Prolog-Based Analysis of Tabular Rule-Based Systems with the XTT Approach*

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

Abstract

This paper presents a new approach to the issue of assuring rule-based systems (RBS) correctness. The principal idea is that verification should be performed on-line, incrementally, during system design. It allows for early detection and handling of knowledge base anomalies and inconsistencies by incorporating a formal Prolog-based analysis of RBS in the design phase. A formal concept of a design tool (XTT) for specifying attributive RBS with a visual editor is outlined.

Introduction

Over thirty years rule-based systems (RBS) prove to constitute one of the most substantial technologies in the area of applied Artificial Intelligence (AI). Modern rule-based systems find applications ranging from medicine to finance and economy, going well beyond traditional rule-based programming paradigm. Some interesting recent applications of rule technology are the ones of business (the so-called *business rules* implementing the knowledge level processing in complex systems) and the *Semantic Web*. Both current research and applications go far beyond the traditional approach (Barr & Markov 2004).

Although rules constitute one of the simplest and most transparent programming paradigms, practical implementation of rule-based systems encounters serious problems. The main issues encountered concern *complete* specification of *non-redundant* and *consistent* set of rules (Liebowitz 1998; Ligęza 2006). This turns out to be a tedious task requiring significant effort of domain experts and knowledge engineers (Knauf 2005).

The approaches to verification and validation of RBS (Chang, Combs, & Stachowitz 1990; Liebowitz 1998; Spreeuwenberg & Gerrits 2002; Vanthienen, Mues, & G.Wets 1997; Vermesan & Coenen 1999) do not solve the problem entirely. Verification performed *after the system is designed* is both costly and late. Moreover, after introducing the corrections of detected errors, the verification cycle must be repeated. The problem consists in the possibility of *introducing new errors* through correction of the old ones.

*Research supported from a KBN Research Project ADDER No.: 4 T11C 035 24

This paper is dedicated to presentation of a new approach to the issue of assuring RBS correctness. The principal idea is that *verification should be performed on-line, incrementally, during the design*; moreover, error detection should lead the designer back through the design, towards error elimination.

Research towards amalgamating the design and verification stages has been undertaken several years ago; some first ideas date back to the so-called ψ-trees (Ligęza 1996) and *tab-trees* (Ligęza, Wojnicki, & Nalepa 2001). It resulted in elaboration of a new approach combining the expressive power of decision trees and attributive decision tables of non-atomic attribute values (Nalepa 2004). A more complete presentation is given in (Ligęza 2006).

The approach presented in this paper allows for an early detection and handling of knowledge base anomalies and inconsistencies by incorporating an on-line formal validation of RBS in the system design phase. Another important feature of the proposed approach consists in *visual* edition of rule-components and connections among them. A formal concept of a design tool for specifying attributive rule-based systems (called eXtended Tabular Trees, XTT for short) with a visual editor is outlined. These concepts have been implemented in a prototype CASE tool called MIRELLA DESIGNER (Nalepa 2004). A design and verification process based on the idea of using PROLOG plugins for analysis is also presented.

The paper is organized as follows: in Sect. 2 an idea of hierarchical design of RBS is outlined; it is centered around a knowledge representation and design method called XTT, described in Sect. 3. XTT has strong logical foundations that allow for definition of a formal approach to the analysis, discussed in Sect. 4. In order to efficiently analyze the XTT structure during the design, a formal transformation from XTT to PROLOG-based representation is introduced in Sect. 5. This enables formal verification with PROLOG procedures performed on-request during the design process.

Hierarchical Design of Rule-Based Systems

The proposed approach and the visual tool introduces strong structuring of the design process. Further, at any stage of partially designed system any knowledge component can be verified and corrected if necessary. In fact, the integration of the design and on-line verification is one of the crucial novel

ideas of the presented approach, constituting perhaps the first implementation of some early ideas initially put forward in (Ligęza 1996), and then followed by (Ligęza, Wojnicki, & Nalepa 2001). Simultaneously, this is a top-down approach, which allows for incorporating hierarchical design. Using XTT as a core, in (Nalepa 2004) an integrated design process, covering the following three phases has been presented. The *Conceptual design*, in which system attributes and their functional relationships are identified; the *logical design with on-line verification*, during which the system structure is represented as an XTT hierarchy, which can be instantly analyzed, verified (and corrected, if necessary) and even optimized on-line using a PROLOG-based framework; and the *physical design*, in which a preliminary PROLOG-based *implementation* is carried out.

