Information & Management 24 (1993) 83–92
North-Holland

Research

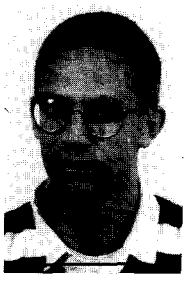
Application areas and added value of knowledge base systems

R.V. Schuwer and R.J. Kusters

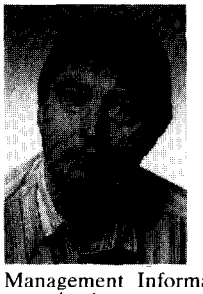
Eindhoven University of Technology, Eindhoven, Netherlands

A knowledge base system is characterized by a separation between application-dependent knowledge and application-independent deduction rules. When used in a business environment, it is not clear what added value this separation has, over conventional systems. It also is not clear what characteristics make a problem tractable for a solution using a knowledge base system. This paper tries to formulate answers to these questions. In order to obtain a sound basis for discussion, a formal model of a knowledge base system is presented.

Keywords: Knowledge base system; Expert system; Application areas; Formal definition



R.V. Schuwer received his MA in Mathematics in 1980 from the Catholic University in Nijmegen (The Netherlands) and his M.Sc. in Informatics in 1989 from the University of Technology in Eindhoven. He worked for four years as a high school teacher and for three years as consultant for the support of end user computing in industry. Since 1987 he is Assistant Professor at the Eindhoven University of Technology. His research is focused on knowledge base systems.



R.J. Kusters received his MA in Econometrics in 1982 from the Catholic University Brabant in Tilburg (The Netherlands). After his graduation he worked for five years as a research assistant at the Eindhoven University of Technology. In 1988 he received his Ph.D. on the basis of a thesis "admission planning in general hospitals" in which production control principles were applied to a non-profit environment. Since 1987 he works as an Assistant Professor of Management Information Systems and Automation at the same institute.

Correspondence to: R.V. Schuwer, Eindhoven University of Technology, Faculty of Industrial Engineering and Management Science, Dept. of Information & Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

Introduction

In recent years many organizations have invested in research aimed at the applicability of knowledge base systems as a solution to business problems. Often prototypes were developed. Criteria for choosing the right problem were generally based on rules of thumb, such as those in Waterman [1986]: the problem must be neither too complex nor too easy; there must be an expert available who is able to formulate his or her way of working etc. These criteria do not say anything about the characteristics of a knowledge base system (namely the separation between knowledge and inference), nor do they give an indication when it is best to tackle a problem with a knowledge base system. In this paper we suggest some answers to this last question. For this it is necessary to have a reasonable idea of the important features of a knowledge base system.

1. Knowledge base systems and backgrounds

In the literature, many different definitions of a knowledge base system can be found. Here we use the following architectural definition [Mars, 1988]: *"A knowledge base system is a computer program, in which as good as possible a separation has been made between application-independent inference rules and application-dependent knowledge."* The difference between a knowledge base system and an expert system is vague. An expert system can be considered a knowledge base system with almost the performance of a human expert. We will use the term "knowledge base system" here.

The evolution of knowledge base systems can be explained in different ways. One starts with

Artificial Intelligence (AI) as the basis, but adding on the explanation and emulation of human acting and thinking. With the development of a knowledge base system one tries to understand how humans think and act. This might result in a computer taking over certain tasks of a human expert. Examples of AI-products are robots, neural networks, computer vision, and natural language processors.

The evolution of knowledge base systems can also be explained as a trend towards a modular construction of software. The goal then is to provide higher quality with better interfaces for maintenance of software and control of the development process. The evolution of knowledge base systems will be discussed from this viewpoint.

The trend towards a more modular construction of software has been an issue since the beginning of automation. The following partition is traditionally found in software systems:

- Operating system and utilities;
- Data;
- Application programs.

Initially these three components existed in a single computer program: each program contained load-, read- and print-functions; e.g. in the Gamma ET-computer of Bull (1957).

