Uncertainty for Programming Languages

J. Ulisses Ferreira Jr.

PhD stud. at the University of Dublin, Department of Computer Science, Dublin 2, Republic of Ireland

Escola Politécnica da UFBA, Rua Caetano Moura, Federação, Salvador, Brasil (Work)

e-mail: *Jose.Ferreira-Jr@cs.tcd.ie* or *ulisses@ufba.br*

***Abstract*— Uncertainty is a research topic which is well known to AI community. This paper adapts an uncertainty model to programming languages for internets describing the operational semantics of the present model.**

I. Introduction

Uncertainty and fuzziness[7] are research topics which are well known to the artificial intelligence community, and some commercial expert systems environments even provide some model for uncertainty, most commonly, production systems with confidence factors. In the last two decades, most expert systems have been based on some uncertainty model, in particular, a few variations of the MYCIN model of uncertainty[9]. However, to date computation under uncertainty and under "the absence of pieces of information" have been little investigated in the programming languages community. Together with the presence of the Internet and the unpredictability of success of underlying connections, to handle with both the unavailable information and uncertainty at the language level is a desirable feature for programming on the Internet.

A thesis of mine is that, once there is a established paradigm for programming agents, uncertainty[8] is probably going to play an important rôle in programming. Models that require many probabilities are mathematically accurate, useful in many applications and even the most appropriate in some cases. However, for agents[2], probabilities may be unable to suitably represent knowledge or belief, because humans do not normally think in terms of probabilities, but instead vaguer and less precise notions[8], while observing and inferring. I refer to such inferences as uncertainty-based inferences. Precise probability is also inefficient.

Agents have to be robust and, because of this, when connections fail or delay, programs should carry on running despite the lack of information. uu, which means "unknown" or "undefined", is a constant in programming languages that can be assigned to any variable of any datatype. uu is part of the PLAIN programming language, which was introduced by the present author[4]. This new constant guarantees both safety and robustness at the same time, because variables are never committed to any value that is not in the problem domain, while uu is not in the problem domain. A specific discussion on uu is in [3]. I believe that (mobile) agents and the Internet itself can provide a new support environment[4] where agent-based systems and techniques of artificial intelligence can succeed. In particular, while one of the original aims in artificial intelligence was to program and represent expert knowledge in systems, now, with agents, we not only expect intelligent machine behaviors, but also to simulate natural intelligence, for agents also represent their corresponding individuals in a complex society. In general, regardless of the kinds of application, agents should contain personal knowledge about their corresponding users, such as their preferences, dislikes and so forth. Here, I can see the rôle of uncertainty-based programming in the technological scenario, and this can partially be due to subjectivity, but partially due to other factors, such as the nature of the subject being represented.

Individuals differ. As an example, some people might take a decision in a given situation when they are 60% sure that they will succeed, while others will only take the risk in the same situation if they are 70% sure that they will succeed. The evaluation itself, i.e. whether 60% or 70%, depends on different subjective perception and internal factors. In this context, it is easy to observe that judgments under thresholds can provide the flexibility needed to represent personal characteristics. The work of reviewing academic articles by others is an example of this kind of inference in science and, because of this, there are often a number of referees.

In this paper, I present an uncertainty model suitable for programming agents or global computing. Given the simplicity needed for programming, the uncertainty model might not be new in the AI community, although I have not found another author for the same model, which is certainly based on the classical MYCIN[9]. The combination of significance with originality of the present work consists in:

- Adapting the MYCIN model and bringing the adapted model at the language level. I add two notions to the MYCIN model:
  - uu.
  - The ability to evaluate each hypothesis as resulting in either $false$ or $true$ (or $unknown$) without exploring all of the premises of the hypothesis. And because the premises of a hypothesis might correspond to or include remote accesses, one of the practical motivations in this paper consists in the ability that agents may have to reduce the number of remote accesses.

- Providing a formalization to the present model.
- Illustrating the importance of uncertainty in programming for global computing.

Moreover, at the time of writing this text, I am assuming that application programs with uncertainty at the language level are absent from the literature on programming languages.

The other sections are organized as follows: Section II introduces related concepts. Section III explains the relative concept of truth in this model, while section IV explains forward and backward evaluation under uncertainty, including how a program with uncertainty can compute efficiently on the Internet. In addition to the model, section IV-D introduces a set of built-in inference operators that can provide other choices to programmers. and section V contains the conclusion.

II. *uu* and Uncertainty

Here I present the formal syntax of a hypothesis declaration, a subset of PLAIN without handlers, with starting symbol hyp:

hyp $\longmapsto$ **hypo** Id factor '{' opthreshold **if** premlist ';' '}'

factor $\longmapsto \varepsilon$ | '(' $Number$ ')'

opthreshold $\longmapsto \varepsilon$ | **threshold** $Number$ ';'

premlist $\longmapsto \varepsilon$ | prem cnf | prem cnf ',' premlist

prem $\longmapsto$ '{' premlist '}' | Id | '(' expr ')' | inferop

'(' premlist ')' | inferop '(' $Number$ ',' premlist ')'

cnf $\longmapsto \varepsilon$ | **cnf** $Number$

inferop $\longmapsto$ **uand** | **uor** | **unot** | **dand** | **dor**

expr $\longmapsto \ldots$ – any three-valued logical expression.

$Number \longmapsto$ – any real number in $[-1.0, +1.0]$.

In the text, ?? can replace **cnf** here.

We all know that the real world is full of rules of causes and consequences and, because of this, as an alternative to deductive clauses such as Horn clauses or **Prolog**-like rules, uncertainty can also be used to permit an agent to make its own decisions. Like most expert systems environments, a PLAIN-like programming language can provide constructs for representing uncertainty as follows:

```
hypo $awebfault$ {
  threshold 0.4;
  if {
    (!$httpaccessok$(“www.aaa.bbb.ccc.dbf”)) cnf 0.2,
    (!$httpaccessok$(“www.ddd.eee.fff.dbf”)) cnf 0.2,
    (!$httpaccessok$(“www.ggg.hhh.iii.dbf”)) cnf 0.2
  }
  when $self$ true { /* make some decision... */ }
}
```

When an agent contains a great number of such hypotheses, the system is deemed to be intelligent, with the ability of making decisions given the complex and subjective nature of the objects (0.2 of certainty, in each of the above cases). Additionally, operators are provided to be used with certainty factors, as well as nesting expressions with their corresponding factors. In such contexts, parentheses surrounding an expression indicates that its negation does not contribute to refute the consequent, it is simply ignored instead.

In this model, I shall make use of two forms of $unknown$ value for variables: uu_i and uu_f, although there is one constant in the language for both meanings, $unknown$. In this paper, uu is a lexical sugar for either uu_i or uu_f at points where the difference does not matter. uu_i is the initial or partial unknown state while uu_f is the final unknown state after the evaluation is performed. I write uu where this difference is not relevant in a particular formula.