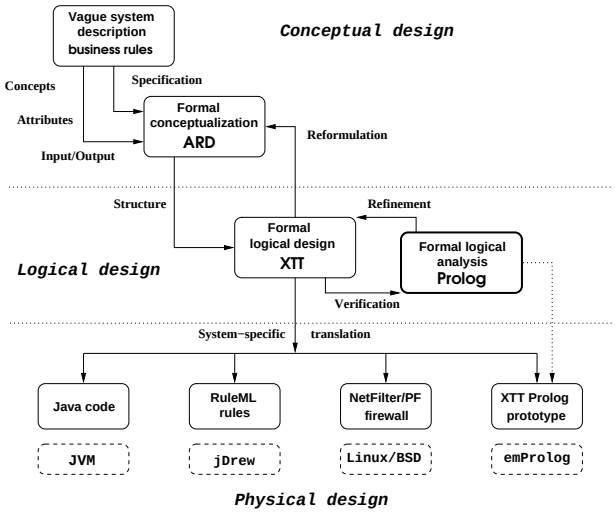

Figure 1: Hierarchical Design, Analysis & Implementation Process with ARD (Attribute Relationship Diagrams), XTT (eXtended Tabular Trees) and PROLOG

One of the most important features of this approach is the *separation of the logical and physical design*, which also allows for a transparent, *hierarchical* design process. The hierarchical conceptual model is mapped to a modular logical structure. The approach addresses three main problems: a visual representation, a functional dependency and logical structure, and a machine readable representation with automated code generation.

In the XTT-based approach the verification can be performed on-line, as an internal part of the design process. At any design stage any XTT component (extended table) can be analyzed. The analysis of a selected property is performed by external PROLOG-based plugins. The results of the analysis can be instantly used to improve the design of the given XTT table. In the current version of the MIRELLA DESIGNER system (Nalepa 2004) the verification modules operate taking as the input the state of the knowledge-base and return the diagnosis in the form of an immediate report.

Foundations of the XTT Approach

Numerous rule-based systems use simple knowledge representation logic based on attributes. Unfortunately, most of the systems allows 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 approach an *extended attributive language* is used. In fact we use *SAL*, the *Set Attributive Language*, as described in (Ligęza 2006). In SAL the atomic formulae are of two basic forms, i.e. $A(o) = t$ and $A(o) \in t$, where t is an arbitrary set of values (a subset of the domain of attribute A[1]). For intuition, $A(o) = t$ allows to say that attribute A takes *all* the values of t for object o; in fact, A is a *generalized attribute* taking set values (Ligęza 2006). Expression $A(o) \in t$ says that attribute A takes *some* value(s) of t (at least one) for object o. Facts of the form $A(o) = t$ are mostly used for efficient specification of the fact-base and in conclusion part of rules while facts such as $A(o) \in t$ are used in preconditions part of rules. Further details on attributive languages can be found in (Ligęza 2006).

Rule Format

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,[2] *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

[1] Formally t can be an infinite set; (both discrete and continuous) however, in practical applications it is often assumed to be a finite discrete one.

[2] In the assumed language explicit negation is not used; only the positive knowledge is represented in the knowledge base. Hence, facts that are no longer true are removed (retracted) from the fact base. This follows the well-known model of PROLOG.

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 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) takes the form presented in Tab. 1.

Table 1: The basic form of an XAT

Info		Prec	Retract	Assert	Decision	Ctrl	
I	Ctx	$A_1 \ldots A_n$	$B_1 \ldots B_b$	$C_1 \ldots C_c$	$H_1 \ldots H_h$	N	E
1	ψ	$t_{11} \ldots t_{1n}$	$b_{i1} \ldots b_{1b}$	$c_{11} \ldots c_{1c}$	$h_{11} \ldots h_{1h}$	g_1	e_1
i	ψ	$t_{i1} \ldots t_{in}$	$b_{i1} \ldots b_{ib}$	$c_{i1} \ldots c_{ic}$	$h_{i1} \ldots h_{ih}$	g_i	e_i
k	ψ	$t_{k1} \ldots t_{kn}$	$b_{k1} \ldots b_{kb}$	$c_{ik} \ldots c_{kc}$	$h_{k1} \ldots h_{kh}$	g_k	e_k

