GENERIC EMBEDDING OF AN UNCERTAIN CALCULUS IN OBJECTS AND RULES

Jérôme EUZENAT, Michel LE, Éric MAZERAN, Michel WEINBERG

ILOG
2, avenue Galliéni, BP85, 94253 GENTILLY Cedex, FRANCE

Abstract

While symbolic knowledge representation and reasoning methods are necessary for almost any kind of knowledge-based application, they often lack numerically represented uncertainty and vagueness. Meanwhile, different applications would require different numeric calculi. SMECI Uncertain Module (SUM) enables to embed an uncertain (or graded) calculus into a multi-paradigm environment (including tasks, rules, objects and multiple-worlds), allowing therefore the object model to take into account uncertain values so that the inference engine can draw uncertain inferences from uncertain and vague premises. The originality of SUM is that it does not make strong assumptions about the calculus used, which only has to respect some fundamental "format" expressed through the design of basic objects and the instantiation of a set of generic primitives. Therefore, SUM is not restricted to numeric truth values but can deal with any kind of values provided with an implementation of the generic interface.

Keywords: uncertainty — vagueness — fuzzy logic — object-based knowledge representation — inference engine.

SMECI

SMECI [3] is a knowledge-based system development environment providing several representation paradigms:

objects to declare the static knowledge as instances of classes and collections of slots.

tasks to represent the applications in structural units (implemented by rules, functions...) on which the control is defined.

rules to draw inferences from premises to conclusions. SMECI provides a powerful rule language including access to objects, logic connectors, variables and quantifications (over a set of known individuals).

worlds which record a particular state of the base. This allows to provide a fine control in a SMECI application by enabling to implement backtracking, concurrent worlds...

Systems such as SMECI are very well suited to build embedded and efficient knowledge-based systems. Meanwhile, their rules and objects only allow to take into account binary truth values (the assertions are true or false). Thus there is a strong need for expressing vagueness and uncertainty. This is the "raison d'être" of SUM.

Uncertainty and vagueness

It is often necessary, for a particular application to go beyond the boolean model of valuation. The needs for non-boolean calculus are manifold (see [1] for more details):

- vagueness enables to represent assertions like "the temperature is high". The assertion itself is vague because it uses the unprecise term "high", but its truth value is precise: it is true or false.
- uncertainty enables to express the vagueness about truth values instead of assertions (e.g. "the temperature is probably (resp. certainly, surely) of 20°").

Vagueness (as well as uncertainty) can be taken into account by quantifying the degree of likelihood (or certainty) for the assertion to be true. Propagating these degrees allows to shade the answers given by a classical knowledge-based system with regard to the vague or uncertain aspect of the initial knowledge.

In order to provide SMECI with non-boolean operators (manipulating uncertain truth values), two main requirements have been emphasized:

integration Non boolean operators must be closely embedded in SMECI in order to be available in each paradigm and to benefit from the interactive development environment.

genericity Since there is not a unique way to model uncertainty and since each application

has its own needs, it is necessary to provide a tool able to handle various calculi.

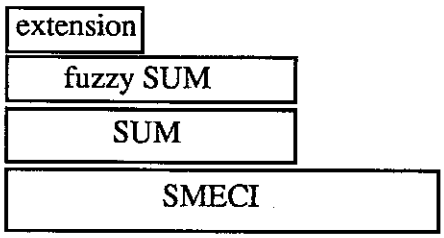

Fig. 1. Layered decomposition of SMECI fuzzy module.

These two requirements have been taken into account in designing the SMECI Uncertain Module (SUM). This module is made of a set of generic primitives which can be overloaded (redefined) in order to deal with a particular implementation of a numeric calculus. The power of object representation enables to closely integrate the uncertain values in the SMECI environment so that these values can be refined for implementing particular behaviors. This aproach has proved its viability in Fuzzy SUM, an instantiation of SUM in fuzzy logic [4].

Fuzzy logic

The respective advantages of different ways of representing uncertainty are not presented here; [1] can be consulted for the properties expected from these representations. The fuzzy logic uses a continuum of truth values (which are called degrees). They are represented by reals in the interval [0. 1.]. The truth values are assigned to the assertions. When assertions are combined together with the help of logical connectors ($\wedge$,$\vee$,$\neg$), the degree of the resulting assertion is the combination of the primitive assertions with operators tied to the connectors. These operators are the minimum for $\wedge$, maximum for $\vee$ and complementation to 1 for $\neg$.

Thus, if the assertion "the temperature is high" is assigned a degree of .8 and "the pressure is 120" is assigned .6, then "the temperature is high or the pressure is 120", "the temperature is 20 and the pressure is 120" and "the pressure is not 120" will be respectively assigned .8, .6 and .4.

