PHILOSOPHICAL ISSUES, 2
Rationality in Epistemology, 1992

Models for Belief Revision

Raymundo Morado*

1 Introduction

These notes on the problems of modelling belief revision arose in part from a paper in which Brian Skyrms maintains that probability calculus is the only sensible universal non–monotonic logic.[1] According to him, a full treatment of induction "should be thoroughly probabilistic".[2]

*The first part of this paper was discussed at the 1990 SOFIA Conference, in Campinas, Brazil. The second part was worked in part with the help of Jonathan Mills, and presented at the 1989 Interamerican Philosophy Congress in Buenos Aires, Argentina. The third part owes a lot to discussions I've had with Michael Dunn, Michael Gasser, Gordon Beavers, and meetings of the Indiana University Philosophy/H&PS ABD Discussion Group.

[1]We say that a relation Θ is monotonic under a function σ if $\Theta(a,b)$ entails $\Theta(\sigma(a),\sigma(b))$. In the case of monotonic reasoning, Θ is the relation "is inferred from" and σ is any function that adds semantic content. In other words, adding information to the premisses increases the information in the conclusion: $a \rightarrow b \Rightarrow a \ \& \ c \rightarrow b \ \& \ c$. Some people prefer the weaker $a \rightarrow b \Rightarrow a \ \& \ c \rightarrow b$, taking $x \ \& \ y \rightarrow y$ for granted.

[2]Skyrms (1990). This paper builds on ideas from Skyrms (1984) and Skyrms (1987).

In these pages I offer only a passing mention of some of the problems and proposed ways to handle a problem familiar in the philosophy of induction: that of an epistemic subject forced by evidential constraints to modify its beliefs. In a very simplified model, this becomes the problem of truth maintenance with updates in data base management. In the first section I mention some of the criticisms levelled against the use of probability theory to deal with non-monotonic inferences. In the second section I explore a specific case: adding or retracting information from a database that works as an analog of a belief system. Here I go into some implementation details and I mention a modal interpretation of the system states. The third section concludes with a brief excursion on the use of relevance logics to model knowledge acquisition.

2 Probabilities and Belief Revision

For a long time the use of probabilistic models has been a point of contention between two different ways of modeling belief in artificial intelligence. The probabilistic treatment, or similar techniques, yielded spectacular results in the seventies with the implementation of expert systems. For example, Shortliffe's MYCIN, for which Heckerman put forward a probabilistic interpretation of the certainty factors in 1986, and Duda's PROSPECTOR which used Bayesian style analysis. There is an extensive bibliography on the subject of coherent propagation of probabilities in Bayesian inference networks (see, for example, Pearl 1987), and some elementary books on Artificial Intelligence like Nilsson and Genesereth (1987) give a thoroughly probabilistic account of reasoning with uncertain beliefs.

The addition of mechanisms to handle belief revision to expert systems is desirable. The standard implementation of an expert system is through a set of conditional rules. If the evidence matches some of the antecedents, the respective rules are triggered and the consequents of them become active. This is a very simple model, but it can take us a long way. The rules might be triggered according to a data base of "evidences". But the action commanded by a rule might be to modify the data base itself. So, some rules can be viewed as functions from and into data bases, which provides us with a crude model of belief revision.

After the success of MYCIN and PROSPECTOR many people tried to extend the probabilistic techniques to other areas. But they soon found that "the information necessary to assign numerical probabilities is not ordinarily available", as McCarthy and Hayes had warned.[3]

This problem of assigning precise numerical values to probabilities has been addressed in many ways. In 1987 Zadeh proposed his *Dispositional Logic* to explain some uses of expressions like *usually.* If we say that the usual temperature is in the sixties, we are not offering a unique value as we would with an average, nor are we saying what we expect. We might actually expect the average temperature to change for good because of the greenhouse effect. Even if *Usually* is not an indication of probability, it can be object of a mathematical treatment in the absence of hard numeric data.