Each rule is represented by one row. In the first column there is the rule identifier, the *Ctx* is the context common for all the rules, $A_1 \ldots A_n$ are the precondition attributes, and $B_1 \ldots B_b$ are the retract, $C_1 \ldots C_c$ are the assert, and $H_1 \ldots H_h$ are the conclusion-defining ones. The *Ctrl* part defines the next rule to be executed (if present) or the else rule in case of failure. Hence, the table can specify both declarative knowledge (rules) and control knowledge (the *Ctrl* column).

The idea of XT has been successfully applied to a number of example RBS, see (Nalepa 2004; Ligęza 2006). For brevity, in order to illustrate this idea a single XT table taken from the Thermostat system of (Negnevitsky 2002) is presented in Tab. 2. The detailed discussion of the whole system containing 18 rules, composed if four XT tables can be found in (Ligęza 2006).

Table 2: XT for rules 3-6. *Context 2:* $aTD \in \{wd, wk\}$.

Inf.	Prec		Retract	Assert	Dec.	Ctrl	
I	aTD	aTM	aOP	aOP	H	N	E
3	wd	[9, 17]	–	dbh		3.7	2.4
4	wd	[0, 8]	–	$ndbh$		3.7	2.5
5	wd	[18, 24]	–	$ndbh$		3.7	2.6
6	wk	–	–	$ndbh$		3.7	2.3

The tables in the original system incorporate the following attributes: aDD – day, aTD – today, aTM – time, aOP – operation, aMO – month, aSE – season, $aTHS$ – thermostat_setting. The following attributes values were used: wd= 'workday', wk= 'weekend' dbh= 'during business hours', $ndbh$= 'not during business hours'. For intuition, the first row of Table 2 represents the rule: **If** today is a work-day **and** the time is between 9:00 and 17:00 **Then** the operation is 'during business hours' (*dbh*). The other three rules define when the operation is 'not during business hours' (*ndbh*).

Based on the idea of the XT, the *eXtended Tabular Trees (XTT)* knowledge representation has been developed (Nalepa 2004). An example of system design with XTT is presented in Figure 2.

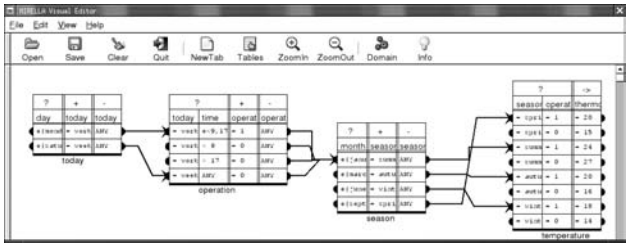

Figure 2: An example of XTT structure

An important feature of XTT is the fact that, besides visual representation, it offers a well-defined, *logical representation* which may be formally analyzed (Nalepa 2004; Ligęza 2006). On the *machine readable code level* XTT can be represented using an XML-based language *XTTML* (*XTT Markup Language*) suitable for import and export operations.

Rule Inference Control

Having the specific rule format, the examination and possible firing of rules defined by XTT hierarchy is performed by the forward-chaining rule interpreter engine. The interpretation begins with the *root table* in the XTT hierarchy. The first row (rule) in the current table is selected, and the row is processed, which includes: checking if the preconditions of the rule are satisfied; retracting all the facts that undergo the specification of the facts given in the *retract* part; asserting all the facts given in the *assert* part; and executing the actions defined in the conclusion part. Then the process continues by going to the table and rule specified with the *next* part if present, if not, the next row in the table is selected and processed; if this was the last row then backtrack to the previous table, or halt the inference process (the system behavior in this step is determined depending on the selected inference mode). In this way it is possible to work out several solutions for certain problems if necessary.

Formal Analysis of XTT Components

Within the proposed XTT approach a formal analysis and verification of selected theoretical properties can be performed. Three key properties have been considered in (Nalepa 2004; Ligęza 2006): *redundancy* – subsumption of rules, *indeterminism* – overlapping rules, and *completeness* – missing rules. Additionally, the XT components can be checked if they are minimal and reduction possibilities are suggested. Reduction of an XT component is performed by gluing two (or more) rules having identical conclusions.

