Running head: RULES VERIFICATION

Logical Representation and Verification of Rules

Antoni Ligęza

Grzegorz J. Nalepa

Institute of Automatics, AGH University of Science and Technology, Kraków, Poland

Institute of Automatics, AGH University of Science and Technology, Kraków, Poland

ABSTRACT

In this chapter an introduction to verification of rules is presented. Logical models of rules, knowledge representation languages and inference rules are introduced in brief. Required characteristics of rule-based systems are analyzed and issues of formal verification techniques for rules are presented in details. The aim is to show how the quality of rules can be improved or kept at a relatively high level, even in environments with numerous active rules. This involves verification methods for rules discussing the expressive power issues of different rule representation logical languages. Moreover, verification checks for rules incorporating consistency, redundancy, completeness and minimal form are presented.

Logical Representation and Verification of Rules

INTRODUCTION AND MOTIVATION

A Knowledge-Based System (KBS) for Business Intelligence (BI) applications can be composed of several heterogeneous components, such as database, fact base, text base, domain ontology specification, multimedia components, etc. However, the core component with respect to efficient knowledge processing is the Inference Engine (IE) operating on Business Rule-Base (BRB). Both the rule-base and the inference mechanism are responsible for *efficient* production of *complete* and *correct* output, fitting the current needs and generated in the right time.

A rule is a basic component of each operational Knowledge Base (KB); in principle, a rule is a statement of the generic form:

IF < conditions > **THEN** < conclusion > .

The **IF** part defines the preconditions, i.e. conditions under which the rule can be fired. The **THEN** part defined conclusions, decision, actions or jus a new fact deduced from the knowledge base. A set of particular inference rules is referred to as a Rule-Base (RB) or – when equipped with inference control mechanism (the so called *inference engine*) – a Rule-Based System (RBS).

A Business Rule (BR) is *a compact statement about an aspect of business* (Morgan, 2002). In fact, a business rule is an instance of a rule oriented towards business domain application. Typically, such a rule covers a chunk of knowledge of certain organization. Initially, such statements are expressed in (restricted) natural language. Contemporary BI applications can contain hundreds and thousands of rules (Morgan, 2002) which constitute mainly man-designed and hand-encoded input to the system. As such, they are prone to different types of errors. Before applying and during evolution of the system such rule bases should undergo several stages of analysis and improvement; this include:

- refinement – a transformation from abstract, general, usually expressed in natural language form into a formal statement of rules in some knowledge-representation language of appropriate expressive power, syntax and semantics,
- verification – proving correctness of the set of rules in terms of some verifiable characteristics; in fact features such as consistency, completeness, and various features of correctness are checked with formal methods. Most of the knowledge engineering papers summarize this as answering the question: „Are we building the product right?”
- validation – checking if the set of rules provides correct answers to specific inputs. In other words, validation consist in assuring that the system is sound and fits the user requirements. Most of the knowledge engineering papers summarize this as answering the question: „Are we building the right product?”
- testing – means to undergo the system a number of runs on specially prepared data and comparing the obtained results with the correct (expected) ones. It is empirical investigation oriented towards evaluation of the rule set quality and discovering bugs.

- correction – is a process of localizing and removing bugs from the set of rules; rules can be modified or replaced by other rules. Certain unnecessary rules can be eliminated, and new rules can be added. Rules can be split to more detailed, specific rules and joined (glued) to form more general rules if necessary.
- improvement – even a correct rule base can be improved. This can be performed by various means: adding new rules, reduction of the rule set, tuning of rules, etc. The goal is to obtain better – in terms of specific quality measures – performance indicators.

In order to be incorporated in information systems business rules must be expressed in some formal language, with well defined syntax and semantics. The refinement of initial set of rules leads to some computer-oriented representation, usually expressed in some general or domain-specific knowledge representation language. On the other hand, disregarding syntactic sugar, the expressive power depends on the logical (and operational) level of the language, and typically spans from simple propositional languages to first (or even higher) order logic. This stage is normally performed by hand, possibly with support of some visual editors and graphical knowledge modeling languages (Ligęza, 2006).

Once formal representation is obtained, formal analysis methods can be applied to check if required characteristics are satisfied. This stage consists in *formal verification* of the knowledge base. It can be performed on-line, incrementally, during the design process. Some formal criteria to be satisfied by such rule bases are to be specified and made operational. Classical and current taxonomies, tools and techniques for Verification of RBS are presented in (Vermesan, 1998; Vermesan & Coenen, 1999). Validation issues are discussed in (Knauf, 2000).

Considering the formal rule representation, this chapter is exclusively devoted to verification issues of rule-based systems in general and BRB systems constituting a current focus of interest. Once the knowledge is specified (or during the modeling process) selected formal characteristics should be checked for satisfaction.

The mission of this chapter is to:

- identify the required characteristics of a RB or BRB/BI system in terms of software quality features,
- translate them into verifiable characteristics and providing a taxonomy of them,
- outline algorithmic approaches to development of appropriate procedures,
- summarize the state-of-the-art with respect to tools and techniques, and
- provide a discussion of open problems and future research directions.

The philosophy of this chapter is simple: starting from required top-level characteristics proceed down to evaluable features and provide knowledge on software tools and techniques useful in practice.

A TOP-DOWN APPROACH TO RULES QUALITY

Starting point: the required characteristics of BI software

A good point to start is to consider the ultimate goal which can be perceived as *achieving an appropriate quality of the knowledge and hence assuring the high quality output*. The quality of software systems is an important focus in todays applications. Although there is no general agreement with respect to a unique definition of software quality, the following features seem to be decisive as for assuring it:

- **Reliability** – it is the capacity to assure the required level of functional services during the specified period of work.
- **Efficiency** – it is evaluation of the level of functional services with respect to required resources.
- **Functional Capability** – it is the capacity of satisfying user requirements with respect to the desired functions to be performed by the system.
- **Portability** – it is the capability of work in different environments.
- **Usability** – it is the easiness of using the software.
- **Maintenability** – it says how easy the system is to maintain.

In case of BI applications and especially ones covering the rule-based components the above features give only a rough outline of what the quality knowledge-base should be like. For example, usability seems to be less important or even unimportant with respect to rule-based knowledge bases – it is user interface component which has decisive influence on that feature.

Requirements for rule-based systems vs. verifiable characteristics

In case of RB and BRB systems the overall quality seems to be influenced by the following characteristic features:

- **Reliability** – the system should work and provide the required services at the required level.
- **Safety** – the system should not only work, but work in a safe way; this depends on elimination of potential failures and menaces, and ability to operate in case of certain faults.
- **Efficiency** – it should work in the best possible way at the minimal requirements of resources, e.g. memory, CPU time, etc.
- **Optimality** – it should work in a best attainable way, in a given situation.

The above features are not completely independent from one another; usually, assuring safety stays in opposite to efficiency. Reliability, on the other hand, is close to safety.[i]

Some further required features include also (at least to certain degree) *functional capability*, *portability* and *maintenability*. However, in case of rule-based systems these features are of somewhat specific nature.

For example, functional capability follows directly from the rule code since in fact designing rule-based systems is very much like writing executable specifications.[ii] Achieving high functional capability is more the problem of the admitted knowledge specification language than the rules – if something can be expressed in the language, an appropriate rule can be added to the knowledge base.

Features such as portability and maintenability on their turn, are achieved in a direct way due to the declarative programming approach which is intrinsic to rule-based systems design. To certain degree they also depend on the knowledge specification language in use. Hence in fact, they are rather loosely related to the quality of the rule-base and the knowledge covered by it.

In the further part the interest will be focused on and around features such as *safety*, *reliability* and *efficiency* of rule bases and how these abstract, far-reaching aims can be translated into local, verifiable characteristics.

Required formal characteristics

The expressive power of knowledge representation languages makes the scope of potential applications combined with modularity of RBS a very general and readily applicable mechanism. However, despite a vast spread-out in working systems, their theoretical analysis seems to constitute still an open issue with respect to analysis, design methodologies and verification of theoretical properties. Assuring *reliability*, *safety* and *efficiency* of rule-based systems requires both theoretical insight and development of practical tools. The general qualitative properties are translated into a number of detailed characteristics defined in terms of logical conditions.

In fact, in order to assure safe and reliable performance, such systems should satisfy certain formal requirements, including completeness and consistency. To achieve a reasonable level of efficiency (quality of the knowledge-base) the set of rules must be designed in an appropriate way. Several theoretical properties of rule-based systems seem to be worth investigating, both to provide a deeper theoretical insight into the understanding of their capacities and assure their satisfactory performance, e.g. *reliability* and *safety*. Some most typical issues of theoretical verification include satisfaction of properties such as *consistency*, *completeness*, *determinism*, lack of *redundancy* or *subsumption*, etc. (see Andert, 1992; Lunardhi & Passino, 1995; Nazareth, 1989).

In the literature there has been a long discussion about classification and taxonomy of verifiable characteristics; some most influencing papers include (Nguyen et al., 1985; Suwa, Scott & Shortliffe, 1985; Chang, Combs & Stachowitz, 1990; Andert, 1992; Coenen, 2000; Lamb & Preece, 1996; Nazareth, 1989; Lunardhi & Passino, 1995; Vanthienen, Mues & G.Wets, 1997; Vermesan, 1998).

Some most general specification of such requirements typically refer to characteristics such as:

- **Redundancy** – is the knowledge specified in an efficient way, so that redundancy is avoided,

- **Inconsistency** – is the knowledge specified with the BRB consistent, where both internal, logical consistency and external one, referring to the consistency with the real world (model) can be considered,
- **Minimal representation** – is the specified knowledge in some minimal form, i.e. it cannot be replaced by another, more efficient and more concise representation,
- **Completeness** – is the knowledge complete, can the system always produce the output,
- **Determinism** – is the provided output unique and repeatable under the same input conditions,
- **Optimality** – is the knowledge optimal, i.e. sufficient for producing the best available solutions.

