# META-INTERPRETERS FOR RULE-BASED REASONING UNDER UNCERTAINTY

by

**Shimon Schocken**

Leonard N. Stern School of Business
Information Systems Department
New York University
40 West 4th Street
New York, NY 10003

and

**Tim Finin**

Unisys Paoli Research Center
Unisys Corporation

**July 1989**

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

**Working Paper Series**

CRIS #210
STERN #89-69

The authors thank the editor and three anonymous referees for their thoughtful comments.

## Abstract

One of the key challenges in designing expert systems is a credible representation of uncertainty and partial belief. During the past decade, a number of rule-based belief languages were proposed and implemented in applied systems. Due to their quasi-probabilistic nature, the external validity of these languages is an open question. This paper discusses the theory of belief revision in expert systems through a canonical belief calculus model which is invariant across different languages. A meta-interpreter for non-categorical reasoning is then presented. The purposes of this logic model is twofold: first, it provides a clear and concise conceptualization of belief representation and propagation in rule-based systems. Second, it serves as a working shell which can be instantiated with different belief calculi. This enables experiments to investigate the net impact of alternative belief languages on the external validity of a fixed expert system.

# 1 Uncertainty and Expert Systems

The ability to model uncertainty and belief revision is now considered a key challenge in designing credible expert systems. Regardless of whether the domain of expertise is medical diagnosis, venture capital, or oil exploration – human experts have to cope with uncertain data and inexact decision rules. Moreover, it is now an established fact that humans, laymen and experts alike, are very poor *intuitive* statisticians (Tversky and Kahneman, [34]). Specifically, human judgement under uncertainty is often irrational, to the extent that rationality is equated with the axioms of utility theory and subjective probability.

There have been several attempts to represent uncertainty and belief revision within the rigid framework of logic, with Carnap's (1954) *inductive logic* [5] being the most seminal treatise on the subject. Notwithstanding its significant philosophical contribution, inductive logic was not meant to serve as a practical modelling framework. And yet two decades later, Carnap's work on the theory of confirmation became the motivation for the *certainty factors* model – a popular belief calculus which was first implemented in 1976 in the MYCIN medical diagnosis system (Shortliffe, [30]). Since then, a wide variety of belief calculi and non-categorical inference methods were developed and implemented by researchers and practitioners. By and large, these methods can be classified into two categories: probabilistic, and quasi-probabilistic.

**Probabilistic methods** include such models as Bayes networks (Pearl, [20]), influence diagrams (Howard and Matheson, [14]), and the Dempster-Shafer theory of evidence (Shafer, [27]). These models enjoy a solid theoretical foundation; they are either consistent with the axioms of probability theory, or they extend it in a clear and explicit manner, as in the case of the Dempster-Shafer model. However, it is now well understood that the marriage between logical inference and probabilistic inference is rather problematic. First, it was shown by Heckerman [13] and other authors that the modular structure of the rule-based architecture is generally inconsistent with the wholistic nature of a joint distribution function. Second, probabilistic inference in a rule-based architecture was shown to be $NP$-hard (Cooper, [9]).

**Quasi-probabilistic** belief-calculi are only partially consistent with the ax-

ioms of subjective probability. These calculi include MYCIN's certainty factors model (Shortliffe, [30]), the ad-hoc Bayesian model used in PROSPECTOR (Duda et al, [11]), and an assortment of similar calculi which are essentially isomorphic although they may differ is some details. Following the great popularity of such rule-based shells as EMYCIN, M.1, and AL/X, quasi-probabilistic belief calculi became the de-facto method of handling uncertainty in applied expert systems. And yet the algebraic structure of these pragmatic models is quite obscure, and their limitations and full potential are not well-understood by practitioners and knowledge engineers.

This paper has three purposes. First, it gives a formal description of the structure of a belief calculus and how it may be integrated with the overall architecture of a rule-based system. Second, the paper presents a methodology designed to test the controversial validity of alternative belief calculi. The question of whether or not a belief calculus credibly represents (or improves) human judgement under uncertainty is of utmost importance, and it may be answered only through experimentation with human subjects. In order to run such experiments, one needs a canonical rule-based architecture which can easily accommodate different belief calculi. This leads to the third purpose of the paper, which is the development of a Prolog meta-interpreter, called $SOLVE$, for non-categorical reasoning. $SOLVE$ is useful in that (a) it gives a clear computational definition of a belief calculus, and, (b) it provides a platform for carrying out experiments with alternative belief calculi.

Although the presentation of $SOLVE$ involves a certain degree of logic programming, the major concern of this paper is the *theory* of rule-based belief calculi, and the *software engineering* issues related to their integration with rule-based logic models. The implementation details of $SOLVE$ and related predicates are presented in a separate appendix. This technical material is intended for readers who are interested in Prolog.

# 2 Rule-based inference and Belief Languages

The mathematical and cognitive underpinnings of rule-based (production) systems are well-known, and the reader is referred to Davis and King [22]

and to Newell [18] for extensive discussions. Due to its proximity to first-order predicate calculus, the rational basis of categorical rule-based inference is normally unchallenged. This validity, however, does not extend naturally to applications involving uncertain facts and heuristic inference rules. Under such conditions, rule-based inference becomes an inexact, non-categorical, classification procedure, designed to map an observed phenomenon on a set of one or more explaining hypotheses (Cohen, [7]). This inexact matching algorithm is carried out by applying modus ponens repeatedly to a set of rules of the form *IF e THEN h WITH DEGREE OF BELIEF Bel*, which, from now on, we denote $e \rightarrow h \ \# \ Bel$[1]. The postfix *Bel* is a degree of belief, which, broadly speaking, reflects an expert's confidence in the logical entailment associated with the implication $e \rightarrow h$. The problem, simply put, is this: given the *prior belief* in $h$ and all the *degrees of belief* that parameterize rules and facts which ultimately imply $h$, how does one compute the *posterior belief* in $h$? In expert systems, this is typically accomplished by some sort of a *belief calculus.*

As the rule-based inference-engine processes rules which ultimately imply an hypothesis, a belief calculus is applied to update the posterior belief in this hypothesis. The process normally terminates when the belief in one or more hypotheses exceeds a certain pre-defined cutoff value. Therefore, a non-categorical belief calculus may be viewed as a "scoring" algorithm, a term coined by Cooper [8]. This algorithm accepts a set of inexact rules and a set of uncertain data, and goes on to "score" a set of competing hypotheses, i.e. compute their posterior beliefs. There exist conditions under which the resulting scores are probabilities, but this is not always the case.

According to Shafer and Tversky [28], the building-blocks of a belief language are *syntax*, *calculus*, and *semantics*. In the context of rule-based inference, *syntax* corresponds to the set of degrees of belief which parameterize uncertain facts, inexact rules, and prospective hypotheses. The degrees of belief associated with rules are elicited from domain experts as the knowledge-base is being constructed. The degrees of belief which parameterize observed or suspected pieces of evidence are obtained interactively through consultation. Posterior degrees of belief are computed through a set of operators collec-

[1]throughout the paper, $e$ and $h$ stand for a *piece of evidence* and an *hypothesis*, respectively

tively known as a *belief calculus*. We take the position that the *semantics* of the language consists of either a normative or a descriptive argument which justifies the validity of the syntax and calculus dimensions of the language.

## 2.1 A Canonical Belief Calculus

In order to propagate degrees of belief in a rule-based architecture consisting of uncertain facts and inexact rules, a belief calculus must be capable of handling three generic types of reasoning: *Boolean conditioning*, *sequential propagation*, and *parallel combination*. This section gives canonical definitions of each of these cases. Elsewhere in the paper we present language-dependant instantiations of these models and give their corresponding logic programming implementations.

Let $h$, $e_1$, and $e_2$ be an hypothesis and two pieces of evidence with current beliefs $Bel(h)$, $Bel(e_1)$, and $Bel(e_2)$, respectively. A non-categorical inference mechanism must be capable of computing the posterior belief in $h$, denoted $Bel(h|.)$, in light of any recursive combination of the following generic evidential relationships:

```
Boolean conditioning:      (e1 OR e2) -> h # Bel

                           (e1 AND e2) -> h # Bel


sequential propagation:    e1 -> e2 # Bel1
                           e2 -> h # Bel2

Parallel combination:      e1 -> h # Bel1
                           e2 -> h # Bel2
```

The exact specification of how to compute the posterior belief in $h$ in any one of the above circumstances is precisely *the* definition of a belief calculus. Although the details of such specifications vary greatly across different belief languages, the basic structure of their underlying calculi is quite invariant.

