Knowledge Representation with Granular Attributive Logic for XTT-based Expert Systems

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 an extension of classical attributive logic using atomic values of attributes. In the proposed logic set values are allowed and various relational symbols are used to form atomic formulae. The proposed language provides a concise and elegant tool for design, implementation and verification of rule-based systems.

Introduction

Attributive Logic (AL, for short) is one based on the use of *attributes* for denoting some properties of objects and values of system variables in a system under consideration. In order to define current characteristics of the system, one selects some specific set of attributes and assigns them current values. A typical atomic formula (fact) takes the form $A(o) = d$, where A is an attribute, o is an object and d is the current value of A for o. For example,

$$temperature(room\_7) = 17$$

means that the current temperature in $room\_7$ is 17 degrees Centigrade. More complex descriptions take usually the form of conjunctions of such atoms and are omnipresent in the AI literature. It is close to the use of *state variables* in control engineering practice and follows the pattern of relational and object-oriented databases.

It is, however, 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 (Genesereth & Nilsson 1987; Jackson 1999; Torsun 1995), logic for computer science or artificial intelligence (Ben-Ari 2001; Liebowitz 1998), the discussion of syntax and semantics of attribute-based logic is omitted in such positions. It is often assumed, that attributive logic is some *technical language* equivalent with respect to *expressive power* to propositional calculus, and as such it is not worth any more detailed discussion. However, it seems that the real reason for the omission of the presentation is that, in fact a more detailed discussion might be not so straightforward, concise and elegant as in the case of classical logics.

Furthermore, for efficient knowledge specification and management it is certainly not enough to employ atomic values only. More complex values of variables and attributes are necessary. In (Ligęza 2002) a granular approach to algebraic knowledge representation and manipulation was put forward. The ideas of *granular sets* and *granular relations* were defined and proposed to put in use in tabular rule-based systems and databases. In this paper the ideas are followed with main focus on the attributive logic approach to represent such granules and granularly specified knowledge. In fact, granular relations of (Ligęza 2002) can be represented with *minterms* (*Granular Conjunctive Formulae*) incorporating attributes taking set values.

This paper follows the presentation of attributive logic (with set values) given in (Ligęza 2006). Two attribute-based logical languages, namely *Atomic Attributive Logic* (AAL) and *Set Attributive Logic (SAL)* are presented in some details. In AAL the attributes can take atomic (unique) values only; in SAL the attributes can take set values, i.e. several different values at a time. Both syntax and semantics of AAL and SAL are presented. The paper follows the line of extensions recently proposed in (Ligęza & Parra 2006). Some patterns of inference rules specific for these granular languages are given. Finally, application of attributive logic for knowledge representation with PROLOG is presented.

An Example

A very short example introducing the ideas of granular attributive logic is presented in brief. Consider on of the rules specifying the Thermostat System (Negnevitsky 2002), (Ligęza 2006). The rule is as follows:

```
Rule 11
if      the day is Monday or
        the day is Tuesday or
        the day is Wednesday or
        the day is Thursday or
        the day is Friday
then    today is a workday
```

Using granular set values for attributes the rule can be represented in a concise and elegant way as:

$$\texttt{Rule1}: aDD \in sWD \longrightarrow aTD = wd,$$

where aDD is an attribute denoting *the day*, sWD is a set denoting the five working days listed in the rule, aTD is the attribute denoting *today*, and wd is a value denoting *workday*.[1]

This simple example is aimed at providing intuitions concerning the idea and advantages of using granular rather then singular values of attributes for knowledge specification. In the further part a syntax, semantic, inference and PROLOG representation of granular logic is outlined.