Initially, all hypotheses in the program contain uu_i. For every logical variable z in the program, if $z \neq uu$, there is an internal state composed by a pair of thresholds $(False, True)$ where $-1 \leq False(z) < True(z) \leq +1$ and which are stated by the programmer and will be clearer in this paper later; a pair of certainty measures (at least, $x(z)$, and at most, $y(z)$) where $-1 \leq x(z) \leq y(z) \leq +1$, which are inferred dynamically; and three alternative external values that correspond to ff, *uu* and tt, namely $false$, $unknown$ and $true$ (internally, there are four values in $\{ff, uu_i, uu_f, tt\}$), one of which is the result from the following conditions intuitively written in PLAIN:

```
logic $val$(logic $z$) =
  if $z.x \geq z.True$ then $true$;
  else if $z.y < z.False$ then $false$;
  else $unknown$;
```

or, in a more mathematical style, still in PLAIN:

$$\textbf{logic } val(\textbf{logic } z) = \begin{array}{ll} true & \textbf{if } z.x \geq z.True, \\ false & \textbf{if } z.y < z.False, \\ unknown & \textbf{otherwise}; \end{array}$$

Thus, I write $z.x$, $z.y$, $z.False$, $z.True$ instead of $x(z)$, $y(z)$, $False(z)$, $True(z)$. In this paper, I omit the variable name when the context applies to all logical variables without the need to specify some particular variable. In this adapted model, by default, $False = 0$ and $True = +0.5$ in the range $[-1.0, +1.0]$.

There is a built-in unary prefix operator, ?, that can be applied to any logical variable. Thus, for a variable z, $?z$ is the certainty measure of z, whose value corresponds to the arithmetic mean of x and y of z. $?z$ is the form in PLAIN for any z while here, in this text, the $z?m$ form is used with the same meaning.

Logical variables are conceptually classified as hypotheses and pieces of evidence. A hypothesis has a premise list, while evidence does not have premises. Premises can be classified as hypotheses, evidence, logical expressions and inference operators (inferop). Besides the certainty measure, logical expressions and inferops can result in $\{ff, uu, tt\}$.

An example of syntax is as follows:

```
hypo $h$(-0.2,0.7) {
  if $a$ cnf 0.3,
    $c$,
    $(b * b - 4 * a * c < 0)$ cnf 0.2,
    $(c)$,
    uand (0.5, $a$ cnf 0.2, $b$, $c$ cnf 0.8 ) cnf 0.9,
    { $a$ cnf 0.2, $b$ cnf 0.2, $c$ cnf 0.3 } cnf 0.8
  ;
}
```

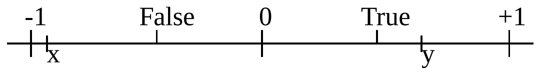
The above hypothesis h contains the thresholds $False = -0.2$ and $True = +0.7$, and its list of premises as follows: a, a logical variable whose certainty factor is 0.3 above (0.3 is actually the certainty factor for the relation with (a, h), but stated here in a simplified way); c, another logical variable whose certainty factor is 1.0 above, by default; the expression $(b * b - 4 * a * c < 0)$ whose certainty factor is 0.2; (c), another expression whose certainty factor equals 1.0, the default certainty factor; one instance of **uand** inference operator with truth threshold 0.5, certainty factor 0.9 and whose premises are variables a, b and c with certainty factors 0.2, 1.0 and 0.8 respectively; and a composition among the premises: a, b and c with certainty factors 0.2, 0.2 and 0.3 respectively, and the result from the composition is meant to be multiplied by the certainty factor 0.8 as will be explained later. As suggested in the above example, premises can be nested. Expressions are always between parentheses, and the semantic difference between a variable and an expression containing only the same variable is that when an expression results in $false$ its effect is null with respect to the hypothesis, i.e. it does neither contribute to prove nor refute the hypothesis. As for an occurrence of a variable in the list of premises, if the variable is false its certainty measure is negative. At first sight the use of expression might appear limited, but since an expression can be written in the list of premises more than once with different certainty factors, the use of expressions is flexible and even slightly more general. Thus, c ?? 0.8 corresponds to $\{(c)$?? $0.8,\ (not\ c)$?? $-0.8\}$.

Like measure, certainty factors are in the real interval $[-1.0, +1.0]$ and are written by programmers to measure how the corresponding premises contribute to prove or refute the hypotheses, depending on the sign, positive or negative. While certainty factors are specified in the program, certainty measure is a dynamic value in the same real interval $[-1.0, +1.0]$. For simulating a subjective judgment, in this model, the concept of truth is relative to the programmers beliefs, as well as to beliefs of possible users when the corresponding system is applied. Therefore, each relation between one premise and one hypothesis or between one premise and one inferop has one certainty factor and one certainty measure attached. Syntactically, certainty factors can be floating-point expressions. I simplify the present model by allowing only constants in that interval. During the evaluation, a certainty measure is multiplied by the corresponding certainty factor, and its result is finally an input to the hypothesis.

By combining hypotheses and premises, the programmer can represent complex knowledge forming an acyclic graph.

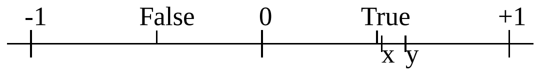
III. Uncertainty Handling

The four figures below represent the four possible states of a logical variable. A possible initial state in which a variable can be regarded as uu_i if the condition $x < False \wedge y \geq True$ holds is as follows:

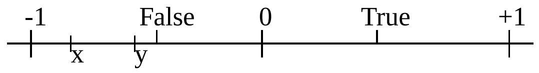


The initial state is not necessarily $x = -1 \wedge y = +1$, but instead both values can be calculated by the compiler.

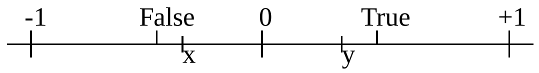
Eventually, a variable can be regarded as $true$, if the condition $x \geq True$ holds:



Or eventually, a variable can be regarded as $false$, if the condition $y < False$ holds:



Or eventually, a variable can be regarded as uu_f, if the condition $x \geq False \wedge y < True$ holds:

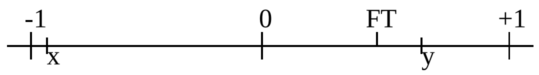


In an alternative model, there might be other values such as *non-False* ($x \geq False \wedge x < True \wedge y \geq True$) and *non-True* ($x < False \wedge y \geq False \wedge y < True$). Both are regarded as $unknown$ (uu_i) in the present model.

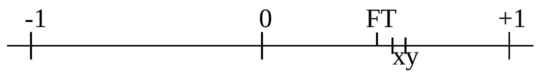
The difference $y - x$ means the amount of non-evaluated evidence. An inconsistent state does not occur in the present model since $x \leq y$ and $False \leq True$ always hold.

A simpler model equates the thresholds, $FT = False = True$. Further, $FT = 0$ is the default value. Graphically, it reduces a hypothesis to only three states:

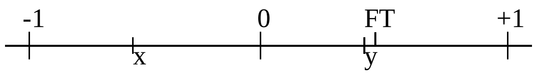
The initial $unknown$ state as the condition $x < FT \leq y$ holds:



Eventually, a variable can be regarded as $true$ once the condition $x \geq FT$ holds:



Or eventually, a variable can be regarded as $false$ once the condition $y < FT$ holds:



Both schemes permit variables to contain uu in the premises and, because of this, hypotheses can also result in uu in the latter case.