This observation leads to the notion of a *canonical belief calculus*, whose three components are described next.

Boolean Conditioning: Consider the rules ($e_1$ or $e_2$) $\rightarrow$ $h$ $\#Bel_1$ and ($e_1$ and $e_2$) $\rightarrow$ $h$ $\#Bel_2$. The degrees of belief $Bel_1$ and $Bel_2$ represent the strengths of the rules if both $e_1$ and $e_2$ are known to be certain. But what if one or both of these pieces of evidence is uncertain? In such cases, the belief calculus first computes the current belief associated with the premise of each rule, i.e. $Bel((e_1\ and\ e_2))$ and $Bel((e_1\ or\ e_2))$. Technically speaking, this computation is carried out through the template functions $F\_and$ and $F\_or$, respectively:

$$Bel((e_1\ and\ e_2)) = F\_and(Bel(e_1), Bel(e_2)) \tag{1}$$

$$Bel((e1\ or\ e2)) = F\_or(Bel(e_1), Bel(e_2)) \tag{2}$$

Once the current belief in a rule's premise is established trough Boolean conditioning, the posterior belief in the rule's conclusion can be computed through sequential propagation.

Sequential Propagation: Rule-based belief calculi make the implicit assumption that the "actual" degree of belief in a rule has to change when the belief in the rule's premise changes. Specifically, let $e \rightarrow h$ $\#Bel(h, e)$ be a rule specifying that "given $e$ (with certainty), $h$ is implied to a degree of belief $Bel(h, e)$," and let the current belief in $e$ be $Bel(e)$. When a rule-based inference engine operates on a knowledge-base, the premise $e$ might be either (a) a terminal fact whose prior belief $Bel(e)$ is specified by the user, or, (b) an intermediate *sub-hypothesis* whose current belief $Bel(e|.)$ was already computed by the system.

Whichever category $e$ falls in, the "actual" degree of belief in the rule, denoted $Bel'(h, e)$, is computed through a variant of the following *sequential propagation function*, $F\_s$:

$$Bel'(h, e) = F\_s(Bel(e), Bel(h, e)) \tag{3}$$

The function $F\_s$ is monotonically increasing in both variables $Bel(e)$ and $Bel(h, e)$. Therefore, $F\_s$ is sometimes referred to in the AI literature as an "attenuation function," designed to carry over the uncertainty associated with a rule's premise into the uncertainty associated with the rule itself.

Parallel Combination: Let $h$ be an hypothesis with current degree of belief $Bel(h)$ and let $e_1 \rightarrow h$ $\#Bel(h, e_1)$ and $e_2 \rightarrow h$ $\#Bel(h, e_2)$ be two rules that bear evidence on $h$ independently. The combined, posterior belief in $h$ in light of $\{e_1, e_2\}$ is given by the following binary *parallel combination function*, $F\_p$:

$$Bel(h, \{e_1, e_2\}) = F\_p(Bel(h), Bel(h, e_1), Bel(h, e_2)) \tag{4}$$

(it is implicitly assumed that $Bel(h, e_1)$ and $Bel(h, e_2)$ were already attenuated by $F\_s$). In order to free the inference process from order and clustering effects, the function $F\_p$ is normally required to be commutative and associative. If these requirements are satisfied, the binary $F\_p$ function can be extended recursively to an $n$-ary parallel combination function. The details of this extension are straightforward.

We now proceed to describe the $CF$ calculus and the likelihood-ratio Bayesian calculus. These models are presented verbatim, and no attempt is made here to either defend their cognitive appeal or argue for or against their normative justification. Such analyses were carried out by Adams [1], Heckerman [13], Grosof [12], Schocken and Kleindorfer [25], and other authors.

## 2.2 The Certainty Factors Language

Following its initial implementation in MYCIN, the certainty-factors calculus has evolved into several forms, all of which may be easily incorporated into the architecture described in this paper. The calculus discussed here adheres to the original model, described in detail by Buchanan and Shortliffe [4].

In the additive $CF$ syntax, a diagnostic rule of the form $e \rightarrow h$ $\#CF(h|e)$ means that $e$ increases the belief in $h$ by the magnitude $CF(h|e)$ which

varies from -1 to 1. If $e$ is irrelevant to $h$, $CF(h|e) = 0$. The extreme case of $e$ being sufficiently convincing to confirm (disconfirm) $h$ in certainty is modeled through $CF(h|e) = 1$ ($CF(h|e) = -1$). There are basically two types of certainty-factors. The $CF$'s associated with rules (e.g. $pacifist(X) \rightarrow democrat(X)$ #0.9) are elicited from a domain expert when the rule-base is being constructed. The $CF$'s associated with uncertain facts (e.g. $pacifist(joe)$ #0.8) are supplied through consultation.

Boolean Conditioning: Consider the categorical disjunctive rule ($e_1$ *or* $e_2$) $\rightarrow$ $h$ which reads: either one of the two pieces of evidence $e_1$ or $e_2$ (known in certainty) can alone establish the hypothesis $h$. How does one extend this rule to situations in which either $e_1$ or $e_2$ are uncertain? this question is complicated by the observation that the uncertainty associated with these facts is not a probability, but, rather, an abstract measure of human belief. Kahneman and Miller [15] have argued that, under these circumstances, the most reasonable rule for Boolean combination is the one used in fuzzy logic (Zadeh, [38]). This rule, which was implemented in MYCIN, sets the belief in a conjunction (disjunction) to the minimal (maximal) belief in its constituents:

$$cf\_and(CF(e_1), CF(e_2)) = min(CF(e_1), CF(e_2)) \tag{5}$$

$$cf\_or(CF(e_1), CF(e_2)) = max(CF(e_1), CF(e_2)) \tag{6}$$

Sequential combination: The $CF$ associated with the diagnostic rule $e \rightarrow h$ $\#CF(h|e)$ is elicited from a domain expert under the assumption that the premise $e$ is known with certainty. When the belief in $e$ is less than certainty, the $CF$ calculus attenuates the rule's degree of belief through the following sequential propagation function:

$$CF'(h|e) = \begin{cases} CF(h|e) \cdot CF(e) & \text{if } CF(e) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Parallel combination: When two rules $e_1 \rightarrow h$ $\#CF(h|e_1)$ and $e_2 \rightarrow h$ $\#CF(h|e_2)$ bear evidence on $h$ independently, their compound increased

belief in $h$ in light of $\{e_1, e_2\}$ is computed through a binary combination function, defined as follows:

$CF(h|e_1, e_2) =$

$$\begin{cases} CF(h|e_1) + CF(h|e_2) \cdot (1 - CF(h|e_1)) & \text{if both } CF\text{'s are positive} \\ -(|CF(h|e_1)| + |CF(h|e_2)| \cdot (1 - |CF(h|e_1)|) & \text{if both } CF\text{'s are negative} \\ \frac{CF(h|e_1) - CF(h|e_2)}{1 - min\{|CF(h|e_1)|, |CF(h|e_2)|\}} & \text{if the } CF\text{'s have mixed signs} \end{cases}$$

## 2.3 The Bayesian Language

In Bayesian languages, a rule of the form $h \rightarrow e \ \#Bel$ reads: the hypothesis $h$ causes the evidence $e$ with a degree of belief $Bel$. There exist several different interpretation of $Bel$. Some Bayesian systems elicit and propagate conditional probabilities of the form $Bel = P(e|h)$. A more balanced Bayesian design would record not only $P(e|h)$ but also $P(e|\overline{h})$, leading to the two-place degree of belief $Bel = [P(e|h), P(e|\overline{h})]$. Finally, the likelihood-ratio Bayesian syntax consists of likelihood-ratios of the form $Bel = P(e|h)/P(e|\overline{h})$. If the prior-odds on $h$, $P(h)/P(\overline{h})$, is known, then Bayes rule dictates that the presence of $e$ will change the posterior odds on $h$ to $P(h)/P(\overline{h}) \cdot P(e|h)/P(e|\overline{h})$. Hence, the Bayesian syntax is multiplicative, unlike the $CF$ syntax, which is additive.

