Antoni Ligęza[*]

KNOWLEDGE REPRESENTATION AND INFERENCE FOR ANALYSIS AND DESIGN OF DATABASES AND TABULAR RULE-BASED SYSTEMS[**]

1. Introduction

Rule-based Systems (RBSs) constitute a powerful tool for specification of knowledge in design and implementation of knowledge-based systems (KBSs) in applied Artificial Intelligence (AI). They provide also a universal programming paradigm for domains such as intelligent control, decision support, situation classification and operational knowledge encoding. Apart from off-line expert systems and deductive data-bases, one of the most useful and successful applications consists in development of wide spectrum of control and decision support systems [9].

In its basic version (considered here) a RBS for control or decision support consists of a single-layer set of rules and a simple inference engine; it works by selecting and executing a single rule at a time, provided that the preconditions of the rule are satisfied in the current state. Possible applications include direct control and monitoring of dynamical processes [4], meta-level control (the so-called expert control), implementation of the low level part of any-time reactive systems, generation of operational decision support, etc. A RBS named *Kheops* [7], being one of classical examples of such systems was applied in the TIGER system [36] developed for gas turbine monitoring.

The expressive power and scope of potential applications combined with modularity make RBSs 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*, *quality* 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 more detailed characteristics defined in terms of logical conditions.

* Institute of Automatics, University of Mining and Metallurgy, Cracow

** This is an extended derivate paper of [19]. The research was supported from KBN Grant No 8 T11C 019 17

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 *quality* [1, 9, 28, 29, 31, 39]. Some most typical issues of theoretical verification include satisfaction of properties such as *consistency*, *completeness*, *determinism*, *redundancy*, *subsumption*, etc. (see [1, 26, 28]). Several papers investigate these problems presenting particular approaches [3, 29, 31, 39]. A selection of tools is presented in [32]. Some modern approaches include [2, 41, 10].

The problems listed above become still more important in case of rule-based methodology applied to on-line, real-time control of dynamic systems (i.e. intelligent control, knowledge-based control) [9, 40], especially if safety issues are to be taken into account. Some of these problems may be of critical nature; for example, in case of lack of completeness, for certain states of the controlled system there are no rules to serve these states. This may make the system unreliable or unsafe. A recent painful example of such lack of safety[1)] was the crash known as *The Warsaw Accident*, when a plane hit on high speed into an earth bank after successful landing under heavy weather conditions; switching to reverse thrust and opening spoilers (both for efficient breaking down) were disabled. For *certain* values of the speed of wheel spinning and weight *no action* was designed to be undertaken. It can be argued that static analysis of theoretical characteristics of such a system, in our case checking for completeness, could perhaps throw some light on missing rules identification and finding specification of gaps in the input state space served by the RBS applied for control.

1.1. Aims of this paper

This paper addresses the issue of analysis and verification of selected properties of a class of such systems in a systematic way. A uniform, tabular scheme of single-level rule-based systems is considered. Such systems can be applied as a generalized form of databases for specification of data patterns (unconditional knowledge), or can be used for defining attributive decision tables (conditional knowledge in form of rules). They can also serve as lower-level components of a hierarchical, multi-level control and decision support knowledge-based system. An algebraic knowledge representation form using extended tabular representation, similar to relational database tables is presented and algebraic bases for system analysis, verification and design support are outlined.

Several characteristics of RBSs are identified and analysed. Special attention is paid to completeness. The problem of completeness verification is mostly studied in [3, 29, 31, 39]. The approaches presented there are mostly based on *exhaustive enumeration* and examination of possible combinations of input variables/conditions; combinatorial explosion is controlled through restriction of the analysis to some local context. A more efficient approach is presented in [33]; the approach proposed there concentrates on meaningful deficiencies and uses heuristics about the nature of some likely deficiencies which improves its performance and makes communication with the user more clear. An interesting solution is used

[1)] Caused in our rough and subjective interpretation by lack of completeness of the control algoritm; in fact the detailed analysis is more complex [37].

in the KHEOPS system [7]; its main idea consists in compiling the set of rules into a decision-tree like structure. The principle goal of building such a representation is to have the possibility of efficient rule evaluation and obtaining an upper-bound for respond time in real-time applications (the response time is bounded by the time necessary for traversing the longest path in the tree). This solution allows also for limited checking of completeness and consistency. The methods, however, are mostly applicable to rules constructed with use of propositional logic (i.e. the zero-order logic) or slightly extended languages (e.g. finite domain attribute-value languages).

1.2. Position statement and state of the art

RBSs provide a powerful tool for knowledge specification and development of practical applications. However, although the technology of RBSs becomes more and more widely applied in practice, due to its relationship to first-order logic and sometimes complex rule patterns and inference mechanisms, they are still not well-accepted by industrial engineers. Further, the 'correct' use of them requires much intuition and domain experience, and knowledge acquisition still constitutes a bottleneck for many potential applications. Software systems for development of RBSs are seldom equipped with tools supporting design of the knowledge-base; for some exceptions see [38, 8]. A recent, new solution is proposed in [42]. However, a serious problem consists in the fact that a complete analysis of properties remains still a problem, especially one supporting the design stage rather than the final verification. This is particularly visible in case of more powerful knowledge representation languages, such as ones incorporating the full first order logic formalism.

Contrary to RBSs, *Relational Data Base Systems* (RDBSs) offer relatively simple, but matured data manipulation technology, employing widely accepted, intuitive knowledge representation in tabular form. It seems advantageous to make use of elements of this technology for simplifying certain operations concerning RBSs. Note that from practical point of view any row of a RDBS table can be considered as a rule, provided that at least one attribute has been selected as an output (and there is a so-called *functional dependency* allowing for determination of the value of this attribute on the base of some other attributes). Thus, it seems that merging elements of RBSs and RDBS technologies can constitute an interesting research area of potential practical importance.

This paper investigates RBSs by means of RDBS-like tabular knowledge representation and algebraic rather than logical tools. A relatively simple approach derived from first-order logic, but incorporated into a RDBS-like framework, is proposed. Further, a hierarchical structure (or even a network-like one) of the RBS can be assumed; for simplicity, only two levels are considered below. The organization of the system is as follows:

- there is an upper level consisting of several *contexts* of work, defined with formula $C_1, C_2, \ldots, C_c$; selection of the context depends on current working conditions and the goal; selection of the current context can be performed by a meta-level decision mechanism, while switching among contexts can be a side effect of lower level rule application;
- for any context C_i there is a simple, uniform (i.e. using the same scheme of attributes), tabular RBS, similar to a decision table, with knowledge representation based on attributes. When the context is selected, the system identifies and applies a single rule; then the cycle is repeated.

The system is organized in a hierarchical way; in the simplest case above one has just two-level specification. The upper level provides context selection mechanism, while the lower level is responsible for rule selection and application. A graphical presentation of the idea of hierarchical system is presented in Figure 1.

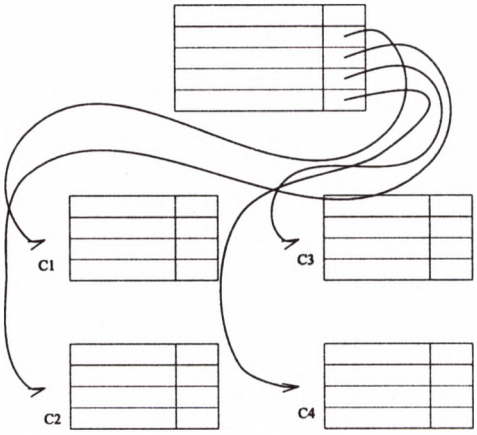

Fig. 1. Graphical presentation of the idea of a hierarchical, two-level tabular rule-based system

In general, one can consider any more complex scheme, including several levels and numerous tables connected according to various patterns. Note that such a multi-tabular system can be (at least potentially) reduced to one big table, similarly as in the case of RDB systems. However, it would not be a reasonable approach, both from the knowledge representation and the analysis point of view – most of the attributes would be useless in most of the rules. Instead, it seems much more appropriate to analyse any particular uniform tabular system within the appropriate context C_i. Thus, the discussion presented in this paper is restricted to the level of a single tabular component; note however, that the meta-knowledge concerning context switching can also be specified with use of appropriate tables.

A single-table system is assumed to be a forward-chaining one, operating according to the following scheme: for given input situation, an applicable rule is searched for, and if found, the rule is fired. The environment of the system may be changed by the system itself, or changes may be due to dynamic environment, as in the case of rule-based control systems [13, 16]. For the idea see Figure 2.

This paper addresses the issues emerging during logical verification of theoretical properties especially in case of such single-layer rule-based reactive systems; a geometric interpretation of the work of the system is presented in Figure 3.

RBSs defined and working as above constitute a class of applicable in a wide spectrum of control and decision support tasks [13, 9, 26]. The right-hand side of any rule is normally a control action (sometimes also an assertion to or deletion from the fact base) or decision; no chaining among the rules takes place. Their standard working cycle proceeds as follows: the current state of the input environment is observed, then a single rule matching the input pattern is selected, and, finally, the selected rule is executed. The whole cycle is repeated in a closed loop.

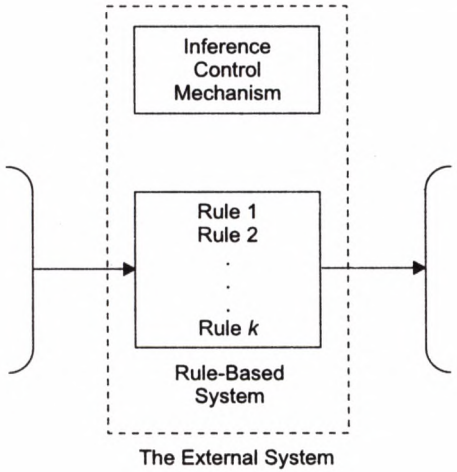

Fig. 2. A general scheme of single-level rule-based system

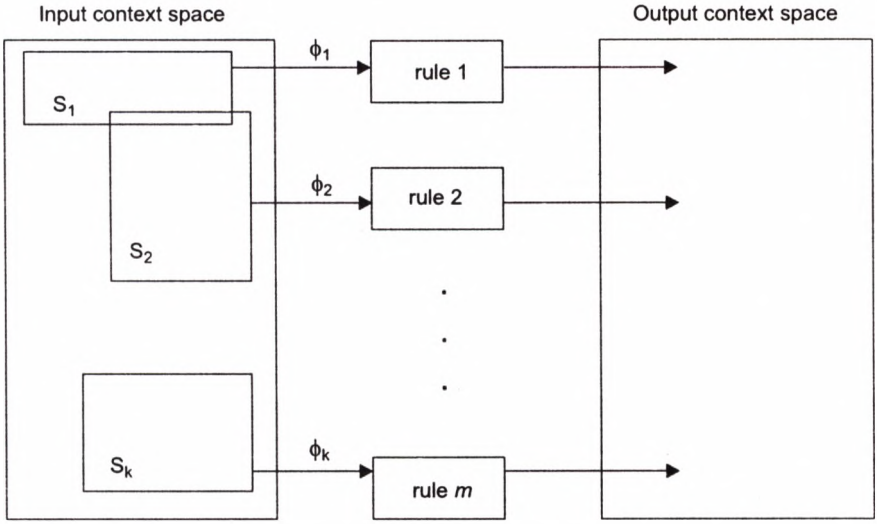

Fig. 3. An abstract geometric presentation of the working scheme of a rule-based control system

Systems as above form a class of simple forward reasoning expert systems with no chaining among rules (application of a single rule results in conclusions/actions). Such systems were discussed in [13, 17, 26]; many example applications are given in [9]. An important area of applications include intelligent control systems with control knowledge specified as a RBS. They can also constitute the lowest part of more complex, hierarchically structured systems where they constitute the core inference tool for any context determined at higher level. For intuition, a system as one presented in Figure 3 is complete, if the rectangles referring to rule precondition formulae cover the input space.

The tabular systems discussed in this paper can also be used as extended RDB paradigm for unconditional knowledge specification. In such a way instead of extensional, data specification with atomic values of attributes, their intensional definition can be provided. In the basic case, set and interval values of attributes can be used to cover a number of specific cases. Depending on the knowledge representation language, also more complex structures (e.g. records, objects, terms) can be used. In such a way *data patterns*, *data covers* or *data templates* can be defined. Both representation and analysis can be then much more concise and efficient. An illustration for this idea is presented in Figure 4.

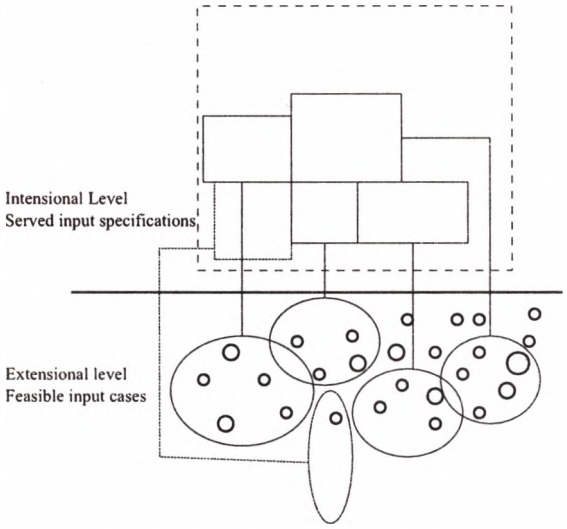

Fig. 4. Graphical presentation of the idea of data templates representation replacing extensional data specification

Note that there are many common points in the analysis of such intensional, unconditional data representation and the verification of tabular RBS properties. For example, checking for determinism of a rule based systems requires verification if their precondition formulae are defining separate sets of states; the same check can be performed to verify if certain data templates describe disjoint sets of data. Analogous situation occurs in checks of completeness, etc. This allows to present and discuss the problems of analysis simultaneously for data templates (extended database paradigm) end tabular systems using a common model for data and knowledge.

1.3. Principal ideas and organization of the paper

The main aim of the paper is to provide elements of algebraic approach for data and knowledge specification and analysis. The main knowledge representation paradigm used throughout this paper is the one of tabular attributive decision tables with non-atomic values of attributes. The so-called *tabular systems* may specify the lower components of a bigger system, especially of the hierarchical tabular one, or may be considered as independent knowl-

edge bases. There is an attempt to pass from the logical level of verification (as analyzed in e.g. [14]) towards more simple, algebraic level of analysis. The main goal of such algebraization is to simplify and make the analysis more efficient, while keeping consistency with logical approach. Simultaneously, on-line verification of even partially specified system is assumed to support the design of such systems. This is achieved by constructive evaluation of anomalies which can occur, such as missing chunks of knowledge, overlapping ones, etc.