In this approach the verification can be performed on-line, as an internal part of the design process. In the current version of MIRELLA, the designed system is verified against the following anomalies: subsumption, indeterminism and incompleteness. In fact, these issues are generic and cover a number of more specific problems, such as rule equivalence, inconsistent rules, etc. Moreover, reduction to minimal form through gluing of table rows with *backward dual resolution* (Ligęza 2006) is supported.

In case of rule specification with tabular systems, the analysis of subsumption is performed as follows. Consider two rules, r and r' given below (with simplified XTT scheme):

$rule$	A_1	A_2	...	A_j	...	A_n	H
r	t_1	t_2	...	t_j	...	t_n	h
r'	t'_1	t'_2	...	t'_j	...	t'_n	h'

The condition for subsumption in case of tabular rule format takes the algebraic form $t'_j \subseteq t_j$, for $j = 1, 2, \ldots, n$ and $h' \subseteq h$. If it holds, then rule r' can be eliminated leaving the more general rule:

$rule$	A_1	A_2	...	A_j	...	A_n	H
r	t_1	t_2	...	t_j	...	t_n	h

The indeterminism analysis is also almost straightforward; in order to have two rules applicable in the same state, their preconditions must have a non-empty intersection. In case of tabular systems this can be expressed as follows. For any attribute A_j there is an atom of the form $A_j = t_j$ in r and $A_j = t'_j$ in r', $i = 1, 2, \ldots, n$. Now, one has to find the intersection of t_j and t'_j – if at least one of them is empty (e.g. two different values; more generally $t_{1,j} \cap t_{2,j} = \emptyset$), then the preconditions are disjoint and thus the rules are deterministic. The check is performed for any pair of rules.

Reduction of XT is performed through gluing rules having identical conclusion part. Several rules can be glued to a single, equivalent rule according to the following scheme:

$rule$	A_1	A_2	...	A_j	...	A_n	H
r^1	t_1	t_2	...	t_{1j}	...	t_n	h
$\vdots$	$\vdots$	$\vdots$		$\vdots$		$\vdots$	$\vdots$
r^k	t_1	t_2	...	t_{kj}	...	t_n	h
$rule$	A_1	A_2	...	A_j	...	A_n	H
r	t_1	t_2	...	T	...	t_n	h

provided that $t_{1j} \cup t_{2j} \cup \ldots \cup t_{kj} = T$. If T is equal to the complete domain, then $T = \_$. Of course, rules $r^1, r^2, \ldots, r^k$ are just some selected rows of the original table containing all rules. The logical foundation for reduction are covered in (Ligęza 2006). In the example system, rules 4 and 5 of Table 2 can be glued, provided that the time specification can be expressed with non-convex intervals (i.e. [00:00-08:00]∪[18:00-24:00]).

Finally, completeness verification can be viewed as a two-stage procedure. First some maximal reduction is performed on the precondition part of a selected table. In the ideal case an empty table (full logical completeness) is confirmed (any set of input values will be served). In the other case one has to check which input specifications are not covered. Here, thanks to allowing for non-atomic values of attributes it is not necessary to go through the list of all possible atomic combinations, i.e. performing the so-called *exhaustive enumeration check*, i.e. analysis of all the elements of the Cartesian product of the precondition attribute domains. In the proposed approach the attribute domains can be divided into subsets (granularized) corresponding to the values occurring in the table; hence the check is performed in a more abstract level and with increased efficiency. Uncovered input specifications define the potentially missing rule preconditions. The system is complete in the sense that there are no admissible (correct) inputs which are uncovered.

Prolog-based XTT Analysis

Prolog-based XTT Representation

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 an XTT model several issues have to be solved: a *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, the knowledge base has to be *separated* from the inference engine, and an inference control mechanism has to be implemented.

Every XTT *cell* corresponds to a certain *fact* in the rule base. A 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:

1. Atomic values, e.g $A(O) = V$, are represented by `f(aA,atomic,V)` term.
2. Negated atomic values, e.g $A(O) \neq V$, are represented by `f(aA,natomic,V)` term.
3. Non-atomic numerical values, such as $A(O) \in < x, y >$, are represented by `f(aA,interval,i(x,y))` term.
4. 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.
5. 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 $Set_i = \{monday, tuesday\}$.
6. 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 $Set_i = \{monday, tuesday\}$.