Recall that our ultimate purpose is to develop a canonical meta-interpreter which can accommodate a wide variety of different belief calculi. With that in mind, we'll focus on a general Bayesian language in which the degree of belief $Bel$ which parameterizes the rule $h \rightarrow e \ \#Bel$ is taken to be the 3-place list $[P(h), P(e|h), P(e|\overline{h})]$. If $e$ is a terminal piece of evidence, the degree of belief in $e$ is taken to be $Bel = P(e)$.

Boolean Conditioning: In quasi-probabilistic Bayesian systems, e.g. PROSPECTOR, the current belief in conjunctions and disjunctions involving uncertain propositions is computed through the same fuzzy logic conventions used in MYCIN:

$$b\_and(P(e_1), P(e_2)) = min(P(e_1), P(e_2)) \tag{8}$$

$$b\_or(P(e_1), P(e_2)) = max(P(e_1), P(e_2)) \tag{9}$$

Sequential propagation: The literature contains several heuristic procedures for Bayesian sequential belief update, e.g. Jeffries rule of conditioning (Shafer, [26]) and PROSPECTOR's interpolation function (Duda et al, [11]). For the sake of brevity, we choose to describe here a simple interpolation function, discussed by Wise [36]. Suppose the knowledge-base contains the rule $h \rightarrow e \ \#[P(h), P(e|h), P(e|\overline{h})]$ and we find out through consultation that the piece of evidence $e$ obtains with the current belief $P(e)$. Before we can calculate the impact of $e$ on the posterior belief in $h$, we attenuate the rule's degree of belief as follows:

$$P'(e|h) = P(e|h) \cdot P(e) + (1 - P(e|h)) \cdot (1 - P(e)) \tag{10}$$

$$P'(e|\overline{h}) = P(e|\overline{h}) \cdot P(e) + (1 - P(e|\overline{h})) \cdot (1 - P(e)) \tag{11}$$

This gives the "actual" rule's degree of belief, $[P(h), P'(e|h), P'(e|\overline{h})]$. Note that (11) is a weighted average of $P(e|h)$ and $P(\overline{e}|h)$, weighted by $P(e)$ and $P(\overline{e})$. (12) is similar.

Parallel combination: Let $h \rightarrow e_1 \ \#Bel_1, \ \cdots \ , h \rightarrow e_n \ \#Bel_n$ be $n$ causal rules with (already attenuated) degrees of belief $Bel_i = [P(h), P(e|h), P(e|\overline{h})]$. The posterior belief in $h$ in light of the evidence $\{e_1, \cdots, e_n\}$ is computed through the following version of (the commutative and associative) Bayes rule:

$$products\_odds = \frac{P(e_1|h)}{P(e_1|\overline{h})} \cdot, \cdots, \cdot \frac{P(e_n|h)}{P(e_n|\overline{h})} \tag{12}$$

$$odds = products\_odds \cdot \frac{P(h)}{P(\overline{h})} \tag{13}$$

The posterior belief in $h$, $P(h|.)$, may be derived through the simple transformation:

$$P(h|.) = \frac{odds}{1 + odds} \tag{14}$$

# 3 On the Validity of Belief Languages

As the previous section illustrates, the $CF$ and the Bayesian languages offer two different representations of uncertainty and partial belief. At the same time, one would hope that the behavior of a CF-based expert system would be compatible with that of a Bayesian expert system, all other things held equal (including the expert and the knowledge-base). This hypothesis can be tested only through experimentation. The design of such experiments and the computational tools which are necessary to support them are discussed in this section.

Consider the familiar problem of rating prospective dates and managing a *little black book*. Suppose a person, denoted hereafter *dater*, wishes to determine whether or not another person is a good match for a blind-date, based on a single telephone conversation. For the sake of simplicity, let's assume that the dater's rationale is represented through the following CF-oriented knowledge-base:

```
nice_voice(X) -> good_looking(X) # 0.4.                    [1]

good_looking(X) or smart(X)                                [2]
              -> date(X) # 0.8.

nice_voice(leslie) # 1.0.                                  [3]

smart(leslie) # 0.7.                                       [4]
```

This knowledge-base has the following interpretation: [1] is a wishful (and inexact) conjecture that blind-daters typically make and then learn that they

should have known better. [2] is an inexact rule of thumb which models the dater's social preferences. [3] is a certain fact about Leslie. S/he sounds good over the telephone. Fact [4] is an inexact estimate of Leslie's IQ.

We see that not unlike other domains of expertise, the dater's "knowledge" and perception of reality are heuristic and subjective, respectively. In the rule-based architecture of [1-4], this non-determinism is represented by the degrees of belief following the # symbol. Note, however, that barring these numbers, [1-4] may be readily translated to a standard logic model. Let's assume that this model is implemented in Prolog, and consider the following query:

```
date(leslie) ?
```

Prolog's response to this query will be the laconic and rather unproductive result "Yes." Under the given semantics, this means: "go ahead and date Leslie." We think that most daters would reject this black and white dichotomy in favor of a finer and more informative matcher. In particular, let's assume that (a) the # degrees of belief in [1-4] were reinstated, and, (b) a certainty-factors oriented meta-interpreter called $SOLVE$ were available. Under these conditions, the original query may be recast as the following meta-query:

```
solve(date(leslie),Bel) ?
```

To which Prolog will answer:

```
Yes, Bel=0.56
```

Like standard Prolog, $SOLVE$ attempts to prove the goal $date(leslie)$, searching for facts and rules which imply this hypothesis categorically. In the process of constructing this proof, however, $SOLVE$ also collects degrees of belief relevant to Leslie and fuses them into $Bel$, the posterior belief in the proposition $date(leslie)$. In a meta-interpreter environment, the $Bel$ variable

is bound and updated on the fly, as a side-effect of the ordinary inference process.

The preceding discussion made the implicit assumption that $SOLVE$ has a built-in belief calculus. In other words, the belief calculus is assumed to be a fixed part of $SOLVE$'s theory. However, in view of Sterling's [31] principles of mixing flavors, it is far more tasteful to define a stand-alone belief calculus, say c, and pass it on to the $SOLVE$ meta-interpreter as a parameter. In this form, the query $solve(h, Bel, c)$ consists of a request to confirm an hypothesis, $h$, and compute its posterior belief, $Bel$, modulo the belief calculus, $c$.

For example, let $cf$ and $b$ be two complex predicates which implement the certainty-factors and the Bayesian calculus, respectively. Assume further that the same dater has specified the degrees of belief which parameterize [1-4] twice, once as certainty factors, and once as conditional probabilities. Now consider the following set of queries:

```
solve(date(leslie),cf,x1) ?

solve(date(pat),cf,y1) ?

solve(date(leslie),b,x2) ?

solve(date(pat),b,y2) ?
```

Suppose that the results of this experiment were $x_1 > y_1$ and $x_2 < y_2$. This would indicate that at least one of the belief languages under consideration fails to capture the human's preferences. Note that excluding the change in the belief calculus and syntax, everything else is kept intact, including Leslie, Pat, the dater's preferences, and the inference-engine. Hence, the experiment measures the net impact of alternative belief language "treatments" on the system's behavior, all other things held equal.

Shafer and Tversky [28] note that there are no formal criteria or general empirical procedures for evaluating probabilistic designs, and go on to conclude that "*the design and analysis of mental experiments therefore represents a*

*challenge to both statisticians and psychologists.*" The validity of this particular experiment must be qualified by two observations: first, the design assumes that the knowledge-base remains constant across different belief languages. Second, if the results of the experiment were indeed incompatible, it would be impossible to say whether this inconsistency was due to the different belief calculi or due to the different sets of degrees of belief. Notwithstanding the external validity of *these* reservations, they don't diminish the value of the experiment. Recall that our objective is to test the wholesome impact of different belief languages on expert systems. This calls for a simultaneous manipulation of both the syntax and the calculus across treatments. Furthermore, in order for this experiment to be internally valid, we must seek a task in which the knowledge-base is sufficiently simple to remain constant across different belief languages.

