XTT+ Rule Design Using the ALSV(FD)

Grzegorz J. Nalepa and **Antoni Ligęza** [1]

Abstract. This paper presents advances in Set Attributive Logic and its application to develop tabular rule-based systems within the XTT framework. The primary goal is to extend the expressive power of simple attributive languages so that it becomes satisfactory for complex applications, including the business rules support. A formal framework of extended Attributive Logic with Set Values over Finite Domains (ALSV(FD)) is presented and specific inference rules are provided with their corresponding prototype in PROLOG.

1 INTRODUCTION

Rule-based systems (RBS) are one of the most efficient paradigms for knowledge representation and automated inference. This is an intuitive and well-established language [5]. However, when it comes to the engineering practice, as well as its scientific aspect, the formal approach to the rule language specification has to be considered. In fact, there are number of specific rule languages based on different formal calculi, from simple propositional logic, through subsets of predicate calculus, to specific higher-order logics [9].

This paper presents advances in *Set Attributive Logic* and its application to develop tabular rule-based systems within the XTT framework. The primary goal is to extend the expressive power of simple attributive languages so that it becomes satisfactory for complex monitoring, control, decision support and business rules applications. A formal framework of extended Set Attributive Logic is presented and specific inference rules are provided. The practical representation and inference issues both at the logical and implementation level are tackled.

2 HEKATE RULE LANGUAGE

In the HEKATE project (`hekate.ia.agh.edu.pl`) an extended rule language is proposed. It is based on the XTT language described in [11]. The version used in the project is currently called XTT+.

The XTT+ rule language is based on the classic concepts of rule languages for rule-based systems [8], with certain important extensions and features, such as:

- strong formal foundation based on attributive logic,
- explicit rulebase structurization,
- extended rule semantics.

In this paper the XTT+ language will be simply referred to as XTT.

In XTT there is a strong assumption, that the rule base is explicitly structured. The rules with same sets of attributes are grouped within decision tables. On the rule level explicit inference control is allowed. In this way, a set of tables is interconnected using links, corresponding to inference control. This makes up a decision-tree like structure, with tables in the tree nodes. In a general case, the XTT is a directed graph, with cycles optionally allowed.

In RBS, a rule has a general format:

```
IF condition THEN decision
```

This format can be used in both forward and backward chaining systems. However, here we focus on the production rule systems, based on the forward chaining paradigm. The power of a rule language stems from the syntax and semantics of the conditional and decision expressions. Number of systems implicitly assume, that this rule format can be extended to the *conjunctive normal form (CNF)*, that is:

```
IF cond1 AND cond2 AND ... AND condN
THEN decision
```

which in fact corresponds to a *Horn* clause φ [1, 9], that is:

$$\varphi = \neg p_1 \vee \neg p_2 \vee \ldots \neg p_k \vee q,$$

Such a clause can be represented as an implication of the form:

$$\varphi = p_1 \wedge p_2 \wedge \ldots \wedge p_k \Rightarrow q.$$

which can be regarded as a rule in the above format, where ps correspond to conditions and q corresponds to the decision. In fact the PROLOG language uses a subset of predicate calculus, restricted to *Horn* clauses [3].

The decision expression can also be a compound one in the CNF. Now the question is what are the conditional and decision expressions. In number of systems these correspond to expressions in the propositional calculus, which makes the semantics somehow limited. Some systems try to use some subsets of predicate logic, which gives much more flexibility, but may complicate a RBS design and the inference process. This is the case of the PROLOG language [2]. In XTT these expressions are in the the *attributive logic* [9] described in more detail in Sect. 4. This gives much more power than the propositional logic, but does not introduce problems of the predicate logic-based inference. In XTT an extended rule semantics is used. These extensions were introduced in [13], and refined in [12].

Let us now move to attributive logic that provides a formal foundation for the rule language.

3 A MOTIVATIONAL EXAMPLE

Consider a simple piece of knowledge expressed with natural language as follows.