Now, 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.

Using the Thermostat example (Tab. 2) 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.

Prolog Inference Engine

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). It is a *forward chaining* interpreter, where the inference control is performed using control statements encoded in rules, i.e. `next-table,next-rule`. The *Halt rule* is defined as a table/rule numbered `0,0` – it stops the inference process. *Backtracking* is possible using an appropriate interpreter mode. The engine is implemented with the main `run/2` predicate, and several auxiliary predicates: `satisfied/1` checks whether the list of facts is valid, `fails/1` is the opposite to the above, `valid/1` checks whether the fact is present in rule base, or can be proved valid, `remove/1`/`add/1` removes/adds the fact from the rule-base, `out/1` outputs the decision.

There are several clauses defining the `run/2` predicate, which constitutes the kernel of the meta-interpreter. Below simplified but illustrative code excerpts for the regular (with no backtracking) operation mode are shown.

```
run(0,_) :- write('*** HALTED: run(0,_) *** '), nl,!.
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,_).
run(_,_) :- mode(backtrack,no), run(0,_).
```

The engine has a simple user shell providing access to its main functions and can be easily extended if needed (Nalepa 2004).

Analysis and Verification Framework

During the logical design of rule-bases incremental synthesis is supported by the on-line, PROLOG-based system analysis and verification framework. It allows for the implementation of different external verification and analysis modules. The framework is integrated with the XTT inference engine.

The external analysis, verification and optimization modules are implemented in PROLOG. They have direct access to the system knowledge base. Each module reads the XTT rule-base and performs the analysis of the given property. It then produces a report. The report can be optionally visualized in the MIRELLA DESIGNER. Since the modules have the access to the PROLOG-based XTT system description, it is also possible to implement dynamic rule correction algorithms.

The general algorithm for checking subsumption is as follows: for every two different rules in a table: 1) check whether the precondition part of the first rule is weaker (more general) than the precondition of the second rule, if so, 2) check whether the conclusion part (including assert, retract, decision parts) of the first rule is stronger (more specific) than the conclusion of the second rule; if so, 3) the second rule is subsumed. A simplified excerpt of the main part of the plugin PROLOG code follows:

```
vsu(T):-
   rule(T,N1,P1,R1,A1,D1,_,_), rule(T,N2,P2,R2,A2,D2,_,_),
   N1 \= N2, subsumes(P1,P2), covers(D1,D2),
   write('*** Rule: '),
   write(T),write('.'),write(N1),write(' subsumes: '),
   write(T),write('.'),write(N2), nl, fail.
vsu(T):-
   write('No more subsumption in table '), write(T), nl.
```

The plugin uses predicates `subsumes/2` and `covers/2`, which are used to compare two lists of facts (whether they are weaker or stronger). The algorithm is also able to check the assert and retract lists of the rules. It is assumed that a rule subsumes another rule if it asserts/retracts more facts than the subsumed rule.

Related Work

The research on verification and validation of rule-based systems has a long tradition.[3] Numerous research have been undertaken in the domain of verification of rule-based systems. Some best known results are recapitulated in a comprehensive book edited by J. Liebowitz (Liebowitz 1998). A recent book focused on verification and validation of knowledge-based systems is (Vermesan & Coenen 1999). A number of tools is listed in (Ligęza 2006).

Using PROLOG for the verification of rule-based systems was also proposed by C. L. Chang et al. in (Chang, Combs, & Stachowitz 1990). Their tool, the Expert Systems Validation Associate (EVA) was a validation system developed at the Lockheed Artificial Intelligence Center in the late 80. It consisted of a set of generic tools to validate any knowledge-based system written in any expert system shell

[3]For almost complete bibliography see: `www.csd.abdn.ac.uk/~apreece/Research/vvbiblio.html`, last update in 1995; for the V&V tools see also `www.csd.abdn.ac.uk/~apreece/Research/vvtools.html`

such as CLIPS, OPS5, and other. The system offered several tools, such as an extended structure checker, extended logic checker, semantics checker, omission checker, rule refiner, control checker, and behavior verifier. The main conceptual differences to our work is that the analysis with EVA was performed *after* the design stage, while our system support on-line verification integrated with visual design. Moreover, the logical background, and hence the class of languages is well defined (for details see (Ligęza 2006)).

