On ALSV Rules Formulation and Inference

Unknown Anonymous and Nameless Nobody
Some University
in a distant country
last street
authors@suniv.distant

Abstract

In the paper knowledge representation and inference issues for rule-based systems are discussed. The paper deals with improving logical calculus of Set Attributive Logic founding an expressive rule language XTT^2. Representation extensions are introduced, and practical inference rules provided. Examples of rule analysis for the language are given. Visual design tool *HQed* assuring rule quality is also presented.

Introduction

The expressiveness of knowledge representation is crucial for intelligent systems (van Harmelen, Lifschitz, and Porter 2007). Hence research on novel representation techniques still remains an active area. Decision rules and other logically equivalent formalisms, such as decision trees and tables play a crucial role in a number of knowledge-based systems (Giarratano and Riley 2005; Morgan 2002). One of the strongest aspects of rules, is their formal description, that allows for a formal analysis and refinement.

This paper continues the research discussed in (**?**). Basing on some advances in Set Attributive Logic (SAL) introduced in (**?**), recently extended through the Attribute Logic with Set Values (ALSV) in (**?**), the paper provides a proposal of decision rules formalization that has an extended expressive power. The ALSV calculus extends the classical Relational Database (Connolly, Begg, and Strechan 1999) knowledge representation capabilities with complex attribute description, allow for the representation and inference with non-atomic values of attributes (the so-called set-valued attributes). Based on ALSV an expressive rule language called XTT^2 (eXtended Tabular Trees) is proposed. It is a development of the XTT language discussed in (**?**).

It the paper the basic introduction to ALSV(FD) is given (ALSV over Finite Domains). The rule language formulation, along with state representation needed for the inference process is discussed.An algebraic text-based rule representation is presented, as well as the visual representation with XTT tables. Inference with this structured rulebase is discussed, with selected elements of the design of a hybrid inference engine is considered. The future work, including implementation issues as well as possible extensions of the rule language is provided.

Motivation

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. It constitutes the bases for construction of relational database tables, attributive decision tables and trees (Klösgen and Żytkow 2002), attributive rule-based systems (**?**) and is often applied to describe state of dynamic systems and autonomous agents.

However, it is symptomatic that while a number of new rule-based solutions, such as Jess or Drools, provide new high-level features, such as Java-integration, network services, etc., the rule representation and inference methods remain essentially the same. The rule languages found in these tools tend to be logically trivial, and conceptually simple. They mostly reuse some basic logic solutions, combined with new language features, mainly borrowed from Java, built on top of classic inference approaches, e.g. Rete.

While these systems integrate well with today business application stacks, they provide little or no improvement in the areas of formalized analysis, visual design, or gradual refinement. This gives motivation to tackle these problems by introducing novel knowledge representation tools.

Set Attributive Logic (SAL) Development

The very basic idea is that attributes can take *atomic* or *set* values. After (**?**) it is assumed that an *attribute* A_i is a function (or partial function) of the form $A_i{:}\, O \to D_i$. 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 $A_i{:}\, O \to 2^{D_i}$, 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 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*).

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).

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: $\mathbf{A}$ – 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. 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 $A_i{:}\, T \rightarrow D_i$ (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 legal atomic formulae of ALSV along with their semantics are presented in Tab. 1, 2 for simple and general attributes.

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.

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 SAL (**?**). If $V = \{d_1, d_2, \ldots, d_k\}$ then $A = V$ is equivalent to

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

Syntax	Description	Relation
$A_i = d$	the value is precisely defined	`eq`
$A_i \neq d$	shorthand for $A_i \in D_i \setminus \{d\}$.	`neq`
$A_i \in V_i$	any of the values $d \in V_i$ satisfies the formula	`in`
$A_i \notin V_i$	is a shorthand for $A_i \in D_i \setminus V_i$.	`notin`

Table 1: Simple attribute formulas syntax

Syntax	Description	Relation
$A_i = V_i$	equal to V_i (and nothing more)	`eq`
$A_i \neq V_i$	different from V_i (at at least one element)	`neq`
$A_i \subseteq V_i$	being a subset of V_i	`subset`
$A_i \supseteq V_i$	being a superset of V_i	`supset`
$A \sim V$	having a non-empty intersection with V_i or disjoint to V_i	`sim`
$A_i \not\sim V_i$	having an empty intersection with V_i (or disjoint to V_i)	`notsim`

Table 2: Generalised attribute formulas syntax

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.

Basic Inference Rules for ALSV(FD)

Since the presented language is an extension of the SAL (**?**), its simple and intuitive semantics is consistent with SAL and clears up some points of it. 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 Tab. 3.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. The inference rules for atomic formulae with generalized attributes is presented in Tab. 4.