Syntax of Attributive Logic

Let there be given the following, pairwise disjoint sets of symbols: O – a set of object symbols, A – a set of attribute names, D – a set of attribute values (the *domains*). It is further frequently assumed that the overall domain D is divided into several sets (disjoint or not), 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 $A = \{A_1, A_2, \ldots, A_n\}$. Then also $D = D_1 \cup D_2 \cup \ldots \cup D_n$, 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 $A_i\colon O \rightarrow D_i$. A generalized attribute A_i is a function (or partial function) of the form $A_i\colon O \rightarrow 2^{D_i}$, 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. A language incorporating such values of attributes will be referred to as SAL (Set Attributive Language). If it does not introduce ambiguity, the qualifier *generalized* may be omitted.

Attribute symbols are used to denote properties of certain objects. To denote the fact that a certain attribute takes a certain value, atomic formulae are used. In case of AAL atomic formulae are defined in the following way (Ligęza 2006).

Definition 1 (AAL) *Let $o \in O$ be a certain object, $A_i \in A$ be an attribute and let $d \in D_i$ be a certain atomic value of the domain of A_i. Any expression of the form $A_i(o) = d$ is an atomic formula of AAL.*

In case of SAL an extended definition is put forward; a modified version of the initial proposal of is given below (Ligęza & Parra 2006).

Definition 2 (SAL) *Let $o \in O$ be some object, $A_i \in A$ be a generalized attribute and let $t \subseteq D_i$ be a certain subset of the domain of A_i. Any expression of the form: $A_i(o) = t$, $A_i(o) \in t$, $A_i(o) \ni t$, $A_i(o) \subseteq t$, $A_i(o) \supseteq t$ and $A_i(o) \sim t$ 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, and is a superset of and have non-empty intersection, respectively. Note that the definition of atomic formulae in SAL (2) covers the one in AAL (1); in fact, any atomic value can be considered as a single-element set. The vice versa is obviously not true. The presented definition is extended w.r.t. the one of (Ligęza 2006) by introducing the forms: $A_i(o) \subseteq t$, $A_i(o) \supseteq t$, $A_i(o) \ni t$ and $A_i(o) \sim t$.

Note also, that the proposed set of relations is 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.

For simplicity, if the object is known, the formulae are simplified to $A_i = d$, $A_i = t$, $A_i \in t$, $A_i \subseteq t$, $A_i \supseteq t$, or $A_i \sim t$ respectively. Without object specification, such simplified formulae are called *selectors* since they can be used for selecting a set of objects satisfying the specific condition. They may be denoted as $[A_i = d]$ and $[A_i = t]$, $[A_i \in t]$, $[A_i \subseteq t]$, $[A_i \supseteq t]$, or $[A_i \sim t]$.

Semantics of Attributive Logic

Below, semantics of atomic formulae of AAL and SAL will be presented. Instead of a formal, theoretical presentation (Ligęza 2006) we rather follow an intuitive discussion based on examples and relativization to simpler formulae.

Consider an atomic formula of AAL of the form $A_i(o) = d$, where d is an atomic value of attribute A_i. The meaning of such a formula is that some *real-world property* (denoted with A_i) of some *real-world object* (denoted with o) takes the unique value (specified by d). We usually think about some *intended meaning* which is defined by a mapping of attribute, object and value denoting symbols into real-world object characteristics, objects and values.

Consider a formula $A_i(o) = t$ of SAL. The interpretation of such a formula can be specified through relativization to the case of AAL. First, consider the case of discrete, finite domains. Using the language of AAL attributes can take only atomic values. After moving to a more expressive language of SAL, attributes can take set values. This means that such an attribute can take more than one value at a time for a given object. Such attributes are specific[2] but useful in numerous practical applications. Especially when the set of values assigned to an object contains numerous values the notation $A_i(o) = t$ can be considered as a useful shorthand.

Consider example of specifying foreign languages known by people. This problem encountered in classical RDBS with atomic values leads to the so-called *fourth normal form*, where values of a set are to be listed in subsequent records.

