A Correct Non-Monotonic ATMS

Ulrich Junker
GMD
Hybrid Inference Systems Research Group
Postfach 1240
5205 Sankt Augustin
Fed. Rep. of Germany

Abstract

In this paper, we investigate technical methods to deal with *exceptions, inconsistencies,* and *ambiguity.* Existing reason maintenance systems are only suitable for some of these problems. *Doyle's TMS* handles exceptions properly, but gets in trouble with non-monotonic odd or even loops. On the other hand, *de Kleer's ATMS* produces too many contexts if there are *exceptions of exceptions.* Therefore, we present a *hybrid reason maintenance system* that benefits from the advantages of the ATMS and TMS. First of all, we use Reiter's default logic as a specification for this system. Second, we develop a *criterion* that decides which assumption sets are valid. Then, we extend the ATMS by a *test* algorithm that takes account of exceptions and obeys this criterion. We *verify* the resulting system using default logic. Finally, we discuss some problems involved with special techniques for resolving inconsistencies (e.g. *TMS-like backtracking).*

1. Introduction

Reason maintenance systems can be applied to capture special phenomena occurring in non-monotonic reasoning. E.g. *Doyle's TMS* [Doyle, 1979] handles *exceptions* properly. On the other hand, the multiple context facility of *de Kleer's ATMS* [de Kleer, 1986a] may be used to deal with *ambiguity* in the presence of *inconsistencies.* However, none of these systems is suitable for non-monotonic reasoning in general. Let us consider some examples that demonstrate the advantages and limits of reason maintenance techniques. For this purpose we use *Reiter's default logic* (DL) [Reiter, 1980] because it is powerful enough to illustrate all problems. On the other hand, it is simple enough to relate its concepts and reason maintenance constructs[1] .

Example 1: *(the familiar Nixon example)*
As usual, quakers typically are doves and republicans normally are hawks. Nobody can be a hawk and a dove, but Nixon is a quaker and a republican. Thus, we get two (simplified) defaults and some premises:

REPUBLICAN,	(REPUBLICAN : MHAWK / HAWK),
QUAKER,	(QUAKER : MDOVE / DOVE),
	DOVE & HAWK $\supset$ FALSE

There are two extensions of this default theory: One contains DOVE and --HAWK, the other one -DOVE and HAWK. First, we want to determine these extensions using the ATMS. Following [Dressier, 1988a] and [Junker, 1988], we can translate a default (a: Mb/ b) into a monotonic justification a, $\mathrm{out}(\neg b) \rightarrow$ b and an assumption out(--b). DL premises are also ATMS premises. Furthermore, assume that modus ponens is applied and that every inference step is protocoled by an ATMS justification. Then, we obtain:

justifications:	REPUBLICAN, out(¬HAWK) $\rightarrow$ HAWK
	QUAKER, out(¬DOVE) $\rightarrow$ DOVE
	DOVE, HAWK $\rightarrow$ FALSE
assumptions:	out(¬DOVE), out(¬HAWK)
nogoods:	{{out(¬DOVE), out(¬HAWK)}}
maximal consistent assumption sets:	{out(¬DOVE)}, {out(¬HAWK)}

In this case, every maximal consistent assumption set corresponds to an extension (and vice versa). *Therefore, ATMS may be used for normal defaults (whose prerequisites are theorems of premises).* In contrast to Doyle's TMS it finds all extensions. However, there are examples where Doyle's system is superior to ATMS:

Example 2: *(working example)*
On workdays, Peter must normally go to work. This is not true, if he has an excuse. If he is ill he normally has an excuse. However, his employer does not accept this if Peter has only caught a cold. Hence, there is an *exception of an exception** Now, assume that Peter is ill on a workday. What happens? Again, we formalize this story using simplified defaults:

WORKDAY,	(WORKDAY: M-EXCUSE / WORK),
ILL,	(ILL: M-TCOLD / EXCUSE)
COLD $\supset$ ILL	

There exists only one extension that contains EXCUSE, but not WORK. If we use Doyle's TMS to compute this extension, we obtain two non-monotonic justifications:

(SL WORK (WORKDAY) (EXCUSE)),
(SL EXCUSE (ILL) (COLD))

[1] We write $(a: Mb_1;\ldots;Mb_k/c)$ for Reiter's defaults, $(a_1,\ldots,a_k/c)$ for inference rules, $a_1,\ldots,a_k \rightarrow c$ for monotonic justifications, and (SL c $(a_1 \ldots a_k)$ $(o_1 \ldots o_j)$) for Doyle's non-monotonic justifications.