Zadeh proposed to analyze dispositions expressed by the phrase "Y is usually S when X is R". *Usually* signals a disposition, understood as a high conditional probability. Since the notion of "high" is fuzzy, *usually* acts as a fuzzy quantifier. Let us see an example. The canonical form reads

$$\text{(usually) } (Y \text{ is } S \text{ if } X \text{ is } R)$$

Now, suppose we make a list of objects in our space, and assign to each a degree in which the object has each of those two properties, S and R:

Object(1)	$R = .8$	$S = .8$
Object(2)	$R = .6$	$S = .9$
Object(3)	$R = .7$	$S = .3$
etc.		

In this case, the truth of the canonical form will be the application of the membership function of the fuzzy quantifier *usually* applied to the proportion of objects which have property S among those that have property R.

[3]McCarthy and Hayes (1969), p. 39.

Zadeh intended his system as an attack to other approaches to inference from commonsense knowledge, such as default reasoning, circumscription, and nonmonotonic reasoning. He contended that such approaches based on logic rather than probability theory "are symbolic rather than computational in nature. What this implies is that such approaches do not provide a system for inference from commonsense knowledge when it is lexically imprecise and/or involves imprecisely known probabilities".[4]

Another way out of the complaint that we do not know the exact probabilities has been to use the Dempster–Shafer theory, which can be used to generate measure–propagation rules. Basically, it substitutes the probability estimate by a pair representing lower and upper bounds on the estimate. This replaces exact numbers with a range of possible ones, and the size of the range reflects our degree of confidence in our own probability attribution. A generalization of this approach can be found in Levi (1974) which uses a class of probability functions to represent a set of beliefs, instead of a single probability function. Assuming that the set is *convex* (closed under "mixtures") it determines a set of associated intervals (but not the other way around). Gäerdenfors and Sahlin (1982) argued that this should be expanded to include a measure of the epistemic reliability of these probability functions.[5]

Against all this there are complains that the way to deal with default reasoning has mostly been, and should continue to be, symbolic. In the classical treatments of default reasoning, we might not know anything about to which degree of confidence an object X has a certain property Y. Yet we

[4]Zadeh (1988), p. 1. Pearl (1987) and Geffner & Pearl (1987) (quoted in Etherington (1988) p. 73 have contended that default reasoning can be viewed as a kind of qualitative (but not fuzzy) probability.

[5]Another interesting case is M.L. Ginsberg who back in 1985 argued for the use of probabilities in the study of default reasoning. But two years latter he "found McCarthy's arguments increasingly compelling" (Ginsberg (1987), p. 7), and eliminated all treatment of probabilities from Ginsberg (1987), which he still purported to be "a fairly complete and representative collection of the work in nonmonotonic reasoning" (Ginsberg (1987), p. 2).

might say that this is so unless something abnormal occurs. This is not a numeric evaluation but a qualitative one that lends itself to purely symbolic manipulation. For instance, some theories of default reasoning would transform the normal expert system rule P if Q into P if (Q and $\neg AB$), where $\neg AB$ means that nothing abnormal takes place. This has been a old practice in both inductive and abductive studies where the clause "other things equal" plays a similar role.

Let us now ask about the logical properties of such relations. The conditional can be weakened by strengthening the antecedent. That is, if we know that P if (Q and $\neg AB$), then we can generate every default rule of the form P if (Q and $\neg AB$ and R). This is not only a property of default reasoning but also valid in classical logic. Adding information in the premisses can only make the previous inference even more certain if the reasoning is monotonic. If it is not, the $\neg AB$ clause should trap any incoherences. For example, P if (Q and $\neg AB$ and $\neg P$) sounds absurd. Normally we think that only logical truths can be derived from their own negations (by *reductio*). Yet, under the assumption that the normal case is to find P whenever we find Q, having Q and $\neg P$ is already abnormal, and so Q and $\neg AB$ and $\neg P$ fails and the rule should never be activated in the expert system. This is attractive because it lets us move from a non-monotonic structure to a monotonic one.

Ginsberg (1987) p. 10, claims that this capacity to generate new rules by means of strengthening antecedents, which holds for monotonic systems and default reasoning in the symbolic approach, fails for the probabilistic one and that this is a defect. Suppose that the probability of a Q being P is greater than N; it doesn't follow that the probability of being P, for a Q which is R, is also greater than N. This fails because we have reduced the space of the Q's to the Q's that are R, in which it might be more or less likely to be P than in the rest of the space.

