A THEOREM-PROVER FOR A DECIDABLE SUBSET OF DEFAULT LOGIC

Philippe BESNARD - René QUINIOU - Patrice QUINTON

IRISA - INRIA Rennes
Campus de Beaulieu
35042 RENNES Cedex
FRANCE

Abstract

Non-monotonic logic is an attempt to take into account such notions as incomplete knowledge and theory evolution. However the decidable theorem-prover issue has been so far unexplored. We propose such a theorem-prover for default logic with a restriction on the first-order formulae it deals with. This theorem-prover is based on the generalisation of a resolution technique named saturation, which was initially designed to test the consistency of a set of first-order formulae. We have proved that our algorithm is complete and that it always terminates for the selected subset of first-order formulae.

I INTRODUCTION

Decision-making is a fundamental feature of some fields of A.I., but it cannot always be supported by a complete world knowledge. Non-monotonic logic [AI 80] is an attempt to remedy to this fact. However, the present realisations in the field of non-monotonic logic avoid the decidability problem by use of heuristics. To apprehend this problem, we have selected a solvable class, the predefinite variable clauses, which determines a decidable subset in the context of default logic. Default logic [Rei 80] allows the inference of consistent though unvalid formulae in a first-order theory. The consistency of a set of formulae, which is a crucial problem in default logic, can nicely be solved by use of a particular resolution technique named the saturation.

First of all, we recall the principles of default logic and then focus on an attractive default class, free-defaults, which allow more plausible inferences. Part III is devoted to the presentation of a special resolution technique named the saturation. We also propose a generalisation of saturation which constitutes the heart of our default theorem-prover presented in part IV. This default prover is complete and always terminates.

II DEFAULT LOGIC

First we introduce the default logic as proposed by Reiter in [Rei 80] and the proof-procedure he has set up. Next, we study a default subset, the free-defaults, which allow additional plausible inferences. A default rewriting rule can be applied to ordinary defaults to obtain free-defaults.

A Definition of default logic

The main goal of default logic is to provide some facilities to represent common-sense knowledge. For example, we know that generally "a bird can fly" (except penguins, ostriches..). This knowledge can be represented as the **default** δ:

$$\frac{\text{bird}(x)\ :\text{Mfly}(x)}{\text{fly}(x)}$$

bird(x) is the prerequisite of the default and fly(x) the consequent (noted CONS(δ)). This default is interpreted as: "if x is a bird and if it is consistent to suppose that x can fly, then infer that x can fly". So defaults are patterns of inference schemata as "until a proof of the contrary, assume...". That is, a default denotes a plausible inference, for which a conclusion cannot be derived by use of classical inference rules.

A default theory is a couple (Δ,T) where Δ is a set of defaults and T a consistent first-order theory. The use of the defaults of Δ extends the theory T by building several consistent first-order theories called extensions, each of which contains T.

Example 1: $T=\{\ A\ ,\ \sim B \vee \sim C\ \}$

$$\Delta=\left\{\ \frac{A\ :MB}{B}\ ,\ \frac{A\ :MC}{C}\ ,\ \frac{C\ :MD}{D}\ \right\}$$

This default theory has 2 extensions:

$E_1=Th(\ A\ ,\ \sim B \vee \sim C\ ,\ B\ )$

$E_2=Th(\ A\ ,\ \sim B \vee \sim C\ ,\ C\ ,\ D\ )$

Definition: A formula Q has a default proof with respect to a default theory (Δ,T) iff there exists a finite sequence $\Delta_0, \Delta_1, \dots, \Delta_k$ of finite subsets of Δ, such that

(1) $T \cup CONS(\Delta_0) \vdash Q$

(2) For $1 \leq i \leq k$

$T \cup CONS(\Delta_i) \vdash Prerequisites(\Delta_{i-1})$

(3) $\Delta_k = \emptyset$

(4) $T \cup (\cup CONS(\Delta_i))$ is satisfiable.

(2) expresses that the prerequisite of a default must be true to let the consequent of this default to be in the extension.

Reiter has shown the completeness result for default proofs, i.e. that a formula Q has a default proof with respect to (Δ,T) iff there exists an extension of (Δ,T) which contains Q. However (1) and (4) show clearly that the extension membership problem is not even semi-decidable. This leads us to consider a subset of first order logic, for which the decision problem is resolved.