The organization of this paper is as follows. First, a taxonomy of verification issues is put forward and a brief overview of the problems is presented. Logical definitions of identified anomalies are provided. Next, basic logical foundations are recapitulated in brief and the definition the so-called *backward dual resolution* (an inference rule) is restated; the method itself is discussed in details in [14, 13, 16, 15]. Then simplified attribute-based tabular knowledge representation based on RDBSs scheme is put forward. The following discussion is aimed at supporting verification of RBSs for design of systems assuring satisfactory level of their reliability and quality; several issues containing redundancy, subsumption reduction, completeness, determinism, etc. are presented and algebraic verification methods are outlined. The paper is closed with concluding remarks.

2. A taxonomy of verification issues

In this section a new, generalized taxonomy of the issues concerning knowledge verification is put forward. The taxonomy covers most of the specific problems considered by other Authors, but groups together similar problems by taking into account the approach to perform appropriate check and deal with a specific anomaly. Further, logical definitions of specific anomaly classes are provided.

2.1. Verification of RBS: a short review

In this section a short review of selected, most common anomalies is presented. A kind of review of the theoretical problems specified by the authors with respect to verification of RBSs properties is provided. The discussion is based mainly on the following positions [1, 3, 27, 28, 34, 32, 33, 35, 26, 39]. However, contrary to typical presentations, the discussion is *functionality-oriented*, i.e. we start from top-level, abstract characteristics required to achieve. Further, practical experience and theoretical discussion as well as former author's papers are taken into account.

Some most general classification of theoretical issues which undergo theoretical analysis can be one referring to the degree of influence they may have on system performance, and ranging from problems of *efficiency* and *elegance* of knowledge representation (e.g. redundancy and subsumption) to certain *critical errors* inside the encoded knowledge (e.g. *incompleteness* and *inconsistency* of specified knowledge). This point of view would be especially important when the analysis is to provide confidence about *reliability* and *safety* of an on-line (or even real-time) KBS working in safety-critical environment. On the other hand, *reliability* and *safety* are always to be considered w.r.t. KBS and its environment considered as two factors having joint influence on each other.

Note that, it is not the KBS itself, but always together with the controlled system and its environment which makes danger, crush or failure to occur. Thus, considering safety problems should be based on more global analysis covering not only the software system but its potential interaction with its environment. For example, redundancy, which is normally considered harmless, may slow down operation of a real-time critical system and lead to some serious consequences since the output would be delayed; on the other hand, even inconsistent or incomplete system can work well for a long time (even for years), provided that no use of its knowledge leading to direct manifestation of inconsistency is done or the uncovered inputs do not occur.

On the other hand, theoretical issues considered from the point of pure KBSs theory (such as correctness, incompleteness, inconsistency, consistency with reality) may appear "mathematically elegant" when analyzed at the level of theoretical definitions, but may turn out to become less attractive and hard to formalize when it comes to practical applications and efficient analysis. Especially purely logical approaches, e.g. ones based on automated theorem proving, would seem promising, but are hardly applicable for realistic systems. The same applies to software verification in general case, where proving consistency of final code with initial specification is a hard, tedious, usually not a realistic task. But even if performed successfully, it is not the of the problems.

2.2. Functional quality assignment

The approach of this paper is relatively simple and conservative, based on good experience and success of RDB systems. Simultaneously, we try to be constructive: an engineering approach is pursued. An attempt is made at presentation of working classification of general issues of interest with respect to theoretical analysis, each of them having relatively different origin and ways to detect and deal with it.

Whenever appropriate, an idea of trouble detection procedure is outlined and suggestions about potential solutions are given. Further, the nature of potential problems is explained is case of omitting the analysis of a particular problem. In order to stay close to both engineering intuition and potential practical solutions, the discussion turns around simplified form of KBSs, i.e. tabular RBSs mentioned above.

From the point of view of top-level, desired functional specification, the following five characteristics are put forward:

1) *safety*, i.e. the design of the tabular system should assure that nothing dangerous would ever happen,
2) *reliability*, i.e. the system should work and achieve its goals, possibly under any external circumstances,
3) *admissibility*, i.e. the system should provide only admissible decisions or conclusions and should satisfy any constraints imposed on it,
4) *quality* i.e. the system should satisfy certain standards, especially satisfy explicit and implicit standards and user requirements,
5) *efficiency*, i.e. the system should work in possibly most efficient way (perhaps even optimal) and should be specified in an efficient way (e.g. with use of minimal number of rules, in the simplest form, etc.).

These top level characteristics appear to be both pairwise dependent and perhaps inconsistent with one another, i.e. satisfaction of one may lead to violating another one. For example, a safe plain would be one which never flies, but such a plain would not be reliable, not to tell about efficiency. Further, unfortunately, these characteristics are hardly expressible with use of a formal specification. Thus, instead, the approach pursued in this paper consists in translating them into quite technical, static analysis of selected features which can be defined formally with use of logical specification. Below, a proposal of a general taxonomy is presented.

2.3. A taxonomy for verification of logical characteristics

The technical classification of theoretical properties of databases (considered within the extended paradigm, i.e. as data templates or knowledge facts) and especially tabular rule based system should possibly cover the complete spectrum of potential deficiencies. At the same time, the structure of the taxonomy should reflect verification paradigms, i.e. features analysed with the same or similar tools should be grouped together. A proposal of a new taxonomy constituting an attempt at satisfying these principles and kept as transparent as possibly is presented below.

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*).

- Issues concerning adequacy of knowledge representation, including:
  - repeated identical data templates or rules,
  - redundant, equivalent data templates or rules,
  - subsumed, less general facts or rules,
  - unnecessary attributes,
  - too many attributes,
  - unusable rules (ones never fired).
- Issues concerning completeness of knowledge representation, including:
  - logical (total) completeness,
  - specific (partial) completeness,
  - detection of incompleteness,
  - identification of gaps,
  - completeness of conclusions.
- Issues concerning internal consistency, including:
  - determinism or uniqueness vs. ambivalence of results,
  - conflict (overlapping) data templates or rules,
  - logical inconsistency.
- Issues concerning external consistency, problems of correctness, including:
  - satisfaction of external local constraints,
  - satisfaction of external global constraints,
  - verification of consistency with specification (if applicable),
  - consistency w.r.t. model (real world); verification w.r.t real system, especially validation and testing on selected examples.

- Issues of minimal knowledge representation:
  - reduction of the set of data templates or rule,
  - partial reduction of data templates or rules,
  - specific partial reduction of data templates or rules,
  - elimination of unnecessary attributes.
- Issues of knowledge manipulation, including:
  - comparison of two tabular systems of data templates or rules, i.e. if one subsumes the other,
  - performing operations (mostly algebraic ones) on one, two or more tabular systems.
- Issues of generalization and learning, including:
  - induction from positive examples, both in case of data templates and rules,
  - induction from positive and negative examples, both in case of data templates and rules,
  - induction from examples and background knowledge (*knowledge-based induction*).
- Issues concerning similarity, including:
  - finding similar examples of data templates and rules,
  - flexible or soft, partial matching procedures,
  - case-based reasoning and adaptation of results.

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 this may be equivalent to simulation of *all* potentially possible executions of the system, and as such it may be computationally intractable[2)].

The proposed classification is somewhat general, but it covers many detailed cases mentioned in the literature. For example, subsumption of rules cover some four sub-cases of *Redundancy in pairs of rules* [28]. The case of *unnecessary IF conditions*, as discussed in [26], is a specific case of rule reduction discussed in this paper. On the other hand, as mentioned above, some checks requiring recursive analysis are not considered here. Note that in case of simple, reactive, forward-chaining systems it may be necessary to apply the same rule (or a sequence of rules) many times in turn, until external event changes the input (for example, in a supervisory system waiting for a special event). Further, a "circular" rule

---

[2)] 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 [29] w.r.t. circular rules detection or COVER [33] for deficiency detection in backward chaining systems. Certainly, a system once checked in a complete mode can be considered reliable and therefore used safely in 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.

may in fact be equivalent to iteration which may be necessary to load a counter, etc. and finishes only after expected amount of repetitions. Thus, "circular" rules are not necessarily considered harmful.

Let us briefly skim through the proposed taxonomy of anomalies. Below, we shall analyse every class of problems in a brief way and we shall point to the principal checking approach. If applicable, logical specification of the appropriate condition to be verified will be provided.

2.4. Adequacy of knowledge representation

The first group refers to issues concerning *adequacy of knowledge representation*, i.e. whether and how the current representation is inefficient. This issue has mostly no influence on system correctness. However, it can slow down its operation, and become a source of problems during modification or extension of the knowledge base.

It is obvious that *repeated identical data templates or rules* should be eliminated, as well as *redundant, equivalent data templates or rules*; the letter may however be difficult to identify without a theory supporting the proof of equivalence. *Redundant rules* (not necessarily identical – see the further discussion) can be detected and removed, leaving no more than one copy for each rule.

The most interesting is the case of *subsumed, less general facts or rules*. A rule of the form $\phi \rightarrow h$ subsumes (is more general than, or is stronger than) a rule $\phi' \rightarrow h'$ if and only if it offers the possibility to draw stronger conclusions from weaker prerequisites, i.e. iff $\phi' \models \phi$ and $h \models h'$. This case will be analysed in details both using logical and algebraic approach. *Subsumed rules* also can be eliminated, which has no influence on logical inference capability. However, in certain cases leaving a subsumed, more specific rule in knowledge base may be purposeful, for example it may affect the conflict resolution mechanism and inference control strategy [26].

The case of *unnecessary attributes (too many attributes)* is not necessarily easy to identify – semantic, domain dependent knowledge is necessary. Such a case is the consequence of existence of functional dependencies among attributes, and since it is relatively well studied in the theory of RDBs, it will not be considered here.

On the other hand, *unusable rules (ones never fired)* can be discovered only with use of analysis of feasible input states. Thus, if the states of infeasible (inadmissible) input states are described with some formula Υ, then if $\Upsilon \models \phi$, the rule $\phi \rightarrow h$ is unusable.

2.5. Issues concerning completeness of knowledge representation

The problem of completeness verification consists in our case in checking if all possible inputs are served by at least one rule. Practically, this means that for any combination of input values and conditions, preconditions of at least one rule should be satisfied.

Logical (total) completeness means that the disjunction of preconditions of all the rules form a tautology, i.e. no matter what input combination occur, it will be served. If there are k rules of the form $\phi_i \rightarrow h_i, i = 1, 2, ..., k$, then logical completeness means in fact that

$$\models \phi_1 \vee \phi_2 \vee ... \vee \phi_k.$$

In practice, not all such combinations may be admissible, or the system may be designed to work only for certain inputs. *Specific (partial) completeness* means that the scope of inputs such that the system is capable of dealing with is explicitly defined with a formula (restricting conditions) specifying the admissible input space. If Ψ is a formula defining some specific context, then the set of rules is specifically complete iff $\Psi \models \phi_1 \vee \phi_2 \vee \ldots \vee \phi_k$.

In both cases, if the system does not satisfy completeness requirements, it may be of interest to determine gaps in the system input, i.e. generate a specification of unserved inputs; this means that *detection of incompleteness* and subsequent *identification of gaps* should be carried out. Logically, one would have to find formulae $\phi'_1, \phi'_2, \ldots, \phi'_m$, such that

$$\models \phi_1 \vee \phi_2 \vee \ldots \vee \phi_k \vee \phi'_1 \vee \phi'_2 \vee \ldots \vee \phi'_m,$$

but such that also

$$(\phi_1 \vee \phi_2 \vee \ldots \vee \phi_k) \wedge (\phi'_1 \vee \phi'_2 \vee \ldots \vee \phi'_m)$$

is never satisfied.

The case of *completeness of conclusions* means that any conclusion which can be specified with the accepted language can (possibly) be achieved. More precisely, having a set of rules as above this would mean that $\models h_1 \vee h_2 \vee \ldots \vee h_k$.

2.6. Issues concerning internal consistency

Problems of internal consistency refer to a case when consistent application of the rules may lead to ambiguous or inconsistent results.

The lack of *determinism or uniqueness* may lead to *ambivalence of results*. *Ambiguous results* may have place in case of when two (or more) rules can be applied for the same input, but their outputs are different. It may be the case that such a result is harmless, or even intended, but in case of reactive control systems the situation like that should be carefully analyzed. From logical point of view, two rules $\phi \rightarrow h$ and $\phi' \rightarrow h'$ are ambiguous iff $\phi \wedge \phi'$ is a satisfiable formula.

A more dangerous case is the one of *conflicting (overlapping) data templates or rules*, i.e. when the simultaneously produced outputs cannot both be correct with respect to the intended interpretation **I** (external world). For example, there are a number of devices which can be in one and only one state (belonging to some specific set of states), and concluding that such a device takes simultaneously two different states leads to physical inconsistency. In the above case two ambivalent rules would be conflicting if apart from overlapping preconditions also the formula $h \wedge h'$ would be unsatisfiable under the assumed interpretation.

The conflict may become *logical inconsistency* if some two conclusions are logically inconsistent, either in direct case (if one is the negation of the other), or in an indirect case (when assuming that both are simultaneously true allows for formal demonstration that there is logical inconsistency). For obvious reasons such problems should be detected and carefully analyzed.

2.7. Issues concerning external consistency, problems of correctness

Problems of external consistency and correctness refer to consistency with externally specified constraints and with the considered real world.

Consistency or *satisfaction of external local constraints*, as well as *satisfaction of external global constraints*, can be verified with logical and algebraic tools. Local consistency usually applies to a single rule. If γ is a local constraint, then local constraint satisfaction can take the form $\gamma \models \phi$, where ϕ is a data template formula or rule precondition formula, etc. Global constraint satisfaction concerns the case of a set of rules as a whole. If Γ is a global constraint, than satisfaction of global constraints takes the form $\Gamma \models \Phi$, where Φ is a formula representing set of data templates or set of formulae preconditions, etc.

Verification of consistency with specification (if applicable) is a hard task; data templates and rule based systems are close to something like *executable specifications*, as they constitute in fact a kind of declarative knowledge expressed in a high level language. However, if logical formal specification is provided, consistency with specification can be performed through checking of logical equivalence.

Consistency w.r.t. model (real world); verification w.r.t real system, especially validation and testing on selected examples seems to be more a practical issue. Consistency with real world can be checked only through testing and it is not of direct interest here; the discussion of it goes far beyond the scope of this paper. Further, a general and satisfactory solution of such problems seems to be hard to achieve. Partial solutions may consist in verifying *potential correctness* [13, 17] through verification of some or all of the theoretical properties mentioned above. It is practically impossible to perform verification against specification since KBSs are close to *executable specifications*; (and this is the case of tabular encoded RBSs considered here). On the other hand, limited testing can be performed (even on real plant or system), but this way of validation always leaves unanswered questions about the behaviour of the system in new, unexplored situations (e.g. unexpected faults). Some initial consideration referring to this issue are presented in [40].

