Tweety - Still Flying

Some Remarks on Abnormal Birds, Applicable Rules and a Default Prover

Gerhard Brewka

Gesellschaft für Mathematik und Datenverarbeitung
Forschungsgruppe Expertensysteme
Postfach 12 40
D 5205 Sankt Augustin, Federal Republic of Germany

ABSTRACT

This paper describes FAULTY, a default prover for a decidable subset of predicate calculus. FAULTY is based on McDermott's and Doyle's Nonmonotonic Logic I and avoids the well-known weakness of this logic by a restriction to specific theories, which are sufficient for default reasoning purposes, however. The defaults are represented in a way that allows explicit control of their applicability. By blocking the applicability of a default the problem of interacting defaults can be avoided.

Keywords: Nonmonotonic Reasoning, Default Reasoning, Theorem Proving, Knowledge Representation

1. Introduction

During the last years the field of nonmonotonic reasoning has attracted many AI researchers. Different kinds of nonmonotonic reasoning have been identified ([McC 85] gives a list of 7 types, this list certainly not being complete), and different formalizations of such reasoning have been proposed. The most influential among these are

- McDermott's and Doyle's *Nonmonotonic Logic I* (NML I) [McD Do 80],
- Reiter's *Default Logic* [Rei 80],
- McCarthy's different versions of *Circumscription* [McC 80], [McC 84].

The main problem with these formalizations is, that they are not semi-decidable. In the case of NML I and Default Logic this stems from the fact, that the provability of a formula may depend on the unprovability of other formulas, and the unprovable formulas of first order logic (FOL) are not semi-decidable. In the case of Circumscription we have to deal with a second order formula, and second order logic is not semi-decidable (but note that Lifschitz [Lif 84] has identified interesting cases, where the Circumscription of a formula is equivalent to a first order formula).

A common answer to this problem is to give up the idea of theoremhood and to replace it by something like ***believability*** or ***reasoned believability***. This is especially the viewpoint taken in Reason/Truth Maintenance Systems as Doyle's TMS [Doy 79], Goodwin's WATSON [Goo 84][Goo 85] or de Kleer's extended ATMS [deK 86]. In these systems a network of dependencies between formulas is constructed, in which the derivability (believability) of a formula never depends on ***unprovability*** of other formulas, but may depend on the fact that other formulas are ***currently unproven***. What is modelled is not the ideal reasoning agent but instead the process of making inferences with limited resources.

A central problem with this approach is clear: the status of a formula may change from believed (**IN**) to disbelieved (**OUT**) or vice versa without adding or deleting any information, simply because the system has made further inferences. Criteria for when to stop making inferences and rely on the systems beliefs are lacking.

A more technical problem are the so called odd loops. A dependency network contains an odd loop whenever belief in a formula somehow depends on disbelief in the same formula. In the case of an odd loop TMS may run forever, WATSON diagnoses the loop and halts if it cannot label formulas correctly as **IN** or **OUT**. But note that WATSON may halt even if the corresponding logical theory is consistent. For instance the set of NML I formulas

1) ¬P -> P
2) M¬P -> ¬P

is consistent, but if WATSON is given these axioms it creates an odd loop and halts (our system has no problem with that case). De Kleer [deK 86] proposes to treat odd loops as contradictions.

Nobody can be very happy with these properties, but we have to live with them if we want a nonmonotonic system with the full expressive power of FOL.

But there is another approach to the problem of non-semidecidability: for many applications we do not need full FOL. The great success of PROLOG has shown this clearly. There are many interesting subsets of FOL which are decidable. If we restrict ourselves to such a subset, then also the nonmonotonic case becomes decidable and theoremhood need not be given up.

This is actually the approach we followed. FAULTY is a default prover that can handle Horn clauses without functions. (Note that Horn clause logic is not the same as PROLOG, we have true negation and negative assertions.) With this restriction FOL is decidable, since the Herbrand universe is finite.

There are two versions of FAULTY now, an older version described in [BreWi 84] [Bre 86] (in German) has recently been reimplemented on a SYMBOLICS Lisp machine. Examples in this paper are taken from the SYMBOLICS version of FAULTY.