The ability to permit premises and hypotheses to result in uu is a flexible characteristic of the present programming model.

Before describing the two directions of evaluations, I should set some default values for a premise z according to its category:

- Hypothesis: as described above, $FT(z)$ is statically given by the programmer. $x(z)$ and $y(z)$ are calculated as the agent is initialized, or at the time of compilation of the program;
- Logical evidence: $FT(z) = +0.5,\ x(z) = -1,\ y(z) = +1$;
- Expression: $FT(z) = +1,\ x(z) = 0,\ y(z) = +1$;
- Inferop: $FT(z) = 0,\ x(z) = -1,\ y(z) = +1$.

IV. Evaluation

Here the present work provides forward and backward evaluations. The former starts from some evidence and goes upwards to the hypotheses while the latter starts from one hypothesis and goes downwards trying to calculate its certainty measure according to its premises. In the following subsections, I provide a scheme, for both forward and backward directions.

Let $[-1, +1] \subset \mathbb{R},\ \mathbb{F} \equiv [-1, +1], \mathbb{V} = \{ff, uu_i, uu_f, tt\}$. For this section, for a logical variable, its *value and certainty measure* can be represented as a tuple in $\mathbb{O} \equiv \mathbb{F} \times \mathbb{F} \times \mathbb{F} \times \mathbb{V} \times \mathbb{F} \times \mathbb{F}$. The elements of such a tuple are the following, whose description order corresponds to left-right order in the tuple:

- The minimum certainty measure x;
- The maximum certainty measure y;
- The certainty measure;
- The logical value;
- The false threshold $False$;
- The true threshold $True$.

Let $(px, py, ??p, pv, pFalse, pTrue)$ be such a tuple, where px and py correspond to the certainty measures (*at least* and *at most*) as explained, and so on. Let $??p$ be $(px + py)/2$. It will always be in this way, computed on demand.

In this paper, uu is the lexical sugar for either uu_i or uu_f.

A. Forward Evaluation

Forward evaluation is invoked by a built-in function call, perhaps imperative, and usually after some data has been entered. After the evaluation has finished, some imperative actions are typically performed once hypotheses become true or false, and perhaps after other conditions.

The forward evaluation starts from the pieces of evidence and goes upwards. Because a hypothesis can be declared as a premise to another hypothesis, the forward evaluation continues recursively until no more hypotheses need to be calculated.

The composition of a list of premises, whose representation in Plain is between a pair of braces, has the same algorithm regardless of the number of premises. Before describing composition, I consider only two premises, namely P with logical value $(px, py, ??p, pv, pFalse, pTrue)$ and Q with $(qx, qy, ??q, qv,\ qFalse, qTrue)$, to the hypothesis H with $(Hx, Hy, ??H, Hv, HFalse, HTrue)$. Considering that $px = py = 0\ \ if\ pv = uu$ in P and $qx = qy = 0\ \ if\ qv = uu$ in Q, the certainty measures of H are defined by a set of formulae computed as follows:

$$Hx = \varphi(px, qx) = \begin{cases} px + (1 - px)\cdot qx, & \text{if } px \geq 0,\ qx \geq 0. \\ px + (1 + px)\cdot qx, & \text{if } px < 0,\ qx < 0. \\ px + qx, & \text{otherwise.} \end{cases}$$

$$Hy = \varphi(py, qy) = \begin{cases} py + (1 - py)\cdot qy, & \text{if } qy \geq 0,\ qy \geq 0. \\ py + (1 + py)\cdot qy, & \text{if } py < 0, qy < 0. \\ py + qy, & \text{otherwise.} \end{cases}$$

$$H.?? = mean(Hx, Hy) = \frac{Hx + Hy}{2}$$

and judgment for the value $Hv = H.v = A(H)$ in the following:

$$H.v = A(H) = \begin{cases} tt & \text{if } H.x \geq H.True. \\ ff & \text{if } H.y < H.False. \\ uu_i & \text{if } (H.x < H.False \vee H.y \geq H.True) \\ & \wedge H.x < H.y. \\ uu_f & \text{if } H.x \geq H.False \wedge H.y < H.True. \end{cases}$$

where $H.v$ denotes the attribute of logical value of H, and $H.x$ and $H.y$ correspond to Hx and Hy, respectively. For strict evaluation, which is an alternative form here (the default form in Plain), the reader shall see that there is a condition, $H.x = H.y$, in addition to the above one for the machine to consider $H.v = uu_f$. See later.

One of the positive points of these formulae is that the evaluation of the corresponding scheme is both commutative and associative. That is, although the premises are statically written in a sequence in the program, they become available dynamically in any order, and the order does not affect the result of the hypothesis certainty measure.

The structure for every hypothesis H follows: Let $\mathbb{P}$ be the finite set of all premises of the program, where every premise is of type $\mathbb{F} \times \mathbb{O}$. Let $\mathbb{H} \equiv \mathbb{O} \times \mathbb{N} \times \mathcal{P}(\mathbb{P})$ be the set of hypotheses of the program. For every hypothesis $H : \mathbb{H}$, besides its value, the list of premises that lead to H is of the form $\langle n, \{H.P_1, H.P_2, ..., H.P_n\}\rangle$. The whole list is addressed as $H.P$, of type $\mathbb{N} \times \mathcal{P}(\mathbb{P})$. The list length is addressed as $H.P.n$.

Every premise P is of type $\mathbb{F} \times \mathbb{O}$. The fist element of P, denoted as $P.??$, is the certainty factor between P and its corresponding hypothesis. From the hypothesis H, given some

$i \in \mathbb{N}$, this certainty factor is referred to as $H.P_i.??$ while the certainty measure of P_i is referred to as $H.P_i?m$.

Thus, for this paper, $\mathbb{H}$ is the set of hypotheses and $\mathbb{P}$ is the set of premises. Further, from now on I generalize the notation by writing $H.x$ and $H.y$, instead of writing Hx and Hy, respectively. And from now on I am going to consider any number of premises. For every hypothesis $\mathcal{H}$, I have its finite list of premises, $P_1, P_2, ..., P_i, ..., P_n$. For every P_i, for $i \in [1, n] \subset \mathbb{N}$, the P_i value is in $\{ff, uu, tt\}$, as well as their certainty measures. Furthermore, in addition to the logical value and certainty measures, every premise contains the certainty factor to link to the hypothesis, that is, the certainty factor helps prove or refute the corresponding hypothesis. Given that every $P_i : \mathbb{F} \times \mathbb{O}$, the first element of that P_i is the certainty factor towards the corresponding hypothesis.

Letting $P_i.v = uu \Rightarrow P_i.x = 0$ and $P_i.y = 0$, the certainty measure of $\mathcal{H}$ is defined by a set of formulae, given new values p_i for some occurrences of $i \in [1, n]$, for some n, as follows:

Let $[-1, +1] \subset \mathbb{R}$, $\mathbb{F} \equiv [-1, +1]$, $\mathbb{V} \equiv \{ff, uu, tt\}$, $\mathbb{O} \equiv \mathbb{F} \times \mathbb{F} \times \mathbb{F} \times \mathbb{V} \times \mathbb{F} \times \mathbb{F}$.

