J Syst Eng (1994)1:39–55

Journal of Systems Engineering

A Real-Time Procedure for Knowledge Processing

Ricardo Gudwin, Fernando Gomide, Márcio Andrade Netto and Maurício Magalhães

UNICAMP–FEE–DCA, CP 6101, 13081-970–Campinas–SP, Brasil

A real-time knowledge processing procedure is proposed for rule-based systems in general and control systems applications in particular. The procedure provides mechanisms that address important characteristics for real-time processing, including focus of attention, integration of symbolic/numeric processing, optimum use of environment and response time warranty. It also supports the trade-off needed in most real-time systems, for example between processing power, response time, data space and inattention. An application concerning supervisory group control of elevators is also included to show the usefulness of the proposed procedure.

Keywords: Artificial intelligence; Real-time computer systems; Computer control; Knowledge engineering

1. Introduction

Real-time knowledge-based or knowledge-processing systems are playing an increasingly important role in transportation, manufacturing, control, robotic and aerospace systems. They are no longer limited to low-level control functions. Control, supervision and monitoring of complex hierarchical systems in dynamic and sometimes unpredictable or hazardous environments, are typical tasks of current manmade systems.

Several large real-time applications are required to operate in environments that are not fully structured. The lack of information and uncertainty of the environment requires the use of problem-solving techniques [1]. Elevator group control is one such application [2,3]. There are many possible situations, e.g.: the state of all elevators; existing calls in the building; completion of previously scheduled cars; combining new hall calls with performance criteria. There are several possible corresponding schedules and as new hall calls appear, the scheduled cars must be revised frequently. Since entering all the possible responses (schedules) into the computer is infeasible, automating the response construction process will be required. Factory scheduling is another such application [4].

Current development in real-time artificial intelligence is driven by a need to make knowledge-based systems work in real-time [1] and a need to integrate knowledge-based approaches to handle the complexities of problem-solving behaviour in control systems [5,6].

Looney [7,8] proposed a matrix procedure for real-time knowledge processing considering only production rules with one antecedent. His procedure, however, does not preserve its form in more realistic and usual situations where several antecedents are present in a rule that prescribes a nonlinear relationship. Furthermore, when handling multiple antecedents, the procedure is difficult to analyse and to predict if it could meet deadlines due to the chaining scheme adopted. This is a critical issue in real-time situations.

Alternative schemes based on a network type of representation for production rules are the RETE [9] and EUREKA [10] procedures. However, both procedures lack the predictability property, which is essential in real-time applications.

More recently, Paul et al. [11] developed an approach which integrates problem-solving method-

Received 10 September 1992

Correspondence and offprint requests to: F. Gomide, UNICAMP–FEE–DCA, CP 6101, 13081-970–Campinas–SP, Brasil.

ology and architectural primitives to reduce the variance at methodology level and at problem-solving level. Using this approach they have shown that problem-solving and real-time tasks coexist within a readily analysable framework.

The purpose of this paper is to present a real-time knowledge processing (RTKP) procedure based on conjunctive and disjunctive matrices and operators. The proposed procedure affords the setting up of the focus of attention mechanisms and guarantees its response time. These are important characteristics that real-time knowledge-based systems should have. Furthermore, the procedure structure is such that parallel implementation is natural. Even with conventional computer architecture the procedure can be implemented to take advantage of CPU parallel bit operations, which may be viewed as a pseudo-parallel setting up. Although we focus on the procedure's propositional logic viewpoint, its extension to fuzzy logic and first-order predicate logic is immediate. Certainty factors can also be easily handled.

The proposed procedure, as opposed to [7], preserves the general structure even when processing rules with multiple antecedents. It also allows serial, pseudo or fully parallel implementations and its extended version supports fuzzy and certainty factor reasoning. These features are not observed in [11]. Furthermore, it differs from both approaches because the exact number of data operations in each procedure call can be determined. Therefore, the response time can be obtained for each change observed on the monitored variables. Other distinguishing features are the 'horizontal expansion' and 'rule base compression' mechanisms used to define the trade-off between memory size and response time. Moreover, a method is provided to avoid underutilisation of relevant information and computations. Actually, the procedure can be implemented using a structure that avoids irrelevant information, as far as control applications are concerned.

The paper is organised as follows. The next section presents the preliminaries and the general definitions needed. The procedure and its main features are addressed in Section 3. Methods for knowledge base analysis and manipulation are presented in Section 4. It is also shown that alternative versions of the procedure are easily obtained to explore the characteristics of the method. A tutorial example is provided in Section 5 to illustrate the idea behind the proposed procedure. Section 6 addresses a discussion about real-time knowledge-based systems and the associated requirements and characteristics. It is shown that the procedure developed agrees with most of them. Next, in Section 7, an application concerning supervisory group control of elevators is described to show the usefulness of the proposed approach. This application is a simplified version of a real world application. Finally, Section 8 presents conclusions.