It sounds paradoxical that Ginsberg should accuse probabilistic treatments of non-monotonic inference on the grounds that it is *only* non-monotonic. But the intuition behind this seems to be that default reasoning is justified because of a secret clause that qualifies all it's conclusions. When we conclude that Tweety flies based only on the information that Tweety is a bird, we do not literally conclude that it flies, but only that it flies *by default*. This implicit qualification gives default reasoning it's legitimacy. The addition of the conjunct $\neg AB$ simply makes explicit this hidden condition. Now, is there a similar secret condition that would restore monotonicity in the probability approach? Kevin Korb has suggested to me the condition of total evidence. As long as this condition holds, there is no possibility of rationally updating the probabilistic conclusion because of the principle that such updating would have to be evidence–driven, but there is no new evidence to be expected.

Now, back to the remark in McCarthy and Hayes (1969) about the difficulty of having probability estimates, they concluded that "Therefore, a formalism that required numerical probabilities would be epistemologically inadequate" (*ib.*). As it stands, this is a very poor argument. It was fleshed out in McCarthy's work on circumscription in 1980. He considered the proposal to understand the assumption of normality as a probability estimate; perhaps abnormal cases are just the ones that are less probable. But this will not do. To use the canonical example, a typical bird flies, but it might be the case that most birds do not; it is also possible that for any randomly selected bird, the probability that it flies falls below whatever positive threshold we select. Nutter (1987) p. 840, mentions the situation each late spring, when infants outnumber adults in the bird population. We might know that something is improbable, and yet call it typical.

McCarthy adds that it is hard to see how we could substitute default reasoning with probabilities. Can we talk of the conditional probability that Tweety has a broken wing, given that we have only been told that Tweety is a bird? Also, assumptions are based on our expectations, and so cannot be objective probabilities. Are they perhaps the so–called "subjective probabilities"? McCarthy thinks not, because when we try to reason about a usual missionaries–and–cannibals puzzle, "we mentally propose to ourselves the normal non–bridge non–sea–monster interpretation *before* considering these extraneous possibilities, let alone their probabilities".

That is, it looks like the sample space is not defined *for us* prior to our calculations. Any computation from a limited statement of the facts, as most expert systems will have to deal with, seems to bypass probability theory (and fuzzy logic, for that matter). The point is not that we can not use probabilities when constructing our beliefs; the point is that we do not use them, and so, any reconstruction of human reasoning in the absence of complete information should be prepared (at least in a big proportion of the cases) to do without probability estimates.

3 Non–monotonicity in Logic Programming

A way to get a handle on the problem of belief revision in expert systems is to consider the problem of negation. The idea is to treat the negation as failure as a report on a model. (The epistemic version would be "Not true, as far as we know at this point".) But, since new information can be added, amounting to a change of model, we need a semantics that *indexes* a previously proved negation to the corresponding model if we want to preserve monotonicity (the negation of p might be no longer provable, nor its consequences; a different approach is the use of non–monotonic logics, circumscription, default reasoning, ATMS's, or *censor rules*).

An epistemic subject capable of facing minimal challenges in the real world (be it a computer or a human), needs to be able to handle incomplete and/or inconsistent descriptions about what states of affairs actually hold. This goes beyond any deductive powers and belongs rather to the inductive construction of a model of reality. The possibility of error means that the structure (the *logic*) of belief revision is non–monotonic if we are to avoid inconsistency. The normal way to avoid inconsistency is to create expensive Truth Maintenance Systems on top of our inferential engine (for deductive databases) that *turn off* enough beliefs to restore consistency.