Suffix functions:

$.x : \mathbb{O} \longrightarrow \mathbb{F}$;	$(x, y, c, v, f, t).x = x$.
$.x : \mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{F}$;	$(x, y).x = x$.
$.y : \mathbb{O} \longrightarrow \mathbb{F}$;	$(x, y, c, v, f, t).y = y$.
$.y : \mathbb{F} \times \mathbb{F} \longrightarrow \mathbb{F}$;	$(x, y).y = y$.
$?m : \mathbb{O} \longrightarrow \mathbb{F}$;	$(x, y, c, v, f, t)?m = c$.
$.v : \mathbb{O} \longrightarrow \mathbb{V}$;	$(x, y, c, v, f, t).v = v$.
$.False : \mathbb{O} \longrightarrow \mathbb{F}$;	$(x, y, c, v, f, t).False = f$.
$.True : \mathbb{O} \longrightarrow \mathbb{F}$;	$(x, y, c, v, f, t).True = t$.

$\alpha : \mathbb{F} \times \mathbb{H} \longrightarrow \mathbb{F}$;
$\alpha(z, \mathcal{H}) = z \cdot \mathcal{H}.P_i.??$
where $\mathcal{H}.P_i.??$ corresponds to the certainty factor between P_i and $\mathcal{H}$.

$\delta : \mathbb{P} \longrightarrow \mathbb{F}$;
$\delta(P) = P.y - P.x$.

$F : \mathbb{H} \times \mathcal{P}(\mathbb{P}) \times \mathbb{N} \longrightarrow \mathbb{F} \times \mathbb{F}$;

$$F(\mathcal{H}, P, n) = \begin{cases} (\ (F(\mathcal{H}, P, n-1)).x + \delta(P) \cdot \alpha(P_n.x, \mathcal{H}), \\ (F(\mathcal{H}, P, n-1)).y\ ) \\ \qquad if\ n > 0 \wedge \alpha(P_n.x, \mathcal{H}) \geq 0. \\ \\ (\ (F(\mathcal{H}, P, n-1)).x, \\ (\ (F(\mathcal{H}, P, n-1)).y + \delta(P) \cdot \alpha(P_n.y, \mathcal{H})) \\ \qquad if\ n > 0 \wedge \alpha(P_n.y, \mathcal{H}) < 0. \\ \\ (\ (F(\mathcal{H}, P, n-1)).x,\ (F(\mathcal{H}, P, n-1)).y) \\ \qquad if\ n > 0 \wedge P_n.v = uu. \\ \\ (\ 0,\ 0\ ) \qquad if\ n = 0. \end{cases}$$

and after having calculated the final values x and y,

$$\mathcal{H}?m = \frac{\mathcal{H}.x + \mathcal{H}.y}{2}$$

where $\mathcal{H}?m$ is the confidence measure of the hypothesis $\mathcal{H}$, calculated as follows:

$$\mathcal{H}?m = \frac{(F(x, y, \mathcal{H}, P_i, n)).x + (F(x, y, \mathcal{H}, P_i, n)).y}{2}$$

In the forward evaluation, because states can be changed, expressions in the list of premises are always evaluated when the virtual machine visits some hypothesis.

Forward evaluation is cyclical, and the process is repeated while hypotheses feed other hypotheses with new certainty measures. The process is repeated by possible imperative handlers whose actions can be triggered just after the virtual machine has proven some particular hypothesis. Since such actions can change the program state, they can require an extra cycle of forward evaluation. The process finishes when the objects involved become stable, i.e. all set of hypotheses have the same values for more than one subsequent cycle.

This model has the advantage of representing and dealing with lack of information. Moreover, unlike Boolean models, the present model represents the condition of being unable to prove or refute a hypothesis.

B. *Operational Semantics - Forward Evaluation*

Here, I present an operational semantics for the forward evaluation during computation of a program Π. Given a program Π as a set of symbols, let $\mathbb{E} \subset \Pi$ be the set of pieces of evidence, $\mathbb{H} \subset \Pi$ be the set of hypotheses, $\mathcal{O}p \subset \Pi$ be the set of occurrences of inference operators, and $X \subset \Pi$ be the set of expressions. Let $\mathbb{P} = \mathbb{E} \cup X \cup \mathbb{H}$ and $E \cap X = E \cap H = X \cap H = \emptyset$. Let $\mathbb{S}$ be the set of states of computation.

Briefly, for our semantics of uncertainty-based inference, I define two relations for forward and backward evaluations, respectively: $\overset{\text{fw}}{\rightsquigarrow}$ and $\overset{\text{bw}}{\rightsquigarrow}$ leading to some state of computation. Another relation, $\overset{\text{eval}}{\rightsquigarrow}$ for expression evaluation, also leads to a state, although I only represent the resulting pair of values, the minimum and maximum certainty measures in $[-1.0, +1.0]$ and the logical value in $\{ff, uu, tt\}$.

$\iota : \mathbb{H} \times \mathbb{S} \times \mathbb{P} \longrightarrow \mathbb{O}$. Intuitively, $\iota(h, s, p)$ is the result from the forward evaluation from state s and premise p to the conclusion h.

Where it is suitable, I write $h.P$ to denote the list of premises of the hypothesis h. I define $\iota(h, s, P)$ to make the sequence of operations explicit in PLAIN, in terms of σ, which in turn results in a pair (x, y), which in its turn is a certainty measure (minimum,maximum), and to state that ι does not change the state, in the PLAIN syntax (although the words and other tokens are not the same):

$\alpha(p, h) \quad = \ p?m \cdot h.p.??;$

$\delta(e, l) =$ **true if** $[e] == head(l)$,
$\delta(e, [h, t]) = \delta(e, t)$ **if** $[e]\ ! =\ h$;

$\sigma(h, s, p) \quad = \ (h.x \ + \ (h.y - h.x) \cdot \alpha(p, h),\ h.y$

$$\begin{array}{lll} & & mean(h.x + (h.y - h.x)\cdot\alpha(p,h), h.y), \\ & & A(h), h.False, h.True) \\ & & \quad \textbf{if } \alpha(p,h) > 0, \\ \sigma(h,s,p) & = & (h.x,\ hy\ +\ (hy - hx)\cdot\alpha(p,h), \\ & & \ mean(h.x,\ hy\ +\ (hy-hx)\cdot\alpha(p,h)), \\ & & \ A(h), h.False, h.True) \\ & & \textbf{if } \alpha(p,h) < 0, \\ \sigma(h,s,p) & = & h\ \ \textbf{if } \alpha(p,h) = 0; \\ \\ \iota(h,s,p) & = & \sigma(h,s,P)\ \textbf{if } \delta(p, h.P), \\ & & \ h\ \textbf{otherwise}; \end{array}$$

where $p \in \mathbb{P}$, $p?m$ is the certainty measure of a premise p and $h.p.??$ is the certainty factor from the premise p to the hypothesis h, which is static from the source program.