2. Preliminaries and General Definitions

A typical RTKP system acting as a direct digital control system is shown in Fig. 1. The RTKP module is connected to information sources and receivers. Sources may be sensors connected to a process, human users or even computer programs in large integrated systems. Receivers can be either actuators or human users or again computer programs. The main idea behind this scheme is that RTKP takes information from a system, processes this information with the knowledge stored in it, and then outputs new information to the system. The outputs are the control decisions. Figure 2 shows an RTKP system performing supervisory control task. The knowledge processing task is encapsulated within a server to guarantee temporal isolation between it and conventional real time tasks [11–13].

Internally, RTKP is divided into four basic parts. First is a preprocessor module responsible for the transformation of input information into the internal representation model used. This model is also responsible for any mathematical treatment that should be necessary to fit the internal representation (by making transformation of variables, for example). The postprocessor module translates the internal representation model into output information in a format as required by the process. Between these two modules are the inference engine

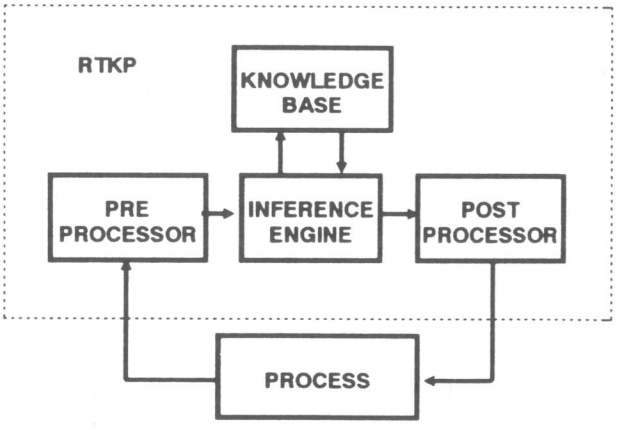

Fig. 1. Typical RTKP structure in direct digital control.

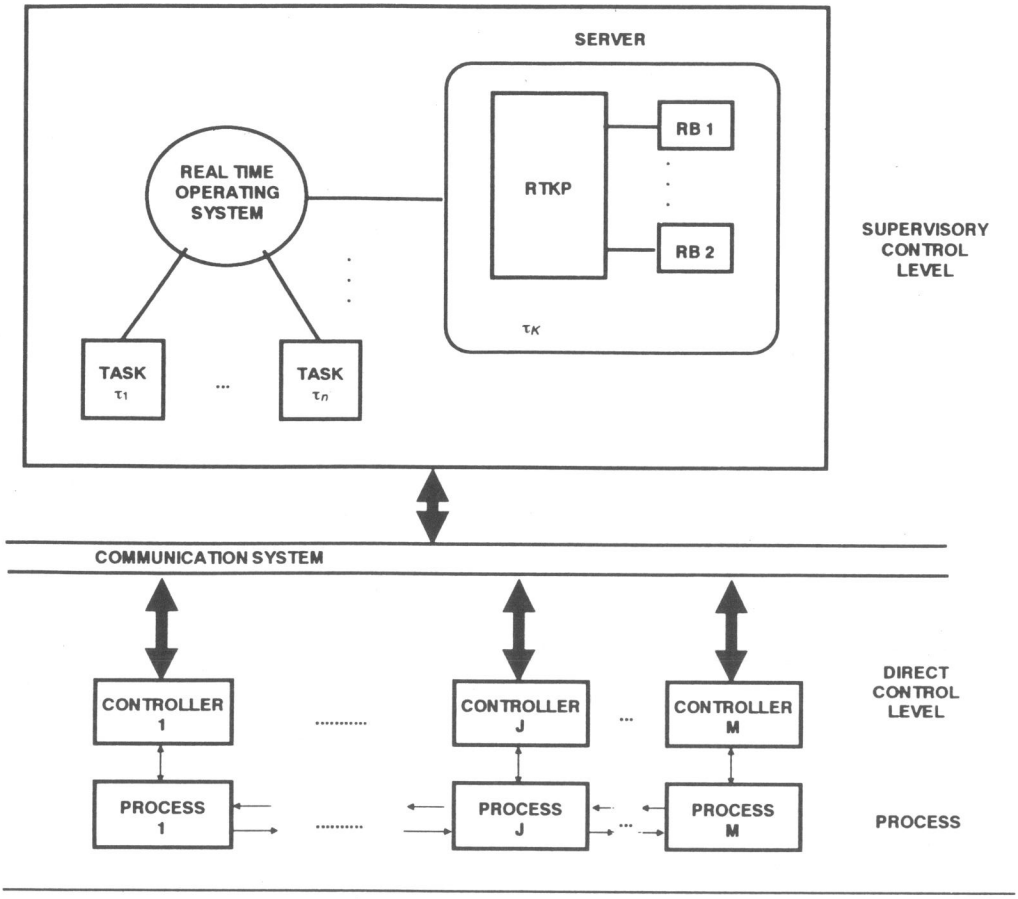

Fig. 2. RTKP in supervisory control systems.

and the knowledge base. The latter is the internal knowledge depository, coded in a usable format. Information provided by the preprocessor module and knowledge base is processed by the inference engine to generate the desired outputs.

The RTKP is defined by:

- Internal representations of information provided by the preprocessor module and information to be converted by the postprocessor module.
- Internal representation of the knowledge base,
- A procedure for the inference engine.