In case of finite sets, an atom of SAL specifying that an attribute takes a set value can be represented in a (logically equivalent) form incorporating atomic values only. Such a representation of several atomic values is also referred to as *internal conjunction*. Thus instead of writing $Knows(Smith) = English \wedge Knows(Smith) = French \wedge Knows(Smith) = Spanish$ one can simply write $Knows(Smith) = \{English, French, Spanish\}$.

In AAL, where attributes can take atomic values, a conjunctive formula can be transformed into a single atom of SAL according to the following scheme

$$[(A_i(o) = d_1) \wedge \ldots \wedge (A_i(o) = d_j)] \equiv [A_i(o) = t], \quad (1)$$

where $t = \{d_1, d_2, \ldots, d_j\}$ is a subset of D.

[1]We keep the notation used in (Ligęza 2006).

[2]Sometimes such attributes are called multiple-valued attributes.

An analogous extension applies to finite, discrete interval representation (sequences). For example, $A_i = [a, b]$ means that all the values belonging to the interval $[a, b]$ are covered.

Note, however, that in case of infinite domains (even discrete ones) transformation of the form (1) is no longer possible. In case of infinite set t, the atom $A_i(o) = t$ does not have finite internal conjunction form replacing it; hence, the expressive power of SAL is intrinsically higher than the one of AAL and so the propositional logic.

A similar problem to the above occurs when one has to specify a long disjunctive formula specifying *possible* atomic values for the same object and attribute.

Consider an atomic formula of the form $A_i(o) \subseteq t$. In fact, in the atomic representation in AAL, where attributes can take atomic values, a disjunctive formula as below can be transformed into a single atom of SAL according to the following principle:

$$[(A_i(o) = d_1) \vee \ldots \vee (A_i(o) = d_j)] \equiv [A_i(o) \subseteq t], \quad (2)$$

where $t = \{d_1, d_2, \ldots, d_j\}$ is a subset of D.

An analogous extension applies to finite, discrete interval representation. For example, $A_i \in [a, b]$ means that all the values belonging to the interval $[a, b]$ are possible values for A_i (both in case of discrete and continuous domain of the attribute).

Note, however, that the above transformation (2) can be applied only in case of finite domains. In case of infinite domains, both countable and continuous, the set notation of SAL has no equivalent in the language of AAL. Hence, the expressive power of SAL is higher than the one of AAL and obviously of propositional logic.

Now, consider a formula of the form $A_i(o) \in t$. This time, it is assumed that attribute A take a unique value belonging to t. Let $\oplus$ denote the *Exclusive Or* logical connective. In case of finite sets the following transformation is possible:

$$[(A_i(o) = d_1) \oplus \ldots \oplus (A_i(o) = d_j)] \equiv [A_i(o) \in t], \quad (3)$$

where $t = \{d_1, d_2, \ldots, d_j\}$ is a subset of D.

As before, the above transformation (3) can be applied only in case of finite domains. In case of infinite domains, both countable and continuous, the set notation of SAL has no equivalent in the language of AAL. Hence, also in this case, the expressive power of SAL is higher than the one of AAL and obviously of propositional logic.

Consider a formula of the form $A_i(o) \supseteq t$. The intuitive meaning is that the set of values of attribute A for object o covers the set t, i.e. constitutes a superset of t. In case of finite domains the interpretation can be presented with use of *external conjunction*. Let $A_i(o) = \{d_1, d_2, \ldots, d_j\}$; this can be interpreted as:

$$(A_i(o) = d_1) \wedge (A_i(o) = d_2) \wedge \ldots \wedge (A_i(o) = d_j). \quad (4)$$

Then, $A_i(o) \supseteq t$ simply means that $t \subseteq \{d_1, d_2, \ldots, d_j\}$.

Next, consider a formula of the form $A_i(o) \ni t$. This time, it is assumed that attribute A take a unique or set value to which t (a single element) belongs. In case of finite sets the following transformation is possible:

$$[(A_i(o) = d_1) \vee \ldots \vee (A_i(o) = d_j)] \equiv [A_i(o) \ni t], \quad (5)$$

where $t \in \{d_1, d_2, \ldots, d_j\}$ is a single-element of D.