Since every hypothesis can have hypotheses, expressions, inference operators and pieces of evidence as premises (for proving or refuting the former hypothesis itself), let $Op(h) \in Op$ be the set of inferops in the list of premises of h, more precisely,

$$\begin{array}{c}\forall i,j \in Op,\ \ (i \in h.P \Rightarrow i \in Op(h)) \wedge \\ (i \in j.P \wedge j \in Op(h) \Rightarrow i \in Op(h))\end{array}$$

Similarly, for some $h \in \mathbb{H}$, let $Obj = E \cup X \cup H$ be the set of all pieces of evidence and expressions and hypotheses of the program, and let $C(h) \subseteq Obj$ be the set of objects that can contribute to prove or refute h. Let $p \in Obj$. Thus, $p \in h.P \Rightarrow p \in C(h)$, $(p \in C(g) \wedge g \in C(h)) \Rightarrow p \in C(h)$, and $(\exists i \in Op(h),\ (p \in i.P)) \Rightarrow p \in C(h)$.

Then, considering that the symbol a in the **forward** command is grammatically a piece of evidence, an operational semantics for the forward evaluation can be as follows:

$$\frac{a \in h.P \quad \langle h, s\rangle \overset{\text{fw}}{\rightsquigarrow} s'[h?m = \iota(h, s', a)]}{\langle \textbf{forward } a, s\rangle \overset{\text{fw}}{\rightsquigarrow} s'}$$

while, as usual, a state (s, s' or s'') followed by an expression between square brackets indicates that the expression holds in the state. For instance, $s[x = 0]$ means that the expression $x = 0$ holds in the state s. The **forward** statement can change the state of more than one hypotheses by propagating uncertainty from a hypothesis to another as follows:

$$\frac{\begin{array}{c} a \in C(h_1) \wedge h_1 \in C(h_2) \\ h_1 \notin C(h_2) \Rightarrow (\langle \textbf{forward } a, s\rangle \overset{\text{fw}}{\rightsquigarrow} s'[h_1?m = \iota(h_1, s', a)]) \\ \langle \textbf{forward } a, s'\rangle \overset{\text{fw}}{\rightsquigarrow} s''[h_2?m = \iota(h_2, s'', h_1)] \end{array}}{\langle \textbf{forward } a, s\rangle \overset{\text{fw}}{\rightsquigarrow} s''}$$

Finally, to state that **forward** is an imperative command in PLAIN, I can insert it in a rule for sequence of commands:

$$\frac{\langle c, s\rangle \overset{\text{fw}}{\rightsquigarrow} s' \quad \langle \textbf{forward } a, s'\rangle \overset{\text{fw}}{\rightsquigarrow} s''}{\langle c; \textbf{forward } a, s\rangle \overset{\text{fw}}{\rightsquigarrow} s''}$$

C. Backward Evaluation

While the forward evaluation is invoked by a command to discover those hypotheses that become proven and those hypotheses that become refuted, the backward evaluation is invoked for a particular hypothetic goal, more precisely, when a hypothesis is used in any evaluating expression such as from imperative constructs. The virtual machine then *tries* to decide that particular hypothesis (i.e. the machine tries to prove or refute it), visiting the premises backwards until it reaches the pieces of evidence, by running handlers to change values of the related evidence. After that, the control is returned to that hypothesis. Finally, the control continues normally in the expression evaluation with the new requested result in $\{ff, uu, tt\}$ while the certainty measures are ignored for the context is deductive.

In a local context, the order of evaluation is not relevant for the result, but since I intend to apply the present model to programming for internet, I propose a way of predicting the best path in terms of efficiency.

Because certain pieces of evidence might not be available at that moment, the hypothesis h that has been expected to be proven or refuted might remain $unknown$, that is, $h.x < h.False \leq h.True \leq h.y$.

An expression as a premise can also result in $\{ff, uu, tt\}$. It is arguable whether it is worth allowing programmers to state some uncertainty inside expressions to allow the virtual machine to use the factors to compute the certainty measure of the expression, in particular if the expression may result in many possible alternative values, such as integer expressions, not only two or three values. Thus, for safety reasons, PLAIN adopts the conservative idea that only logical variables have certainty measures.

1) The Most-Interesting First Strategy: In this section I introduce *uncertain lazy computation with the most-interesting first strategy*, which is probably useful for global computers[1]. Although it is easy to evaluate all premises of the requested hypothesis sequentially, I introduce a strategy that tries to minimize the number of premises necessary to prove the hypothesis, or to refute it, or both (this case happens where some premises contribute to prove the hypothesis while other premises contribute to refute it. In this case, both subsets are relevant). Such a strategy becomes particularly significant in programming for a global computer because remote accesses are considerably much more expensive than use of local resources, although Internet 2 and other global networks in the future tend to minimize this difference. Thus, I classify three possible intentions for a hypothesis containing uu_i:

- $\mathcal{I}_t$: To make the hypothesis be *true*.
- $\mathcal{I}_f$: To make the hypothesis *false*.
- $\mathcal{I}_{tf}$: To make the hypothesis either *true* or *false* (or uu_f).

All logical variables containing uu_i, as well as all non-lazy expressions, are valuable. For a valuable hypothesis or logical evidence or expression or inferop occurring as a premise p of a hypothesis h, in $\mathcal{I}_t$ what I look for is $(h.True - h.x)\ /\ |h.p.??|$: for more than one hypotheses or pieces of evidence or expressions or inferops, the smaller this value is, the more interesting p is, but I only consider positive results for the premises. Similarly, in the intention $\mathcal{I}_f$ what I look for is $(h.y - h.False)$ / $|h.p.??|$: the smaller the value is, the more interesting p is, and here I consider only non-negative results for the premises. In $\mathcal{I}_{tf}$ what I look for is the value of $(h.True - h.x\ +\ h.y - h.False)$ / $|h.p.??|$: the smaller, the more interesting, and here only positive values are included in comparisons. For future work, every of these three values can be multiplied by the number of variables with uu_i and that play the rôle of p, for including the associated cost in the strategy. Sometimes this cost may be important because, as an example, connections might have to be set in order to associate values to variables.

Then, intuitively, the strategy consists in calculating these values for every premise of some hypothesis, and then to choose the smallest value obtaining the premise to be exploited first, in the backward direction. Then, the process is repeated until the hypothesis is no longer valuable, that it, until its logical value is either $true$, $false$ or uu_f. Although the strategy does not generally guarantee the most efficient proof or refutation, it is natural and efficient. Therefore, I refer to this final measure as the most *interesting*.

