Granular Logic with Variables for Implementation of Extended Tabular Trees

Antoni Ligęza and **Grzegorz J. Nalepa**
Institute of Automatics,
AGH – University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
ligeza@agh.edu.pl gjn@agh.edu.pl

Abstract

This paper presents proposals of certain extensions to the XTT knowledge representation model, a method of tabular specification for rule-based systems. The extensions concern introduction of variables, functionally dependent attributes, constraints, fuzzy rules and time-delayed values.

Introduction

Over thirty years rule-based systems prove to constitute one of the most substantial technologies within applied Artificial Intelligence (AI) (Liebowitz 1998; Jackson 1999; Negnevitsky 2002). Rules of various forms implement the core of numerous applications, including expert systems, decision support systems, control and monitoring systems and knowledge-based systems in general (Liebowitz 1998; Jackson 1999; Hopgood 2001). Knowledge specification with rules is used both for definition of domain knowledge as well as meta-knowledge concerning inference control. With use of rules one can specify bases of purely declarative knowledge (both flat and hierarchical ones), procedural knowledge including control features, and documentation knowledge covering explanations.

The technology of rule-based systems (RBS) is based on strong logical foundations (Ben-Ari 2001; Ligęza 2006). In fact, logic constitutes the core formalism for knowledge specification and inference. Rule-based programming paradigms, including PROLOG (Covington, Nute, & Vellino 1996), are based on logical deductive inference.

Although rules constitute perhaps one of the simplest and most transparent programming paradigms, practical implementation of RBS encounters some important problems. A non-trivial system may contain less than fifty rules and simultaneously it may be difficult to handle. The main problems encountered concern *complete* specification of *non-redundant* and *consistent* set of rules. This turns out to be very tedious task requiring far-going effort.

The main issue for successful design and development of a rule-based system is to find an appropriate language for knowledge representation. A recent proposal consists in using the so-called XTT – *eXtended Tabular Trees* (Nalepa 2004; Ligęza 2006), a combination of advanced decision tables organized within tree-like strictures. In this approach syntactically similar rules operating in the same context are grouped within special attribute decision tables; this introduces structuralization of the knowledge base and enables formal verification of independent components (Ligęza 2006). However, the current approach suffers from several limitations with restricted expressive power being at the first place.

This paper presents proposals of several extensions of the XTT technology with respect to knowledge representation and several related issues. In particular, the following aspects are the main focus of this work:

- a new scheme of the XTT tables, allowing for direct specification of relational symbol (from a predefined set of six introduced symbols) is put forward as the consequence of defining Set Attributive Logic as a basic knowledge representation language,
- introduction of named variables as attribute values,
- introduction of new attributes (variables) as functions of initial attributes,
- introduction of functional constraints over these variables,
- introduction of functional (calculable) values defined by rule antecedent,
- hierarchical structure of new XTT – multi-layer structure of tables.

Moreover, the problems of using fuzzy inference rules, adaptation and learning, with use of a Wang-Mendel type algorithm (Wang & Mendel 1992) are mentioned in brief. The design and development approach can be extended over incorporating cases and learning from examples by generalization.

As the last point, the approaches to verification and validation of such rule-based systems do not solve the problem in an entire way. Verification performed after the system is designed is both costly and late. Moreover, if errors are detected, they must be corrected, and after introducing the corrections the cycle of verification must be repeated. The problem consist in the possibility of introducing new errors through correction of the old ones – there in no warranty that the verification-correction cycle is finite. In the paper an incremental verification approach is pursued.

Recapitulation of Granular Attributive Logic

Syntax of Set Attributive Language

In this section a concise recapitulation of the Granular Attributive Logic (GAL), as introduced in (Ligęza 2006) is presented in brief.

The basic concept in attributive logic is the notion of a *fact* or an *atomic formula* (*atom*, for short). This is an atomic statement of the form

$$A_i(o_j) = d_{ij}, \tag{1}$$

where A_i is an attribute, o_j is some object and d_{ij} is the (current) value of attribute A_i for object o_j. For example, the formula $\mathit{temperature}(\mathit{room}) = 17$ denotes the fact that the current temperature in the considered room equals 17 degrees Centigrade, and $\mathit{position}_X(\mathit{robot}) = 123.45$ denotes the current X-coordinate position of the robot.

