# An Attribute Logic with Set Values for Rule-Based Systems

Antoni Ligęza
Institute of Automatics,
AGH – University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
Email: ligeza@agh.edu.pl

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

***Abstract*—This paper presents advances in Set Attributive Logic for application in tabular rule-based systems developed within the XTT framework. An ultimate goal of this work is to extend the expressive power of simple attributive language for efficient dealing with set values. 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 level and implementation level are presented.**

## I. Introduction

Attributive logics constitute a simple yet widely-used tool for knowledge specification and development of rule-based systems. In fact in a large variety of applications in various areas of Artificial Intelligence (AI) [1] and Knowledge Engineering (KE) attributive languages constitute the core knowledge representation formalism. The most typical areas of applications include rule-based systems [2], [3], expert systems (ones based on rule formalism) [4], [5], [6] and advanced database and data warehouse systems with knowledge discovery applications [7] and contemporary business rules and business intelligence components (e.g. Jess, Drools) [8].

This paper presents advances in Set Attributive Logic introduced in [3], as well as its application to develop tabular rule-based systems within the XTT framework [9]. The primary goal is to extend the expressive power of simple attributive language 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 with their corresponding prototype in Prolog.

## II. Basic Attributive Logic

### A. Classics of Attributive Languages

No wonder that using logics based on attributes is one of the most popular approaches to define knowledge. Not only it is very intuitive, but it follows simple technical way of discussion where the behavior of physical systems is formalized by providing the values of system variables. This kind of logic is omnipresent in various applications through its very generic character; it constitutes the bases for construction of relational database tables, attributive decision tables and decision trees [7], attributive rule-based systems [3] and is often applied to describe state of dynamic systems and autonomous agents. Some most typical examples include expert systems and decision support systems, as well as rule-based control and monitoring systems as well as diagnostic systems.

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 AI and KE applications [10], [4], [5], logic for computer science or Artificial Intelligence [11], [2], the discussion of syntax and semantics of attribute-based logic is omitted in such positions[1].

On the contrary, it is often assumed by default, that attributive logic is some kind of *technical language* equivalent with respect to its *expressive power* to propositional calculus, and as such it is not worth any more detailed discussion. On the other hand, it seems that some of the real reasons for the omission of the presentation is that, a more detailed discussion might be not so straightforward, concise and elegant as in the case of classical logics. In fact, as it follows from some first attempts presented in [3] this issue requires a more detailed study.

The most typical way of thinking about attributive logic use for knowledge specification may be put as follows:

- first, one has to define *facts*, typically of the form

  $$A = d$$

  or

  $$A(o) = d,$$

  where $A$ is a certain attribute, $o$ an object of interest and $d$ is the attribute value.
- second, facts are perceived as propositional logic atomic formulae,
- third, the syntax and semantics of propositional calculus are freely used.

This basic approach is sometimes extended with use of certain syntax modifications. For example, in [7] the discussion is

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

[1]Note that even in the four-volume handbook of *Logics for Artificial Intelligence* edited by D.Gabbay et. al the *Attribute Logic* has not deserved a few pages of formal presentation and analysis of properties. Even wikipedia does not cover this issue.

extended, so that the rules take the form:

$$A_1 \in V_1 \wedge A_2 \in V_2 \wedge \ldots A_n \in V_n \longrightarrow A_{n+1} = d.$$

Following this line of extended knowledge specification, various relational symbols can be introduced, e.g. $A_i > d$ (for ordered sets; this can be considered as a shorthand for $A_i \in V_i \setminus V_d$, where $V_d$ is the set of all the values of $A_i$ less than or equal to $d$) or $A_i \neq d_i$ (this can be considered as a shorthand for $A_i \in V_i \setminus \{d_i\}$).

Note however, that extending the syntax in such a way preserves the limitation that an attribute can take a single value at a time. Further, without providing a clearly defined semantics for the language and some formal inference rules it may lead to severe problems. This follows from the fact that atoms are no longer *logically independent* (which is the basic, also implicit assumption of propositional logics [3]). For example, having a rule such as

$$Temperature > 100 \longrightarrow WaterState = boiling$$

and a piece of knowledge like $Temperature > 123$, we would not be able to fire the rule using classical inference rules[2].