> The regular class hours are from 8:00 to 18:00. If all the teaching hours are located within regular class hours then the salary is regular. If the teaching hours goes beyond the regular class hours then the salary is special.

[1] Institute of Automatics, AGH – University of Science and Technology, Al. Mickiewicza 30, 30-059 Kraków, Poland email: `gjn@agh.edu.pl` `ligeza@agh.edu.pl`

The problem is to formalize these two rules with attributive logic. Let *RCH* stays for regular class hours, and *TH* for teaching hours. We can define a fact like:

$$RCH = \{8, 9, 10, 11, 12, 13, 14, 15, 16, 17\},$$

and

$$TH = \{10, 11, 12, 16, 19, 20\}$$

to specify a case of teaching hours. Note that teaching hours may form any subset of $\{0, 1, 2, 3, \ldots, 23\}$ (not necessarily a convex interval).

Now, to express the rules we need an extended attributive logic employing set values of attributes and some powerful relational symbols. For example, we can have:

$$R1: \ TH \subseteq RCH \longrightarrow Salary =' regular'$$

and

$$R2: \ TH \sim NRCH \longrightarrow Salary =' special'$$

where

$$NRCH = \{0, 1, 2, 3, 4, 5, 6, 7, 18, 19, 20, 21, 22, 23\}$$

is a specifications of non-regular class hours, and sim means a non-empty intersection. Note that an attempt to specify the rules with attribute logic based on atomic values (even if relational symbols such as $<$, $\leq$, $>$ and $\geq$ are allowed) would lead to a very long and at least clumsy set of hardly readable rules.

4 ATTRIBUTIVE LOGIC

Attributive logics constitute a simple but widely-used tool for knowledge specification and inference. In fact in a large variety of applications in various areas of Artificial Intelligence (AI) [14] and Knowledge Engineering (KE) attributive languages constitute the core knowledge representation formalism. The most typical areas of applications include rule-based systems [8, 9], expert systems (ones based on rule formalism) [6, 15] and advanced database and data warehouse systems with knowledge discovery applications [7] and contemporary business rules and business intelligence components (e.g. Jess, Drools).

However, it is symptomatic that although Propositional Logic and Predicate Logic (in the form of First-Order Predicate Calculus) have well-elaborated syntax and semantics, presented in details in numerous books covering logic for knowledge engineering [4, 6, 15], logic for computer science or Artificial Intelligence [1, 8], the discussion of syntax and semantics of attribute-based logic is omitted in such positions.

In a recent book [9] the discussion of attributive logic is much more thorough. The added value consist in allowing that attributes can take *set values* and providing formal syntax of the *Set Attributive Logic* (SAL) with respect to its syntax, semantics and selected inference rules.

The very basic idea is that attributes can take *atomic* or *set* values. After [9] it is assumed that an attribute A_i is a function (or partial function) of the form $A_i{:}\, O \rightarrow D_i$. A generalized attribute A_i is a function (or partial function) of the form $A_i{:}\, O \rightarrow 2^{D_i}$, where 2^{D_i} is the family of all the subsets of D_i. The atomic formulae of SAL can have the following three forms: $A_i = d$, $A_i = t$ or $A_i \in t$,

where $d \in D$ is an atomic value from the domain D of the attribute and $t = \{d_1, d_2, \ldots, t_k\}$, $t \subseteq D$ is a set of such values. The semantics of $A_i = d$ is straightforward – the attribute takes a single value. The semantics of $A_i = t$ is that the attribute takes *all* the values of t (the so-called *internal conjunction*) while the semantics of $A_i \in t$ is that it takes *some* of the values of t (the so-called *internal disjunction*).

The SAL as introduced in [9] seems to be an important step towards the study and extension of attributive logics towards practical applications. On the other hand it still suffers from lack of expressive power and the provided semantics of the atomic formulae is poor.