Fuzzy logic allows not only to assign degrees but also to propagate these degrees from assertions to others. This will be used in reasoning.

Embedding the calculus in SUM

Generally speaking, SUM is grounded on a very basic calculus model. The values (called degrees in the remainder) must range within an interval of an ordered domain. The calculus must include operators able to combine degrees together with regard to logical connectors ($\wedge$,$\vee$,$\neg$). For example, Fuzzy SUM uses the calculus of fuzzy logic (domain of reals within the [0. 1.] range, three connectors corresponding respectively to min, max and complementation to 1). The calculus is extended to extentionnal $\exists$ and $\forall$ quantifier as n-ary $\wedge$ and $\vee$ operators (returning their identity element when provided with zero argument). This implements the minimal calculus on which an instance of SUM is grounded. In Fuzzy SUM, they correspond to the `S-fuzzy-and`, `S-fuzzy-or`, `S-fuzzy-not`, `S-fuzzy-any` and `S-fuzzy-every` primitives. Additionnal functions are provided, for example for checking if a value is indeed a degree.

Objects and uncertain slot values

Object-based knowledge representation has already proved its flexibility and power. The full power of objects is used for implementing uncertain slots values in SMECI. SUM considers as uncertain the slots whose value domain is an object, instance (local or remote) of the uncertain-slot class. Thus, uncertain slot values are implemented as objects. The uncertain-slot class is provided by the instantiation of SUM. Like any other class, it can be refined in order to implement a particular behavior. For example, in Fuzzy SUM, the class `S.FuzzyValue` is the root class for fuzzy slots. It has a slot called `value` which contains, for each value of the slot, the value-degree pair. `S.FuzzyValue` is refined in `S.MultiFuzzyValue` whose value is a fuzzy set expressed in extension.

The manipulation of uncertain values uses the generic interface of these objects through primitives enabling to get, set or remove the value of an (ordinary) object slot (`S-fuzzy-get-value`, `S-fuzzy-remove-value`, `S-fuzzy-set-value`) or through primitive manipulating the degree of the slot value (`S-fuzzy-get-degree`). These primitives use the generic interface of the `S.FuzzyValue` objects (methods `get-value`, `rem-value`, `set-value` and `get-degree`) which can be overloaded.

Therefore, instead of representing value-degree pairs like in the `S.FuzzyValue` class, a

`funcFuzzyValue` class can be defined whose behavior is to compute the degree instead of storing it. It has a `function` slot containing the name of the function to use in order to compute the degree associated to a value. This function will be used by the redefinition of the `get-degree` method. This allows to implement fuzzy sets based on functions.

The SUM inference system

SUM allows the declaration of "uncertain rules". These rules are handled by a particular engine which considers the graded truth degrees as token manipulated through generic operators. This section describes the processing of the engine assuming that basic premises are assigned an uncertain truth degree. Next section describes how premises are handled.

In SMECI, rule premises are combined by using the logical operators: `or`, `and`, `not`, `thereexist`, `foreach`. In SUM, they are overloaded by the specialized connectors provided by the instantiation calculus (`S-fuzzy-or`, `S-fuzzy-and`, `S-fuzzy-not`, `S-fuzzy-any` and `S-fuzzy-every`).

Once the premise degrees have been combined using the connectors, the engine is provided with an instantiation set of the rules variables and a uncertain truth degree (called the *firing degree*). At that stage, it is possible to control the firing of the rule according to its firing degree. The engine invokes a predicate (`S-fuzzy-considerable-p`) which will return `true` if the rule is to be fired. This predicate can be overloaded for implementing a particular system. In Fuzzy SUM, it checks that the firing degree is superior to a given threshold (whose default value is 0.) and a primitive is provided which enables to set the threshold value.

Once the rule is fired, the engine will perform the actions in conclusions of the rules. These actions can be divided in two sets: the logical actions which assert the values of some slots and the procedural actions which run a LISP procedure. In the second case, the procedure can eventually consult the firing degree with the help of the `S-rule-conclusion-cf` function. Thus, the rule of Fig. 2 is able to qualify the action on the input valve by taking the firing degree into account.

Each logical conclusion is graded with a *confidence degree* local to the rule and uses the firing degree resulting from premise filtering. The logical actions are performed by a generic function on uncertain slot values whose arguments are the uncertain slot value to be modified, the firing degree and the confidence degree associated to the conclusion. This function can then compute the degree assigned to the asserted value in an appropriate way. For example, in Fuzzy SUM, the degree associated with the value is the result of the maximisation of its previously holding degree and the minimization of the firing degree and the confidence degree. Thus, the effect of the rule is constrained to set a value with a degree:

- lower than the confidence degree given to the conclusion;
- higher than the degree already holding for the value.