### B. Set Attributive Logic

In a recent book [3] the discussion of attributive logic is much more thorough. The added value consist in allowing that attributes can take *set values* and providing some formal framework 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 [3] it is assumed that an *attribute* $A_i$ is a function (or partial function) of the form $A_i \colon O \to D_i$. Here $O$ is a set of object and $D_i$ is the domain of attribute $A_i$. A *generalized attribute* $A_i$ is a function (or partial function) of the form $A_i \colon O \to 2^{D_i}$, 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: $A_i(o) = d$, $A_i(o) = t$ or $A_i(o) \in t$, 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. If the object $o$ is known (or unimportant) its specification can be skipped; hence we write $A_i = d$, $A_i = t$ or $A_i \in t$, for simplicity.

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 *one* or *some* of the values of $t$ (the so-called *internal disjunction*).

As an example for the necessity of SAL one can consider the specification of working days (denoted with $WDay$) given as

$$WDay = D,$$

where $D$ is the set of working days, $D = \{Monday, Tuesday, Wednesday, Thursday, Friday\}$. Now one can construct an atomic formula like $CurrentDay \in D$, or a rule of the form:

$$DayOfInterest \in D \longrightarrow Status(OfficeOfInterest) = open.$$

The SAL as introduced in [3] 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 towards Finite Domains (FD) and its expressive power is increased through introduction of new relational symbols. The semantics is also clarified. 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).

## III. Attribute Logic with Set Values over Finite Domains

An extension of SAL was proposed in [12]. Both the syntax and semantics were extended and clarified. Here some further details to support set values of attributes over finite domains are discussed.

### A. Introduction to ALSV(FD)

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:

**A** – a finite set of attribute names,
**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. The set can be ordered, partially ordered, or unordered; in case of ordered (partially ordered) sets some modifications of notation is allowed.

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 $A_i \colon T \to D_i$ (i.e. taking a single value at any instant of time) and *generalized* ones of the form $A_i \colon T \to 2^{D_i}$ (i.e. taking a set of values at a time); here $T$ denotes the time domain of discourse.

### B. Syntax of ALSV(FD)

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.

[2]Well, some expert system shells, such as PC-SHELL (see http://aitech.pl/) are capable of performing the so-called *intelligent unification* and hence succeed to carry on with the inference; this has however nothing to do with logic, it is just hard-wired implementation of a specialized match mechanism which works only for predefined symbols.

*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 $\otimes$ is stay 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, relational symbols such as $>$, $>=$, $<$, $=<$ can be used with the straightforward meaning.

### C. Semantics of ALSV

The semantics of the proposed language is presented below in an informal way. The semantics of $A = V$ is basically the same as the one of (Sec. II-B) [3]. If $V = \{d_1, d_2, \ldots, d_k\}$ then $A = V$ is equivalent to

$$A \supseteq \{d_1\} \wedge A \supseteq \{d_2\} \wedge \ldots \wedge A \supseteq \{d_k\},$$

i.e. the attribute takes all the values specified with $V$ (and nothing more).

The semantics of $A \subseteq V$, $A \supseteq V$ and $A \sim V$ is defined as follows:

$$A \subseteq V \equiv A = U$$

where $U \subseteq V$, i.e. $A$ takes *some* of the values from $V$ (and nothing out of $V$),

$$A \supseteq V \equiv A = W,$$

where $V \subseteq W$, i.e. $A$ takes *all* of the values from $V$ (and perhaps some more), and

$$A \sim V \equiv A = X,$$

where $V \cap X \neq \emptyset$, i.e. $A$ takes *some* of the values from $V$ (and perhaps some more). As it can be seen, the semantics of ALSV is defined by means of relaxation of logic to simple set algebra.

## IV. Basic Inference Rules for ALSV(FD)

Since the presented language is an extension of the SAL (Set Attributive Logic) presented in [3], 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 automated inference in a rule-based system, 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. I. The table is to be read as follows: if an atomic formula in the leftmost column holds, and a condition stated in the same row is true, the to appropriate atomic formula in the topmost row is a logical consequence of the one from the leftmost column.