The knowledge base is composed of two parts: rule base and fact base. A fact is the truth value associated to a particular proposition (or a term) used to store knowledge. A fact base is a set of terms, where each term has a meaning related to the process (an associated proposition). Propositions are considered within the propositional logic framework, although they could also represent predicates or fuzzy propositions. In the proposed procedure, the fact base is represented by a fact vector where each component is related to a term and contains its truth value.

The rule base can be viewed in two different representations. The first representation is for purposes of knowledge acquisition and analysis (called the virtual representation). It is presented as a collection of rules like: if ⟨antecedent⟩ the ⟨consequent⟩, where ⟨antecedent⟩ represents a conjunctive association of terms and ⟨consequent⟩ represents a disjunctive association of terms. Each term is the symbolic representation of a proposition, as in the fact base. A typical rule like

if A and B then C

has a meaning 'if the proposition A is true and the proposition B is true, then proposition C should be true'. It is relevant now to review the main requirements that real-time knowledge processing systems should fulfil [1]:

- Integration of numeric/symbolic processing
- Continuous operation
- Focus of attention mechanisms
- Interrupt manager services
- Optimum use of environment
- Response time warranty
- Temporal data processing

- Truth data maintenance
- Dispensing of explanatory modules

3. Procedure Description

3.1. Fact Base Codification

The real-time inference engine considers two fact bases. The first, called input fact base, has the truth values of terms evaluated by the preprocessor module. The second, called output fact base, contains truth values of the terms after the inference procedure.

Each fact base will be represented by a vector where each component is a boolean variable. This vector has as many elements as the number of terms used in the rule base. There is a one-to-one relation between each term and a vector component.

Definition 1: *fact vector*. Let E be the set of terms which represent the propositional symbols over a universe of discourse:

$$E = \{e_i \mid e_i \text{ propositional symbol}, 1 \leq i \leq n\}$$

Let Card denote the cardinality of a set. For the set E

$$\text{Card}(E) = n$$

Let $\mathbf{x}$ be a vector such that

$$\mathbf{x} = \{x_j \mid x_j \in \{0,1\}, 1 \leq j \leq \text{Card}\,(E)\}$$

Since the terms of a fact base (FB) are a subset of E, the fact base can be represented by vector $\mathbf{x}$ with its components assuming the corresponding values:

$$x_k = \begin{cases} 1, & \text{if the truth value of } e_k \text{ is } \textit{true} \\ 0, & \text{if not} \end{cases}$$

$$k = 1, \ldots, n$$

The vector $\mathbf{x}$ represents the fact base FB and it will be called a fact vector. The fact bases will be codified as fact vectors. An input fact base is denoted by $\mathbf{x}_i$, called input fact vector. The output fact base is denoted by $\mathbf{x}_o$, called output fact vector.

3.2. Rule Base Codification

The rule base used by the procedure is a mapping of its virtual representation. It will be represented by two operational matrices, **C** (the conjunctive matrix) and **D** (the disjunctive matrix).

The pair of matrices **C** and **D** is called the operational rule base since their structures store rules' information to be used effectively during inference. The inference procedure is derived from operational matrices, operators and the fact vector.

Definition 2: *propositional rule*. Let E be as in definition 1. A rule R is defined by the union of its syntatic and semantic structures. The syntatic structure of a rule is defined as a word sequence, each being a set of characters such as:

if ⟨antecedent⟩ then ⟨consequent⟩

where ⟨antecedent⟩ relates from 1 up to $(n-1)$ elements of E, conjunctively associated by the boolean operator $\wedge$ ('and'), and ⟨consequent⟩ corresponds to an element of E.

Example:

Let $E = \{e_1, e_2, \ldots, e_{10}\}$

Antecedent examples:

- e_1 and e_2 and e_5
- $e_1 \wedge e_2 \wedge e_5$
- e_7 and e_9
- e_3

Consequent examples:

- e_1
- e_7
- e_4

Complete rule example:

- if e_1 and e_4 then e_2

Let A be a subset of E such that the elements of A are members of the ⟨antecedent⟩ of rule R syntactic structure, that is:

$$A = \{a_i \mid a_i \in E, a_i \text{ being member of } \langle\text{antecedent}\rangle, 1 \leq i \leq \text{k} \quad k < n\}$$

Let Q be a singleton of E such that its element q is a consequent of rule R syntactic structure:

$$Q = \{q \mid q \in E, q \text{ is a consequent}\}$$

The semantic sturcture of a rule is defined by associating it to a propositional logic sentence as below:

$$\bigwedge_i (a_i) \Rightarrow q$$

where each a_i is an element of A, q is an element of Q, $\wedge$ is the conjunctive logic operator, and $\Rightarrow$ is the implication logic operator from the propositional logic.

This means that the truth of all elements of A

(rule antecendent) implies in the truth of q (rule consequent).

Definition 3: *virtual rule base.* A virtual rule base (VRB) is a set of rules R_j where $1 \leq j \leq m$, and m is the number of rules.

$$\text{VRB} = \{R_j \mid R_j \text{ is a rule}, 1 \leq j \leq m\}$$

Definition 4: *semantic content.* Let E be as in definition 1.

