A Logic Model for Rule-Based Expert Systems

József Sziray

Department of Informatics, Széchenyi University
Egyetem tér 1. 9026 Győr, Hungary
E-mail: sziray@sze.hu

Abstract: This paper deals with the calculations in the reasoning process of rule-based expert systems, where inference chains are applied. It presents a logic model for representing the rules and the rule base of a given system. Also, the fact base of the same expert system is involved in the logic model. The proposed equivalent representation manifests itself in a logic network. After that, a four-valued logic algebra is introduced. This algebra is used for the calculations where forward chaining is carried out. Next, the notion of line-value justification is described. This operation is applied in the backward chaining process, also on the base of the previously introduced four-valued logic. The paper describes two exact algorithms which serve for the forward and backward chaining processes. These algorithms can be implemented by a computer program, resulting in an efficient inference engine of an expert system. The achieved result enhances the reliability and usability of the intelligent software systems which is extremely important in embedded environments.

Keywords: Expert system, rule base, inference chains, computational complexity, multi-valued logic.

1. Introduction

The application area of embedded systems and the related economical and reliability requirements imply a specific hardware-software structure that is significantly different from the resources available in modern high-end systems. The relatively low processing performance, small memory space and the safety prescriptions have resulted in various architectural properties and programming solutions [1]. In case of real-time safety-critical systems the reaction time for the external events is a key issue [2]. It means that the speed of the calculations is a critical factor. On the other hand, the same applies to the memory consumption.
In many cases, artificial intelligence is realized within the frames of expert systems. This approach has gained a wide-spread use in controlling railway stations, dangerous chemical processes, power stations, airplane flights, medical systems, etc. These applications are equally related to safety-oriented systems.

As known, the most common form of storing knowledge in expert systems is the use of rules. It means that the knowledge base (long-term memory) consists of rules and facts. The other component of such an expert system is the inference engine which is the most important factor for a successful operation. An inference engine usually works in a fixed manner, for example, it could be designed as either data driven (i.e., forward reasoning or forward chaining) or goal driven (i.e., backward reasoning or backward chaining), however, most of the modern systems may well use both ways of reasoning [3]-[6].

The major concern related to the inference processes is their excessive computational amount. The algorithmic complexity derives from the fact that the task to be solved belongs to the so-called NP-complete problems. As known, NP-complete problems have a computational complexity for which there exists no upper bound by a finite-degree polynomial of the problem size. It means actually that the number of the computational steps is finite, but unpredictable [6]-[9]. Here the problem size can be expressed by the number of rules in the knowledge base. Due to the described features of the computations, the execution speed of the software is a crucial factor. This feature concerns especially the embedded real-time systems, where the response time must always be kept within a previously specified limit.

The paper deals with the calculations in the reasoning process of rule-based expert systems, where inference chains are applied. It presents a logic model for representing the rules and the rule base of a given system. Also, the fact base of the same expert system is involved in the logic model. The proposed equivalent representation manifests itself in a logic network. After that, a four-valued logic algebra is introduced. This algebra is used for the calculations where forward chaining is carried out. Next, the notion of line-value justification is described. This operation is used in the backward chaining process, also on the base of the previously introduced four-valued logic. The paper describes two exact algorithms which serve for the forward and backward chaining processes. These algorithms can be implemented by a computer program, resulting in an efficient inference engine of an expert system.

2. Fundamental Concepts

In a rule-based system, any rule consists of two parts: the IF part, called the antecedent (premise or condition) and the THEN part, called the consequent (conclusion or action). The basic syntax of a rule is:

\[
\text{IF } \text{<antecedent>} \text{ THEN } \text{<consequent>}.\]
In general, a rule can have multiple antecedents joined by the keywords AND (conjunction), OR (disjunction), or a combination of both. Negation of an antecedent is also allowed. In this case the NOT operator is used. For example,

IF the spill is liquid
AND the spill pH < 6
AND the spill smell is vinegar
THEN the spill material is acetic acid.

Forward chaining is an inference method where rules are matched against facts to establish new facts, finally reaching a conclusion. In case of backward chaining the system starts with what it wants to prove, and tries to establish the facts it needs to prove the initial fact. The components of the reasoning process that are applied, constitute the so-called inference chain.

The knowledge base consists of the set of rules (rule base), and the set of facts (fact base), where the rule base is permanent, while the fact base contains an initial set of facts depending on the actual task to be solved, and it changes in accordance with the concrete reasoning process.