Table II
INFERENCE RULES FOR ATOMIC FORMULAE FOR GENERALIZED ATTRIBUTES

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

Table I
INFERENCE RULES FOR ATOMIC FORMULAE FOR SIMPLE ATTRIBUTES

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

Table III
INCONSISTENCY CONDITIONS FOR PAIRS OF ATOMIC FORMULAE

| $\not\models$ | $A = W$ | $A \subseteq W$ | $A \supseteq W$ | $A \sim W$ |
|---|---|---|---|---|
| $A = V$ | $W \neq V$ | $V \not\subseteq W$ | $W \not\subseteq V$ | $V \cap W \neq \emptyset$ |
| $A \subseteq V$ | $W \not\subseteq V$ | $V \cap W = \emptyset$ | $W \not\subseteq V$ | $W \cap V = \emptyset$ |
| $A \supseteq V$ | $V \not\subseteq W$ | $V \not\subseteq W$ | _ | _ |
| $A \sim V$ | $V \cap W \neq \emptyset$ | $W \not\subseteq V$ | _ | _ |

The summary of the inference rules for atomic formulae with generalized attributes (where an atomic formula is the logical consequence of another atomic formula) is presented in Table. II.

In Table I and Table II 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 I and Table II can be used for checking if preconditions of a formula hold or verifying subsumption among rules.

For further analysis, e.g. of intersection (overlapping) of rule preconditions one may be interested if two atoms cannot simultaneously be true and if so — under what conditions. For example formula $A \subseteq V \wedge A \subseteq W$ is inconsistent if $V \cap W = \emptyset$. Table III specifies the conditions for inconsistency.

The interpretation of the Table III is straightforward: if the condition specified at the intersection of some row and column holds, then the atomic formulae labelling this row and column cannot simultaneously hold. Note however, that this is a satisfactory condition only.

Table III can be used for analysis of determinism of the system, i.e. whether satisfaction of precondition of a rule implies that the other rules in the same table cannot be fired.

Table IV
A GENERAL SCHEME OF AN XTT TABLE

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

## V. RULES IN ATTRIBUTE LOGIC

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 [3]. 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.

### A. Rule Format

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 [3].

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. IV

In Table IV 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 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).

### B. Rule Firing

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 I for simple attributes and the first row of Table II 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 [3].

In order to avoid repeated checking of preconditions a *propagation mechanism* is proposed for satisfaction and falsification of atomic formula within the table. Let $c(i, j)$ denote the atomic formula related to the cell located in row $i$ and column $j$. The idea can be summarized as follows:

- once the table is defined it is searched top-down (off-line) for establishing dependencies between any atomic cell $c(i, j)$ of rule $i$ and all the cells $c(k, j)$ (in the same column) of any rule $k$, where $k > i$;
- in case some two cells $c(i, j)$, $c(k, j)$ satisfy a condition of logical consequence as specified in Table I, a positive link $p(i, k, j)$ is established; all the links are kept in memory;
- in case some two cells $c(i, j)$, $c(k, j)$ satisfy a condition of logical inconsistency as specified in Table II, a negative link $n(i, k, j)$ is established; all the links are kept in memory;
- during execution phase, if a cell $c(i, j)$ is checked and the related atomic formula is satisfied, the truth value is propagated for the transitive closure defined with use of the positive links; the respective atoms are marked *true* and do not need to be checked in this turn;
- during execution phase, if a cell $c(i, j)$ is checked and the related atomic formula is true, the false value is propagated for the transitive closure defined with use of the negative links; the respective atoms are marked *false* and the corresponding rules are eliminated from this cycle.

This mechanism saves computational effort corresponding to repeated precondition checking and saves time in case some preconditions are logically dependent (one is logical consequence of the other or they are mutually exclusive).

Table V
A TABULAR RULE-BASED SYSTEM FOR DEFINING BUSINESS HOURS

| $I$ | $A_1$ | $A_2$ | $C$ |
|---|---|---|---|
| 3 | $V_1$ | [9:00, 17:00] | $dbh$ |
| 4 | $V_1$ | [0:00, 9:00] | $ndbh$ |
| 5 | $V_1$ | [17:00, 24:00] | $ndbh$ |
| 6 | $V_2$ | – | $ndbh$ |

### C. An Example

As a simple example consider the rules for determining thermostat settings [13] further explored in [3]. The original set has 18 rules forming four groups. Below we consider only four rules using the same attributes for defining preconditions and working within the same context.