For simplicity of notation, if there is only one object, or the object is known by some default assumption, or the attribute covers the definition of the object (e.g. *current_date* means in fact $\mathit{date}(\mathit{now})$), one can also omit specification of the object and write $A_i = d_i$.[1]

Let there be given the following, pairwise disjoint sets of symbols: O – a set of object symbols, $\mathbf{A}$ – a set of attribute names, $\mathbf{D}$ – a set of attribute values (the *domains*). It is further frequently assumed that the overall domain $\mathbf{D}$ is divided into several sets (disjoint or not) D_i, such that any of these sets defines the domain of some attribute. More precisely, a finite set of attributes is considered to be specified as $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$. Then also $\mathbf{D} = D_1 \cup D_2 \cup \ldots \cup D_n$, where D_i is the domain for attribute A_i, $i = 1, 2, \ldots, n$.

After (Ligęza 2006) it is assumed that an attribute A_i is a function (or partial function) of the form $A_i : O \to D_i$. Note however, that in practical applications one may need more complex attributes, taking *several values* at a time. For example, $\mathit{work\_days} = \{\mathit{Monday}, \mathit{Tuesday}, \mathit{Wednesday}, \mathit{Thursday}, \mathit{Friday}\}$. Hence, the notion of a *generalized attribute* is introduced in (Ligęza 2006). A generalized attribute A_i is a function (or partial function) of the form

$$A_i : O \to 2^{D_i}, \tag{2}$$

where 2^{D_i} is the family of all the subsets of D_i.

The generalized attributes are of limited use in some modern relational databases and object-oriented databases. After (Ligęza 2006) the language incorporating set values and generalized attributes defined as by (2) will be referred to as SAL (Set Attributive Language) while the language using simple unique-valued attributes as the one of (1) will be referred to as AAL (Atomic Attributive Language). Further on, if it does not introduce ambiguity, the qualifier *generalized* may be omitted.

In case of SAL an extended definition of atomic formulae is put forward (Ligęza & Parra 2006; Ligęza & Nalepa 2007).

[1]However, note that if there are more than one object to which the attribute can be applied, an expression of the form $A_i = d_i$ constitutes a *selector* defining a set of objects satisfying this condition; such a construction is widely used in relational databases in the SQL `SELECT` command, but is not allowed here.

Definition 1 (SAL) *Let $o \in O$ be some object, $A \in \mathbf{A}$ be a generalized attribute and let $t \subseteq D$ be a certain subset of the domain of A. Any expression of the form: $A(o) = t$, $A(o) \in t$, $A(o) \ni t$, $A(o) \subseteq t$, $A(o) \supseteq t$ and $A(o) \sim t$ are legal atomic formulae of SAL.*

For intuition, the meaning of the atomic formulae is: equal to (covers all the elements of t), is a single element of, is a set and covers some single element, is a subset of, is a superset of and have non-empty intersection, respectively. Note that the definition of atomic formulae in SAL (1) covers the one in AAL; in fact, any atomic value can be considered as a single-element set. The presented definition is extended w.r.t. the one of (Ligęza 2006) by introducing the forms: $A(o) \subseteq t$, $A(o) \supseteq t$, $A(o) \ni t$ and $A(o) \sim t$.

Recall that that the proposed relations are not independent. For example, $A_i(o) = t$ can perhaps be defined as $A_i(o) \subseteq t \wedge A_i(o) \supseteq t$; but it is much more convenient to use "=" directly. Further, as in the case of AAL, if the object is unique or known, the formulae are simplified to $A = d$, $A = t$, $A \in t$, $A \subseteq t$, $A \supseteq t$, and finally $A \sim t$, respectively.

Semantics of SAL

Semantics of SAL can be explained in a straightforward way through reduction to AAL-like formulae and with use of set algebra. Let $t = \{d_1, d_2, \ldots, d_k\}$ be a set of atomic values, $t \subseteq D$, where D is the domain of some attribute A. Let also $s \subseteq D$. For simplicity we skip the object in the following notation. Atomic formulae of SAL have the following intuitive meaning:

- $A = t$ is equivalent to $A = d_1 \wedge A = d_2 \wedge \ldots \wedge A = d_k$,
- $A \in t$ is equivalent to $A = d_1 \vee A = d_2 \vee \ldots \vee A = d_k$,
- $A \ni t$ is equivalent to $A = s$ and $s \ni t$,
- $A \subseteq t$ is equivalent to $A = s$ and $s \subseteq t$,
- $A \supseteq t$ is equivalent to $A = s$ and $s \supseteq t$,
- $A \sim t$ is equivalent to $A = s$ and $s \cap t \neq \emptyset$.