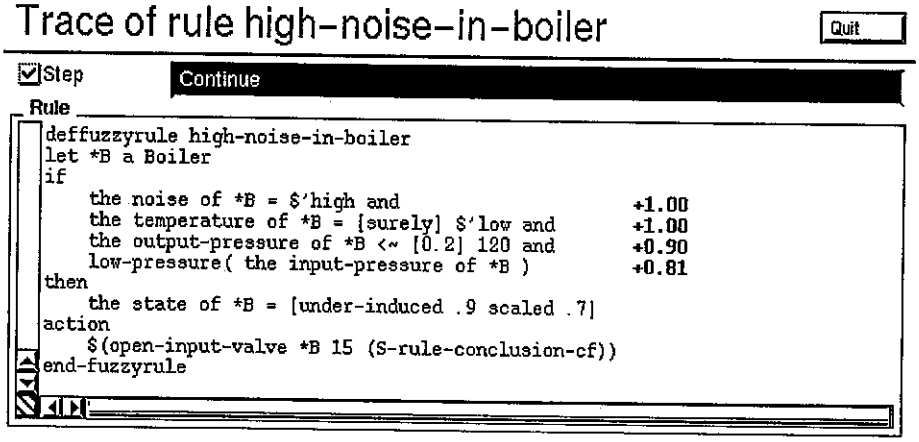

Fig. 2. The SMECI rule tracer is able to trace fuzzy rules taking into account the fuzzy values. Degrees on the right are the premise degrees. It is noteworthy that almost no degree appear in the rule definition.

In Fig. 2, the firing degree is .81 (minimum of 1., 1., .9 and .81). Thus, if there is no other value in the slot, the values assigned to `state` of `*B` will be of .81 (minimum of .81 and .9) for `under-induced` and .7 (minimum of .81 and .7) for `scaled`. If `scaled` already had a degree of .9 then the rule has no effect on that degree (.9 would be the maximum between .81 and .9).

While SMECI is not properly a logic prover, if rules only use logical conclusions and 0. as truth threshold, Fuzzy SUM will respect the fuzzy logic semantics. The firing of such a rule is equivalent to a fuzzy proof by elimination for each of the value-degree pairs asserted in conclusion.

Vague and uncertain premises handling

Not only can SUM rules handle both uncertain and vague premises, but also boolean ones (for instance, in the rule of Fig. 2, the first premise is a boolean one). An overloadable function `S-lisp-to-fuzzy`, used by the inference engine, enables to coerce a boolean truth value into a uncertain truth degree. In Fuzzy SUM, it converts `false` to 0. and `true` to 1. On the other hand it is possible to use fuzzy slots values in normal rules, but they have to be defuzzyfied. The very simple defuzzyfication function `S-fuzzy-to-lisp` is provided, which turns to true each degree superior to a given threshold and others to zero. Nonetheless, it is possible to define other defuzzification functions.

SMECI rules use comparators for the values such as equal (=), superior (>), belongs to (&),... The premise "`the temperature of *B = [surely] low`" is such a premise (using =). SUM allows these comparators to be overloaded. If the values to compare are those of uncertain slots, the unifier will invoke user-defined comparators which can return the appropriate truth degree. In Fuzzy SUM, these comparators use the fuzzy set theory for computing the degree of compliance with the fuzzy predicate.

Moreover, SMECI allows to use tolerant numeric comparators which return boolean values if the predicate is satisfied in a particular interval (e.g. "the temperature is around 20", can be rewritten as "the temperature is equal to 20 with a tolerance of 2.5%"). In SUM, this tolerance can be used for computing a degree. For example, in Fuzzy SUM, the degree of the former assertion will be graded to 1. if the temperature is 20, to 0. if it is 19.5 and to .5 if it is 19.75. In Fig. 2, the `output-pressure` of the boiler is compared with 120 allowing a tolerance of 20%. These vague predicates are also overloadable and can, in Fuzzy SUM, use a parameterized interpolation function (which, of course, decreases monotonically from 1. to 0.). This allows to convert vagueness into certainty.

Another feature of SUM is the opportunity to add modalities. Modalities are functions internal to the degree domain. They enable to strengthen or weaken the degree assigned to a premise. They correspond to natural language scalar adverbs such as "certainly", "surely", "maybe"... From a list of such scalar adverbs, Fuzzy SUM can define a complete set of modalities based on the interpolated inclusion in an interval. In short, modalities apply vagueness to uncertain degrees. In Fig. 2, `surely` is used with the equality comparator in order to reinforce the premise degree. Fig. 3 shows a set of modalities as they are defined automatically.

