IFAC

International Federation of Automatic Control

IFAC

PREPRINTS
Volume III
Parallel Sessions

9TH SYMPOSIUM ON INFORMATION CONTROL IN MANUFACTURING

INCOM'98

Advances in Industrial Engineering

June 24-26, 1998
NANCY - METZ, FRANCE

Organised by

CRAN

Sponsored by

CNRS CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

INRIA

IFP

Photo Ville de Metz

IFAC

INTERNATIONAL FEDERATION OF AUTOMATIC CONTROL

IFAC

PREPRINTS OF THE

9th SYMPOSIUM ON INFORMATION CONTROL IN MANUFACTURING

June 24-26, 1998
NANCY - METZ, FRANCE

Advances in Industrial Engineering

Volume III
Parallel Sessions

Edited by

Gérard MOREL (CRAN - Henri Poincaré University of Nancy 1)

François B. VERNADAT (LGIPM, Metz University)

HANDLING FUZZY INFORMATION IN MILORD II[1]

Josep Puyol-Gruart, Lluís Godo, Carles Sierra

Artificial Intelligence Research Institute (IIIA)
Spanish Scientific Research Council (CSIC)
Campus UAB. 08193 Bellaterra, Catalonia, Spain
E-mail: {puyol,godo,sierra}@iiia.csic.es

Abstract: **Milord II** is a modular language for knowledge-based systems. In this paper we concentrate on the parts of the language and the theoretical foundations related to approximate reasoning: the use of local many-valued logics based on linguistic terms, which are the language constructs related to fuzzy sets, and finally the language deductive mechanism. *Copyright © 1998 IFAC*

Keywords: Knowledge-Based Systems; Many-valued Logic; Fuzzy Logic.

1. INTRODUCTION

Milord II is a modular language for knowledge-based systems. In this paper we concentrate on the parts of the language and the theoretical foundations related to approximate reasoning and fuzzy modelling. In (Puyol-Gruart, 1996; Puyol-Gruart and Sierra, 1997; Puyol-Gruart *et al.*, 1998) the interested reader can find a complete description of the language and its logical semantics.

The structural construct of **Milord II** is the *module*. A program consists of a set of modules that can recursively contain other modules, then forming a hierarchy.

A module declaration contains the next sets of components: *hierarchy*, *interface*, *deductive knowledge* and *control knowledge* (In Figure 1 we can see the code of an example of module declaration).

Hierarchy: Is a set of submodule declarations.

Interface: It has two components, the import and the export interface. That is, which variables and propositions may be asked to the user (import), and those whose value is computed by the module (export).

Deductive knowledge: It contains a dictionary declaration, that is, the set of variables and propositions (those belonging to the interface of the module and other intermediate ones) and their attributes; a set of rule declarations, that is, a set of propositional weighted rules; and an inference system declaration, that is, the local logic of the module.

Control knowledge: It is expressed in a meta-language which acts by reflection over the deductive knowledge and the hierarchy of submodules[2].

In the subsequent sections we will describe in detail the different elements of the deductive component of a module.

We begin with the logical foundations of Milord II, based on a family of many-valued logics. First, we describe the algebras of truth-values. After that, we concentrate on variables, propositions and rules. Variables and propositions are the simplest knowledge representation units and propositions and predicates over expressions containing variables are the basic elements to build rules. Finally the deductive system based on *specialisation* is sketched.

[1] This research has been partially supported by the Spanish Comisión Interministerial de Ciencia y Tecnol... through the project SMASH (TIC96-...

[2] A detailed explanation of this component can be found ... Puyol-Gruart, ...

```
Module example =
  Begin
      Export fever, seriousness
      Import temperature
      Deductive knowledge
      Dictionary:
          Predicates:
          temperature= name: "Temperature"
                       question: "Which is the temperature?"
                       type: numeric
          fever4 = type: (l "low" (37,37.3,37.6,38),
                          m "medium" (37.6,38,38.5,39),
                          h "high" (38.5,39,43,43))
                   Relation: needs_quantitative temperature
          Rules:
            R001 If fever4 is high then conclude seriousness is q_true
          Inference system:
             Truth values = (false, sl_true , q_true, f_true, true)
             Conjunction = Truth table
                ((false false false false false)
                (false sl_true sl_true sl_true sl_true)
                (false sl_true q_true q_true q_true)
                (false sl_true q_true f_true f_true)
                (false sl_true q_true f_true true))
      End deductive
  End
```

Fig. 1. Example of a module declaration.