An interesting work oriented towards simple practical application is the one of J. Vanthienen (Vanthienen, Mues, & G.Wets 1997). His tool, the PROLOGA system allows for knowledge specification with hierarchical decision tables. It enables verification of several theoretical characteristics, such as subsumption, completeness, conflict, etc. The main limitation of the system is that it supports propositional language for knowledge representation only. XTT is of incomparably higher expressive power and provides inference control mechanisms and dynamic knowledge modification.

A classical work on verifying logically specified rule-based systems is the one of Preece (Preece, Bell, & Suen 1992); a more recent paper concerns a method for structure-based testing of rule-based systems (Preece *et al.* 1998). Some interesting recent report on the VALENS system are presented in (Gerrits & Spreeuwenberg 2000); it is one of rare tools incorporating verification capability; however, it follows the classical approach where the verification is performed off-line, for a completed knowledge base. No details on the language and its expressive power nor about the technical aspects of verification were reported in (Gerrits & Spreeuwenberg 2000).

Concluding Remarks

The paper presents an outline of a new approach to design and verification of rule-based systems. It is argued that a reasonable solution should consist in an integrated design and verification procedure allowing for on-line verification of a partially designed system. A new idea for structural knowledge representation, the eXtended Tabular Trees (XTT), is incorporated.

The presented concept of tabular systems (XTT) seems to provide a new quality in knowledge representation. Simultaneously, it constitutes a step on the way to the *algebraization* of knowledge which seems important both for efficiency reasons and making the approach close to engineering practice. XTT offers the possibility of *visual knowledge representation* which seems very important for practical applications. It also incorporates the possibility of *hierarchical* representation and *development* of a RBS. Finally, it enables interleaving the *verification* and *design* stages, so that a possibly correct system is designed and developed.

References

Barr, V., and Markov, Z., eds. 2004. *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, Miami Beach, Florida, USA*. AAAI Press.

Chang, C. L.; Combs, J. B.; and Stachowitz, R. A. 1990. A report on the expert systems validation associate (eva). *Expert Systems with Applications* 1(3):217–230.

Gerrits, R., and Spreeuwenberg, S. 2000. Valens: A knowledge based tool to validate and verify an aion knowledge base. In *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000*, 731–738.

Knauf, R. 2005. The engineering of system refinement or: What ai learnt from software engineering. *Fachberichte Informatik Universität Koblenz-Landau* (28th German Conference on Artificial Intelligence (KI-2005), Workshop on Knowledge Engineering and Software Engineering, Koblenz, Germany, ISSN 1860-4471):59–70.

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

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

Ligęza, A. 1996. Logical support for design of rule-based systems. reliability and quality issues. In Rousset, M., ed., *ECAI-96 Workshop on Validation, Verification and Refinment of Knowledge-based Systems*, volume W2. Budapest: ECAI'96.

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

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

Negnevitsky, M. 2002. *Artificial Intelligence. A Guide to Intelligent Systems*. Addison-Wesley.

Preece, A. D.; Bell, R. D.; and Suen, C. Y. 1992. Verifying knowledge-based systems using the cover tool. In *IFIP Congress (3)*, 231–237.

Preece, A. D.; Grossner, C.; Chander, P. G.; and Radhakrishnan, T. 1998. Structure-based validation of rule-based systems. *Data Knowl. Eng.* 26(2):161–189.

Spreeuwenberg, S., and Gerrits, R. 2002. Requirements for successful verification in practice. In *Proceedings of 15th International Florida Artificial Intelligence Research Society Conference 2002 Society (FLAIRS-2002), Pensacola, FL, USA, May 14-16, 2002*, 221–225. Menlo Park, CA: AAAI Press.

Vanthienen, J.; Mues, C.; and G.Wets. 1997. Inter-tabular verification in an interactive environment. In J. Vanthienen, F. H., ed., *EUROVAV-97, 4th European Symposium on the Validation and Verification of Knowledge Based Systems*, volume II. Leuven, Belgium: Katholieke Universiteit Leuven. 155–165.

Vermesan, A., and Coenen, F., eds. 1999. *Validation and Verification of Knowledge Based Systems. Theory, Tools and Practice*. Kluwer Academic Publisher.