The semantic content of a rule R_j is defined as a function φ which maps all possible interpretations for E (I_E) in {false, true}, considering the logic sentence that corresponds to the semantic structure of rule R_j.

For $A_j \subset E$ and $Q_j \subset E$, if the semantic structure of R_j is:

$$(a_{j1} \wedge a_{j2} \wedge \ldots \wedge a_{jk}) \Rightarrow q_j$$

then:

$$\varphi(R_j,J_E) : ((a_{j1} \wedge a_{j2} \wedge \ldots \wedge a_{jk}) \Rightarrow q_j) \rightarrow \{\text{false, true}\}$$

Definition 4 can be extended to a rule base VRB by mapping all possible interpretations of E in {false, true}, considering the logic sentence which corresponds to the union of the semantic content in each individual rule of VRB:

$$\varphi(\text{VRB},I_E)=\varphi(R_1,I_E)\vee \varphi(R_2,I_E) \vee\ldots\vee \varphi(R_m,I_E)$$

A pair of matrices **C** (the conjunctive matrix) and **D** (the disjunctive matrix), as defined below, makes up an operational rule base which will effectively perform inference.

3.3. C Matrix Structure

In matrix **C**, each row is associated to the list of antecedent terms of a rule in VRB, with each column representing one symbol. Thus, the **C** matrix has dimension $m \times n$, where m is the number of rules and n is the number of symbols in VRB.

Definition 5: **C** *matrix structure.* Let E be as in definition 1. Let VRB be a virtual rule base. Let R_i be the ith rule of VRB. Let A_i be the set of terms which are members of rule R_i antecedent. The **C** matrix structure is defined as:

$$\mathbf{C} = \{c_{ij} \mid c_{ij} = f(i,j,), 1 \leq i \leq m, 1 \leq j \leq n\}$$

$$f(i,j) = \begin{cases} 1 & \text{if } e_j \in A_i \\ 0 & \text{if not} \end{cases}$$

3.4. D Matrix Structure

The ith row of matrix **D** represents rules which have the symbol i as a consequent. Therefore, the **D** matrix has dimension $n \times m$, where n is the number of symbols and m is the number of rules in the rule base.

Definition 6: **D** *matrix structure.* Let E be as in definition 1. Let VRB be a virtual rule base. Let R_j be the jth rule of VRB. Let Q_j be the set of terms which are members of rule R_j consequent.

The structure of matrix **D** is defined by

$$\mathbf{D} = \{d_{ij} \mid d_{ij} = g(i,j), 1 \leq i \leq n, 1 \leq j \leq m\}$$

$$g(i,j) = \begin{cases} i & \text{if } e_i \in Q_j \\ 0 & \text{if not} \end{cases}$$

Definition 7*: rule base coding.* By rule coding it is meant a modification of rule's syntactic structure, but maintaining its semantic content. Therefore, rule base coding is a modification at the rule base representation, preserving its semantic content and meaning.

Let VRB be a virtual rule base. Let **C** ($m \times n$) and **D** ($n \times m$) be, respectively, the conjunctive and disjunctive matrices. A rule base VRB codification is defined as a mapping:

$$\text{Cod}(\text{VRB},\{\mathbf{C},\mathbf{D}\}) : \text{VRB} \rightarrow \{\mathbf{C},\mathbf{D}\},$$

$$\varphi(\text{VRB},I_\text{E}) = \varphi(\{\mathbf{C},\mathbf{D}\},I_\text{E}),\forall I_\text{E}$$

Actual implementation of the inference procedure uses **C** and **D** matrix representation for knowledge processing, since the semantic content is the same of VRB, that is, $\varphi(\text{VRB},I_E)=\varphi(\{\mathbf{C},\mathbf{D}\},I_E),\forall I_E$. This equivalence is useful because at the level of a man–machine interface (for knowledge acquisition and validation, for instance), the virtual representation is more appropriate. For real-time processing, **C** and **D** matrix representation provides much better computing performance.

Note that, once the rule base is defined, **C** and **D** matrices can be determined off-line.

3.5. Conjunctive Matrix Operator

The conjunctive matrix operator is an operator which does the disjunction of the rules consequents. Its definition is similar to the matrix multiplication operator, changing the algebraic product and sum by the corresponding boolean product and sum.

Definition 8: *conjunctive matrix operator.* Let $\wedge$ be the conjunctive boolean scalar operator ('and'). Let $\vee$ be the disjunctive boolean scalar operator ('or').

Let **M** be a matrix of dimension $n \times m$ with boolean elements m_{ij}. Let **x** be a boolean vector of dimension m. Let **y** be a boolean vector of dimension n.

The conjunctive matrix operator $\dot{\wedge}$ is defined by

$$\mathbf{y} = \dot{\wedge}(\mathbf{M},\mathbf{x}) = M\dot{\wedge}\mathbf{x} = \{y_i | y_i = \bigvee_j (m_{ij} \wedge x_j)\}$$

where $\bigvee_j$ is the boolean summation, that is

$$\bigvee_j z_j = z_1 \vee z_2 \vee \ldots \vee z_m$$

3.6. Disjunctive Matrix Operator

The disjunctive matrix operator does the conjunction of antecedent terms. It is dual to the conjunctive matrix operator, i.e. we take the same definition, changing the boolean scalar operators $\wedge$ to $\vee$ and vice versa.