The described approach to assuring high quality is based on selection of verifiable characteristics (such as ones mentioned above) which are responsible for safety, reliability and efficiency. Then the initial qualitative requirements are translated to and expressed with requirements for satisfaction of such precisely defined characteristics. Each type of such characteristics is responsible for certain types of failures – hence, if one is able to check that the required characteristics are met, one may also be sure that the corresponding anomalies cannot occur. Below, taxonomies of such anomalies and characteristics are discussed.

KNOWLEDGE REPRESENTATION VERSUS VERIFICATION ISSUES

In this section a short review of accessible and applicable tools and techniques will be provided. The focus will be on generic techniques, which can be potentially widely-applied, and as such addressed to wider audience, and practical tools, especially ones accessible and checked for practical applications.

Knowledge representation languages and inference issues

The particular issues of verification are strongly dependent on the knowledge-representation language, i.e. the level of expressive power of it. Basically, one can distinguish the following types of languages:

- propositional logic languages,
- simple attributive logic languages (with atomic values of attributes),
- extended attributive logic languages (attributes can take set values),
- Datalog level languages (predicates yes, terms no),
- Prolog/First Order Logic (FOL) based languages,
- higher-order logical languages.

The main problem is that the higher the expressive power is, the more complicated verification becomes. In case of FOL and higher order logics some of the features become unmanageable w.r.t verification in realistic applications.

Static characteristics of declarative knowledge versus operational control issues

In practical RBS and BRB systems certain verification issues can be solved in an operational manner through providing efficient specification of inference control rules. A classical example is that indeterminism can be solved by introducing priorities among rules. Operational control can help avoiding problems such as indeterminism, circularity or even incompleteness. These issues are hardly covered by domain literature.

Verification versus knowledge representation form

Apart from specific level of knowledge representation language, the rules can take different forms, e.g.:

- various forms of decision lists,
- various forms of decision tables,
- various forms of decision trees,
- decision diagrams, including Ordered Binary Decision Diagrams,
- flat sets of rules,
- structured, hierarchical sets of rules,
- complex rules with operational and functional components,
- complex rules including specification of inference control,
- systems with rules and meta-rules.

Again, in practical BRB systems certain verification issues can be solved in an a manner depending on the specific knowledge representation form in use. As for now, these issues are hardly covered by domain literature.

A three-level model of business rules knowledge representation

The knowledge in BI systems is presented and processed at different level of abstraction. A basic model of business rules knowledge representation is a three-level one; it is cover the following levels:

- *knowledge presentation level* – this is the outermost level where the knowledge is presented to the user; typically it is in the form of restricted natural language,
- *logical level* – this is the level of symbolic knowledge representation used for knowledge analysis, verification and inference; typically this is one of logical languages (see below),
- *internal knowledge representation level* – this is the level at which the knowledge is encoded in the system; typically it can be implemented as Prolog facts or using XML/RuleML structures.

Consider an example of simple business rule as below (BRForum, 2005):

If the driver is male and is under the age of 25, then young driver.

This rule is presented at the knowledge presentation level; it is written in a natural language. At the logical level, this rule can be expressed as:

$$sex(Driver) = male \wedge age(Driver) < 25 \rightarrow classAge(Driver) = young.$$

The internal representation of the rule in Prolog can be as follows:

```
rule([sex(Driver,male),less_than(age(Driver),25)],[classAge(Driver,young)]).
```

In fact, verification is performed at the logical level (with use of the implemented below structures). The practical verification approaches must take into account the level and the type of the logical language in use.

LOGICAL LANGUAGES FOR RULES

A key issue for formal analysis, verification and validation is precise definition of the knowledge representation language. To do so both the syntax and semantics of the language should be specified.

Unfortunately, most of the rule-based systems provide some kind of application oriented language, taking expression patterns from classical procedural or object-oriented languages. In such a case the language itself is often loosely embedded in logic and it is hard to say what kind of logic it refers to. As a consequence no formal properties of the language are specified and it is hard to perform automated reasoning for analysis of rules and other knowledge components. Elements of procedural knowledge specification make the thing of formal verification even worse.

In this section we discuss the issue of logical language for knowledge specification. Three most popular languages of choice are presented in brief. The syntax of each language is outlined and some characteristic constructions used in rule-based systems are presented. In the next section some inference rules in form specific for the languages are presented.

The languages of choice are the following:

- propositional logic,
- attributive logic,
- Datalog order logic,
- Prolog (first-order predicate calculus) logic.

Below the syntax and characteristic features of these languages are discussed. Unlike as in most of the logical handbooks some practical aspects and application oriented details are put forward.

Propositional logic

Propositional logic is the simplest and perhaps the most popular one. Simultaneously its expressive power is relatively low and in numerous practical cases insufficient. The importance and role of propositional logic follows from the fact that most of the logical constructions and inference rules can be easily demonstrated with use of this simple logic.

The basic element of propositional logic language are *propositional variables* or *propositional symbols*, often denoted with $p, q, r, p_1, q_1, r_1, p_2, q_2, r_2, \ldots$, etc. In logic, they are referred to as

atomic formulae or *atoms* for short. They denote some statements. They can be assigned some meaning, e.g. $p = the\_temperature\_is\_below\_17\_degrees\_Centigrade$ or $q = the\_income\_is\_high$. They can also be assigned some truth values; in classical logic just *true* or *false*.

It is usually assumed (although most of the logical textbooks do not state it explicitly) that propositional symbols are *logically independent*. This means that they can be assigned any logical value independent from one another. Unfortunately, in most of practical applications this is not true. Depending on the assigned meaning various logical dependencies among propositional symbols can be imposed.

The simplest example concerns implicit negation. Let $p = the\_income\_is\_over\_1000\_EUR$ and let $q = the\_income\_is\_less\_than\_or\_equal\_to\_1000\_EUR$. In fact q is negation of p and vice versa. Hence $p \Rightarrow \neg q$ and $q \Rightarrow \neg p$ and obviously $p \wedge q$ is always false, while $p \vee q$ is always true.

The second example refers to the problems of finite domains (or a finite set of decision possibilities). Let us assume that there are three admissible colors of a car, e.g. red, yellow, and blue. Let $p = The\_car\_is\_red$, $q = The\_car\_is\_yellow$, and $r = The\_car\_is\_blue$. Obviously any of the atoms being true implies that the other two are false (at least provided that a single-color paint is allowed). Further, the conjunction of any two of the atoms is false while $p \vee q \vee r$ is always true.

A basic construction (in fact a formula) in rule-based system is a conjunction of atomic formulae or their negations. After (Ligęza, 2006) a formula

$$\phi = q_1 \wedge q_2 \wedge \ldots \wedge q_n \tag{1}$$

will be referred to as *a simple formula* or a *minterm*. In (1) all the q_i, $i = 1,2,\ldots,n$ are *literals* i.e. atoms or negated atoms. Atoms without negations are called *positive literals*, while the ones preceded with negation sign are *negative* ones.

If all the literals in a simple formula are positive, the formula is called a *simple positive formula* or a *positive minterm.*

Simple formulae can form more complex ones. A useful example is the so-called *Disjunctive Normal Form*, or a *DNF* for short. A formula is in DNF if and only if it is of the form:

$$\Phi = \phi_1 \vee \phi_2 \vee \ldots \vee \phi_n, \tag{2}$$

where each ϕ_i, $i = 1,2,\ldots,n$ is a simple formula.

Both simple formulae and formulae in DNF play the most important roles in rule-based systems and their analysis. The preconditions of rules are in most cases defined with simple formulae or formulae in DNF. Further, any more complex formulae can be transformed into an equivalent DNF form (Ligęza, 2006).

Attributive logic

Attributive logic is a kind of simple logic employing attributes to describe system properties. Attributes are a kind of variables, normally taking a single value at a time. The expressive power of attributive logic is higher than the one of propositional one; simultaneously, attributive logic in a natural way fits the requirements encountered when dealing with modelling many phenomena in domains of control, diagnosis, business, etc. In fact, the state of most of the systems can be described by providing a set of numerical or symbolic values of selected parameters. The importance and role of attributive logic follows from the fact that it is sufficient for most of the practical applications while the logical constructions stays intuitive and readable for domain experts.

The basic elements of any attributive logic language are *attributes* and their values for development of atomic formulae. The very basic idea is that attributes can take *atomic* values[iii]. After (Ligęza, 2006) it is assumed that an attribute A_i is a function (or partial function) of the form $A_i : O \rightarrow D_i$, where O is a set of objects and D_i is the domain of attribute A_i. The basic facts can have the following forms (for simplicity we omit the specification of the object):

$$A = d, \tag{3}$$

and

$$A \in t \tag{4}$$

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.

Let $t = \{d_1, d_2, \ldots, d_k\}$. The semantics of (4) is given by the following equivalence:

$$A \in t \equiv (A = d_1) \veebar (A = d_2) \veebar \ldots \veebar (A = d_k),$$

i.e attribute A takes exactly one of the k values forming the set; the symbol $\veebar$ stays for logical exclusive-or relation. As an example one can consider the specification of work days (denoted with sWD) given as

$$sWD = \{Monday, Tuesday, Wednesday, Thursday, Friday\}.$$

As an example one can consider the specification of a requirement that the day is a workday

$$Day \in \{Monday, Tuesday, Wednesday, Thursday, Friday\},$$

which can be shortened to

$$Day \in sWD.$$

As in the case of propositional logic, it is usually assumed (although most of the logical textbooks do not state it explicitly) that the attributes in use are *logically independent*. This means that they can be assigned any value (from their domains) independent from one another. Unfortunately, in most of practical applications this is not true. Depending on the assigned meaning various logical dependencies among attributes and their values can be imposed.