B Defaults without prerequisite or free-defaults

Defaults may have counter-intuitive behaviours ([Rei 81]). In addition to those already cited, we present below an example of such a default:

$T = \{ \sim fly(Max) \}$ and $\delta = \frac{bird(x) \ :Mfly(x)}{fly(x)}$

The only extension of the default theory $(\{\delta\},T)$ is T itself. It doesn't contain ~bird(Max), that we expect to be true, because Max doesn't fly while generally birds do.

Consider $\delta' = \frac{:Mbird(x) \Rightarrow fly(x)}{bird(x) \Rightarrow fly(x)}$ $\left(\neq \frac{:M(x) \ \ bird(x) \Rightarrow fly(x)}{(x) \ \ bird(x) \Rightarrow fly(x)} \right)$

δ' leads to a unique extension $E' = Th(\sim fly(Max), bird(Max) \Rightarrow fly(Max))$ and E' contains ~bird(Max). If later the formula bird(Max) & ~fly(Max) becomes true, then the default δ' instantiated by Max cannot be used and ~bird(Max) is not true in the new default theory $(\delta', T \cup \{bird(Max)\})$

The default without prerequisite δ' expresses a permanent knowledge in opposite to the situational knowledge expressed by δ. By δ' we always know that if x is a bird then he flies. By δ only when x is a bird we know that he flies. We now generalise the previous transformation:

$$\delta = \frac{P(x) \ :MR(x)}{R(x)} \qquad \longmapsto \qquad f(\delta) = \frac{:MP(x) \Rightarrow R(x)}{P(x) \Rightarrow R(x)}$$

$f(\delta)$ is called a free-default.

We have proved the following result [Bes 83]:

Theorem 1: for any extension E of a default theory (Δ,T), there exists an extension E' of $(f(\Delta),T)$ such that $E \subset E'$

III SATURATION

In this chapter we first introduce a kind of resolution technique named the saturation. Next, we present the saturation by sets, a generalisation of the saturation which will be useful in the definition of our default proof-procedure.

A Definition of the saturation

The saturation [Bos 81] is a particular resolution technique, decidable for predefinite variable clauses. A clause is a predefinite variable clause if:

- the only function symbols that occur in the clause are constant symbols.
- all the variables occuring in a positive literal occur also in a negative one.

This restriction is acceptable for an application since

- in the context of databases, formulae are function-free
- all the clauses without function symbol, in which the type of all the variables is defined by a predicate, are predefinite clauses.

For example, the clause $C(x_1, \dots, x_n)$ can be transformed in the predefinite variable clause $Type_1(x_1) \& \dots \& Type_n(x_n) \Rightarrow C(x_1, \dots, x_n)$.

Definition: a set C of predefinite variable clauses is saturated if each resolvant of 2 clauses of C either is a tautology or is subsumed by a clause of C.

Theorem 2: a saturated set of predefinite variable clauses is inconsistent iff it contains the empty clause.

The saturation of a set of predefinite variable clauses produces, by resolution, a saturated logically equivalent set of predefinite variable clauses. Due to the previous theorem, the saturation gives a means to test the satisfiability of a set of predefinite variable clauses. The principle of saturation is to produce, first, all the resolvants one parent of which is the most recently produced clause. The tautologies and the clauses subsumed by one of their ancestors are not produced. A-ordering [Kow 69] for the literals and resolution upon the leftmost literal permit to restrict the number of inferences.

Example 2: saturation of F

F= { ~C v E , B v C , ~B v D , A v C , ~A }

c_1 = ~C v E

c_2 = B v C

c_3 = ~B v D

c_4 = r(c_2, c_3)= C v D

c_5 = r(c_1, c_4)= D v E

c_6 = A v C

c_7 = ~A

c_8 = r(c_6, c_7)= C

c_9 = r(c_1, c_8)= E

The set {c_1, ..., c_9} is saturated. The empty clause has not been produced then the set F is consistent.

B Saturation by sets

It is obvious that a clause subsumed by a clause produced later in the saturation process is useless. So is the clause c_5 subsumed by c_9 in example 2. It is possible to reduce the number of these useless clauses by relaxing the order of resolvant production. To reach this goal we have extended the notion of saturation.