2. LOCAL LOGICS

The logical foundations of Milord II have been the main topic of several papers, for instance (Godo *et al.*, 1989; López de Màntaras, 1990; Esteva *et al.*, 1994; Puyol-Gruart *et al.*, 1998). The approximate reasoning capability of Milord II is based on a family of finitely-valued logics which are local to each module and declared by an algebra of truth-values.

An Algebra of truth-values $A_{n,T} = \langle A_n, \preceq, N_n, T, I_T \rangle$ is a finite linearly ordered residuated lattice with a negation operation, that is:

Truth-values: $\langle A_n, \preceq \rangle$ is a chain of n elements (user-defined linguistic terms): $0 = a_1 \preceq a_2 \preceq \ldots \preceq a_n = 1$ where the bottom 0 and the top 1 are the booleans *False* and *True* respectively. For instance:

```
Truth values =
      (false, sl_true , q_true,  f_true, true)
```

where `sl_true`, `q_true` and `f_true` stand for *slightly true*, *quite true* and *fairly true* respectively.

Conjunction: the operator T is a binary operation such that the following properties hold $\forall a, b, c \in A_n$:

- T1: $T(a,b) = T(b,a)$
- T2: $T(a, T(b,c)) = T(T(a,b), c)$
- T3: $T(0,a) = 0$
- T4: $T(1,a) = a$
- T5: if $a \preceq b$ then $T(a,c) \preceq T(b,c)$ for all c

An example of such operator, representing the *min* conjunction, $T(a_i, a_j) = a_{min(i,j)}$, is:

```
Conjunction = truth table
      ((false false   false  false   false)
       (false sl_true  sl_true sl_truey sl_true)
       (false sl_true  q_true  q_true   q_true)
       (false sl_true  q_true  f_true   f_true)
       (false sl_true  q_true  f_true   true))
```

The **negation** operation N_n is a unary operation defined as $N_n(a_i) = a_{n-i+1}$, the only definable order-reversing involutive mapping in $\langle A_n, \preceq \rangle$, i.e. it holds: N1) if $a \prec b$ then $N_n(a) \succ N_n(b), \forall a, b \in A_n$ and N2) $N_n(N_n(a)) = a$. The **implication** operator I_T is defined by residuation with respect to T, i.e. $I_T(a,b) = Max\{c \in A_n \mid T(a,c) \preceq b\}$.

As it is easy to notice from the above definition, an algebra is completely determined as soon as the set of truth-values A_n and the conjunction operator T are chosen. So, varying these two features we may generate different many-valued logics. For instance, taking $T(a_i, a_j) = a_{min(i,j)}$ or $T(a_i, a_j) = a_{min(n,n-i+j)}$ we get the well-known Gödel's or Lukasiewicz's semantics (truth-tables) for finitely-valued logics respectively.

2.1 *Intervals*

After the definition of the algebra of truth-values we extent it to an algebra of intervals of truth-values, this extension is the one actually used in Milord II. We have three reasons to do that: we can see in the modus ponens operation above that we need to deal with intervals if we want to

chain rules; we will see in the next section that to model the communication between modules with different logics we need to use intervals; and finally, imprecision of numerical values is translated to intervals of truth-values for qualitative terms represented as fuzzy sets.

The extension to intervals of the above operators are: $N_n^*([a,b]) = [N_n(b), N_n(a)]$, $T^*([a,b],[c,d]) = [T(a,c), T(b,d)]$.

2.2 *Intermodule Communication*

Different modules can have different local logics. We allow this because an important part of any problem solving method is the way the programmer will deal with the uncertainty in each subproblem: a richer set of linguistic terms can help in giving more precise answers to queries; different connectives represent different rule interpretations, and hence different deductive behaviours; even changing just the name of the terms from a module to another can make the knowledge represented in them more readable.

The main problem that has to be addressed in a system with local logics is how modules communicate and which are the properties that are to be satisfied by that communication process. That is, how a module has to interpret the answer to a query made to a submodule endowed with a different logic. In the case of Milord II we do so by specifying, in the local logic declaration of a module, a renaming function that maps the linguistic terms of the local logics of submodules into intervals of the linguistic terms of the module's local logic.