As before, the above transformation (5) can be applied only in case of finite domains. In case of infinite domains, both countable and continuous, the set notation of SAL has no equivalent in the language of AAL. Hence, also in this case, the expressive power of SAL is higher than the one of AAL and obviously of propositional logic.

Finally, consider an atomic formula of the form $A_i(o) \sim t$. The meaning is that the intersection of $A_i(o)$ and t is a non-empty set, i.e. $A_i(o) \cap t \neq \emptyset$.

Granular Formulae in Attributive Logic

More complex formulae in attributive logic can be generated from the atomic ones with the use of logical connectives (Ligęza 2006). For the purposes of this paper we are interested only in *simple conjunctive formulae* (the so called minterms) and in *Disjunctive Normal Form* (DNF) formulae.

For specification of rules it is important to have the possibility to precisely define a set of constraints which have to be satisfied at the same time instant. For the purpose of that one normally uses a conjunction of atomic formulae (with or without negation). Such a conjunction is normally called a *simple conjunctive formula* or a *minterm* (Ligęza 2006).

For the sake of specifying rules in this paper we shall define a special form of conjunctions of positive atomic formulae (a subset of minterms). Since such a conjunction can be interpreted as a *granule* in the conceptual space defined by the attributes, we shall call it a *Granular Conjunctive Formula* (GCF) or a *box*.

Definition 3 (Granular Conjunctive Formula) *Let $A_1, A_2, \ldots, A_n$ be some attributes, $o_1, o_2, \ldots, o_n$ denote certain objects, and let $t_1, t_2, \ldots, t_n$ be some subsets of the domains of given attributes, where $t_i \subseteq D_i$, for $i = 1, 2, \ldots, n$. A formula ψ,*

$$\psi = A_1(o_1) = t_1 \wedge A_2(o_2) = t_2 \wedge \ldots \wedge A_n(o_n) = t_n \quad (6)$$

will be called a (strong) Granular Conjunctive Formula or a (strong) box. The Cartesian Product $t_1 \times t_2 \times \ldots \times t_n$ will be called a granule.

Often $o_1 = o_2 = \ldots = o_n$, i.e. the attributes describe the same object. In terms of relational databases such a formula can be interpreted as a table of the scheme defined by the sequence of attributes $A_1, A_2, \ldots, A_n$ and covering all the Cartesian Product $t_1 \times t_2 \times \ldots \times t_n$, but in case of infinite t_i (e.g. $t_i \subset R$), there is no finite relational model.

Definition 4 (Weak Granular Formula) *Let $A_1, A_2, \ldots, A_n$ be some attributes, $o_1, o_2, \ldots, o_n$ denote certain objects, and let $t_1, t_2, \ldots, t_n$ be some subsets of the domains of given attributes, where $t_i \subseteq D_i$, for $i = 1, 2, \ldots, n$. A formula ϕ,*

$$\psi = A_1(o_1) \sqsubseteq t_1 \wedge A_2(o_2) \sqsubseteq t_2 \wedge \ldots \wedge A_n(o_n) \sqsubseteq t_n \quad (7)$$

will be called a Weak Granular Conjunctive Formula or a weak box; here $\sqsubseteq$ denotes either $\subseteq$ or $\in$. The Cartesian Product $t_1 \times t_2 \times \ldots \times t_n$ will be called a boundary granule.

In order to cover some arbitrary area (especially a non-convex one) located in the Cartesian Product of the attribute domains $D_1 \times D_2 \times \ldots \times D_n$ one may need a disjunctive composition of Granular Conjunctive Formulae. Such a formula is a specific case of the Disjunctive Normal Formula (DNF) and it is defined as follows.