Among the valuable premises, the virtual machine can make use of $d(h, \mathcal{I})$ to identify the most interesting premise (that is, its index in $[1, n]$) in a list P with n valuable premises of a given hypothesis h, some intention $\mathcal{I}$. The function definition is the following:

Valuable premises only
$\epsilon(\emptyset) \quad = \emptyset,$
$\epsilon(\{P\} \cup Q) \quad = \textbf{if } \psi(P) \textbf{ in } \{false, uu_f, true\}$
  $\textbf{then } \epsilon(Q) \textbf{ else } \{P\} \cup \epsilon(Q);$

$\psi(x, y, F, T) = true \textbf{ if } x \geq T,$
  $false \textbf{ if } y < F,$
  $uu_f \textbf{ if } F \leq x \wedge y < T,$
  $uu_i \textbf{ otherwise};$

Unary ψ
$\psi(P) \quad = \psi(P.x, P.y, P.False, P.True);$

$d(h, \mathcal{I}) \quad = \ -2 \textbf{ if } \epsilon(h.P) = \emptyset,$
$d(h, \mathcal{I}) \quad = \ d(h, \epsilon(h.P), \mathcal{I}) \textbf{ otherwise};$

Ternary d
$d(h, \{P_i\}, \mathcal{I}) \quad = \ i,$

with intention $\mathcal{I}_t$
$d(h, \{P_i\} \cup Q, \mathcal{T}) = \textbf{ if}$
  $0 < (h.True - P_i.x)\ /\ |h.P_i.??| \leq$
  $(h.True - h.P_{d(h,Q,\mathcal{T})}.x)\ /\ |h.P_{d(h,Q,\mathcal{T})}.??|$
  $\textbf{then } i \textbf{ else } d(h, Q, \mathcal{T}),$

with intention $\mathcal{I}_f$
$d(h, \{P_i\} \cup Q, \mathcal{F}) = \textbf{ if}$
  $0 \leq (P_i.y - h.False)\ /\ |h.P_i.??| \leq$
  $(h.P_{d(h,Q,\mathcal{F})}.y - h.False)\ /\ |h.P_{d(h,Q,\mathcal{F})}.??|$
  $\textbf{then } i \textbf{ else } d(h, Q, \mathcal{F}),$

with intention $\mathcal{I}tf$
$d(h, \{P_i\} \cup Q, \mathcal{U}) \ = \textbf{if}$
  $0 < (h.True - P_i.x\ +\ P_i.y - h.False)\ /\ |h.P_i.??| \leq$
  $(h.True - h.P_{d(h,Q,\mathcal{U})}.x + h.P_{d(h,Q,\mathcal{U})}.y - h.False)\ /$
  $|h.P_{d(h,Q,\mathcal{U})}.??|$
  $\textbf{then } i \textbf{ else } d(h, Q, \mathcal{U});$

Finally, the backward evaluation finishes only when the hypothesis is no longer valuable. The c function repeats d until that condition holds, as follows:

$U(h, x, y) = (\varphi(h.x, x), \varphi(h.y, y), mean(\varphi(h.x, x), \varphi(h.y, y)),$
  $\psi(\varphi(h.x, x), \varphi(h.y, y), h.False, h.True), h.False, h.True);$

$c(h, \mathcal{I}) \quad = \textbf{if } d(h, \mathcal{I}) = -2 \textbf{ then } h \textbf{ else } c(h, h.P, \mathcal{I});$

Ternary c (every hypothesis has at least one premise)
$c(h, \{P_i\}, \mathcal{I}) \quad = \ \textbf{if } \psi(P_i) = uu_f \textbf{ then } h \textbf{ else}$
  $\textbf{let } p = newstate(P_i) \textbf{ in } U(h, p.x \cdot h.P_i.??, p.y \cdot h.P_i.??),$

$c(h, P, \mathcal{I}) \quad = h \textbf{ if } \epsilon(P) = \emptyset,$
$c(h, P, \mathcal{I}) \quad = \textbf{let } j = d(h, P, \mathcal{I});\ p = newstate(P_j) \textbf{ in}$
  $c(U(h, p.x \cdot h.P_i.??, p.y \cdot h.P_i.??), P \backslash \{P_j\}, \mathcal{I})$
  otherwise;

Thus, the operational semantics for using a hypothesis in some lazy expression can be as follows:

$$\frac{\langle c(h, \mathcal{I}_{tf}), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \quad \langle \textbf{lazy}(h), s'\rangle \overset{\text{bw}}{\rightsquigarrow} (s', \iota(h, s', c(h, \mathcal{I}_{tf})))}{\langle \textbf{lazy}(h), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', \iota(h, s', c(h, \mathcal{I}_{tf})))}$$

Accordingly, there are also built-in functions in the programming language for the other intentions:

$$\frac{\langle c(h, \mathcal{I}_t), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \quad \langle h, s'\rangle \overset{\text{bw}}{\rightsquigarrow} (s', \iota(h, s', c(h, \mathcal{I}_t)))}{\langle \textbf{lazy}(\textbf{prv}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', \iota(h, s', c(h, \mathcal{I}_t)))}$$

$$\frac{\langle c(h, \mathcal{I}_f), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \quad \langle h, s'\rangle \overset{\text{bw}}{\rightsquigarrow} (s', \iota(h, s', c(h, \mathcal{I}_f)))}{\langle \textbf{lazy}(\textbf{rft}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', \iota(h, s', c(h, \mathcal{I}_t)))}$$

for trying to prove $(\mathcal{I}_t)$ and refute $(\mathcal{I}_f)$, respectively, some hypothesis h. The above rules are not complete.

2) Lazy and Strict Computations: In the backward evaluation, for the requested hypothesis, there can be lazy and strict computations. Lazy computation is the one which makes use of the most-interesting first strategy while strict computation evaluates all premises of z in any case until $z.v = uu_f$ for every logical variable z.

By default, the virtual machine adopts strict computation and, to specify lazy computation, the programmer writes the **lazy** keyword. Thus, **lazy** is a unary-prefix function with one of the highest precedences.

During the lazy computation of backward evaluation, for every variable z, one of the conditions $z.x \geq z.True$, or $z.y < z.False$, as stated previously, is enough to prove or refute the hypothesis, respectively. Thus, an operational semantics for uncertain lazy computation is as follows:

$$\frac{s[h.x \geq h.True]}{\langle \textbf{lazy}(h), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s, tt)}$$

$$\frac{s[h.y < h.False]}{\langle \textbf{lazy}(h), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s, ff)}$$

$$\frac{s[h.x \geq h.False \ \wedge \ h.y < h.True]}{\langle \textbf{lazy}(h), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s, uu_f)}$$

$$\frac{\begin{array}{c}\text{let } z = \iota(h, s', h.P_{c(h,\mathcal{I})}) \quad \langle c(h,\mathcal{I}), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \\ \langle h?m, s'\rangle \overset{\text{bw}}{\rightsquigarrow} (s', z) \quad s'[h.False \leq z.x \wedge z.y < h.True]\end{array}}{\langle \textbf{lazy}(h), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', uu_f)}$$

$$\frac{\begin{array}{c}\langle c(h,\mathcal{I}), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \quad \langle h?m, s'\rangle \overset{\text{bw}}{\rightsquigarrow} (s', \iota(h, s', h.P_{c(h,\mathcal{I})})) \\ \iota(h, s', h.P_{c(h,\mathcal{I})}).x \geq h.True\end{array}}{\langle \textbf{lazy}(h), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', tt)}$$

$$\frac{\begin{array}{c}\langle c(h,\mathcal{I}), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \quad \langle h?m, s'\rangle \overset{\text{bw}}{\rightsquigarrow} (s', \iota(h, s', h.P_{c(h,\mathcal{I})})) \\ \iota(h, s', h.P_{c(h,\mathcal{I})}).y < h.False\end{array}}{\langle \textbf{lazy}(h), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', ff)}$$