In this paper an improved and extended version of SAL is presented in brief. For simplicity no object notation is introduced. The formalism is oriented toward Finite Domains (FD) and its expressive power is increased through introduction of new relational symbols. The practical representation and inference issues both at the logical level and implementation level are tackled. The main extension consists of a proposal of extended set of relational symbols enabling definitions of atomic formulae. The values of attributes can take singular and set values over Finite Domains (FD).

4.1 ALSV(FD)

An extension of SAL was proposed in [10]. Both the syntax and semantics were extended and clarified. Here some further details to support set values of attributes over finite domains are discussed.

The basic element of the language of *Attribute Logic with Set Values over Finite Domains* (ALSV(FD) for short) are attribute names and attribute values. Let us consider:

$\mathbf{A}$ – a finite set of attribute names,
$\mathbf{D}$ – a set of possible attribute values (the *domains*).

Let $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$ be all the attributes such that their values define the state of the system under consideration. It is assumed that the overall domain $\mathbf{D}$ is divided into n sets (disjoint or not), $\mathbf{D} = D_1 \cup D_2 \cup \ldots \cup D_n$, where D_i is the domain related to attribute A_i, $i = 1, 2, \ldots, n$. Any domain D_i is assumed to be a finite (discrete) set.

As we consider dynamic systems, the values of attributes can change over time (or state of the system). We consider both *simple* attributes of the form $A_i{:}\, T \rightarrow D_i$ (i.e. taking a single value at any instant of time) and *generalized* ones of the form $A_i{:}\, T \rightarrow 2^{D_i}$ (i.e. taking a set of values at a time); here T denotes the time domain of discourse.

Let A_i be an attribute of $\mathbf{A}$ and D_i the sub-domain related to it. Let V_i denote an arbitrary subset of D_i and let $d \in D_i$ be a single element of the domain. The atomic formulae of ALSV(FD) are defined as follows.

Definition 1 *The legal atomic formulae of ALSV for simple attributes are:*

$$A_i = d, \tag{1}$$

$$A_i \neq d, \tag{2}$$

$$A_i \in V_i, \tag{3}$$

$$A_i \notin V_i. \tag{4}$$

Definition 2 *The legal atomic formulae of ALSV for generalized attributes are:*

$$A_i = V_i, \tag{5}$$

$$A_i \neq V_i, \tag{6}$$

$$A_i \subseteq V_i, \tag{7}$$

$$A_i \supseteq V_i \tag{8}$$

$$A \sim V, \tag{9}$$

$$A_i \not\sim V_i. \tag{10}$$

In case V_i is an empty set (the attribute takes in fact no value) we shall write $A_i = \{\}$. In case the value of A_i is unspecified we shall write $A_i =$ `NULL` (a database convention). If we do not care about the current value of the attribute we shall write $A =$ _ (a PROLOG convention).

The semantics of the atomic formulae as above is straightforward and intuitive. In case of the first three possibilities given by (1), (2), (3) and (4) we consider A_i to be a simple attribute taking exactly one value. In case of (1) the value is precisely defined, while in case of (3) any of the values $d \in V_i$ satisfies the formula. In other words, $A_i \in V_i$ is equivalent to $(A_i = d_1) \otimes (A_i = d_2) \otimes \ldots \otimes (A_i = d_k)$, where $V_i = \{d_1, d_2, \ldots, d_k\}$ and $\otimes$ stays for exclusive-or. Here (2) is a shorthand for $A_i \in D_i \setminus \{d\}$. Similarly, (4) is a shorthand for $A_i \in D_i \setminus V_i$.

The semantics of (5), (2) (7),(8), (9), and (10) is that A_i is a generalized attribute taking a set of values equal to V_i (and nothing more), different from V_i (at at least one element), being a subset of V_i, being a superset of V_i, having a non-empty intersection with V_i or disjoint to V_i, respectively.

More complex formulae can be constructed with *conjunction* ($\wedge$) and *disjunction* ($\vee$); both the symbols have classical meaning and interpretation.

