propositional prover which is similar to a tableau prover.

2. **Label-Backtracking:** For finding extensions, we describe Doyle's label methods as meta rules deriving statements of the form $\text{IN}(p)$ and $\neg\text{IN}(q)$. If a node q of an odd loop is labelled with IN and OUT we obtain a contradiction in this meta reasoning system. Hence, we can again apply backtracking to identify and retract the label assumptions being responsible for the failure.

3. **Theory Revision:** In Doyle's latter paper on the TMS (cf. [9], [10]), he gives a precise motivation for backtracking: Doing a minimal change if the current belief state becomes inconsistent because of new information. However, the implemented backtracking has not been changed to achieve this requirement. Brewka discusses some problems of minimal changes and backtracking [1]. We do not further pursue this topic because it is beyond the scope of this paper.

2 Doyle's TMS

Doyle's TMS maintains a set of nodes and a set of justifications. Justifications can be applied to nodes and thus derive further nodes. In [15] we have introduced a special notation for this purpose which is based on operators and their closures. This section cites the basic notions of this paper.

Let U be an arbitrary domain and *apply* : $2^U \rightarrow 2^U$ be a monotonic and compact operator. Hence, it satisfies

$$\text{if } X \subseteq Y \text{ then } apply(X) \subseteq apply(Y) \tag{1}$$

$$\text{if } q \in apply(Y) \text{ then } \exists X \subseteq Y : X \text{ is finite and } q \in apply(X) \tag{2}$$

Usually, we repeat application of such an operator. This motivates the following definition of a closure operation:

Definition 2.1 *(cited from [15]) Let U be a domain and apply : $2^U \rightarrow 2^U$ be a monotonic and compact operator. The closure $apply^*(X)$ is the minimal set that contains X and is closed w.r.t. apply, i.e.*

1. $X \subseteq apply^*(X)$
2. $apply(apply^*(X)) \subseteq apply^*(X)$.

It is easy to show that closedness and minimality are equivalent to closedness and groundedness which are used to define TMS-extensions (cf. [9] and [19]). In the latter approach, finite derivation chains are used to avoid circular arguments:

Lemma 2.1 *(cited from [15]) Let apply : $2^U \rightarrow 2^U$ be a monotonic, and compact operator. Let T be a subset of U. The following conditions are equivalent:*

1. $T = apply^*(\emptyset)$
2. $apply(T) = T \subseteq apply^*(\emptyset)$
3. $apply(T) \subseteq T$ *and* $\forall q \in T\ \exists q_1, \ldots, q_k : q_i \in apply(\{q_1, \ldots, q_{i-1}\})$ *for* $i = 1, \cdots, k$ *and* $q = q_k$.

Now, we can turn our attention to Doyle's TMS. This system is supplied with a justification network $\nu := (N, J)$. N is a set of nodes which are justified by justifications in J. A justification j consists of an in-list I, an out-list O, and a consequent c. Since we miss a unique notation in the literature we write $\langle in(I), out(O) \rightarrow c\rangle$ for j which is similar as in [19]. Informally, a justification can be applied if every element of its in-list is valid, but no element of its out-list. In [15], we have used monotonic and compact operators $apply_{J,Y} : 2^N \mapsto 2^N$ for applying justifications. For this purpose, we must handle in-list and out-list separately by using an extra index Y. Otherwise, we would not get monotonicity:

$$apply_{J,Y}(X) := \{c \mid \langle in(I), out(O) \rightarrow c\rangle \in J, I \subseteq X \text{ and } O \cap Y = \emptyset\} \tag{3}$$

This operator satisfies some interesting monotonicity properties:

1. if $X_1 \subseteq X_2$ then $apply_{J,Y}(X_1) \subseteq apply_{J,Y}(X_2)$
2. if $Y_1 \subseteq Y_2$ then $apply_{J,Y_1}(X) \supseteq apply_{J,Y_2}(X)$
3. if $J_1 \subseteq J_2$ then $apply_{J_1,Y}(X) \subseteq apply_{J_2,Y}(X)$

A justification network can have extensions (or admissible labellings). As discussed in [15], an extension T can be characterized as the minimal set which is closed w.r.t. $apply_{J,T}$. According to lemma 2.1 this is equivalent to usual definitions in terms of groundedness and closedness (cf. [9], [19]):

Definition 2.2 *(cited from [15]) Let $\nu = (N, J)$ be a justification network. A subset T of N is an extension of ν iff $T = apply^*_{J,T}(\emptyset)$.*

If a network has no extension then it is called incoherent.

3 Deductive Backtracking

In [15], we showed that the TMS can be used to find extensions of autoepistemic and default theories[1]. For this purpose, we supplied sufficient monotonic justifications for first-order derivations (in addition to non-monotonic justifications for defaults). Thus, we splitted J into a set M of monotonic justifications and a set NM of non-monotonic justifications. In this section, we provide a better encoding of the monotonic part for a special kind of theories, namely propositional horn theories. We consider default theories $\Delta = (D, W)$ where W consists of horn clauses $a_1 \wedge \ldots \wedge a_k \supset c$ and D of defaults $(a : \neg b_1; \ldots ; b_k; c/c)$ s.t. a, a_i, and b_j are propositional constants and c is a propositional constant or the unsatisfiable constant $\perp$.