The use of computers in the study of logic falls mainly in two categories: computer assisted instruction (CAI), especially for the teaching of logic, and what is sometimes called *Computational Logic.* Computational logic includes *logic programming, theorem-proving, rewrite-rule systems and production-rule systems.*[6] Logic programming is generally based on resolution and unification, which have clear declarative semantics, control structures like backtracking, normally modeled as depth-first searches, and side-effects. Side-effects pose important problems. While they have clear procedural semantics and provide useful or efficient control of computations, operations of the like of *asserting* and *retracting* are hard to put in declarative terms; they seem to detract from the *purity* of logic programming and to have an inherently imperative nature. Even more: not only they seem to be non-declarative, but also to induce non-monotonicity. Let us examine an example:

Negation can be represented in different ways inside a system with querying capabilities. A data base which includes the use of rules is called a *deductive database* and, from a theoretical point of view, it *is just a logic program.*[7] So, we can treat the problem of knowledge maintaining in databases as a especial case of the general problem of knowledge maintaining in theorem proving systems.

We must distinguish between negation as failure, represented in Prolog with `not( )` or with `\+( )`, from negation of terms. Negation as failure applies to clauses without purely syntactic manipulation of negated terms. We can use negative literals like $\neg p$ in

$$\neg p : - r, \neg s.$$

but it is more effective to use a context dependency to know whether a literal is to be taken as negated or not. This helps resolution, as pointed out in Mills (1989) p. 5.

As an example we transform all truth-functional premisses into a list of conjuncts of their Conjunctive Normal Form. Each conjunct is a clause which has been transformed itself into a list containing two lists: the positive atoms and the negated ones. So, the premisses become:

[6] Gabriel (1984) p. 1.
[7] Lloyd (1984) p. 62.

```
[
  [[positives], [negatives]],
   [[positives], [negatives]],
   ...
]
```

and now resolution can operate simply by checking occurrences of the same term in the list of positives of one clause, and the set of positives of a second one. Our proof is complete if we generate the empty clause `[ [ ], [ ] ]`.

3.1 DEMONS

`assert` and `retract` pose a problem for the declarative interpretation of Prolog programs. They can be seen as forcing recompilation. The problem with dynamic code is that what is true of a predicate at a moment in our computation might not be true later, and therefore a predicate subject to assertion or retraction would be non–logical.

Assertion and Retraction can be viewed as movements from a certain state of the world (or of our knowledge of it) to another one. We can define an *accessibility* relation in terms of a binary function that takes an operation (assertion or retraction), a possible world, and a set of propositions, and returns a possible world.

We can implement a Prolog system in which each clause generates at compile time a clause template. This is handed to a *demon*, that is, a clause specific routine that copies and fills the template at run time, relieving the full compiler from having to deal with anything but very complex clauses. Such demons were first constructed by Overbeek while working with Mills in 1985. The demon is sometimes called an *assertive* demon because retraction can be implemented based on the clauses that were already compiled. We can concentrate on demons that handle simple clauses without more than one uninstantiated variable. This avoids having to keep symbol tables, allocate permanent variables and do occurs–checking. (For complex clauses the demon can still trap into the full compiler.)

Demons are useful to have a meta–logical view of `assert` and `retract`. *An* `assert` *or a* `retract` *is a meta–operator* *which changes the universe of discourse of the Prolog computation. Demons and their templates specify rules for constructing new universes of discourse from contexts and higher–order objects (templates).*[8] This is a nice illustration of the view that a predicate like **assert** acts not as a function inside a normal declarative world, but as a relation among them. (Actually, Mills proposes a temporary universe of discourse during which the template is instantiated and the procedure linked back; but we can abstract from this and think of the action of the demon as instantaneous.)

3.2 MODEL THEORETIC SEMANTICS

We can think of the work of the demon as a relation between possible worlds. A demon is a binary function that takes a possible world and a clause and returns a possible world. This model theoretic approach resembles the working of temporal operators that take a world *along* the stream of time and produce a new world, and can be generalized to all the side–effects. For instance, Ashcroft (1976) claims that *an assignment statement is really an equation between "histories", and a whole program is simply an unordered set of such equations.* So, the expression of an assignment as

```
I = I + 1
```

is misleading. We certainly do not want to say that the value of *I* is its own successor, but that it *becomes* so. Adding an explicit temporal dimension would clarify the semantics. For instance, in *Lucid*

```
next I = I + 1
```

is read as saying that *at each stage in the history of I the value of I at the next stage is the current value of I plus one.*[9]

This ideas have been developed for the *InTense* language. InTense is essentially a superset of Prolog. It evolved as an extension of W. Wadge's Chronolog by the addition of space

[8]Mills (1988), 6.3.5.
[9]Ashcroft (1977) p. 520.

indices. The basic additions to Prolog are: a built–in treatment of spatial and temporal dependencies (Chronolog has only temporal ones) and multiple infinite streams of formulas (instead of static Horn–clauses).