For instance, the translation of the terms of a module B, B_7 = {false, hard_true, somewhat_true, half_true, quite_true, very_true, definite} to a module containing a local logic with terms A_5 = {false, sl_true, q_true, f_true, true} could be expressed as:

```
Renaming B/false ==> false
         B/sl_true ==> [false, hard_true]
         B/q_true ==> [hard_true, very_true]
         B/f_true ==> [very_true, definite]
         B/true ==> definite
```

Milord II also checks whether the proposed translation between local logics terms satisfies some requirements related to inference preserving criteria (Agustí *et al.*, 1994).

3. PROPOSITIONS AND VARIABLES

Variables and propositions are the simplest knowledge representation units in Milord II. They are named structures that represent the concepts dealt within a module. Their declaration is made by binding an atomic name (identifier) with a set of attributes. The attributes may be a long name, the type, relations with other variables or propositions, a question and user-defined attributes.

The type is the only attribute that is mandatory in variable and proposition declarations and determines the set of allowed values they can take, apart from the special value *unknown*, meaning ignorance of the value.

There are three types of propositions, *boolean*, *many-valued* and *fuzzy*; and three types of variables, *numerical*, *linguistic* and *set* (see Figure 2).

3.1 *Propositions*

Boolean propositions: They are concepts which can only be evaluated to either *false* or *true*. For instance *fever1* would be a boolean predicate (it is false or true):

```
fever1 = Type: boolean
```

Many-valued propositions: The concepts represented as many-valued propositions are those whose truth may be graded. For instance, if we use a subjective criteria to appreciate if a patient has fever, we can declare *fever* as a many-valued proposition (we can say that *fever is f_true*):

```
fever2 = Type: many-valued
```

Those propositions whose belief can be a matter of degree are to be declared as many-valued propostions as well. We consider that the only allowed *belief propositions* are of the form

$$Belief(p),\ Belief(\neg p)$$

where p is an atom. The underlying assumption in this type of propositions is that we somehow identify the belief-degree on p with the truth-degree of a new proposition q saying that "p is *believable*", written $Belief(p)$, that is, q = "$Belief(p)$" and hence q is declared as many-valued (Hájek *et al.*, 1995).

Fuzzy Propositions: Vagueness of concepts as *fever* can be quantified by the degree of membership of a numerical measured value (in this case *temperature*) once *fever* is represented by a fuzzy set over temperatures (see Figure 3).

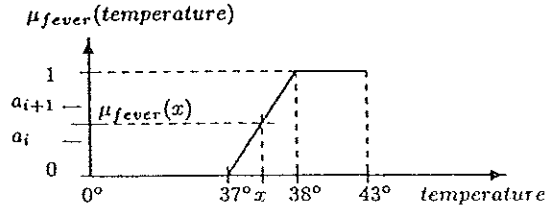

Fig. 3 Fuzzy set representing the concept *fever*.

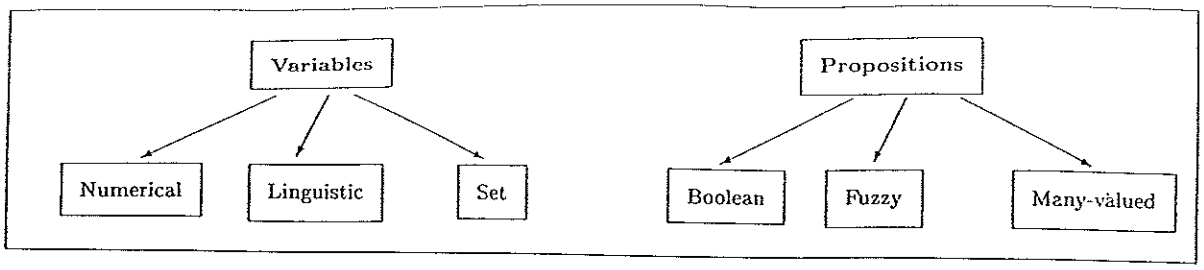

Fig. 2. Types of variables and propositions.

A fuzzy proposition is declared by giving the four points trapezoidal membership function. An example of declaration of *fever* as a fuzzy set is:

```
fever3 = Type: fuzzy (37,38,43,43)
         Relation: needs_quantitative temperature
```

The values of fuzzy propositions are still intervals of linguistic terms. The way of computing the interval is done, in this case, by the application of the fuzzy membership function to the numerical value of the numerical variable appearing in a system-defined relation named *needs_quantitative*. This function returns a number in the interval $[0,1]$. The final answer is the minimum interval of linguistic terms [3] containing that number.