The first partitioning of components occurred in a separation of system and application software. Greater efficiency could be reached by generalizing the system tasks. At first this system software contained only simple functions, but the development of multiprogramming led to more complex operating systems with memory management; e.g., that of the IBM 1410 (1961).

The second partitioning was the separation between data and application software; e.g., in COBOL (1961). The simultaneous use of data by several users at one time led to the separation of data management functions from the application software and the development of DBMS; e.g. with DBMS IDS (1965).

The next logical step was the separation of knowledge from the application software. This resulted in knowledge base systems. The assumption was made that an application program contains two types of knowledge: domain dependent knowledge of data and the way in which the data can be manipulated and domain independent deduction (inference) rules. In knowledge base systems, these two types of knowledge are separated from the application program and stored apart in a knowledge component, which consists of:

1. domain dependent knowledge about the data,
2. domain dependent deduction rules,
3. domain independent deduction rules,

The first two form the application dependent knowledge; the last are the application indepen-

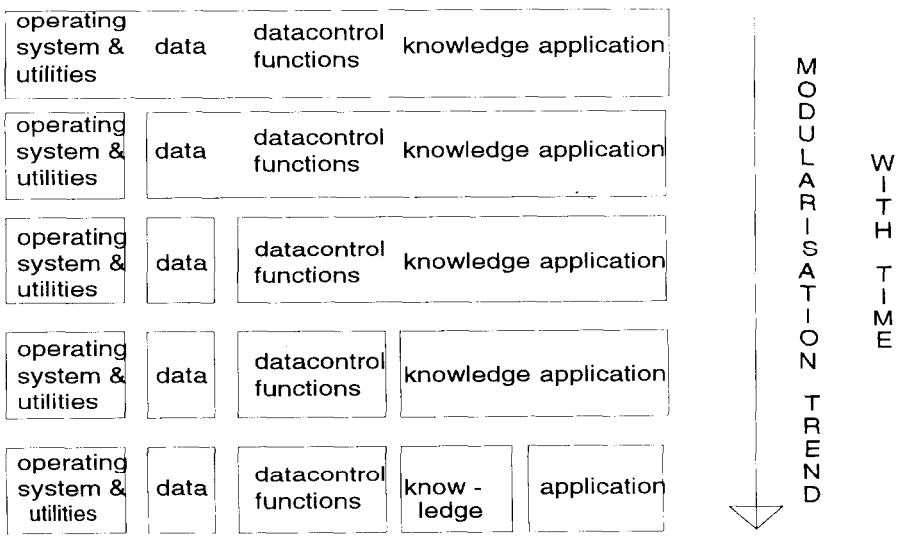


Fig. 1. Modularization trends of software.

dent inference rules. In the application program there is also a user interface and the program operates under program control.

Figure 1 presents this development in a schematic way, where a conventional information system (not included the system software) usually only contain:

- Data;
- Data management component;
- Application program component.

Thus the following differences can be found:

- The knowledge base system has a separate knowledge component. This is part of the application program in conventional information systems.
- In a knowledge base system there is no component for the management of knowledge. Future development may lead to Knowledge Base Management Systems (KBMS).

The knowledge component is the characteristic component of a knowledge base system. A formal description of a knowledge base system is therefore valuable for our discussion.

2. A formal description of a knowledge base system

Several questions are not answered in the informal definition of a knowledge base system:

- What exactly are facts and rules?
- What is an inference mechanism?
- How do facts, rules, and the inference mechanism influence each other?

Also, the definition does not give information about the design of a knowledge component. To get a clearer view of these issues, it is necessary to start with a formal description or model, providing an unambiguous description of the essential characteristics of a knowledge base system. Furthermore the working of and the cooperation between the different components is clearer. For an exact mathematical description of the model, see Schuwer and Eiben [1991].

To illustrate the definition we use the following example:
Buying a kitchen will confront us with a large number of possible configurations which are subject to several constraints. A system which supports the configuration of a kitchen must answer questions as "Is it possible to combine a Philips microwave with a certain type of Bauknecht oven?" or "Which types of dishwashers do fit within a kitchen-cupboard with a width of 45 cm.?" The system must also be able to check if a configuration complies to all constraints.