2.8. Issues of minimal knowledge representation

Problems of minimal or maybe optimal representation refer to the possibility of transformation of the initial knowledge table to some, possibly simplest, form, i.e. simplification. This can be obtained through reduction of the table. The reduced form, should, however, be logically equivalent to the input table.

Reduction of the set of data templates or rules means replacing two or more items with a single, equivalent rule by an operation resembling "gluing" the preconditions of them. Note that from logical point of view all of the activities referring to cleaning-up the above anomalies do lead to logically equivalent knowledge base, but a simpler (more elegant) one. The reduction can be total (i.e. maximal; no further reduction is possible) or partial. Such *partial reduction of data templates or rules* may lead to a specific, unique form, called *canonical form* [21]. It can be noticed that the result of total reduction may be, in general case, not unique. It may be wise then, to stop at some stage of partial reduction, but such that the result would be defined in a unique way. More on that can be found in [21].

Specific partial reduction of data templates or rules is a kind of reduction where two or more values, terms or formulae are replaced with another, more general one. Normally, reduction leads to elimination of unnecessary terms or formulae; in case of specific partial reduction the term or formula "survives" but it is generalised w.r.t. the original one.

Elimination of unnecessary attributes may be performed either with use of semantic knowledge about functional dependencies (as mentioned above) or through total reduction (if applicable). In such a case the output does not appear to depend on some attribute(s), which disappear during the reduction process.

2.9. Issues of knowledge manipulation

Having one or more tabular systems, various operations on them are possible. In fact, it is possible to specify algebraic operations, analogous to one of the set algebra or relational database algebra. Such operations can be specified at extensional level or intensional level. The first category refers directly to relational database algebra. The second one requires that the operations are defined and implemented at the higher level of abstraction, but as such they can be much more efficient and concise.

Two basic kinds of operations on tabular systems can be defined. The first one is *comparison of two tabular systems of data templates or rules*, i.e. if one subsumes the other, or, perhaps, they are equivalent. Logically, the check can be expressed as $\Psi \models \Phi$.

Further, having one, two, or more tabular systems of the same attributive scheme, *performing operations (mostly algebraic ones) on one, two or more tabular systems* can be considered. Such operations may include sum, intersection, difference and completion, as well as more complex ones. Some initial discussion of those topics is presented in [21].

2.10. Issues of generalization and learning

Generalization and learning – in contrast to classical understanding of *verification*, induction and learning (generalization) from a set of low-level rules were put together with other issues. This is in order to underline that *verification* should be considered as a part of the RBSs *synthesis*. In fact, it seems reasonable to attempt at direct synthesis of "correct" RBS rather then afterward verification and correction.

One can consider *induction from positive examples, both in case of data templates and rules*, *induction from positive and negative examples, both in case of data templates and rules*, or *induction from examples and background knowledge (knowledge-based induction)*. Note that induction is not a valid logical inference rule, so induction as such may lead to incorrect results. Induction of a knowledge base Φ from a set of $\phi_1, \phi_2, ..., \phi_n$ examples can be formally defined as an attempt to satisfy the condition $\phi_1, \phi_2, ..., \phi_n \models \Phi$ while Φ should be significantly simpler (concise, readable, intentionally specified) than the input formulae defining specific cases.

Since this paper is devoted to basic logical analysis of "traditional" issues considered in verification and validation, these problems are left untouched. Even a very short analysis of abundant literature on rule induction from examples, inductive inference and learning systems would go far beyond the intended contents of this paper, however certainly an attempt at such an analysis would constitute a challenging issue for future investigations.

2.11. Issues concerning similarity

Last but not least, finding and reasoning with similar cases can be of practical interest. This area requires flexible, soft data pattern matching capabilities. Its potential applications concerns case-based reasoning, a technology of reuse of similar solutions for similar problems.

The issues of potential interest include *finding similar examples of data templates and rules, flexible or soft, partial matching procedures*, and *case-based reasoning and adaptation of results*. Such problems, although close to real-life applications, are hard to be expressed with simple logical means, and are outside the scope of interest of this paper. Some initial remarks on them can be found in [43].

In the further part of this paper only some of the problems specified above are considered; some other problems, such as induction or case-based reasoning, have wide literature study with no definite, general solutions, or are skipped as ones having no useful solutions.

3. Logical foundations

Logical foundations for the approach presented in this paper come from the ideas of First Order Predicate Calculus (FOPC); some basic ideas are given, for example, in [5]. In practice, the interest here concerns the concept of logical entailment ($\models$), subsumption, and an inference rule dual to resolution.

A simple, geometric interpretation of logical formulae could provide intuitive interpretation useful in understanding the ideas outlined below. A universe of items (objects, entities, states) is considered to be perceived as a kind of some space. Any point in this space is described with some formula of the accepted language. In order to define exactly one point, the formula must be as precise as possible. A more general formula describes perhaps a set of points, i.e. a subset of the space under consideration.

Let U be the universe of discourse, and let U_1 and U_2 be two subsets of U. Further, let Ψ_1 and Ψ_2 be two formulae describing all the items of U_1 and U_2 respectively, and only the items belonging to these sets. Then, the logical entailment

$$\Psi_1 \models \Psi_2$$

is understood as the relation

$$U_1 \subseteq U_2$$

in terms of the geometric interpretation. This means that a more general formula (in our case Ψ_2) describes a wider spectrum (a bigger subset of U) then a formula which is more specific. This kind of interpretation comes directly from systems science and control theory, but it is universal and omnipresent in common language as well. For details see [13, 16, 17].

For intuition, a very precise formula is one in the form of conjunction of atoms (or literals); more atoms in the conjunction provide more requirements the underlying object must satisfy, so a smaller set of possible items is described. Formulae constructed as conjunctions of literals will be referred to as *simple formulae*, and if no negative literal occurs in such a formula they are also called *positive simple formulae*. For intuition, roughly speaking they correspond to records of a RDB table.

On the other hand, a formula having more components in disjunction appears to describe a bigger subset of the universe of discourse, as it can be more easily satisfied (by more items). In terms of algebraic operations, conjunction corresponds thus to intersection ($\cap$) of respective subsets, disjunction – to sum ($\cup$), and logical entailment to the subset relation ($\subseteq$).

To complete the discussion, variables (iff any) occurring in the formulae are by default existentially quantified. Negation of a formula means complement of the appropriate set.

For the purpose of this paper only two basic types of formulae will be used: these are *simple formulae* meaning simple conjunctions of literals (in general, mostly positive literals will be considered; a *literal* is an atomic formula or its negation) and *normal formulae* being disjunctions of simple formulae (i.e. the so-called Disjunctive Normal Form, DNF). Inference will be based on *subsumption* (more precisely θ-*subsumption* in the purely logical case) and a general method for logical inference which is here *backward dual resolution* (or *bd-resolution* for short) [14, 15].

A simple formula ψ *subsumes* simple formula ϕ if and only if $\phi \models \psi$. It can be shown [6], that for certain class of formulae (ones which are not *self-resolving*, i.e. resolving with a copy of itself [6]), subsumption can be equivalently expressed by the following, simpler condition

$$\psi\theta \subseteq \phi \tag{1}$$

for some substitution θ and provided that ϕ is not a tautology. Since the formulae discussed in this paper are not self-resolvable (we assume independence of attributes used to form atoms), condition (1) will be used to check for subsumption. Note that variables are considered to be existentially quantified, and this certain kind of uncertainty – introduction of a variable in place of a constant term makes a formula more general. The condition of θ-*subsumption* can replace logical entailment of the form $\phi \models \psi$, as it is a specific case of it for positive simple formulae.

The bd-resolution inference method [13, 16, 14, 15] is in fact dual to classical resolution described in [5]; furthermore, it works backwards, in the sense that normally the disjunction of the parent formulae is a logical consequence of the generated bd-resolvent – the direction of generation of new formulae is inverse with respect to the one of logical entailment.

For intuition, let us consider two propositional logic formulae, $\psi_1 \wedge \omega$ and $\psi_2 \wedge \neg\omega$, where ω is a propositional symbol. The basic form of the bd-resolution rule is as follows

$$\frac{\psi_1 \wedge \omega, \psi_2 \wedge \neg\omega}{\psi_1 \wedge \psi_2},$$

where $\psi_1 \wedge \psi_2$ is the *bd-resolvent* of the parent formulae. Contrary to classical resolution, the disjunction of the parent formulae is a logical consequence of their bd-resolvent. BD-resolution can be directly applied to rule preconditions for completeness verification. For intuition, if the parent formulae constitute preconditions of some two rules, the above inference shows that they cover (more precisely: constitute a complete set of rules for) any environment satisfying their bd-resolvent $\psi_1 \wedge \psi_2$; in fact, in any such environment either ω or $\neg\omega$ must hold. *Backward dual resolution* constitutes a universal method of theorem proving [14, 13, 15, 16]; the particular forms of bd-resolution inference rule are also given there. Below, a definition of generalized bd-resolution for further use is given. For simplicity the problem of substitutions is omitted (it can be handled by appropriate use of the assumed (partial) interpretation **I**).

Definition 1. (Generalized bd-resolution)

Let $\psi_1 = \psi^1 \wedge \omega^1, \psi_2 = \psi^2 \wedge \omega^2, \ldots, \psi_j = \psi^j \wedge \omega^j$ *be some conjunctive formulae, such that* $\omega^1, \omega^2, \ldots, \omega^j$ *are certain formulae satisfying the so-called completeness condition of the form* $\omega \models_I \omega^1 \vee \omega^2 \vee \ldots \vee \omega^j$. *Then formula*

$$\psi = \psi^1 \wedge \psi^2 \wedge \ldots \wedge \psi^j \wedge \omega \tag{2}$$

will be called a generalized bd-resolvent *of* $\psi_1, \psi_2, \ldots, \psi_j$.

Again, the disjunction of parent formulae is a logical consequence of the bd-resolution under the assumed interpretation **I**.

Proposition 1

Let ψ *be a formula defined by (2), where the completeness condition is satisfied as above. Then*

$$\psi \wedge \omega \models_I \psi_1 \vee \psi_2 \vee \ldots \vee \psi_j.$$

This proposition provides in fact a possibility of specific "partial" resolution of the initial formulae. Roughly speaking, it provides the possibility of "gluing" several rules by finding a common specialization (a "cover") of selected components of the preconditions of mother formulae. For ω being the empty formula (always true) we have the strongest, pure form of bd-resolution.

In case one would like to keep the logical equivalence, we have the following proposition.

Proposition 2

Let ψ *be a formula defined by (2), where the completeness condition is satisfied as above, and let* ω *be logically equivalent to* $\omega^1 \vee \omega^2 \vee \ldots \vee \omega^j$. *Then* $\psi \wedge \omega$ *is logically equivalent to* $\psi_1 \vee \psi_2 \vee \ldots \vee \psi_j$.

Again, for ω being the empty formula (always true) we have the strongest, pure form of bd-resolution, where logical equivalence is kept between the premises (the input formulae) and the conclusion (the output). The inference rule given in Proposition (2) constitutes in fact a very general formulation of a principle allowing for reduction of detailed representations of objects to a simpler but equivalent representation; its variants are used in automata theory for combinatorial circuit simplification (at the level of propositional logic) and in machine learning.

For checking logical entailment and verification of completeness the following two theorems may be further useful. The first theorem [13, 16] provides the possibility to split the general problem for entailment checking among disjunctive formulae into a number of relatively simpler subproblems.

Theorem 1. Separability of logical entailment

Let Ψ *and* Φ *be two disjunctive normal formulae*:

$$\Psi = \psi_1 \vee \psi_2 \vee \psi_3 \vee \ldots \vee \psi_n \tag{3}$$

and

$$\Phi = \phi_1 \vee \phi_2 \vee \phi_3 \vee \ldots \vee \phi_m \tag{4}$$

where ψ_j *and* ϕ_j *are simple conjunctions of literals for* $i = 1, 2, 3, \ldots, n, j = 1, 2, 3, \ldots, m$. *Then* $\Phi \models \Psi$ *if and only if* $\phi_j \models \Psi$ *for any* $j \in \{1, 2, 3, \ldots, m\}$.

The given above theorem makes the initial problem a bit simpler – instead of checking if a normal formula of the form (3) follows from another normal formula of the form (4), it is always enough to check whether (3) logically follows from any of the simple formulae constituting (4). Thus the general case is split into a number (m) of simpler, separate problems, where the task is to check if a generalization of the following form takes place

$$\phi \models \Psi \tag{5}$$

and where Ψ is still a normal formula given by (3), but ϕ is a simple formula. In terms of geometric interpretation, the theorem simply means that a collection of sets is covered by another set if and only if every set of the collection is covered.

The second theorem provides a sufficient condition for proving logical entailment; it will be referred to as one concerning *direct generalization*, since it provides a condition (sufficient for generalization) which is possible to be checked almost directly, via a sequence of single matchings.

Theorem 2. Direct entailment

Let Ψ *and* Φ *be two disjunctive normal formulae, defined by equations (3) and (4), respectively. Now, if for any simple conjunctive formula* $\phi \in \Phi$ *there exists a simple conjunctive formula* $\psi \in \Psi$, *such that* $\phi \models \psi$, then $\Phi \models \Psi$.

COROLLARY 1. Let Ψ and Φ be two normal formulae given by (3) and (4), respectively. Then, if there exists some simple formula $\psi \in \Psi$ such that $\phi \models \psi$ for any simple formula $\phi \in \Phi$, then $\Phi \models \Psi$.

The logical background provides a formal framework for knowledge representation and inference in FOPC and similar languages. For the sake of the paper they constitute the background underlying an algebraic approach followed in the rest of this paper in case of tabular systems.

4. Tabular knowledge representation

In the following part the knowledge representation formalism will be restricted to *tabular systems*, both for data templates and rule-based systems. For the price of somewhat limited expressive power we gain simpler, algebraic approach, easier to understand and present in a graphical form. Moreover, analysis should be computationally more efficient. Certain practical extensions as explicit use of sets and intervals are also introduced.

Tabular representation of RBSs is aimed at simplified, readable knowledge representation in a uniform manner. Since the selected format follows the RDBS pattern, both notation (language) and certain operations can be used. It is mostly appropriate for rules of uniform structure. Further, a table provides natural frames for specification and analysis of rule-based systems. Thanks to its transparency the analysis becomes more intuitive. Moreover, the uniform tabular shape allows for introduction of concise matrix notation.