Some rules for **prv**:

$$\frac{s[h.x \geq h.True]}{\langle \textbf{lazy}(\textbf{prv}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s, tt)}$$

$$\frac{s[h.y < h.False]}{\langle \textbf{lazy}(\textbf{prv}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s, ff)}$$

$$\frac{s[h.x \geq h.False \ \wedge \ h.y < h.True]}{\langle \textbf{lazy}(\textbf{prv}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s, uu_f)}$$

$$\frac{\begin{array}{c}\text{let } z = \iota(h, s', h.P_{c(h,\mathcal{I}_t)}) \quad \langle c(h,\mathcal{I}_t), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \\ \langle h?m, s'\rangle \overset{\text{bw}}{\rightsquigarrow} (s', z) \quad s'[h.False \leq z.x \wedge z.y < h.True]\end{array}}{\langle \textbf{lazy}(\textbf{prv}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', uu)}$$

$$\frac{\begin{array}{c}\langle c(h,\mathcal{I}_t), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \quad \langle h?m, s'\rangle \overset{\text{bw}}{\rightsquigarrow} (s', \iota(h, s', h.P_{c(h,\mathcal{I}_t)})) \\ \iota(h, s', h.P_{c(h,\mathcal{I})}).x \geq h.True\end{array}}{\langle \textbf{lazy}(\textbf{prv}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', tt)}$$

$$\frac{\begin{array}{c}\langle c(h,\mathcal{I}_t), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \quad \langle h?m, s'\rangle \overset{\text{bw}}{\rightsquigarrow} (s', \iota(h, s', h.P_{c(h,\mathcal{I}_t)})) \\ \iota(h, s', h.P_{c(h,\mathcal{I})}).y < h.False\end{array}}{\langle \textbf{lazy}(\textbf{prv}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', ff)}$$

and some rules for **rft**:

$$\frac{s[h.x \geq h.True]}{\langle \textbf{lazy}(\textbf{rft}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s, tt)}$$

$$\frac{s[h.y < h.False]}{\langle \textbf{lazy}(\textbf{rft}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s, ff)}$$

$$\frac{s[h.x \geq h.False \ \wedge \ h.y < h.True]}{\langle \textbf{lazy}(\textbf{rft}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s, uu_f)}$$

$$\frac{\begin{array}{c}\text{let } z = \iota(h, s', h.P_{c(h,\mathcal{I}_f)}) \quad \langle c(h,\mathcal{I}_f), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \\ \langle h?m, s'\rangle \overset{\text{bw}}{\rightsquigarrow} (s', z) \quad s'[h.False \leq z.x \wedge z.y < h.True]\end{array}}{\langle \textbf{lazy}(\textbf{rft}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', uu)}$$

$$\frac{\begin{array}{c}\langle c(h,\mathcal{I}_f), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \quad \langle h?m, s'\rangle \overset{\text{bw}}{\rightsquigarrow} (s', \iota(h, s', h.P_{c(h,\mathcal{I}_f)})) \\ \iota(h, s', h.P_{c(h,\mathcal{I})}).x \geq h.True\end{array}}{\langle \textbf{lazy}(\textbf{rft}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', tt)}$$

$$\frac{\begin{array}{c}\langle c(h,\mathcal{I}_f), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \quad \langle h?m, s'\rangle \overset{\text{bw}}{\rightsquigarrow} (s', \iota(h, s', h.P_{c(h,\mathcal{I}_f)})) \\ \iota(h, s', h.P_{c(h,\mathcal{I})}).y < h.False\end{array}}{\langle \textbf{lazy}(\textbf{rft}(h)), s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', ff)}$$

On the other hand, for uncertain strict computation, the request for the value of a hypothesis can also mean to exploit all its premises in order to get a more accurate certainty measure. The order of computation in the list of premises is static as follows:

$b(h,\mathcal{I}) \ = \ $ **if** $d(h,\mathcal{I}) \ = \ -2$ **then** h **else** $b(h, h.P, \mathcal{I})$;

Ternary b
$b(h, P, \mathcal{I}) \ = \ h$ **if** $P = \emptyset$;
$b(h, P, \mathcal{I}) \ = \ $ **let** $p = newstate(P_1)$ **in**
$b(U(h, p.x \cdot h.P_1.??, p.y \cdot h.P_1.??), P \backslash \{P_1\}, \mathcal{I})$ **otherwise**;

The operational semantics of the strict evaluation is as follows:

$$\frac{s[h.v = uu_f]}{\langle h, s\rangle \overset{\text{bw}}{\rightsquigarrow} (s, h.v)}$$

$$\frac{\begin{array}{c}s[h.v \neq uu_f \wedge h.x < h.y] \wedge \langle b(h,\mathcal{I}), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \\ \iota(h, s', h.P_1).x \geq h.False \wedge \iota(h, s', h.P_1).y < h.True\end{array}}{\langle h, s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', uu_f)}$$

$$\frac{\begin{array}{c}s[h.v \neq uu_f \wedge h.x < h.y] \quad \langle b(h,\mathcal{I}), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \\ \iota(h, s', h.P_1).y \geq h.True\end{array}}{\langle h, s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', tt)}$$

$$\frac{\begin{array}{c}s[h.v \neq uu_f \wedge h.x < h.y] \quad \langle b(h,\mathcal{I}), s\rangle \overset{\text{bw}}{\rightsquigarrow} s' \\ \iota(h, s', h.P_1).x < h.False\end{array}}{\langle h, s\rangle \overset{\text{bw}}{\rightsquigarrow} (s', ff)}$$

D. Inference Operator - Inferop

Inferops result in a pair of type $(\mathbb{F} \times \mathbb{F}) \times \mathbb{V}$, in which the first member is a pair which in turn consists of the *at least* and the *at most* certainty measures, in $[-1.0, +1.0]$, and the second member, in the outermost pair, is a logical value in $\{ff, uu, tt\}$. The operational semantics for the built-in inferops are:

Uncertain and:

$$\frac{\langle a, s\rangle \overset{\text{eval}}{\rightsquigarrow} (u, \alpha) \quad \langle b, s\rangle \overset{\text{eval}}{\rightsquigarrow} (v, \beta) \quad min(u.y, v.y) < T}{\langle \textbf{uand}(T, a, b), s\rangle \overset{\text{bw}}{\rightsquigarrow} ((min(u.x, v.x), min(u.y, v.y)), ff)}$$

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\alpha) \quad \langle b,s\rangle \overset{\text{eval}}{\rightsquigarrow} (v,\beta) \quad min(u.x,v.x) < T \wedge min(u.y,v.y) \geq T}{\langle \mathbf{uand}(T,a,b),s\rangle \overset{\text{bw}}{\rightsquigarrow} ((min(u.x,v.x),min(u.y,v.y)),uu_f)}$$