WORKDAY and ILL are labelled with IN because they are premises. COLD is labelled with OUT because it has no justification at all. Then, EXCUSE gets IN and the only justification for WORK is invalid. Therefore, WORK is labelled with OUT just as we expected. Figure 1 shows the complete labelling using Goodwin's notation [Goodwin, 1987].

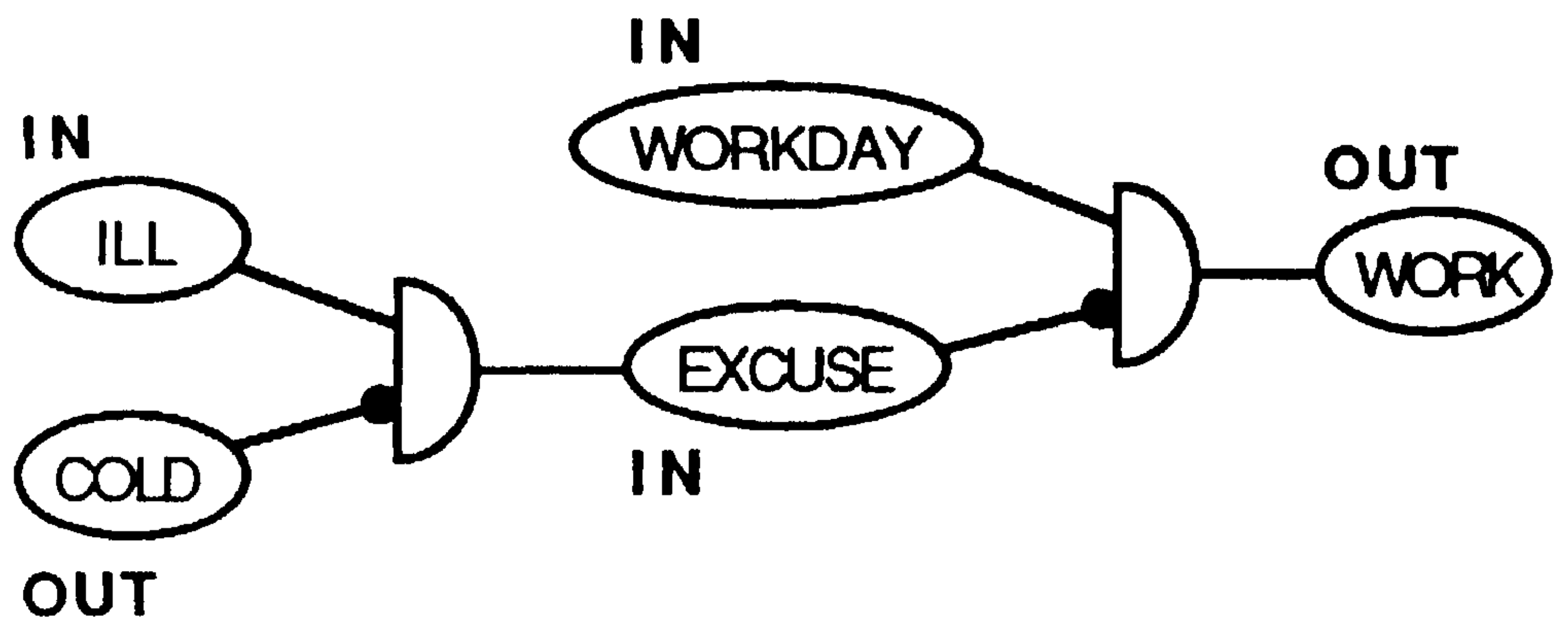

Figure 1: Complete labelling for the working example

However, if we apply the ATMS we get in trouble. In addition to example 1, we use extra consistency rules to ensure that no context contains out(x) and x:

justifications:	WORKDAY, out(EXCUSE) -> WORK
	ILL, out(COLD) -> EXCUSE
	EXCUSE, OUt(EXCUSE) -> FALSE
	COLD, out(COLD) -> FALSE
assumptions:	OUt(COLD), out(EXCUSE)
label of COLD:	0
label of EXCUSE:	{(out(COLD)}}
label of WORK:	{{out(EXCUSE))}
no goods:	{(out(COLD), out(EXCUSE))}
maximal consistent assumption sets:	{out(COLD)}, {out(EXCUSE)}

Astonishingly, there are two maximal consistent assumption sets. One, namely (out(COLD)}, corresponds to the extension mentioned above. However, also the set (out(EXCUSE)} which infers WORK is consistent! The ATMS detects the conflict between both conflicing defaults, but it does not know anything about priorities between interacting defaults. Expressions like out(COLD) are simple assumptions. Reasons for such expressions are ignored. The same problem arises if we apply de Kleer's original encoding of non-monotonic justifications [de Kleer, 1986b]. The TMS, however, uses a special rule to label a node with out:

If all justifications of a node x are invalid label x with OUT.