Consider constant, finite set of attributes of interest, $A = \{A_1, A_2, ..., A_n\}$. For any attribute A_i let D_i denote the domain of this attribute, $i = 1, 2, \ldots, n$. The domain can be a finite one, i.e. $D_i = \{d^1, d^2, ..., d^{m_i}\}$, or infinite, e.g. $D_i \subseteq \mathbf{R}$, where $\mathbf{R}$ is the set of real numbers. Attributes $A_1, A_2, ..., A_n$ denote some properties of interest, selected for expressing the domain knowledge of the analyzed system, when operating in a specific local context. They are aimed at representation of precondition knowledge for the rules. It is implicitly assumed that the attributes are independent on one another.

Further, let us consider a specific attribute H with the domain $D_H = \{h^1, h^2, ..., h^{m_h}\}$. This attribute is aimed at describing the output of the rules, e.g. hypotheses, decisions or other output values.

In the basic statement, the structure of a single condition (atomic formula) is as simple as

$$\psi_{ij} \equiv (A_i = d_j),$$

where A_i is one of the attributes and d_j is its current value. In a more advanced formulation one can admit the following form of atomic formulae

$$\psi_{ij} \equiv (A_i = t_j),$$

where t_j is either a precise value from the domain of attribute A_i ($t_j \in D_i$), or a subset of the domain of A_i ($t_j \subseteq D_i$), or just underscore ('_') to denote any (unspecified) value from the domain (as in logic programming the so-called *universal variable*).

Tabular system can be used both for data templates representation and for representation of rules. In the former case no decisional attribute H is present; the particular rows of the table represent formulas describing individual items or sets of them (if some attribute takes a non-atomic value). In case of rules, a specific column with the decision or conclusion attribute H is present, and the other attributes play the role of variables used in the specification of preconditions.

In general, the tabular representation follows the pattern of RDB systems. They can be used to represent both *Data* and *Knowledge* (*D&K*), so they provide a common representation for those two classes of information [19, 20].

4.1. What is data

An *atomic data item* is some piece of information represented in certain accepted language, and:

- as precise as possible (within the selected language),
- meaningful (having some interpretation),
- positive (no negation is used),
- unconditional.

Examples of data items include: propositional formula, ground atomic formula of predicate calculus, O-A-V fact, PROLOG fact, etc.

A *data item* is a conjunctive combination of atomic data items. Examples of data items include: ground conjunctive formulae, records (of atomic data items) in RDBs, etc.

Data are a collection of data items. Examples include RDB system tables, collections of ground conjunctive formulae, etc.

Data and knowledge can be differentiated by their intended interpretation: a data item (such as attribute value, record, table) is considered to be *data* if the main intended use of it is to provide static, detailed and precise image of some fragment of real world while a knowledge item (such as fact, simple conjunctive formula, DNF formula, and especially rules) is intended to provide more general knowledge defining universal or local properties of the world. From practical point of view, one can consider data to be the part of knowledge expressed with the *finest granularity* and unconditional.

4.2. What is knowledge

An *atomic knowledge item* is any data item and any more general elementary item of the accepted language, which:

- may contain variables/sets/intervals/structures (according to the selected language),
- meaningful (having some interpretation),
- positive or negative,
- perhaps conditional.

Examples of atomic knowledge items include: atomic formula of predicate calculus, extended O-A-V fact, PROLOG fact with variables, etc.

A *knowledge item* is a conjunctive combination of atomic knowledge items. Examples of knowledge items include: conjunctive formulae, records (of atomic knowledge items), etc.

Knowledge is a collection of knowledge items. Examples include relational database-like tables specifying data templates or rules, decision tables, collections of logical formulae, PROLOG programs.

If the specification contains variables (e.g. universally quantified, or defining some scope ones) or it is true only under certain conditions (e.g. takes the form of rules, allows for deduction or any other form of inference), then it should be normally considered to be *knowledge*. However, in the uniform, simplified model proposed in this paper explicit distinction is in fact not necessary. A RDB table would be normally considered as data, but it may be considered as most detailed knowledge as well. On the other hand, tabular system of data templates can be considered as extensional specification of data.

4.3. Rules in tabular systems

Any rule in the system is assumed to be of the form

$$r_i: (A_1 = t_1) \wedge (A_2 = t_2) \wedge \ldots \wedge (A_n = t_n) \rightarrow H = h_k$$

and it states that if the values of attributes A_1, A_2, ..., A_n are as specified above, then the output value of attribute H is h_k.

Now, taking into account the advantage of the uniform form of all the rules in the system, the set of rules can be specified in a transparent, tabular form (one resembling database format) as follows

$$\mathbf{B} = \begin{array}{c|ccccccc|c} \textit{rule} & A_1 & A_2 & \dots & A_j & \dots & A_n & H \\ r_1 & t_{11} & t_{12} & & t_{1j} & & t_{1n} & h_1 \\ r_2 & t_{21} & t_{22} & & t_{2j} & & t_{2n} & h_2 \\ \vdots & \vdots & \vdots & \dots & \vdots & \dots & \vdots & \vdots \\ r_i & t_{i1} & t_{i2} & \dots & t_{ij} & \dots & t_{in} & h_i \\ \vdots & \vdots & \vdots & & \vdots & & \vdots & \vdots \\ r_m & t_{m1} & t_{m2} & \dots & t_{mj} & \dots & t_{mn} & h_m \end{array} \qquad (6)$$

The above table represents m uniformly structured rules. Using the matrix notation one can also write $\mathbf{B} = [\mathbf{r}\ \Phi\ \mathbf{h}]$, here $\mathbf{r}$ is the leftmost column vector of rules names, Φ is the matrix of attribute values, and $\mathbf{h}$ is the rightmost column vector of conclusions (decisions). Further, the precondition matrix Φ can be (logically) written as $\Phi = \phi_1 \vee \phi_2 \vee \phi_3 \vee \dots \vee \phi_m$, or in the matrix form as

$$\Phi = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_m \end{bmatrix},$$

where $\phi_i = (A_1 = t_{i1}) \wedge (A_2 = t_{i2}) \wedge \dots \wedge (A_n = t_{in})$. From now on Φ will denote the logical formula corresponding to tabular form Φ and vice versa.

Alternatively, system **B** can be represented in a decision tree form. A schematic presentation is given in Figure 5.

A1	A2	A3	A4	A5	H
d11	d12	d13	d14	d15	h1
d21	d22	d23	d24	d25	h2
d31	d32				
d41					
d21	d22	d23	d24	d75	h7
d21	d22	d93	d94	d95	h9

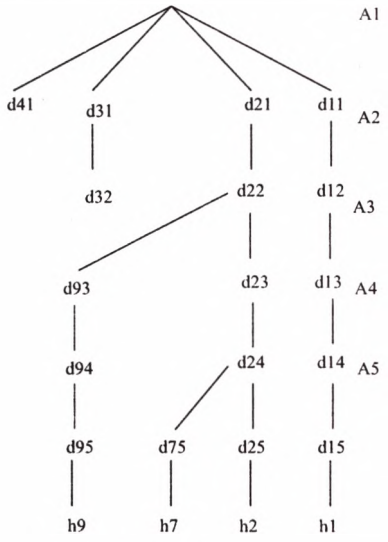

Fig. 5. The idea of tabular system and its tree-like form

The tree is one with n branching levels (a level corresponds to an attribute), the root is normally assigned the attribute A_1, and the branches below correspond to the appropriate values of this attribute (these may be both elementary items and complex ones, such as subsets, intervals, etc.). Similarly, any node of depth i corresponds to attribute A_i, and the branches below such node correspond to different values of this attribute. Finally, the leaves are assigned the h values, i.e. conclusions or decisions. Any path from root to a leaf node corresponds to one row of the appropriate matrix and thus it represents one rule of the system.

4.4. Basic operations and properties

The basic operations include classical logical connectives, i.e. conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$). It is important, however, to point to some issues specific to the discussed knowledge representation scheme as well as to some notational solutions.

First, any row of the matrix can be considered as a separate rule of the form

$$r_i : (A_1 = t_{i1}) \wedge (A_2 = t_{i2}) \wedge \ldots \wedge (A_n = t_{in}) \rightarrow H = h_i .$$

Note that, using the proposed representation, all the rule preconditions are of the same length and they all have identical structure (the order of attributes). Thus, comparison of the rules becomes very simple and intuitive.

Second, in the extended representation, the values of attributes can be expressed by subsets of the respective domains. This allows for direct representation of disjunction through the so-called *internal disjunction*, which is based on the following notational transformation

$$[(A_j = d_1) \vee (A_j = d_2) \vee \ldots \vee (A_j = d_i)] \equiv A_j = D',$$

where $D' = \{d_1, d_2, \ldots, d_i\}$. For simplicity, the notation $A_j = D'$ is used instead of $A_j \in D'$, which would probably be more correct from mathematical point of view. An analogous extension applies to interval representation. For example, $A_j = [a, b]$ means that all the values belonging to the interval $[a, b]$ are covered (both in case of discrete and continuous domain of the attribute).

Third, although negation is not represented explicitly in the tabular form, it can be represented implicitly due to the following equivalence

$$\neg(A_j = D') \equiv (A_j = \overline{D'}),$$

where $\overline{D_i}$ is the complement of D', i.e. $D' \cup \overline{D'} = D_j$, $D' \cap \overline{D'} = \varnothing$.

Further, note that there is also

$$(A_j = \_) \equiv (A_j = D_j)$$

having the consequence that $\neg(A_j = \_) \equiv (A_j = \varnothing)$.

Now, consider two rows of the system matrix, r_k and r_l defining data templates described with simple formulae ϕ_k and ϕ_l, respectively.

Let:

$$\phi_k = (A_1 = t_{k1}) \wedge (A_2 = t_{k2}) \wedge \ldots \wedge (A_n = t_{kn})$$

and

$$\phi_l = (A_1 = t_{l1}) \wedge (A_2 = t_{l2}) \wedge \ldots \wedge (A_n = t_{ln})$$

denote the appropriate formulae expressed within the attributive language. Since the terms t_{ij} representing the values of the attributes can be subsets (in fact, single elements, subsets or intervals) of the respective domains, the following proposition can be stated.

Proposition 3

Simple formula ϕ_k *subsumes simple formula* ϕ_l ($\phi_l \models \phi_k$) *if and only if* $t_{lj} \subseteq t_{kj}$ *for any* $j = 1, 2, ..., n$.

In the above proposition all the attributes are assumed to be different (no correlation) and, for simplicity, single values are considered equivalent to single element subsets of the respective domains. The check for subsumption is in fact stated in purely algebraic form.

Depending on the current needs, one can admit various notational possibilities, e.g. if the set of values for attribute A_i is ordered, one can use typical algebraic symbols such as $<, >, \leq, \geq$. For example, if $D_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ then $A_i = \{0, 1, 2, 3\}$ can be denoted as $A_i \leq 3$ and $A_i = \{3, 4, 5\}$ can be denoted as $3 \leq A_i \leq 5$, or rather $A_i \subseteq [3, 5]$ etc.

Now, let us consider formulae

$$\phi_1 = \phi^1 \wedge (A_i = t_1), \phi_2 = \phi^2 \wedge (A_i = t_2), ..., \phi_j = \phi^j \wedge (A_i = t_j).$$

Further, let $t = t_1 \cup t_2 \cup \ldots \cup t_j$, i.e. t covers all the possible values of attribute A_i specified by the sum of $t_1, t_2, ..., t_j$. The *completeness condition* is of the form $(A_i = t) \models_1 (A_i = t_1) \cup (A_i = t_2) \cup \ldots \cup (A_i = t_j)$; note that verification of completeness condition can be carried out with purely algebraic means.

Proposition 4

If the completeness condition is satisfied, the bd-resolvent takes the form

$$\phi^1 \wedge \phi^2 \wedge \ldots \wedge \phi^j \wedge (A_i = t).$$

Of course, if $t = D_i$, we have the most strong, pure bd-resolution; in such a case $(A_i = t)$ is omitted in the bd-resolvent. Roughly speaking, as before the key issue for successful "gluing" of simple formulae is that certain attribute takes some or all of its possible values across these formulae. If in at least one formula ϕ_i, the specification of the attribute value is empty (i.e. it is of the form $A_i = \_$), the completeness condition is also satisfied and the bd-resolvent defined as above can be generated.

Note that the most interesting case is if $\phi^1 = \phi^1 = \ldots = \phi^j = \phi$; in such a case the "reduction" of the formulae is especially efficient and elegant and the resulting bd-resolvent takes the simple form of $\phi \wedge A_i = t$ (or just ϕ *if* $t = D_i$).

Having in mind the geometric interpretation of D&K representation within tabular systems, one can consider algebraic operations on knowledge items, rows of the tables and even the tables themselves. An idea of such algebra is introduced in [21].

The negation (classical) of the matrix Φ is defined as follows. Since Φ can be interpreted as the DNF of the rules preconditions ($\Phi = \phi_1 \vee \phi_2 \vee \phi_3 \vee \ldots \vee \phi_m$), then the negation of Φ is the negation of a well-defined logical formula; it is denoted as $\neg\Phi$ or as $\overline{\Phi}$. Note that $\overline{\Phi}$ is also the complement of Φ as understood in RDBS theory.

Calculation of $\overline{\Phi}$ can be conveniently performed with use of the tree form of matrix Φ. The idea consists in considering the complete tree for the attributes A_1, A_2, ..., A_n, and marking the paths covered by Φ. Since the initial tree corresponds to the DNF of *true* (the canonical form of *true*), the unmarked paths corresponds to $\overline{\Phi}$. Further, it is not necessary to consider most elementary version of the tree, i.e. one with the branchings corresponding to singular values of any attribute – the appropriate, disjoint and complete subsets can be considered, which makes the calculation of the negation much more efficient. However, in general calculating the complement of a table may result in quite a computationally exhaustive procedure.

Below, we shall discuss some more interesting issues concerning the proposed taxonomy of anomalies. Some details concerning logical and algebraic definitions will be provided and approaches to build the appropriate checking procedures will be outlined.

5. Problems concerning adequacy of knowledge representation

5.1. Redundant rules

A rule is considered *redundant* if it succeeds in the same situation as another rule and both the rules have the same conclusions, e.g. [29, 39]. For example, two or more identical rules are redundant, but there may two other sources of redundancy. First, the rules like $p \wedge q \rightarrow h$ and $q \wedge p \rightarrow h$ are logically equivalent, and thus redundant. Elimination of such a case in tabular RBSs is straightforward, it follows from the imposed order of attributes, the same for all rules. Second, redundancy may be an effect of the possibility to express some facts in more than one way within the accepted language for knowledge representation. For example, the following rules are equivalent (and thus one of them is redundant), although they are not identical: $A \in [2, 5] \rightarrow h$, $A \in [2, 4] \cup [3, 5] \rightarrow h$. This and similar sources of redundancy are eliminated in tabular systems provided that simplification of attribute values to unique forms is carried out. Thus, for practical use it should be assumed that the values of attributes are reduced to a single item, like set or interval.