Modal logic has been used in theories of probability in arithmetic and *dynamic logic* in theories of computation and action in general.[10] The treatment of time offered here is limited to discrete representations (hours, days, etc.), not to a continuous one. There are good reasons for this. Besides the problem of representing continuous quantities in digital computers with finite registers, tense predicate logic of the real time line has inherent problems; e.g., it is not effectively axiomatizable.

There is a consideration that might support the use of a discrete temporal logic for logic programming. Consider the case of self–modifying code. If we encounter a clause like

```
assert(assert(p)).
```

the demon in charge might have to construct another demon.

Or, in a procedure like:

```
p :- q, r.
r.
q :- retract(p :- q).
```

we are deleting inside the search tree and might not be able to backtrack past it to the parent call.

We can see a treatment of this problem in the Quintus manual, in chapter 14, (version 2.3) where we read that *the definition of an interpreted procedure that is to be visible to a call is effectively frozen when the call is made.* That is, the retraction does not alter the procedure till after the call has been completed, emphasising that the minimum units of *time* in the history of the program are the procedure calls. The retraction or assertion does not affect an ongoing query. This means that in implementations like Quintus the accessibility relation is inherently discrete, to witness, in terms of procedure calls, and therefore a discrete temporal semantics like the one offered by InTense is acceptable for these

[10]Benthem (1985) p. 3.

cases. Chriss Moss compared recently this situation to *the classic readers/writers method of ensuring file consistency. If you have already started reading (i.e. activated a query) then the file (relation) stays the same, but any new readers (activations) get the new version.*

InTense programs can be viewed extensionally: ground terms restricted (through their intensional parameters) to points or possible worlds in a time–space field. The accessibility relation is given with prefix operators. `first` signals the "root" world, while `prev`, and `next` move us along a discrete temporal axis. (Similar observation hold for the space operators `initial`, `prior` and `rest`.)

Let us see an example in action. The following InTense procedure in the file "fib.int" produces an infinite stream of Fibonacci numbers:

```
drive :- nl,
         fib(X),
         write(fib(X)),
         next drive.

first fib(0).
first next fib(1).
fib(X) :-      prev fib(Y),
          prev prev fib(Z),
          X is Y + Z.
```

would produce the following session:

```
/u2/morado/intense> intense

ASU I_n_T_e_n_s_e ver.1 (Jan.10,1989)

Query World < t0=0 >
?- consult(fib).

Consulting fib.int....consulted.
yes.

Query World < t0=0 >
?- drive.

fib(0)
fib(1)
fib(1)
fib(2)
fib(3)
fib(5)
fib(8)
fib(13)
[till stack overflow]
```

And this is a transcript of the traced run of the first Fibonacci numbers, without all the `nl/0`, `write/1`, or builtin calls:

```
(1) CALL: first0 drive :
(1) CALL: first0 fib(_73781) :
(1) EXIT: first0 fib(0) :
fib(0)

(0) CALL: first0 next0 drive :
(0) CALL: first0 next0 fib(_73949) :
(0) EXIT: first0 next0 fib(1) :
fib(1)

(-1) CALL: first0 next0 next0 drive :
(-1) CALL: first0 next0 next0 fib(_74117) :
(0) CALL: first0 next0 fib(_74205) :
(0) EXIT: first0 next0 fib(1) :
(0) CALL: first0 fib(_74213) :
(0) EXIT: first0 fib(0) :
(0) CALL: first0 next0 next0 _74117 is 1 + 0 :
fib(1)

(-3) CALL: first0 next0 next0 next0 drive :
(-3) CALL: first0 next0 next0 next0 fib(_74437) :
(-2) CALL: first0 next0 next0 fib(_74525) :
(-1) CALL: first0 next0 fib(_74573) :
(-1) EXIT: first0 next0 fib(1) :
(-1) CALL: first0 fib(_74581) :
(-1) EXIT: first0 fib(0) :
(-1) CALL: first0 next0 next0 _74525 is 1 + 0 :
(-3) CALL: first0 next0 fib(_74533) :
(-3) EXIT: first0 next0 fib(1) :
(-3) CALL: first0 next0 next0 next0 _74437 is 1 + 1 :
fib(2)
```

and so on.