To sum up, both reason maintenance systems are restricted to special cases: ATMS is suitable for normal defaults, but gets in trouble with exceptions of exceptions. Doyle's TMS works correctly if the default theory is stratified or acyclic (i.e. the justification network contains no non-monotonic odd or even loop). Then the existence of a unique extension is guaranteed. If there are even loops there may be different extensions, but the TMS finds only one. If there are odd loops a TMS may enter an endless loop [Doyle, 1979] or stop without definite result [Goodwin, 1987].

In the sequel, we develop a *hybrid reason maintenance system* that benefits from advantages of the ATMS and the TMS and overcomes their limitations. We use the ATMS as a basic system that records the applications of non-monotonic or monotonic rules in different contexts. However, not all assumption sets occurring in ATMS labels are valid in the sense that they describe a subset of any extension. Therefore, a separate test algorithm is needed that takes account of exceptions. For this purpose we adapt Doyle's Out-labelling rule mentioned above. We also want to verify the resulting system. Therefore we need a suitable *specification of non-monotonic reasoning.* Again, we choose Reiter's default logic.

2. Theoretical Considerations

In this section, we develop the theoretical framework for realizing non-monotonic reasoning using the ATMS. First of all, we consider Reiter's definition of a default logic extension.

Definition 1: *Let A = (D,W) be a closed default theory (D is a set of defaults and W is a set of first-order formulas). An extension of A is a fixpoint of an operator T that is defined as follows: Let S be an arbitrary set of first-order formulas. Then T(S) is the smallest set such that*

(1) r(S) contains W
(2) T(S) is closed with respect to first order derivability
(3) if (a:Mb1;...;Mb_n/c) ∈ D,a ∈ T(*S*), *-b_i∉ S*
then c e T(S).

Can we determine T(S) using a monotonic reasoner for multiple contexts? At least, the definition of *T* resembles the usual definition of a closure of a monotonic inference system: Let I = (L, AX, R) be an arbitrary *monotonic inference system* (L is a formal language, AX is a set of axioms, and R contains inference rules). The closure Th1 of a subset P of L can be defined by: Th1(P) is the smallest set such that:

(4) Th1(P) contains AX and P
(5) if (a1.....a_k/c) ∈ R and a_t ∈ Th1(P) then c ∈ Th1(P).

Both definitions mainly differ in the condition -b_i∉ S used in (3). In the monotonic case, we don't ask whether a formula is not contained in a set. Fortunately, this unusual condition only refers to S, not to T(S). We can introduce additional expressions of the form out(—.bj) for -b_i∉ S. Furthermore, we replace S by a corresponding set A(S) containing such out-expressions. Then we ask out(-b_i) ∈ Th_1(A(S)) instead of -b_i∉ S. Let O be the set of all *out-expressions* needed for any default of D. Then A(S) must contain out(x) if and only if S does not contain x:

A(S) := {out(x) ∈ O/x ∉ S}

There is another difference between both definitions. (2) requires *completeness* with respect to first-order derivability. Therefore, we need a complete and sound inference system for first-order derivability. Later, we will restrict the first-order language to horn clauses and consider only derivability of atomic formulas. Then modus ponens will be sufficient to achieve (2).