There is no explicit use of negation. The proposed set of relations is selected for convenience and as such is not completely independent. For example, $A_i = V_i$ can perhaps be defined as $A_i \subseteq V_i \wedge A_i \supseteq V_i$; but it is much more concise and convenient to use "=" directly. Various notational conventions extending the basic notation can be used. For example, in case of domains being ordered sets symbols such as $>$, $>=$, $<$, $=<$ can be used.

4.2 BASIC INFERENCE RULES FOR ALSV(FD)

Since the presented language is an extension of the SAL (Set Attributive Logic) presented in [9], its simple and intuitive semantics is consistent with SAL and clears up some points of it. For example, the *upward* and *downward consistency rules* do hold and can be formulated in a more elegant way. Let V and W be two sets of values such that $V \subseteq W$. We have the following straightforward inference rules for atomic formulae:

$$\frac{A \supseteq W}{A \supseteq V} \tag{11}$$

i.e. if an attribute takes all the values of a certain set it must take all the values of any subset of it (downward consistency). Similarly

$$\frac{A \subseteq V}{A \subseteq W} \tag{12}$$

i.e. if the values of an attribute takes values located within a certain set they must also belong to any superset of it (upward consistency). These rules seem a bit trivial, but they must be implemented for enabling inference, e.g they are used in the rule precondition checking.

The summary of the inference rules for atomic formulae with simple attributes (where an atomic formula is the logical consequence of another atomic formula) is presented in Table. 1. In Table 1 and Table 2 the conditions are *satisfactory* ones. However, it is important to note that in case of the first rows of the tables (the cases of $A = d_i$ and $A = V$, respectively) all the conditions are also *necessary* ones. The interpretation of the tables is straightforward: if an atomic formula in the leftmost column in some row i is true, then the atomic formula in the topmost row in some column j is also true, provided that the relation indicated on intersection of row i and column j is true. The rules of Table 1 and Table 2 can be used for checking if preconditions of a formula hold or verifying subsumption among rules.

Table 1. Inference rules for atomic formulae for simple attributes

$\models$	$A = d_j$	$A \neq d_j$	$A \in V_j$	$A \notin V_j$
$A = d_i$	$d_i = d_j$	$d_i \neq d_j$	$d_i \in V_j$	$d_i \notin V_j$
$A \neq d_i$	_	$d_i = d_j$	$V_j = D \setminus \{d_i\}$	$V_j = \{d_i\}$
$A \in V_i$	$V_i = \{d_j\}$	$d_j \notin V_i$	$V_i \subseteq V_j$	$V_i \cap V_j = \emptyset$
$A \notin V_i$	$D \setminus V_i = \{d_j\}$	$V_i = \{d_j\}$	$V_j = D \setminus V_i$	$V_j \subseteq V_i$

4.3 RULES IN ALSV(FD)

ALSV(FD) has been introduced with practical applications for rule languages in mind. In fact, the primary aim of the presented language is to extend the notational possibilities and expressive power of the XTT-based tabular rule-based systems [9]. An important extension consist in allowing for explicit specification of one of the symbols $=, \neq, \in, \notin, \subseteq, \supseteq, sim$ and $\not\sim$ with an argument in the table.

Consider a set of n attributes $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$. Any rule is assumed to be of the form:

$$(A_1 \propto_1 V_1) \wedge (A_2 \propto_2 V_2) \wedge \ldots (A_n \propto_n V_n) \longrightarrow RHS$$

where $\propto_i$ is one of the admissible relational symbols in ALSV(FD), and RHS is the right-hand side of the rule covering conclusion and perhaps the retract and assert definitions if necessary; for details see [9].