In this paper we will justify our decision to base FAULTY on NML I and show how we overcome the wellknown

weakness of McDermott's and Doyles logic. We then describe how defaults are represented and how we deal with the problem of interacting defaults. An informal description of FAULTY's proof procedure follows and the paper concludes with an example dialogue.

2. Why NML I?

When we decided to implement a default prover, we first had to choose a basic nonmonotonic formalism for it.

We found that Circumscription was not a good candidate for our purposes. What we wanted to build was a system that could be used also by people not well trained in formal logic or even higher order logic. And our - may be very subjective - opinion is that Circumscription is quite difficult to use.

The main reason is that it is not enough to represent defaults using the abnormal-predicate **AB** as proposed by McCarthy [McC 84]. The effects of circumscribing **AB** crucially depend on the variables chosen for the Circumscription. If we have for instance

1) BIRD(x) & ¬AB(aspect1,x) -> FLIES(x)
2) BIRD(Tweety)

then we get **FLIES(Tweety)** as intended circumscribing **AB z** using **AB** and **FLIES** as variables. But if we add

3) PENGUIN(x) -> ¬FLIES(x)
4) OSTRICH(x) -> ¬FLIES(x)

we only get the desired result **FLIES(Tweety)** if we use **PENGUIN** and **OSTRICH** as additional variables.

For untrained people it is not easy to see what the effects of changes in the axiom set are and which variables to use when circumscribing **AB**. And we did not have a good idea how to follow McCarthy's suggestion [McC 84, p. 302] to use a "policy" database containing metamathematical statements for our intended general purpose default prover. We therefore preferred to choose among McDermott/Doyle's NML I and Reiter's default logic.

The main difference between NML I and Default Logic is that defaults in NML I are represented within the logical language and not as a kind of meta-statement as in Reiter's logic. This allows defaults and ordinary axioms to be handled in a uniform manner and it turns out to be very easy (as we will see) to adapt standard resolution proof techniques for NML I. This was the reason we chose to base FAULTY on NML I.

There is a well-known problem with NML I however: it is too weak, as McDermott and Doyle themselves pointed out. The modal operator **M** does not capture the full meaning of consistency, as intended. For instance the theory

{Mq, ¬q}

is consistent in NML I. This weakness has led to a lot of activity and authors, among them McDermott himself, have tried to strengthen the logic [McD 82] [Luk 84] [Moo 85].

It is certainly right, that NML I is too weak, but too weak for what? For reasoning about consistency. But this does not mean that it cannot be used for default reasoning purposes, if some restrictions are complied with. We restrict NML I in the following way:

1) *The modal operator* **M** *is only admitted in default rules as*

M A & B & M C -> C

where **A** *is a special literal, as will be explained in the next section, B and C are formulas (but the default must be representable as a horn clause).*

2) *We are only interested in the provability of formulas not containing* **M**.

With these restrictions the undesired consequences of the weakness of NML I disappear. No statements about the consistency or inconsistency of a formula can be made; it only can be expressed that a formula is derivable if this formula (and another special formula, see next section) is consistent. And the only way to find out if a formula is consistent is to try to prove its negation. With our restrictions NML I can model default reasoning adequately.

One more question has to be discussed here. In NML I (and similarly in Default Logic) the derivable formulas are defined in terms of fixed points. Since there may be any number of fixed points, the question arises, wether we want to define the derivable formulas as the intersection of fixed points (as McDermott and Doyle do), or wether we want formulas contained in at least one fixed point (Reiter's approach). Assume we have the following facts:

1) Most computer scientists are not millionaires.
2) Most Rolls Royce drivers are millionaires.
3) John is computer scientist and drives a Rolls Royce.

If we would build a prover following Reiter's approach and ask that prover "is it true that John is a millionaire?", then the prover would answer "yes", given the above axioms. But if we now ask "is it true that John is not a millionaire?" we get again the answer "yes", a somewhat unusual behavior for a prover.

Another unusual property of our system would be that if it has derived a fact **A** and another fact **B**, it does not follow that also the conjunction **A & B** is derivable.

This one fixed point approach is also pursued in Doyle's and Goodwin's Reason/Truth Maintenance Systems. These systems can be thought of as approximating the construction of one (arbitrarily chosen) fixed point.