Let E and F be sets of predefinite variable clauses. The saturation of F on E, produces the set noted S(E,F) recursively defined by

S(E,∅)=E

S(E,F)=S(E ∪ {c} , F' − {c})

where .c ∈ F

.r ∈ F' if

*either r ∈ F

*or r is not subsumed by a clause of E and r is the resolvant upon the leftmost literal of c with a clause of E.

This process is called saturation by sets.

Example 3: E={ A v B } and F={ ~A }

S(E,F)=S({ A v B } , { ~A })

=S({ A v B , ~A } , { B })

=S({ A v B , ~A , B } , ∅)

={ A v B , ~A , B }

Theorem 3: S(E,F) is finite iff E and F are finite.

Theorem 4: S(E,F) is saturated if E is saturated.

Corollary 5: S(∅,C) is a saturated set logically equivalent to C.

IV DEFAULT PROOF BY SATURATION

A The default prover

We now only consider default theories (Δ, T) where Δ is a free-default set. Let Δ' be a subset of Δ and Q the formula to be proved. Remark that Prerequisites(Δ')=∅ thus

(2') T ∪ CONS(Δ") |− Prerequisites(Δ')

(3') Δ" = ∅

It is worth-noting that the steps (2) and (3) of the default proof (§ II) are immediatly verified. To solve (1) and (4) of the default proof, we saturate T ∪ CONS(Δ') and T ∪ CONS(Δ') ∪ {~Q}. By use of the saturation by sets, these steps can be joined because

S(∅, T ∪ CONS(Δ') ∪ {~Q})

|=| S(S(∅, T ∪ CONS(Δ')), {~Q})

In this way no clause is computed twice. In addition, during the elaboration of S(∅, T ∪ CONS(Δ') ∪ {~Q}) the clauses of S(∅, T ∪ CONS(Δ')) are computed first. So if the empty clause belongs to S(∅, T ∪ CONS(Δ')) then T ∪ CONS(Δ') is unsatisfiable and the saturation process is stopped (because no default proof can be found, given that (4) cannot be true). Then it is useless to test if Q is a consequence of T ∪ CONS(Δ'). During the search of a refutation of T ∪ CONS(Δ') ∪ {~Q} we have discovered one of T ∪ CONS(Δ'). This never happens with linear resolution, for example.

Let Δ={δ_1, ..., δ_n}. The proof procedure evaluates the consistency of T ∪ CONS(Δ') for Δ'=∅, Δ'={δ_1}, Δ'={δ_2}, Δ'={δ_1, δ_2}... (Δ' ranges over the set of the subsets of Δ). It then computes successively S(∅, T), S(∅, T ∪ CONS(δ_1)), S(∅, T ∪ CONS(δ_2)), S(∅, T ∪ CONS(δ_1, δ_2))...

The preceding remark remains true, so:

S(∅, T ∪ CONS(δ_1)) |=| S(S(∅,T), CONS(δ_1))

S(∅, T ∪ CONS(δ_2)) |=| S(S(∅,T), CONS(δ_2))

S(∅, T ∪ CONS(δ_1, δ_2))

|=| S(S(∅, T ∪ CONS(δ_1)), CONS(δ_2))

If for example S(∅, T ∪ CONS(δ_i)) contains the empty clause then, for every Δ' subset of Δ containing δ_i, S(∅, T ∪ CONS(Δ')) also contains the empty clause and it is useless to compute it. Due to this optimisation, only the consistent or minimally inconsistent sets T ∪ CONS(Δ') are computed.

By operating like this, a default proof is nothing but a global saturation, S(∅, T ∪ CONS(Δ) ∪ {~Q}), knowing that only some subsets of these clauses are really computed. Note also that

$$S(\emptyset,\ T \cup CONS(\Delta) \cup \{\sim Q\})$$
$$|=|\ S(S(\emptyset,\ T \cup CONS(\Delta)),\ \{\sim Q\})$$

$S(\emptyset,\ T \cup CONS(\Delta))$ can then be computed once and for all and remains the same for every query. The saturation by sets provides a kind of compilation of the extensions of a default theory.