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 form a regular matrix. Together with the conclusion part this can be expressed as in Tab. 3

Table 3. 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 3 the symbol $\propto_{ij} \in \{=, \neq, \in, \notin\}$ for simple attributes and $\propto_{ij} \in \{=, \neq, \subseteq, \supseteq, \sim, \not\sim\}$ for the generalized ones. In practical

Table 2. Inference rules for atomic formulae for generalized attributes

$\models$	$A = W$	$A \neq W$	$A \subseteq W$	$A \supseteq W$	$A \sim W$	$A \not\sim W$
$A = V$	$V = W$	$V \neq W$	$V \subseteq W$	$V \supseteq W$	$V \cap W \neq \emptyset$	$V \cap W = \emptyset$
$A \neq V$	_	$V = W$	$W = D$	_	$W = D$	_
$A \subseteq V$	_	$V \subset W$	$V \subseteq W$	_	$W = D$	$V \cap W = \emptyset$
$A \supseteq V$	_	$W \subset V$	$W = D$	$V \supseteq W$	$V \cap W \neq \emptyset$	_
$A \sim V$	_	$V \cap W = \emptyset$	$W = D$	_	$V = W$	_
$A \not\sim V$	_	$V \cap W \neq \emptyset$	$W = D$	_	$W = D$	$V = W$

applications, however, the most frequent relation are $=$, $\in$, and $\subseteq$, i.e. the current values of attributes are *restricted* to belong to some specific subsets of the domain. If this is the case, the relation symbol can be omitted (i.e. it constitutes the default relation which can be identified by type of the attribute and the value).

The current values of all the attributes are specified with the contents of the knowledge-base (including current sensor readings, measurements, inputs examination, etc.). From logical point of view it is a formula of the form:

$$(A_1 = S_1) \wedge (A_2 = S_2) \wedge \ldots \wedge (A_n = S_n), \qquad (13)$$

where $S_i = d_i$ ($d_i \in D_i$) for simple attributes and $S_i = V_i$, ($V_i \subseteq D_i$) for complex.

Having a table with defined rules the execution mechanism searches for ones with satisfied preconditions. The satisfaction of preconditions is verified in an algebraic mode, using the dependencies specified in the first row of Table 1 for simple attributes and the first row of Table 2 for the complex ones.

The rules having all the preconditions satisfied can be fired. In general, rules can be fired in parallel (at least in theory) or sequentially. For the following analysis we assume the classical, sequential model, i.e. the rules are examined in turn in the top-down order and fired if the preconditions are satisfied. Various mechanisms can be used to provide a finer inference control mechanism [9].

4.4 ATTRIBUTE DOMAINS

It is assumed that for each XTT attribute a *type* has to be stated. A type is named and it specifies: the base type and the domain. In the design of the XTT+ attributive language the set of base types was introduced in a way that simplifies the low-level interpreter integration with Prolog, Java, and RDBMS.

An example definition could be as follows:

- suppose we have a natural language specification: "some temperature",
- create attribute type,
  - pick a attribute type name, e.g. "Temperature"
  - decide what base type to use, in this case it could be a float,
  - define the domain by specifying constraints, e.g. -100, 100 depending on the conditions,
  - decide whether the domain is ordered – in case of symbolic base type, in this case numbers are ordered,
- create new attribute, with given,
  - attribute type, in this case of "Temperature",
  - name, e.g. sensor_temperature,
  - decide whether the attribute can take only single values, or also multiple values.

Generalized attributes are unordered or ordered sets (lists). This means attributes are in fact *multi-valued*. Some applications for this features are e.g.: a set of languages a person speaks, or storing subsequent values (changing in time).

4.5 RULE FIRING

The XTT+ rule firing process is coherent with the regular RBS semantics. It involves: condition checking and decision execution.

The condition checking can be described as a pattern matching process, where the condition evaluates true or false. The condition is an expression in the CNF build of expressions in the ALSV(FD).

The decision execution is where actions are possible. In a general case, the XTT+ rule decision involves: attribute value change context switching through inference control links event triggering. In XTT it is assumed, that the *whole* system state is described by the means of attributes.