In Table 3 and Table 4 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)

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

Table 3: Inference for atomic formulae, simple 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 4: Inference for atomic formulae, generalized attributes

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

Table 5: Inconsistency conditions for atomic formulae pairs

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 3 and Table 4 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 5 specifies the conditions for inconsistency.

The interpretation of the Table 5 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 5 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.

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 (**?**). 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 (**?**).

State Representation and Inference

When processing information, the current values of attributes form the state of the inference process. The values of attributes can, in general, be modified in the following three ways: 1) by an independent, external system, 2) by the inference process, and 3) as some clock-dependent functions. The first case concerns attributes which represent some process variables, which are to be incorporated in the inference process, but depend only of the environment and external systems. As such, the variables cannot be directly influenced by the XTT system. Examples of such variables may be the external temperature, the age of a client or the set of foreign languages known by a candidate. Values of such variables are obtained as a result of some measurement or observation process. They are assumed to be put into the inference system via a *blackboard* communication method; in fact they are written directly into the internal memory whenever their values are obtained or changed.

The second case concerns the values of attributes obtained at certain stage of reasoning as the result of the operations performed in RHS of XTT. The new values of the attributes can be: asserted to global memory (and hence stored and made available for any components of the system), or kept as values of internal process variables. The first solution is offered mostly for permanent changes; before asserting new values typically and appropriate retract operation is to be performed so as to keep a consistent state. In this way also the history (trajectory) of the system can be stored, provided that each value of an attribute is stored with a temporal index. The second solution is offered for value passing and calculations which do not require permanent storage. For example, if a calculated value is to be passed to some next XTT component and it is no longer used after, it is not necessary to store it in the global memory.

The State of the System

The current state of the system is considered as a complete set of values of all the attributes in use at a certain instant of time. The concept of the state is similar to the one in dynamic systems and state-machines. The representation of the state should satisfy the following requirements:

1. the specification is *internally consistent*,
2. the specification is *externally consistent*,
3. the specification is *complete*,
4. the specification is *deterministic*,
5. the specification is *concise*.

The first postulate says that the specification itself cannot be inconsistent at the syntactic level. For example, a simple attribute (one taking a single value) cannot take two different values at the same time. In general, assuming independence

of the attributes and no use of explicit negation, each value of an attribute should be specified once. The second postulate says, that only *true* knowledge (with respect to the external system) can be specified in state. In other words, facts that are syntactically correct but false cannot occur in the state formula. The third postulate says, that *all* the knowledge true at a certain instant of time should be represented within the state. The four postulate says that there can be no disjunctive knowledge specification within the state. Finally, the fifth postulate says that no unnecessary, dependent knowledge should be kept in the state. In databases and most of the knowledge bases this has a practical dimension: only true facts are represented explicitly.

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: $(A_1 = S_1) \wedge (A_2 = S_2) \wedge \ldots \wedge (A_n = S_n)$ 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.

In order to cover realistic cases some explicit notation for covering unspecified, unknown values is proposed; this is so to deal with the data containing the NULL values imported from a database. The first case refers to unspecified value of an attribute as a consequence of inappropriateness. A formula of the form $A = \perp$ means that the attribute A takes an empty set of values (no value at all) at the current instant of time (or forever) for the object under consideration. For example, the attribute `Maiden_Name` or `The_Year_of_Last_Pregnancy` for a man is not applicable and hence it takes no value for all men. The second case refers to a situation that the attribute may be applied to an object, but it takes no value. This will be denoted as $A = \emptyset$. For example, the formula `Phone_Number`$=\emptyset$ means that the considered person has no phone number. The third case is for covering the `NULL` values present in relational databases. A formula of the form $A =$ `NULL` means that attribute A takes an unspecified value.

Rule Firing

In order to fire a rule all the precondition facts defining its LHS must be true within the current state. The verification procedure consists in matching these fact against the state specification. A separate procedure concerns simple (single-valued) attributes, and a separate one is applied in case of complex attributes. The following tables provide a formal background for preconditions matching and rule-firing procedure: Tab. 6 defines when a precondition of the form $A \propto d$ is satisfied with respect to given state, and Tab. 7 defines the principles for matching precondition defined with set-valued attributes against the state formula.

Representation and Inference with Tables

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, forming a table. Every cell in the table corresponds to a ALSV(FD) formula. The visual table representation can be observed in the tool screenshot in Fig. 1.

Having a table with defined rules the execution mechanism searches for ones with satisfied preconditions. This satisfaction is verified in an algebraic mode, using the dependencies specified in the first row of Table 3 for simple attributes and the first row of Table 4 for the complex ones.

The rules having all the preconditions satisfied can be fired. 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 (**?**).

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 3, 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 4, 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).