The simplest example here concerns hidden redundancy (following from a functional dependency). For example, consider atomic formulae $Date(today) = '2008/04/26'$ and $Day(today) = Friday$. In fact, it was Saturday on the 26-th of April, 2008. Hence these both formulae cannot be simultaneously true.

The second example refers to the problems of finite domains (or a finite set of decision possibilities) of the attributes and taking a single value by an attribute at a time. Let us assume that there are three admissible colors of a car, e.g. red, yellow, and blue. Let there be three atomic attributive formulae as follows: $Color(car) = red$, $Color(car) = yellow$, $Color(car) = blue$, Obviously any of the atoms being true implies that the other two are false (at least provided that a single-color paint is allowed). Further the conjunction of any two of the atoms is false while the disjunction of all of three of them is always true.

As in the case of propositional calculus a simple formula is a conjunction of atoms. Recall that in the simple attributive logic we have just two kinds of atomic formulae, i.e. $A = d$ and $A \in t$; for simplicity the specification of the object is omitted. The main difference is that in most applications only positive formulae are in use. Note that the negation of the fact that $A = d$ can be expressed as $A \in D'$, where $D' = D \setminus \{d\}$. Similarly, the negation of $A \in t$ can be expressed as $A \in t'$, where $t \cup t' = D$ and $t \cap t' = \varnothing$.

The other constructions and conventions for attributive logic are mostly the same as in case of propositional logic; so is the definition of DNF.

Datalog logic

One of the limitations of simple attributive logic consists in lack of the possibility to define n-ary relations. This can be done in Datalog logic using n-place predicative symbols.

Atomic formulae in Datalog logic are defined as follows. Let p be a relational (predicative) symbol of n arguments. Then

$$p(t_1, t_2, \ldots, t_n)$$

is an atomic formula of Datalog logic; t_i is either a constant symbol or a variable, $i = 1,2,\ldots,n$.

In classical Datalog there are two basic forms of formulae. These are *facts* and *clauses*. A fact is an atomic formula (followed by a full stop). A set of facts can be listed one after another, e.g.:

```
color(car1,red).
color(car2,yellow).
color(car3,blue).
```

Such a set of facts can be interpreted as a conjunction; in fact all the listed atomic formulae are true. Hence, simple (positive and negative) formulae, which are defined as in propositional logic, can be expressed as sets of facts. They can also be presented as a conjunction of facts. In Datalog, the conjunction symbol ($\wedge$) is replaced with comma.

Clauses are formulae used for inference; in fact they specify the inference rules. A clause is any formula such as

$$h :- p_1, p_2, \ldots, p_n.$$

A strictly logical form of a clause is:

$$p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow h,$$

or

$$\neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n \vee h$$

which are logically equivalent. Here p_i and h are positive atoms of Datalog. Such a clause is also called a *Horn clause* (one having at most one positive literal).

Prolog/First-order logic

The language of the highest expressive power is the one of First-Order Predicate Calculus (Ben-Ari, 2001; Ligęza, 2006). The main difference in comparison to Datalog is that definition of complex, structural object is allowed. Such objects are defined with *terms*.

Let f be an n-ary functional symbol, i.e. a name of an object having n arguments. A term is:

- any constant name c,
- any variable X,
- any expression $f(t_1, t_2, \ldots, t_n)$,

where $t_1, t_2, \ldots, t_n$ are terms; the definition is of inductive nature.

An atomic formula in Prolog logic is any expression of the form:

$$p(t_1, t_2, \ldots, t_n),$$

where p is an n-ary predicate symbol and $t_1, t_2, \ldots, t_n$ are terms.

Simple (positive and negative) formulae are defined as in propositional logic. They can be expressed as sets of facts (as in Datalog) or a simple conjunction of atoms (where the conjunction symbol ($\wedge$) is replaced with comma).

Clauses in Prolog are exactly the same as in Datalog. In fact, Datalog can be regarded as a *flat* version of Prolog (since there are no terms in Datalog).

Inference rules

Even in simple propositional logics there are numerous rules for logical inference (Ben-Ari, 2001; Ligęza, 2006). For the sake of verification issues we are interested in a limited subset of all possible rules. The discussion is limited to inference among atoms (literals) and simple formulae (minterms).

Let p, q and r denote atomic formulae of any of the above languages, and let ϕ, ψ, and φ denote some simple formulae. In practice we are interested in the following three inference issues:

- **logical implication**: having two simple formulae ϕ and ψ how to check if
$$\varphi \models \psi,$$
- **logical exclusion**: having two simple formulae ϕ and ψ how to check if
$$\phi \wedge \psi$$
is an unsatisfiable formula,
- **logical reduction**: having two simple formulae ϕ and ψ (logically: a disjunction of them) how to check if they can be replaced with a single, logically equivalent formula φ, i.e. in a rule of the form:
$$\frac{\phi, \psi}{\varphi}$$
so that $\phi \vee \psi$ is logically equivalent to φ.

In general, all the above problems can be solved with any of the inference rules and theorem proving approach. For example, this can be the famous resolution method (Chang & Lee, 1973; Gabbay, Hogger & Robinson, 1993) or any other approach to logical deduction.

For the sake of efficiency, however, the discussion will be restricted to limited but more efficient inference paradigms. Below logical solutions to the above three issues are presented at the level of first order logic.

The first problem is the one of logical implication among simple formulae. In case of first-order logic one must take into account the existence of variables. So substitutions must be applied. For checking if logical implication holds we have the subsumption condition (Ligęza, 2006); an appropriate rule can be defined as follows.

Assume ϕ and ψ are two simple formulae, and both of them are satisfiable; the problem is to check if $\varphi \models \psi$. Let $[\phi]$ denote the set of atoms of formula ϕ. If there exists a substitution θ such that

$$[\psi\theta] \subseteq [\phi] \tag{5}$$

then also

$$\varphi \models \psi. \tag{6}$$

Moreover, if ψ is not self-resolvable (it cannot be resolved by dual resolution with a copy of itself) then also (6) implies (5). The appropriate theorem can be found in (Ligęza, 2006). Note that all the variables are considered to be implicitly existentially quantified; this follows from the underlying assumption of considering formulae dual to classical resolution. In practice, this means that we are looking for substitutions satisfying the **IF** part of rules (Ligęza, 2006).

The case of logical exclusion – in case of first order logic – requires that the conjunction of $\phi \wedge \psi$ is unsatisfiable. Let us consider the case of non-resolvable formulae. This in fact means that there must exist a pair of complementary literals (an atom and its negation) belonging to $\phi \wedge \psi$.

Logical reduction can be performed with dual resolution rule (Ligęza, 2006). For intuition, consider first the case of two simple formulae $\phi \wedge p$ and $\psi \wedge \neg p$. The dual resolution rule takes the following form:

$$\frac{\phi \wedge p, \psi \wedge \neg p}{\phi \wedge \psi}.$$

In fact, there is logical equivalence between $(\phi \wedge p) \vee (\psi \wedge \neg p)$ and $\phi \wedge \psi$; the proof can be found in (Ligęza, 2006). The best effect of reduction can be achieved if $\phi = \psi$; in such a case we have:

$$\frac{\phi \wedge p, \phi \wedge \neg p}{\phi}.$$

Obviously, if ϕ is empty, the reduction leads to an empty formula (always true).

PRACTICAL INFERENCE RULES FOR ATOMIC FORMS

Consider the inference rules in case of practical implementation of the four logical languages presented above.

First consider the case of logical implication among simple formulae of propositional logic, attributive logic, Datalog simple formulae and full First-Order Logic simple formulae. Let ϕ and ψ denote satisfiable simple conjunctive formulae. We have to consider the following cases.

Logical implication among simple formulae

Propositional logic. The case of propositional logic is the most simple one. In case of simple conjunctive formulae there is:

$$\varphi \models \psi \ \ iff \ \ [\psi] \subseteq [\varphi], \tag{7}$$

so all the atoms of ψ must appear in ϕ; it is assumed that the atoms are logically independent. In case they are not, it is assumed that all the logical dependencies of the form $p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$ are given explicitly as a set R of the rules. In this case one can find the transitive closure of ϕ with respect to the set of given dependencies; let us denote it as $C_R(\phi)$. Then (7) can be replaced by

$$\varphi \models \psi \ \ iff \ \ [\psi] \subseteq [C_R(\varphi)]. \tag{8}$$

Obviously, it may be not necessary to find the full $C_R(\phi)$; any medium-stage set of atoms F such that all of them follows from ϕ satisfying $[\psi] \subseteq F$ will be enough.

Attributive logic. In this case the situation is a bit more complex. Assume there exists only single atomic formula referring to a specific attribute both in ϕ and in ψ; further the attributes are independent from one another. We have the following rule:

$$\varphi \models \psi \ \ iff \ \ \forall p \in [\psi] \ \exists q \in [\varphi] : q \models p, \tag{9}$$

so all the atoms of ψ must be logical consequences of some atoms of ϕ; moreover, normally it is assumed that the attributes are logically independent. In order to make (9) effective we have to know the rules for inference in attributive logic (Ligęza, 2006).

In the simple attributive logic with atomic values of attributes we have the following rules for inferring an atom from another one.

$$\frac{A \in s}{A \in t}, \tag{10}$$

provided that $s \subseteq t$; this rule is known as the so-called *upward consistency* (Ligęza, 2006). As a straightforward consequences of rule (10) we have that

$$\frac{A = d}{A \in t}, \tag{11}$$

provided that $d \in t$, and a trivial rule $A = d \vDash A \in \{d\}$.

Note however, that if there are two (or more) atoms referring to the same attribute, the situation becomes more complex. As an example consider the following rule:

$$\frac{A \in s, A \in t}{A \in s \cap t}, \tag{12}$$

known as the *intersection consistency rule* (Ligęza, 2006). Note that this rule can be generalized over the case of three and more constraints on the same attribute in the preconditions. In case of generalized attributes (ones taking a set of values at a time) the inference rules becomes even more complex; some current work on that can be found in [7].