Consider a simple example $\Delta = (\{\frac{:a}{a}, \frac{:b}{b}, \frac{:c}{c}\}, \{a \wedge b \supset d, c \wedge d \supset \perp\})$. According to [15], we get the following justifications:

$$NM = \{\langle out(\neg a) \rightarrow a\rangle, \langle out(\neg b) \rightarrow b\rangle, \langle out(\neg c) \rightarrow c\rangle\}$$

$$M = \{\langle in(b, c) \rightarrow \neg a\rangle, \langle in(a, c) \rightarrow \neg b\rangle, \langle in(a, b) \rightarrow \neg c\rangle\}$$

The three normal defaults are translated into three non-monotonic justifications in a modular way. To retract such a justification we need an argument for a negative formula. For this purpose, we determine all proofs of $\neg a$, $\neg b$, and $\neg c$, thus obtaining three monotonic justifications. This translation step is not modular. However, we want a more modular translation for horn theories and translate every horn clause into a justification:

$$M_W := \{\langle in(a_1, \ldots, a_k) \rightarrow c\rangle \mid a_1 \wedge \ldots \wedge a_k \supset c \in W\} \tag{4}$$

[1]Indeed, the same network can be used for both logic due to Konolige's work on their relation [16]. For weakly grounded extensions, a slight revision of a TMS-extension has been introduced in [15]. The results of this paper can easily be adapted to these +-extensions

This set is complete w.r.t. the derivation of contradictions and propositional constants. Let $\mathcal{C}_W$ be the set of all propositional constants occurring in W. If $X \subseteq \mathcal{C}_W$ and $X \cup W \not\models \perp$ then

$$apply^*_{M_W}(X) = \{q \in \mathcal{C}_W \mid X \cup W \models q\} \tag{5}$$

We can also detect inconsistencies:

$$\perp \in apply^*_{M_W}(X) \text{ iff } X \cup W \models \perp \tag{6}$$

Applied to our example, we get $\langle in(a, b) \rightarrow d\rangle$ and $\langle in(c, d) \rightarrow \perp\rangle$. This network is complete for positive queries, but not for negative ones. E.g. we cannot infer $\neg a$ from b and c. However, for TMS we need direct arguments, whereas classical theorem provers mainly perform refutation proofs. However, the deduction theorem can be used to transform results of refutation proofs into direct arguments:

$$\text{if } W \cup X \cup \{q\} \models \perp \text{ then } W \cup X \models \{\neg q\} \tag{7}$$

In the sequel, we adapt this special case of the deduction theorem to the TMS getting deductive backtracking. In the TMS-network $(\ldots, M_W)$ we miss justifications for negated consequents $\neg c$ of defaults $(a : b_1; \ldots; b_k; c/c) \in D$. In order to detect them, we just add c to a global assumption set $A \subseteq \mathcal{C}_W$. If TMS detects an inconsistency it invokes a backtracking procedure to detect the involved assumption set $N \subseteq A$ by tracing back the applied justifications. Then backtracking selects an assumption c in N and creates a new justification $\langle in(N - \{c\}) \rightarrow \neg c\rangle$ which is justified by 7. In our example, $\langle in(b, c) \rightarrow \neg a\rangle$ may be added after $\perp \in apply^*_{M_W}(\{a, b, c\})$ has been detected. Now, we get a missing justification for a negated consequent. Thus, the set of first-order justifications can dynamically be completed by backtracking.

For this application, it is easy to characterize the needed backtracking justifications. Let C_D be the set of consequents of the defaults in D.

$$\begin{aligned} B_{(D,W)} := \; & \{\langle in(C) \rightarrow \neg c\rangle \mid C \text{ is minimal subset of } C_D \text{ s.t.} \\ & \perp \in apply^*_{M_W}(C \cup \{c\}), c \in C_D\} \end{aligned} \tag{8}$$

Then $M_W \cup B_\Delta$ is complete w.r.t. the relevant negative queries.

Thus, we use dependency-directed backtracking to get justifications for negative literals. Hence, backtracking dynamically completes the set of classical justifications of a network by exploiting the deduction theorem. It is important that it is only applied to the monotonic justifications of the TMS. Thus, contraposition concerning defaults is avoided.

4 TMS as a Propositional Prover

In addition to prove negative literals, we use deductive backtracking to do reasoning by cases and to use TMS as a propositional prover. Since the monotonic part of TMS is similar to the propositional horn clause prover of Dowling and Gallier it is not astonishing that TMS can be viewed as a propositional prover [7]. We especially need this view in section 6 to realize label-backtracking.