The system "knows" that the same function `fib()` has different values according to its place in the sequence that the system maintains. For instance, at a given moment t we might assert P, but at time $t + n$ retract it and at time $t + n + m$ assert the negation of P. As long as P is not independently derivable at $t + n + m$, there is no danger of contradiction.

But now, how do we know P is not independently derivable? This problem is computationally undecidable unless we severely restrict the expressive powers of the system, a problem Truth Maintenance Systems have to face.

Representation of time and space is by default static in most programming languages. A simple implementation of InTense in Prolog only requires adding temporal and spatial tags to each clause for each absolute temporal operator, plus some extra clauses to make explicit the relation between the relative ones.

Inside each world we have monotonicity. Asserting or retracting just take us to a different world. Since each clause is implicitly tagged with the world at which it holds, we do not have to worry about the side–effects. Actually, `assert` and `retract` can be given a declarative interpretation and, in this sense, no longer cause any side–effects! As Mitchell says, *each world in the InTense universe for a program can be treated as a separate monotonic system.*[11]

3.3 PROBLEMS WITH THE POSSIBLE WORLD SEMANTICS

We have achieved consistency at the price of relativizing our results to a given moment in the computation. In terms of a theorem prover, the base information DB1 which let us deduce the set T1 of theorems is now replaced with DB2 which has an associated set T2 of theorems. (For normal cases Tn will be a superset of DBn, but we do not need to presuppose completeness of our inference rules here.) The question now is: How much of my previous work of theorem proving can

[11]Mitchell (1988), p. 12.

be consistently carried into the new possible world? That is, how to define the intersection C of common theorems between T1 and T2?

The obvious way is to keep an extra field or tag for each theorem pointing to the ancestors from which it was deduced. If one of the ancestor was affected by the assertion or retraction, then we propagate the changes down the proof tree using some traversal algorithm like the ones used for GC tags. If a full list of ancestors is maintained for each theorem, then we can postpone the definition of our new set of theorems and evaluate their status by need. The accessibility relations could establish guards for the use or previously proved theorems. The guards would be saying "Do not use if descendant of retracted clause". (A further complication will be that case when `X` was obtained through the success of `not(Y)`, because an assertion of `Y` has to be considered as a retraction of `not(Y)`.)

We have talked about using temporal semantics to interpret changes in the epistemic states of a data base oriented system or a theorem prover, as accessibility relations from epistemic states to epistemic states, in analogy to the transformations from possible worlds to possible worlds in modal logic. In the standard interpretation, a possible world is a set of propositions, informally the ones that are *true* at that world. If we substitute the ontological notion of truth for an epistemological notion of "computational provability in the system", we can represent sequences of changes in a database or the set of valid inferences that a theorem prover can make at a given moment, as changes through the space of possible worlds. I proposed the language *InTense* to use temporal indices for truth maintenance because *InTense* can keep track of the temporal indexings automatically thanks to its built–in recognition of discrete sequences.

4 Relevance logic

Normally the reason to retract *P* arises from information to the effect that *P* is false; after all, if we do not retract *P*, later true information could lead us to a contradiction. But we could take another approach, namely to countenance the presence of contradictions, while localizing them in such a way that even if we can extract all the consequences from a contradictory epistemic state, we still avoid collapsing into a logically trivial system in which any proposition would be provable.

Belnap[12] proposed that *minor inconsistencies in its data should not be allowed to lead (as in classical logic) to irrelevant conclusions.* Instead of discarding contradictory information whenever it is so recognized, we would want to recover all the information fed to the computer, perhaps by several different sources.

We want the computer to report anything it has been told, even if the information is recognized as false because it is contradictory. (This also means that the computer should not report it as being true, but rather as being "trusted" by the system.) This is a natural way of using negation as failure and affirmation as success.