Now, let L be a first-order language and A = (D,W) be a closed default theory using L. Furthermore, let LI = (L, AX, R) be a complete inference system for first-order derivability in L. We extend this inference system to obtain a tool for determining T(S). We add the premises of W as axioms and translate every default of D into an inference rule as follows. We apply a default $(a:Mb_1,...;Mb_k/c)$ and derive c if we infer a, $out(-b_1),...,out(--.b_k)$:

MD := $\{(a,out(-b1),...,out(-b_n)/c) \quad / \quad (a:Mb_1;...;Mb_k/c) \in D)$

Then we obtain a new inference system *D1* ;= *(L U 0, AX ∪ W, R ∪ MD).* Because we are not interested in derivation of out-expressions we modify the *closure* Th_{DI} of DI slightly: Let A be a set of out-expressions. Then *Th(A):=* $Th_{DI}(A)$ *n L.* Now, we can formulate the first result:

Theorem 1: *Let S be a set of first-order formulas. Then US) = Th(A(S))*

Proof: in [Junker, 1988]

Thus, we are able to determine T(S) using a monotonic reasoner if A(S) is specified. However, we are only interested in fixpoints of T.is there a simple criterion that decides which assumption sets correspond to extensions? A fixpoint S satisfies: S = T(S) = Th(A(S)). Then an assumption out(x) is contained in A(S) if and only if x is not contained in S = Th(A(S)). Therefore we define:

Definition 2: *A subset A of O is an extension base iff for every out(x) € 0 either out(x) € A or x € Th(A) (not both!).*

We already know that this criterion is a necessary condition for an assumption set obtained from a fixpoint S. Is it also a sufficient condition for a fixpoint ? Assume that A is an extension base. Then out(x) is contained in A if and only if x ∉ Th(A) if and only if out(x) E A(Th(A)). Since every out-expression satisfies this fact the sets A and A(Th(A)) arc equal. Hence, Th(A) is an extension because of Th(A) = Th(A(Th(A))) = r(Th(A)). Now, we can summarize:

Theorem 2: *If E is an extension then A(E) is an extension base. If A is an extension base then Th(A) is an extension.*

We illustrate this result using example 2. We get four cases because there are two out-expressions, namely out(COLD), out(EXCUSE):

$A = \emptyset$:	$Th(A) = \{WORKDAY, ILL, COLD \supset ILL\}$
$A = \{out(EXCUSE)\}$:	$Th(A) = Th(\emptyset) \cup \{WORK\}$
$A = \{out(COLD)\}$:	$Th(A) = Th(\emptyset) \cup \{EXCUSE\}$
$A = \{out(EXCUSE), out(COLD)\}$:	$Th(A) = Th(\emptyset) \cup \{EXCUSE, WORK\}$

In the first and second case, we miss out(COLD) and don't infer COLD. In the last case, we assumed out(EXCUSE) and inferred EXCUSE. Thus, we only obtain an extension base in the third case, where out(COLD) is assumed and EXCUSE is inferred.

3. Which defaults & formulas are needed ?

Because we only want to develop a reason maintenance system we introduce some restrictions concerning the defaults and first-order formulas. First of all, we assume that the number of defaults and premises *is finite.* In [Doyle, 19791, reason maintenance nodes are viewed as propositional constants. We interpret monotonic justifications of the form (SL c $(a_1 \ldots a_k)$ ()) as propositional implications (or horn clauses). Therefore, we restrict the language L to *propositional horn clauses.* We use the special symbol FALSE as a consequent of horn clauses without positive literal. In section 4 and 5, we consider only defaults of the form $(a:M\text{-}b_1;...;M\text{-}b_k/c)$ where $a,b_1,...,b_k,c$ are propositional constants. Such defaults correspond to non-monotonic justifications of the form (SL c (a) $(b_1 \ldots b_k)$) (cf. [Brewka, 1989]).

Now, it is easy to satisfy the completeness requirement of (2). We are only interested in the derivation of propositional constants. There are two cases: If an assumption set is consistent we need only *generalized modus ponens* to derive any constant. If an assumption set is inconsistent (i.e. FALSE is derivable) then any formula can be derived. Modus ponens is also sufficient to detect inconsistencies. Besides the set MD, we take only account of inference rules of the form

$(a_1 \ldots a_k, a_1 \& \ldots \& a_k \supset c/c)$

where $a_1,...a_k$ are constants and c is a constant or FALSE.