Let P be a set of propositional implications of the form $a_1 \wedge \ldots \wedge a_k \supset c$ where the a_i's are literals and c is a literal or $\perp$. We translate them directly into justifications

$$J'_P := \{\langle in(a_1, \ldots, a_k) \rightarrow c \mid a_1 \wedge \ldots \wedge a_k \supset c \in P\} \tag{9}$$

Furthermore, we add some justifications to detect inconsistent cases:

$$J_P := J'_P \cup \{\langle in(q, \neg q) \rightarrow \perp\rangle \mid q \text{ is a constant occurring in } P\} \tag{10}$$

These justifications constitute a network ν_P. Finally, we consider all atoms whose negation occurs as a precondition:

$$Q_P := \{a \mid a_1 \wedge \ldots \wedge \neg a \wedge \ldots \wedge a_k \supset c \in P\} \tag{11}$$

Now, we develop an algorithm that checks consistency of the premise set P. In some aspects, it is similar to a tableau prover because it starts refutation proofs for negative preconditions. However, it records a refutation proof by a new justification for the negation of an involved assumption. The algorithm is supplied with a set J of monotonic justifications and an assumption set A. Both sets characterize the current state and can be changed. The justifications have to be correct w.r.t. the premise set P (i.e. $apply_J^*(X) \subseteq \{q \mid P \cup X \models q\}$). In contrast to tableau provers, we allow a liberal strategy for introducing and eliminating assumptions. Justifications generated by backtracking help to avoid earlier states. We distinguish three cases:

- The current state is incomplete w.r.t. Q_P, i.e. we can neither derive q, nor $\neg q$ for some $q \in Q_P$. Then we start a refutation proof for $\neg q$ by adding q to the current assumption set.

- The current state is consistent and complete. Then we can define a model $\mathcal{M}$ for P by taking $\mathcal{M} \models q$ iff $q \in apply_J^*(A)$ for every propositional constant q. Hence, P is satisfiable.

- The current state is inconsistent since we can derive $\perp$ using its assumptions and justifications. Then dependency-directed backtracking is invoked as in [8] to find a conflict set N by tracing back applied justifications. This set contains all assumptions which are responsible for the inconsistency. If it is empty then P is inconsistent because the set of justifications is correct. Otherwise, dependency-directed backtracking selects an element q of the conflict set N and creates a justifcation $\langle in(N - \{q\}) \rightarrow \neg q\rangle$. Hence, q cannot be chosen again if $N - \{q\}$ has already been selected. However, we can again encounter this contradiction if we choose the assumptions in another ordering. Then a second justification is generated. This time, another assumption will be selected. Hence, termination is guaranteed since after some time all elements of the conflict set are exhausted.

Thus, we use TMS and dependency-directed backtracking to get a sound and complete propositional prover. Below, we list the precise algorithm. It returns *false* if and only if $P \cup A$ is inconsistent. Otherwise, it returns the final set of justifications and the final assumption set:

Algorithm 1 DDB$(J, A) \equiv$

```
1. if $\perp \in apply^*_J(A)$ then
   let $N$ be a conflict set (s.t. $N \subseteq A$, $\perp \in apply^*_J(N)$)
   if $N = \emptyset$ then false else
   select $q$ from $N$ s.t. $\langle in(N - \{q\}) \rightarrow \neg q\rangle \notin J$;
   DDB$(J \cup \{\langle in(N - \{q\}) \rightarrow \neg q\rangle\}, A - \{q\})$ else

2. if $\forall q \in Q_P$: $q$ or $\neg q$ is in $apply^*_J(A)$ then $(J, A)$ else

3. select $q \in Q_P$ s.t. $q, \neg q \notin apply^*_J(A)$
   DDB$(J, A \cup \{q\})$
```

We presume that the algorithm is only supplied with justifications which are in J_P or are obtained by backtracking.

Theorem 4.1 *Let P be a finite set of propositional implications. Let A be a set of propositional constants and J be a set of monotonic justifications that includes J_P and is correct w.r.t. P (i.e. $apply^*_J(X) \subseteq \{q \mid P \cup X \models q\}$ for all sets X of propositional formulas). Then*

1. DDB(J, A) *terminates*

2. DDB(J, A) *returns* false *iff $P \cup A$ is inconsistent*

5 Labelling = Meta Reasoning

As pointed out in section 1, we also want to use backtracking for finding extensions. To enable this we review existing methods for this task. Usually, an extension of a justification network is computed by propagating labels. If a node is IN then it is contained in an extension T, if it is OUT then it is not in T. Propagation rules can complete this partial information about an extension. In this section, we investigate these rules on different levels: (1) We explore the corresponding properties of extensions enabling the labelling steps. (2) We map the labelling problem to a propositional theory stating the dependencies between labels. (3) In the next section, we encode this meta theory into justifications for labels. Such a label-justification corresponds to Doyle's *supporting nodes.* They enable us to identify the sources of an odd loop error and to retract an assumed-label (i.e. to do label-backtracking).