3.2 *Variables*

Numerical variables: The value of a numerical variable is a real number. For instance, *temperature* could be a numerical variable declared as:

```
temperature = Type: numeric
```

Set variables: Set variables are conjunctive fuzzy sets. For instance, we can consider that the variable *treatment* is a (fuzzy) set of *antibiotics*. It is declared given its domain as:

```
treatment=Type:(etambutol, aciclovir, ganciclovir)
```

The value of the set variable *treatment* is the degree of membership of every *antibiotic* to the variable *treatment* (i.e. $\mu_{treatment}(etambutol) = [\alpha, \beta]$), that is, its characteristic function. Given different set variables we could be interested in applying fuzzy set relations and operations. For instance, we can compare different treatments by computing its intersection degree or inclusion degree (see Section 4.1).

Linguistic Variables: They get values from a user-defined finite set of linguistic values (fuzzy sets).

Similarly to the case of fuzzy propositions, we can declare a linguistic variable by giving, for every linguistic value, a trapezoidal fuzzy set with respect to a numerical variable.

[3] By default we consider the set of linguistic terms to be uniformly distributed in the interval $[0,1]$.

In Figure 4 we can see a new representation of the concept *fever* by means of three fuzzy sets, *low*, *medium* and *high*.

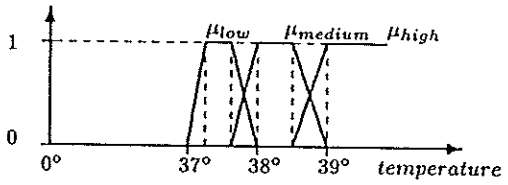

Fig. 4. Fuzzy sets representing the concept *fever*.

The corresponding declaration of this new interpretation of the *fever* concept would be:

```
fever4 = Type: (l "low"    (37,37.3,37.6,38),
                m "medium" (37.6,38,38.5,39),
                h "high"   (38.5,39,43,43))
         Relation: needs_quantitative temperature
```

Notice that as in the case of fuzzy propositions it is necessary to declare the same relation *needs_quantitative*, with a numerical variable; in this case, again *temperature*.

4. RULES

A rule is composed of an identifier, a premise (a conjunction of conditions), a conclusion, and a truth-value (see Figure 5). The truth-value of a rule is a linguistic term belonging to the local logic of the module.

The evaluation of a condition or a conclusion is always an interval of truth-values. In the case of conditions containing variables, the language is provided with a set of predefined predicates that apply on them to produce as result intervals of truth values.

4.1 *Conditions of rules*

Premises of rules are conjunctions of elemental conditions either in affirmative or in negative form (by means of the connective no). Elemental conditions can be: propositions or predicates over expressions containing variables.

