Judgemental reasoning for expert systems

Dr Tim Niblett

The Turing Institute
George House
36 North Hanover Street
GLASGOW G1 2AD

Abstract

We present a new system for plausible reasoning in expert systems. It is an extension of the Horn clause subset of first order logic, and is distinguished by its use of non-numeric certainties and the assignment of certainties to logical formulae rather than events. Model theoretic and ftxpoint semantics are sketched for this language. An implementation in the logic programming language Prolog is described and the advantages of the system are discussed.
Key words- Plausible reasoning, probability, expert systems, logic programming

1. Introduction

Some kind of judgemental or probabilistic reasoning forms an essential part of many expert systems. The need for such reasoning arises whenever knowledge is incomplete or data uncertain. This is the case in many of the problem domains to which expert systems are applied. In this paper we present a system for judgemental reasoning which differs in two respects from the more usual approaches. It uses a non-numeric representation for belief, and assigns certainties to logical formulae rather than events. It is an extension to the work described in (Shapiro 1983).

We shall argue that this system meets the following criteria which should be fulfilled by any system of plausible reasoning used in expert systems.

(1) Clear semantic*
We should be able to provide a clear semantics for our reasoning system. Both the knowledge engineer building an expert system and the end user of the system should know what certainties are and the run-time behaviour of the system should reflect these meanings in a well denned manner. If this requirement is not met it is difficult to develop general purpose debugging and maintenance tools for knowledge bases.

(2) Ease of explanation.
A determining feature of an expert system is its ability to explain its reasoning on demand. We require that the derivation of certainties be easily explainable. This entails that the mechanism by which certainties are combined be itself capable of explanation.

(3) Expressive power
The system should be powerful enough to express an expert's knowledge about uncertainty flexibly and naturally.

We distinguish two sources of uncertainty. Firstly, uncertainty about observations (or facts) in the application domain. Secondly, uncertain rules (or heuristics) which express an expert's general knowledge about the domain. Our system, following the approach of Shapiro (Shapiro 1963) uses an extension of the Horn clause subset of first order logic. Facts ore represented by atomic formulae (atoms), rules by non-unit clauses.

The fundamental idea is to extend the the range of truth values assigned to atoms from the set *[false ,true }* to *certainties.* Certainties are lattices and have operations of least upper bound ($\cup$) and greatest lower bound ($\cap$) denned on them. We are relaxing the restriction that certainties be totally ordered, and specifying that they are partially ordered. This permits the use of non-numeric certainties to express knowledge about uncertainty. The defined operations allow us to combine certainties duing the inference process. The rules used by the system are of the form $\langle A \leftarrow B, f \rangle$ where $A \leftarrow B$ is a *Horn clause* and / is a *certainty function* specifying how the certainties of the atoms in *B* are combined to provide a certainty for *A.* The ability to specify a combination function for each rule provides flexibility, although in practice we find that most rules use a general combination function.

In the next section we describe the formal apparatus underlying our system. Both a model theoretic and fixpoint semantics are provided in the manner of (van Emden & Kowalski 1976V This provides evidence that we can satisfy criteria (1) and (2) above. We then describe a simple application of the system, which has been implemented in Prolog as part of a complete expert system shell. This provides evidence that we can satisfy criterion (3) above.

2. The formal apparatus

2.1. Basic definitions

Definition
A certainty space C is a complete lattice. This is a partially ordered (under <) set C, with operations $\bigcup$ and $\bigcap$ (least upper bound and greatest lower bound respectively) defined for every subset of C.

Definition
We extend £ to sequences over C by defining: $\langle c_1, \ldots, c_n \rangle \leq \langle c'_1, \ldots, c'_n \rangle$ iff $c_1 \leq c'_1$ and ... and $c_n \leq c'_n$.

Definition
A function f from sequences of certainties to certainties in some certainty space C is monotone iff for all sequences s_x and s_2 of length n over C, $s_1 \leq s_2$ implies $f(s_1) \leq f(s_2)$.

Definition
A logic program with uncertainties is a finite (non-empty) set P of pairs of the form $\langle A \leftarrow B, t \rangle$ where $A \leftarrow B$ is a definite (or Horn) clause, and *t* is a monotone function from sequences of certainties to certainties.

2.2. Semantics

We can now define the semantics of logic programs

with uncertainties as an extension of that given by van Emden and Kowalski (van Emden and Kowalski 1976) for definite clause programs without uncertainties.

Definition
The Herbrand Universe U(P) is defined recursively as,

(1) The set of constant symbols in P (or the constant symbol a if there are none).