In general, redundant rules can be eliminated without influencing the overall behaviour of the KBS; thus the number of rules can be reduced. Consider general case of two equivalent rules; the rules can be replaced by a single rule according to the following scheme

$$\begin{array}{l} r\colon\ \phi \rightarrow h \\ r'\colon\ \phi' \rightarrow h' \\ \hline r\colon\ \phi \rightarrow h \end{array}$$

if it is possible to verify that $\phi' \models \phi$ and $\phi \models \phi'$ and simultaneously $h' \models h$ and $h \models h'$. The verification procedure is based on the application of Proposition (3). In practice, if the language of tabular system is restricted to single, discrete values of attributes and single sets (unique way of notation of attribute values), redundant rules form identical rows of the table.

5.2. Subsumption of rules

Let us consider the most general case of subsumption; some particular definitions are considered in [1, 29, 39]. A rule subsumes another rule if the following conditions hold:

- the precondition part of the subsuming rule is *weaker* (more general) than the precondition of the subsumed rule (the subsuming rule succeeds in more situations than the subsumed one),
- the conclusion part of the subsuming rule is *stronger* (more specific) than the conclusion of the subsumed rule (the subsuming rule provides more information than the subsumed one).

Particular instances of subsumption follow from holding either the first or the second condition, while keeping equivalence or identity of the other parts of the rules. The case of redundant rules [1, 29, 39], i.e. when one of them succeeds in the same situation as another rule and both the rules have the same conclusions can be considered as the simplest case of subsumption.

Let the rules r and r', satisfy the following assumption: $\phi' \models \phi$ and $h \models h'$. The subsumed rule can be eliminated according to the following scheme

$$\begin{array}{l} r\colon\ \phi \rightarrow h \\ r'\colon\ \phi' \rightarrow h' \\ \hline r\colon\ \phi \rightarrow h \end{array}$$

For intuition, a subsumed rule can be eliminated because it produces weaker results and requires stronger conditions to be satisfied; thus any of such results can be produced with the subsuming rule. Using matrix notation we have

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

The condition for subsumption in case of the above tabular format takes the form $t_j' \subseteq t_j$, for $j = 1, 2, \ldots, n$ and $h' \subseteq h$.

For example, in the following tabular system the first rule subsumes the second one

rule	A_1	A_2	A_3	A_4	H
r	7	[2, 9]	[3, 5]	$\{r, g, b\}$	$\{a, b, c\}$
r'	7	[3, 5]	4	$\{b, r\}$	$\{a, c\}$

The check is based on pure algebraic test of Proposition 3. In general case, the check for subsumption may require $m(m-1)/2$ comparisons among rules (where m is the number of rules in the analyzed table), however most of them will fail quickly. In large tables, the rules may be further structured w.r.t. partition of certain attributes, so that subsumption may be checked only within smaller subgroups of rules.

6. Issues concerning completeness of knowledge representation

For intuition, a RBS is considered to be *complete* if there exists at least one rule succeeding for any possible input situation. In literature [1, 29, 39] there are two basic approaches to completeness verification. The most popular one consists in exhaustive enumeration of possible input data and systematic inspection of a given set of rules versus a table containing all possible parameters and conditions combinations. This kind of approach can be called an *exhaustive completeness check* [1]. Some examples of this approach are presented in [29, 39]. The other approach consists in a run-time validation of the expert system with use of selected set of test cases [40]. Selected test problems should also provide an exhaustive list of possible cases. Some other approaches of this kind are also discussed in [1]. In [14, 13, 16, 17] a more general, first-order logic based approach is presented. The approach does not require exhaustive enumeration and testing of possible cases; instead, a proof-like procedure based on backward dual resolution is put forward.

In the following subsection logical (total) completeness, specific (partial) completeness and detection of missing rule preconditions will be discussed in turn. In all subsections the same set of rules will be considered, i.e.:

$$\begin{array}{llll} r_1: & \phi_1 & \rightarrow & h_1, \\ r_2: & \phi_2 & \rightarrow & h_2, \\ \vdots & \vdots & \vdots & \vdots \\ r_m: & \phi_m & \rightarrow & h_m. \end{array}$$

or equivalently, given by **B** (equation (6)). Further, in fact, only preconditions of the rules are of interest for completeness verification.

6.1. Logical completeness of a rule-base system

The approach proposed here comes from purely logical analysis [14, 22, 18] and does not require exhaustive enumeration of possible cases; instead a proof-like, algebraic procedure is put forward.

Consider the joint disjunctive formula of rule precondition of the form

$$\Phi = \phi_1 \vee \phi_2 \vee \ldots \vee \phi_m.$$

The condition of logical completeness for the above system is

$$\models \Phi \tag{7}$$

which simply means that Φ is a tautology. However, recall that in tabular RBSs the negation is usually not present explicitly. This means that no tautology of the type $\alpha \vee \neg\alpha$ can be present, and thus reduction to such type of tautology is not possible. Instead, reduction taking into account limited domains of system attributes can take place. However, contrary to the reduction operation aimed at minimizing the number of rules (and thus applied only

to rules having identical conclusions), this time reduction can be applied to all the rule preconditions, disregarding their conclusions.

Thus, the purely logical condition (7) can be replaced by a practical condition of the form

$$\mathrm{Red}(\Phi) = \square \tag{8}$$

where $\mathrm{Red}(\Phi)$ is the maximal reduction of table Φ and $\square$ denotes an empty table (for reduction of tabular systems see some following section).

6.2. Specific completeness

In most of practical cases the analyzed system may be logically incomplete. It can be designed to work in some limited context C. The restrictions may follow from physical limitations on system parameters, specific "local" character of the situation to be served, limited knowledge of system designer or may be the consequence of physical infeasibility of certain inputs.

Let C denote the operating context for the above system. The *specific* (partial) *completeness condition* can be stated as follows

$$C \models \Phi \tag{9}$$

where Φ denotes the disjunction of precondition formulae. Again, from logical point of view the verification could be purely logical and it could be proved by means of automated deduction (e.g. by direct use of bd-resolution [13, 14]). However, taking into account the tabular knowledge representation, an algebraic method for specific completeness verification would be suggested.

First, instead of considering m rule preconditions, one can apply maximal reduction of the formulae; most likely partial reduction will be applied, and the operation will result with a smaller set of k rules. Formally, we have $\mathrm{Red}(\Phi) = \Phi^*$, where Φ^* is the reduced k-rows table. As above, reduction is carried out disregarding the rule conclusions (which are different from one another), i.e. *any* two (or more) precondition formulae can be selected for reduction.

Now, three outputs are possible: the resulting table can be empty (no problem, full logical completeness holds), it can have exactly one row, or it can have k rows, where $k > 1$. The case when $k = 1$ is again simple: the completeness check expressed by (9) can be replaced by

$$C \models \Phi^* \tag{10}$$

Since Φ^* is a single-row formula ($k = 1$) the check of (10) is equivalent to subsumption checking; thus it can be performed with use of Proposition 3. If C consists of a several row table (logically: a disjunction of several simple formulae), than the subsumption check must hold for any row of C.

The most complex is the case of Φ^* being a table of more than one row (and perhaps further irreducible). In such a case, one has to make use of the theorem on separability of logical entailment (Theorem 1) or the theorem on direct entailment (Theorem 2). Let C be the disjunction of the form $C = c_1 \vee c_2 \vee \ldots \vee c_c$, where c_i denotes the i-th row of C, $i = 1, 2, ..., c$.

The sufficient condition for partial completeness takes the form

$$\forall i \in \{1, 2, ..., c\} \; \exists j \in \{1, 2, ..., k\}: c_i \models \phi_j^* \tag{11}$$

where ϕ_j^* denote the *j-th* row of table Φ^*. The geometric interpretation of condition (11) is straightforward – any elementary context c_i must be covered by precondition formula of some rule.

Note that during reduction of the formulae also more specific formulae can be used if necessary (i.e. subsumed ones). This may be crucial if some two formulae taken together cover certain formula c_i, but they are irreducible in a direct way (e.g. some of their conditions are different; in such a case specialization of some conditions may be necessary to obtain identical parts of the formulae).

An alternative way may consist in splitting the context defining formula C into smaller parts, and this seems to be more straightforward solution from computational point of view. In such a case, the separation values of selected attributes should be carefully selected so as to avoid too detailed split. A reasonable way may consists in selecting the values occurring in Φ^*, i.e. ones which still exist in the description language after maximal possible reduction. This means that characteristic values of attributes can guide the split operation.

In practice, the splitting operation of every simple formula can be performed by the procedure outlined below:

- for any attribute A consider all its values (sets or intervals) appearing both in c_i and the table Φ; denote the values as $V_1, V_2, ..., V_m$;
- generate partition of the domain of A_i as a collection of sets $\mathbf{B} = \{B_1, B_2, ..., B_n\}$ (the so-called *blocks*), such that $B_i \cap B_j = \varnothing$ and $B_1 \cup B_2 \cup \ldots \cup B_n = D_i$, the biggest ones but such that for every V_i, $i = 1, 2, ..., m$ there exist $B^1, B^2, \ldots, B^k \in \mathbf{B}$ such that $V_i = B^1 \cup B^2 \cup \ldots \cup B^k$, i.e. every value V_i can be expressed as a sum of some selection of blocks of **B**; generation of **B** for intervals can be done by finding all intersections of the type $V_i \cap V_j$, $i, j \in \{1, 2, ..., m\}$ but for nominal sets also intersections of three, four, five, etc. sets should be considered as possible candidates (any superset is immediately eliminated);
- split every value of selected attribute A_i in c_i into largest possible sets being sums of blocks, say $B^1, B^2, \ldots, B^{c_i}$ and replace a particular row c_i in with appropriate *ci* rows;
- repeat the procedure for every attribute A in c_i;
- for the resulting table being the output of the procedure apply the check following from condition (11);
- the system is specifically complete if every row is covered by some row of Φ.

To illustrate the idea one can employ a simple graphical interpretation for visualisation of the completeness check. Every rule – being a row of the table – forms a path covering some selection of attribute values in the n-th dimensional attributes space (see Fig. 6). The values of specific attributes generate partitions of the domains of attributes.

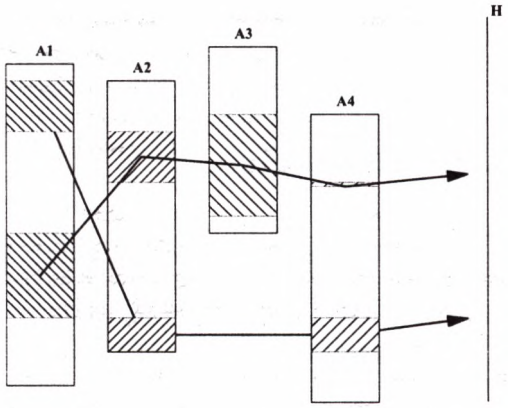

Fig. 6. The concept of attribute space and interpretation of rule as a path covering selection of values

Further, note that after performing the reduction of rule preconditions (if possible, a maximal one), the rules tends to cover bigger areas of selected attributes. In some best cases, certain attributes may also turn out to be unimportant, since all the values of them are covered.

Note that in case of finite domains, the split could be done by taking into account single elements of the attribute domains; this, however, would be equivalent to exhaustive enumeration of the input cases. The power of the algebraic approach consists in operating on sets of input cases which can be achieved by a high-level split (not too detailed one).

Finally, let us assume that the values of attributes to be covered were split w.r.t the partition, e.g. given by boundary values induced by the values appearing in rule preconditions and the constraints, after the rules have been maximally reduced (in fact, the preconditions of rules were reduced to a minimal number of maximally general formulae). Now, the check for completeness consists in tracing if all the combinations resulting from the split of the context constraint is covered by some of the "thickened" paths generated by the reduced set of rules.

The graphical interpretation is in the Figure 7.

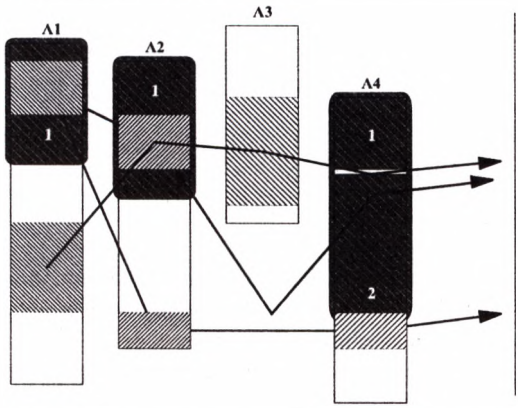

Fig. 7. The idea of covering the attribute space with more general, reduced set of rules

In general, the check may suffer from combinatorial explosion; however, due to operating on blocks (sets, intervals) rather than on individual values, the number of detailed checks is significantly reduced with respect to the methods based on exhaustive enumeration.

6.3. Missing precondition identification

Determination of missing preconditions can be based on the above scheme for completeness verification. As before, assume that C is a tabular form of a formula specifying the required area to be covered with preconditions of the rules. Further, let $\mathrm{Red}(\Phi)=\Phi^*$, denote the maximally reduced table of formulae preconditions. Considering any of the attributes, say A_i, let V_i denote the set of characteristic values of this attribute, still occurring in the preconditions after reduction. These may be boundary values defining boundaries of intervals (in case of ordered attributes) or simple subsets of attribute domain (in case of attributes for which their domain is an unordered set). Note that the stronger reduction is possible, the less characteristic values are still left.

Now let us use the values V_i to split the domain of any attribute into (possibly maximal) intervals or subsets. In case two intervals or subset overlap, a third interval or subset can be distinguished, so that the split forms a partition – no two intervals/subsets overlap, and their sum gives the complete domain.

A simplified graphical interpretation of the idea is presented in Figure 8.