Datalog logic. In Datalog logic in general case one has to refer to more complex inference rules. A generic approach may consists in using automated theorem proving approach, for example using the universal resolution rule or a dual resolution for convenience. Let us assume that dual resolution rule is the choice. A derivation with this rule will be denoted as $\varphi \vdash_{DR} \psi$. On the base of the theorems assuring completeness and validity of the rule we have:

$$\varphi \vDash \psi \;\; iff \;\; \varphi \vdash_{DR} \psi. \tag{13}$$

This means that in general case one has to apply the full automated theorem proving apparatus.

Note however, that in most of the practical examples the situation is not so bad; if ϕ is a non self-resolvable formula (it cannot be resolved with a copy of itself) then the right-hand condition $\phi \square_{DR} \psi$ can be replaced with a simple test for subsumption employing a unification. This is so for example in case of positive formulae – since no negation occurs, no resolution is possible. Hence, for such formulae we have:

$$\varphi \vDash \psi \;\; iff \;\; [\psi\theta] \subseteq [\varphi]. \tag{14}$$

i.e. there must exists a substitution θ such that all the atoms of $\psi\theta$ must appear in ϕ; it is assumed that the atoms are logically independent.

In case they are not, it is assumed that all the logical dependencies of the form $p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$ are given explicitly as a set R of the rules. In this case one can find the

transitive closure of ϕ with respect to the set of given dependencies; let us denote it as $C_R(\phi)$. Then (14) can be replaced by

$$\varphi \models \psi \ \ iff \ \ [\psi\theta] \subseteq [C_R(\varphi)]. \tag{15}$$

Obviously, it may be not necessary to find the full $C_R(\phi)$; any medium-stage set of atoms F such that all of them follows from ϕ satisfying $[\psi] \subseteq F$ will be enough.

Prolog/First-Order logic. The case of Prolog is very much the same as in Datalog. First-Order logic in general case one has to refer to more complex inference rules and the main difference consists unification – this time complex terms require full classical unification algorithm.

A straightforward generic approach may consists in using automated theorem proving approach, for example using the universal resolution rule or a dual resolution for convenience of direct proving that logical implication holds. Let us assume that dual resolution rule is the choice. A derivation with this rule will be denoted as $\varphi \vdash_{DR} \psi$. On the base of the theorems assuring completeness and validity of the rule we have:

$$\varphi \models \psi \ \ iff \ \ \varphi \vdash_{DR} \psi. \tag{16}$$

This means that in general case one has to apply the full automated theorem proving apparatus.

Note however, that in most of the practical examples the situation is not so worse; if ϕ is a non self-resolvable formula (it cannot be resolved with a copy of itself) then the right-hand condition $\varphi \vdash_{DR} \psi$ can be replaced with a simple test for subsumption employing complete unification algorithm. This is so for example in case of positive formulae – since no negation occurs, no resolution is possible. Hence, for such formulae we have:

$$\varphi \models \psi \ \ iff \ \ [\psi\theta] \subseteq [\varphi]. \tag{17}$$

so there must exists a substitution θ such that all the atoms of $\psi\theta$ must appear in ϕ; it is assumed that the atoms are logically independent.

In case they are not, it is assumed that all the logical dependencies of the form $p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$ are given explicitly as a set R of the rules. In this case one can find the transitive closure of ϕ with respect to the set of given dependencies; let us denote it as $C_R(\phi)$. Then (17) can be replaced by

$$\varphi \models \psi \ \ iff \ \ [\psi\theta] \subseteq [C_R(\varphi)]. \tag{18}$$

Obviously, it may be not necessary to find the full $C_R(\phi)$; any medium-stage set of atoms F such that all of them follows from ϕ satisfying $[\psi] \subseteq F$ will be enough.

Logical exclusion

Again, we shall be considering two simple formulae, namely ϕ and ψ. The issue of interest is the decide whether the joint formula:

$$\phi \wedge \psi$$

is inconsistent (unsatisfiable). Again, we have to discuss the four cases depending on the logical language in use.

Propositional logic In case of propositional logic the basic test for inconsistency is straightforward – one has to check if a pair of complementary literals exists or not. Let $Inc(\varphi)$ denote that formulae φ is inconsistent. We have the following rule:

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \exists p \in [\varphi], \exists q \in [\psi] : p = \neg q. \tag{19}$$

A bit more complex situation can occur iff the atomic formulae are no longer logically independent. In such a case (19) can be replaced with more general rule:

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \exists \phi : [\phi] \subseteq \left([\varphi] \cup [\psi]\right) \wedge Inc(\phi). \tag{20}$$

Again, if there exists a set of inference rules R, condition (20) can be extended to the case of transitive closure; we have:

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \exists \phi : [\phi] \subseteq C_R([\varphi] \cup [\psi]) \wedge Inc(\phi). \tag{21}$$

The last case is the most general one – a conjunction of two simple formulae is inconsistent iff there exist an unsatisfiable subformula of their transitive closure.

Attributive logic The case of attributive logic requires more complex rules taking into account the specific character of this language – there is no explicit negation, so a simple pair of inconsistent literals cannot be found. Instead of that one has to look for inconsistent pairs (or more complex simple formulae) of attributive logic atoms. The following simples rule for detecting inconsistency

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \exists \left(A = d_1\right) \in [\varphi], \exists \left(A = d_2\right) \in [\psi] : d_1 \neq d_2. \tag{22}$$

says, that a simple conjunctive formula is inconsistent iff a single atom is declared to take two different values.

The above rule can be generalized into the following form:

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \exists (A \in s) \in [\varphi], \exists (A \in t)[\psi] \wedge s \cap t = \varnothing. \tag{23}$$

This rule will be referred to as *empty intersection rule*.

Again a bit more complex situation can occur if the attributes are no longer logically independent. In such a case (23) can be replaced with more general rule:

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \exists \phi : [\phi] \subseteq [\varphi] \cup [\psi] : Inc(\phi). \tag{24}$$

Moreover, if there exists a set of inference rules R, condition (24) can be extended to the case of transitive closure; we have:

$$Inc(\phi \wedge \psi) \;\; iff \;\; \exists \varphi : [\varphi] \subseteq C_R([\phi] \cup [\psi]) \wedge Inc(\varphi). \tag{25}$$

The last case is the most general one – a conjunction of two simple formulae is inconsistent iff there exist an unsatisfiable subformula of their transitive closure.

Datalog logic In case of Datalog logic a general test for inconsistency can be performed with general inference rules, such as resolution or dual resolution. Assume derivation with resolution rule is denoted with $\vdash_R$. We have the following rule:

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \varphi \wedge \psi \vdash_R \bot, \tag{26}$$

where $\bot$ is an empty formula always false.

Note however, that in most of the practical examples the situation is not so bad; recall that $\phi \wedge \psi$ is a simple formula (one composed of single atomic formulae). This means, that if it is an inconsistent formula there must exist a pair of atoms resolving to the empty clause (a single-step inference process). Hence, he check for inconsistency becomes as follows:

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \exists p \in [\varphi], \exists q \in [\psi], \exists \theta : p\theta = \neg q\theta, \tag{27}$$

i.e. literals p and q can be resolved upon.

Note that in the case of positive formulae – since no negation occurs – no resolution is possible. Hence, generally, in such a case no inconsistency can occur. In case the atomic formulae are not logically independent the only possibility is to define all the inconsistent simple formulae (the so-called invariants or impossible situations). In such a case (27) can be replaced with more general rule:

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \exists \phi : [\phi] \subseteq \left([\varphi] \cup [\psi]\right) \wedge Inc(\phi). \tag{28}$$

Again, if there exists a set of inference rules R, condition (28) can be extended to the case of transitive closure; we have:

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \exists \phi : [\phi] \subseteq C_R([\varphi] \cup [\psi]) \wedge Inc(\phi). \tag{29}$$

The last case is the most general one – a conjunction of two simple formulae is inconsistent iff there exist an unsatisfiable subformula of their transitive closure.

Prolog/First-Order logic The case of full First-Order logic is generally similar to the Datalog case; the main difference is that a full unification algorithm for complex terms must be applied. In general, a test for inconsistency can be performed with general inference rules, such as resolution or dual resolution. Assume derivation with resolution rule is denoted with $\vdash_R$. We have the following rule:

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \varphi \wedge \psi \vdash_R \bot, \tag{30}$$

where $\bot$ is an empty formula always false.

Note however, that in most of the practical examples the situation is not so bad; recall that $\phi \wedge \psi$ is a simple formula (one composed of single atomic formulae). This means, that if it is an inconsistent formula there must exist a pair of atoms resolving to the empty clause (a single-step inference process). Hence, he check for inconsistency becomes as follows:

$$Inc(\phi \wedge \psi) \;\; iff \;\; \exists p \in [\phi], \exists q \in [\psi], \exists \theta : p\theta = \neg q\theta, \tag{31}$$

i.e. literals p and q can be resolved upon.

Note that in the case of positive formulae – since no negation occurs – no resolution is possible. Hence, generally, in such a case no inconsistency can occur. In case the atomic formulae are not logically independent the only possibility is to define all the inconsistent simple formulae (the so-called invariants or impossible situations). In such a case (31) can be replaced with more general rule:

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \exists \phi : [\phi] \subseteq \left([\varphi] \cup [\psi]\right) \wedge Inc(\phi). \tag{32}$$

Again, if there exists a set of inference rules R, condition (32) can be extended to the case of transitive closure; we have:

$$Inc(\varphi \wedge \psi) \;\; iff \;\; \exists \phi : [\phi] \subseteq C_R([\varphi] \cup [\psi]) \wedge Inc(\phi). \tag{33}$$

The last case is the most general one – a conjunction of two simple formulae is inconsistent iff there exist an unsatisfiable subformula of their transitive closure.