In contrast to Doyle, we use three labels DERIVED, IN, and ¬IN (or OUT). Thus, we can distinguish between deriving a node and specifying that it must be derived. Below, we list the meaning of our labels:

$$\begin{aligned} \text{DERIVED}(q) &\leftrightarrow q \in \mathit{apply}^*_{J,T}(\emptyset) \\ \text{IN}(q) &\leftrightarrow q \in T \\ \neg\text{IN}(q) &\leftrightarrow q \notin T \end{aligned} \tag{12}$$

These labels express partial information about an extension which can be completed. The simplest step is applying a justification to a set of derived nodes. However, we must ensure that all elements of its out-list are not contained in T:

Lemma 5.1 *Let $\nu = (N, J)$ be a justification network and T a subset of N. If $D \subseteq \mathit{apply}^*_{J,T}(\emptyset)$, $O \subseteq N - T$, and $q \in \mathit{apply}_{J,N-O}(D)$ then $q \in \mathit{apply}^*_{J,T}(\emptyset)$*

Proof: $\mathit{apply}_{J,T}(D)$ is an upper bound of $\mathit{apply}_{J,N-O}(D)$ since T is a subset of $N - O$. Hence, $q \in \mathit{apply}_{J,T}(\mathit{apply}^*_{J,T}(\emptyset))$, which is a subset of $\mathit{apply}^*_{J,T}(\emptyset)$. □

If T is an extension it contains derived nodes:

Lemma 5.2 *Let $\nu = (N, J)$ be a justification network and T an extension of ν. If $q \in \mathit{apply}^*_{J,T}(\emptyset)$ then $q \in T$.*

Proof: Since T is an extension $q \in \mathit{apply}^*_{J,T}(\emptyset) = T$. □

Unlike these two properties, the next one is very involved: We want to know when a node is not contained in an extension using the partial information. A first solution is to check whether a node cannot be derived even if all unlabelled nodes are added to T. However, this is not sufficient as the following example demonstrates:

$\langle \rightarrow a\rangle$
$\langle in(a,b) \rightarrow c\rangle \quad \langle in(c), out(d) \rightarrow b\rangle$
$\langle out(b) \rightarrow e\rangle$

Assume that we already know that $a \in T$ and $d \notin T$. Then we cannot proceed with labelling because of the monotonic loop containing b and c. We cannot get a valid justification for one of these nodes even if all unlabelled nodes except the elements of the loops are added to T. Now, we formulate this precisely:

Lemma 5.3 *Let $\nu = (N, J)$ be a justification network and T an extension of ν. If $I \subseteq T$, $O \subseteq N - T$, and there exists a $Q \subseteq N$ s.t. $Q \cap apply_{J,I}((N - O) - Q) = \emptyset$ then $Q \cap T = \emptyset$.*

Proof: We immediately obtain $Q \cap apply_{J,T}(T - Q) = \emptyset$ by putting in a lower bound. $apply_{J,T}(T - Q)$ is a subset of $apply_{J,T}(T)$, which is equal to T since T is an extension. Both results lead to $apply_{J,T}(T - Q) \subseteq T - Q$. Hence, $T - Q$ is closed w.r.t. $apply_{J,T}$. Therefore, $T - Q$ is a superset of $apply^{*}_{J,T}(\emptyset)$, which is equal to T. Thus, we have shown that Q is disjoint to T. □

Later we use two special cases which are easier to handle:

Lemma 5.4 *Let $\nu = (N, J)$ be a justification network and T an extension of ν. If $I \subseteq T$, $O \subseteq N - T$, and there exists a $q \notin apply_{J,I}(N - O)$ then $q \notin T$.*

Lemma 5.5 *Let $\nu = (N, J)$ be a justification network and T an extension of ν. If $I \subseteq T$, $D \subseteq N$, and $apply_{J,I}(D) \subseteq D$ then $(N - D) \cap T = \emptyset$.*

Proof: Choose $Q := N - D$ and $O = \emptyset$. If the condition-part of the lemma is satisfied then $apply_{J,I}(N - Q) \cap Q = \emptyset$. Hence, $(N - D) \cap T = \emptyset$ according to lemma 5.3. □

Above, we showed how partial information about T can be completed if T is an extension. Using the methods above, we can also detect extensions. Let I be a set of IN-labelled nodes that is closed w.r.t. $apply_{J,I}$. After applying the rule in lemma 5.5 to $D := apply^{*}_{J,I}(\emptyset)$ all remaining elements in $N - D$ are labelled with OUT. Thus, we either obtain a contradiction since some elements of I are labelled OUT or I is an extension.

Now, we are ready to encode the dependencies between labels into first-order logic. Since we obtain only ground formulas it is easy to obtain an equivalent propositional theory. We use some abbreviations to handle sets of nodes: We write $\text{DERIVED}^{*}(\{q_1, \ldots, q_k\})$ for the conjunction $\text{DERIVED}(q_1) \wedge \ldots \wedge \text{DERIVED}(q_k)$. Similarly, we take $\text{IN}^{*}(\{q_1, \ldots, q_k\})$ for $\text{IN}(q_1) \wedge \ldots \wedge \text{IN}(q_k)$ and $\text{OUT}^{*}(\{q_1, \ldots, q_k\})$ for $\neg\text{IN}(q_1) \wedge \ldots \wedge \neg\text{IN}(q_k)$. Since we can represent only finite sets the translation works only for finite networks.