A within-subject experiment which follows these guidelines was carried out by Schocken [24]. The "experts" were professors and senior Ph.D. students in a decision sciences department. The context was a real-life inference problem taken from the domain of faculty recruiting. Ten resumes of hypothetical prospective candidates were given to each subject, who rank-ordered them in terms of increasing likelihood of potential academic success. Each subject then underwent an elaborate knowledge elicitation procedure administered by the experimenter, who played the role of a knowledge engineer. The result of this interaction was a rule-base which presumably captured the ranking rationale of the human expert. The subjects were then assigned randomly to two groups, I and II.

Subjects in group I were asked to express their degrees of belief in each rule using the certainty factors language, and group II subjects expressed their belief in terms of subjective probabilities. Two months later, the subjects were recalled, and the very same sequel ensued: first, each subject generated a second "human ranking" of the (very same) ten candidates. Next, each subject was presented with the same rule-base that he or she has provided originally, with one exception: the degrees of belief which parameterized the rules were omitted. Finally, the knowledge elicitation treatment was switched: subjects from groups I and II underwent a probabilistic and $CF$ elicitation, respectively. This completed the data gathering stage of the experiment.

At that point, two expert systems (per subject) were constructed, using a $CF$ and an a Bayesian versions of the $SOLVE$ meta-interpreters. The two systems varied only in their dependence on the $CF$ and on the Bayesian calculi. The systems were then fed with the subject's rule-base and the set of ten (encoded) resumes, and went on to compute the rankings of the candidates in terms of certainty factors and posterior probabilities. Note that the two systems were identical in their reliance on the same rule-base, fact-base, and expert. These factors were tightly controlled, varying only the syntax and the belief calculus "treatment."

The data analysis part of the experiment consisted of comparing the various rankings of candidates generated by humans and machines, using standard statistical rank-correlation tests. The correlation between the two "human rankings" (which were spaced two months apart) were used to control for consistent subjects. The correlation between human rankings and machine rankings were used to test a series of hypotheses regarding the descriptive and external validity of alternative belief languages.

The *descriptive validity* of the Bayesian ($CF$) language was estimated through the correlation between the human ranking and the Bayesian ($CF$) machine ranking. The *external validity* of the Bayesian ($CF$) language was estimated through the correlation between the Bayesian ($CF$) ranking and the pooled human ranking of those subjects who were professors at the decision sciences department. These professors were actively involved in real hiring decisions, and their pooled ranking was therefore viewed as a gold standard against which other rankings could be pitted.

The results of the experiment were somewhat surprising. The $CF$ and the Bayesian language scored highly and similarly in terms of descriptive validity. At the same time, the Bayesian language outperformed the $CF$ language in terms of external validity. The details of the experiment and the results are reported in Schocken [24], and are somewhat irrelevant to the present paper. Our chief concern here is research methodology, and, in particular, the design of the meta-interpreters which enable such experiments.

# 4 Meta-Interpreters as Logic Models of Expert Systems

The meta-interpreter presented below was developed to support experimentation with alternative belief languages. The objective was to develop a computational environment which (a) simulates a standard rule-based inference algorithm, and, (b) allows a great deal of design flexibility with respect to creating and modifying alternative belief calculi. In the process of developing these tools, we became aware of a paper by Sterling [31] describing the analogy between Lisp Flavors and Prolog meta-interpreters. Sterling's paper provides an elegant theoretical framework within which our work can be described in terms of mixing flavors.

The basic notion of *logic programs with uncertainties* is due to a paper of this title by Shapiro [29]. In a logic program with uncertainty, rules and facts are parameterized by some sort of a degree of belief. The inference algorithm is extended to compute posterior beliefs in goals as a side-effect of standard reasoning. Belief computations can be performed either within the logic program itself (e.g. Clark and McCabe, [6], Alvey et al, [2]), or at higher, meta-level of interpretation (Shapiro, [29]).

A Meta-interpreter is an interpreter of a language written in the same language. In Prolog, meta-interpreters have proven to be particularly useful in building expert system shells. The basic idea is that Prolog is already a very capable first-order inference-engine; turning this raw power into a full-featured shell is basically a matter of adding functionalities to the standard language. For the sake of modularity, this is best accomplished by creating specialized meta-interpreters and enhancing them incrementally (Sterling, [31]).

Prolog meta-interpreters were developed to add a number of essential capabilities found in most commercial expert system shells. For example, Hammond and Sergot [19] extended the basic inference-engine with a "query the user" facility which obtains missing information through interactive consultation. Sterling and Lalee [32] developed techniques to explain the system's line of reasoning. A number of authors, e.g. Dincbas [10] and Pereira [21], have

shown how the fixed control structure of Prolog can be short-cut and modified to suit various inferential needs. Baldwin and Monk [3] developed a meta-interpreter for inexact reasoning based on the Dempster-Shafer model.

Wirth's [35] software engineering principle of *programs = algorithms + data structures* is well-known. The expert systems analogy of this principle is as follows:

```
expert          =    knowledge      +    inference          [5]
system               base                mechanism
```

This paper takes the modularity principle one step further, achieving what may be described symbolically as:

```
inference       =    inference      +    belief             [6]
mechanism            engine              calculus
```

The *inference engine* was implemented through the $SOLVE$ meta-interpreter, which is completely independent of the details of the *belief calculus* and the syntax of the *knowledge base*. This modularity facilitates comparative experimentation with alternative belief calculi, leaving the rest of the system intact. For example, consider an experiment designed to compare the recommendations generated by a $CF$-based and a Bayesian-based systems. This experiment will make use of two logic models which may be described symbolically, as follows:

```
inference       =    SOLVE          +    CF                 [7]
mechanism1                               calculus

inference       =    SOLVE          +    Bayesian           [8]
mechanism2                               calculus
```

The remainder of this section presents logic models of the two major components of [5] – the *knowledge base* and the *inference mechanism*. The glue that

integrates these components (the + sign) is also discussed. Logic models of the certainty factors and the Bayesian calculi are presented in a separate Appendix. Due to the modularity of *SOLVE*, the details of these calculi have no impact on the overall architecture, and we therefore delay their discussion to a later stage.

## 4.1 A Canonical Knowledge-base

This section presents a logic model of a non-categorical knowledge-base. From a logic modeling perspective, this knowledge-base is simply a set of predicates. In order to merge these predicates with *SOLVE*'s theory, however, some parsing and pre-processing must take place. The discussion of these issues serves to highlight the snytachtical differences of the *CF* and the Bayesian languages, and the ease by which this pluralism is accommodated by the *SOLVE* meta-interpreter.

Going back to the dating example, consider the following subset of an hypothetical, CF-oriented, knowledge-base:

```
/* rule-base */

rich(X) -> date(X) # 0.2.

age(X,Age) and Age>18 and Age<35 -> date(X) # 0.3.

good_looking(X) -> date(X) # 0.8.

salary(X,Salary) and Salary>75000 or
       parent(X,Parent) and salary(Parent,SalaryP) and
                           SalaryP>150000
                -> rich(X) # 0.9.

/* fact base */

age(nicky,28).
```

```
parent(nicky,bob).
salary(bob,160000).
salary(nicky,20000) # 0.8.
good_looking(pat) # 0.95.
salary(pat,0).
age(pat,24).
potential_date(nicky).
potential_date(pat).
```

Figure 1

In order to merge this knowledge-base with the meta-interpreter, we have to convert its rules and facts into a *generic clausal form* consistent with Prolog's syntax. The strategy taken here is to (a) convert inexact rules of the form $e \rightarrow h \ \#Bel$ into the generic clause $(h, e, Bel)$, and, (b) convert uncertain facts of the form $e \ \#Bel$ into the generic clause $(e, true, Bel)$. The generic clause is important because this is the only data structure that $SOLVE$ recognizes.

Since the direction of rules and the semantics of degrees of belief vary across different belief languages, each language requires a specialized parser. The remainder of this section presents a certainty-factors parser and a Bayesian parser. The section concludes with some general remarks on other functions which may be incorporated in more sophisticated parsers.