When building a knowledge base system, in a similar way to a conventional system, the important relations of interest in the domain must be input. In our example, this is a set of relation symbols, combined with variables and the logical NOT-sign (¬), such as:

micro(id#,type,price,colour)
dishwasher(id#,type,price,colour,width,contents)
colour_ok_o_m(id#_oven,id#_micro)
colour_ok_m_d(id#_micro,id#_dishwasher)
colour_ok_o_d(id#_oven,id#_dishwasher)
cupboard(id#,type,width)

Also the "constant" elements must be given. In the example those are the types of the pieces of apparatus (Philips, Bauknecht, Miele,...), the colours (white, grey, black,...), the prices etc. Relation symbols can be combined with constants. A combination of a relation symbol with variables or constants with or without the "¬" will be called a *literal*. When the literal does not contain variables, it is called *ground*. A set of ground literals is called a *database-state* (DB-state). In this paper an element of a DB-state is called a *fact*. A DB-state will therefore often be called a factbase. This part of a knowledge base system is comparable with the datacomponent (database) of a conventional information system. Examples of such DB-states are:

u_1 = { micro(20,philips,500,white), oven(10, philips,4000,white) }
u_2 = { colour_ok_o_m(10,20), colour_ok_m_d(20,30), colour_ok_o_d(10,30) }
u_3 = { colour_ok_o_m(10,20), colour_ok_m_d(20,30) }
u_4 = { micro(20,philips,500,white), micro(20, bauknecht,600,white) }

u_1, u_2 and u_3 are examples of DB-states which represent the "Universe of Discourse" correctly. DB-state u_2 however has some redundancy. When it is known, that colour_ok_o_m(10,20) and colour_ok_m_d(20,30) then one also "knows", that

colour_ok_o_d(10,30) is correct. These dependencies between facts can be given in the form of *rules*. The general form of such a rule is:

IF (a number of facts are known)
THEN (a new fact may be concluded)

In the case of u_2 the rule is as follows:

IF colour_ok_o_m(X,Y) AND
colour_ok_m_d(Y,Z)
THEN colour_ok_o_d(X,Z) (1)

A finite set of rules is called a *rulebase*.

DB-state u_4 shows, that *constraints* must be given to ensure, that a DB-state really gives a correct representation of the Universe of Discourse. In the case of u_4 this constraint will state, that id-numbers must be unique within a DB-state. Such a constraint is like a filter for a DB-state. In the sequel we are only concerned with so called *feasible* DB-states, which fulfil all formulated constraints. The set of all feasible DB-states will be called the *feasible Database-universe* (DB-universe).

Rules from the rulebase can be used on a DB-state to extend it. New facts are added such that a new feasible DB-state will be obtained (so another element of the DB-universe is created). In the example, rule (1) can be used with DB-state u_3 to create a new DB-state (in this case the DB-state u_2). This can be done repeatedly until no new facts can be deduced. In this way a rulebase gives a structure to the feasible DB-state: it can be split up into a set of "basic" facts (always present in the database) and a set of deducible facts. Furthermore there is a set of "forbidden information": due to the demand of consistency of a DB-state, all literals, that are the opposite to those in the feasible DB-state are FALSE. One could call the union of the set of basic facts, the set of deducible facts, and the set of forbidden information the *range of knowledge* of u and R. The remaining set of literals are all those literals, where no assertions can be made (*inaccessible information*). This structure can be described with a function k. For each feasible DB-state u k(u) is the union of u and the set of deducible facts. k(u) itself is also a feasible DB-state. Because knowledge from the rulebase is used we will call the function k a *knowledge function*.

When L denotes the set of all possible literals that can be made with the relation symbols and the constants ($u \in U$, R and k), the different subsets of L are:

u which contains the basic facts.
$k(u) \setminus u$ the deducible facts.
$\{\neg A \mid A \in k(u)\}$
the set of forbidden information
$L - (k(u) \cup \{\neg A \mid A \in k(u)\})$
the set of inaccessible information

This leads to the following definition: Suppose a set of constraints is given. A *knowledge model* is a tuple $\langle U,R,k \rangle$ such that:

- U is a feasible DB-universe,
- R is a set of rules (the rulebase),
- k is the knowledge function, determined by U and R.