Definition 5 *Let $\psi_1, \psi_2, \ldots, \psi_k$ be some strong GCF. A formula Ψ of the form*

$$\Psi = \psi_1 \vee \psi_2 \vee \ldots \vee \psi_k \tag{8}$$

will be called a (strong) Granular DNF.

In case of weak GCF components we have the following definition.

Definition 6 *Let $\phi_1, \phi_2, \ldots, \phi_k$ be some weak GCF. A formula Φ of the form*

$$\Phi = \phi_1 \vee \phi_2 \vee \ldots \vee \phi_k \tag{9}$$

will be called a Weak Granular DNF.

Inference in Attributive Logic

Firstly, consider two subset symbols $s, t \subseteq D$. Obviously, for any object symbol $o \in O$ and any attribute $A_i \in A$, if t is a subset of s the following rule holds:

$$\frac{A_i(o) = s}{A_i(o) = t} \tag{10}$$

Rule (10) will be referred to *downward consistency rule* or *subset consistency rule*. The meaning of the downward consistency rule is obvious — if an attribute takes values from a certain set, then certainly its values stay within any subset of that set.

By analogy, consider two subset symbols $s, t \subseteq D$. Obviously, for any object symbol $o \in O$ and any attribute $A_i \in A$, if s is a subset of t the following rule holds:

$$\frac{A_i(o) \subseteq s}{A_i(o) \subseteq t} \tag{11}$$

Rule (11) will be referred to *upward consistency rule* or *superset consistency rule*. The meaning of the upward consistency rule is obvious — if an attribute takes values from a certain set, then certainly its values stay within any superset of that set. Note that, the rule holds also for the relational symbol $\in$, i.e.:

$$\frac{A_i(o) \in s}{A_i(o) \in t} \tag{12}$$

as a specific case. Further, the following rule holds as well: if t is a subset of s the following rule holds:

$$\frac{A_i(o) \supseteq s}{A_i(o) \supseteq t} \tag{13}$$

In fact, this is and inverse (or dual) rule to (11).

As an example consider two atomic formulae as follows: $A_i(o) = t$ and $A_i(o) \subseteq s$. Taking into account the interpretation, if $t \subseteq s$, then $[A_i(o) = t] \models [A_i(o) \subseteq s]$.

In the table in Fig. 1 we examine when the atoms located in the left-hand column imply atoms in the header (for simplicity, A is used instead of A_i). The table specifies inference possibilities among atomic formulae for checking satisfaction of rule preconditions. In fact, if the fact base contains facts of the form specified in the left-hand column, the preconditions contains facts of the header line, and the appropriate relation holds, the satisfaction of atomic part of the precondition can be claimed.

Application of Inference Rules for Verification

The inference rules specified for Attributive logic can be applied for checking if preconditions of a rule are satisfied and in verification of theoretical properties of rule-based systems (Ligęza 2006). Note that for

- checking satisfaction of rule preconditions (rule firing),
- checking logical relationship among preconditions of rules (e.g. rule subsumption),
- checking covering of a set of states by a set of rules (completeness check),
- detection of overlapping rule preconditions (indeterministic and inconsistent rules)

the basic operation consists in checking for logical consequence. More details on that are given in (Ligęza 2006).

Rule-Based Systems in Attributive Logic

Numerous rule-based systems use simple knowledge representation logic based on attributes. Unfortunately, most of the systems allow for use of very simple atomic formulae only. Two most typical examples are of the form $A = d$ and $A(o) = d$, where A is an attribute, o is an object, and d is an *atomic* value of the attribute. In this way the specification of attribute values is restricted to atomic values only.

In the proposed XTT approach (Nalepa 2004) an *extended attributive language* is used. In fact we use *SAL*, the *Set Attributive Language*. XTT uses extended attributive decision rules for the construction of rule-based systems. A rule is based on the basic rule format but includes both the *control statement* and *dynamic operation* definitions. Hence, it can operate on the system memory and show where to pass control in an explicit way. The full rule format incorporates the following components: *a unique identifier* of the rule (it can be the name or the number of the rule, or both), *a context formula* defining the context situation in which the rule is aimed to operate, *preconditions* of the rule (specifying the logical formula that has to be satisfied in order that the rule can be executed), *a dynamic operation specification* with the use of *retract* and *assert* parts, *a conclusion/decision* part being the output of the rule, and *a control specification* with the use of the *next* part.