Logical reduction

The aim of this section is to provide practical rules for reduction of two (or more) simple formulae. Assume we consider two consistent simple formulae, different only with a single atom. The goal is the find a minimal formula equivalent to conjunction of the initial ones.

Propositional logic In case of propositional logic reduction of simple formulae is based on direct application of the dual resolution rule (Ligęza, 2006). The basic form preserving logical equivalence is as follows:

$$\frac{\varphi \wedge p, \varphi \wedge \neg p}{\varphi}. \tag{34}$$

Note, that in the above case the formulae may differ at a single atom only; in a pure logical form these must be complementary literals.

A more complex case cover the logically dependent atoms. assume $q \equiv p_1 \vee p_2$. A generalized rule takes the form:

$$\frac{\varphi \wedge p_1, \varphi \wedge p_2}{\varphi \wedge q}. \tag{35}$$

In the above, atoms p_1 and p_2 are glued and replaced by a single, logically equivalent atom q.

The above rule can be generalized towards gluing three or more formulae through gluing subformulae of them. The appropriate rule takes the form

$$\frac{\varphi \wedge \phi_1, \varphi \wedge \phi_2, \ldots, \varphi \wedge \phi_k}{\varphi \wedge \phi}, \tag{36}$$

provided that $\varphi \equiv \varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_k$.

Attributive logic In case of attributive logic reduction of simple formulae is also based on direct application of the dual resolution rule (Ligęza, 2006). This time, however, no complementary atoms in the sense of classical logic exist. The basic form preserving logical equivalence consists in gluing through putting singular values into a common set an it is as follows:

$$\frac{\varphi \wedge A = d_1, \varphi \wedge A = d_2}{\varphi \wedge A \in \{d_1, d_2\}}. \tag{37}$$

Note, that in the above case the formulae may differ at a single atom only. If A can take only the values d_1 or d_2 (in fact $\{d_1, d_2\}$ is the whole domain of A) then the result can be simplified into ϕ.

The above rule can be immediately generalized over the case of k values. It takes the form:

$$\frac{\varphi \wedge A = d_1, \varphi \wedge A = d_2, \ldots, \varphi \wedge A = d_k}{\varphi \wedge A \in \{d_1, d_2, \ldots, d_k\}}. \tag{38}$$

Again if $\{d_1, d_2, \ldots, d_k\}$ is the set of all the values A can take, the result is simplified to ϕ.

The most general form of the gluing rule is

$$\frac{\varphi \wedge A \in t_1, \varphi \wedge A \in t_2, \ldots, \varphi \wedge A \in t_k}{\varphi \wedge A \in t}, \tag{39}$$

provided that $t = t_1 \cup t_2 \cup \ldots \cup t_k$.

Datalog logic In case of Datalog logic reduction of simple formulae is based on direct application of the dual resolution rule (Ligęza, 2006). The basic form preserving logical equivalence is as follows:

$$\frac{\varphi \wedge p, \varphi \wedge \neg p}{\varphi}. \tag{40}$$

Note, that in the above case the formulae may differ at a single atom only; in a pure logical form these must be complementary literals. No substitution can be applied so as to preserve logical equivalence.

A more complex case cover the logically dependent atoms. Assume $q \equiv p_1 \vee p_2$. A generalized rule takes the form:

$$\frac{\varphi \wedge p_1, \varphi \wedge p_2}{\varphi \wedge q}. \tag{41}$$

In the above, atoms p_1 and p_2 are glued and replaced by a single, logically equivalent atom q.

The above rule can be generalized towards gluing three or more formulae through gluing subformulae of them. The appropriate rule takes the form

$$\frac{\phi \wedge \varphi_1, \phi \wedge \varphi_2, \ldots, \phi \wedge \varphi_k}{\phi \wedge \varphi}, \tag{42}$$

provided that $\varphi \equiv \varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_k$.

Prolog/First-Order logic The case of Prolog logic is very much the same as the one of Datalog. Since no unifying substitutions may be applied (because logical equivalence must be preserved) the rules (40), (41) and (42) can be applied in case of reduction of first-order simple formulae with terms in a direct way.

AN EXAMPLE OF A RULE-BASED SYSTEM

In order to provide some intuition as well as illustrate the ideas concerning verification of rule-based systems a simple but complete example of such system is presented below in some details. We shall refer to rules of this example throughout the rest of this chapter.

The example comes from (Negnevitsky, 2002). It was further explored in (Ligęza, 2006) and other papers. The advantage of the demonstration rule-base is that although it is a small-size system composed of 18 rules, it is a non-trivial one, it has potential possibility of practical application and has a clear interpretation and relatively simple internal structure. In fact, it can be used to explain almost all the issues concerning verification of rule-based systems.

The considered rule-based control system is aimed at setting the required temperature in a room, depending on the hour, day, season, etc. After (Negnevitsky, 2002) we shall refer to the example as THERMOSTAT. For convenience of the Reader the original knowledge base given in (Negnevitsky, 2002), pages 41--43, is listed below[iv].

```
/* THERMOSTAT: A DEMONSTRATION RULE-BASE */
Rule: 1
    if     the day is Monday
    or     the day is Tuesday
    or     the day is Wednesday
    or     the day is Thursday
    or     the day is Friday
    then   today is a workday
 Rule: 2
    if    the day is Saturday
    or    the day is Sunday
    then   today is the weekend
 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'
 Rule: 7
    if    the month is January
    or    the month is February
    or    the month is December
    then    the season is summer
 Rule: 8
    if    the month is March
    or    the month is April
    or    the month is May
    then    the season is autumn
```

```
Rule: 9
   if   the month is June
   or   the month is July
   or   the month is August
   then   the season is winter
Rule: 10
   if   the month is September
   or   the month is October
   or   the month is November
   then   the season is spring
Rule: 11
   if   the season is spring
   and   operation is 'during business hours'
   then  thermostat_setting is '20 degrees'
Rule: 12
   if   the season is spring
   and   operation is 'not during business hours'
   then  thermostat_setting is '15 degrees'
Rule: 13
   if   the season is summer
   and   operation is 'during business hours'
   then  thermostat_setting is '24 degrees'
Rule: 14
   if   the season is summer
   and   operation is 'not during business hours'
   then  thermostat_setting is '27 degrees'
Rule: 15
   if   the season is autumn
   and   operation is 'during business hours'
   then  thermostat_setting is '20 degrees'
Rule: 16
   if   the season is autumn
   and   operation is 'not during business hours'
   then  thermostat_setting is '16 degrees'
Rule: 17
   if   the season is winter
   and   operation is 'during business hours'
   then  thermostat_setting is '18 degrees'
Rule: 18
   if   the season is winter
   and   operation is 'not during business hours'
   then  thermostat_setting is '14 degrees'
```

Let us briefly analyze the rule-base. It becomes visible that the set of rules can be divided into four groups of rules, so that the rules put into a single group are structurally identical and work in the same context. Within each group the preconditions of the rules are constructed with use of the same linguistic variables (attributes) and the rules have the same output variable.

The first group is formed from rules 1 and 2. As the input they take the name of the day and the output is the decision whether it is a workday or the weekend.

The second group is composed of rules 3, 4, 5, and 6. As the input they take the type of the day (workday, weekend) and the precise hour. The output is the decision concerning whether it is 'during business hours' or not.

The third group is composed of rules 7,8,9, and 10. As the input they take the current month and as the output they provide decision what season it is.

Finally, the fourth group contains 8 last rules. On the base of the season (autumn, winter, spring, summer) and period of work ('during business hours', 'not during business hours') they provide specification of the set-point temperature (its exact value).

In order to define the rules in a more formula way let us define the attributes and their domains. We shall follow the notation introduced in (Ligęza, 2006).

After short analysis of the rules the following attributes can be found. The name of each attribute starts with 'a':

- *aDD* – day,
- *aTD* – today,
- *aTM* – time,
- *aOP* – operation,
- *aMO* – month,
- *aSE* – season,
- *aTHS* – thermostat_setting (the temperature).

Further the following domains and values of the attributes will be used; the name of each set starts with 's':

- *sWD* = {*Monday, Tuesday, Wednesday, Thursday, Friday*},
- *sWK* = {*Saturday, Sunday*},
- *sSUM* = {*January, February, March*},
- *sAUT* = {*March, April, May*},
- *sWIN* = {*June, July, August*},
- *sSPR* = {*September, October, November*},
- *sum* = `summer',
- *aut* = `autumn',
- *win* = `winter',
- *spr* = `spring',
- *wd* = `workday',
- *wk* = `weekend'
- *dbh* = `during business hours',
- *ndbh* = `not during business hours'.

In the above sWD defines the set of workdays and sWK – weekend days. $sSUM$, $sAUT$, $sWIN$, $sSPR$ are respectively the sets of months defining summer, autumn, winter and spring. The other symbols are explained above. Now the rules can be presented in a symbolic, concise way as follows.