An element u of U and the rulebase R together form a *knowledge base*. A knowledge base thus is a set of facts and rules. This agrees with most definitions found in the literature (e.g. [Waterman, 1986]).

A knowledge model is the foundation of a knowledge base system (KBS). We first define a KBS as a program to compute the range of knowledge of a knowledge base. This process starts with a query from the user. A query is a set of literals (not necessarily ground) whose elements will be called *hypotheses*. For each hypothesis the KBS has to select the set which contains it or to find facts whose constants can be substituted into the hypothesis to get the literal. The ultimate goal is to let the KBS do this for the whole query in order to give an answer (to *prove* it). This process can be described step by step. After each step, the state of the KBS can be described by the state of the database (it can have grown by adding proven facts) and the state of the query (it can have grown by adding sub-hypotheses or it can have diminished by deleting hypotheses, that have been proven). For this purpose a KBS consists of an inference procedure able to do this reasoning process. The process can be characterized by two sets of metarules:

- In the query, a hypothesis will be chosen. This is determined by one or more metarules, the *goal-selectionrules*.
- When the hypothesis cannot be answered with the existing DB-state, the system will deduce

facts, until the hypothesis can be answered, or k(u) is reached. For this process of deduction one or more rules are selected and used with the DB-state u. For this the system will use another set of metarules, the *rule-selectionrules*.

In practice the knowledge base and the inference procedure are not enough to answer the query. One often has to go across the border of the range of knowledge of the knowledge base and has to do an assertion about a piece of inaccessible information. For this purpose the KBS uses a set of extended metarules to answer the query. Those will be called *E-rules*. A well-known example is the Closed World Assumption: assume the (ground) hypothesis False when all attempts to proof the hypothesis fail. This rule assumes that all knowledge is available in the system.

Strategies for executing a reasoning process can be described in the way an inference procedure uses goal-selectionrules, rule-selectionrules, and E-rules. When the system uses a "backward chaining" strategy, the rule-selectionrule selects a rule, where the head matches the hypothesis-to-prove. When a "forward chaining" strategy is used, the rule selected must satisfy all the predicates in the body (that is they must be TRUE).

This gives the following definition: A *knowledge base system* is a computer program which consists of:

–U	a feasible DB-universe
–u ∈ U	a feasible DB-state
–R	a rulebase
–G	a set of goal-selectionrules
–S	a set of rule-selectionrules
–E	a set of E-rules
–IP(G,S,E)	an inference procedure

This is illustrated in the appendix. The model can be used to identify knowledge base systems. It can therefore be used to evaluate AI-tools (languages or shells). This gives an indication of problems that can be expected when using a tool in a specific situation.

Although a more or less precise description of a knowledge base system has now been presented, it is not yet obvious what added value is provided by the separation of knowledge and deduction rules.

3. The added value of a knowledge base system

In the terminology of Bemelmans [1987], in evaluating the added value of an information system a distinction is made between:

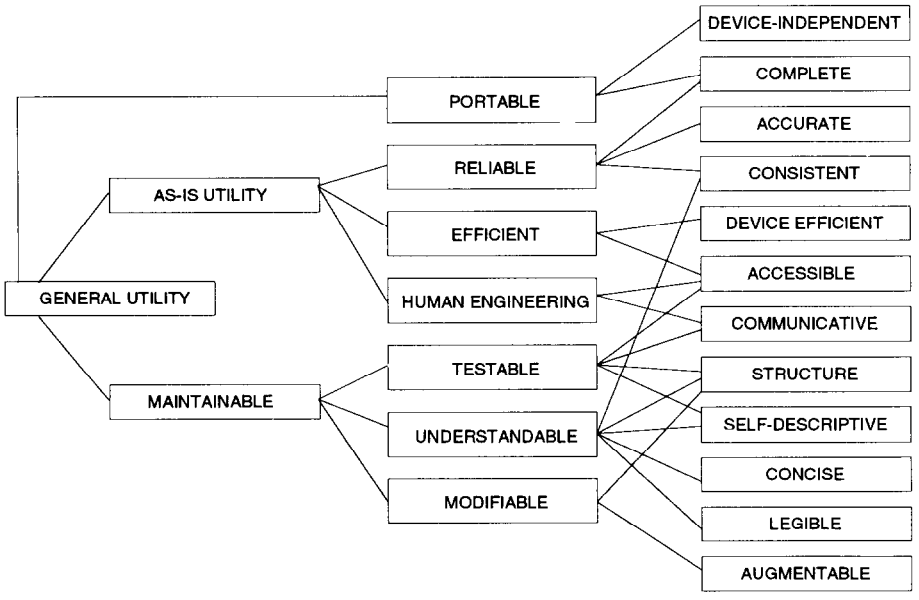


Fig. 2. Software quality characteristics tree.

- functional requirements: those requirements that indicate which data have to be processed and supplied (WHAT the system must do?),
- non-functional requirements (also called performance or quality indicators): the conditions under which data processing and supply must take place (HOW will the system do it?).

The added value of a knowledge base system can also be considered in this way.

The specific architecture of a knowledge base system will not add to its functionality. In principle, all functionality that can be provided by a knowledge base system can also be provided by a traditional information system. From this point of view it comes as no surprise that a system originally designed as a knowledge base system, is often implemented in a traditional way.

Of course in the end all programs can be compiled into machine language. At that point, it makes no difference how the statements were derived. Therefore there is no reason to assume a difference in functionality. Thus the added value of a knowledge base system is found in its ability to fulfil certain non-functional requirements with less effort.

In Boehm et al. [1978] a classification of these non-functional requirements is presented. This so-called 'quality tree' is represented in *Figure 2* hierarchically. Looking at the lowest level, the knowledge base system will have an advantage in the following ways:

Consistent	Explicit definition of the knowledge provides for better checks of the knowledge.
Accessible	The fact that the different components of the system are explicitly defined makes it possible to access them separately.
Structure	From the definition it follows that a knowledge-base system is well structured.
Self-description	Since no information is provided of the knowledge other than the knowledge itself, self-description is assured.
Legible	Separation of knowledge makes it easier to acquire information.
Augmentable	Since the components are defined separately, it is easier to add to these modules.

If we take these characteristics and look at the software quality characteristics tree we see that on higher level the characteristics **testable**, **understandable** and **modifiable** are influenced. Following this through to the next higher level, we see that **maintainable** is the high-level non-functional requirement that is influenced by the decision to design a system as a knowledge-base system. This means that using a knowledge-base system to implement a particular problem has advantages that increase the effectivity and the efficiency of managing the knowledge in the system.

When designing a knowledge base system it is necessary to map the knowledge in terms of the representation technique to be used in the eventual implementation. The development of a knowledge model does not necessarily have to remain restricted to knowledge base systems. When the knowledge used is documented on a conceptual level, insight in this knowledge will increase. This will increase insights into the whole system, providing for greater maintainability.

4. Recognizing problem characteristics

Our analysis is aimed at the components of the knowledge base, namely the sets u, an element of the feasible database universe U, and R, the rulebase. We consider properties of these sets that might indicate the advisability of a knowledge base system solution.

We will first consider the set u, where the facts and relations between them are represented (data and datastructure). If we are talking about a set of data where high demands are required use is made of a Data Base Management System (DBMS), justified by the following properties [Everest, 1986]:

- data are used by multiple users and multiple applications,
- the size of the set of data is large,
- changes in the data occur regularly.

Looking at the rulebase R, we also have to find properties that would make it advisable to be able to manage it. In general, it is desirable to be

able to control a situation when it is complex, or changes regularly, or when several parties are involved in it. Compare this with the DB-state. We now translate these general properties into those that have meaning within the setting of a rulebase. This results in the following properties:

- High complexity and/or size of the knowledge. When these increase, there will be a demand for better control of this rulebase, and this will be facilitated within the architecture of a knowledge base system. It is easier to control the knowledge base when it is represented separately then when it is "hidden" within code.
- If changes in the knowledge occur relatively often, the changes are easier to make when the knowledge is stored separately. This way one avoids large parts of the application having to be rewritten each time a change occurs. A specific case occurs when the knowledge base is being developed: it can then be considered incomplete.