A Certainty-factors Parser: A knowledge-base with certainty-factors is translated into generic clauses through the following parser:

```
parse(H,E,Bel) :- (E -> H # Bel).          [9]
parse(E,true,Bel) :- (E # Bel).           [10]
parse(E,true,1) :- E.                     [11]
```

This code reads as follows: [9] matches the rule $E \rightarrow H \ \#Bel$ with the clause $(H, E, Bel)$. [10] matches the uncertain fact $E \ \#Bel$ with the clause $(E, true, Bel)$. Finally, certain facts of the form $E$ (with no attached degrees

of belief) are defaulted by [11] to the clause $(E, true, 1)$ which reads: *E is true with certainty.* The latter convention allows us to freely mix certain and uncertain facts in the same knowledge-base, and, at the same time, relieves us from the tedium of assigning a 1.0 degree of belief to such certain facts as $parent(nicky, bob)$. Instead, we let the system take care of this nuisance as a side-effect of parsing.

A Bayesian Parser: Recall that in our Bayesian language, the degree of belief *Bel* which parameterizes the rule $h \rightarrow e \ \#Bel$ is taken to be the 3-place list $[P(h), P(e|h), P(e|\overline{h})]$. If $e$ is a terminal piece of evidence, the degree of belief in $e$ is taken to be $Bel = P(e)$. With that in mind, the Bayesian parser is defined as follows:

```
parse(H,E,Bel) :- (H -> E # Bel).                [12]
parse(E,true,Bel) :- (E # Bel)                   [13]
parse(E,true,[0.9999,1,1]) :- E.                 [14]
```

The meaning of [12] and [13] is identical to their corresponding meaning in the $CF$ parser, but note that the direction of the rule in the right-hand side is reversed. When the parser detects a certain fact through [14], it defaults its prior probability to 0.9999. The difference between this and the more plausible 1 is due to an uninteresting technical detail.

Similarly to the $CF$ parser, the role of [12-14] is to translate rules and facts into the generic clause $(H, E, Bel)$ which is recognizable by the $SOLVE$ meta-interpreter. Note that no attempt is made here to unpack compound degrees of belief into their three individual components. This task is left where it belongs – the belief calculus level. This again illustrates how a modular design can relieve the inference-engine from unnecessary technical clutter.

Other Uses of Parsers: Thoughtful use of parsers and pre-processors allow the designer to modify the syntax of a belief language and its corresponding knowledge-bases without tinkering with the rest of the system. For example, suppose we wish to leave the $CF$ calculus intact, and, at the same time, elicit degrees of belief that vary from -100 to 100 instead of -1 to 1 (this is

normally done by most $CF$ knowledge engineers). This leads to rules and facts of the form:

```
likes(X,sushi) -> date(X) # -10.
nationality(X,japan) -> likes(X,sushi) # 90.
nationality(tomo,japan).
```

Following the standard $CF$ requirement that degrees of belief be restricted to the interval $[-1,1]$, we can pre-process the knowledge-base as follows:

```
parse(H,E,Bel)  :-  (E -> H # Bel1),
                    Bel is Bel1/100.
parse(E,true,Bel)  :-  (E # Bel1),
                       Bel is Bel1/100.
parse(E,true,1)  :-  E.
```

One can easily envision other useful applications of $PARSE$ beyond this trivial example. In PROSPECTOR, for example, there is a provision for representing belief in evidence through qualitative terms, e.g. *occasional*, *rare*, etc. Those statements are then transformed into probabilities, e.g. 0.1 and 0.01, respectively (Duda et al, [11]). In a similar vein, Lichtenstein and Newman [16] concluded empirically that verbal descriptions of uncertainty may be mapped on ranges of probabilities. These verbal-numeric mappings can be made explicit as a side-effect of parsing, through the following syntactical sugar:

```
parse(E,true,Bel) :- (E # Bel_text),
                     translate(Bel_text,Bel).

translate("occasional",0.1).
translate("rare",0.01).
etc.
```

To sum up, the parser shields the inference-engine from the syntactical idiosyncrasies of the underlying belief language. This separation enables us to

elicit and represent rules and facts in a variety of forms, and, at the same time, process them through a canonical inference mechanism which operates on a collection of generic clauses of the form $(H, E, Bel)$. The details of this mechanism are presented in the next section.

## 4.2 A Canonical Inference Mechanism

We assume that any knowledge-base, regardless of how complex, is a recursive union of the four generic inferential structures depicted in the following figure:

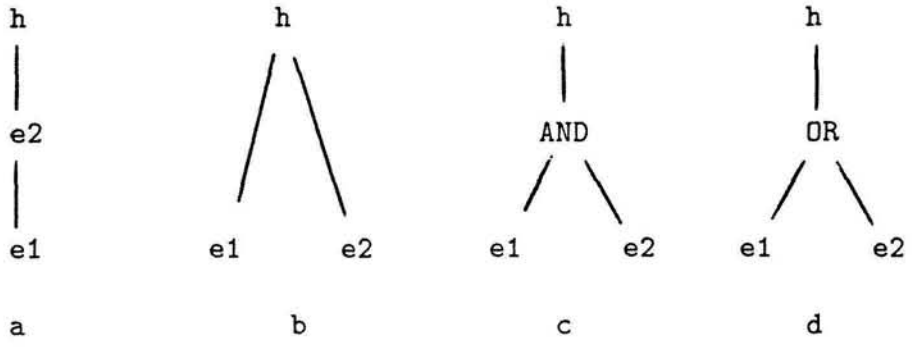


Figure 2

In a non-categorical knowledge-base, these structures are also parameterized by degrees of belief. Hence, the $SOLVE$ meta-interpreter (the inference engine) must employ a belief calculus which specifies how to combine degrees of belief in the cases of (a) sequential propagation (b) parallel combination (c) conjunctive conditioning, and (d) disjunctive conditioning. In section 2.1, the functions which specify this calculus were denoted $F\_s$, $F\_p$, $F\_and$, and $F\_or$, respectively. Since we want our meta-interpreter to be independent of the calculus, we pass these functions as parameters to $SOLVE$'s theory. The $SOLVE$ predicate if defined as follows:

```
solve(true,1,F_s,F_p,F_and,F_or) :- !.                          [15]

solve((H1 or H2),Bel,F_s,F_p,F_and,F_or) :-                     [16]
            solve(H1,Bel1,F_s,F_p,F_and,F_or),                  [17]
            solve(H2,Bel2,F_s,F_p,F_and,F_or),                  [18]
            apply(F_or(Bel1,Bel2),[Bel]).                       [19]

solve((H1 and H2),Bel,F_s,F_p,F_and,F_or) :-                    [20]
            solve(H1,Bel1,F_s,F_p,F_and,F_or),                  [21]
            solve(H2,Bel2,F_s,F_p,F_and,F_or),                  [22]
            apply(F_and(Bel1,Bel2),[Bel]).                      [23]

solve(H,Bel,F_s,F_p,F_and,F_or) :-                              [24]
             parse(H,_,Belp),!,                                 [25]
             bagof(Belx,                                        [26]
                    (parse(H,E,Bel_rule),                       [27]
                     solve(E,Bel_e,F_s,F_p,F_and,F_or),         [28]
                     apply(F_s,[Bel_e,Bel_rule,Belx)),          [29]
                   Bels),                                       [30]
                apply(F_p,[Belp,Bels,Bel]).                     [31]

solve(E,1,F_s,F_p,F_and,F_or) :- E,!.                           [32]
solve(E,0,F_s,F_p,F_and,F_or).                                  [33]
```

The base-fact [15] of $SOLVE$, which is ground, assigns a belief of 1 to the constant hypothesis *true*. The subsequent handling of Boolean conditioning in [16-23] is self-explanatory. In [25], $PARSE$ is used to check if the hypothesis $H$ is present in the knowledge-base, and, if so, to bind *Belp* to its prior belief. The $BAGOF$ predicate accomplishes a few things. First, it looks (through parsing) [27] for all the rules $E \rightarrow H$ *Bel_rule* whose conclusion is $H$ . For each such rule, $SOLVE$ is applied recursively to compute the posterior belief in the premise $E$, yielding *Bel_e* [28]. This current belief, in turn, is used by *F_p* to attenuate the original rule's degree of belief, *Bel_rule*, into *Belx* [29]. Attenuated degrees of belief are strung together (via $BAGOF$) into the list *Bels* [30].