*Rule 3 if* today is workday *and* the time is 'between 9 am and 5 pm' *then* operation is 'during business hours'.
*Rule 4 if* today is workday *and* the time is 'before 9 am' *then* operation is 'not during business hours'.
*Rule 5 if* today is workday *and* the time is 'after 5 pm' *then* operation is 'not during business hours'.
*Rule 6 if* today is weekend *then* operation is 'not during business hours'.

Let there be given the following attributes: $A_1$ denoting *today* and $A_2$ denoting *time* and $C$ denoting conclusion. The values for $C$ can be: $dbh$ means *during business hours* and $ndbh$ – *not during business hours*. Let $V_1$ denote the set of working days (from Monday to Friday) and let $V_2$ denote weekend days (Saturday and Sunday). The rules can be encoded within the following Table V. Before execution, the following positive and negative links can be established: $p(3, 4, 1)$, $p(3, 5, 1)$, $p(4, 5, 1)$, $n(3, 6, 1)$, $n(4, 6, 1)$, $n(5, 6, 1)$, $n(3, 4, 2)$, $n(3, 5, 2)$, $n(4, 5, 2)$.

For example, $p(3, 4, 1)$ means that if the first precondition of rule 3 is found true, so is the first precondition of rule 4; $n(3, 6, 1)$ means that if the first precondition of rule 3 is found true, then the first precondition of rule 4 is false.

In case of firing rule 3, 4 or 5 all the other rules cannot be fired; if these rules fail to be fired, rule 6 can be executed.

### D. Rule Analysis

The advantage of tabular rule-based systems defined with use of attributive logic is that analysis of the rules becomes simpler and can be performed with algebraic tools. As an example consider two typical cases, e.g. detection of subsumption and overlapping preconditions which may lead to conflict or indeterminism.

*1) Subsumption:* Consider two rules given by two rows of a table; the rules are of the form:

$$A_1 \propto_1 V_{i,1} \wedge A_2 \propto_2 V_{i,2} \wedge \ldots A_n \propto_n V_{i,n} \longrightarrow RHS_i$$

for being some $i1$ and $i2$. For simplicity we consider that the $RHS_{i1} = RHS_{i2}$ (Right Hand Side, the conclusions are identical. Rule $i1$ subsumes rule $i2$ if always when $i2$ can be fired $i1$ can be fired as well.

The analysis for subsumption cane be performed with help of Table I and Table II. In order to conclude that subsumption holds one is to check that

$$A_j \propto_j V_{i2,j} \models A_j \propto_j V_{i1,j},$$

for $j = 1, 2, \ldots, n$. In case it is true, rule $i2$ can be eliminated.

*2) Indeterminism and Inconsistency:* Once again, consider two rules $i1$ and $i2$ given by two rows of a table as in section V-D1. A first step to discover indeterminism is two check if the rules can be fired together i.e. if their preconditions can be satisfied simultaneously.

The analysis for subsumption cane be performed with help of Table II. In order to conclude that the preconditions cannot be satisfied at the same time one has to check that

$$\not\models A_j \propto_j V_{i2,j} \wedge A_j \propto_j V_{i1,j},$$

for at least one value of $j \in \{1, 2, \ldots, n\}$. In case it is true, the set of rules is deterministic, i.e at any time during execution only a single rule can be fired. If not, the pairs (or bigger groups) of rules should be further analyzed to eliminate potential inconsistency..

## VI. Prototype Implementation

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(<domain_name>,<list_of_values>)
domain(d7,[1,2,3,4,5,6,7]).
domain(day_of_week,[monday,tuesday,wednesday,
             thursday,friday,saturday,sunday]).
domain(season,[spring,summer,autumn,winter]).

% attribute(<attribute-name>,<attribute-type>,
%            <attribute-domain>).
attribute(aDN,atomic,d7).
attribute(aDD,atomic,day_of_week).
attribute(aSE,atomic,season).
attribute(sDN,set,d7).
attribute(sDD,set,day_of_week).
attribute(sSE,set,season).
```

By convention, atomic attribute names start with 'a' (e.g. `aDN`) while the generalized attributes names starts with 's' (e.g. `sDN`).

The atomic formulae (facts) are represented as terms of the type `fact/4` with four arguments; here are some examples of facts f1-f4:

```
% 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 lists are used to represent set values.