The way the set R is handled is also of importance:

- The knowledge is shared by several users or applications. This occurs relatively seldom but security problems will result; e.g. classified facts may be inferred from others. In such a case admittance control is required.
- The order in which rules can be used is dependent on the specific state of u. It is possible that, in a certain DB-state, rule A has to be used before rule B, while in another DB-state the reverse may be the case; e.g., when little data are available in DB-state u and R is relatively large, it is advisable to choose a forward chaining strategy. However, when u is large and R is small, a backward chaining strategy would be in order. Thus the choice of the rules to be used and the order in which to use them depends on the contents of u. Flexible use of rules is aided by storing them in a knowledge base.

We have now described properties that argue the advisability of proper management of the sets; the potential for this management is offered by a knowledge base system solution. This leads to the following classification:

Situation 1: Both for R and for u, there is no necessity for extra effort that provides better management. In this situation a knowledge base system is not needed.

Table 1
Overview of possible solutions, given the properties of the problem area.

		u	
		weak	strong
R	weak	no DBMS/KBMS (situation 1)	DBMS (situation 2)
	strong	Knowledge Base System (situation 3)	KBMS (situation 4)

Situation 2: Properties of u indicate that data management is needed, but no such indications exist for R. Then a DBMS is indicated as solution.

Situation 3: Management for R is needed, but not for u. This can be tackled using the present generation of knowledge base systems. These systems provide sufficient support for the management of knowledge but are lacking in the management of data.

Situation 4: If management of both u and R is required, then both the present generation knowledge base system (insufficient capabilities for data management) and the present DBMS (insufficient capabilities for knowledge management) are incapable of fulfilling the demand. In this situation, the need for a Knowledge Base Management System (KBMS) arises.
This is summarized in *Table 1.*

In a KBMS, apart from the database, rulebase, sets of metarules, and the inference procedure, functions have to be available for the maintenance of these components.

This gives the following definition of a KBMS: A KBMS is a group of computer programs in which:

- the components of a knowledge base system can be defined,
- the database can be maintained,
- the rulebase can be maintained,
- the sets G, S and E can be maintained,
- the security of these components can be managed.

Note that the functionality of a KBMS encompasses both the functionality of a DBMS as that

of a "traditional" knowledge base system. As an example of an implementation of a KBMS, see Van Herwijnen et al [1990].

5. Conclusions

The knowledge base systems can be considered a logical step in historic evolution. In order to determine if a problem can be adequately solved using a knowledge base system, the knowledge must be analyzed into a set of data (u) and a set of rules (R). Based on the properties of these sets (size, complexity, completeness, robustness and the order of use of rules), the appropriateness of the implementation mechanism can be determined. The main argument that is used for this choice is the need for management of the sets u and R.

Note:
The authors would like to thank Prof. Dr. T.M.A. Bemelmans for his comments on earlier versions of this paper.

References

Bemelmans, T.M.A., Bestuurlijke informatiesystemen en automatisering (Management information systems and automation), Stenfert Kroese, Leiden, 1987 (in Dutch).

Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., MacLeod, G.J., and Merrit, M.J., Characteristics of software quality, North-Holland Publishing Company, Amsterdam, 1978.

Everest G.C., Database Management, McGraw-Hill Book Company, New York, 1986.

Herwijnen, J. van, Houten, E.G. van, Houtsma, M.A.W. and Romkema, H.M., Implementatie van een regel-gebaseerd kennissysteem in een relationele database-omgeving (Implementation of a rule-based knowledge base system in a relational database environment), *Informatie*, Vol. 32, nr. 1, pp. 14–21, 1990 (in Dutch).