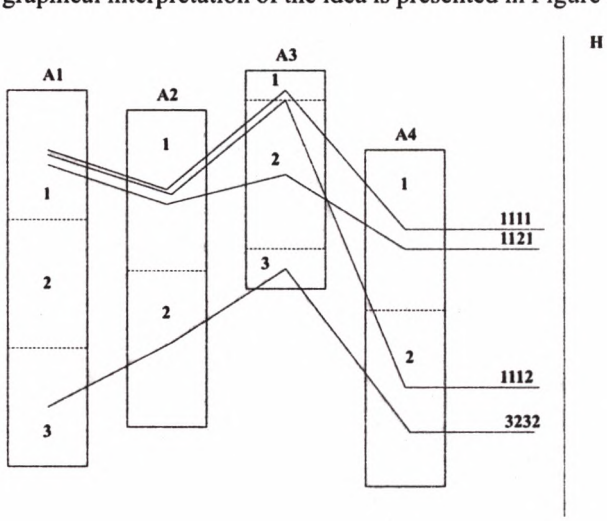

Fig. 8. Systematic checking for completness with reduced rules in the space of attributes

The subareas of the attributes resulting after the split can be numbered in an arbitrary way; thus, to any path corresponding to a rule there corresponds a sequence of numbers listing the covered areas for subsequent attributes. If attribute A_i is partitioned into k_i subareas (and there are n attributes), then any sequence is of the form $\alpha^1\alpha^2\ldots\alpha^n$ (of length n), where $\alpha^i \in \{1, 2, \ldots, k_i\}$; for some eliminated attribute $A_j\,\alpha^j = \_$. There are also $k_1\,k_2\ldots k_n$ possible sequences.

Note that certain preconditions can cover more than one such sequence; covering a sequence eliminates it from further consideration. The test for covering is straightforward – it consists in a check for subsumption. All the sequences which are not covered by some rule identify potential gaps in the system, i.e. define the uncovered inputs (incompleteness).

The outlined method for determining missing preconditions (possible for further analysis) is constructive, but in case of larger systems may suffer from combinatorial explosion. This effect, however, is significantly minimized here in comparison to the approaches based on direct exhaustive enumeration. First, maximally reduced form of precondition table Φ^* is used only; this restricts the division of any attribute to limited number of areas, and not to all its possible values. Second, the method is aimed at analyzing a local system, operating in some well-defined, rather narrow context C. This is possible thanks to introduction of hierarchical structure of the system. Third, splitting C is necessary only w.r.t. its rows which are not initially covered by Φ^*. Finally, for reduced preconditions where the values of certain attributes are unimportant (the attributes are eliminated), the number of potential paths covered is multiplied by the factor equivalent to the number of areas to which the attribute was split. From logical point of view, the way of determining incompleteness consists in generating the $\neg(\Phi^*)$ formula.

7. Issues concerning internal consistency

In this section the problem of determinism and two following issues, i.e. the one of conflict and inconsistency are discussed.

7.1. Determinism

A set of rules is *deterministic* iff no two different rules can succeed for the same state. A set of rules which is not deterministic is also referred to as *ambivalent*.

The idea of having a deterministic system consists in a priori elimination of "overlapping" rules, i.e. ones which operate on a common situation. The aim of analysis is obvious – to detect (distinguish) the case of two or more rules applicable in the same situation.

Consider the following two rules:

$$r: \ \phi \ \rightarrow \ h,$$
$$r': \ \phi' \ \rightarrow \ h'.$$

From purely logical point of view the system is deterministic iff the conjunction of the precondition formulae $\phi \wedge \phi'$ is unsatisfiable. For certain technical systems it is enough to check that such a conjunction is unsatisfiable under some specific interpretation referring to the domain of interest [13]. A typical example is of the form $\phi = switch(on)$ and $\phi' = switch(off)$, and obviously the formula $\phi \wedge \phi'$ is false under the intended interpretation – the *switch* can be either *on* or *off*.

Consider now the case of attribute knowledge representation, as below:

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'

Calculation of $\phi \wedge \phi'$ is straightforward: for any attribute A_j there is an atom of the form $A_j = t_j$ in ϕ and $A_j = t'_j$ in ϕ', $i = 1, 2, \ldots, n$. 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 $t_j \cap t'_j = \varnothing$), then the rules are disjoint. The check is to be performed for any pair of rules.

7.2. Conflict and inconsistency

From practical point of view deterministic systems are easier for implementation. In case of indeterministic system there may be the case that two or more rules are simultaneously applicable. It is the problem then of the so-called *conflict resolution mechanism* to select a single rule to be fired. Note that if a system is deterministic, no conflict resolution mechanism is necessary. On the other hand, in certain systems indeterminism is inherent in the set of rules, while conflict situations are to be solved with appropriate inference control mechanism. In design of knowledge rule-based systems one can encounter further theoretical problems; two most important ones following from the lack of determinism are as follows (see also [29, 39]).

1) Conflict – two (or more) rules are applicable to the same input situation but the results are conflicting (under the assumed interpretation).
2) Inconsistency – here understood as logical inconsistency (unsatisfiability under any interpretation).

Note that problems of *conflicting* and *inconsistent* rules [29, 39] are specific cases of indeterminism. In tabular systems with no explicit negation purely logical inconsistency cannot occur; it always follows from the intended interpretation and thus it falls into the class of conflicts.

The case of conflicting rules is a potential source of errors, e.g. ambiguous, and therefore unpredictable behaviour. Conflicting rules, however, are a subclass of indeterministic ones; thus, any pair of such rules will be eventually discovered after checking for determinism; they should be further analyzed by domain experts. This is so, since different conclusions of two overlapping rules do not necessarily mean "real conflict", i.e. both suggested solutions can be admissible. This is the typical case of *Decision Support Systems* suggesting one or several solutions valid for certain situation. Depending on the inference strategy, either one of them (selected in an arbitrary way or according to some predefined strategy) or even all of them can be applied. On the other hand, a real conflict exists usually in case of unique decision to be undertaken, e.g. if some resources are indivisible or some variables can be assigned a unique value only.

8. Issues concerning external consistency

External consistency understood as satisfaction of external local constraints, as well as satisfaction of external global constraints, can be defined with use of constraint definition formula Γ. In the tabular formalism it takes form of a tabular system. The constraint satisfaction condition is of the form $\Gamma \models \Phi$. Note that, checking for satisfaction of external consistency is – from logical point of view – analogous to checking for completeness. Thus the same approach as in the case of specific completeness can be applied.

9. Issues of minimal knowledge representation

Minimization of knowledge representation can be achieved by elimination of redundant and subsumed rules and by appropriate joining together selected rules which are in certain sense "complementary". The main idea of the second case is based on the principles of backward dual resolution and is referred to as *reduction of rules*.

9.1. Total and partial reduction

Reduction of rules is an operation similar to finding minimal representation for propositional calculus formulae or boolean combinatorial circuits. The main idea of reduction of rules is to minimize the number of rules without influencing the potential capabilities of the system for inferring new knowledge. An interesting possibility consists in replacing a number of rules having the same conclusions with a single equivalent rule.

Consider k rules with the same conclusion, such that their preconditions differ only with respect ω_i, $i = 1, 2, \ldots, k$, where $\omega_i = (A_j = t_{ij})$ defines the value of the same single attribute A_j. Assume that the following completeness condition holds, i.e. $\models \omega_1 \vee \omega_2 \vee \ldots \vee \omega_k$. With respect to Proposition 4 the following reduction scheme can be applied:

$$
\begin{array}{lllll}
r^1: & \phi \wedge \omega_1 & \rightarrow & h \\
r^2: & \phi \wedge \omega_2 & \rightarrow & h \\
\vdots & \vdots & \vdots & \vdots \\
r^k: & \phi \wedge \omega_k & \rightarrow & h \\
\hline
r: & \phi & \rightarrow & h
\end{array}
$$

For intuition, the preconditions of the formulae are replaced by a joint condition representing the disjunction of them; roughly speaking, the sets described with the preconditions are "glued" together into a single set. The resulting rule is logically equivalent to the set of initial rules.

Using the tabular knowledge representation, reduction takes the following form:

rule	A_1	A_2	...	A_j	...	A_n	H
r^1	t_1	t_2	...	t_{1j}	...	t_n	h
r^2	t_1	t_2	...	t_{2j}	...	t_n	h
⋮	⋮	⋮		⋮		⋮	⋮
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_n	h

provided that $t_{1j} \cup t_{2j} \cup \ldots \cup t_{kj} = D_j$. Of course, the rules r^1, r^2, ..., r^k are just some selected rows of the original table containing all the rules. If **B** denotes the original table, its (maximally) reduced form will be denoted as Red(**B**) = **B***.

Note that some strong requirement for reduction consists in the fact that the values of attributes other than the reduced one must be identical. This is important if one insists on preserving logical equivalence. In some cases, however, it may be of interest to produce a new rule more general with respect to the selected attribute (i.e. making use of gluing its partial values), while admitting a "slight" restriction concerning the rest of the preconditions. On the base of Definition 2, the following form of reduction can also be proposed:

$$\begin{array}{lll} r^1\colon & \phi^1 \wedge \omega_1 & \rightarrow h \\ r^2\colon & \phi^2 \wedge \omega_2 & \rightarrow h \\ \vdots & \vdots & \vdots \ \vdots \\ r^k\colon & \phi^k \wedge \omega_k & \rightarrow h \\ \hline r\colon & \phi^1 \wedge \phi^2 \wedge \ldots \wedge \phi^k & \rightarrow h \end{array}$$

This may be reasonable provided that formula $\phi^1 \wedge \phi^2 \wedge \ldots \wedge \phi^k$ is useful and not too restrictive, and may be applied in completeness verification. Note that full logical equivalence may be no longer preserved, however, disjunction of the preconditions of mother rules follows from the precondition of the reduced formula. If the resulting reduced rule can be applied, then at least one of the rules above can be applied as well, and the produced conclusion is of course identical.

In general, the reduction can be total (maximal) or partial, depending on the needs. The total reduction may result in several, different results, sometimes hard to compare. Partial reduction can be stopped at certain point, e.g. when the canonical form is generated [21], which allows for easy comparison of results and further algebraic operations.

9.2. Specific partial reduction

In certain cases a complete reduction as shown above may turn out to be inapplicable; however, it may still be possible to simplify the set of rules if only the sub-formulae ω_i, $i = 1, 2, \ldots, k$, can be replaced with a single equivalent formula. In our case, a collection of certain elements can be always replaced by a subset containing all of them (and nothing more), while a collection of intervals can be replaced with their sum (which may be a single, convex interval). In general, let us assume that $\omega_1 \vee \omega_2 \vee \ldots \vee \omega_k \models \omega$, and $\omega \models \omega_1 \vee \omega_2 \vee \ldots \vee \omega_k$. The reduction can take the following logical form:

$$\begin{array}{lll} r^1\colon & \phi \wedge \omega_1 & \rightarrow h \\ r^2\colon & \phi \wedge \omega_2 & \rightarrow h \\ \vdots & \vdots & \vdots \ \vdots \\ r^k\colon & \phi \wedge \omega_k & \rightarrow h \\ \hline r\colon & \phi \wedge \omega & \rightarrow h \end{array}$$

Formula ω must be expressible within the accepted language. In case of a single attribute the internal disjunction can be applied just by specifying the appropriate subset.

Using the tabular knowledge representation, partial reduction takes the following form:

rule	A_1	A_2	...	A_j	...	A_n	H
r^1	t_1	t_2	...	t_{1j}	...	t_n	h
r^2	t_1	t_2	...	t_{2j}	...	t_n	h
⋮	⋮	⋮		⋮		⋮	⋮
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 $t_{1j} \cup t_{2j} \cup ... \cup t_{kj} = t$. As above, the rules r^1, r^2, ..., r^k are just some selected rows of the original table containing all the rules. Moreover, we extend the notation on partial reduction as well, i.e. if **B** denotes the original table, its (maximally) reduced form using also partial reduction whenever possible will be denoted as $\text{Red}(\mathbf{B}) = \mathbf{B}^*$ (or $\text{Red}(\Phi) = \Phi^*$).

Note also that in the case of partial reduction generation of more restrictive formulae is possible in a way analogous to the case of pure reduction. The transformation takes the form:

$$\begin{array}{ll} r^1\colon \phi^1 \wedge \omega_1 & \rightarrow h \\ r^2\colon \phi^2 \wedge \omega_2 & \rightarrow h \\ \vdots \quad \vdots & \vdots \quad \vdots \\ r^k\colon \phi^k \wedge \omega_k & \rightarrow h \\ \hline r\colon \phi^1 \wedge \phi^2 \wedge ... \wedge \phi^k \wedge \omega & \rightarrow h \end{array}$$

and it may be reasonable if the formula $\phi^1 \wedge \phi^2 \wedge ... \wedge \phi^k \wedge \omega$ is not too restrictive. The idea of canonical form applies as well to specific partial reduction.

10. Supporting design of tabular rule based systems

In general, the proposed approach can be used to support design of tabular system in the following two ways:

1) static, ex-post verification of the partially or completely developed system in order to suggest improvements; such a procedure can be repeated in several cycles;
2) on-line, dynamic support of design by instant detection of anomalies occurring during the design process and generation of appropriate information for the developer of the system.

The first approach is rather obvious and may be applied in a rather straightforward way; in fact, to certain degree it is a part of todays practice in development of rule based systems. However, it suffers from obvious disadvantages, including repetition of the design procedure. Moreover, errors in design discovered after completing edition of the knowledge base may result in hard to deal with problems, and improvement of the rules may cause new errors to appear.

The second approach, proposed initially in [11] and [12] seems to be much more attractive and efficient. However, it requires development of a special software tool supporting the design.

10.1. An intuitive introduction

For intuition, let us consider a possible design process of a simple temperature control system. This will be a rule based system capable of applying the two-valued and three-valued switch type algorithm (relay type) if stabilisation of some standard temperature is required, or controlling the temperature towards its maximal or minimal value if required. All of this takes place only if the control of temperature is required.

In order to simplify systematic design we shall use a tree-like graphical representation for displaying combinations of various conditions. The idea is similar to standard binary *decision trees*; however, the proposed construction is a bit more general and allows for use of any finite number of branching conditions, description of the branching conditions with first order level languages, and hierarchical design. The idea of such a tree corresponding to a part of the design process is presented in Figure 9.