Examples of premises containing only proposi-

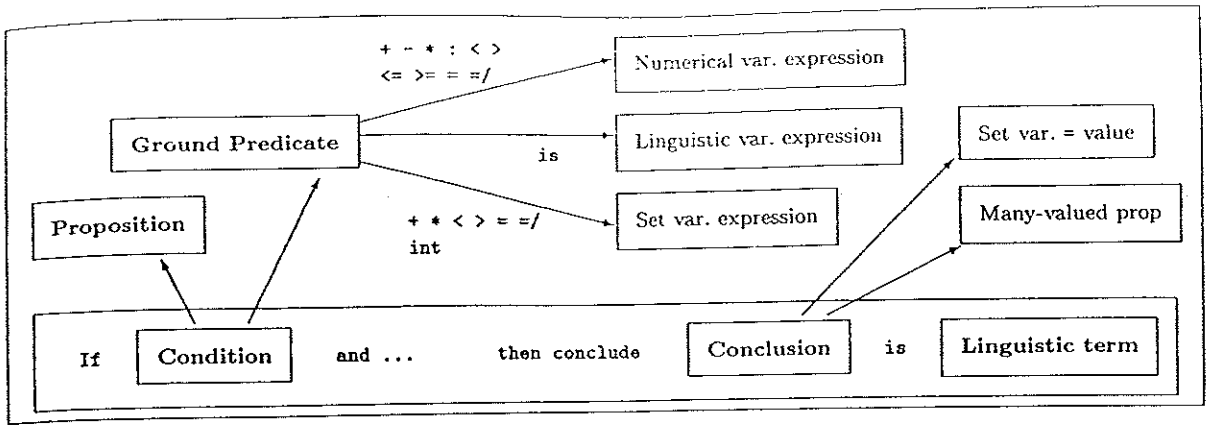

Fig. 5. Rule structure.

```
R003 If allergy then  conclude ...
```

or in negative form:

```
R003 If no(allergy) then  conclude ...
```

Numerical predicates: A numerical expression is composed by numbers, numerical variables and arithmetic operations (+,-,*,:). The evaluation of an expression of this type returns a number over which we can apply predicates like < (less), > (greater), <= (less or equal), >= (greater or equal), = (equal), and / = (different). For instance:

```
R003 If temperature > 39  then  conclude ...
```

Set predicates: We can build set expressions in a similar way as numerical expressions[4] (see Table 1).

Relations		Operations	
syntax	meaning	syntax	meaning
a < b	$A \subset B$	a + b	$A \cup B$
a > b	$A \supset B$	a * b	$A \cap B$
a = b	$A = B$		
a /= b	$A \neq B$		
a int b	$A \cap B \neq \emptyset$		

Table 1. Fuzzy relations and operations.

The operations "+" and "*" are interpreted as the fuzzy set union $\cup$ and the fuzzy set intersection $\cap$ respectively, defined as usual by:

Union: $\forall \omega, \mu_{F \cup G}(\omega) = max(\mu_F(\omega), \mu_G(\omega))$
Intersection: $\forall \omega, \mu_{F \cap G}(\omega) = min(\mu_F(\omega), \mu_G(\omega))$

The allowed fuzzy relations are < (subset), > (superset), = (equal), /= (different) and int (intersection degree). The binary predicates apply over the evaluations of two expressions of the same type. The evaluation of these predicates is, as before, an interval of truth-values.

Now we define the inclusion, intersection degree and equality between two fuzzy sets and their meaning.

Inclusion: $R_{\subset}(F,G) = min(\mu_{F \cup G})$
Intersection degree: $R_{\cap}(F,G) = max(\mu_{F \cap G})$
Equality: $R_{=}(F,G) = min(R_{\subset}(F,G), R_{\supset}(F,G))$

These relations[5] return degrees of inclusion, intersection and equality respectively between two fuzzy sets. Using them we can express, for instance, the degree of inclusion of two treatments:

```
R004 If treatment1 < treatment2 then conclude ...
```

or the degree of intersection of the union of two treatments with a crisp set:

```
R004 If (treatment1 + treatment2) int (aciclovir)
       then conclude ...
```

All these operations are standard (Zadeh, 1965).

Linguistic predicates: They are of the form "*linguistic_variable* is *linguistic_value*". For instance:

```
R004 If fever4 is high then conclude ...
```

given a variable t (temperature), the predicate fever is high returns the value of $\mu_{high}^{fever4}(t)$.

It is also possible to define linguistic predicates of the form "*linguistic_variable* is (*value_1* or *value_2* or ...)". Consider for instance:

```
R004 If fever4 is (medium or high) then conclude ...
```

given a variable t (temperature), the predicate fever4 is (medium or high) returns the value of:

$$max(\mu_{medium}^{fever4}(t), \mu_{high}^{fever4}(t))$$

4.2 *Conclusions of rules*

The syntax of the conclusion of rules is simpler than conditions. Conclusions may appear on affirmative or on negative forms. Only many-valued propositions and set variables can be used as conclusions in rules. For instance, a many-valued proposition conclusion has the form:

[4] We decided to overload these predicates to reduce the number of total predefined predicates of the lan-

[5] Notice that $R_{\subset}(F,G)$ and $R_{\cap}(F,G)$ definitions corre- the usual notions of necessity and

```
R003 If ... then conclude fever2 is definite
```

and a set variable conclusion has the form:

```
If ... then conclude treatment=aciclovir is true
```

For a set variable the value of every element of its domain is concluded independently. For instance, for the set *treatment* with domain (*etambutol,aciclovir,ganciclovir*) the rule above gives the value [*true, true*] to the membership degree of antibiotic *aciclovir* to *treatment*.