Hence, the semantics, and thus dealing with the formulae of SAL, can be reduced to AAL level with auxiliary set algebra operations.

Rule Representation with SAL within Extended Tabular Trees

The main idea behind SAL is that it allows for efficient representation of knowledge in rule-based systems. Any rule is of the form:

$$A_1(o) \propto_1 t_1 \wedge A_2(o) \propto_2 t_2 \wedge \ldots A_n(o) \propto_n t_n \longrightarrow \mathit{RHS}$$

where $\propto_i$ is one of the admissible relational symbols in SAL, i.e. $=, \in, \ni, \subseteq, \supseteq, \sim$, and RHS is the right-hand side of the rule covering conclusion and perhaps the retract and assert definitions if necessary; for details see (Ligęza 2006).

Knowledge representation with eXtended Tabular Trees (XTT) incorporates extended attributive table format. Further, similar rules are grouped within separated tables, and the whole system is split into such tables linked by arrows representing the control strategy.

Consider a set of m rules incorporating the same attributes $A_1, A_2, \ldots, A_n$. In such a case the preconditions can be grouped together and for a regular matrix. Together with the conclusion part this can be expressed as in Tab. 1

Table 1: A general scheme of an XTT table

Rule	A_1	A_2	$\ldots$	A_n	H
1	$\propto_{11}\ t_{11}$	$\propto_{12}\ t_{12}$	$\ldots$	$\propto_{1n}\ t_{1n}$	h_1
2	$\propto_{21}\ t_{21}$	$\propto_{22}\ t_{22}$	$\ldots$	$\propto_{2n}\ t_{2n}$	h_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
m	$\propto_{m1}\ t_{m1}$	$\propto_{m2}\ t_{m2}$	$\ldots$	$\propto_{mn}\ t_{mn}$	h_m

In table 1 the symbol $\propto_{ij}$ can be replaced by any of the admissible relational symbols $=$, $\in$, $\ni$, $\subseteq$, $\supseteq$, $\sim$, and RHS. In practical applications, however, the most frequent relation is $\subseteq$, i.e. the values of attributes for certain objects are restricted to belong to some specific subsets of the domain. If this is the case, the table can take simplified form presented in Tab. 2, which was originally assumed in (Ligęza 2006).

Table 2: A general scheme of an XTT table

Rule	A_1	A_2	$\ldots$	A_n	H
1	t_{11}	t_{12}	$\ldots$	t_{1n}	h_1
2	t_{21}	t_{22}	$\ldots$	t_{2n}	h_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
m	t_{m1}	t_{m2}	$\ldots$	t_{mn}	h_m

There are several advantages of grouping syntactically similar rules into tables. They include easier analysis and design of the rule-base, transparency, reuse of XTT components and verification of XTT tables with algebraic rather than logical methods (Ligęza 2006). On the other hand the expressive power remains too restricted for more complex applications. Below proposal of certain specific extensions are discussed.

Granular Logic with Variables

Representation of knowledge with XTT tables given by Tab. 1 and Tab. 2 does not allow to specify more complex formula preconditions, where, for example, direct comparison of attribute values is necessary. Below we discuss an extension of the basic formalism of SAL towards Variable Set Attributive Logic (VSAL).

Motivation

Consider a simple example of two conditional attributes, namely C denoting the cost and G denoting the gain (income). Let B denote the business result with two symbolic values, i.e. 'good' and 'bad'. We have two simple intuitive rules:

$$\begin{array}{l} rule_1: G > C \longrightarrow B =' good' \\ rule_2: G < C \longrightarrow B =' bad' \end{array} \quad (3)$$

Unfortunately, in the attributive logic such as SAL it is impossible to encode these rules and hence they cannot be represented within the pure XTT formalism.