Definition 5.1 *Let $\nu = (N, J)$ be a finite justification network. The* label-theory *P_ν of ν consists of propositional implications which are obtained from the following ground instances by uniquely replacing ground atoms by constants:*

1. *for all $I \subseteq N, O \subseteq N, q \in N$ s.t. $q \in apply_{J,N-O}(I)$:*

$$\text{DERIVED}^{*}(I) \wedge \text{OUT}^{*}(O) \supset \text{DERIVED}(q) \tag{13}$$

2. *for all $q \in N$:*

$$\text{DERIVED}(q) \supset \text{IN}(q) \tag{14}$$

3. for all $I \subseteq N, O \subseteq N, q \in N$ s.t. $q \notin apply_{J,I}(N - O)$:

$$\text{IN}^*(I) \land \text{OUT}^*(O) \supset \neg\text{IN}(q) \tag{15}$$

4. for all $I \subseteq N$:

$$\text{IN}^*(I) \supset \text{OUT}^*(N - apply^*_{J,I}(\emptyset)) \tag{16}$$

P_ν contains no other formulas.

It does not matter that the size of P_ν is large compared with the size of ν. For two reasons, we will not explicitly generate and represent all formulas in P_ν. First, we only use this theory to describe and understand an existing labelling algorithm (e.g. Goodwin's system). Second, we can ignore subsumed formulas:

- If $\text{DERIVED}^*(I) \land \text{OUT}^*(O) \supset \text{DERIVED}(q)$ is in the label-theory then q is an element of $apply_{J,N-O}(I)$. Hence there exists an applicable justification $\langle in(I'), out(O') \rightarrow q\rangle \in J$ s.t. $I' \subseteq I$ and $O' \subseteq O$. Hence, $\text{DERIVED}^*(I') \land \text{OUT}^*(O') \supset \text{DERIVED}(q)$ is included in our label-theory and subsumes the first formula.

- Now, consider elements $\text{IN}^*(I) \land \text{OUT}^*(O) \supset \neg\text{IN}(q)$ of the label-theory, which are instances of 15. Then q is not in $apply_{J,I}(N - O)$. Hence, every justification for q is blocked. A justification is blocked if one of its in-list elements is in O or one of its out-list elements is in I. If we collect these elements in two subsets $I' \subseteq I$ and $O' \subseteq O$ we obtain a formula $\text{IN}^*(I') \land \text{OUT}^*(O') \supset \neg\text{IN}(q)$ of our label-theory which subsumes the first formula.

- Finally, consider rule 16. If $I_1 \subseteq I_2$ then $N - apply^*_{J,I_1}(\emptyset) \subseteq N - apply^*_{J,I_2}(\emptyset)$. Hence, if we find a set I satisfying $\text{IN}^*(I)$ and apply the terminate-rule 16 to I then we need not apply it to subsets of I because we have already derived their consequents.

Below, we list the main theorem of this section. A network has an extension if and only if its label-theory is consistent.

Theorem 5.6 *Let $\nu = (N, J)$ be a finite justification network. P_ν is consistent iff ν is coherent.*

Proof: We use models to establish a connection between the label-theory and conditions expressed in terms of *apply*.
($\Rightarrow$): Suppose that P_ν is consistent. Then there exists a model $\mathcal{M}$ of P_ν. Let T be the extension of the predicate IN in $\mathcal{M}$, and D be the extension of DERIVED in $\mathcal{M}$. Hence, $\mathcal{M}$ satisfies $\text{DERIVED}^*(D)$, $\text{IN}^*(T)$, and $\text{OUT}^*(N - T)$. Since $\mathcal{M}$ is a model of P_ν we recognize that the following properties are satisfied:

1. $\mathit{apply}_{J,N-(N-T)}(D) \subseteq D$
2. $D \subseteq T$
3. $N - \mathit{apply}^*_{J,T}(\emptyset) \subseteq N - T$

Then D and T are supersets of $\mathit{apply}^*_{J,T}(\emptyset)$. Due to the third property, T is also a subset of $\mathit{apply}^*_{J,T}(\emptyset)$ and therefore is an extension.
($\Leftarrow$): Suppose that ν has an extension T. Let $\mathcal{M}$ be an interpretation s.t.

- $\mathcal{M} \models \text{DERIVED}(q)$ iff $q \in \mathit{apply}^*_{J,T}(\emptyset)$
- $\mathcal{M} \models \text{IN}(q)$ iff $q \in T$

We immediately observe that

- $\mathcal{M} \models \text{DERIVED}^*(I)$ iff $I \subseteq \mathit{apply}^*_{J,T}(\emptyset)$
- $\mathcal{M} \models \text{IN}^*(I)$ iff $I \subseteq T$
- $\mathcal{M} \models \text{OUT}^*(O)$ iff $T \subseteq N - O$

We want to show that $\mathcal{M}$ is a model of P_ν. For this purpose, we consider the lemmas 5.1, 5.2, 5.4, 5.5. Assume that $\mathcal{M}$ satisfies the precondition of an implication of the form 13, 14, 15, or 16. Then $\mathcal{M}$ satisfies also the conclusion of the implication according to these lemmas. Hence, $\mathcal{M}$ is a model of P_ν. □

From this proof, we obtain an immediate corollary: Extensions of a network correspond to models of the label-theory of the network.