The existent expert systems build up the knowledge base in a usual data-base structure, and their inference engine applies an exhaustive search through all the rules during each cycle. The aim of the search is to find the appropriate rules for which the antecedents or the consequents satisfy the actual conditions. As a consequence of this process, systems with a large set of rules (over 100 rules) can be slow, and thus they may be unsuitable for real-time applications, especially in the field of embedded systems [4].

In the following a novel knowledge representation based on Boolean algebra and logic networks will be presented. On this base, a four-valued logic system is introduced. This new model results in a significantly more efficient inference processing than the classical one. The computational improvement is estimated to be at least two orders of magnitude, which is due to the small memory usage and fast operations in the logic domain.

3. The Use of Boolean Algebra and Logic Networks

The relations of Boolean algebra can also be used for the rule-based systems. As it is well-known, Boolean logic involves two values: 0 (false) and 1 (true), where the following three basic operations are used: logic AND (denoted by the multiplication point (·)), logic OR (denoted by the addition sign (+), and logic NOT (denoted by an apostrophe succeeding the actual variable). For instance, A' means the negation of A.

It can be easily seen that the logic conditions within the rules can directly be substituted by the corresponding Boolean operations and logic gates [10]. As an example let us consider the following set of rules, where the facts are denoted by capital letters:
IF C AND D THEN L,
IF NOT E THEN K,
IF L OR K THEN P,
IF E AND M THEN Q.

The Boolean description of the above rules is the following:

\[
\begin{align*}
L &= C \cdot D, \\
K &= E', \\
P &= L + K, \\
Q &= E \cdot M.
\end{align*}
\]

These four rules can be represented by four logic gates: two AND gates, one NOT gate, and one OR gate. Now, if we connect the inputs and outputs of these gates in accordance with the identical letters, the logic network of Figure 1 will be obtained.

It should be noted here that in case of a simple direct rule, for example,

IF U THEN V,

its corresponding Boolean form will be

\[
V = U.
\]

This relation is represented by a YES gate which does not modify its input value.

---

![Figure 1. The logic network for the rule base](image-url)
4. The Use of a Four-Valued Logic System

4.1. The truth tables

As known, the original Boolean algebra is based on a two-valued logic, i.e., on 0 and 1. These are called determined values as well. If a fact is true in the inference process, then its logic variable will have the value 1, if it is false then its value is 0. However, as far as the general algebraic treatment of rule bases is concerned, it requires more than these two values. It can be proved that the number of necessary and sufficient values is four [11], [12]. It means that in addition to 0 and 1, two more values are to be involved. These are as follows:

1) The indifferent or don’t care logic value: d. It is interpreted in such a way that the network line which carries this value can take on either 0 or 1 freely, without influencing the computational results.

2) The unknown logic value: u. In this case we have not any knowledge about the concrete logic value (0, 1 or d) of the network line carrying u.

The treatment of the four values can be extended to the basic Boolean operations. This extension is summarized in the truth tables of Table 1, below:

<table>
<thead>
<tr>
<th>AND</th>
<th>0</th>
<th>1</th>
<th>d</th>
<th>u</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>d</td>
<td>u</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>d</td>
<td>d</td>
<td>u</td>
</tr>
<tr>
<td>u</td>
<td>0</td>
<td>u</td>
<td>u</td>
<td>u</td>
</tr>
<tr>
<td>OR</td>
<td>0</td>
<td>1</td>
<td>d</td>
<td>u</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>d</td>
<td>u</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>d</td>
<td>d</td>
<td>1</td>
<td>d</td>
</tr>
<tr>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
<td>u</td>
</tr>
<tr>
<td>NOT</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>u</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 1. Truth tables of the four-valued logic system*

It should be remarked that there are other noteworthy logic systems with four values, e.g., those proposed in [13] and [14]. In these systems (and also others), 0, 1, “divergent” and “meaningless” are used. The basic deviation is that the values in the other systems are interpreted and applied for a differing purpose.

Next we are going to show how the given truth tables are used for forward and backward chaining. To reach this goal, consider the rule base above and the logic network belonging to it (see Figure 1). Let the initial set of facts be as follows:

\[ T_0 = \{A, B, C, D, E, G, H\}. \]
4.2. The forward chaining procedure

In our representation, the forward chaining is performed in the following way:

**Step 1:** \( C = 1 \) and \( D = 1 \), since they both are in the fact base, which results in \( L = 1 \), so \( L \) is placed in the fact base.