There are at least two solutions to this problem. First, one can define a new, functionally dependent attribute $Z = G - C$ and specify the rules in the XTT form as in Tab. 3.

Table 3: A simple table with dependent attribute Z

Rule	G	C	Z	B
1	_	_	> 0	'good'
2	_	_	< 0	'bad'

Unfortunately, it is not always the case that a new, dependent attribute can be defined in a simple way. Then, one has to use functional constraints imposed on the set of attributes and a procedure for checking if these constraints are satisfied. For intuition, this can be expressed with the Table 4.

Table 4: A simple table with functional constraints $f(X, Y)$

Rule	G	C	f(X,Y)	B
1	X	Y	$X > Y$	'good'
2	X	Y	$X < Y$	'bad'

Note that new variables, namely X and Y were introduced here to denote the current values of attributes G and C, respectively. The reason for it is four-fold:

- attributes are functions mapping objects into some predefined domains; the introduced variables denote just their current values,
- some functional constraints can be defined once (using a specific set of variables names) and reused for several different sets of attributes,
- coreference constraints can be expressed with variables,
- more sophisticated rules can be expressed with use of relational symbols between attributes and variables, e.g. $G \subset X$ and then with constraints over the variables; further, constrain propagation can be defined.

Obviously, the definition of constraints must be in the form of an evaluable formula. The check for its satisfaction can be left to an external deterministic procedure.

Syntax of Variable Set Attributive Logic

The syntax of Variable Set Attributive Language (VSAL) covers two extensions with regard to SAL.

First, there is a new form of atomic formulae introduced. Let X be a variable. The new form of a legal atomic formulae is

$$A(o) = X,$$

where plays the role of placeholder for the current value of attribute A for object o.

Second, let $X_1, X_2, \ldots, X_k$ denote some variables, and let $D_1, D_2, \ldots, D_k$ be there domains, respectively. Let F

be a definition of certain relation, $F \subseteq D_1 \times D_2 \times \ldots \times D_k$. Then

$$f(X_1, X_2, \ldots, X_k) \tag{4}$$

is a characteristic predicate of a certain relation F. In fact, $f(X_1, X_2, \ldots, X_k)$ can be evaluated to *true* or *false* for any specific values of the variables. The expression given by (4) is a legal atomic formula of VSAL.

Semantics of VSAL

The semantics of VSAL is a straightforward extension of the one of SAL. An atomic formula of the form $A(o) = X$ cannot be assigned a truth value until the value of X is specified. Hence one needs a variable assignment σ defining the current value of X (it can be an atomic or set value).

In practice, the value of X is defined by the current value of attribute A. The variables play a role of the so-called *coreference constraints* – the current values of attributes are passed over to the functional constraints. Then $f(X_1, X_2, \ldots, X_k)$ is evaluated to *true* if and only if the tuple of values of respective attributes $A_1, A_2, \ldots, A_k$ satisfies $(d_1, d_2, \ldots, d_n) \in F$.

Use of Variables in Rule Conclusions

The other reason for introducing variables is that in numerous systems it is necessary to transfer the values of input attributes from the precondition section to the conclusions.

Consider another problem when the task is to define that if the gain is up to 1000, one has to pay 20% tax, while if it is above 1000, one has to pay 200 and 30% of the additional part. Again this cannot be expressed within simple XTT formalism. For intuition, we have to define rules as presented in Fig. 1.

Here the solution requires the possibility to specify the values of output attributes being functionally dependent on the input attribute values.

Tables with Variables, Dependent Attributes, Functional Constraints and Functional Output

The extended form of XTT tables encoding rules allows for use of variables as attribute values. The set of initial, independent attributes, can also be extended towards the use of new, complex attributes, functionally dependent on the initial ones. Their values can be calculated since the values of initial attributes are established.

Extended tables can also contain functional constraints specification. Such constraints constitute a powerful tool for expressing required relationships among attribute values.

For efficient decision making and in control applications it may be necessary to calculate the output on the base of current input values. This is now possible thanks to the functional output definition, applicable both for the *retract*, *assert* and decisional part of the rules.

A generic scheme of extended rules is presented in Fig. 2; for simplicity the number of the rule, the context and the control part are committed.