(2) All atoms of the form $p(t_1, \cdots, t_n)$ where the t_i are in U(P).

Definition
The Herbrand base H(P) of a logic program P *is* the set of all ground atoms formed by using predicate symbols from P with ground terms from the Herbrand Universe U(P).

Definition
An interpretation of a logic program with uncertainties P is a function from H(P) to a certainty space C.

Definition
An interpretation I_1 is $\leq$ an interpretation fe iff $I_1(a) \leq I_2(a)$ for all a in H(P).

Definition
A model M of P is an interpretation of P satisfying the following condition:

For any clause $\langle A \leftarrow B, f \rangle$ in P and any ground instance $A' \leftarrow B'_1 \& \cdots \& B'_n$ of the clause, then $M(A') \geq f(M(B'_1), \cdots, M(B'_n))$

2.2.1 Model theoretic semantics

Definition
Given two models M_1 and M_2 for P we define the intersection $(\cap)$ of the two models pointwise as $M_1 \cap M_2(a) = M_1(a) \cap M_2(a)$

Proposition
Given two models M and M' for P. the intersection of the two models $M \cap M'$ is a model.

Proof
(omitted)

Theorem
$M(P) = \bigcap_{M \text{ a model}} M$ exists and is the least model of P.

Proof
The proof is straightforward.

The meaning of the logic program P is defined to be the least model H(P) of P

2.2.2 Fixpoint semantics

We now provide a constructive definition of M(P) by showing it to be the least fixpoint of a function Tp from interpretations to interpretations.

Definition
Let Tp(*I*) be defined pointwise on an interpretation 1 as follows:

$T_P(I)(a) = \bigcup_{\lambda \in A_a} (f_\lambda(I))$ where $A_a = \{a \leftarrow b \mid a \leftarrow b$ *is a ground instance of a clause in* P$\}$ and $f_\lambda(I) = f(I(b_1), \ldots, I(b_n))$ where $\lambda = a \leftarrow b_1 \& \cdots \& b_n$.

Proposition
Tp is monotone over the lattice of interpretations, and moreover the lattice of interpretations (with po <) is complete.

Proposition
Tp has a least fixpoint.

Proof
Since Tp is a monotone function on a complete lattice, Tp has a complete lattice of fixpoints and therefore a least fixpoint.

We can elaborate on this result by proving that T_P is continuous as well as monotone if the underlying certainty functions we use are continuous.

Definition
A certainty function f from sequences of certainties to certainties is continuous if $f(\bigcup_{x \in X} x) = \bigcup_{x \in X} f(x)$ for all chains X.

Proposition
If the underlying certainty functions f are continuous then the corresponding functions f_x are continuous

Proposition
If the underlying certainty functions f are continuous then T_P is a continuous function, that is: $\bigcup_{n \in \Pi} T_P(I_n) = T_P(\bigcup_{n \in \Pi} I_n)$ for all chains $\{I_n \mid n \in \Pi\}$ of interpretations.

Proof
(omitted)

Proofs omitted above can be obtained from the author on request.

This completes our discussion of the semantics of logic programs with uncertainties, and shows that the meaning of a logic program with uncertainties can be approximated computationally The certainty computed at run-time is a lower bound on the true certainty.

3. An example

To illustrate the practical application of the above we shall describe an implementation in current use at the Turing Institute. This is only one possible application of the system, experience with alternative implementations would be very useful.

Certainties

A certainty is a *basic certainty* or an *ordered tuple of certainties,* A basic certainty is a pair *ArJ* where *A* is an atom and J is its *justification,* or the empty certainty []. A justification is a conjunction of literals. It is convenient to represent a certainty as an ordered tuple of trees where the leaves of the trees are basic certainties. An example is shown in Figure 1 where an english "translation" of the certainty is provided. In the following we shall refer to this certainty or its translation indifferently. The application domain is that of claiming travel expenses for trips by car for someone employed in Holland.

The certainty illustrated is the certainty of the atom *entitled to (john, 120)* and can be regarded as a structure of assumptions that are necessary to establish the truth of the assertion. These assumptions are:

(1) The trip was in Holland.
justified by the fact that John travelled Less than 300km.

(2) John did not travel by boat (unjustified)

(3) It would cost less to travel by car than by plane
This assumption rests in turn on the assumption

that the trip was in Holland (see (1) above), and the assumption that it costs less to travel by car in Holland than by plane.