The above components can be presented as follows:

$$\begin{array}{ll}
\texttt{rule(i):} & \texttt{context} = \psi \ \ \texttt{and} \\
 & [A_1 \in t_1] \wedge [A_2 \in t_2] \wedge \ldots \wedge [A_n \in t_n] \\
 & \longrightarrow \\
 & \texttt{retract}(B_1 = b_1, B_2 = b_2, \ldots, B_b = b_b), \\
 & \texttt{assert}(C_1 = c_1, C_2 = c_2 \ldots, C_c = c_c), \\
 & \texttt{do}(H_1 = h_1, H_2 = h_2, \ldots, H_h = h_h), \\
 & \texttt{next(j)}, \texttt{else(k)}.
\end{array}$$

where ψ defines the specific environmental and internal conditions under which the rule is valid, $[A_1 \in t_1] \wedge \ldots \wedge [A_n \in t_n]$ is the regular precondition formula, $B_1 = b_1, B_2 = b_2, \ldots, B_b = b_b$ is the specification of the facts to be retracted from the knowledge base, $C_1 = c_1, C_2 = c_2, \ldots, C_c = c_c$ is the specification of the facts to be asserted to the knowledge base, $H_1 = h_1, \ldots, H_h = h_h$ is the

$\models$	$A = t$	$A \in t$	$A \ni t$	$A \subseteq t$	$A \supseteq t$	$A \sim t$
$A = s$	$t \subseteq s$	$s \in t$	$t \in s$	$s \subseteq t$	$t \subseteq s$	$s \cap t \neq \emptyset$
$A \in s$	$t = \{s\}$	$s \subseteq t$	$t = s$	$s \subseteq t$	$t = \{s\}$	$s \subseteq t$
$A \ni s$	$s = t$	_		$s = t$	$t = \{s\}$	$s \in t$
$A \subseteq s$	$s = \{t\}$	$s \in t$	$t \in s$	$s \subseteq t$	_	$s \subseteq t$
$A \supseteq s$	$t \subseteq s$	$t \in s$	$t \in s$	_	$t \subseteq s$	$s \subseteq t$
$A \sim s$	_	_	$s = \{t\}$	_	_	$s \subseteq t$

Figure 1: Inference in SAL.

specification of conclusions forming a direct output of the rule, *next(j)* is the control specification, possibly including the PROLOG *cut* marker.

Rules of a similar attribute structure can be easily combined into a special form of decision table, the *Extended Attributive Table* (Ligęza 2006; Nalepa 2004) (XAT, or *eXtended Table* (XT), for short).

An important feature of XTT is the fact that, besides visual representation important for the design (Ligęza, Wojnicki, & Nalepa 2001; Nalepa 2004; Nalepa & Ligęza 2005), it offers a well-defined, *logical representation* (Nalepa 2004; Ligęza 2006). The XTT hierarchy can be mapped to the corresponding PROLOG code.

Transformation from XTT tables to a PROLOG-based representation allows for obtaining a *logically equivalent* code that can be executed, analyzed, verified, optimized, translated to another language, transferred to another system, etc.

In order to fully represent the XTT model using SAL the *fact* (Object-Attribute-Value triple) representation has to specified, the attribute *domains* with all the constraints have to be represented, a rule syntax has to be defined.

Every XTT *cell* corresponds to a certain *fact* in the rule base. An XTT fact is represented by the following term:

$$f(< attribute\_name >, < value\_type >, < value >)$$

where `attribute_name` is an XTT attribute name prefixed by a lower-case `a` in order to prohibit upper-case names (they would be interpreted as variables in PROLOG); `value_type` is one of `{atomic, set, interval, natomic, nset, ninterval}`, and `value` is the attribute value held in cell, possibly a non-atomic one.