4, Techniques for Multiple Contexts

The results of section 2 are still independent from the ATMS. If we want to determine extensions we must consider a lot of overlapping contexts. Therefore, we can benefit from the facilities of the ATMS. First we translate the concepts of section 2 into the ATMS terminology:

Every premise of the inference system DI is also a premise of ATMS. An out-expression out(x) is added as an assumption to the ATMS. Furthermore, every inference rule $(a_1,...a_k/ c)$ mentioned in the paragraph above is transformed into a justification $a_1,...a_k$ —> c. Now, we can check whether a formula x is contained in Th(A) by inspecting its ATMS-label which is denoted by LABEL(x):

x € Th(A) iff LADEL(x) contains a subset of A or
A is inconsistent

Furthermore, we can check the inconsistency of an assumption set by inspecting the set NOGOODS that is maintained by the ATMS and contains all minimal inconsistent assumption sets:

A is inconsistent iff NOGOODS contains a subset of A

However, it is still possible that x is inferred from an assumption set containing out(x). Consider the set (out(EXCUSE), out(COLD)} of the working example where EXCUSE Th(A) and out(EXCUSE) € A. According to definition 2, this set is no extension base. Even if the default theory is extended by new defaults or premises EXCUSE € Th(A) remains true. Such a set is completely useless. We can use the consistency mechanism of the ATMS to get rid of such sets. Following [Dressier, 1988a], we introduce additional *constraints* of the form

x, out(x) —> FALSE

for every out-expression. Now, assume that a set A containing out(x) infers x. Then it infers FALSE and is handled as a nogood. These additional nogoods do not represent logical inconsistencies, but are also used to get rid of assumption sets that can never become extension bases. Therefore, we call them *pruning nogoods.*

With respect to this extended form of inconsistency, extension bases are maximal consistent assumption sets of the ATMS. If we extend an extension base A by a further assumption out(y) $\notin$ A then y $\in$ Th(A $\cup$ {out(y)}) because of y $\in$ Th(A). Normally, an extension base is consistent. Only if FALSE $\in$ Th($\emptyset$)) then there is an inconsistent extension base, namely the empty set:

Lemma 1: *If A is a non empty extension base then A is a maximal consistent set of out-expressions. An inconsistent set A is an extension base if and only if FALSE $\in$ Th($\emptyset$) and $\emptyset$.*

Example 2 shows that there are still maximal consistent sets that are no extension bases (e.g. {out(EXCUSE)}). Therefore, the ATMS techniques are not sufficient

5. Techniques for Exceptions

In a lot of cases, we are not interested in complete extensions, but in a particular formula x. We want to know the extensions containing x. For this purpose, we can inspect the ATMS label of x. It contains minimal sets of out-expressions that infer x, but we don't know whether these sets are contained in any extension base. In example 2, we are interested in the formula WORK the label of which consists of the set (out(EXCUSE)}. Then we ask whether this set is valid in the following sense:

Definition 3: *A set A of out-expressions is valid if A is contained in any extension base.*

How can we check this property? We try to add further out-expressions to A. For every out-expression out(x), we choose out(x) or test whether x can be inferred afterwards. Thus, we obtain a binary search tree and apply a backtracking procedure. In the sequel, we ignore the special case that the empty set is inconsistent. Hence, we drop inconsistent supersets of A. Then we obtain a first version of a test algorithm. We assume that the elements of O are ordered in a sequence out(x1), ...out(x_k).

Algorithm SLOWTEST

```
input:    an assumption set A and an optional number i
          (default value: 1)
output:   true, if there is a consistent extension base E
              that  {out(x_i),....,out(x_k)} ∪ A ⊇ E ⊇ A
          false, otherwise

ij NOGOODS contains a subset of A then false else
if i > k and LABEL(x) contains a subset of A
    for all out(x) ∈ O - A then true else
if i> k then false else
if SLOWTEST(, ∪ {out(x_i)}, i+]) then true
    else SLOWTEST(A, i+1).
```

This algorithm does not handle exceptions efficiently. Consider a long chain consisting of (:M-A$_i$ / Ai+1) where i=l,...,k. At worst, SLOWTEST needs 0(2^k) steps, while Doyle's TMS finishes after k steps. How does TMS gain this efficiency ? In contrast to SLOWTEST, TMS exploits dependencies between out-assumptions. There are circumstances, where a formula x cannot be derived even if out-assumptions are added. Then we must choose out(x). In section 1, we showed that TMS uses a rule to determine an out-label of a node. We need a corresponding argument for the ATMS-based system.

Can we check whether xi is inferred by a superset of A or not? If A already infers XI we cannot choose out(xi) any more. Otherwise, XI$\notin$ Th(A). Now, assume that every out-assumption out(xj) occurring in LABEL(XJ) has been considered before out(x_i) (i.e. j < i). Then the remaining out-assumptions out(x|),...,out(x^) do not contribute to any proof of xi. We summarize this fact after introducing another term: Let PRED(x) := (out(y) $\in$ O I $\exists$ A $\in$ LABEL(x): out(y) $\in$ A} be the set of all out-expressions that can be used to prove x. We call members of PRED(x) *predecessors* of out(x).