We believe that it is better to remain agnostic in the case of conflicting evidence, following McDermott and Doyle in this point: for FAULTY only the formulas contained in the intersection of the fixed points are provable (but since many people seem to like the other approach too, now it is possible to run FAULTY in a mode where it derives a formula if it is contained in at least one fixed point).

3. The representation of defaults.

The problem of interacting defaults has been discussed broadly, see especially [Rei Cri 81]. One problem among others arises if we have a default that is more specific than another, for instance

1) ADULT(x) & M MARRIED(X) -> MARRIED(x)
2) STUDENT(x) & M ¬MARRIED(x) -> ¬MARRIED(x)

In this case we certainly want to be able to derive that a student named **John** is unmarried. But since default 1) creates a fixed point containing **MARRIED(John)** we cannot derive what we want. The question now is: how can we block the second unwanted fixed point from being created? What goes wrong is that default 1) is applied to students, but we do not want it to be applied in this case. So we have to find a way to explicitly control the applicability of a default rule. For that purpose we need a standard predicate **APPL** for *applicable* (precisely, we have a set of predicates APPLi, where i is the arity of the default, but this is not important here) and write our default in the following way:

3) M APPL(R1,x) & ADULT(x) & M MARRIED(x))
-> MARRIED(x)

Here the constant **R1** is used as a unique name for default 3) itself.

Now we can very easily block the applicability of a default by simply stating

4) STUDENT(x) -> ¬APPL(R1,x)

and we derive that John is unmarried, as it was our intuition.

This approach turns out to be very similar to McCarthy's use of the **AB** ("abnormal") predicate [McC 84] (but it has been developed independently from McCarthy and was first described in [BreWi 84]). McCarthy writes defaults as

5) BIRD(x) & ¬AB(aspect1 (x)) -> FLIES(x)

and circumscribes the formula **AB z**. To solve the problem of interacting defaults he uses *cancellation of inheritance axioms* like

6) OSTRICH(x) -> AB(aspect1 (x)).

Recall that Circumscription is a kind of minimization technique. Minimizing abnormalty now is very similar to maximizing the applicability of defaults, since defaults express what normally holds. And exactly this maximization is achieved with our approach, since defaults are applicable if not explicitly stated otherwise. And the close similarity between McCarthy's *cancellation of inheritance axioms* and our *blocking of default applicability axioms* is obvious (note that one of the subtitles of [Gro 85] is "Maximizing Defaults is Minimizing Predicates").

McCarthy himself was not too happy about his indexed aspects. He writes [McC 84, p. 299]:

The aspects themselves are abstract entities, and their unintuitiveness is somewhat a blemish on the theory.

Perhaps our use of names for the defaults is a bit more intuitive, in spite of the self-reference being introduced into the defaults. Note that this self-reference cannot lead to paradoxes, since **APPL** is used in a restricted way: it is only allowed in defaults under the scope of the modal operator and (negated) in the right side of the *blocking of default applicability axioms*. Moreover we can very easily hide the representation of defaults. The FAULTY user specifies defaults in a very natural way without having to be concerned about APPL's or M's. He simply writes

(R1 (BIRD(_x) ==> FLIES(_x)))

where **R1** is the name of the default, and FAULTY does the right thing ("==>" is to be read as "typically implies").

4. FAULTY's proof procedure

FAULTY's proof procedure is essentially a generalization of McDermott and Doyle's procedure for nonmonotonic propositional logic [McD Do 80]. The easiest way to explain it is to give some examples. Let's talk about Tweety again:

1) BIRD(Tweety)
2) M APPL(R1, x) & BIRD(x) & M FLIES(x) -> FLIES(x)

Now, of course, we want to prove **FLIES(Tweety)**. FAULTY first runs a standard unit resolution refutation proof, where **MQ** is, for all formulas **Q**, treated as a literal. We cannot derive the empty clause but we get the interesting clause

3) ¬M APPL(R1,Tweety) v ¬M FLIES(Tweety)

This formula is interesting, because it only contains literals beginning with **¬M**, we call such clauses *M-clauses*. **M** is intended to mean "is consistent", so if we knew that **APPL(R1,Tweety)** and **FLIES(Tweety)** were consistent, we could finish our proof. Now the only way to show that these formulas are consistent is to show that their negation is not provable. We therefore start two other proofs, one for **¬FLIES(Tweety)**, the other one for **¬APPL(R1,Tweety)**. In both cases the proofs fail without yielding M-clauses (they get the status **OPEN**). This allows us to add **M APPL(R1,Tweety)** and **M Flies(Tweety)** in our first proof, and the empty clause is derivable in this proof now (the proof becomes **CLOSED**: **FLIES(Tweety)** is proven.