The "punch line" of $SOLVE$ is [31]. When we get to this point, the list $Bels$ consists of all the attenuated degrees of belief associated with all the rules whose conclusion is $H$. Since this list is constructed recursively, $Bels$ encapsulates all the evidence that $SOLVE$ drew from all the reasoning chains whose ultimate conclusion is $H$. At that point, the parallel combination function $F\_p$ is applied to fuse this information with the prior belief $Belp$, yielding the ultimate outcome of $SOLVE$, i.e. the posterior belief in $H$, $Bel$.

To sum up, $solve(H, Bel, F\_s, F\_p, F\_and, F\_or)$ implements an exhaustive depth-first search, pruning all the rules and facts which bear evidence on $H$, either directly or indirectly. As a side-effect of this process, the program computes the posterior belief in $H$ modulo the *variable* belief calculus $< F\_s, F\_p, F\_and, F\_or >$. When $SOLVE$ branches horizontally, $F\_p$ is used to combine the degrees of belief originating from rules whose direct conclusion is $H$. When $SOLVE$ backtracks from a vertical recursive call, $F\_s$ is used to synthesize the belief committed to $H$ from lower-levels of reasoning. If a Boolean "fork" is encountered, either $F\_and$ or $F\_or$ are invoked to compute the posterior belief coming out of the fork.

Note again that the predicates $< F\_s, F\_p, F\_and, F\_or >$ are left unspecified. This is done in purpose, in order to highlight the modularity and top-down design of $SOLVE$. In a separate appendix, we give logic models of the CF calculus, $< cf\_s, cf\_p, cf\_and, cf\_or >$, and the Bayesian calculus, $< b\_s, b\_p, b\_and, b\_or >$, and explain how they may be bound to $SOLVE$'s theory.

In the context of larger logic model, the fully instantiated $SOLVE$ meta-interpreter may be viewed as simply yet another predicate. Therefore, one can blend $SOLVE$ with other predicates in a variety of different ways. For example, assume that the dater from Section 4.1 wishes to print a list of dating candidates sorted in decreasing order of composite attractiveness. This version of the *little black book* may be created through the following ($CF$-oriented) predicate:

```
create(Book) :-
        bagof((X,Rating),
                (potential_date(X),
                 solve(date(X),Rating,cf_s,cf_p,cf_and,cf_or)),
                 Xs),
        sort(Xs,Book).
```

Given the database listed in Figure 1 (consisting of only two potential dates), the goal *create*(*Book*) will yield the response:

```
Book = [(pat,0.832),(nicky,0.426)].
```

The casual nature of the dating example should not obscure the underlying seriousness of the $SOLVE$ meta-interpreter. Consider, for example, a medical diagnosis application. In this context, potential dates and their perceived characteristics correspond to prospective diseases and symptom manifestations, respectively. Sub-hypotheses, like $rich(X)$, correspond to clinical syndromes or intermediate diagnoses. Dating rules are analogous to text-book medical knowledge and heuristic inferences of experienced experts. Under this interpretation, the evaluation of a prospective date is analogous to the diagnosis of a certain patient, and the goal *create*(*Book*) would probably become *rank*(*Diseases*). Given a certain knowledge-base and a set of observed symptoms, this goal gives a list of all the potential diseases that this patient might have, in decreasing order of likelihood.

# 5 Conclusions and Future Research

The validity of alternative belief languages can be investigated in two different and complementary methodologies: analytical, and empirical. The *analytic approach* is chiefly concerned with comparing belief calculi to well-known normative criteria, e.g. probability theory or predicate logic. This line of research leads quite clearly to the realization that, not unlike the humans that they attempt to model, all rule-based belief calculi contain varying degrees

of normative violations. Nonetheless, the extent of these violations is not well understood, and the sensitivity of the system's advice to such violations is still an open question.

In spite of their normative deficiencies, rule-based belief calculi are widely-used in commercial expert systems. Moreover, it might be that a careful design of the underlying knowledge-base might ensure that normative violations are kept to a minimum. With that in mind, there is a crucial need for an *empirical approach* to investigating the external validity of alternative belief calculi. This line of research simulates real settings in which the expertise of human subjects is elicited and represented via different belief languages. The experiments then pit the systems' recommendations with (a) the judgment of the humans that they claim to model, and (b) an external norm – a gold standard – which may be a either a credible expert opinion, or, preferably, the actual "true state of the world."

The empirical approach is very novel. There exist only a few empirical studies which pit alternative belief languages, e.g. Mitchell [17], Yadrick et al [37], Wise [36], and Schocken [24]. These studies attempt to understand the conditions under which one belief language performs better than another. Therefore, they have important prescriptive implications on knowledge engineering.

One limitation that inhibited more research in this direction has been a lack of a common benchmark environment. Such an environment ought to simulate the mechanics of rule-based inference and, at the same time, allow a great deal of design flexibility in terms of tinkering with alternative belief calculi without touching the rest of the system. We feel that Prolog and meta-interpreters like $SOLVE$ provide a very flexible environment in which such experiments can be carried out. We hope that these and similar tools will promote further research on the questionable validity of rule-based inference under uncertainty.

# 6 Appendix: Implementation Details

## 6.1 Knowledge Representation

Consider the set of rules and facts depicted in Figure 1. Let us assume that this knowledge-base is physically stored and maintained in a standard text file called $KBASE$. How can we merge the contents of this file with a standard Prolog database? ideally, we would like to simply prove the goal $consult(kbase)$. This, however, won't work, since the $KBASE$ contains non-standard Prolog terms. This difference may be resolved through the following predicate:

```
define_syntax :-  op(255,xfy,->),
                  op(254,xfx,#),
                  op(254,xfx,or),
                  op(253,xfx,and).
```

Each application of the system predicate $op(P, A, T)$ defines the token $T$ as a new, non-standard Prolog operator. The precedence and associativity properties of $T$ are given by $P$ and $A$, respectively. The actual values of these arguments vary from one Prolog implementation to another and are of little interest.

## 6.2 The $CF$ Calculus

The $CF$ calculus presented in Section 2.2 is implemented below through the four predicates $< cf\_and, cf\_or, cf\_s, cf\_p >$:

```
/* CF Boolean conditioning functions.
input: Bel1, Bel2
output: Bel */

cf_and(Bel1,Bel2,Bel) :- min(Bel1,Bel2,Bel).
```

```
cf_or(Bel1,Bel2,Bel) :- max(Bel1,Bel2,Bel).

/* CF sequential propagation function.
input:     Bel_e: current belief in premise,
        Bel_rule: rule's degree of belief,
output:      Bel: attenuated rule's degree of belief  */

cf_s(Bel_e,Bel_rule,Bel) :- max(0,Bel_e,Bel_max),
Bel is Bel_rule * Bel_max.

/* Binary CF parallel combination function.
input:  X,Y: two CF's of independent rules which render
             evidence to the same hypothesis)
output:   Z: combined CF  */

cf_p_2(X,Y,Z) :- X>=0, Y>=0,
                 Z is X+Y*(1-X),!.

cf_p_2(X,Y,Z) :- ((X=<0,Y>=0) ; (X>=0,Y=<0)),
                  abs(X,A), abs(Y,B), min(A,B,C),
                  Z is (X+Y)/(1-C),!.

cf_p_2(X,Y,Z) :- X<0, Y<0,
                 abs(X,A), abs(Y,B),
                 Z is -(A+B*(1-A)),!.

/* n-ary CF parallel combination function */
input:    Xs: a set {cf(h|e1),...,cf(h|en)}
output:  Bel: cf(h|e1,...,en)

cf_p(_,[],0).
cf_p(_,[X|Xs],Bel) :- cf_p(_,Xs,Bel_Xs),
                      cf_p_2(X,Bel_Xs,Bel).
```

The explicit omission of the first variable in the *cf_p* function underscores the fact that the *CF* language ignores prior beliefs. This can be seen clearly in

the definition of the base-fact of $cf\_p$, which models the *state of insufficient reason* (Savage, [23]). This case, which is characterized by an empty set of degrees of belief, causes $cf\_p$ to assign a posterior belief of 0 to the hypothesis in question. This is consistent with the additive $CF$ philosophy, in which the absence of any relevant evidence on $h$ causes the belief in $h$ to neither increase nor decrease. In a Bayesian language, one would normally model this case by setting the posterior belief in the hypothesis to its prior belief.