In order to represent different attribute types and value domains the following rules are established:

- Atomic values, e.g $A(O) = V$, are represented by `f(aA,atomic,V)` term.
- Negated atomic values, e.g $A(O) \neq V$, are represented by `f(aA,natomic,V)` term.
- Non-atomic numerical values, such as $A(O) \in < x, y >$, are represented by `f(aA,interval,i(x,y))` term.
- Negated non-atomic numerical values, such as $A(O) \notin (x, y)$, or $A(O) \notin < x, y >$) are represented by `f(aA,ninterval,i(x,y))` term.
- Non-atomic symbolic values, such as $A(O) \in \{monday, tuesday\}$ are represented by `f(aA,set,Seti)` term, where `Seti` is a predefined set $Set_i = \{monday, tuesday\}$.
- Negated non atomic symbolic values, such as: $A(O) \notin \{monday, tuesday\}$ are represented by `f(aA,nset,Seti)` term, where `Seti` is a predefined set $Set_i = \{monday, tuesday\}$.

Considering that every attribute domain has lower and upper constraints, and there is a predefined real numbers precision, every relational expression with numerical value can be mapped to an *interval* of the form: $< v1, v2 >$.

Rules are represented as PROLOG *facts*. 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). Using PROLOG for meta-programming (writing an interpreter) is a standard approach used in the implementation of advanced PROLOG applications. The extended rule syntax is:

```
rule(table-num, rule-num, precondition-list,
    retract-list, assert-list, decision-list,
    next-table, next-rule in next-table).
```

In this application the *else* part is implicitly considered to be the next rule in the current table.

The whole table-tree structure of XTT is represented by one *flat* rule-base. The rule-base is separated from the inference engine code. All tables have unique identifiers (numbers), and rules are assigned unique numbers too. This allows for a precise inference control. For example, an excerpt of rule-base for the Thermostat example (Ligęza 2006) is represented by the following PROLOG code:

```
rule(2,3, [f(aTD,atomic,wd), f(aTM,interval,i(9,17))],
     [f(aOP,atomic,_)], [f(aOP,atomic,true)], [], 3,7).
rule(2,4, [f(aTD,atomic,wd), f(aTM,interval,i(0,8))],
     [f(aOP,atomic,_)], [f(aOP,atomic,false)], [], 3,7).
rule(2,5, [f(aTD,atomic,wd), f(aTM,interval,i(18,24))],
     [f(aOP,atomic,_)], [f(aOP,atomic,false)], [], 3,7).
rule(2,6, [f(aTD,atomic,wk)],
     [f(aOP,atomic,_)], [f(aOP,atomic,false)], [], 3,7).
```

where: `aTD`, `aTM`, `aOP`, are abbreviated attribute names: *today*, *time*, *operation* respectively. In order to interpret XTT rules there is a need for a *meta-interpreter*. As a proof-of-concept an XTT meta-interpreter engine has been developed and described in detail in (Nalepa 2004; Nalepa & Ligęza 2006). A code-excerpt from the PROLOG inference engine for proving logical satisfaction in a limited version of the granular attributive logic is presented below.

```
satisfied([]) :- !.
satisfied([Fact|Facts]) :-
```

```
  valid(Fact), satisfied(Facts).
fails([Fact|_])  :-
  \+ valid(Fact), !.
fails([_|Facts]) :-
  fails(Facts).
valid(f(A,atomic,V))  :-
  f(A,atomic,V), !.
valid(f(A,natomic,V)) :-
  f(A,atomic,W), V \== W, !.
valid(f(A,set,Set))   :-
  f(A,atomic,V), set(Set,SetValue),
  member(V,SetValue),!.
valid(f(A,nset,Set))  :-
  f(A,atomic,V), set(Set,SetValue),
  \+ member(V,SetValue),!.
valid(f(A,interval,i(B,E)))  :-
  f(A,atomic,V), V >= B, V =< E.
valid(f(A,ninterval,i(B,_))) :-
  f(A,atomic,V), V < B.
valid(f(A,ninterval,i(_,E))) :-
  f(A,atomic,V), V > E.
```