Attributive Logic with Time Delays

Most of the rule-based systems are in fact *reactive* ones. They operate according to a very simple principle: the *current* state of the input variables is analyzed and the rules having satisfied preconditions are fired. However, in certain applications, e.g. in the control or signal analysis domain, the preconditions of the rules must incorporate not only current values of input signals, but the *past* values as well.

Motivation

Consider a simple example of signal analysis. The input consists of a sequence of values $X(t_0 + \Delta t \times i) = X_i$, where t_0 is some initial instant of time and Δt is the basic time interval of signal measurement. For example, determining the current sign of the first derivative can be done with the following three rules represented within Table 5.

Table 5: Rules for determining the sign of the first derivative

X_{i-1}	X_i	$X_i - X_{i-1}$	Sign
_	_	> 0	+
_	_	< 0	-
_	_	$= 0$	0

In general, applications which detect, analyze and use information about changes of the input signals, such as first, second and higher derivatives or accumulated values, such as moving average for certain period of time, must access and use past, historical values of the signals. A classical example of such task is the prediction and control of dynamic systems.

Note that access to variable history is not limited to numerical signals. The rules in Tab. 6 are used by car drivers to predict the next value of the traffic light signal:

Table 6: Rules for traffic lights prediction

Light-before	Light-now	Light-next
red	yellow	green
green	yellow	red
yellow	red	yellow
yellow	green	yellow

Temporal Formalism

The simplest and perhaps most useful from practical point of view temporal formalism is one used in signal analysis and control theory. The time domain is divided into small and equal temporal intervals. The signals are measured only at indexed time instants. The intervals should be small enough to capture the expected changes and the dynamic behavior of the system.

Consider a specific attribute A_i. The value of this attribute at time instant k will be denoted as a pair (A_i, k), or simply as $A_i(k)$. Now, if the rule based system uses the current value as well as m past values, the table representing such

Rule	G	C	f(X,Y)	g(X,Y)	T
1	X	Y	$X > Y$	$X - Y \leq 1000$	$(X - Y) * 0.2$
2	X	Y	$X > Y$	$X - Y > 1000$	$200 + (X - Y - 1000) * 0.3$

Figure 1: A simple table with functional output

Figure 2: The basic form of an XAT

Prec	Dependent	Constraints	Retract	Assert	Decision
$A_1 \dots A_n$	Z	F	B	C	H
$X_1 \dots X_n$	$z(X_1, \dots, X_n)$	$f(X_1, \dots, X_n)$	$r(X_1, \dots, X_n)$	$a(X_1, \dots, X_n)$	$h(X_1, \dots, X_n)$

rules should be extended and can have the following form represented with Table 7.

Table 7: Rules with past values of attribute A_i

$A_i(k-m)$	$A_i(k-m+1)$	...	$A_i(k)$	D
$d_i(k-m)$	$d_i(k-m+1)$	...	$d_i(k)$	d

where $d_i(j)$ is the value of attribute A_i at time instant j.

A more general solution might require complex temporal knowledge representation formalism. In general, there are numerous temporal logics for specific purposes (Ben-Ari 2001). Practical formalisms referring to exact time fall into two basic categories: ones based on the concept of *interval* of time, and ones based on the concept of *point* of time. They may also refer to *relative* or *absolute* time. Up-to-date, however, temporal formalism have not found way for practical applications in rule-based systems (Liebowitz 1998; Cheng 2002).

Fuzzy Rules and Fuzzy Inference

In practical applications it is often the case that matching rule preconditions against the current values of state variables must be performed with some tolerance. In other words, the boundaries such as threshold values are to certain degree imprecise. In such a case a useful approach accepted by practitioners consists in defining *fuzzy rules* with preconditions specification based on *fuzzy sets*. The material below is based on the classical approach to fuzzy sets and fuzzy rule-based systems (Ibrahim 2004; Negnevitsky 2002).

The Case of Discrete Variables

Consider a discrete set $Z = \{z_1, z_2, \dots, z_k\}$. Let μ be a function of the form:

$$\mu: Z \rightarrow [0, 1].$$

The pair $\mathcal{Z} = (Z, \mu)$ is called a discrete fuzzy set. In case of finite sets it can be also written as a set of pairs (z_i/μ_i), where $\mu_i = \mu(z_i)$, $i = 1, 2, \dots, k$. Hence $\mathcal{Z} = \{z_1/\mu_1, z_2/\mu_2, \dots, z_k/\mu_k\}$. Note that the crisp set Z can be a set of numbers, but it can be a set of purely symbolic values as well.