Definition 9: *disjunctive matrix operator.* Let $\wedge$ be the boolean scalar conjunctive operator ('and'). Let $\vee$ be the boolean scalar disjunctive operator ('or'). Let **M** be a matrix of dimension $m \times n$ with boolean elements m_{ij}. Let **x** be a boolean vector of dimension n. Let **y** be a boolean vector of dimension m.

We define the disjunctive matrix operator $\dot{\vee}$ as follows:

$$\mathbf{y} = \dot{\vee}(\mathbf{M},\mathbf{x}) = \mathbf{M}\dot{\vee}\mathbf{x} = \{y_i | y_i = \bigwedge_j (m_{ij} \vee x_j)\}$$

where $\bigwedge_j$ is the boolean product, that is

$$\bigwedge_j z_j = z_1 \wedge z_2 \wedge \ldots \wedge z_n$$

3.7. Inference Step

During inference, it is necessary to define a step where all rules are applied to a fact base, reassigning its truth values to the inferred ones. This scheme is defined as an inference step.

Definition 10: *inference step.* Let **C** and **D** be the conjunctive and disjunctive matrices respectively, which represents the operational rule base. Let $\dot{\wedge}$ and $\dot{\vee}$ be the conjunctive and disjunctive matrix operators. Let Neg (**A**)=$\bar{\mathbf{A}}$ be the elementwise operation of boolean negation, for matrices. Let **x** +**y** be the elementwise boolean summation for vectors. Let x_i and x_o be two fact vectors. The inference step is defined by the following equation:

$$\mathbf{x}_o = \text{InferenceStep}(\mathbf{x}_i) = \mathbf{x}_i + \mathbf{D}\dot{\wedge}(\bar{\mathbf{C}}\dot{\vee}\mathbf{x}_i)$$

Theorem 1: *inference step.* Let VRB be a virtual rule base. Let **C** and **D** be the codification of VRB into the conjunctive and disjunctive matrices. Let $\dot{\wedge}$ and $\dot{\vee}$ be the conjunctive and disjunctive matrix operators. Let + be the vector boolean summation. Let Neg(**A**)= $\bar{\mathbf{A}}$ be the elementwise operation of boolean negation for matrices. Let $\mathbf{x}_i$ be the input fact vector and $\mathbf{x}_o$ the output fact vector. Starting with a generic input fact vector and applying the following formula:

$$\mathbf{x}_o = \mathbf{x}_i + \mathbf{D}\dot{\wedge}(\bar{\mathbf{C}}\dot{\vee}\mathbf{x}i)$$

an output fact vector $\mathbf{x}_o$ is obtained such that the truth value of all facts that can be inferred by applying rules of VRB over the facts represented by $\mathbf{x}_i$, including input facts truth values, are contained in $\mathbf{x}_o$.

Proof. The proof of the theorem is divided in two parts. In the first part, it is proven that the operation $(\bar{\mathbf{C}}\dot{\vee}\mathbf{x}_i)$ leads to an intermediary vector **y**, which holds the truth value of the antecedent of each rule. In the second part, it is proven that with the truth value of each rule antecedent and by performing the operation $\mathbf{D}\dot{\wedge}\mathbf{y}$, we get, by 'modus ponens', the truth value of all entities which are members of the consequents of rules. Therefore, one obtains all facts that can be inferred when starting with an input fact vector $\mathbf{x}_i$. The boolean vector summation of this result with the original input fact vector produces an output fact vector which holds the truth value of all entities from the universe of discourse, after application of VRB.

Part 1: By definition of disjunctive matrix operator, we have

$$\mathbf{y} = \bar{\mathbf{C}}\dot{\vee}\mathbf{x} = \{y_i | y_i = \bigwedge_j (c_{ij} \vee x_j)\}$$

The value of **y** is the boolean product:

$$\mathbf{y} = (\bar{c}_{i1} \vee x_1) \wedge (\bar{c}_{i2} \vee x_2) \wedge \ldots \wedge (\bar{c}_{in} \vee x_n)$$

By the definition of matrix **C**, we have that $c_{ij} = 1$ if entity e_j belongs to the antecedent of rule R_i, being 0 if not. If $c_{ij} = 0$, then $\bar{c}_{ij} = 1$. In those cases, the term $(\bar{c}_{ij} \vee x_j)$ will always be 1, for any value of x_j. Thus, the value of those terms will not change the result of the boolean product. Considering a set K_i^C, of indices k such that $\bar{c}_{ik} = 1$, and a set L_i^C of indices l that $\bar{c}_{il} = 0$, we can split the boolean product into two parts:

$$y_i = (\bigwedge_k (\bar{c}_{ik} \vee x_k)) \wedge (\bigwedge_l (\bar{c}_{il} \vee x_l))$$

where $\bar{c}_{ik}$ is always 1 $\forall k \in K_i^C$ and $\bar{c}_{il}$ is always 0 $\forall l \in L_i^C$.