5. DEDUCTION

The object level deductive mechanism in each module is oriented to *rule specialisation*. By rule specialisation we refer to the process of simplifying rules by means of dropping conditions from the premise as soon as they are known to be true. For instance, in classical logic, if we know that a condition A is true, the rule $r : A \wedge B \rightarrow C$ can be specialised into the rule $r' : B \rightarrow C$.

From a logical point of view, rules in Milord II are many-valued propositional implications of the form

$$(p_1 \wedge \cdots \wedge p_n \rightarrow q, W)$$

where the p_i's stand for conditions (a proposition or a ground predicate —see Section 4.1), q is the conclusion (a proposition or a set variable expression —see Section 4.2), and W is an upper interval of truth-values, that is, of the form $[a, 1]$. Moreover, factual data (either coming from the user or deduced by the system) can be always be represented by a pair (p, V), where p again stands for a proposition and V an interval of truth-values.

In this many-valued logical framework, deduction by rule specialisation is basically carried out by means of the following set of many-valued inference rules:

R1: from (p, V) infer $(\neg p, N^*(V))$
R2: from $(\neg p, V)$ infer $(p, N^*(V))$
R3: from (φ, V_1) and (φ, V_2) infer $(\varphi, V_1 \cap V_2)$
R4: from (p_i, V) and $(p_1 \wedge \cdots \wedge p_n \rightarrow q, W)$ infer $(p_1 \wedge \cdots \wedge p_{i-1} \wedge p_{i+1} \wedge \cdots \wedge p_n \rightarrow q, MP^*_T(V, W))$

where $MP^*_T(V, W)$ is the minimal interval of truth-values containing all solutions for z of the family of functional equations $I_T(a, z) = b$ varying $a \in V$ and $b \in W$.

$R1$ and $R2$ are not-introduction and not-elimination rules, $R3$ is for combination of two different truth-value intervals for the same proposition, and $R4$ is the expression of the specialisation rule[6] in the many-valued setting. All these inference rules are sound with respect to the semantics induced by the truth-value algebras introduced in section 2 (see (Puyol-Gruart *et al.*, 1998) for further details on the logical component of Milord II).

6. IMPLEMENTATION

Milord II has been implemented in Common Lisp (an interpreter) and in C (a compiler producing code for the interpreter). This software is available free for research and educational purposes. Fresh versions for Macintosh[7], PC and Unix machines can be obtained by *anonymous ftp* at ftp.iiia.csic.es/milord. You can also find previous versions and more information on Milord II at http://www.iiia.csic.es/~milord.

7. REFERENCES

Agustí, J., F. Esteva, P. Garcia, L. Godo, R. López de Màntaras and C. Sierra (1994). Local multi-valued logics in modular expert systems. *Journal of Experimental and Theoretical Artificial Intelligence* 6(3), 303–321.

Esteva, F., P. Garcia-Calves and L. Godo (1994). Enriched interval bilattices: An approach to deal with uncertainty and imprecision. *Uncertainty, Fuzzyness and Knowledge–Based Systems.*

Godo, L., R. López de Màntaras, C. Sierra and A. Verdaguer (1989). Milord: The architecture and management of linguistically expressed uncertainty. *International Journal of Intelligent Systems* 4, 471–501.

Hájek, P., L. godo and F. Esteva (1995). Fuzzy logic and probability. In: *Proceedings of the Uncertainty in Artificial Intelligence Conference, UAI-95* (P. Besnard and S. Hanks, Eds.). Morgan Kaufmann. San Francisco, USA. pp. 237–244.

López de Màntaras, Ramon (1990). *Approximate Reasoning Models.* Ellis Horwood Series in Artificial Intelligence.

Puyol-Gruart, J. and C. Sierra (1997). Milord II: a language description. *Mathware and Soft Computing* 4(3), 299–338.

Puyol-Gruart, J., L. Godo and C. Sierra (1998). Specialisation calculus and communication. *International Journal of Approximate Reasoning.* To appear.

Puyol-Gruart, Josep (1996). *MILORD II: A Language for Knowledge-Based Systems.* Vol. 1 of *Monografies del IIIA.* IIIA–CSIC.

Zadeh, L. A. (1965). Fuzzy sets. *Inf. Control* 8, 338–353.

[6] Above we are assuming $n \geq 2$. For $n = 1$, $R4$ should be read as: *from* (p, V) *and* $(p \rightarrow q, W)$ *infer* $(q, MP^*_T(V, W))$.

[7] Macintosh is a trademark of Apple Computer, Inc.