Normally, a discrete fuzzy set contains several pairs of the form (z_i/μ_i). If $\mathcal{Z} = \{z/\mu\}$ then $\mathcal{Z}$ is called a *singleton*.

The Case of Continuous Variables

In case of continuous values the fuzzy membership function can be defined to be any function with the range falling into the interval $[0, 1]$. In practice, the most typical functions used are triangular and trapezoidal ones as well as based on left or right half of the trapezoid.

Below a single, general function defining the fuzzy degree is introduced. The function covers several well-known cases of particular membership functions.

Definition 2 (The π function (fuzzy)) *Let Z be a convex interval of numeric values, e.g. $Z \subset \mathbb{R}$, and let α, β, γ and δ be numbers belonging to S and such that $\alpha \leq \beta \leq \gamma \leq \delta$. We define the $\pi^{\langle \alpha,\beta,\gamma,\delta \rangle}$ function as follows:*

$$\begin{array}{rcll}
\pi^{\langle \alpha,\beta,\gamma,\delta \rangle}(\alpha) & = & 0 & \text{for} \quad \alpha < \beta \\
\pi^{\langle \alpha,\beta,\gamma,\delta \rangle}(\alpha) & = & 1 & \text{for} \quad \alpha = \beta \\
\pi^{\langle \alpha,\beta,\gamma,\delta \rangle}(\beta) & = & 1 & \\
\pi^{\langle \alpha,\beta,\gamma,\delta \rangle}(\gamma) & = & 1 & \\
\pi^{\langle \alpha,\beta,\gamma,\delta \rangle}(\delta) & = & 1 & \text{for} \quad \delta = \gamma \\
\pi^{\langle \alpha,\beta,\gamma,\delta \rangle}(\delta) & = & 0 & \text{for} \quad \delta > \gamma
\end{array}$$

and is constructed by piecewise linear interpolation among these points. The pair $\mathcal{Z} = (Z, \pi)$ is a fuzzy set.

We distinguish the following special cases of function π:

1. *$\Delta^{\alpha,\beta,\delta} = \pi^{\langle \beta,\beta \rangle}$ obtained for $\alpha < \beta = \gamma < \delta$; this is the so-called* triangle function,
2. *$\Gamma = \pi^{\langle \alpha,\beta,\gamma \rangle}$ obtained for $\alpha < \beta < \gamma = \delta$; this function is a fuzzy step-wise function,*
3. *$Z = \pi^{\langle \beta,\gamma,\delta \rangle}$ obtained for $\alpha = \beta < \gamma < \delta$.*

In majority of practical applications the trapezoidal function together with the functions based on it, provides a satisfactory solution for defining fuzzy membership functions. Note that its shape is defined by four distinct parameters, the shape of triangular function by three, and the shape of the other two function just by two parameters.

Fuzzy Rule Preconditions

Definition of rule precondition in the fuzzy case is analog to the crisp case. In the crisp case the basic atomic condition is of the form: $A \in t$, and the check results in a single logical value, i.e. *true* if the current value of attribute A belongs to set t and *false* otherwise. In the case of fuzzy preconditions one has to check if $A \in \mathcal{Z}$, and the result is the fuzzy membership coefficient belonging to interval $[0,1]$. It is normally assumed that the check produces negative answer if $\mu(A) = 0$ and positive one if $\mu(A) > 0$.

Now consider a rule having j atomic preconditions incorporating fuzzy sets. Let μ_i denote the fuzzy coefficient obtained for the $i - th$ atomic condition. The total coefficient is obtained as

$$\mu = \mu_1 \circ \mu_2 \circ \ldots \circ \mu_j,$$

where $\circ$ is any t-norm operator (such as min or times). A rule is fired iff and only if $\mu > 0$. In typical fuzzy rule-based systems it is possible to fire several rules at the same time. If more than one rule is fired, the results of the rules are combined to form a single output.