Finally, the rule language of SMECI is extendible. It is thus possible to add other fuzzy predicates in the rule language. This is the case, in Fig. 2, of the `low-pressure` predicate.

Uncertain rules firing control

The evaluation and firing of rules are managed by inference control structures. SMECI's inference engine provides several facilities for representing inference structures and for parameterizing the firing of rules. There are two control layers:

- the task layer which deals with macroscopic control of the reasoning process. It is independent from the evaluation of rules and thus is not affected by uncertainty;
- the rule-base control layer which aims at defining a local control space for a given rule set.

As far as uncertainty handling is a means to measure the conclusiveness of a rule on a gradual scale, the engine can take advantage of this in order to solve rule conflict. Thus, several control strategies are provided with SUM which enable:

- the parallel firing of distinct rules (in different worlds) on their best instantiation set (i.e. the one with the higher firing degree).
- the firing of the rule which has the best firing degree only on the instanciation set corresponding to that degree.

Moreover, the integration of SUM in the inference engine allows to use the fuzzy module in the control strategy (e.g. by using a best-first strategy grounded on fuzzy evaluation).

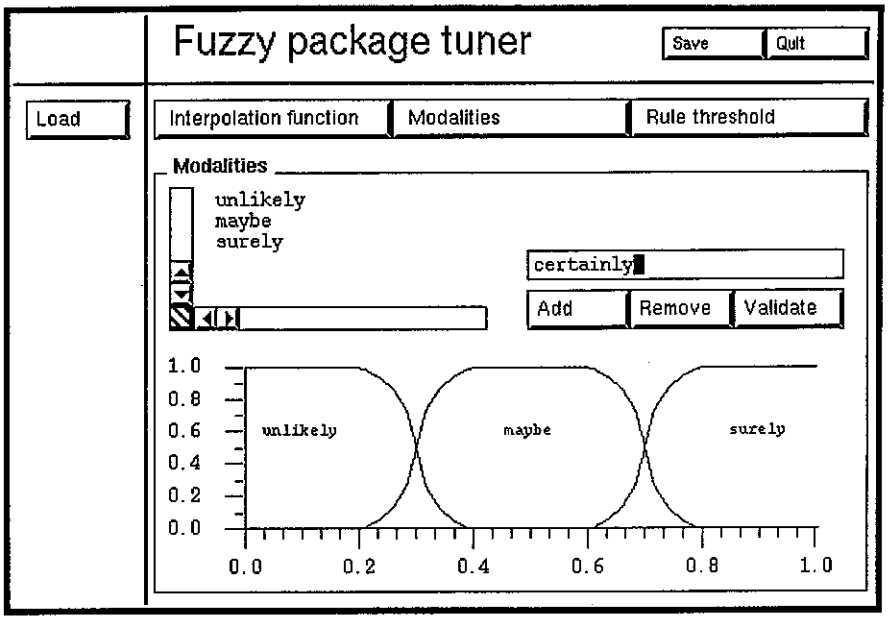

Fig. 3. Ordered modalities can be defined automatically by using one of the features of the fuzzy tuner. They are intervals homogeneously distributed throughout the interval [0 1] whose extremities smoothly decrease due to the interpolation function.

Conclusion

The SMECI uncertain module is already implemented and commercialized together with Fuzzy SUM. The architecture of SUM (see Fig. 1.) allows for three levels of evolution. First, Fuzzy SUM can be tuned with the interpolation function, the definition of modalities and thresholds through the Fuzzy tuner. Second, Fuzzy SUM can be extended by defining new classes of fuzzy slots and new fuzzy predicates and modalities. Last, Fuzzy SUM can be totally replaced by another instantiation based on another calculus.

While only numeric calculi have been discussed here, the generic feature of the module can be used in order to integrate a wide range of calculi (following the methodology of SaMaRis [2] for TMS). Hence, the values could be possibility couples [1], numeric or temporal intervals, fuzzy functions, ATMS environments...

References

[1] D. Dubois and H. Prade, Possibility theory: an approach to computerized processing and uncertainty, New-York: Plenum press, 1988

[2] J. Euzenat and L. Buisson, SaMaRis, un environnement pour l'expérimentation et l'étude du maintien des raisonnements, in Proceedings of the 8ième congrès reconnaissance des formes et intelligence artificielle, 1991, pp. 1233-1248.

[3] SMECI, reference manual, 1.65 release, ILOG, Gentilly (FR), May 1992

[4] L. Zadeh, Fuzzy sets as a basis for a theory of possibility, *Fuzzy sets and systems* Vol. 1, pp. 3-28, 1978