Rule Analysis Examples

The advantage of tabular rule-based systems defined with ALSV(FD) is that rule analysis 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.

Subsumption: Consider two rules given by two rows of a table; the rules are of the form introduced previously. for being some $i1$ and $i2$. For simplicity we consider that the $RHS_{i1} = RHS_{i2}$ (Right Hand Side), the conclusions are

$\models$	$A = d_j$	$A \neq d_j$	$A \in V_j$	$A \notin V_j$	$A = \_$	$A = \top$
$A = d$	$d = d_j$	$d \neq d_j$	$d \in V_j$	$d \notin V_j$	*true*	*false*
$A = \bot$	*false*	*false*	*false*	*false*	*false*	*true*
$A = \emptyset$	*false*	*false*	*false*	*false*	*true*	*false*
$A =$ NULL	*false*	*false*	*false*	*false*	*false*	*false*

Table 6: Inference principles for firing rules, case of single-valued attributes

$\models$	$A = W$	$A \neq W$	$A \subseteq W$	$A \supseteq W$	$A \sim W$	$A \not\sim W$	$A = \_$	$A = \top$
$A = S$	$S = W$	$S \neq W$	$S \subseteq W$	$S \supseteq W$	$S \cap W \neq \emptyset$	$S \cap W = \emptyset$	*true*	*false*
$A = \bot$	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*true*
$A = \emptyset$	$W = \emptyset$	$W \neq \emptyset$	*true*	$W = \emptyset$	*false*	*true*	*true*	*false*
$A =$ NULL	*false*	*false*	*false*	*false*	*false*	*false*	*false*	*false*

Table 7: Inference principles for firing rules, case of general attributes

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 Tab. 3 and 4. 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.

Indeterminism and Inconsistency: 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 can be performed with help of Table 4. 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 $j \in \{1, 2, \ldots, n\}$. If 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.

Inference Engine Design

The presented rule and inference formalization issues are an important part of the XTT^2 inference engine design specification. XTT^2 is a redesign of the original XTT (**?**) using ALSV(FD) as the formal foundation. The new XTT Prolog engine includes the support for the ALSV(FD) logic, but is also integrated with the C/C++/Java runtime to provide a flexible runtime solution.

The XTT inference engine, or rule meta-interpreter, also known as *HeaRT* (HeKatE RunTime) goals are interpreting XTT logic encoded in the *Hekate Meta Representation* (HMR), analyze XTT rules in XTT tables, provide inference control by interpreting links, manage the attributes by the *Blackboard Architecture*, by synchronizing attribute values between the system and the environment, Extended functions, provide: on-line formal verification with refinement capabilities, a bridge to the HQed editor, OO bridge to Java in the MVC pattern (Model-View-Controller), (HeaRT= Model, Java=View), standalone logic server with TCP.

Several interpreter types are considered. These try to address issues such as how to control the inference (an inference control strategy) both at the level of a single table and the inter-tabular level. Further, if cycles are necessary, how many times is the interpreter to repeat a single run. Here only the simplest one is described. The *Single Pass* interpreter is the simplest scenario: the interpreter is executed externally by the user, system, etc. input attribute callbacks provide input facts the inference is run, output attribute values can be sent by callbacks.

Rule Algebraic Notation (HMR)

The interpreter accepts XTT^2 rules in the following textual algebraic representation, *Hekate Meta Representation* (HMR). It is to be generated by the design tool (e.g. HQed) and directly run by HeaRT. Table scheme (logically also rule prototype), and rule encoding are:

```
xttscheme:  [a1,...,an], [h1,...,hm].
xttrule:    [Table, Id]: [c1,...,cn]
       then [d1,...,dn], [Table, Id].
```

HMR provides a human readable XTT2 description.

XTT Communication Issues

The assumptions for system modeling with ALSV rules are: the system is modeled with the use of attributes (state variables!), the state is fully described with attribute values, rule firing can possibly change system state by changing attribute values, rules can execute actions external to the system (these actions do not change system state), and the state can be changed from the outside. So it could be summarized, that "communication" between the system and the world is conducted using attributes and their values.

In the proposed *Blackboard Architecture* attributes are considered shared resources. The XTT logic system (S) can access the attribute values in the tables. The values can be accessed from the outside of the system (environment(E)), and updated by both the environment and the system simultaneously. A simple locking approach is exercised. The access and update policy depends on the attribute class considered: input (S $\leftarrow$ E), output (S $\rightarrow$ E), internal (S $\rightarrow$ S), and communication (S $\leftrightarrow$ E). The actual value modification is provided by a middle layer of *attribute callbacks*, predicates run by the interpreter to pull or push attribute values.

In this approach the communication is a simple issue so does not require an in-depth discussion. However, the practical engine integration with other runtime environments on the *architectural* level is crucial (see Drools+JBoss).