The default proof procedure presented here is complete and decidable [Bes 83], so the prover has the wanted properties:

Theorem 6: the prover is complete.

Theorem 7: the prover always terminates.

B Example of a default proof by saturation

$T=\{ C \vee D,\ E\}$

$\delta_1=\dfrac{:M\sim C \vee \sim E}{\sim C \vee \sim E}$ $\delta_2=\dfrac{:M\sim D \vee \sim E}{\sim D \vee \sim E}$ $\delta_3=\dfrac{:MB \vee \sim E}{B \vee \sim E}$

The default theory $(\{\delta_1,\ \delta_2,\ \delta_3\},\ T)$ has 2 extensions:

$E_1=Th(T \cup CONS(\delta_1,\ \delta_3))$
$E_2=Th(T \cup CONS(\delta_2,\ \delta_3))$

This default theory is submitted to the prover:

$S_1=S(\emptyset,\ T)=\{\ C \vee D\ ,\ E\ \}$

$S_2=S(S(\emptyset,\ T),\ CONS(\delta_1))$
$\quad=S_1 \cup \{\ \sim C \vee \sim E\ ,\ D \vee \sim E\ \}$

$S_3=S(S(\emptyset,\ T),\ CONS(\delta_2)) = S_1 \cup \{\ \sim D \vee \sim E\ \}$

$S_4=S(S(S(\emptyset,\ T),\ CONS(\delta_1)),\ CONS(\delta_2))$
$\quad=S_2 \cup \{\ \sim E\ ,\ \square\ \}$

$S_5=S(S(\emptyset,\ T),\ CONS(\delta_3))= S_1 \cup \{\ B \vee \sim E\ \}$

$S_6=S(S(S(\emptyset,\ T),\ CONS(\delta_1)),\ CONS(\delta_3))$
$\quad=S_2 \cup \{\ B \vee \sim E\ \}$

$S_7=S(S(S(\emptyset,\ T),\ CONS(\delta_2)),\ CONS(\delta_3))$
$\quad=S_3 \cup \{\ B \vee \sim E\ \}$

The remaining clauses of $S_8=S(S_4,CONS(\delta_3))$ are not computed since S_4 contains the empty clause. Now we can use these sets of clauses to answer any query. To answer a query Q we evaluate $S(S_i,\ \{\sim Q\})$ with i increasing in order to take advantage of the inclusion of the S_i's.

For example, a default proof of D is given by:

$S(S_1,\ \{\sim D\})=\{\ C \vee D\ ,\ E\ ,\ \sim D\ \}$

$S(S_2,\ \{\sim D\})=S(S_1,\ \{\sim D\})$
$\qquad \cup \{\ \sim C \vee \sim E\ ,\ D \vee \sim E\ ,\ \sim E\ ,\ \square\ \}$

V CONCLUSION

We have proposed a theorem-prover for a subset of default logic. The default theories of this subset only contains free-defaults and all the consequents and axioms can be transformed in predefinite variable clause form. Our theorem-prover based upon saturation by sets is complete and always terminates. We are about implementing it in PROLOG.

The main advantage of the prover is to deal efficiently with querying and default updating, once the compilation phase has been realised. However, the prover is not adapted to axiom updating. This provides a direction for further investigations as extending the subset of formulae the theorem-prover is able to deal with.

REFERENCES

[AI 80] Special issue on non-monotonic logic
Artificial Intelligence, vol. 13, 1980

[Bes 83] Besnard, P.
A proof-procedure for a non-monotonic logic
Technical Note 198, University of Rennes I, 1983

[Bos 81] Bossu, G. & Siegel, P.
La saturation au secours de la non-monotonie
Thesis, University of Aix-Marseille II, 1981

[Kow 69] Kowalski, R. & Hayes, P. J.
Semantic trees in automatic theorem-proving
Machine Intelligence 4, pp. 87-101, 1969

[Rei 80] Reiter, R.
A logic for default reasoning
Artificial Intelligence vol. 13, pp. 81-132, 1980

[Rei 81] Reiter, R. & Criscuolo, G.
On interacting defaults
Proc. IJCAI-81 pp. 270-276, Vancouver, 1981