## 6.3 The Bayesian Calculus

The Bayesian calculus presented in Section 2.3 is implemented in below through the four predicates $< b\_and, b\_or, b\_s, b\_p >$:

```
/* Bayesian Boolean Conditioning: */

b_and(Bel1,Bel2,Bel) :- min(Bel1,Bel2,Bel).
b_or(Bel1,Bel2,Bel) :- max(Bel1,Bel2,Bel).

/* Bayesian sequential propagation function.
    Input:   Bel_e: P(e)
             P0: P(h),
             Q1: P(e|h),
             Q2: P(e|h),
     Output: P0: P(h),
             P1: P'(e|h),
             P2: P'(e|h)  */

b_s([P0,Q1,Q2],Bel_e,[P0,P1,P2]) :-
                 P1 is Q1*Bel_e + (1-Q1)*(1-Bel_e),
                 P2 is Q2*Bel_e + (1-Q2)*(1-Bel_e).

 /* Bayesian Parallel combination function:
input:   Bels: a set of 3-place degrees of belief {Bel1,...,Beln}
        Prior: prior belief in the hypothesis, P(h)
Output:     P: posterior belief in the hypothesis, P(h|e1,...,en)
```

```
b_p([Prior|_],Bels,P) :-
              mult(Bels,Product_Odds),
              Odds is (Prior/(1-Prior)) * Product_Odds,
              P is Odds/(1+Odds).

mult([],1).
mult([[X1,X2]|Xs],Product) :- mult(Xs,Bel_Xs),
                              Product is (X1/X2)*Bel_Xs.

/* A more efficient, tail-recursive version of MULT
can be defined as follows: */

mult(List,Product) :- mult1(List,1,Product).
mult1([],Accumulator,Accumulator).
mult1([[_,X1,X2]|Xs],Acumulator,Product) :-
                       NewAccumulator is Accumulator*X1/X2,
                       mult1(Xs,NewAccumulator,Product).
```

## 6.4 Miscellaneous predicates

The following predicates complete the definition of the *SOLVE* meta-interpreter:

```
apply(Predicate,Args) :- Predicate=..PredList,
                         append(PredList,Args,GoalList),
                         goal=..GoalList,
                         call(Goal).

bagof(X,G,_) :- asserta(found(mark)),G,asserta(found(X)),fail.
bagof(_,_,L) :- collectFound([],M),!,L=M.
collectFound(Lin,Lout):- getNext(X),!,collectFound([X|Lin],Lout).
collectFound(Lin,Lin).
getNext(X) :- retract(found(X)),!,not(X==mark).

max(X,Y,X) :- X>=Y.
```

```
max(X,Y,Y) :- X<Y.

min(X,Y,X) :- X=<Y.
min(X,Y,Y) :- X>Y.

abs(X,Z) :- X<0, Z is -(X).
abs(X,X) :- X>=0.

append([],Ys,Ys).
append([X|Xs],Ys,[X|Zs]) :- append(Xs,Ys,Zs).

/* the following predicate sorts a list of pairs [X,Y]
   in decreasing order of Y */

sort([X|Xs],Ys) :- sort(Xs,Zs), insert(X,Zs,Ys).
sort([],[]).
insert([X1,X2],[],[[X1,X2]]).
insert([X1,X2],[[Y1,Y2]|Ys],[[Y1,Y2]|Zs]) :-
            X2>Y2, insert([X1,X2],Ys,Zs).

insert([X1,X2],[[Y1,Y2]|Ys],[[X1,X2],[Y1,Y2]|Zs]) :-
            X2=<Y2.
```

## 6.5 Cooking Instructions

The practice of incremental enhancements of meta-interpreters was analyzed by Sterling [31]. This analysis, which draws its terminology from object programming, suggests that Prolog meta-interpreters are analogous to Lisp Flavors. Using Sterling's language (which is underlined), the modules $PARSE$ and $< F\_s, F\_p, F\_and, F\_or >$ are orthogonal enhancements to the $SOLVE$ flavor, in that the computations necessary for incorporating them are completely separate. The $PARSE$ predicate amounts to a behavioral enhancement: it extends the computation performed by $SOLVE$ without changing the meta-goal of the enhanced meta-interpreter. This is done simply by adding the parsing predicates to $SOLVE$'s theory. The fla-

vor $solve(H, Bel, F\_s, F\_p, F\_and, F\_or)$ is a structural enhancement of the vannila flavor $solve(H)$, in that the extra arguments $< F\_s, F\_p, F\_and, F\_or >$ (which are bound to a fixed belief calculus $< f\_s, f\_p, f\_and, f\_or >$) are used to compute $Bel$ as $H$ is being solved.

So, now that all the ingredients have been provided, the creation of a rule-based inference system is merely a matter of mixing flavors. Let $p$ and $q$ be two Prolog predicates whose extended theory is stored in two text files whose names are also $p$ and $q$, respectively (the *extended theory* of $p$ includes $p$'s theory and the theory of all the predicates mentioned in $p$'s theory. In what follows, when we say *add p to q* or *mix p and q* we mean "prove the goals $consult(p)$ and $consult(q)$."

With this terminology in mind, to prepare a CF-oriented inference system follow this set of instructions:

1. Create a CF-oriented knowledge-base and save it in a file called $KBASE$
2. Prove the goal $define\_syntax$
3. Mix the $CF\ PARSE$ predicate with the $SOLVE$ flavor
4. Add the predicates $cf\_s$, $cf\_p$, $cf\_and$, $cf\_or$
5. Mix the resulting inference system with the knowledge-base $KBASE$
6. Confirm the hypothesis $h$ and compute its posterior certainty-factor by proving the meta-goal $solve(h, Bel, cf\_s, cf\_p, cf\_and, cf\_or)$.

To prepare a Bayesian-oriented inference system, follow this set of instruc-

tions:

1. Create a Bayesian-oriented knowledge-base and save it in a file called $KBASE$
2. Prove the goal $define\_syntax$
3. Mix the Bayesian $PARSE$ predicate with the $SOLVE$ flavor
4. Add the predicates $b\_s$, $b\_p$, $b\_and$, $b\_or$
5. Mix the resulting inference system with the knowledge-base $KBASE$
6. Confirm the hypothesis $h$ and compute its posterior belief by proving the meta-goal $solve(h, Bel, b\_s, b\_p, b\_and, b\_or)$.

## 6.6 A Note on Function Variables in Prolog

Throughout the paper, the belief calculi functions were specified using the conventional algebraic notation $Y = f(X)$. In Prolog, this notation has no meaning. Instead, the logic programming equivalent of the computation $Y = f(X)$ is normally the predicate $f(X, Y)$. This goal is made to succeeds always, unifying the variable $Y$ to the value $f(X)$. For example, the successor function $s(X) = X + 1$ is implemented through the predicate $s(X, Y)$ : $-Y$ $is$ $X+1$. When we ask Prolog to prove the goal $s(3, Y)$, Prolog succeeds and binds $Y$ to 4 as a side-effect.

Now, things become slightly more complicated if we wish to treat the functor $f$ itself as a variable. This is precisely what is required in the $SOLVE$ meta-interpreter, which uses a belief calculus without knowing its exact specification. From a design standpoint, the ideal solution is to pass the four predicates $< f\_s, f\_p, f\_and, f\_or >$ as parameters to the $SOLVE$ predicate, creating a goal of the form $solve(h, Bel, f\_s, f\_p, f\_and, f\_or)$. In this context, the predicates $< f\_s, f\_p, f\_and, f\_or >$ are meant to instantiate the variables $< F\_s, F\_p, F\_and, F\_or >$ in $SOLVE$. However, this type of quantification is beyond the scope of first-order predicate calculus. This limitation can be overcome via Prolog's "univ" =.. operator. Among other

things, this operator may be used to bind variables to functions. For example, consider the following $APPLY1$ predicate, defined in by Sterling and Shapiro, [33]):

```
apply1(F,Xs) :- Goal=..[F|Xs], Goal.
```

The goal $apply1(f, Xs)$ causes Prolog to apply the function $f$ to the argument list $Xs$. For example, the goal $apply1(s, [3, Y])$ will succeed, resulting with $Y = 4$.