Based on Belnap's ideas, Melvin Fitting and Mike Dunn have been working on different implementations of a Prolog extension that allows queries to succeed along two different views of data processing. There are two ordering of the data that produce different lattice configurations. As a model of knowledge acquisition, we use an approximation lattice in which the only relations we have are

- No–info $\leq$ False
- No–info $\leq$ True
- False $\leq$ Both
- True $\leq$ Both

to signal that from no information we can go on to asserting the truth or falsehood of a proposition, and from any of these to registering in the data base that it is *both* true and false.

[12]Belnap, Nuel D., Jr. (1976). "A useful four–valued logic". *Modern uses of multiple–valued logic; Proceedings of the 1975 International Symposium on Multiple–valued Logic.* G. Epstein and J. M. Dunn (ed.), Reidel.

Since this is a partial ordering, the relation is monotonic by anti–symmetry.

As a model of deducibility relations, the lattice is flipped on its side and we have the following ordering:

- False $\leq$ No–info
- False $\leq$ Both
- No–info $\leq$ True
- Both $\leq$ True

This is called the Logical Four–valued Bilattice (FOUR).

The system is so designed that it implements a version of relevance logic with the property of being paraconsistent, that is, from any given contradiction not everything can be deduced. The computer implementation that I am working on tries to use semantic tableaux and coupled trees, but it is still in the drawing board stage, so I will talk about a more mature implementation by Melvin Fitting called Q–Log.

Q–Log is a logic programming language based on the bilattice FOUR, and implemented using an SLD generalization based on semantic tableaux.[13] The idea is to take a series of goals and rules and

1. Select one goal
2. Use some rule to resolve with the goal
3. Repeat 1 until we cannot resolve anymore.

Let's put it a tad less informally:[14]

Given a program P, a set of goals G and a selection function S, we can define an "SLD–refutation of $P+G$ using S" as a finite SLD–derivation with the same P, F and S, that ends in the empty clause. Now, a SLD–derivation is a sequence

[13]SLD stands for Selection–function–based Linear resolution for Definite clauses. This comes from R.A. Kowalski and D. Kuehner's "Linear Resolution with Selection Function", *Artificial Intelligence*, 2, pp. 227–260, 1971.

[14]I follow J. W. Lloyd (1984), with minor changes in terminology.

$G, G_1, G_2, \ldots$ of goals produced by substituting the goal in G_i selected by S, with the result of resolving it with some rule in P. Resolution in turn is unifying the head of the rule with the goal selected. The resulting goals are those in the tail of the rule after the unifier is applied to them.

Axioms are written using the syntax: `<head> if <body>`. `<head>` is atomic, and `<body>` is any formula built up from atoms, including constants true and false, using **neg** **(prefix)**, **and**, **or**, **otimes** and **oplus** (all **infix**). The guard connective, :, is also allowed in `<body>`. In addition, there is a built-in binary predicate, eq. `eq(X, Y)` is turned into Prolog's `X = Y`. (`X == Y` in Quintus) `neg eq(X, Y)` is turned into Prolog's `X \= Y` (`X \== Y` in Quintus). A `<body>` must always be present. Axioms are stored in the form of Prolog facts: `axiom(-)`.

A further topic of research is to extend Q–log to true quantification with coupled trees for greater efficiency. Also, we need to work on the justification of this system in terms of its plausibility. The problems I have encountered are many and insidious, but the most unintuitive ones are related to the fact that P v $-P$ fails in 4–valued logic. (This might be considered a feature and not a bug by some.) A further reason for dismay is the treatment of contradictions. We know that (P and Q) is contradictory if P is contradictory or Q is contradictory. This is a sufficient condition, but not a necessary one. For instance, let us follow the trace for `(a or neg(a))`:

```
| ?- query(a or neg(a)).

** (2) 0 Call (compiled): closes([f a or neg a]) ?
** (4) 0 Exit (compiled): conjunctive(f a or neg a) ?
** (5) 0 Exit (compiled):
                 components(f a or neg a,f a,f neg a) ?

** (6) 0 Call (compiled): closes([f a]) ?
** (6) 0 Fail (compiled): closes([f a]) ?

** (15) 0 Call (compiled): closes([f neg a]) ?
** (16) 0 Exit (compiled): negative(f neg a) ?
** (17) 0 Exit (compiled): component(f neg a,t a) ?
** (18) 0 Call (compiled): closes([t a]) ?
** (18) 0 Fail (compiled): closes([t a]) ?
** (15) 0 Fail (compiled): closes([f neg a]) ?

** (2) 0 Fail (compiled): closes([f a or neg a]) ?
Q-Log No
```