5 PROTOTYPE IMPLEMENTATION EXAMPLE

In the prototype implementation of the knowledge base, rules and the interpreter are developed in PROLOG. A meta-programming approach is followed. This allows for encoding virtually any structured information. Note that in such a case the built-in PROLOG inference facilities cannot be used directly, there is a need for a meta-interpreter (however, this gives more flexibility in terms of rule processing).

Example domains and attributes specification in PROLOG follows:

```
domain(d7,[1,2,3,4,5,6,7]).
attribute(aDN,atomic,d7).
attribute(sDN,set,d7).
```

The atomic formulae (facts) are represented as terms of the type `fact/4` with four arguments; here are some examples:

```
%%% fact(<attribute-type>,<attribute-name>,
%   <relation>,<attribute-domain>)
fact(atomic,aDN,eq,7).
fact(atomic,aDD,in,[monday,wednesday,friday]).
fact(set,sDD,sim,[monday,wednesday,friday]).
fact(set,sSE,subseteq,[spring,summer,autumn]).
```

Facts are used mostly in rule preconditions. The meaning of the above facts is as follows: `f1: sDN=7`, `f2: aDD∈[monday,wednesday,friday]`, `f3: sDD∼[monday, wednesday,friday]`, and `f4: sSE⊆[spring,summer,autumn]`. PROLOG list are used to represent set values.

The state of the system is represented by all the facts true in that state. Recall that the form $A = d$ and $A = V$ are allowed for state specification.

```
%%% state(<state-identifier>,
%   <attribute>,<value>,<type>).
state(s17,aDD,atomic,friday).
state(s17,aSE,atomic,spring).
state(s17,sDN,set,[1,3,5,7]).
```

Note that using set values in state specification increases drastically the expressive power. This is a bit similar to the Cartesian Product: in state `s17` the attribute `sDN` takes all the values from `[1,3,5,7]`.

Inference, i.e. checking logical consequence defined by first rows of Table 1 and Table 2 is performed with the `valid/s` predicate defined as follows:

```
valid(f(atomic,A,eq,Value),State):-
   state(State,A,atomic,StateValue),
   Value ==  StateValue,!.
valid(f(atomic,A,neq,Value),State):-
   state(State,A,atomic,StateValue),
   Value =\= StateValue,!.
valid(f(atomic,A,in,SetValue),State):-
   state(State,A,atomic,StateValue),
   member(StateValue,SetValue),!.
valid(f(atomic,A,notin,SetValue),State):-
   state(State,A,atomic,StateValue),
   \+member(StateValue,SetValue),!.
valid(f(set,A,eq,SetValue),State):-
   state(State,A,set,StateValue),
   eqset(SetValue,StateValue),!.
valid(f(set,A,neq,SetValue),State):-
   state(State,A,set,StateValue),
   neqset(SetValue,StateValue),!.
valid(f(set,A,subseteq,SetValue),State):-
   state(State,A,set,StateValue),
   subset(SetValue,StateValue),!.
valid(f(set,A,supseteq,SetValue),State):-
   state(State,A,set,StateValue),
   subset(StateValue,SetValue),!.
valid(f(set,A,sim,SetValue),State):-
   state(State,A,set,StateValue),
   intersect(SetValue,StateValue,[_|_]),!.
valid(f(set,A,notsim,SetValue),State):-
   state(State,A,set,StateValue),
   intersect(SetValue,StateValue,[]),!.
```

The excerpt of the implementation code presented above includes only symbolic domains. The definitions for the remaining domains are similar to the ones presented here. Currently the use of CLP (*Constraint Logic Programming*) PROLOG extensions are being investigated.

6 CONCLUDING REMARKS

Providing an expressive yet formally described rule language is of a high importance for practical rule design and implementation. This paper presents extensions of Set Attributive Logic as presented in [9]. In the proposed logic both atomic and set values are allowed and various relational symbols are used to form atomic formulae. The proposed language provides a concise and elegant tool of significantly higher expressive power than in case of classical attribute logic. It can be applied for design, implementation and verification of rule-based systems.