Since $(\bar{c}_{ik} \vee x_k)$ is always equal to 1, $\forall k \in K_i^C$, we have that

$$(\bigwedge_k (\bar{c}_{ik} \vee x_k)) = 1$$

and therefore

$$y_i = (\bigwedge_l (\bar{c}_{il} \vee x_l))$$

On the other hand, since $\bar{c}_{il}$ is always equal 0, $\forall l \in L_i^C$, we have that

$$y_i = \bigwedge_l x_l$$

We must remember that if $\bar{c}_{il} = 0$, ($c_{il} = 1$) then e_l belongs to the antecedent of rule R_i (by definition), $\forall l \in L_i^C$. Therefore, the truth value of y_i can only be 1 if all values of x_l are 1. Since all entities e_1 belong to the antecedent of rule R_i, this means that the conjunction of all x_l is equivalent to the conjunction of the truth value of entities which are members of the antecedent term of rule R_i. Then the value of y_i is equivalent to the truth value of the antecedent of ruleR_i.

Part 2: From the definition of conjunctive matrix operator,

$$\mathbf{x}_0 = \mathbf{D} \dot{\wedge} \mathbf{y} = \{x_{0i} \mid x_{0i} = \bigvee_j (d_{ij} \wedge y_j)\}$$

From **D** matrix structure definition, $d_{ij} = 1$ if e_i belongs to the consequent of rule R_j and 0 if not. Considering a set K_i^D, of indices k so that $d_{ik} = 1$, and a set L_i^D, of indices l so that $d_{il} = 0$, we can split the boolean summation in two parts:

$$x_{oi} = (\bigvee_k (d_{ik} \wedge y_k)) \vee (\bigvee_l (d_{il} \wedge y_l))$$

where d_{ik} is always 1, $\forall k \in K_i^D$ and d_{il} is always 0, $\forall l \in L_i^D$.

If d_{il} is always 0, the term $(d_{il} \wedge y_l)$ is always 0, for any value y_l, $\forall l \in L_i^D$.

Thus, we have that

$$\bigvee_l (d_{il} \wedge y_l) = 0$$

and therefore

$$x_{oi} = (\bigvee_k (d_{ik} \wedge y_k))$$

Since d_{ik} is also always equal to 1, $\forall k \in K_i^D$:

$$x_{oi} = \bigvee_k y_k \qquad (1)$$

OBS – modus ponens: By 'modus ponens' rule of inference it is meant that

$$\frac{A \, , A \Rightarrow B}{B}$$

i.e. if we have A, and there exists a relation that A implies B, then we can deduce B.

Remembering that if $d_{ik} = 1$, then $\forall k \in K_i^D$, $e_i \in Q_k$ (where Q_k keeps the consequent of rule k), which means that e_i is the consequent of rule R_k.

The semantic content or rule R_k can be rewritten as follows:

$$y_k \Rightarrow e_i$$

here, for $\forall k \in K_i^D$, the expression $y_k \Rightarrow e_i$ becomes true, in other words, it is a valid formula.

By 'modus ponens':

$$\frac{y_k \, , y_k \Rightarrow e_i}{e_i}$$

for all k belonging to $K^D{}_i$. Then e_i can be deduced from rule k if $y_k = 1$.

Since $y_k = 1$, $\forall k \in K_i^D$, we have by definition that $x_i = 1$ and therefore

$$x_{oi} = \bigvee_k y_k$$

which is equivalent to (1).

This proves the second part of the theorem and completes the theorem proof.

3.8. Inference Cycle

The inference cycle is necessary to provide the chaining among rules, when it exists. An inference cycle is the successive application of inference steps, taking as input to each step the output of the last step, until it does not change anymore.

Definition 11: *inference cycle.* The inference cycle is defined as a finite number of successive applications of inference steps over a fact vector, taking as input in each step the output of the last step. Let $\mathbf{x}_i$ be the input fact vector, $\mathbf{x}_o$ the output fact vector, and $\mathbf{x}$ an auxiliary fact vector. Let $(:=(a,b) \equiv a := b)$ be the vector attributive operator, defined as attributing to each element of the left side parameter, the value of the element with same index from the right side: $a[i] = b[i]$, $\forall i$. Let $\neq$ be the vector unequal operator, defined as

$$\neq(x,y) \equiv \mathrm{x} \neq \mathrm{y} = \begin{cases} \text{false} & \text{if } x_i = y_i, \forall \, \mathrm{i} \in [1,n] \\ \text{true} & \text{if not} \end{cases}$$

The inference cycle is the following procedure:

```
procedure INFERENCE_CYCLE
BEGIN
    x_o := x_i;
    WHILE (x_o ≠ x)
    BEGIN
        x := x_o;
        x_o := InferenceStep(x);
    END
END
```

The inference cycle is the kernel of the inference engine. After an inference cycle, the output fact vector holds the result of a complete inference.