In this paper we define a more powerful version of $APPLY$, as follows:

```
apply(Predicate,Args) :- Predicate=..PredList,
                         append(PredList,Args,GoalList),
                         goal=..GoalList,
                         call(Goal).
```

Defined that way, the first argument of $APPLY$, *Predicate*, can be either an atomic symbol naming a predicate, or, alternatively, a term representing a predicate with some of its arguments supplied. For example, $apply(s, [3, Y])$ will yield $Y = 4$, and so will $apply(s(3), [Y])$. As yet another example of the utility of $APPLY$, consider the following numeric computation of the square-root function, using Newton's approximation formula:

```
sqrt(X,Y) :- apply(newton(0.01),[X,Y]).
newton(Epsilon,X,Y) :- iterate(Epsilon,X,Y,1).
iterate(Epsilon,X,Y,Y) :- Diff is X-Y*Y,abs(Diff,Z),Z=<Epsilon,!.
iterate(Epsilon,X,Y,Z) :- NewZ is (X/Z+Z)/2,
iterate(Epsilon,X,Y,Newz)
```

Defined that way, the parameter of the $NEWTON$ predicate, currently set to 0.01, specifies the precision level of the $SQRT$ function. That is, $Y$ is guaranteed to be within a 0.01 neighborhood of the true value of $\sqrt{X}$. In this example, $sqrt(4, Y)$ will yield $Y = 2.0006$.

To sum up, we see that the term representing the predicate in our definition of $APPLY$ is the equivalent of a *closure* in a Lisp-based functional language.

# References


[1] J.B. Adams. Probabilistic reasoning and certainty factors. In B. G. Buchanan and E. H. Shortliffe, editors, *Rule-Based Expert Systems*, pages 263–271, Addison-Wesley, 1984.

[2] Myers C. D. Alvey, P. L. and M. F. Greaves. An analysis of the problems of augmenting a small expert system. In M. A. Bramer, editor, *Research and Development in Expert Systems*, pages 61–72, Cambridge University Press, 1986.

[3] J.F. Baldwin and M.R.M. Monk. *SLOP - a System for Support Logic Programming.* Technical Report, University of Bristol, 1986. I.T.R.C. research report.

[4] B.G. Buchanan and E.H. Shortliffe. Uncertainty and evidential support. In B. G. Buchanan and E. H. Shortliffe, editors, *Rule-Based Expert Systems*, pages 217–219, Addison-Wesley, 1984.

[5] R. Carnap. *Logical Foundations of Probability.* University of Chicago Press, 1954.

[6] K. L. Clark and F. G. McCabe. Prolog: a language for implementing expert systems. In D. Hayes, Michie and W. H. Pao, editors, *Machine Intelligence 10*, pages 455–470, Ellis Horwood, 1982.

[7] P. Cohen, A. Davis, et al. Representativeness and uncertainty in classification systems. *The AI Magazine*, Fall:139–149, 1985.

[8] G.F. Cooper. *NESTOR: a Computer Based Medical Diagnosis Aid That Integrates Casual and Probabilistic Knowledge.* PhD thesis, Stanford University, 1984.

[9] G.F. Cooper. *Probabilistic Inference Using Belief Networks is NP-Hard.* Technical Report, Stanford University, 1987. KSL-87-27, Medical Computer Science Group, Knowledge Systems Laboratory.

[10] M. Dincbas. Metacontrol of logic programs in metalog. In *Proc. of FGCS, Tokyo, Japan*, pages 361–370, November, 1984.

[11] R.O. Duda, P.E. Hart, and N.J. Nilsson. *Development of a Computer-Based Consultant for Mineral Exploration.* Technical Report, SRI, 1977. International Projects 5821 and 6415.

[12] B.N. Grosof. Evidential information as transformed probability. In J.F. Lemmer and L. Kanal, editors, *Uncertainty in Artificial Intelligence*, pages 272–294, North Holland, 1986.

[13] D. E. Heckerman. Probabilistic interpretation for mycin's certainty factors. In J. F. Lemmer and L. Kanal, editors, *Uncertainty in Artificial Intelligence*, North Holland, 1986.

[14] R.A. Howard and J.E. Matheson. Influence diagrams. In R.A. Howard and J.E. Matheson, editors, *Reading on the Principles and Applications of Decision Analysis*, pages 721–762, Strategic Decisions Group, Menlo Park CA, 1981.

[15] D. Kahneman and D. T. Miller. Norm theory: comparing reality to its alternatives. *Psychological Review*, 93(2):136–153, 1986.

[16] S. Lichtenstein and J. R. Newman. Empirical scaling of common verbal phrases associated with numerical probabilities. *Psychonomic Science*, 9:563–564, 1967.

[17] D.H. Mitchell. *The Shape Experiment.* Technical Report, Northwestern University, 1986.

[18] A. Newell. Production systems: models of control structure. In W. Chase, editor, *Visual Information Processing*, pages 463–526, Academic Press, New York, 1973.

[19] Hammond P. Micro-prolog for expert systems. In K. L. Clark and F. G. McCabe, editors, *Micro-Prolog: Programming in Logic*, Prentice-Hall, 1984.

[20] J. Pearl. Fusion, propogation and structuring in belief networks. *Artificial Intelligence*, September, 1986.

[21] L. Pereira. Logic control with logic. In *Proc. of the First International Logic Programming Conference, Marseille*, pages 9–18, 1982.

[22] Davis R. and J. J. King. The origin of rule-based systems in ai. In B. G. Buchanan and E. H. Shortliffe, editors, *Rule-Based Expert Systems*, pages 20–52, Addison-Wesley, 1984.

[23] L.J. Savage. *The Foundations of Statistics.* Wiley, 1954.

[24] S. Schocken. *Quasi-Probabilistic Rule-Based Inference.* ICIT Press, 1989.

[25] S. Schocken and P. R. Kleindorfer. Artificial intelligence dialects of the bayesian belief language. *IEEE Transactions on Systems, Man, and Cybernetics*, 1989. Forthcoming.

[26] G. Shafer. Jeffrey's rule of conditioning. *Philosophy of Science*, September:337–362, 1981.

[27] G. Shafer. *A Mathematical Theory of Evidence.* Princeton University Press, 1976.

[28] G. Shafer and A. Tversky. Languages and designs for probability judgment. *Cognitive Science*, 9:309–339, 1985.

[29] E. Y. Shapiro. Logic programs with uncertainties: a tool for implementing rule-based systems. In *Proc. of the IJCAI, Krlsruhe, West Germany*, pages 529–532, 1983.

[30] E.H. Shortliffe. *Computer-Based Medical Consultation: MYCIN.* American Elsevier, 1976.

[31] L. S. Sterling. Meta-interpreters: the flavors of logic programming? In *Proc. of the Workshop on the Foundations of Deduction, Databases, and Logic Programming, Washington, D.C.*, 1986.

[32] L. S. Sterling and M. Lalee. An explanation shell for expert systems. *Computational Intelligence*, 1986 (to appear).

[33] L. S. Sterling and E. Y. Shapiro. The art of prolog. pages 280–283, The MIT Press, 1986.

[34] A. Tversky and D. Khaneman. Judgement under uncertainty: heuristics and biases. *Science*, 185:1124–1131, 1974.

[35] N. Wirth. *Algorithms + Data Structures = Programs.* Prentice-Hall, Englewood Cliffs, N.J., 1976.

[36] B.P. Wise. Experimentally comparing uncertain inference systems in probability. In J. F. Lemmer and L. Kanal, editors, *Uncertainty in Artificial Intelligence 2*, pages 89–102, North Holland, 1988.

[37] R.M. Yadrick, B.M. Perrin, D.S. Vaughan, P.D. Holden, and K.G. Kempf. Evaluation of uncertain inference models i: prospector. In J.F. Lemmer and L. Kanal, editors, *Uncertainty in Artificial Intelligence 2*, pages 77–88, North Holland, 1988.

[38] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.