A Process Theory of Non-monotonic Inference

James W. Goodwin

Department of Computer and Information Science
Linkoping University
Linkoping, Sweden

Abstract: Artificial Intelligence needs a formal theory of the *process* of non-monotonic reasoning. Ideally, such a theory would decide, for every proposition and state of the process, whether the program should believe the proposition in that state, or remain agnostic. Without non-monotonic inference rules, non-monotonic inferences cannot be explained in the same relational, rule-based fashion as other inferences. But with such rules, theoremhood is often formally undecidable and thus a useless criterion for our purpose. So how could any system be a "non-monotonic logic programming language"?

Our method uses the language, inference rules and proofs of non-monotonic logics, but ignores theoremhood. Instead, it defines states of the reasoning process, and focuses on *current proof* as the criterion for belief. It defines "admissible beliefs" and "valid proof" for given states, and we prove in [5] that a belief is currently admissible iff it is currently proven. The primitive non-monotonic condition is "currently unproven".

The theory, Logical Process Theory, can accept a range of non-monotonic logics. It was inspired by Doyle's RMS [3] and is similar to his more recent theory in [4|. A model implementation, WATSON, exists and has been used to write a small diagnostic reasoner, which reasons non-monotonic ally using violation of expectations and an abstraction hierarchy.

1. DEFINITION OF LOGICAL PROCESSES

Briefly, Logical Process Theory (LPT) accepts a triple **<L,I,£>**, where L and *I* are the object language and inference rule of some suitable logic, and £, the *enumeration,* is a control parameter. The theory then specifies the *admissible statet* of an inference process which obeys that logic and makes inferences in the order indicated by £ A state consists simply of the set of inferences made so far, and the set of formulas taken to be believed by the machine executing the process. At no point is theoremhood in the given logic defined or used by LPT itself; LPT deals always with finite, computable states.

LPT requires of L only that it be a denumerable set of formulas; their grammatical structure and semantics is opaque to LPT and hence unrestricted.

The inference rule I may be any set of *inference ttept.* An inference step is a triple <M,N,c>, where the *monotonic antecedents* M are a set of formulas of L; the *non-monotonic antecedents* N are a set of formulas of L; and the *consequent* c is a formula of L. We also require some finite bound on the site of M and N for all inferences in J.

Informally, an inference **<M,N,c>** may be read: *if all the formula in* M *are currently believed, and none of the formulat in* N are *currently believed, then* c *mutt be currently believed.* Note that this is stated as a constraint, not as an imperative (not as "if ... then infer c"). The constraint is understood to apply any state of the process after the inference was made. Because it is a conditional constraint, it does not necessarily force belief in c; for the same reason, it can be safely made at any time, whether its antecedent conditions are satisfied at that time or not.

No further restrictions are placed on L or J by LPT. Normally L would be defined by a grammar, and J by a set of schemas or rules. The rules of default logic |10], for example, have the form:

$$\frac{a \mid b}{c}$$

where a, b and c are single formulas; this may be expressed in LPT by the schema **<{a},{not b},c>.**

This research wot sponsored by the Swedish Board of Technical Development.

We prefer the alternative of introducing non-mono tonicity directly into the language L by a modal operator, as in [9|. WATSON uses an operator UNPROVEN. In this case it is convenient to represent the standard first-order inference rules as inferences in which N is always empty, and to obtain all non-monotonicity via one inference schema for introducing the modal operator, **<{},{a},"(UNPROVEN a)">.** This may be read "if a is not currently believed, then "(UNPROVEN a)" must be currently believed".

Since L is denumerable and the sizes of M and N have a constant bound, *I* is also denumerable. We denote enumerations of J by the control parameter £. Intuitively, £ represents the order in which individual inference steps are taken by a Logical Process. To take an inference step simply means to add it to the database; the current database in the j:th state is therefore just the set of the first j inferences of £.

To represent sets of beliefs, we define an *IN/OUT libelling* as a total function from L to the set of labels {IN,OUT}. A state of the machine is then a pair **s = <D,G>** of a database **D** and an IN/OUT labelling G.

An individual inference **d = <M,N,c>** is valid under an IN/OUT labelling G if every formula in M is labelled IN by G, and every formula in N is labelled OUT by G; *4* is *invalid* under G otherwise. Where it is clear which labelling or state is intended, we say simply that *4* is valid.

A state *<D,G>* is *relaxed* if for each **d∈D,** either the consequent of *d* is labelled IN by G, or *4* is invalid under G.

A relaxed state satisfies all the constraints in the database locally, but to eliminate circular proofs, a non-local ordering must also be imposed. Therefore: a state <D,G> is *well-founded* iff there exists a partial order **≺** over Lu J such that

1. For every formula s labelled IN by G, there b a **d∈D** which is valid under G, has s as its consequent, and **d≺s,** and
2. For every valid **d∈D,** for every formula s in the monotonic antecedents M of **d, s≺d.**

Finally, an *admissible* state is one which is both relaxed and well-founded, and its labelling b said to be an admissible labelling for its database.

Having defined the admissible states, the main task at hand b to relate them to proofs. A *proof of a formula* c b a tree of inferences with c at the root, in which each node has for each of its monotonic antecedents a daughter node proving that antecedent, and has no other daughter nodes. A proof b *valid in a Hate* <P,G> if every inference in the proof tree b contained in *V* and b valid under G.