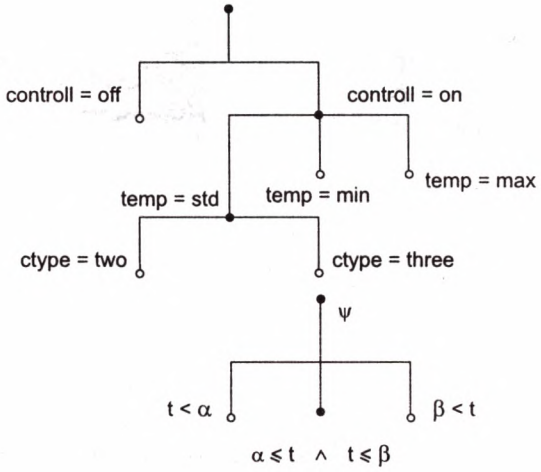

Fig. 9. An example ψ-tree for design

The design process can proceed as follows. We start with the root node and select some predicate (or more complex set of formulae) to define the initial branching. A reasonable idea is to select a formula of relatively high level, i.e. defining as general conditions as possible, and providing contexts for further specification. In our case let it be the first condition specifying if the control is required (control = on) or not (control = off). Now, leaving the left branch temporarily pending, we further develop the right branch; note that now we are in the context defined by the formula control = on. The next selected formula may concern the mode of temperature regulation, i.e. if some steady temperature is required (temp = std), the minimal temperature should be achieved (temp = min) or the maximal one (temp = max). Again, leaving the two right branches temporarily pending (possibly for further continuation of design), we develop the left branch. Note, however, that now the context is defined by the conjunction of formulae assigned to the branches on the path from the root node to the current leaf node; in our case the formula is of the form control = on ∧ temp = std. Now we have to select the controller type. Assume there are given two possibilities: a two-level

one (ctype = two) and a three-level one (ctype = three). We can proceed with the design in an analogous way, but a reasonable idea may be to split the tree into the mother initial tree and several subtrees whose roots are joined to the leaves of the mother tree. In Figure 9 we leave the context defined by ψ = control = on $\wedge$ temp = std $\wedge$ ctype = three and start development of a new, smaller child-tree for the context defined by ψ. It will be referred to as a ψ-tree, and it describes the design process for the context ψ. In our case the branching condition refers to the current temperature, and we have three possibilities:

1) $t < \alpha$,
2) $(\alpha \leq t) \wedge (t \leq \beta)$,
3) $\beta < t$.

The final step is to assign to any leaf node a control action to be undertaken. For example, for the final result of the partial design process presented above (there are still pending leaf nodes to be developed), the following set of rules may be generated:

$r1$: $\psi \wedge (t < \alpha) \rightarrow$ set(heat, on),
$r2$: $\psi \wedge (\alpha \leq t) \wedge (t \leq \beta) \rightarrow$ set(heat, off), set(cool, off),
$r3$: $\psi \wedge (\beta < t) \rightarrow$ set(cool, on).

Obviously, when operating in the context defined by ψ, the preconditions can be simplified to the formulae in parentheses.

Having in mind the above example it would be worthwhile to analyze some steps of the design process. First, the formula defining the branching condition is selected to be possibly general and reasonable for the current design context; although there is no unique, well defined selection procedure, this seems to be a reasonable, expert- acquired way of design procedure. Second, defining the precise branching alternatives we take care to *cover* all the existing possibilities; from logical point of view, under the assumed technical interpretation **I**, the following conditions are satisfied:

$\models_{\mathbf{I}}$ control = off $\vee$ control = on,
$\models_{\mathbf{I}}$ temp = std $\vee$ temp = min $\vee$ temp = max,
$\models_{\mathbf{I}}$ ctype = two $\vee$ ctype = three,
$\models_{\mathbf{I}}$ $(t < \alpha) \vee [(\alpha \leq t) \wedge (t \leq \beta)] \vee (\beta < t)$.

In other words, we attempt at satisfying the local *completeness condition* when defining branching possibilities. Third, we also try to define them so that the alternative formulae do not overlap, i.e. no two of them can be satisfied at the same time. In fact, from logical point of view we have:

$\not\models_{\mathbf{I}}$ control = off $\wedge$ control = on,
$\not\models_{\mathbf{I}}$ temp = std $\wedge$ temp = min,
$\not\models_{\mathbf{I}}$ temp = min $\wedge$ temp = max,
$\not\models_{\mathbf{I}}$ temp = std $\wedge$ temp = max,
$\not\models_{\mathbf{I}}$ ctype = two $\wedge$ ctype = three,
$\not\models_{\mathbf{I}}$ $(t < \alpha) \wedge (\alpha \leq t) \wedge (t \leq \beta)$,
$\not\models_{\mathbf{I}}$ $(\alpha \leq t) \wedge (t \leq \beta) \wedge (\beta < t)$,
$\not\models_{\mathbf{I}}$ $(t < \alpha) \wedge (\beta < t)$.

Such procedure is justified since we would like to obtain a *complete* and *deterministic* set of rules. For intuition, as the precondition formulae are defined by conjunctions of formulae assigned to branches on paths from the root to the leaf nodes, at any node we cover all the reasonable branching possibility; hence completeness is assured. Simultaneously, no two "paths" can be validated as true at the same time, since they must be different at at least one node and due to exclusion of the branching conditions they cannot be true at the same time. These problems are referred to in some details throughout this paper.

Finally, by splitting the design tree into a mother tree and several child-subtrees we achieve the possibility of performing hierarchical, two-level design. First we design the meta-rules, allowing for the so-called *context-switching*. Executing such a rule defines a context (set of states) and a subset of all the rules such that only rules of the subset can be used for the currently selected context. Then we define the sets of rules for any specific contexts by development of a Ψ-tree for any context defined by Ψ. Of course, this procedure can be turn to several-level ones as well.

10.2. The Ψ-trees for design support

In this section basic concepts concerning a specific form of semantic trees (to be called Ψ-trees) used for design of complete sets of rules for some context Ψ are presented.

Let N denote a set of nodes. We assume the common definition of a tree (there is a distinguished *root node* with no ancestor and any other node n' belonging to the tree has exactly one ancestor). A tree will be denoted with t, and $t(N)$ will denote a tree build from nodes belonging to N; the set of all nodes of tree $t(N)$ will be denoted with $N(t)$. The root node of any tree t will be denoted with $root(t)$.

Now let FOR denote a set of considered formulae, and let Ψ denote a distinguished formula. By **I** we shall denote the set of intended interpretations.

Definition 2

Let f denote any mapping of the form $f: N \rightarrow FOR \cup \{\Psi\}$. A Ψ-tree is any finite tree $t(N)$ satisfying the following auxiliary conditions:

- $f(root(t(N))) = \Psi$,
- *for any* $n \in N(t)$, $f(n) \in FOR$.

For intuition, a Ψ-tree represents different (some or all) possible branchings into different more detailed situations of the analyzed system for some stable context defined by Ψ. Any node $n \in N(t)$ represents a situation described with conjunction of all the formulae assigned to the nodes belonging to the path from $root(t)$ to n (including Ψ and $f(n)$). Such formulae will be denoted with lower-case ψ or ϕ. Roughly speaking, going down the tree along some path beginning in the root node we add some new requirements (defined by the formulae met on the way) to be satisfied in the described situation. Obviously, the greater the depth of a node is, the "more particular" situation is described by the appropriate formula determined by the path from root to this node. Finally, the paths ending with the leaf nodes determine a set of "most detailed" situations within the context situation defined by Ψ.

A Ψ-tree can be used as a basic tool for guiding the generation of rules precondition by domain expert. The idea is to develop such a tree in a top-down mode in a systematic way.

Alternative conditions referring to some predicate or formula are represented by branching in the tree. Along the paths from the root to leaves precondition formulae (conjunctions of the appropriate conditions) are synthesized.

Now, the most important idea consists in such a generation of the Ψ-tree that the set of the most detailed situations described by all the formulae assigned to paths ending with leaf nodes "cover" the initial situation defined by the context formula Ψ. With respect to this problem the following definition is proposed.

Definition 3

Let $\psi_1, \psi_2, \ldots, \psi_k$ *denote all the conjunctive formulae assigned to all the paths from* root(*t*) *to the leaf nodes of some* Ψ*-tree t. The tree is referred to as a complete* Ψ*-tree iff* $\Psi \models_I \psi_1 \vee \psi_2 \vee \ldots \vee \psi_k$.

Note that this definition assures in fact the specific completeness of the system (within the context defined by Ψ. Further, the aim of the above definition is obvious – any object of the considered system belonging to the context defined by Ψ will be eventually covered by the subsets of the universe (more precisely, by at least one of them) defined with formulae $\psi_1, \psi_2, \ldots, \psi_k$. This is stated with the following proposition justifying the use of complete Ψ-trees for design of rule preconditions.

Proposition 5

A complete Ψ*-tree assures specific completeness of the developed rule-based system with respect to* Ψ.

With reference to the mentioned tool for proving completeness (i.e. bd-resolution), the goal of developing Ψ-trees is obvious; a structure of a complete Ψ-tree provides a straightforward strategy of bd-resolution theorem proof for the condition defining completeness, i.e. iff $\Psi \models_I \psi_1 \vee \psi_2 \vee \ldots \vee \psi_k$. The following proposition provides a sufficient condition for completeness of any considered Ψ-tree.

Proposition 6

A Ψ*-tree is complete if the following conditions hold:*

- *for any non-leaf node n being an immediate ancestor (parent) of some leaf nodes and a conjunctive formula* ψ *determined by the path from root to this node there is*

$$\psi \models_I \psi^1 \vee \psi^2 \vee \ldots \vee \psi^j \tag{12}$$

  i.e. the formulae satisfy the completeness condition (12) and therefore it is possible to resolve $\psi^1, \psi^2, \ldots, \psi^j$ *generating formula* ψ *and,*

- *the generated reduced in such a way tree is still complete;*

here $\psi^1, \psi^2, \ldots, \psi^j$ *are the formulae assigned to all the child nodes of n.*

The proof is by induction with respect to the tree size. The second condition is necessary only if some of the variables occurring in ψ occurs also in the formulae $\psi^1, \psi^2, \ldots, \psi^j$ (resolving the formulae may require substituting for a variable and thus, by influencing the path above, it may violate the completeness of the tree resulting from bd-resolution application).

For intuition, the construction of a complete Ψ-tree provides a strategy for a proof (derivation) of the root context formula Ψ from the set of all formulae determined by the paths from root to leaf nodes.

Further, let us take a closer look at the step consisting in developing some tree t' from a tree t by extending one of the leaf nodes belonging to t with a set of its successors; in fact this is a basic step in the design process. The problem is that not necessarily all the formulae ψ^i determined by some paths in the extended tree must be satisfiable under the assumed set of interpretations **I**. Let $\Psi \wedge \psi^1, \ldots, \Psi \wedge \psi^i$ denote the satisfiable formulae, and let

$$\Psi \wedge \psi^{i+1}, \ldots, \Psi \wedge \psi^j$$

denote the unsatisfiable ones (here the context defined by Ψ is taken into account). Of course, there is no need to develop the initial tree t with respect to nodes $n^{i+1}, \ldots, n^j$ being the final nodes described by the unsatisfiable formulae $\Psi \wedge \psi^{i+1}, \ldots, \Psi \wedge \psi^j$; roughly speaking, pruning the unsatisfiable paths corresponds to deleting unsatisfiable formulae in a disjunction (the one of Def. 3). The "partially" developed tree will be a complete tree as well; this is a weaker version of Proposition 6.

Proposition 7

A Ψ-tree is complete if the following conditions hold:

- *for any non-leaf node n (being an immediate ancestor of some leaf nodes) and a conjunctive formula ψ determined by the path from root to this node there exist formulae $\psi^1, \psi^2, \ldots, \psi^j$ such that*

$$\psi \models_{\mathbf{I}} \psi^1 \vee \psi^2 \vee \ldots \vee \psi^j \tag{13}$$

  where $\psi^1, \ldots, \psi^i$ are the formulae assigned to all the child nodes of n and such that $\Psi \wedge \psi^l$ is satisfiable for $l = 1, 2, \ldots, i$, and where $\Psi \wedge \psi^l$ are unsatifiable formulae for any $l = i+1, \ldots, j$;
- *resolving (hypothetical) of the formulae assigned to leaf nodes given by (13) leads to reduced but still complete Ψ-tree.*

The above proposition may contribute to significant reduction of both the length of precondition formulae and, with respect to size of the tree, the number of rules as well.

Further, the question of determinism should be raised. As follows from the presented analysis, a sufficient condition for determinism is constituted by unsatisfiability of the conjunction of any two precondition formulae. Note that if any branching in the tree incorporates not only complete but exclusive set of conditions, e.g. as in the introductory example (under the assumed interpretation), then no two precondition formulae can be satisfied at the same time time. This can be recapitulated in the following proposition.

Proposition 8

A Ψ-tree t defines a deterministic set of rules if one of the following conditions hold:

- *for any non-leaf node n*

$$\psi^i \wedge \psi^j \tag{14}$$

  is an unsatisfiable formula, i.e. no two formulae describing diffrent paths can be satisfied at the same time (i.e. exclusive branching conditions are always specified; ψ^i and ψ^j are the formulae assigned to the child nodes of n), or
- *for any two paths going through a node with non-exclusive conditions immediately below, exclusive conditions (for the paths) are added at some node(s) below.*

Thus, the Ψ-tree can be used for simultaneous generation of not only complete set of rules, but by considering exclusive conditions at any branching node deterministic set of rules is obtained at the same time. In case overlapping conditions have to be used at some branching (e.g. for conditions simplification), the specific paths may be marked during design as ones potentially referring to nondeterministic rules. A further check can be done after the design.

In construction of deterministic and complete systems, an inevitable occurring problem is the one of combinatorial explosion; obviously, a complete and deterministic system with state rather than situations used as preconditions, and based on n logical conditions (e.g atomic formulae) should have 2^n rules. Note however, that in our case this problem is significantly reduced, because:

1) the representation language allows for representation of situations rather than states, so there is one rule for a subset of the state space, possible that one containing many states;
2) some of the potential precondition formulae describe physically infeasible states, and thus they can be omitted; the completeness is still assured by Proposition 2;
3) we discuss theoretical approach and ideal case; in practical design one can agree to partial completeness and incomplete determinism.

Further, the complexity problems can be significantly reduced by hierarchal design.

The problem of finding minimal representation (reduction) cannot be solved directly during generation of the tree in a general way. This is because only rules with the same conclusions are likely to be reduced; the conclusions, however, are assigned to the precondition formulae after the tree is generated. However, the use of Ψ-trees for design allows for immediate *local minimization* of the number of generated rules in a straightforward way.

The main idea of minimization consists in joining two or more rules with "slightly different" preconditions. The idea of *local minimization* consist in performing this operation at any branching node, directly after developing it – the current leaf nodes leading to the same rule action can be joined at once. Or, even better, we simply do not perform "too detailed" branching, that's all! This seems to be reasonable and efficient enough in most cases, but in a general case a post-design minimization may be necessary.

Note that if any path in the tree is assigned different conclusion or action, no reduction is possible. If two or more paths have the same conclusion, then they should be analyzed for possible reduction. Both reduction and partial reduction can be applied. Note that due to the completeness condition and validity of bd-resolution, reduction of rules does not violate completeness – the reduced set of rules must be complete as well. Further, if the set of rules was deterministic, the reduced set of rules will be deterministic as well.

10.3. OSIRIS – a design tool

Based on the ideas presented above, an experimental tool for development of rule-based systems named OSIRIS was designed and implemented [42]. This tool was also described in [23] and [24]. In [42] a new idea for graphical representation of combined tabular systems (and their parts) together with decision trees was developed. These are the so-called *tab-trees* or *tree-table* representation.