In the paper new inference rules specific for the introduced logic are presented and examined. New inference possibilities constitute a challenge for efficient precondition matching algorithm. Algebraic solutions are proposed. Knowledge representation and some excerpt from inference engine implemented in PROLOG is described. Components of a rule-based system in form of extended attributive decision tables (the so-called XTT paradigm) are presented and their characteristics and applications are outlined.

Future work includes a more robust implementation of the type system, tighter integration with a Java-based runtime, as well as an interface do RDBMS.

ACKNOWLEDGEMENTS

The paper is supported by the Hekate Project funded from 2007–2009 resources for science as a research project.

References

[1] Mordechai Ben-Ari, *Mathematical Logic for Computer Science*, Springer-Verlag, London, 2001.

[2] Ivan Bratko, *Prolog Programming for Artificial Intelligence*, Addison Wesley, 3rd edn., 2000.

[3] Michael A. Covington, Donald Nute, and André Vellino, *Prolog programming in depth*, Prentice-Hall, 1996.

[4] Michael R. Genesereth and Nils J. Nilsson, *Logical Foundations for Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., Los Altos, California, 1987.

[5] Adrain A. Hopgood, *Intelligent Systems for Engineers and Scientists*, CRC Press, Boca Raton London New York Washington, D.C., 2nd edn., 2001.

[6] Peter Jackson, *Introduction to Expert Systems*, Addison–Wesley, 3rd edn., 1999. ISBN 0-201-87686-8.

[7] *Handbook of Data Mining and Knowledge Discovery*, eds., Willi Klösgen and Jan M. Żytkow, Oxford University Press, New York, 2002.

[8] *The Handbook of Applied Expert Systems*, ed., Jay Liebowitz, CRC Press, Boca Raton, 1998.

[9] Antoni Ligęza, *Logical Foundations for Rule-Based Systems*, Springer-Verlag, Berlin, Heidelberg, 2006.

[10] Antoni Ligęza and Grzegorz J. Nalepa, 'Knowledge representation with granular attributive logic for XTT-based expert systems', in *FLAIRS-20 : Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference : Key West, Florida, May 7-9, 2007*, eds., David C. Wilson, Geoffrey C. J. Sutcliffe, and FLAIRS, pp. 530–535, Menlo Park, California, (may 2007). Florida Artificial Intelligence Research Society, AAAI Press.

[11] Grzegorz J. Nalepa and Antoni Ligęza, 'A graphical tabular model for rule-based logic programming and verification', *Systems Science*, **31**(2), 89–95, (2005).

[12] Grzegorz J. Nalepa and Igor Wojnicki, 'Proposal of visual generalized rule programming model for Prolog', in *17th International conference on Applications of declarative programming and knowledge management (INAP 2007) and 21st Workshop on (Constraint) Logic Programming (WLP 2007) : Wurzburg, Germany, October 4–6, 2007 : proceedings : Technical Report 434*, eds., Dietmar Seipel and et al., pp. 195–204, Wurzburg : Bayerische Julius-Maximilians-Universitat. Institut für Informatik, (september 2007). Bayerische Julius-Maximilians-Universitat Wurzburg. Institut für Informatik.

[13] Grzegorz J. Nalepa and Igor Wojnicki, 'Visual software modelling with extended rule-based model : a knowledge-based programming solution for general software design', in *ENASE 2007 : proceedings of the second international conference on Evaluation of Novel Approaches to Software Engineering : Barcelona, Spain, July 23–25, 2007*, eds., Cesar Gonzalez-Perez and Leszek A. Maciaszek, pp. 41–47. INSTICC Press, (july 2007).

[14] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, 2nd edn., 2003.

[15] I. S. Torsun, *Foundations of Intelligent Knowledge-Based Systems*, Academic Press, London, San Diego, New York, Boston, Sydney, Tokyo, Toronto, 1995.