The state of the system is represented by all the facts true in that state. Recall that only the precise forms $A = d$ 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 I and Table II 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 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 an XTT is represented by one *flat* rule-base. Every row in a table corresponds to a single rule. The rule-base is separated from the inference engine code. All tables have unique identifiers (numbers), and rules are assigned unique numbers too. For example, an excerpt of rule-base for the Thermostat example presented in section V-C 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. , and `yes,no` stand for attribute value *no/during business hours*.

### A. Prolog Inference Engine

In order to interpret XTT rules there is a need for a *meta-interpreter*. The inference process is performed by a meta-interpreter.

As a proof-of-concept an XTT meta-interpreter engine has been developed and described in detail in [14]. A code-excerpt from the PROLOG inference engine for proving logical satisfaction in a limited version of the granular attributive logic is presented below.

```
run(Table,Rule) :-
   mode(backtrack,no),
   rule(Table,RuleInTable,LP,LR,LA,LD,
        NTable,NRule),
   ok_rule(Rule,RuleInTable),
   nonvar(NTable),
   satisfied(LP),
   remove(LR),
   add(LA),
   out(LD),
   write('*** Fired rule: '),
   write(Table), write('/'),
   write(RuleInTable),write(' *** '),nl,!,
   run(NTable,NRule).

run(Table,Rule) :-
   mode(backtrack,no),
   rule(Table,RuleInTable,LP,LR,LA,LD,
        NTable,_),
   ok_rule(Rule,RuleInTable),
   var(NTable),
   satisfied(LP),
   remove(LR),
   add(LA),
   out(LD),
   write('*** Fired rule: '),
   write(Table), write('/'),
   write(RuleInTable),write(' *** '),nl,
   run(0,_).
```

The above code (an excerpt) show two (out of 16) cases of interpreting a rule in a table. The first clause concerns the case where next table and next rule are specified explicitly. The second clause stops the interpreter when the next table is not given.

## VII. CONCLUDING REMARKS

This paper presents extensions of Set Attributive Logic as presented in [3]. 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. The new ideas of the paper include attributive logic of high expressive power for development of powerful rule-based systems, new ideas of design of he structure of the rule-base, and new approach to inference control.

One observation is that the propagation mechanism can be made even more efficient through reasoning from false preconditions as well. In fact, tables analogous to Table I and Table II for propagation of truth values if a cell is found to be false can be defined (due to limited place it was not presented here).

Second, one can consider extraction of the formulae related to cells and building (through an off-line procedure) a Rete-type network [15] incorporating rules of Table I or a decision tree in a way analogous to the compilation of rules inside KHEPOS rule-based interpreter [16].

## REFERENCES


[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Prentice-Hall, 2003.

[2] J. Liebowitz, Ed., *The Handbook of Applied Expert Systems*. Boca Raton: CRC Press, 1998.

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

[4] P. Jackson, *Introduction to Expert Systems*, 3rd ed. Addison–Wesley, 1999, iSBN 0-201-87686-8.

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

[6] J. C. Giarratano and G. D. Riley, *Expert Systems*. Thomson, 2005.

[7] W. Klösgen and J. M. Żytkow, Eds., *Handbook of Data Mining and Knowledge Discovery*. New York: Oxford University Press, 2002.

[8] T. Morgan, *Business Rules and Information Systems. Aligning IT with Business Goals*. Boston, MA: Addison Wesley, 2002.

[9] G. J. Nalepa and A. Ligęza, "A graphical tabular model for rule-based logic programming and verification," *Systems Science*, vol. 31, no. 2, pp. 89–95, 2005.

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

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

[12] A. Ligęza and G. 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*, D. C. Wilson, G. C. J. Sutcliffe, and FLAIRS, Eds., Florida Artificial Intelligence Research Society. Menlo Park, California: AAAI Press, may 2007, pp. 530–535.

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

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

[15] C. Forgy, "Rete: A fast algorithm for the many patterns/many objects match problem," *Artif. Intell.*, vol. 19, no. 1, pp. 17–37, 1982.

[16] J.-P. Gouyon, "Kheops users's guide," Report of Laboratoire d'Automatique et d'Analyse des Systemes, Toulouse, France, Tech. Rep. 92503, 1994.