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\alpha) \quad \langle b,s\rangle \overset{\text{eval}}{\rightsquigarrow} (v,\beta) \quad min(u.x,v.x) \geq T}{\langle \mathbf{uand}(T,a,b),s\rangle \overset{\text{bw}}{\rightsquigarrow} ((min(u.x,v.x),min(u.y,v.y)),tt)}$$

Uncertain or:

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\alpha) \quad \langle b,s\rangle \overset{\text{eval}}{\rightsquigarrow} (v,\beta) \quad max(u.y,v.y) < T}{\langle \mathbf{uor}(T,a,b),s\rangle \overset{\text{bw}}{\rightsquigarrow} ((max(u.x,v.x),max(u.y,v.y)),ff)}$$

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\alpha) \quad \langle b,s\rangle \overset{\text{eval}}{\rightsquigarrow} (v,\beta) \quad max(u.x,v.x) < T \wedge max(u.y,v.y) \geq T}{\langle \mathbf{uor}(T,a,b),s\rangle \overset{\text{bw}}{\rightsquigarrow} ((max(u.x,v.x),max(u.y,v.y)),uu_f)}$$

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\alpha) \quad \langle b,s\rangle \overset{\text{eval}}{\rightsquigarrow} (v,\beta) \quad max(u.x,v.x) \geq T}{\langle \mathbf{uor}(T,a,b),s\rangle \overset{\text{bw}}{\rightsquigarrow} ((max(u.x,v.x),max(u.y,v.y)),tt)}$$

Uncertain not:

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\gamma) \quad 1-u.x < T}{\langle \mathbf{unot}(T,a),s\rangle \overset{\text{bw}}{\rightsquigarrow} ((1-u.y,1-u.x),ff)}$$

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\gamma) \quad 1-u.y < T \wedge u.x \geq T}{\langle \mathbf{unot}(T,a),s\rangle \overset{\text{bw}}{\rightsquigarrow} ((1-u.y,1-u.x),uu_f)}$$

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\gamma) \quad 1-u.y \geq T}{\langle \mathbf{unot}(T,a),s\rangle \overset{\text{bw}}{\rightsquigarrow} ((1-u.y,1-u.x),tt)}$$

Note that **unot** inverts the minimum and maximum values and swaps the positions.

Determinable and: (for any $f > 0$)

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\alpha) \quad \langle b,s\rangle \overset{\text{eval}}{\rightsquigarrow} (v,\beta) \quad u.y < T \vee v.y < T}{\langle \mathbf{dand}(T,a,b)\ \mathbf{cnf}f,s\rangle \overset{\text{bw}}{\rightsquigarrow} ((-f,-f),ff)}$$

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\alpha) \quad \langle b,s\rangle \overset{\text{eval}}{\rightsquigarrow} (v,\beta) \quad (u.x < T \vee v.x < T) \wedge u.y \geq T \wedge v.y \geq T}{\langle \mathbf{dand}(T,a,b)\ \mathbf{cnf}f,s\rangle \overset{\text{bw}}{\rightsquigarrow} ((0,0),uu_f)}$$

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\alpha) \quad \langle b,s\rangle \overset{\text{eval}}{\rightsquigarrow} (v,\beta) \quad u.x \geq T \wedge v.x \geq T}{\langle \mathbf{dand}(T,a,b)\ \mathbf{cnf}f,s\rangle \overset{\text{bw}}{\rightsquigarrow} ((f,f),tt)}$$

Determinable or:

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\alpha) \quad \langle b,s\rangle \overset{\text{eval}}{\rightsquigarrow} (v,\beta) \quad u.y < T \wedge v.y < T}{\langle \mathbf{dor}(T,a,b)\ \mathbf{cnf}f,s\rangle \overset{\text{bw}}{\rightsquigarrow} ((-f,-f),ff)}$$

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\alpha) \quad \langle b,s\rangle \overset{\text{eval}}{\rightsquigarrow} (v,\beta) \quad u.x < T \wedge v.x < T \wedge (u.y \geq T \vee v.y \geq T)}{\langle \mathbf{dor}(T,a,b)\ \mathbf{cnf}f,s\rangle \overset{\text{bw}}{\rightsquigarrow} ((0,0),uu_f)}$$

$$\frac{\langle a,s\rangle \overset{\text{eval}}{\rightsquigarrow} (u,\alpha) \quad \langle b,s\rangle \overset{\text{eval}}{\rightsquigarrow} (v,\beta) \quad u.x \geq T \vee v.x \geq T}{\langle \mathbf{dor}(T,a,b)\ \mathbf{cnf}f,s\rangle \overset{\text{bw}}{\rightsquigarrow} ((f,f),tt)}$$

for any $f > 0$.

Finally, users are able to define their own inferops.

V. Conclusion

The Internet has raised many issues in programming. The fact that connections are neither reliable nor efficient makes us consider the possibility of programming with uncertainty[10].

I formally introduced uncertainty as a programming language feature, considering the current scenario in the presence of code mobility and the Internet. Additionally, the present uncertainty model permits evaluation with partial information, by considering *unknown*[3] as part of the model in both the variables used as premises and in resulting hypotheses. I also introduced a number of inference operators. As a result, *uu* can permit the unification of this paradigm with others.

Acknowledgement: I would like to thank Hélio Silva for having discussed the present subject.

References

[1] L. Cardelli. Global computation. *ACM Computing Surveys*, 28A(4), 1996.

[2] U. Ferreira. Chiron: a framework for mobile agent systems. In G. E. Lasker, J. Dospisil, and E. Kendall, editors, *Advances in Mobile Agents Systems Research. Proceedings of the 12th International Conference on System Research, Informatics & Cybernetics*, volume 1: Theory and Applications, pages 12–22. The International Institute for Advanced Studies in Systems Research and Cybernetics, August 2000.

[3] U. Ferreira. *uu* for programming languages. *ACM SIGPLAN Notices*, 35(8):20–30, August 2000.

[4] U. Ferreira. Programming languages features for some global computer. In *Proceedings of SSGRR 2003s International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, e-Medicine, and Mobile Technologies on the Internet*. Scuola Superiore G. Reiss Romoli e Telecom Italia Learning Services, From 28 July to 3 August 2003.

[5] I. Hacking. *What Is a Logical System?*, chapter What Is Logic?, pages 1–33. Number 4 in Studies in Logic and Computation. Claredon Press, Oxford University, 1994.

[6] N. K. Kasabov. *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering*. The MIT Press, Cambridge, Massachusetts; London, England, 1996.

[7] B. Kosko. *Fuzzy Thinking: The New Science of Fuzzy Logic*. Harper-CollinsPublishers, Flamingo, 1994.

[8] H. Kyburgh Junior. *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 3: Nonmonotonic Reasoning and uncertain reasoning, chapter Uncertainty Logics, pages 397–438. Oxford University Press, 1994.

[9] E. H. Shortlife. *Computer-Based Medical Consultations: MYCIN*. New York, 1976. Elsevier.

[10] J. F. Sowa. *Knowledge Representation: logical, philosophical, and computational foundations*. Brooks/Cole, 511 Forest Lodge Road, Pacifi c Grove,CA, 2000.