Mars, N., Onderzoek van niveau: Kennistechnologie in wording (High level research: the growth of knowledge technology), *Informatie*, Vol. 30, nr. 2, pp. 84–90, 1988 (in Dutch).

Schuwer R.V., Eiben A.E., Knowledge Base Systems: A formal model, Eindhoven University of Technology Computing Science Note 91/20, 1991.

Waterman, D.A., A guide to expert systems, Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

Appendix

The domain of the KBS refers to the configuration of a kitchen. For the sake of simplicity, only part of the configuration will be considered. There will be rules on possible combinations of a micro-wave and an oven. Combinations are restricted by the following rules:

- If the oven has a built-in micro-wave, one is not allowed to choose a separate micro-wave as well. Such ovens are offered by Philips and cost over $2000.
- Only certain colour-combinations of an oven and a micro-wave are allowed.

The following literals will be used:

Oven(Id#,Type,Price,Colour).
Micro(Id#,Type,Price,Colour).
Cook_ok(Id#_Oven,Id#_Micro).
Colour_ok(Id#_Oven,Id#_Micro).
Combination_ok(Id#_Oven,Id#_Micro).

The meaning of "Id# = 0" will be "Not chosen". A "-" will denote a "don't care" (it doesn't matter which value the corresponding field will have).

Given the following knowledge base ⟨u,R⟩ and metarules:

u = { oven(10,philips,4000,white),
oven(11,philips,1500,white),
oven(12,bauknecht,1100,white),
micro(20,philips,500,white),
micro(21,philips,600,grey) }

R = { combination_ok(X,Y), colour_ok(X,Y)
→ cook_ok(X,Y), (1)
oven(X,B,P,-), B = philips, P > 2000,
micro(Y,-,-,-), Y = 0
→ combination_ok(X,Y), (2)
oven(X,-,-,-), X ≠ 0, micro(Y,-,-,-), Y ≠ 0
→ combination_ok(X,Y), (3)
oven(X,-,-,white), micro(Y,-,-,white), Y ≠ 0
→ colour_ok(X,Y), (4)
oven(X,-,-,grey), micro(Y,-,-,white), Y ≠ 0
→ colour_ok(X,Y), (5)
oven(X,-,-,white), micro(Y,-,-,black), Y ≠ 0
→ colour_ok(X,Y), (6)
oven(X,-,-,-), micro(Y,-,-,-), Y = 0
→ colour_ok(X,Y) } (7)

G = { "Select the hypothesis according to the textual order" }

S = { "Select the rules according to the textual order" }

E = { "IF the hypothesis_to_solve is a ground hypothesis
THEN answer is "FALSE"
ELSE answer is "No solution"
ENDIF" }

The inference procedure
IP(G,S,E) =
{ "Select rules from R according to the backward chaining strategy" (1)
"IF hypothesis_to_solve is selected
THEN Look into database for unification (2a)
IF no success
THEN Select rule_to_use (2b)
ENDIF
ENDIF"
"IF no rule_to_use can be found
THEN Backtrack
ENDIF" } (3)

Note that the E-rule in this situation follows the Closed World Assumption.
The set of hypotheses to be answered is:
H = { cook_ok(11,20), (1)
colour_ok(15,21) } (2)
The questions (the hypotheses formulated in H) are answered in the following way:

1. Goalselection
   Result (according to goal-selectionfunction): cook_ok(11,20).
2. Metarule (2a) from IP(G,S,E).
   Result: no success.
3. Metarule (2b) from IP(G,S,E)
   (according to metarule 1 from IP(G,S,E), the KBS will look for a rule, where the head matches the hypothesis_to_solve).
   Result (according to S): rule (1).
4. (Again, according to metarule 1 from IP(G,S,E))
   Add subgoals to H.
   Result (taken into account the goal-selection-function):
   H = { combination_ok(11,20),
   colour_ok(11,20), cook_ok(11,20), colour_ok(15,21)}
5. Goalselection
   Result: combination_ok(11,20).
6. Metarule (2a) from IP(G,S,E).
   Result: no success.
7. Metarule (2b) from IP(G,S,E).
   Result (analogous to steps 3 and 4, rule (2) is chosen from R):
   H = { oven(11,B,P,-), B = philips, P > 2000,
   micro(20,-,-,-), 20 = 0,
   combination_ok(11,20),...}
8. Goalselection
   Result: oven(11,B,P,-).
9. Metarule (2a) from IP(G,S,E).
   Result:
   oven(11,philips,1500,grey).
   (The subgoal is removed from H and all occurrences of B and P in H are substituted)
   H = { philips = philips, 1500 > 2000, ...}
10. Goalselection
    Result: philips = philips (TRUE)
    (The subgoal is removed from H)
    H = { 1500 > 2000, micro(20,-,-,-), combination_ok(11,20),...}
11. Goalselection
    Result: 1500 > 2000 (FALSE)
    (This situation is analogous to the situation, that no rule_to_use can be found to unify this hypothesis. Therefore, according to metarule (3) from IP(G,S,E), backtracking takes place. All subgoals, which were added at step 7, will be removed)
    H = { combination_ok(11,20),
    colour_ok(11,20), cook_ok(11,20),
    colour_ok(15,21)}
12. Metarule (2b) from IP(G,S,E)
    Result (according to S): rule (3).
13. (analogous to step 4, subgoals are added to H).
    Result:
    H = { oven(11,-,-,-), 11 ≠ 0, micro(20,-,-,-),
    20 ≠ 0, combination_ok(11,20),...}
14. (Analogous to step 9 the subgoals, which were added in the last step all can be removed from H, because they can be unified. The subgoal combination_ok(11,20) has now be proven and can be removed from H and added to u.)
    Result:
    H = { colour_ok(11,20), cook_ok(11,20),
    colour_ok(15,21)}
    u = { combination_ok(11,20),
    oven(10,philips,4000,white),...}
15. Goalselection
    Result: colour_ok(11,20).
16. Metarule (2a) from IP(G,S,E)
    Result: no success.
17. Metarule (2b) from IP(G,S,E)
    Result (according to S): rule (4).
18. (Analogous to step 4, the subgoals will be added to H)

Result:
H = { oven(11,-,-,white), micro(20,-,-,white), 20 ≠ 0, colour_ok(11,20,...}

19. Goalselection
Result: oven(11,-,-,white).
20. Metarule (2a) from IP(G,S,E)
Result: success
(The subgoal will be removed from H)
H = { micro(20,-,-,white), 20 ≠ 0, colour_ok(11,20),...}
21. (Analogous to steps 19 and 20 the two remaining subgoals which were added in step 18 can be proven. As a result the subgoal colour_ok(11,20) has also been proven and will be removed from H and added to u.)
Result:
H = { cook_ok(11,20), colour_ok(15,21)}
u = { colour_ok(11,20), combination_ok (11,20), oven(10,philips,4000, white),...}
22. (The hypothesis cook_ok(11,20) has been proven and will be removed from H and added to u)
Result:
H = { colour_ok(15,21)}
u = { cook_ok(11,20), colour_ok(11,20), combination_ok(11,20), oven(10,philips,4000,white),...}
23. Goalselection
Result: colour_ok(15,21)
24. Metarule (2a) from IP(G,S,E)
Result: no success.
25. Metarule (2b) from IP(G,S,E)
Result (rule (4) has been selected from R)
H = { oven(15,-,-,white), micro(21,-,-,white), 21 ≠ 0, colour_ok(15,21)}
26. Goalselection
Result: oven(15,-,-,white)
27. Metarule (2a) from IP(G,S,E)
Result: no success.
28. Metarule (2b) from IP(G,S,E)
Result: no success
29. Metarule (3) from IP(G,S,E)
Result:
H = { colour_ok(15,21) }
30. (Analogous to steps 25 to 29 also rules 5, 6 and 7 will be selected from R. None of the rules leads to a proof for the hypothesis. The hypothesis must therefore be solved with the E-rule.)
Result (answer is FALSE. The hypothesis can be removed from H):
H = { }

Because H is the empty set now, the query is solved. The answer, which will be given by the KBS is "FALSE".