Theorem 2: *theorem of inference cycle.* Let VRB be a virtual rule base. Let **C** and **D** be the codification of VRB through the conjunctive and disjunctive matrices. Let $\dot{\wedge}$ and $\dot{\vee}$ be the conjunctive and disjunctive matrix operators. Let + be the boolean summation vector operator. Let $\mathbf{x}_i$ be the input fact vector and $\mathbf{x}_o$ be the output fact vector. Let P be the operator equivalent to an inference step:

$$P(\mathbf{x}_i) = \mathbf{x}_i + \mathbf{D} \dot{\wedge} (\bar{\mathbf{C}} \dot{\vee} \mathbf{x}_i)$$

The successive application of P, taking as input of subsequent applications the last output, will result in the complete inference from a virtual rule base over a generic interpretation represented by $\mathbf{x}_i$.

Proof: By Theorem 1, the inference step is equivalent to the application of all rules of a virtual rule base VRB over a generic fact vector. The successive application of the P operator over a fact vector $\mathbf{x}_i$, taking the output of each operation as input to the next one, is equivalent to the successive application of all rules from a rule base over a fact base. If there exists chaining between rules (for example, forming a chain of g levels), after applying the P operator g times, all activated rules are in fact fired. Then a complete inference over such interpretation is derived.

4. Knowledge Base Analysis and Manipulation

Knowledge-based systems structure was conceived to separate knowledge from its manipulation. The knowledge manipulation performance depends heavily on how the knowledge base is represented for processing [14]. For real-time systems this is particularly important, because it will run under time constraints. Therefore, a methodology for rule base analysis and manipulation is needed. The purpose of this section is to address this question.

4.1. Rule Base Analysis

The most critical part of a knowledge base is the rule base, because it provides the most opportunities for reorganization to achieve better performance. It has been shown that a rule base can be described as the **C**, **D** pair. It would be interesting to manipulate these matrices as an attempt to get a new pair of matrices, with the same semantic content, but with reduced dimensions for increased performance.

One important question to be considered is rule chaining. Rule chaining occurs when an antecedent term of one rule is also a consequent of another rule. Here, the application of just one inference step will not provide complete inference. To have complete inference, several inference steps are necessary. Given a base with m rules and several chains, at most m steps of inference will be necessary, which is the case when all the rules are chained.

Chaining elimination seems to be a key to reducing and predicting processing time.

Another factor that should be considered is the need to preserve all entities used in the original rule base throughout the inference procedure. If we want to preserve the classic logic structure, this is necessary to maintain the coherence of the theorems that define the inference. Now, if our goal is to apply logic for control purposes, where the main issue is not the manipulation of pure mathematical logic but the effective use of knowledge for process control, we can relax the intermediate information, since it will not be relevant to the final control action. Therefore, the information to be processed can be reduced, which improves system performance in terms of processing time and memory space requirements. To deal with this aspect, we propose a method for rule base compression with deletion of intermediate information.

To address these issues, we begin by presenting a method for horizontal expansion, and then a method for rule base compression.

4.2. Rule Base Horizontal Expansion

The goal of the rule base horizontal expansion method is to eliminate chaining from the rule base, to avoid it during the on-line inference processing. The main idea is to transform a rule base with chaining into another rule base without chaining, but with the same semantic content.

The method consists in including additional rules in the rule base. The added rules are those generated by the chaining of other rules. Then, instead of applying several inference steps, only one is necessary to get the same result as if all the rules would be applied. Therefore, once we have a horizontal rule base, only one inference step is enough for complete inference.

Starting with the **C** and **D** matrices, the horizontal expansion method generates two new matrices $\mathbf{C}'$ and $\mathbf{D}'$.

We know that if chaining exists it is generated by a term that is used in the consequent of the

rule. First, the method swaps the rule base to verify the rule's consequents. Second it verifies if the rule's consequents appear as antecedents of any rule. If so, a new rule is generated by syllogism, and added to the new rule set.

The **C** matrix structure is specially interesting for performing the horizontal expansion, since the swap for a term can be made by just searching the column relative to that term to verify if c_{ij} is equal to 1 or not.

Definition 12: *horizontal rule base.* We define a horizontal rule base as a rule base for which an inference cycle corresponds to just one inference step.

Definition 13: *horizontal expansion procedure.* Let $E = \{e_i\}$ be the universe of discourse. Let VRB be a virtual rule base. Let **C** and **D** be the operational matrices that codify the virtual rule base VRB. Let C_{i*} be the ith row and C_{*j} be the jth column of matrix **C**. Let D_{i*} be the ith row and D_{*j} be the jth column of matrix **D**. Let m be the number of rules of VRB, in other words, the number of rows of matrix **C**. let m' be the number of rules of VRB after (or during) the horizontal expansion procedure.

We define the horizontal expansion of a rule base by the following procedure:

```
procedure HORIZONTAL_EXPANSION
VAR
  i,j : indices;
BEGIN
  m' :=m;
  FOR i = 1 TO m'
  BEGIN
   FOR j = 1 TO m
     BEGIN
     Find k as d_ki = 1;                    (2)
      IF c_jk = 1                           (3)
      BEGIN
        C_(m'+1)* := C_i* + C_j*;           (4)
         c_(m'+1)k :=0;                     (5)
        D_*(m'+1) :=D_*j;                   (6)
         m':=m'+1;
      END
     END
  END
END
```

Theorem 3: *horizontal expansion.* Let VRB be a virtual rule base, codified as its corresponding operational matrices **C** and **D**. Then, the operational matrices **C**′ and **D**′ generated by the horizontal expansion procedure correspond to a virtual rule base VRB′, which has the same semantic content of VRB. **C**′ and **D**′ are a horizontal rule base.