Theorem 5.7 *Let $\nu = (N, J)$ be a justification network. There exists an injective mapping of the set of extensions of ν to the set of models of P_ν.*

6 Label-Backtracking

In the last section, we described dependencies between labels by propositional formulas. In section 4, we translated a propositional theory into justifications and we used backtracking to add missing justification. Hence, we could compute labellings in the same fashion: Translate the *label-theory* P_ν of a justification network $\nu = (N, J)$ into justifications for labels and apply backtracking to complete this set. Such a *label-justification* is an element of J_{P_ν} and therefore monotonic. As an example consider $\langle \mathit{in}(\{\text{IN}(a), \neg\text{IN}(b), \neg\text{IN}(c)\}) \rightarrow \text{OUT}(d)\rangle$. However, it is expensive to generate all elements of P_ν. This theory has only been needed to describe a labelling algorithm. It is easy to generate justifications for labels by such an algorithm. Moreover, TMS already records such meta justifications: If a node is labelled *supporting nodes* for this label are determined and stored.

In the sequel, we refine the algorithm *DDB* of section 4 in order to compute labellings. We get the following peculiarities:

- The label-justifications in J_{P_ν} are implicitly defined by the standard justifications in J.
- Subsumed elements of P_ν can be ignored. It is sufficient to consider examples of formulas $\text{DERIVED}^*(I) \wedge \text{OUT}^*(O) \supset \text{DERIVED}(q)$ where I and O are the in-list and out-list of a justification of q. For formulas of the form 15, we take into account instances $\text{IN}^*(I) \wedge \text{OUT}^*(O) \supset \neg\text{IN}(q)$ where for every justification of q either I contains an out-list element or O contains an in-list element. Other elements are not in I and O.
- Additional label-justifications can be generated by backtracking and are collected in a set B.
- We use the usual TMS-forward-propagator to compute $apply^*_{J_{P_\nu} \cup B}(A)$, and to record the elements of this closure, as well as a set $L \subseteq J_{P_\nu} \cup B$ of the *applied label justifications.* Thus, we get a current state L which is updated in every step. Before the consequents of new label assumptions or justifications are determined an unlabel procedure retracts the consequents of retracted assumptions. The derived labels can be extracted from the label-justifications in L:

$$Cons(L) := \{l \mid \langle in(C) \to l \rangle \in L\}$$

- The set L of applied label-justifications is used to find the conflict set in presence of a label failure. This is easy because every label has at most one justification in L.
- The set Q_{P_ν} of choice points consists of all labels of the form $\text{IN}(q)$ where $q \in N$. Thus, it is supplied implicitly.

Thus, the main work consists in developing a suitable forward-propagator *Label* which is supplied with a set $L \subseteq J_{P_\nu} \cup B$ of applied label justifications and computes the closure $apply^*_{J_{P_\nu} \cup B}(A)$. Since elements of L have been applied earlier we require that $Cons(L) \subseteq apply^*_{J_{P_\nu} \cup B}(A))$. To get such a propagator, we modify usual systems as in [8] or [13] as follows:

- An applied label-justification for $\text{DERIVED}(q)$, $\text{IN}(q)$ or $\neg\text{IN}(q)$ in L is attached to the corresponding node q. For this purpose, we need three properties DERIVED, IN and OUT per node whose values are a label-justification, an indication *assumed* for a label-assumption, or an indication *unknown*. A single justification per label is sufficient. The justification for a label-contradiction is stored separately.

- Following sets are extracted from the current state L:

$$\begin{aligned} I_L &:= \{q \mid \text{IN}(q) \in Cons(L)\} \\ O_L &:= \{q \mid \neg\text{IN}(q) \in Cons(L)\} \\ D_L &:= \{q \mid \text{DERIVED}(q) \in Cons(L)\} \end{aligned} \tag{17}$$

  Since L consists of applied elements of $J_{P_\nu} \cup B$ we get

$$\begin{aligned} I_L &\subseteq \{q \mid \text{IN}(q) \in apply^*_{J_{P_\nu} \cup B}(A)\} \\ O_L &\subseteq \{q \mid \neg\text{IN}(q) \in apply^*_{J_{P_\nu} \cup B}(A)\} =: O \\ D_L &\subseteq \{q \mid \text{DERIVED}(q) \in apply^*_{J_{P_\nu} \cup B}(A)\} \end{aligned} \tag{18}$$

  Since A contains no assumption of the form DERIVED(q) a DERIVED-label is only obtained by applying a justification from J to other nodes having a DERIVED-label. All out-list elements of such a justification must have a $\neg$IN-label. Therefore, we obtain $D_L \subseteq apply^*_{J,N-O}(\emptyset)$ which is needed below.

- The relabel procedure works as usual. If some label assumptions D are retracted from A then the set L is updated. Every justification for an element of $apply^*_L(A) - apply^*_L(A - D)$ is removed from L (i.e. the corresponding property of the TMS-node is replaced by *unknown*).

- Applicable label-justifications are detected by looking for nodes q satisfying a corresponding condition (e.g. $q \in apply_{J,N-O_L}(I_L)$). If such a test is satisfied it is easy to generate the corresponding label-justification.

- A condition $q \in apply_{J,Y}(X)$ is transferred in a more operational form. According to definition 3 it is equivalent to $\exists\langle in(I), out(O) \rightarrow q\rangle \in J : \forall i \in I : i \in X$ and $\forall o \in O : o \notin Y$.