John is entitled to 120 guilders
subject to:
The trip was in Holland
justified by
the distance was less than 300km
John didnt travel by boat
(default assumption)
It cost more to travel by car than plane
justified by
The trip was in Holland
(see above,..)
The plane costs more *in* Holland
(default assumption)

The English translation of a certainty
figure 1

This example has been chosen to illustrate both the general nature of certainties and that our system can be applied to domains where a numerical measure of certainty would be inappropriate. We now turn our attention to the combination of certainties, and the possibility of interaction between certainties.

Combination of certainties

Our default combination function for certainties is concatenation. Given certainties $<c_1,\ldots,c_{t_1}>$, ..., $<c_{t_{m-1}+1},\ldots,c_n>$ the combination function combines these to give $<c_1, \cdots, c_n>$. This reflects a default assumption that there is no interaction between different assumptions, similar in some ways to the assumptions of conditional independence of events made by Prospector (Gaschnig 1980). It is possible for different assumptions to interact and our system provides an elegant mechanism for describing such interaction. We illustrate with an example based on Figure 1. Let us assume that we wish to combine that certainty with the following one

The trip was abroad justified by *ths length of ths trip was mors than 3 days*

These certainties are incompatible, since a trip cannot both be in Holland and abroad (by definition). The certainty function can be set up to provide the combined certainty with value *false*. In our current implementation this can be done in a general way by specifying via a meta-logical assertion that the values *holland* and *abroad* are exclusive for the predicate *location.* In general our implementation allows a threshold to be established for certainties, and the proof mechanism rejects all solutions which fall below the threshold.

We wish to emphasise that the plausible reasoning mechanism can handle interactions or constraints between certainties in a knowledge based way. Constraints can be explicitly represented and reasoned with, and suitable explanations supplied to the user. This is not possible with a system using numeric certainties.

4. Implementation

The mechanism described above has been implemented as part of an expert system shell, called YAPES, written in Prolog.YAPES is an interpreter for logic programs, either in normal Horn clause logic or in Horn clause logic extended with the judgemental reasoning mechanism. The shell contains all the normal facilities for explanation and justification, implemented via meta-logical predicates. In addition the YAPES interpreter can call the underlying Prolog interpreter if desired.

We should note finally that the use of numeric certainty factors is not ruled out in our system. The interval CO, 1] (a lattice whose order is total) used by Shapiro (Shapiro 1963) is an obvious candidate. More interesting perhaps is a combination of numeric and non-numeric certainties. This is the idea advocated by Keynes (Keynes 1921 pp 20-40) in his Treatise on Probability, where he asserted that some though not all probabilities were inherently non-comparable.

5. Conclusion

We have presented a system for plausible reasoning which is a conservative extension of the Horn clause subset of first order logic. We have argued that this system can provide the clear semantics, ease of explanation and expressive power necessary in expert systems.

Our approach embeds the plausible reasoning mechanism within logic, and in particular logic programming systems. This has the advantage that all the normal debugging tools can be used immediately, and provides a smooth interface between "normal" programs and programs with uncertainties. This work can be seen as an extension of the work of Shapiro (Shapiro 1963) in three ways

(1) Certainties can be non-numeric but also amenable to combination and comparison.

(2) The scope of certainty functions has been extended to ordered sequences of certainties rather than multisets. This allows the certainty function to distinguish between literals in a clause.

(3) A fixpoint semantics has been presented for continuous certainty functions, This provides a computational characterisation of the behaviour of the system.

The system can be considered as performing a condensation of a proof. In Horn clause logic the condensation is either true or false; using numeric certainties extends this to the set (0.1]. With a lattice as our certainty structure the condensation can be as large as the proof itself, although normally we are concerned with the assumptions that made during the proof.

From this perspective the system can be seen as performing a top-down version oftruth maintenance (Doyle 1979). We are currently investigating the extension of the allowable certainty functions to provide combination functions for procedures rather than individual clauses. This will allow for proof condensation at the level of all proofs for a goal, rather than for single proofs. In this way meta-level statements such as "if this goal can be proved in these two ways then..." can be represented.

6. References

[1] Doyle, J." A truth maintenance system." Artificial Intelligence 12 (1979) 231-272.

[2] van Emden, M.H. and R.A.Kowalski, "The semantics of predicate logic as a programming language." JACM 23:4 (Oct) (1976) 733-742

[31 Gaschnig, J. "Development of uranium exploration models for the Prospector consultant system," Final Report, SRI International, Menlo Park.California, March 1980.

[4] Keynes, J.M. A Treatise on Probability. London, England: MacMillan,1921.

[5] Shaplro.E.Y." Logic programs with uncertainties" In Proc. WCAI-83. Karlsruhe, West Germany, August, 1983, pp. 529-532.