Lemma 2: *Let A and PRED(xi) be subsets of {out(xj), ...,out(xi-1)}. Let B be a consistent superset of A and a subset of A U {out(xi),...,out(x0}. Then B infers xi; if and only if A infers Xi.*

If the prerequisites of lemma 2 are satisfied we can easily check whether we must add or drop out(x). Thus, we can improve SLOWTEST if we order the set of out-expressions in a useful sequence. In example 2, we start with out(COLD) because LABEL(COLD) and PRED(COLD) are empty. Because out(COLD) is the only predecessor of out(EXCUSE) we add it and get a complete sequence. Things are not so easy if there are non-monotonic loops. E.g. consider a non-trivial odd loop consisting of three defaults:

(:M-iA/B), (:M-B/C), (:M-C/A)

Then out(A) is a predecessor of out(B) which is a predecessor of out(C). However, out(C) is predecessor of out(A). Therefore, we must cut off this loop, choose any assumption e.g. out(A) and investigate two cases. We can generalize this idea:

Definition 4: *A cut Cut is a minimal subset of O (with respect to set inclusion) such that the elements of O can be ordered in a sequence out(x1),...,out(x0 that satisfies for all i = 1,...,k: out(xi) € Cut or PRED(xi) is a subset of {out(xj). . . . .out(x^j)}.*

We can compute such a cut and a sequence using an algorithm with quadratic time complexity. We only determine a cut and a sequence if the ATMS has finished. Afterwards, we can apply a fast modification of SLOWTEST to any assumption set of interest.

Algorithm TEST

```
input:    an assumption set A and an optional number i
          (default value: 1)
output:   true, if there is a consistent extension base E
              that  {out(xi),...,out(x0} ∪ A ⊇ E ⊇ A
          false, otherwise
```

```
if NOGOODS contains a subset of A then false else
if i > k and LABEL(x) contains a subset of A
    for all out(x) ∈ Cut - A then true else
if i > k then false else
if out(x_i) ∈ A or LABEL(x_i) contains a subset of A
    then TEST(A, i+1) else
if out(x_i) ∉ Cut then TEST(A ∪ {out(x_i)}, i+1) else
if TEST(A ∪ {out(x_i)}, i+1) then true else TEST(A, i+1).
```

The improvements of TEST are explained briefly: If out(x) is already contained in A or x is inferred from A we have no choice. Otherwise, if out(x) is no cut assumption, out(x) must be added to A because x cannot derived from A (cf. lemma 2). Only for cut assumptions, we have to consider both cases. If we don't choose a cut assumption out(x) we must check afterwards whether x Gan be inferred.

Now, we apply TEST to the working example:

input from	LABEL(COLD) = ∅,
ATMS:	LABEL(EXCUSE) = {{out(COLD)}},
	NOGOODS = {{out(EXCUSE), out(COLD)}}
sequence:	out(COLD), out(EXCUSE)
cut:	∅

First, we try TEST({out(EXCUSE)}). Since COLD cannot be proved the algorithm adds out(COLD) and calls TEST({out(EXCUSE), out(COLD)}, 2). Then the first argument is a nogood and TEST returns false. Hence, the single set of the label of WORK is invalid. However, calling TEST({out(COLD)}) results in true: Because out(COLD) is already contained in A, TEST({out(COLD)}, 2) is called immediately. Then we can infer EXCUSE and must not add out(EXCUSE). As there is no cut assumption TEST({out(COLD)}, 3) yields true.

In contrast to this stratified example a cut assumption, say out(A), is introduced for the odd loop mentioned above. Then we obtain the sequence out(A), out(B), out(C). If we call TEST(0) we first choose out(A). Then we can proof B and must not add out(B). Hence, we cannot prove c and must add out(C) yielding a nogood, namely (out(A), out(B)}. Therefore, we go back and drop out(A). Then we must add out(B) and drop out(C). However, we cannot prove A using (out(B)}. Hence, TEST(0) results in false. TEST has detected that there is no extension at all.