Conclusions

This paper presents a development of logical and algebraic approach of (Ligęza & Parra 1996), (Ligęza 2006). In particular, it presents attributive logic for knowledge representation and knowledge management in knowledge engineering applications. A particular interest was devoted to *Set Attributive Logics* (SAL) where attributes can take set values. Five basic forms of atomic formulae of SAL were introduced. The relations symbols used were $=$ (equality), $\in$ (element of), $\subseteq$ (subset), $\supseteq$ (superset) and $\sim$ (non-empty intersection).

In this kind of logic, when attributes can take set values, conjunctive formulae can be interpreted as *granules* in the conceptual space of attributes (Ligęza 2002). Such a granular formula can be used for efficient specification of preconditions of inference rules. Moreover, thanks to the phenomenon of internal conjunction, it can specify facts and relations in the knowledge base in an efficient way. This is so because a granular formula covers a number of formulae incorporating atomic values only (of AAL).

Specific rules of inference for the introduce logic were also presented. Such rules are necessary when checking preconditions of rules expressed with SAL. The introduced logic is applied in the *eXtended Tabular Trees* (XTT) formalism for knowledge representation (Ligęza 2006; Nalepa 2004).

The benefits of using granular attributive logic include much more concise specification of knowledge than in the case of logic using atomic values only followed by possibilities of more efficient knowledge verification and easier design. The price of higher expressive power is that verification procedures require more complex checks based on logical inference and not just simple comparison (as in the case of atomic values of attributes). An idea of a tool for supporting the design, implementation and verification of rule-based systems with granular logic is presented in (Nalepa 2004), (Ligęza 2006).

References

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

Genesereth, M. R., and Nilsson, N. J. 1987. *Logical Foundations for Artificial Intelligence*. Los Altos, California: Morgan Kaufmann Publishers, Inc.

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. ISBN 0-8493-3106-4.

Ligęza, A., and Parra, P. F. 1996. Towards logical analysis of tabular rule-based systems. In Trappl, R., ed., *Proc. of the 13th European Meeting on Cybernetics and Systems Research (EMCSR'96)*, volume 2, 1211–1216.

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.; Wojnicki, I.; and Nalepa, G. 2001. Tab-trees: a case tool for design of extended tabular systems. In et al., H. M., ed., *Database and Expert Systems Applications*, volume 2113 of *Lecture Notes in Computer Sciences*. Berlin: Springer-Verlag. 422–431.

Ligęza, A. 2002. Granular algebra: Towards a calculus of semipartitions for analysis, manipulation and verification of tabular systems. In Trappl, R., ed., *Proc. of the 16th European Meeting on Cybernetics and Systems Research (EMCSR'02)*, 806–811.

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

Nalepa, G. J., and Ligęza, A. 2005. Conceptual modelling and automated implementation of rule-based systems. In Krzysztof Zieliński, T. S., ed., *Software engineering : evolution and emerging technologies*, volume 130 of *Frontiers in Artificial Intelligence and Applications*, 330–340. IOS Press.

Nalepa, G. J., and Ligęza, A. 2006. Prolog-based analysis of tabular rule-based systems with the xtt approach. In Sutcliffe, G. C. J., and Goebel, R. G., eds., *FLAIRS 2006 : proceedings of the nineteenth international Florida Artificial Intelligence Research Society conference : [Melbourne Beach, Florida, May 11–13, 2006]*, 426–431. FLAIRS. - Menlo Park: Florida Artificial Intelligence Research Society.

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, 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.

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