The first group of rules determines whether it is a workday of the weekend:

$$\begin{aligned} &Rule\ 1: \quad aDD \in sWD \quad \rightarrow aTD = wd \\ &Rule\ 2: \quad aDD \in sWK \quad \rightarrow aTD = wk \end{aligned} \tag{43}$$

The second group of rules decides if the operation is during business hours or not:

$$\begin{aligned} &Rule\ 3: \quad aTD = wd \wedge aTM \in [9,17] \quad \rightarrow \quad aOP = dbh \\ &Rule\ 4: \quad aTD = wd \wedge aTM \in [0,9) \quad \rightarrow \quad aOP = ndbh \\ &Rule\ 5: \quad aTD = wd \wedge aTM \in (17,24] \quad \rightarrow \quad aOP = ndbh \\ &Rule\ 6: \quad aTD = wk \quad \rightarrow \quad aOP = ndbh \end{aligned} \tag{44}$$

The third group of rules defines the season:

$$\begin{aligned} &Rule\ 7: \quad aMO \in sSUM \quad \rightarrow \quad aSE = sum \\ &Rule\ 8: \quad aMO \in sAUT \quad \rightarrow \quad aSE = aut \\ &Rule\ 9: \quad aMO \in sWIN \quad \rightarrow \quad aSE = win \\ &Rule\ 10: \quad aMO \in sSPR \quad \rightarrow \quad aSE = spr \end{aligned} \tag{45}$$

Finally, the fourth group of rules providing the final output temperature is:

$$\begin{aligned} &Rule\ 11: \quad aSE = spr \wedge aOP = dbh \quad \rightarrow \quad aTHS = 20 \\ &Rule\ 12: \quad aSE = spr \wedge aOP = ndbh \quad \rightarrow \quad aTHS = 15 \\ &Rule\ 13: \quad aSE = sum \wedge aOP = dbh \quad \rightarrow \quad aTHS = 24 \\ &Rule\ 14: \quad aSE = sum \wedge aOP = ndbh \quad \rightarrow \quad aTHS = 27 \\ &Rule\ 15: \quad aSE = aut \wedge aOP = dbh \quad \rightarrow \quad aTHS = 20 \\ &Rule\ 16: \quad aSE = aut \wedge aOP = ndbh \quad \rightarrow \quad aTHS = 16 \\ &Rule\ 17: \quad aSE = win \wedge aOP = dbh \quad \rightarrow \quad aTHS = 18 \\ &Rule\ 18: \quad aSE = win \wedge aOP = ndbh \quad \rightarrow \quad aTHS = 14 \end{aligned} \tag{46}$$

Using the XTT (eXtended Tabular Trees) knowledge representation formalism presented in (Ligęza, 2006) one can represent these rules in a very concise and transparent way in a decision-like table form. The first two rules form the following table 1.

Table 1. The first group of rules for determining the type of day

Rule	*aDD*		*aTD*
1	*sWD*	→	*wd*

2	*sWK*	→	*wk*

The second group of rules forms the Table 2.

Table 2: The second group of rules for determining the type of operation

Rule	*aTD*	*aTM*		*aOP*
3	*wd*	[9,17]	→	*dbh*
4	*wd*	[0,9)	→	*ndbh*
5	*wd*	(17,24]	→	*ndbh*
6	*wk*	_	→	*ndbh*

The third group of rules forms the Table 3.

Table 3. The third group of rules for determining the current season

Rule	*aMO*		*aSE*
7	*sSUM*	→	*sum*
8	*sAUT*	→	*aut*
9	*sWIN*	→	*win*
10	*sSPR*	→	*spr*

Finally, the fourth group of rules can be specified with the following Table 4.

Table 4: The fourth group of rules for determining the current temperature set-point

Rule	*aSE*	*aOP*		*aTHS*
11	*spr*	*dbh*	→	20
12	*spr*	*ndbh*	→	15

13	*sum*	*dbh*	→	24
14	*sum*	*ndbh*	→	27
15	*aut*	*dbh*	→	20
16	*aut*	*ndbh*	→	16
17	*win*	*dbh*	→	18
18	*win*	*ndbh*	→	14

Note that knowledge representation with use of the XTT tables provides a unique opportunity to perceive and analyze the with algebraic tools rather then logical ones.

A TAXONOMY OF VERIFIABLE CHARACTERISTICS FOR RULE-BASED SYSTEMS

The proposed taxonomy is a hierarchical one (two-level), and functionally similar detailed features are grouped together. The classification is applicable both to facts representing unconditional knowledge and rules divided into preconditions (*LHS*) and conclusion (*RHS*). The issues concerning various anomalies can be grouped and presented as follows:

- **Redundancy:** identical rules, subsumed rules, equivalent rules, unusable rules (ones never fired).
- **Consistency:** ambiguous rules, conflict, ambivalent rules, logical inconsistency.
- **Reduction:** reduction of rules, canonical reduction of rules, specific reduction of rules, elimination of unnecessary attributes.
- **Completeness:** logical completeness, specific (physical) completeness, detection of incompleteness, identification of missing rules.
- **Determinism** pairwise rule determinism, system determinism,
- **Optimality** optimal form of individual rules, locally optimal solutions, optimal knowledge covering, optimal solutions.

Recall that the above taxonomy is considered in the context of simple tabular systems, i.e. no rule chaining problems are taken into account (such as chains leading to contradiction, potential loops, dead-end condition, unreachable conclusion, etc.). Such problems usually require more complex analysis e.g. recursive analysis of potential chains of rules combined in our case with potential inputs occurring at any stage; in a worst case scenario this may be equivalent to simulation of *all* potentially possible executions of the system, and as such it may be computationally intractable. [v]

The proposed classification is somewhat general, but it covers many detailed cases mentioned in the literature.

Further topics which may be covered in the chapter include the following proposals.

REDUNDANCY AND SUBSUMPTION

The very first step in verification of rule-based systems is to check if there are *redundant* rules. A redundant rule is one which in fact is not necessary in the system. The system works with and without redundant rules is the same, at least from functional point of view. This is so, since a redundant system of rule and its non-redundant version are logically equivalent.

Redundant rules should be eliminated. There are three basic reasons for that:

- redundant rules make the whole system less transparent and more difficult to analyze,
- redundant rules may slow down the execution of the system,
- in case of modification of the rule-base redundant rules may lead to inconsistency.

Consider a knowledge base (a rule-base) R. This knowledge base is redundant if there exists at least one component r of it, such that $R \setminus \{r\}$ is logically equivalent to R. In fact, if R is consistent it is enough to check if $R \setminus \{r\} \models R$.

There are the following possibilities while redundancy can occur:

- identical rules in the rule-base,
- equivalent rules in the rule-base,
- subsumed rules in the rule-base.

A somewhat specific is the case of *unusable rules* i.e. ones which never will be fired. There can be three basic reasons while a rule will never be used:

- the preconditions of the rule are inconsistent,
- the preconditions of the rule will never be satisfied due to some properties of the feasible states of the system,
- the preconditions of the rule contain a unique atomic formula not appearing in other rules, the state description, and other part of the system (a *single, isolated item*).

Although there seems to be several specific cases of redundancy, in fact, the case of *subsumption* seems to be the most general one (for example, rule equivalence can be considered as two-directions subsumption; although in case of unusable rules one can speak in terms of specific subsumption). Let us consider this issue in some details.

In general, a subsumed rule r' is one such that, thee exists a rule r replacing it in every situation. This requires that:

- the preconditions of r are weaker than (or equivalent to) the ones of r',
- the conclusion of r is stronger than (or equivalent to) the one of r'.

Let the rules be of the form $r : \varphi \to h$ and $r' : \varphi' \to h'$. The above requirements can be formalized in a straightforward way as follows:

$$\varphi' \models \varphi,$$

and

$$h \models h'.$$

depending on the language (propositional, attributive, Datalog, Prolog) the checks for subsumption can be performed with rules discussed in a preceding section.

A redundant rule can be eliminated from the rule-base without influencing the possibility of inferring new knowledge.

Consider a simple example of subsumption in attributive logic. let us have two rules as follows:

$$r : aTD = wk \rightarrow aOP = ndbh,$$

and

$$r' : aTD = wk \wedge aTM \in [9-17] \rightarrow aOP = ndbh.$$

obviously $(aTD = wk \wedge aTM \in [9-17])$ $aTD = wk$, i.e. rule r' has stronger preconditions (unnecessary atom). Rule r' is subsumed by rule r and as such can be eliminated.

Another more complex example of subsumption can be as follows. Consider rules

$$r : aTD = wd \wedge aTM \in [9-17] \rightarrow aOP = dbh,$$

and

$$r' : aTD = wd \wedge aTM \in [10-15] \rightarrow aOP = dbh.$$

Again subsumption holds, this time because $aTM \in [10-15] \models aTM \in [9-17]$ (this is due to the *upward consistency rule*, see (10).

As a practical example let us check if there are redundant rules in the four groups of rules of the THERMOSTAT system.

Consider the first group of rules given by (43). It is immediate to see that no logical implication holds neither among the preconditions nor among the conclusions. In fact, both preconditions and conclusions are mutually exclusive.

Further, it is easy to check that no subsumption among rules of group 2 (44), group 3 (45), and group 4 (46) takes place. Further, comparison of rules from different groups is useless due to different schema of preconditions and conclusions.

Note that checking for subsumption becomes even more simple when analysis the XTT (tabular) forms of the rules. in his case one can compare only the attribute values placed in two analyzed rows of the table.

INDETERMINISM AND INCONSISTENCY

Problems of indeterminism, sometimes referred to as ambiguity, occurs when two (or more rules) can be fired at the same time, but they produce different results. In general, when two (or more) rules can be fired if their preconditions formulae are not pairwise inconsistent, and there exists a feasible state of the system such that all the preconditions are satisfied.

In such a case various not required things can happen. In general, there are the following three possibilities:

- indeterministic behavior of the system: depending on which rule is fired, the system behaves in a different way,
- conflicting rules: depending on which rule is fired, conflicting results are produced. In case both rules are fired a real conflict occurs,
- logical inconsistency: the same as above but the inconsistency is of logical character.

The general idea in this section is that it is worth to analyze the sets of rules aimed at working in the same context and detect all such pairs of them that can be fired simultaneously. The further analysis is usually left to domain experts and system designer.

Consider two rules $r:\phi \to h$ and $r':\phi' \to h'$. In order to check if the rules are such a pair one is to check if the preconditions of them form a satisfiable formula or not. Hence, the generic check consists in finding if joint precondition formula

$$\phi \wedge \phi'$$

is satisfiable or not. Since in fact we would like to have a deterministic set of rules[vi] the basic approach consists in proving that the above formula is unsatisfiable. This can be done with the rules presented in a preceding section.