In the worst case, the complexity of TEST is $0(n * 2^m)$ where n is the number of all out-assumptions and m is the number of cut assumptions or choice points. If there are no non-monotonic loops we can avoid cut assumptions and the complexity reduces to O(n). Otherwise, TEST explores different alternatives if it chooses a wrong assumption of an even loop or runs into an odd loop. Efficiency is strongly influenced by the number of cut members and their ordering.

TEST is a non-incremental algorithm because it presupposes that all proofs for some formulas have been found. That's not so bad because all monotonic inference steps are performed by an incremental system, namely the ATMS. Furthermore, we should only call TEST after the problem solver added all justifications.

6. Dealing with Inconsistencies

We are still not able to handle the Nixon example because we restricted our default language in section 3. What is the special problem of the Nixon example ? Nixon is a dove and a hawk per default. However, the conjunction of both properties implies a *logical inconsistency.* How can we resolve this inconsistency? In default logic, we just require that the consequent c of a default is consistent by writing (a: M-ib; Mc / c). Such an expression is similar to a *semi-normal default.* Let (aj: M-ity; Mc, / Cj) be some defaults (i=l,...,n) whose consequents imply an inconsistency as in the Nixon example. Assume that 1= denotes first-order derivability. Then we can select an arbitrary c^ and apply the deduction theorem (as in [de Kleer, 1988]):

(*) *if* $\{c_1,...,c_n\}$ |= *FALSE then* $\{c_1,...,c_n\} - \{c_k\}$ |= $\neg c_k$

Then (ak: M-b_k; Mck / ck) is blocked if we apply the other defaults. In this case, there are no implicit priorities between the defaults. Thus, a logical inconsistency can lead to multiple extensions.

Now, we can use defaults of the form (a:M—>b_1;...;M—b_m; Mc / c) to resolve logical inconsistencies. If we translate such a default according to section 2 we obtain an out-expression out(—.c) and must find all proofs for a negative constant -ic. Are there special techniques to facilitate this task? Can we exploit (*) and start a refutation proof for -ck?

For this purpose, Doyle's TMS uses *dependency-directed backtracking.* If TMS detects an inconsistency by labelling FALSE with IN it traces back and finds the non-monotonic justifications underlying the contradiction. As above, it selects an arbitrary justification (SL ck (a) (*b1* ... b_m)) and any bj. Then it justifies bj with (SL bj (c1 ... ck-1] ck+1 ...c_n) (b1 ... b_{j-1} b_{j+1} ... b_m)). Hence, TMS justifies bj instead of $\neg c_k$! However, we could slightly change the backtracking procedure if we want to achieve (*).

Can we adapt dependency-directed backtracking to assumption-based techniques? Dressier proposes the following meta rule that inspects nogoods:

(**) *if* $\{out(x_1),...,out(x_n)\}$ *is a nogood*
then $\{out(x_1),...,out(x_n)\} - \{out(x_k)\}$ *justifies* x_k

Compared with (*), there are some important differences:

Because rule (**) inspects nogoods it refers to out-expressions instead of consequents of non-monotonic justifications.

Furthermore, we apply (**) to all nogoods while (*) *requires first-order-derivability.* In our RMS, the translated default rules and constraints of the form x, out(x) → FALSE do not correspond to first-order inference steps. Especially, if we apply (**) to pruning nogoods our DL-specification is violated. Consider again the pruning nogood (out(EXCUSE), out(COLD)} of the working example (cf. section 4). Then, rule (**) adds (out(EXCUSE)} to the label of COLD and (out(EXCUSE)} becomes another extension base. This is wrong because no DL-extension contains COLD. Therefore, we must not apply (**) to pruning nogoods.

However, if we restrict (**) to logical nogoods we get another problem. The ATMS does not detect every logical nogood:

(:MA / A), (:M-A /B), (:MC /C), A & B & C $\supset$ FALSE

Here, we obtain a pruning nogood {out(-A), out(A)} and a logical nogood {out(-A), out(A), out(-C)}. Because the ATMS stores only minimal nogoods it ignores the second one. Thus, we are not able to apply (**) to (out(--.A), out(A),out(-.C)},

Finally, rule (**) implicitly applies defaults in 'backward' direction. In the following example, it infers -A from out(B) using the nogood (out(-A), out(B)}:

(:MA / A),   (A :M-.B / FALSE)

Because of these problems, we do not get a sound method that finds refutation proofs of -x if we inspect nogoods consisting of out-assumptions. If we want to prove -x directly we need further inference rules (and justifications). For example, we can use *contraposition* rules:

(—c, $a_{1'}$ ..., aj-1, aj+]....a_k, a1&...&ak$\supset$ c I—ajk)
(a1, ..., aj-1, $a_{+j+1,}$,..., ak, a1&....&a^ $\supset$ FALSE I —aj)

where c,a],...,ak are constants. With these prerequisites we obtain the missing labels in the Nixon example:

LABEL(-DOVE) = {(out(4L\WK)})
LABEL(-HAWK) = {{out(-DOVE))}

This approach works for examples where unit clause resolution is complete w.r.t. the derivation of negated formulas.