**Step 2:** \( E = 1 \), because \( E \) is in the fact base, from which it follows that \( K = 0 \), but \( L = 1 \) alone implies \( P = 1 \), so \( P \) is placed in the fact base.

**Step 3:** The fact base does not contain \( M \). In our logic system it can be interpreted as \( M = u \). Though \( E = 1 \), due to the unknown value of \( M \), this is not sufficient to imply \( Q = 1 \). It means that \( Q = u \), thus \( Q \) cannot be placed in the fact base.

Here the final conclusion of the forward chaining was that fact \( P \) is true alone.

The above computational procedure can also be called forward tracing of the logic values. It means that we calculate the output values of the logic gates with knowledge of the gate-input values. As a matter of fact, this kind of tracing values is nothing else than logic simulation, which is a really fast process on computers.

4.3. The backward chaining procedure

The same way as before, the backward chaining procedure involves backward tracing of the logic values through the network. Now the input values of a gate have to be determined with knowledge of the actual gate-output value. In this case the goal is to justify that a primary output value is 1, i.e., a selected fact is true. It requires a successive decision process which is also called line-value justification [11], [12], [15]. As known, line-value justification is a procedure with the aim of successively assigning input values to the logic elements in such a way that they are consistent with each previously assigned value. (This concept is an auxiliary calculation process for justifying an initial set of logic values in a network, first applied in the so-called D-algorithm, for two-valued logic [16].)

The backward tracing of the logic values can also be performed in accordance with the four-valued truth table. However, this principle differs in some points from the forward tracing. In case of forward tracing, the output value of a gate is to be calculated with knowledge of the input values at the gate. If the inputs are given then they determine the output unambiguously. On the other hand, for a given output value at a gate not only one input combination can be assigned, there may be more than one possible choices. If two-input gates are considered, the possible choices are summarized in Figure 2 and Figure 3. If the number of inputs at a gate were more than two, it would increase the number of choices, but would not cause any difference in principle.
When performing this process the following viewpoints have to be taken into consideration:

- Only the determined logic values, 0 and 1, have to be traced back, i.e., these values are to be justified at the gate inputs. The value of \( d \) needs no justification, so it is unnecessary to trace it back. The output value of \( u \) is justified only by the input values of \( u \).

- Since \( d \) does not require justification, it is worth assigning the minimum number of determined values to the gate inputs, while leaving the others at the value of \( d \).

- In this logic system, the determined values and \( u \) are consistent only with \( d \). This fact is to be taken into consideration when a network line has already a previously assigned value, and another value is required at the same line. Whenever a contradiction, i.e., inconsistency occurs, we have to make a new choice or change the last possible decision.
In our example (Figure 1), the computations proceed as follows:

**Step 1:** The proof of \( P = 1 \): At first let \( K = 1 \) and \( L = d \), which are the minimally necessary assignments. Now from \( K = 1 \) it follows that \( E = 0 \), which is a contradiction, for \( E \) is in the fact base, so \( E = 1 \) holds.

**Step 2:** We have to modify our previous decision: Now let \( L = 1 \) and \( K = d \). In this way \( L = 1 \) is justified by \( C = 1 \) and \( D = 1 \), without any contradiction.

**Step 3:** It is unnecessary to trace back the value \( K = d \), since the indifferent value does not require justification. So the proof of \( P \) has been finished.

**Step 4:** The proof of \( Q = 1 \): This condition requires that both inputs to the AND gate be 1, i.e., \( E = 1 \) and \( M = 1 \). Since \( E \) is a member of the fact base, \( E = 1 \) holds. However, \( M \) is missing from the fact base, which means that \( M = u \). In this case it is impossible to justify (prove) that \( Q = 1 \).

5. Concluding Remarks

This paper has presented a logic model which is directly applicable for inference chains in expert systems. Both forward and backward reasoning can be performed on the base of the model. In comparison with the conventionally organized knowledge bases, the calculations using this four-valued logic can advantageously be organized and carried out in embedded computing systems due to the following reasons:

- The storage requirement of the four logic values at the network lines is negligible: only two bits are necessary and sufficient for coding them.
- Computations among logic values are *ab ovo* fast and efficient. This fact manifests itself especially when bit-level implementation is applied.
- The data-base structure of a logic network is comparatively simple, and requires minimal memory space. Only the gate types and the input-output connections of the gates are to be encoded and stored. The forward and backward tracing are carried out directly on this network structure.

References