Table 1

to prove	FLIES(Tweety)	¬APPL(R1,Tweety)	¬FLIES(Tweety)
yields	¬M APPL(R1,Tweety) v ¬M FLIES(Tweety)	-	-
labeling	CLOSED	OPEN	OPEN

Table 1 shows the (sub)proofs created. Only the interesting derived clauses are contained in the table.

Things are not always that easy, however. Let's look at our millionaires example again (RRD stands for Rolls Royce driver, CS for computer scientist, MILL for millionaire):

1) M APPL(R2,x) & RRD(x) & M MILL(x) -> MILL(x)
2) M APPL(R3,x) & CS(x) & M ¬MILL(x) -> ¬MILL(x)
3) RRD(Jim) & CS(Jim)

Trying to prove **MILL(Jim)** we get the proofs shown in Table 2.

The interesting thing here is that we can consistently label the proofs of our example in two different ways as failed (**OPEN**) or successfully finished (**CLOSED**). If we label the proof for **¬MILL(Jim) OPEN**, **M MILL(Jim)** can be added in all proofs and the proof for **MILL(Jim)** gets **CLOSED**. But we can do it also the other way around: labeling the proof for **MILL(JIM) OPEN** makes the proof for **¬MILL(Jim) CLOSED**. These different labelings correspond exactly to the different fixed points of our theory. Since there is one labeling in which the proof for **MILL(Jim)** is **OPEN**, **MILL(Jim)** is not contained in all fixed points and hence cannot be derived.

Generally a FAULTY proof for a goal consists of two steps. The first step, the construction of (sub)proofs, can semi-formally be described in the following way:

push the goal onto the agenda
until the agenda is empty do
 remove the top element from the agenda
 and start a proof for it
 if the empty clause is derived, mark this proof **CLOSED**
  else if no M-clause is derived, mark this proof **OPEN**
   else for each literal **¬M Q** *in each derived M-clause*
    unless
     -Q *is contained in the agenda or*
     there is already a proof for **¬Q**
    push **¬Q** *onto the agenda*

This proof construction phase terminates, since there is only a finite number of possible instances of literals beginning with **¬M**.

Secondly, all admissible labelings for the still unlabeled proofs have to be found. To find out if a labeling is admissible, one proceeds as follows: for each proof for **¬Q** with the label **OPEN** the literal **M Q** is to be added to all proofs. Now in all **OPEN** proofs the empty clause must be underivable, in all **CLOSED** proofs the empty clause must be derivable.

The goal is proven, if its (sub)proof is **CLOSED** in all admissible labelings.

This proof procedure is of course not the way FAULTY actually proceeds. There are some ways to cut the number of created proofs and the check of admissible labelings can easily be done by a propositional prover, but this is beyond the scope of this paper.

5. Example

The following example shows how a FAULTY knowledge base is defined.

```
(deffaulty-kb flying-objects
        (axioms
         (bird tweety)
         (penguin hansi)
         (bird fred)
         (not flies fred)
         (airplane jumbo)
         (penguin _x -> bird _x)
         )
        (defaults
         (r1 (bird _x ==> flies _x))
         (r2 (penguin _x ==> not flies _x))
         (r3 (airplane _x ==>  flies _x))
         (r5 (flies _x ==> has-wings _x))
         (r6 (has-wings _x ==> has-feathers _x)
         )
        (exceptions
         (penguin _x -> not appl r1 _x)
         (airplane _x -> not appl r6 _x)))
```

Table 2

to prove	MILL(Jim)	¬APPL(R2,Jim)	¬MILL(Jim)	¬APPL(R3,Jim)
yields	¬M APPL(R2,Jim) v ¬M MILL(Jim)	-	¬M APPL(R3,Jim) v ¬M ¬MILL(Jim)	-
labeling 1	CLOSED	OPEN	OPEN	OPEN
labeling 2	OPEN	OPEN	CLOSED	OPEN