- If a label error is detected the process of computing the closure of $apply_{J_{P_\nu}}$ is stopped. In this case, the supervising algorithm *DDB* finds $\perp$ in the result and needs no other elements of the closure.

- Handling the termination rule is more difficult since we need $apply^*_{J,I_L}(\emptyset)$. For this purpose, we consider the set D_L of nodes having a DERIVED-label. If D_L is closed w.r.t. $apply_{J,I_L}$ it is a superset of $apply^*_{J,I_L}(\emptyset)$. As mentioned above it is a subset of $apply^*_{J,N-O}(\emptyset)$. If no label failure occurs then $I_L \subseteq N - O$. In this case D_L is equal to $apply^*_{J,I_L}(\emptyset)$. Hence, if D_L is closed w.r.t. $apply_{J,I_L}$ we can label every node q not in D with OUT. Some of these nodes may already have an IN-label.

In this case, we get a contradiction because the current labelling does not satisfy the groundedness condition. However, since the newly labelled nodes q get a large label justification $\langle in(I_L) \rightarrow \neg\mathrm{IN}(q)\rangle$ there are a lot of sources of the contradiction and dependency-directed backtracking is here not very helpful in contrast to errors which are caused by odd loops. Nevertheless, we need these large label-justifications in order to realize our label-theory properly. Technically, we could avoid the generation of these justifications and just look for IN-labelled elements which got an ¬IN-label after application of the termination rule.

Thus, we obtain the following forward propagator:

Algorithm 2 Label(L) $\equiv$

```
1. if $\exists q \in N : \mathrm{IN}(q), \neg\mathrm{IN}(q) \in Cons(L)$
   then return $L \cup \{\langle in(\{\mathrm{IN}(q), \neg\mathrm{IN}(q)\}) \rightarrow \bot\rangle\}$ else

2. if $\exists\langle in(C) \rightarrow l\rangle \in B : C \subseteq Cons(L)$ and $l \notin Cons(L)$
   then Label($L \cup \{\langle in(C) \rightarrow l\rangle\}$)

3. if $\exists q \in N : \mathrm{DERIVED}(q) \notin Cons(L)$ and $\exists\langle in(I), out(O) \rightarrow q\rangle \in J$ :
           $\forall i \in I : \mathrm{DERIVED}(i) \in Cons(L)$ and
           $\forall o \in O : \neg\mathrm{IN}(o) \in Cons(L)$
   then let $C$ be a subset of $Cons(L)$ needed to satisfy the condition above;
           Label($L \cup \{\langle in(C) \rightarrow \mathrm{DERIVED}(q)\rangle\}$)

4. if $\exists q \in N : \mathrm{DERIVED}(q) \in Cons(L)$ and $\mathrm{IN}(q) \notin Cons(L)$
   then Label($L \cup \{\langle in(\mathrm{DERIVED}(q)) \rightarrow \mathrm{IN}(q)\rangle\}$)

5. if $\exists q \in N : \neg\mathrm{IN}(q) \notin Cons(L)$ and $\forall\langle in(I), out(O) \rightarrow q\rangle \in J$ :
           $\exists i \in I : \neg\mathrm{IN}(i) \in Cons(L)$ or
           $\exists o \in O : \mathrm{IN}(o) \in Cons(L)$
   then let $C$ be a subset of $Cons(L)$ needed to satisfy the condition above;
           Label($L \cup \{\langle in(C) \rightarrow \neg\mathrm{IN}(q)\rangle\}$)

6. if $\exists q \in N : \mathrm{DERIVED}(q) \notin Cons(L)$ and $\exists\langle in(I), out(O) \rightarrow q\rangle \in J$ :
           $\forall i \in I : \mathrm{DERIVED}(i) \in Cons(L)$ and
           $\forall o \in O : \mathrm{IN}(o) \notin Cons(L)$
   then return $L$

7. else Label($L \cup \{\langle in(I_L) \rightarrow \neg\mathrm{IN}(q)\rangle \mid \mathrm{DERIVED}(q) \notin Cons(L)\}$
```

Finally, we consider an example for label-backtracking. Let's take a network $\nu = (N, J)$ consisting of two even loops, and one odd loop, which depends on the first even loop. Hence, J consists of

$$\begin{array}{lll} \langle out(a) \rightarrow b \rangle & \langle out(b) \rightarrow a \rangle & \\ \langle out(c) \rightarrow d \rangle & \langle out(d) \rightarrow c \rangle & \\ \langle in(b), out(e) \rightarrow f \rangle & \langle out(f) \rightarrow g \rangle & \langle out(g) \rightarrow e \rangle \end{array}$$

We assume that our labelling algorithm introduces the following assumptions during labelling: (1) $\neg\text{IN}(a)$, (2) $\neg\text{IN}(c)$, (3) $\neg\text{IN}(e)$. After that, it detects an odd loop error since it also derives $\text{IN}(e)$. Hence, $\perp \in apply^*_{J_{P_\nu}}(\{\neg\text{IN}(a), \neg\text{IN}(e)\})$. Backtracking generates a new justification for the negation of the latest assumption in this nogood, namely $j_1 \equiv \langle in(\neg\text{IN}(a)) \rightarrow in(e) \rangle$. The last assumption is retracted. Now, we get again a contradiction: $\perp \in apply^*_{J_{P_\nu} \cup \{j_1\}}(\{\neg\text{IN}(a)\})$. Now, backtracking retracts $\neg\text{IN}(a)$, thus skipping the second assumption. The new backtracking justification is $j_2 \equiv \langle \rightarrow \text{IN}(a) \rangle$.