XTT Design Tool Support

One of the main features of the XTT^2 method is the compact visual representation. From the designer point of view

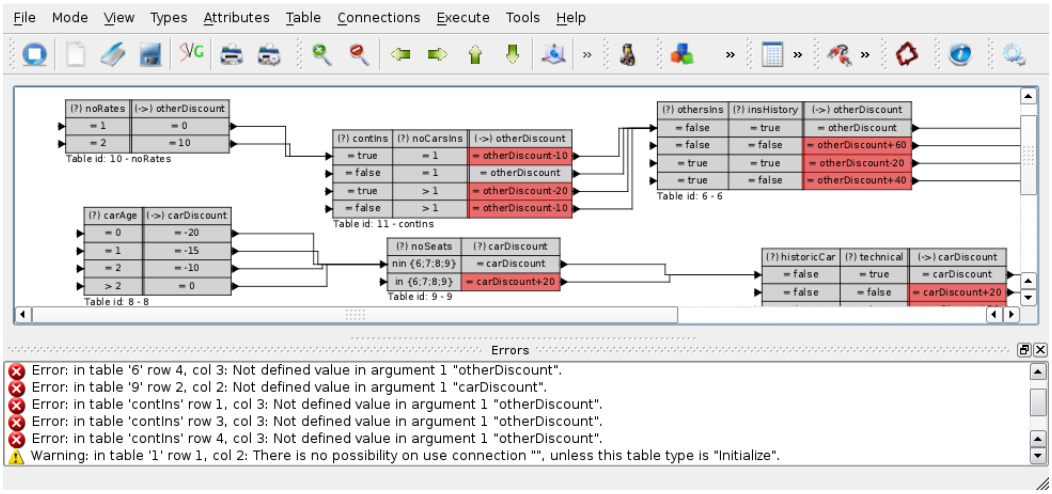

Figure 1: HQed editing session, the XTT rulebase structure with anomalies detected

it needs to be supported by a CASE tool in order to be effective. The *HQed* (Kaczor 2008) tool (Fig. 1) uses the rule prototypes generated in the conceptual design, and supports the actual visual process of the logical design of XTT2 tables. It is a cross-platform tool written in C++ and the Qt library Available under the the GNU GPL from `ai.ia.agh.edu.pl/wiki/hekate:hqed`.

One of the most important editor features is the support for XTT *rulebase quality assurance*, provided in several aspects: condition specification constraints, structured rulebase syntax, gradual model refinement, with partial simulation logical rule model quality. The first aspect is provided by number of editing features providing strict user data verification. Every attribute value entered into XTT cells (corresponding to the ALSV(FD) formulas) is checked against the attribute domain. On the other hand the user is hinted during the editing process, with feasible attribute values.

The rulebase syntax may be checked against anomalies, e.g. incomplete rule specification, malformed inference specification, including missing table links. The editor allows for gradual rules refinement, with an online checking of attribute domains, as well as simple table properties, such as inference related dead rules. In case of simple tables it is possible to emulate and visualize the inference process.

However, the main quality feature being developed is a plugin framework, that allows for integrating Prolog-based analysis plugins (being part of the inference engine) to check formal properties of the XTT rule base, including completeness, redundancy, or determinism. Here analysis is performed on the logical level, where the rows of the XTT tables are interpreted and analyzed as ALSV(FD) formulas.

The output from HQed is a rulebase encoded in HMR, that can be executed using a Prolog-based inference engine.

Conclusions and Future Work

This paper presents extensions of Set Attributive Logic (**?**). In the proposed logic both atomic and set values are allowed and various relational symbols are used to form atomic formulae. The proposed rule language provides a concise and elegant tool of significantly higher expressive power than in case of classical rule systems.

In the paper new inference rules specific for the introduced logic are examined. They constitute a challenge for efficient precondition matching, so algebraic solutions are proposed. The original contribution also includes enhanced state representation and design tool support. Built on the ALSV(FD) XTT^2 allows for a compact visual representation, with the rule inference formally described.

Future work includes practical inference engine implementation with the plugin framework, integrated on the architectural level with the Java runtime.

Warning Some self-citations have been removed, so they do not appear in the reference list.

References

Connolly, T.; Begg, C.; and Strechan, A. 1999. *Database Systems, A Practical Approach to Design, Implementation, and Management*. Addison-Wesley, 2nd edition.

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

Kaczor, K. 2008. Design and implementation of a unified rule base editor. Master's thesis, AGH Univerity of Science and Technology. Supervisor: G. J. Nalepa.

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

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

van Harmelen, F.; Lifschitz, V.; and Porter, B., eds. 2007. *Handbook of Knowledge Representation*. Elsevier Science.