Consider the case of attributive logic. Let us have two rules as follows:

$$r: A_1 \in t_1 \wedge A_2 \in t_2 \wedge \ldots \wedge A_n \in t_n \to H = h_1,$$

and

$$r': A_1 \in s_1 \wedge A_2 \in s_2 \wedge \ldots \wedge A_n \in s_n \to H = h_2.$$

A satisfactory condition for the rules to be deterministic is that for at least one $i \in \{1,2,\ldots,n\}$ there should be

$$t_i \cap s_i = \varnothing.$$

In fact, this is a case of the empty intersection rule (23).

As an example consider the rules of the THERMOSTAT system. Let us check the pairs of rules within the four groups. It can be easily observed that the system is perfectly deterministic. No two rules can be applied at the same time.

MINIMAL REPRESENTATION

Verification of minimal representation consists in attempting if reduction of rules (gluing) is possible. Reduction of the rule-base consists in transformation of the initial rule base to some *minimal form*, i.e. one logically equivalent to the initial rule-base, but having minimal number of rules. A minimal rule base is usually easier to analyze, modify and verify; it is also more efficient for practical applications. In case of hardware implementation it contains less components. A rule-base with small number of rules can be interpreted faster and hence such a system will be more efficient.

In general, analysis of minimal representation is performed via reduction of the number of rules and it can be achieved in three ways:

- elimination of unnecessary rules (identical, subsumed, unusable),
- replacing two (or more rules) with a single, equivalent rule,
- replacing three (or more rules) with two (or more), so that the number of rules decreases, while preserving the logical equivalence.

The first approach was discussed in the section covering subsumption. The third one seems quite a complicated one, and it seems that one cannot expect some universal solutions here. The second approach consists in gluing together two (or more) rules. The logical method consists in selecting rules with the same conclusion part and gluing their preconditions (if possible) with use of dual resolution (Ligęza, 2006).

The principle of gluing two rules into a single equivalent rule is as follows. Let us consider two rules, r_1 and r_2 given by

$$r_1 : \varphi \wedge p_1 \rightarrow h$$

and

$$r_2 : \varphi \wedge p_2 \rightarrow h,$$

respectively. Note that the preconditions are different with respect to a single atomic formula, while the conclusions are the same. If there exist an atomic formula q, expressible within the accepted knowledge representation language, and logically equivalent to the disjunction $p_1 \vee p_2$. Then these two rules can be replaced with a single rule r of the form

$$r : \varphi \wedge q \rightarrow h.$$

Rule r is logically equivalent to rules r_1 and r_2.

In fact, one can glue together two or more rules having the same conclusions, provided that their preconditions can be glued together to obtain an equivalent rule expressible within the accepted language. The appropriate reduction rules for the four different logical languages under discussion are presented in section 5.3.

A more general version of reduction may concern k rules ($k \geq 2$). A general scheme of such a reduction is based on (42).

Consider a set of k rules, where rule i, $i \in \{1,2,\ldots,k\}$ is of the form:

$$r_i : \varphi \wedge \phi_i \rightarrow h. \tag{47}$$

All the rules must have identical conclusion h and there must exist formula $\varphi \equiv \varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_k$. In such a case the k rules can be exchanged with a single equivalent rule r of the form:

$$r : \varphi \wedge \phi \rightarrow h. \tag{48}$$

Note that reduction becomes especially elegant and efficient in case φ is an empty formula always true; in such a case the length of the precondition formula decreases.

As an example, let us see if there are any reduction possibilities in the example THERMOSTAT rule base.

In case of the first group of rule, rules 1 and 2 have different conclusions; hence, reduction cannot be considered.

In the case of the second table rules 4 and 5 can be reduced to a single rule of the form:

$$Rule\ 4-5:\ aTD = wd \wedge aTM \in ([0,9) \cup (17,24]) \rightarrow aOP = ndbh$$

provided that an atom of the form $aTM \in ([0,9) \cup (17,24])$ is accessible within the knowledge representation language[vii].

Rules in the third group have pairwise different conclusions; no reduction is possible.

In the fourth group of rules only rules 11 and 15 have the same conclusion. In fact, they can be reduced to (at least potentially) a single equivalent rule of the form

$$Rule\ 11-15:\ aSE \in \{spr, aut\} \wedge aOP = dbh \rightarrow aTHS = 20$$

This again requires that atomic formula $aSE \in \{spr, aut\}$ must be legal within the accepted language.

COMPLETENESS

A *complete* rule-based system is one able to fire at least one rule for any input. This means that the preconditions of the rules should cover all the space of possible inputs.

The early discussion of completeness can be found in (Suwa, Scott & Shortliffe, 1985) and it concerns the ONCOCIN system and (Nguyen et al., 1985) with respect to the LES system. More precisely, (Suwa, Scott & Shortliffe, 1985) identifies the problem of *missing rules* as *a situation exists in which a particular inference is required, but there is no rule that succeeds in that situation and produces the desired output*. Further, some pragmatic considerations define *An exhaustive syntactic approach for identifying missing rules would assume that there should be a rule that applies in each situation defined by all possible combinations of domain variables*. In (Nguyen et al., 1985) *missing rules* are referred to as *a situation in which some values (called legal values) of an object's attribute are not covered by any rules's IF clauses*. A nice discussion of verification and validation issues covering the exhaustive completeness check is given in (Andert, 1992).

A first definition of completeness can be found in (Cragun & Steudel, 1987): *Completeness means that a knowledge base is prepared to answer all possible situations that could arise within its domain [...] Completeness checking is a debugging aid which finds logical cases that have not been considered; in other word, missing rules*.

A. Preece in his works refers to missing rules as *deficiency* or *incompleteness* (Preece et al., 1991; Preece, 1991, 1993) in (Preece, 1993) he writes *A knowledge base is deficient if there is some set of inputs for which it will infer no conclusion*.

A more detailed definitions of completeness of rule-based systems are introduced in (Ligęza, 1993a) and (Ligęza, 1993b). They refer to *logical* (syntactic level) and *physical* (model level) completeness. At both these levels *total* and *partial* (specific) completeness was defined. This line of defining completeness is followed in (Ligęza, 2006) and, to some extent, below.

Completeness can be of two types:

- purely logical completeness, when the preconditions of the formulae form a *tautology*; in such a case, for *any* input formula at least one of the rules could be fired,
- specific completeness, i.e. at least one rule will be fired within a certain context of work.

The full, purely logical completeness can seldom be assured. It seems that only in case of relatively simple systems composed of several rules full completeness can be achieved and verified. This is, for example, the case of propositional logic based rules and systems, e.g. some combinatorial circuits such as 2,4 or 8 input NAND or NOR gates, etc. In general, in case of n binary inputs there are as many as 2^n potential input states which must be cover by the rule preconditions; for some considerations and examples see (Cragun & Steudel, 1987). In case of first-order logic, completeness can be proved with use of the *dual resolution rule* (originally: *backward dual resolution rule*) for some simple cases (Ligęza, 1993a, 1993b).

On the other hand, some minimal requirements w.r.t. the knowledge base completeness are always present. Roughly speaking, a set of rules under discussion should cover and serve some predefined set of cases, perhaps as large as possible. Hence, it seems reasonable to define and evaluate completeness of a rule based system within a set of given contexts. In fact, any group of similar rules (forming an independent component of the knowledge base) should be evaluated separately. In practical cases *specific* completeness will mostly be checked, i.e. covering a pre-specified set of cases.

Let us consider a set of rules R aimed to work within some context defined with formula Δ. Checking for specific completeness requires checking if

$$\Delta \models \Omega,$$

where Ω is the joint precondition formula of all the rules of R.

Note that when checking for completeness there are two relaxations w.r.t. reduction to minimal form; these are:

- first, unifying substitutions can be applied if necessary; this follows from the fact that applying a substitution leads to a less general formula, so the initial one is a logical consequence of it,
- second, one can glue preconditions of any formulae, not necessarily the ones with the same conclusions (this is the coverage of inputs that matter).

The ideal result would be if we obtain an empty formula always true; this would in fact mean that the system is *logically complete*. Unfortunately, as mentioned before, this may be mostly the case of small, technical systems or artificial, academic ones.

In realistic system the situation is not so nice. Unfortunately, although Δ is a simple conjunctive formula[viii], Ω is a DNF formula. In such a case there are three basic possibilities for verification of completeness:

1. Perform maximal possible reduction of Ω; if the reduced form is a simple conjunctive rule, apply the test based on logical implication checking for simple formulae described in section 5.

2. Perform a split of Δ into a DNF formula composed of smaller, more precise simple formulae $\Delta_1, \Delta_2, \ldots, \Delta_k$, such that each Δ_i is covered by some component simple formula of Ω.
3. In case such a reduction and split does not lead into the final result, one may try to perform a full proof that $\Delta \models \Omega$ using the dual resolution principle in the form appropriate for the logic in use.

Obviously, combination of the three approaches as above is possible.

Let us see how the check for completeness works in the case of the THERMOSTAT rule-base.

Consider the first group of rules. Gluing preconditions of rules 1 and 2 leads to formula

$$aDD \in sWD \cup sWK.$$

Note that $sWD \cup sWK$ constitutes in fact the complete domain aDD (there are no other possibilities). Hence the set of rules is complete within the context of work, i.e. provided that the attribute aDD takes any value from its domain.

In case of the second group of rules, preconditions of rules 3, 4, and 5 can be glued to formula

$$aTD = wd.$$

This formula can be further glued with preconditions of rule 6 leading to

$$aTD \in \{wd, wk\}.$$

This means that this set of rules is also complete w.r.t a given context.

In case of the third group of rules the analysis is even simpler. Preconditions of rules 7, 8, 9, and 10 can be glued to

$$aMO \in aSUM \cup aAUT \cup aWIN \cup aSPR.$$

Again, we arrive at full domain of aMO and this set of rules is specifically complete.