The main idea of the tool consists in building a hierarchy of tabular systems [25]. This hierarchy is based on the Ψ-tree structure. Each row of a OAV table is right connected to the other OAV table. Such a connection implies logical AND relation in between.

The component tabular systems used in tree-table representation are divided into two kinds:

1) attribute tables,
2) action tables.

Attribute tables are the attribute part of a classical OAT, Action tables are the action part. There is one logical limitation. While attribute tables may have as many rows as needed (a number of columns depends on the number of attributes), action tables may have only one row, it means that the specified action, or set of actions if there is more then one column, may have only one value set, which preserves consistency.

An example of a tree-table representation is given in Figure 10. Please note, that a tree--table representation is similar to Relational-Data-Base (RDB) data representation scheme.

The main features of the tree-table representation are very simple, readable and engineers acceptable knowledge representation, remaining in part the RDB tables, composed with:

- hierarchical, tree-like representation,

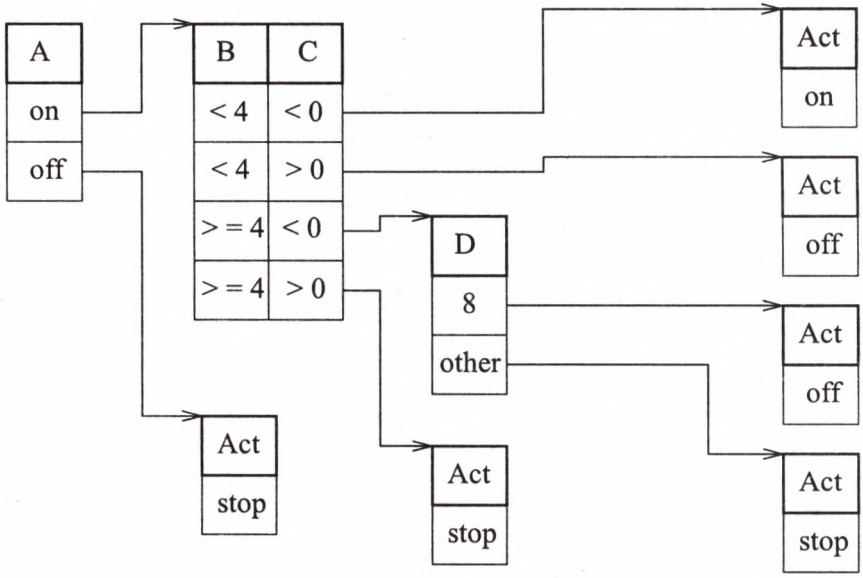

Fig. 10. An example of a tree-table representation

- highly efficient way of visualisation with high data density,
- power of the decision table representation,
- analogies to the RDB data representation scheme.

The OSIRIS tool is a multi-module system designed for UNIX environments (tested under Debian GNU/Linux, Sun Solaris 2.5). It consists of: a graphical environment for computer aided development of rules, a code generator for Kheops system, a validator, which provides completeness checking and a run-time environment for created rules. The architecture of the OSIRIS and further details are described in [42] and also in [23] and [24].

11. An example

After [30] let us consider the following system:

- A_1:=age; $D_1 = \{y, p, q\}$, where:

  y – young,
  p – pre-presbyotic,
  q – presbyotic;

- A_2:=spectacle; $D_2 = \{m, h\}$, where:

  m – myope,
  h – hypermyope;

- A_3:= astigmatic; $D_3 = \{n, y\}$, where:

  y – yes,
  n – no;

- A_4:=tear production rate; $D_4 = \{r, n\}$, where:

  r – reduced,
  n – normal;

- D:=type of contact lenses (decision attribute); $D_D = \{H, S, N\}$, where:

  H – hard contact lenses,
  S – soft contact lenses,
  N – no contact lenses.

Consider the following Optician's Decision Table (Tab. 1), being a perfect example of a tabular system.

Table 1 can be regarded as a complete decision table (ready for use) or as a specification of some tabular rule based system. A brief analysis assures us that there are no redundant or subsumed rules. The system is deterministic and complete. The following reduction is possible.

Table 1

Number	Age	Spectacle	Astigmatic	Tear p.r.	Decision
1	y	m	y	n	H
2	y	n	y	n	H
3	p	m	y	n	H
4	q	m	y	n	H
5	y	m	n	n	S
6	y	n	n	n	S
7	p	m	n	n	S
8	p	n	n	n	S

Table 1 cont.

Number	Age	Spectacle	Astigmatic	Tear p.r.	Decision
9	q	n	n	n	S
10	y	m	n	r	N
11	y	m	y	r	N
12	y	n	n	r	N
13	y	n	y	r	N
14	p	m	n	r	N
15	p	m	y	r	N
16	p	n	n	r	N
17	p	n	y	r	N
18	p	n	y	n	N
19	q	m	n	r	N
20	q	m	n	n	N
21	q	m	y	r	N
22	q	n	n	r	N
23	q	n	y	r	N
24	q	n	y	n	N

Reduction to complete domains (total reduction in which no subsets of the domains are admissible) leads to Table 2.

Table 2

Number	Age	Spectacle	Astigmatic	Tear p.r.	Decision
1	y	–	y	n	H
2	–	m	y	n	H
3	y	–	n	n	S
4	p	–	n	n	S
5	–	n	n	n	S
6	–	–	–	r	N
7	p	n	y	–	N
8	q	m	n	–	N
9	q	n	y	–	N

A further total reduction is possible; as the result the Table 3 is obtained.

Table 3

Number	Age	Spectacle	Astigmatic	Tear p.r.	Decision
1	y	–	y	n	H
2	–	m	y	n	H
3–4	{y, p}	–	n	n	S
5	–	n	n	n	S
6	–	–	–	r	N
7–9	{p, q}	n	y	–	N
8	q	m	n	–	N

The output table is in its minimal form (maximally reduced one). It is still complete (logically equivalent to its input form) and deterministic.

12. Concluding remarks

The paper presents an extended outline of knowledge representation and inference for analysis, verification and design of tabular systems. Tabular sytems, follow the pattern of RDBs and attributive decision tables, and provide a powerful paradigm for data and knowledge representation. Logical characteristics of such systems are defined and some elements for algebraization of analysis and verification of theoretical properties of tabular rule-based systems are presented. A taxonomy of issues concerning formal verification is put forward. It is shown that several theoretical properties can be analysed through simple algebraic operations instead of logical inference. Algebraic notation, close to RDBS model, seems to be simpler and more intuitive than pure logic, so it might be more easily accepted by practitioners.

The main ideas and extensions proposed in this paper include:

- extended, common model for data and knowledge (apart from atomic values, also sets and intervals can be used for values encoding);
- hierarchical approach and limitation of the verification to local context; analysis of global contexts can also be performed at the higher level;
- algebraic approach to analysis and verification of theoretical properties;
- a new approach to design of rule-based systems promoting on-line verification during the design stage (a guided design);
- a new taxonomy of anomalies in rule-based systems.

The size of the tabular rule-based systems is assumed to be limited, mainly thanks to the multi-level structure of the system. Due to limited size the local verification can be quite efficient. A similar procedure can be applied at the higher level for analysis of theoretical properties.

References

[1] Andert E.P.: *Integrated knowledge-based system design and validation for solving problems in uncertain environments*. Int. J. of Man-Machine Studies, 36, 1992, 357–373

[2] Bendou A., Ayel M.: *Validation of rule bases containig constrains*. ECAI '96. Workshop on Validation, Verification and Refinement of Knowledge-Based Systems, 1996, 120–125

[3] Cragun B.J., Steudel H.J.: *A decision-table-based porcessor for checking completness and consistency in rule-based expert systems*. Int. J. of Man-Machine Studies, 26, 1987, 633–648

[4] Ligęza A. *et al.*: *Supervision systems*. http://eia.udg.es/iitap/monografia/index-eng.html

[5] Geneserth M.R., Nilsson N.J.: *Logical Foundations of Artifical Intelligence*. Los Altos, California, M. Kaufmann Publ. Inc. 1987

[6] Gottlob G.: *Subsumption and implication*. Information Processing Letters, 24, 1987, 109–111

[7] Gouyon J.-P.: *Kheops users'guide*. Report of Laboratoire d'Automatique et d'Analyse des Systemes, 92503, 1994

[8] AITECH Katowice: *Sphinx 2.3*. http://www.aitech.gliwice.pl

[9] Laffey T. *et al.*: *Real-time knowledge-based syste*m. AI Magazine, Spring: 1998, 27–45

[10] Lamb N., Preece A.: *Verification of multi-agent knowledge-based system*. ECAI'96 Workshop on Validation, Verification and Refinement of Knowledge-Based Systems, 1996, 114–119

[11] Ligęza A.: *Towards design of complete rule-based control systems*. In: Karba R., Kocijan J., (Eds), IFAC/IMACS International Workshop on Artificial Intelligence in Real-Time Control, Bled, Slovenia, 1995, 189–194

[12] Ligęza A: *Logical suport for design of rule -based systems, realability and quality issues*. In: Rouset M.C., (Ed.), ECAI-96 Workshop on Validation, Verification and Refinement of Knowledge-Based Systems, Budapest, 1996, vol. W2 , 28–34

[13] Ligęza A.: *Logical foundations for knowledge-based control control systems – knowledge representation, reasoning and theoretical properties*. Scientific Bulletins of AGH, Automatics 63(1529), 1993, 144

[14] Ligęza A.: *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, 1, 1993, 132–137

[15] Ligęza A.: *Backward dual resolution, direct proving of generalization.* Information Modeling and Knowledge Bases. V, 1994, Principles and formal techniques, 336–349

[16] Ligęza A.: *Logical foundations for knowledge-based control systems, part. i: Language and reasoning*. Archives of Control Sciences, 3 (XXXIX) (3–4), 1994, 289–315

[17] Ligęza A.: *Logical foundations for knowledge-based control systems, part ii: Representation of states, transformations, and analysis of theoretical properties*. Archives of Control Sciences, 4 (XL) (1–2), 1994, 129–166

[18] Ligęza A.: *Logical analysis of completeness of rule-based systems*. EUROVAV '97. The 4th European Symposium on the Validation and Verification of Knowledge Based Systems, 1997, 19–29

[19] Ligęza A.: *Towards logical analysis of tabular rule-bases systems*. Proceedings of the 9th European Workshop on Database and Expert Systems Aplications, Vienna, Austria, 1998, 30–35

[20] Ligęza A.: *Intelligent data and knowledge analysis and verification: towards a taxonomy of specyfic problems*. Proceedings of the 5th European Symposium on Verification and Validation and of Knowledge Based Systems and Components: Validation and Verification of Knowledge Based Systems: Theory, Tools and Practice, EUROVAV '99, 1999

[21] Ligęza A.: *Elements of algebraic data analysis for verification of qualitative properties*. Proceedings of the Conference Knowledge Aquisition from Databases, 2000, 18–28

[22] Ligęza A., Parra P.F.: *Towards logical analysis of rule-based systems*. Proceedings of the 13th European Meeting on Cybernetics and System Research, 1996, 2, 1211–1216

[23] Ligęza A. Nalepa G.J., Wojnicki I.: *Analysis of the selected problems of design and implementation suport of rule-based systems on the base of kheops system* (in polish). Real Time Systems 2000, Cracow, Institut of Automatic AGH, Szmuc T. and Klimek R., (Eds), 2000, 53–64

[24] Ligęza A., Wojnicki I., Nalepa G.J.: *Design and implementation support of rule based systems* (in polish). II Polish National Conference on Software Engineering, Zakopane – Cracow, 2000, Zieliński K. (Ed.) Wydawnictwa AGH, 229–236

[25] Ligęza A.: *An introduction to knowledge-based process monitoring, supervision and diagnosis. Basic ideas, problems and theoretical foundations*. 1997, Working notes

[26] Lunhardi A.D., Passino K.M.: *Verification of qualitative properties of rule-based expert systems*. Applied Artificial Intelligence, 1995, 9, 587–621

[27] Marek W.: *Basic properties of knowledge base systems*. The Knowledge Frontier, 1987, 137–160

[28] Nazareth D.L.: *Issues in the verification of knowledge in rule-based system*. Int. J. Man-Machine Studies, 1989, 30, 255–271

[29] Nguyen T.A. *et al.*: *Checking an expert systems knowledge base for consistency and completeness*. Proceedings of the 9th IJCAI, 1985, 375–378

[30] Pawlak Z.: *Rough Sets. Theoretical Aspects of Reasoning about Data*. Dordrecht/Boston/London, Kluver Academic Publishers 1991

[31] Perkins W.A. *et al.*: *Knowledge base verification*. Topics in Expert System Design, 1989, 353–376

[32] Preece A.D.: *Methods for veryfying expert system knowledge bases*. Technical Report, 1991, 37

[33] Preece A.D.: *A new approach to detecting missing knowledge in expert system rule bases*. Int. J. of Man-Machine Studies, 38, 1993, 161–181

[34] Preece A.D. *et al.*: *Verifying rule-based systems*. Technical Report, 1991, 25

[35] Preece A.D., Shinghal R.: *Foundation and application of knowledge base verification*. Technical Report, 1991, 26

[36] Travé-Massuyéz L., Milne R., Nicol C., Quevedo T.J.: *Knowledge based gas turbine condition monitoring*. In: Cooper C., Macintosh A., (Ed.), Applications and Innovations in Expert Systems III, volume III, Oxford, SGES Publications, Cambridge 1995, 23–43

[37] Sinton A.J.: *A safety analysis of the airbus A320 braking system design*. Master's thesis, A M.Sc. Thesis, University of Stirling, Department of Computing and Mathematics, 1994

[38] Attar Software: *Xpertrule 3.0*. http//www.attar.com/pages/info_xr.htm

[39] Suwa M. Scott C.A., Shortlife E.H.: *Completenses and consistency in rule-based expert system*. Rule-Based Expert Systems, 1985, 159–170

[40] Tepandi J.: *Verification, testing, and validation of rule-based expert systems*. Proceedings of the 11th IFAC World Congress, 1990, 162–167

[41] Harmelen F. *Applying rule-based anomalies to kads inference structures*. ECAI '96 Workshop on Validation, Verification and Refinement of Knowledge-Based Systems, 1996, 41–46

[42] Wojnicki I.: *Design and implementation of graphical user interface for computer aided local design of kheops knowledge based system*. Master's thesis, Academy of Mining and Metalurgy, 2000

[43] Zbroja S., Ligęza A.: *Case-based reasoning within tabular systems. Extended structural data representation and partial matching*. Proceedings of the Conference Flexible Query Answering Systems, 2000, 199–201