Fuzzy Rule Output

In case a single rule is fired, the current value of μ can be applied in the following ways:

- instead of a single crisp-case output value d a singleton (d, μ) is produced; such a pair can be interpreted as 'd with degree μ',
- μ can be used to shape the output fuzzy value (limit) as in the case of Mamdani inference rules,
- μ can be used a kind of *certainty degree* in case of logical output values.

When several rules are fired, the value of μ is used to shape the fuzzy output functions (the Mamdani system) or as weighting coefficients (the Takagi-Sugeno system).

Learning Fuzzy Rules

A fuzzy rule-based system can become a learning one. This is especially important when a number of training cases are available, but no expert knowledge covering them is available. In such a case a learning or adaptation algorithm can be used to build a set of rules.

In the case of fuzzy rule-based systems practical results can be obtained with relatively simple approach of Wang-Mendel type algorithm (Wang & Mendel 1992).

Verification of Fuzzy Rules

Verification of fuzzy sets of rules is different than in the case of AAL or SAL. For example, rule preconditions are overlapping by intension and one can measure a degree of that. Completeness and conflict also become *fuzzy* phenomena and degrees of them are calculated instead of formal verification. A note on the approach is presented in (del Acebo *et al.* 1998).

Conclusions

In this paper some fundamental extension to the XTT design method for RBS were presented. They aim at enhancing the expressiveness of the XTT language, with use temporal and fuzzy rules. The paper gives an informal description of these proposed extensions. It is hoped, that these experimental features could widen the application area of the method.

Acknowledgments The research presented in this paper is supported from a MNiSW Research Project *HeKatE* No.: N516 024 32/2878, Agreement No.: 2878/T02/2007/32.

References

Ben-Ari, M. 2001. *Mathematical Logic for Computer Science*. London: Springer-Verlag.

Cheng, A. M. K. 2002. *Real-Time Systems. Scheduling, Analysis and Verification*. Hoboken, New Yersey: John Wiley & Sons, Inc.

Covington, M. A.; Nute, D.; and Vellino, A. 1996. *Prolog programming in depth*. Prentice-Hall.

del Acebo, E.; Oller, A.; de la Rosa, J. L.; and Ligęza, A. 1998. Static criteria for fuzzy systems quality evaluation. In del Pobil, A. P.; Mira, J.; and Ali, M., eds., *Tasks and Methods in Applied Artificial Intelligence*, volume II of *Lecture Notes in Artificial Intelligence 1416*. Berlin, Heidelberg: Springer-Verlag. 877–887.

Hopgood, A. A. 2001. *Intelligent Systems for Engineers and Scientists*. Boca Raton London New York Washington, D.C.: CRC Press, 2nd edition.

Ibrahim, A. M. 2004. *Fuzzy Logic for Embedded Systems Applications*. Embedded Technology. Burlington, MA: Elsevier Science.

Jackson, P. 1999. *Introduction to Expert Systems*. Addison–Wesley, 3rd edition. ISBN 0-201-87686-8.

Liebowitz, J., ed. 1998. *The Handbook of Applied Expert Systems*. Boca Raton: CRC Press.

Ligęza, A., and Nalepa, G. J. 2007. Knowledge representation with granular attributive logic for XTT-based expert systems. In Wilson, D. C.; Sutcliffe, G. C. J.; and FLAIRS., eds., *FLAIRS-20 : Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference : Key West, Florida, May 7-9, 2007*, 530–535. Menlo Park, California: Florida Artificial Intelligence Research Society.

Ligęza, A., and Parra, P. F. 2006. A granular attribute logic for rule-based systems management within extended tabular trees. In Trappl, R., ed., *Cybernetic and Systems*, volume 2, 761–766. Austrian Society for Cybernetic Studies.

Ligęza, A. 2006. *Logical Foundations for Rule-Based Systems*. Berlin, Heidelberg: Springer-Verlag.

Nalepa, G. J. 2004. *Meta-Level Approach to Integrated Process of Design and Implementation of Rule-Based Systems*. Ph.D. Dissertation, AGH University of Science and Technology, AGH Institute of Automatics, Cracow, Poland.

Negnevitsky, M. 2002. *Artificial Intelligence. A Guide to Intelligent Systems*. Harlow, England; London; New York: Addison-Wesley. ISBN 0-201-71159-1.

Wang, L.-X., and Mendel, J. M. 1992. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-22(6):1414–1427.