7. Related Work

We started this work by examinating the ideas of O. Dressier. In [Dressier, 1988a], he already proposed how to encode defaults using monotonic justifications and out-assumptions. Furthermore, he introduced the meta rule to resolve inconsistencies. However, he did not verify his approach using default logic and did not specify a semantics for the meta rule. After the author had detected the problem imposed by exceptions and presented a first solution, O. Dressier developed an own test algorithm to deal with exceptions of exceptions. In contrast to our test, it inspects justifications instead of labels and uses a backward strategy (cf. [Dressier, 1988b]). This algorithm is yet not verified and a precise comparison is a goal of further discussions.

[de Kleer and Reiter, 1987] and [de Kleer, 1988] generalize ATMS by propositional clauses, but do not capture non-monotonic justifications. Brown et. al. [1987] translate justifications into boolean equations where non-monotonic antecedents are handled by the complement operator. Their approach is also non-incremental. If we add a new justification for x we must exchange the equation of x. Furthermore, they need extra effort to find well-founded (or grounded) solutions.

8. Conclusion

We developed a correct non-monotonic reason maintenance system that is able to deal with exceptions, inconsistencies, and multiple extensions. It overcomes the limits of Doyle's TMS and ATMS. TMS only finds a single extension and gets in trouble with odd loops. ATMS produces too many extensions if there are exceptions of exceptions. Therefore, we extended the ATMS by an additional test algorithm which uses an efficient technique for exception handling. Its complexity is nearly linear in a lot of examples. We implemented this algorithm in Common Lisp and used the hybrid RMS to realize a simple default prover.

We verified our hybrid reason maintenance system using Reiter's default logic (DL). To enable this task, we related DL concepts to the concepts of monotonic inference systems and replaced the fixpoint criterion by a simpler criterion for assumption sets.

Finally we discussed some problems caused by special techniques for contradiction handling. We proposed to resolve logical inconsistencies by semi-normal defaults. Then the reason maintenance system must be able to derive negated formulas.

Acknowledgements

Especially, I am grateful to Gerd Brewka and Hans Voss for their support and encouragement. Furthermore, I would like to thank Johan de Kleer, Paul Morris, and Oskar Dressier who spent time for discussion.

References

[Brewka, 1989] G. Brewka, Nonmonotonic Reasoning: From Theoretical Foundation Towards Efficient Computation, *Dissertation,* University of Hamburg

[Brown ct. al., 1987] A. Brown, D. Gaucas, D. Benanav, An Algebraic Foundation for Truth Maintenance, Proc. *IJCA1 87*

[de Kleer, 1986a] J. de Kleer, An Assumption-based TMS, *Artificial Intelligence* 28

[de Kleer, 1986b] J. de Kleer, Extending the ATMS, *Artificial Intelligence* 28

[de Kleer, 1988] A General Labelling Algorithm for Assumption-based Truth Maintenance, *Proc. AAAI 88*

[Doyle, 1979] J. Doyle, A Truth Maintenance System, *Artificial Intelligence* 12

[Dressier, 1988a] O. Dressier, Extending the Basic ATMS, *Proc. ECAI 88*

[Dressier, 1888b] O. Dressier, An Extended Basic ATMS, *Proceedings of the 2nd International Workshop on Non-Monotonic Reasoning,* Springer LNCS 346

[Goodwin, 1987] J. Goodwin, A Theory and System for Non-Monotonic Reasoning, *PhD Dissertion,* University of Linkoping

[Junker, 1988] U. Junker, Reasoning in Multiple Contexts, *Arbeitspapiere der GMD 334*

[Reiter, 1980] R. Reiter, A Logic for Default Reasoning, *Artificial Intelligence* 13

[Reiter and de Kleer, 1987] J. de Kleer, R. Reiter, Foundations of Assumption-based Truth Maintenance Systems, *Proc. AAAI 87*