(Premises are introduced by inferences with empty M and N. Such inferences are always valid, and may appear at any point in the enumeration. This allows inference and input of new external information to be arbitrarily interleaved.)

2. PROPERTIES OF LOGICAL PROCESSES

The main theorem about Logical Processes correlates belief with current proof: *for any admittible ttate t and any formula c of L, there it a valid proof of c in $ iff e it labelled IN in t.* (Proof to appear in [5]).

Some additional properties are as follows (proofs again in [5]):

Admissible states are physically representable: any database b finite, and in any admissible state, the number of formulas labelled IN is finite.

Non-monotonic belief behavior b actually obtained: a formula may be IN in one state and OUT in a later state. This may result either (1) because of new premises being given, or (2) merely due to further inference based on information already available, or any combination of the two.

We call both cases "non-monotonic inference", because the set of currently admbsible beliefs may vary non-monotonically as the process continues. From the perspective of a process theory of inference, there b no very natural distinction between them, nor much reason to try to find one.

(We have several times met the objection that "non-monotonicity" has a standard definition, according to which the case 1 above b "non-monotonicity" and case 2 b something else. Now, "non-monotonic logic" has indeed a precise and well-establbhed meaning: any logic in which adding premises may delete theorems. But neither case 1 nor case 2 mentioned theorems; they described non-monotonic *inference,* as a process in which current proof of a formula may come and go. Thb b a reasonable use of the term "non-monotonic inference", consistent with its common informal use in AI. Certainly, non-monotonicity of belied during the

inference process is a different matter than non-monotonicity of theorems under addition of premises. The former is computable where the latter is not, and the former explains case 2 where the latter does not allow the question to be put. For Artificial Intelligence, those are advantages. Note also that the set of inference rules which are "non-monotonic" is the same in either sense.)

In a non-monotonic LP, a given database may have zero, one or many admissible labelling. Moreover, its predecessor or successor may independently have zero, one or many admissible labelling*. It is therefore possible that an admissible state s may be "inaccessible" under a given £, in the sense that the process cannot reach s while passing only through admissible states. However, there is always a permutation of the initial subsequence of £ up to and including s, such that every state up to s is admissible.

By a proof adapted from [I], for a database to have rero admissible labelling, it must contain a configuration called an "odd loop", which is usually a kind of "Liar's Paradox" argument. By ruling this case out, we guarantee continuity of the sequence of admissible states, and also make possible an efficient reason maintenance algorithm for finding them.

Since the formulas of L are opaque to LPT, LPT cannot say anything about their consistency, in either a model theoretic sense, or in terms of provability of a contradiction. This is only to be expected, however, of a process theory of non-monotonic reasoning, since the purpose of non-monotonicity is to allow a reasoning agent to make unsound but reasonable inferences and then back out of them if they turn out to be inconsistent due to new information or - again, we stress the focus on process - simply due to further reasoning. The only guarantee given by LPT, that of admissibility, is that the system knows exactly what it does and does not have current proof of.

3. WATSON: A MODEL IMPLEMENTATION

WATSON is a model implementation in LISP of a non-monotonic Logical Process. It is a pure forward chaining system. The inference rules are "wired in" to the interpreter, which does nothing but add dependencies (inferences permitted by the rules) to the database. All valid inferences are drawn. Unlike previous systems [2,7] there is no way to "escape to lisp" and manipulate dependencies.

How much can be done with pure non-monotonic inference alone? DIAGNOSE is a small diagnostic reasoner written in WATSON, which uses non-monotonicity both to reason by violation of expectations, and to work downwards in an abstraction hierarchy. So the system is not trivial.

Some inference rules which we cannot state in the format allowed by LPT include a forward chaining analog of negation by failure; conditional proof as in |3] and thus dependency directed backtracking as well; and inferring the set of all currently proven instances of a formula (contrast circumscription [8]). We have already extended WATSON to use these rules, as well as to allow expression of control in the language: "reasoned control of reasoning". Revision of LPT to account for these extensions is in progress.

REFERENCES

Charniak, Eugene et al. *Artificial Intelligence Programming.* Lawrence Erlbaum, New Jersey, 1980.

de Kleer, Johan et al. "AMORD: Explicit Control of Reasoning". In *ACM SIGPLAN Notice,,* 12:8 (1977).

Doyle, Jon. "A Truth Maintenance System". In *Artificial Intelligence* 12 (1079) pp. 231-272.

Doyle, Jon. "The Ins and Outs of Reason Maintenance". In *Proe. IJCAI-8S,* pp. 349-351.

Goodwin, James W. "WATSON: A Dependency Directed Inference System". Forthcoming dissertation, Linkping University, Sweden.

Goodwin, James W. "An Improved Algorithm for Non-monotonic Dependency Net Update". Tech. Report LITH-MAT- R 82-23, Linkopiug University, Sweden.

McAllester, D. A. "Reasoning Utility User's Manual". AI Memo 667, MIT AI Lab, Cambridge, Ma., 1982

McCarthy, John. "Circumscription - A Form of Non-Monotonic Reasoning". In *Artificial Intelligence,* 13:1 (1980) pp. 27-39.

McDermott, Drew and Jon Doyle. "Non-Monotonic Logic I". In *Artificial Intelligence,* 13:1 (1980) pp. 41-72.

Reiter, R. "A Logic for Default Reasoning". In *Artificial Intelligence,* 13:1 (1980) pp. 81-132.