7 Conclusion

In this paper, we elaborated two precise applications of dependency-directed backtracking for Doyle's TMS. Deductive backtracking dynamically completes the set of justifications needed for first-order proofs. Thus, the monotonic part of the TMS becomes an efficient propositional prover. Label-backtracking detects the sources of an odd loop error and retracts an assumed label. Thus, we get an improved TMS that determines an extension whenever one exists. However, some work is still needed to integrate deductive and label backtracking. We also have to elaborate how to achieve incrementality when new justifications are added to the network. In this case, elements of its label-theory are replaced.

As an important prerequisite for label-backtracking, we developed a propositional theory describing the labelling steps and the precise dependencies between labels. In some sense, this theory can be viewed as a semantics for labelling algorithms. In contrast to Brown et. al. [2], we solved the groundedness problem and obtained that coherence is equivalent to consistency of the label-theory. Label-backtracking generates only justifications for labels, not for nodes. Thus, we have a clear distinction between normal and backtracking-justifications as required by Brewka in [1].

As a final remark, we can state that Doyle's TMS already contains the basic ideas and concepts for the work presented in this paper. However, it has not been clear how to combine these techniques appropriately. The main obstacle to more progress has been the lack of the right mathematical formulation, namely the *apply*-operator. This operator simplified a lot of properties and proofs very strikingly.

Acknowledgements

I would like to thank Gerd Brewka, Joachim Hertzberg, and Kurt Konolige for fruitful discussions.

References

[1] Brewka, G., On Minimal Change: A Critique of the Architecture of Nonmonotonic TMS, in: Brewka, G. and Junker, U., Aspects of Non-Monotonic Reasoning, *TASSO Report No 1*, GMD, St. Augustin, Fed. Rep. of Germany, March 1990.

[2] Brown, A., Gaucas, D. and Benanav, D., An Algebraic Foundation for Truth Maintenance, in: *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milano, Italy, 1987, p. 973–980.

[3] Cravo, M.R., Martins, J.P., A Syntactic Approach to Defaults and Belief Revision, in: *DRUMS: Report of the RP1 First Workshop*, Marseille, France, February 1990.

[4] de Kleer, J., An Assumption-based TMS, *Artificial Intelligence* **28** (1986), p. 127–162.

[5] de Kleer, J., Extending the ATMS, *Artificial Intelligence* **28** (1986), p. 163-196.

[6] de Kleer, J., A General Labelling Algorithm for Assumption-based Truth Maintenance, in: *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988.

[7] Dowling, W.F. and Gallier, J.H., Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae, *Journal of Logic Programming* **3** (1984), p. 267–284.

[8] Doyle, J., A Truth Maintenance System, *Artificial Intelligence* **12** (1979), p. 231–272.

[9] Doyle, J., Some Theories of Reasoned Assumptions: An Essay in Rational Psychology, Carnegie Mellon University, CMU CS-83-125, 1983.

[10] Doyle, J., The Ins and Outs of Reason Maintenance, in: *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, F.R.G., 1983, p. 349–351.

[11] Dressler, O., An Extended Basic ATMS, in: Reinfrank, M., de Kleer, J., Ginsberg, M.L., Sandewell, E. (Eds.), Non-Monotonic Reasoning, 2nd International Workshop, Springer LNCS 346, 1989, p. 143–163.

[12] Dressler, O., Problem Solving with the NM-ATMS, in: *Proceedings of the Ninth European Conference on Artificial Intelligence*, Stockholm, Sweden, 1990, p. 253–258.

[13] Goodwin, J.W., A Theory and System for Non-Monotonic Reasoning, *Ph.D. Dissertation*, University of Linköping, Linköping, Sweden, 1987.

[14] Junker, U., A Correct Non-Monotonic ATMS, in: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989, p. 1049–1054.

[15] Junker, U. and Konolige, K., Computing the Extensions of Autoepistemic and Default Logics with a Truth Maintenance System, in: *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, 1988, p. 278–283.

[16] Konolige, K., On the Relation between Default and Autoepistemic Logic, *Artificial Intelligence* **35** (1988), p. 343–382.

[17] Martins, J.P and Shapiro, S.C., A Model for Belief Revision, *Artificial Intelligence* **35** (1988), p. 25–80.

[18] Moore, R.C., Semantical Considerations on Nonmonotonic Logic, *Artificial Intelligence* **25** (1985), p. 75–94.

[19] Reinfrank, M., Dressler, O. and Brewka, G., On the Relation between Truth Maintenance and Autoepistemic Logic, in: *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, MI, 1989, p. 1206–1212.

[20] Reiter, R., A Logic for Default Reasoning, *Artificial Intelligence* **13** (1980), p. 81–132.

[21] Russinoff, D.M., An Algorithm for Truth Maintenance, *MCC Technical Report No. AI-062-85*, MCC, Austin, Texas, 1985.