Finally, a similar analysis can be performed for the fourth group of rules. First, note that preconditions of rules 11 and 12, 13 and 14, 15 and 16, and finally 17 and 18 can be glued to respectively $aSE = spr$, $aSE = sum$, $aSE = aut$, $aSE = win$. In the next step these formulae can be glued together and we obtain

$$aSE \in \{spr, sum, aut, win\}.$$

Obviously, we arrive at specific logical completeness again.

FUTURE TRENDS

Formal verification and validation of business rules systems becomes more and more important, as these systems gain popularity in the enterprise application. There are three main reasons for that.

First of all mission-critical applications (e.g. involving financial responsibility) or strategic planning require reliable software. Even simple faults or defects in the rule base of a decision system may cause system failure and significant financial loss and a failure to succeed in a highly competitive market.

Secondly, new expressive rule languages allow for flexible knowledge modelling at various levels of abstraction. However, they still require strict formal foundations in order to provide formal verification features.

Third, it is more and more common that various business partners expect or even explicitly require high level of dependability, often with a formal proof of system correctness.

Finally, software certification which becomes a standard in governmental contracts implies formal verification of software as indispensable.

As the future trends in the area of business software verification the following trends can be observed:

- extensive use of visual modelling with instant analysis capabilities,
- integrated design and verification process,
- use of dedicated verification tools during, or after the rule design, such as PVS (see http://pvs.csl.sri.com), or LibRT VALENS (see http://www.librt.com)

CONCLUSION

This chapter provides a concise overview of the state of the art with respect to the formal verification of rule-based systems, a simple and generic classification of knowledge representation languages with respect to expressive power and inference capabilities was put forward. Four basic language levels were identified: propositional logic, simple attributive logic, a Datalog-level logic, and First Order Predicate (or Prolog level) logic. In fact, the majority of the rule representation languages currently used is based on these. The presented classification is important both for discussion of inference methods, as well as for determining inference rules, and approaches to verification.

As the main point a discussion of three kinds of inference rules for *logical inference*, *logical exclusion*, and *logical reduction* (for all of these four languages) was carried out. The presented inference rules contribute to practical criteria of knowledge verification. The addressed issues include subsumption (in a general case redundancy), indeterminism, minimal representation of knowledge and completeness.

Some further issues seem really important and constituting the core of rule-based systems. However, they seems also to have to be covered by separated chapters. These include: inference control strategies, rule precondition matching, conflict resolution, etc., as well as details of knowledge representation languages (the kind of logic in use), tools and technologies for formal knowledge management, and more complex attribute logics, especially ones taking set values.

REFERENCES

Andert, E. P. (1992). *Integrated Knowledge-Based System Design and Validation for Solving Problems in Uncertain Environments.* Int. J. of Man-Machine Studies, 36, 357-373.

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

BRForum. (2005). *Userv Product Derby Case Study* (Tech. Rep.). Business Rules Forum.

Chang, C. L., Combs, J. B., & Stachowitz, R. A. (1990). *A Report on the Expert Systems Validation Associate (EVA).* Expert Systems with Applications, 1 (3), 217-230.

Chang, C.-L., & Lee, R. C.-T. (1973). *Symbolic Logic and Mechanical Theorem Proving.* New York and London: Academic Press.

Coenen, F. (2000). *Validation and Verification of Knowledge-Based Systems: Report on EuroVav'99.* Knowledge Engineering Review 15:2, 187–196.

Cragun, B. J., & Steudel, H. J. (1987). *A Decision-Table-Based Processor for Checking Completeness and Consistency in Rule-Based Expert Systems.* Int. J. Man-Machine Studies, 26, 633-648.

Gabbay, D. M., Hogger, C. J., & Robinson, J. A. (Eds.). (1993). *Handbook of Logic in Artificial Intelligence and Logic Programming (Vol. 1).* Oxford: Clarendod Press.

Knauf, R. (2000). *Validating Rule-Based Systems. a Complete Methodology [Habilitation Thesis]*. Ilmenau, Germany: Ilmenau Technical University, Department of Computer Science and Automation, Chair of Artificial Intelligence.

Lamb, N., & Preece, A. (1996). *Verification of Multi-Agent Knowledge-Based Systems.* ECAI'96 Workshop on Validation, Verification and Refinement of Knowledge-Based Systems, 114-119.

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

Ligęza, A. (1993a). *Logical Foundations for Knowledge-Based Control Systems – Knowledge Representation, Reasoning and Theoretical Properties.* Scientific Bulletins of AGH, Automatics 63 (1529), 144 pp., Kraków.

Ligęza, A. (1993b). *A Note on Backward Dual Resolution and Its Application to Proving Completeness of Rule-Based Systems.* Proceedings of the 13th Int. Joint Conference on Artificial Intelligence (IJCAI), Chambery, France, pp. 132-137.

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

Ligęza, A., & Nalepa, G. J. (2007, may). *Knowledge Representation with Granular Attributive Logic for XTT-Based Expert Systems.* In D. C. Wilson, G. C. J. Sutcliffe, & FLAIRS (Eds.), Flairs-20 : Proceedings of the 20th International Florida Artificial Intelligence research society conference : Key West, Florida, May 7-9, 2007 (pp. 530–535). Menlo Park, California: AAAI Press.

Lunardhi, A. D., & Passino, K. M. (1995). *Verification of Qualitative Properties of Rule-Based Expert Systems.* Applied Artificial Intelligence, 9, 587–621.

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

Nazareth, D. L. (1989). *Issues in the Verification of Knowledge in Rule-Based Systems.* Int. J. Man-Machine Studies, 30, 255-271.

Negnevitsky, M. (2002). *Artificial Intelligence. A Guide to Intelligent Systems.* Harlow, England; London; New York: Addison-Wesley. (ISBN 0-201-71159-1)

Nguyen, T. A., & et al. (1985). *Checking an Expert Systems Knowledge Base for Consistency and Completeness.* Proceedings of the 9-th IJCAI, 375-378.

Preece, A. D. (1991). *Methods for Verifying Expert System Knowledge Bases.* Technical Report, 37 pp.

Preece, A. D. (1993). *A New Approach to Detecting Missing Knowledge in Expert System Rule Bases.* Int. J. of Man-Machine Studies, 38, 161-181.

Preece, A. D., & et al. (1991). *Verifying Rule-Based Systems.* Technical Report, 25 pp.

Suwa, M., Scott, C. A., & Shortliffe, E. H. (1985). *Completeness and Consistency in Rule-Based Expert System.* In B. G. Buchanan & E. H. Shortliffe (Eds.), Rule-Based Expert Systems (p. 159-170). Reading, Massachusetts: Addison-Wesley.

Vanthienen, J., Mues, C., & G.Wets. (1997). *Inter-Tabular Verification in an Interactive Environment.* In F. H. J. Vanthienen (Ed.), Eurovav-97, 4th european symposium on the validation and verification of knowledge based systems (Vol. II, p. 155-165). Leuven, Belgium: Katholieke Universiteit Leuven.

Vermesan, A. (1998). *The Handbook of Applied Expert Systems.* In (chap. Foundation and Application of Expert System Verification and Validation.). CRC Press.

Vermesan, A., & Coenen, F. (Eds.). (1999). *Validation and Verification of Knowledge Based Systems. Theory, Tools and Practice.* Boston: Kluwer Academic Publisher.

KEY TERMS AND DEFINITIONS

verification process formally checking the well-defined properties of a rule-based system against its formal specification,

redundancy a rule base is redundant if it contains more rules than necessary; this means that some of the rules can be removed without changing the inferencing possibilities of the original rule base. This includes the cases of identical rules, subsumed rules, equivalent rules, unusable rules (ones never fired).

determinism a system is deterministic if for every potential input it produces a well-defined unique output; in case the same input and memory state enforces the same unique output to be generated.

completeness for every potential input there exists at least one user-defined rule that can be fired to react in a designed way to the input.

consistency in terms of pure logic (the so-called logical consistency) a system is consistent if any legal inference does not lead to unsatisfiable statement.

---

[i] The best analogy to clearly show the meaning and scope of these notions may be based on referring to an airplane: it is perfectly safe if it never starts – however, it is not reliable since it does not do its duty; obviously, it would also be inefficient. A reliable airplane performs according to desired schedule, it is safe if after any take-off there is exactly one landing, and it is efficient if the cost per passenger per one kilometer is lower than for other planes.

[ii] One may say: `You get whatever you want provided that you know how to ask for that in the accepted language'.

[iii] Extension of attributive logic towards attributes taking set values are considered in (Ligęza, 2006)

[iv] Note that, in fact from the point of view of *control theory* the specification is not one of a *thermostat* aimed at temperature stabilization at a certain *set point*. The presented system is a higher-level decision system providing the specification of the *set point* determining procedure.

[v] Well, analysis of complete set of cases of execution can be performed for certain, not-too-large systems; this, in fact, is carried out by Prolog execution mechanism, and was also implemented in several rule-based verification systems, such as CHECK (Nguyen et al., 1985) w.r.t. circular rules detection or COVER [18] for deficiency detection in backward chaining systems. Certainly, a system once checked in a complete mode can be considered reliable and therefore can be safely used in the future, perhaps in multiple copies. On the other hand, we believe that for more complex systems, such complete checking is rather infeasible, especially if the system incorporates a language equivalent to first-order logic with equality and interpreted functions, but can be dealt with by keeping dynamic track of executed rules and results obtained at subsequent stages of inference, i.e. it can be performed on-line, during work of the system. This, however, would require that appropriate procedures are built-in into the rule interpreter.

[vi] This is an obvious choice, however, in numerous practical applications one can admit indeterministic rules; the further dealing with that is left to the inference control mechanism.

[vii] Most of such languages allow only for specification and use of convex intervals.

[viii] In fact, it is the most frequent case, but if not, it can be split into separate checks each for a single DNF component.