The *blocking of default applicability axioms* are called *exceptions* in the definition. The axioms are taken partly from [McC 84]. The above definition creates the knowledge base as an instance of a Zetalisp Flavor. We can send messages to this knowledge base, the most interesting message is certainly **:PROVE**. Here are some examples

(send flying-objects :prove '(flies tweety))

yields: **PROVABLE**

(send flying-objects :prove '(flies hansi))

yields: **UNPROVABLE**

(send flying-objects :prove '(not flies hansi))

yields: **PROVABLE** (if the first exception were missing, we would not get this result)

(send flying-objects :prove '(has-wings jumbo))

yields: **PROVABLE**

(send flying-objects :prove '(has-feathers jumbo))

yields: **UNPROVABLE**

6. Problems and future work

The main problem with FAULTY is efficiency, naturally. A set of standard resolution proofs, which themselves are expensive enough, must be run. But we are not too pessimistic about that. First we think, a slow implementation is better than none at all, and second there is much room for parallelization in FAULTY's proof procedure, so we can hope for much better efficiency when parallel computers become available.

Another concern is that FAULTY does not record results, since it is a pure prover. The purpose of the Reason/Truth Maintenance Systems mentioned in the introduction, however, was not only to make nonmonotonic inferences, but also to keep track of inferences made so far. This allows axioms to be changed without having to recompute everything. Now it's a natural idea to combine the two approaches and to build a reason maintenance system, where all **IN** formulas are actually theorems of the underlying axioms and all **OUT** formulas are actually unprovable, not only currently unproven. This system, to be called TINA (This **I**s **N**o **A**cronym), is under development.

Acknowledgements

The first version of FAULTY was built in close cooperation with K.H. Wittur. Thanks also to F. di Primio, who is the 'father' of BABYLON, the expert system building tool developed in our research group.

REFERENCES

[Bre Wi 84]
Brewka, G. and Wittur, K.H.
Nichtmonotone Logiken.
Universität Bonn, Informatik Berichte Nr. 40., 1984.

[Bre 86]
Brewka, G.
Uber unnormale Vögel, anwendbare Regeln und einen Default Beweiser.
Proc. GWAI (German Workshop on Artificial Intelligence) 85, 1986.

[deK 86]
de Kleer, J.
Extending the ATMS.
Artificial Intelligence 28, 1986.

[Doy 79]
Doyle, J.
A Truth Maintenance System.
Artificial Intelligence 12, 1979.

[Goo 84]
Goodwin, J.
WATSON: A Dependency Directed Inference System.
Proc. Non-Monotonic Reasoning Workshop, 1984.

[Goo 85]
Goodwin, J.
A Process Theory of Non-monotonic Inference.
Proc. IJCAI 85.

[Gro 84]
Grosof, B.
Default Reasoning As Circumscription.
Proc. Non-Monotonic Reasoning Workshop, 1984.

[Lif 84]
Lifschitz, V.
Some Results on Circumscription.
Proc. Non-Monotonic Reasoning Workshop, 1984.

[Luk 84]
Lukaszewicz, W.
Nonmonotonic Logic for Default Theories.
Proc. ECAI 1984.

[McC 80]
McCarthy, J.
Circumscription - A Form of Non-Monotonic Reasoning.
Artificial Intelligence 13, 1980.

[McC 84]
McCarthy, J.
Applications of Circumscription to Formalizing Common Sense Reasoning.
Proc. Non-Monotonic Reasoning Workshop, 1984.

[McD 82]
McDermott, D.
Nonmonotonic Logic II: Nonmonotonic Modal Theories.
JACM Vol. 29 No. 1, 1982.

[McD Do 80]
McDermott, D. and Doyle, J.
Non-Monotonic Logic I.
Artificial Intelligence 13, 1980.

[Moo 85]
Moore, R.C.
Semantical Considerations on Nonmonotonic Logic.
Artificial Intelligence 25(1), 1985.

[Rei 80]
Reiter, R.
A Logic for Default Reasoning.
Artificial Intelligence 13, 1980.

[Rei Cri 81]
Reiter, R. and Criscuolo, G.
On Interacting Defaults.
Proc. IJCAI 1981.