Q–log begins by negating it for reductio into `[f a or neg(a)]`. Since this is a conjunctive "Alpha", it has as components `f a` and `f neg(a)`. But, to close this branch the only weapon we are given is to either close `[f a]` or `[f neg(a)]`, each on its own! Since the absurdity of one of the conjuncts is not necessary for the absurdity of the conjunction (only sufficient), the system is not able to close the branches and announces that `(a or neg(a))` is not a theorem.

5 REFERENCES

Ashcroft, E. A., and Wadge, W. W. (1976). "Lucid —A Formal System for Writing and Proving Programs". *SIAM v J. Comptg.*, vol. 5, No. 3, pp. 336-354.

Ashcroft, E. A., and Wadge, W. W. (1977). "Lucid, A Nonprocedural Language with Iteration". *Communications of the ACM*, vol. 20, No. 7, pp. 519- 526.

Benthem, Johan van (1985). *A Manual of Intensional Logic.* CLSI.

Etherington, David W. (1988). *Reasoning With Incomplete Information*, Morgan Kaufmann Publishers, London.

Gabriel, John; Tim Lindholm, E. L. Lusk, and R. A. Overbeek (984). "A Tutorial on the Warren Abstract Machine for Computational Logic". Argonne National Laboratory report ANL-84-84.

Gärdenfors, P., and N.-E. Sahlin (1982). "Unreliable probabilities, risk taking, and decision making", *Synthese* 53, pp. 361-386.

Geffner, H. and Pearl, J. (1987). "Sound Defeasible Inference", TR CSD-8700XX R-94, Department of Computer Science, University of California, Los Angeles.

Ginsberg, Matthew L. (1987). *Readings in Nonmonotonic Reasoning*, Morgan Kaufmann Publishers, Los Altos, California.

Levi, I. (1974). "On indeterminate probabilities", *Journal of Philosophy* 71, pp. 391–418.

Lloyd, John Wylie (1984). *Foundations of Logic Programming.* Springer Verlag.

McCarthy and Hayes (1969). "Some Philosophical Problems from the Standpoint of Artificial Intelligence", in Matthew L. Ginsberg (ed.), *Nonmonotonic Reasoning*, (1987), Morgan Kaufmann, pp. 26-52.

Mills, Jonathan Wayne (1988). Ph. D. Dissertation. Arizona State University.

Mills, Jonathan Wayne (1989). "Compiling and Executing Theorems Efficiently as Logic Programs". Handout for C690.

Mitchell, William Howard (1988). "Intensional Horn Clause Logic as a Programming Language — It's Use and Implementation". M. S. Thesis, Arizona State University.

Nilsson, N. J., and Genesereth, M. R. (1987). *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann, California.

Nutter, J. T. (1987). "Reasoning, Default", in S. Shapiro (ed.), *Encyclopedia of Artificial Intelligence*, New York, pp. 840-848.

Pearl, J. (1987). "Probabilistic Semantics for Inheritance Hierarchies with Exceptions", TR CSD-8700XX R-93, Department of Computer Science, University of California, Los Angeles.

Skyrms, Brian (1984). *Pragmatics and Empiricism*, Yale U. P.

Skyrms, Brian (1987). "Dynamic Coherence and Probability Kinematics", *Philosophy of Science*, pp. 1-20.

Skyrms, Brian (1990). "Coherence, Probability and Induction", to be published in the Proceedings of the 1990 SOFIA Conference, Campinas, Brazil.

Zadeh, L. A. (1988) "Dispositional Logic and Commonsense Reasoning", CSLI Report No. CSLI-88-117.