Proof: The horizontal expansion procedure searches all the consequents of rules (2), verifying if they are antecedents of other rules (3). If they are, it adds the antecedent terms of the rule analysed with these of the base set (4). In the sequel, it takes off the entity which is the consequent of the rule from this set (5) and places the consequent of the base set as the consequent of the new rule. This is equivalent to the syllogism of the two rules. Each rule added in the base set eliminates one possible chaining that would be necessary for complete inference. This implies that, after the global search, no new chaining will be necessary. Therefore we get a horizontal rule base.

It should be noted that the method increases the size of the rule base, since $m' \geq m$. This means increasing storage space, but since chaining is avoided, the processing time is predictable. There is a trade-off between processing time and storage space to be considered. However, if a compression procedure is also applied in the horizontal rule base, we finally get a rule base that may be smaller than the original rule base. This implies less processing time and fewer memory requirements.

4.3. Rule Base Compression

In knowledge-based control systems, sensor information is extracted from the process and translated into facts to be processed with the rules by the inference procedure. Because of inference, control decisions are generated and translated into control signals that drive the process. The control decisions are due to a control strategy defined by a relation coded in the rule base. From the input/output point of view, sensor information is processed by a relation to generate the control decision. This means that for control purposes, only the terms associated with the sensor information and the control decisions are needed to be explicitly represented in the rule base. Moreover, it is not necessary to have in the input fact vector the corresponding terms whose truth value is not known at the beginning of inference. The output fact vector does not have to contain the information that is already included in the input fact vector. Thus, the fact vector can be partitioned in three sub-vectors. The first sub-vector, called input sub-vector, has as components those terms associated with the sensor input. The intermediate sub-vector has as its components those terms that are neither associated with sensor input nor associated with control decisions. The output

sub-vector has as components those terms associated with the control decisions.

From the dicussion above we may conclude that if an efficient scheme is available to recode and to preserve the semantic content of a rule base with those characteristics, not only would the size of the rule base be diminished (which means less storage space) but also the processing time would decrease and the predictability increase. The rule base compression method provides such a recoding scheme. After horizontal expansion, the rule base compression method first discards the intermediate terms from the input and output fact vectors. The corresponding **C** matrix columns and **D** matrix rows are also discarded. Next, the terms of the output sub-vector are eliminated from the input fact vector, and the terms of the input sub-vector are eliminated from the output fact vector. The **C** matrix columns corresponding to the terms of the output sub-vector and the **D** matrix rows corresponding to the terms of the input sub-vectors are deleted. We then finally get a compressed rule base, which recodes the original rule base and preserves its meaning. With this recoding procedure, an inference step provides complete inference (no cycles are necessary) since the inference procedure becomes:

$$\mathbf{x}'_o = \mathbf{D}'' \dot{\wedge} (\bar{\mathbf{C}}'' \dot{\vee} \mathbf{x}'_i) \tag{7}$$

Definition 14: *rule base compression.* Let VRB be a virtual rule base. Let VRB′ be a horizontal rule base, generated by horizontal expansion of virtual rule base VRB. Let **C** and **D** be the matrices which codify horizontal virtual rule base VRB′. Let E be the universe of discourse. Let A_j be the set of entities which are members of the antecedent term of rule R_j. Let A be the set which is the result of the union of all sets A_j, i.e. the set of all entities which are used as antecedent terms. Let Q_j be the set of entities which are members of the consequent term of rule R_j. Let Q be the set generated by the union of all sets Q_j, i.e. the set of all entities which were used as rule consequents.

Let N be the set of input entities, i.e. the set of entities which are used only as antecedent terms of VRB′:

$$N = \{n_i \mid n_i \in A, n_i \notin Q, 1 \leq i \leq \text{Card}(A-(A\cap Q))\} \tag{8}$$

Let I be the set of intermediate entities, i.e. the set of entities which are both used in antecedent and consequent terms of the VRB′ rules:

$$I=\{i_k \mid i_k \in A, i_k \in Q, 1 \leq k \leq \text{Card}\,(A \cap Q)\} \tag{9}$$

Let S be the set of output entities, i.e. the set of entities used only as consequent terms of the VRB′ rules:

$$S=\{s_l \mid s_l \in Q, s_l \notin A, 1 \leq l \leq \text{Card}\,(Q-(A\cap Q))\} \tag{10}$$

Thus $E = N \cup I \cup S$.

Let $\mathbf{x}_i$ be the input fact vector and $\mathbf{x}_o$ the output fact vector.

We define the compression procedure by the following:

procedure COMPRESSION
BEGIN
Discard all $e_i \in I$ from $\mathbf{x}_i$ and $\mathbf{x}_o$. (11)
Discard from **C** all rows C_{i*} as $\exists j(e_j \in I \wedge c_{ij} = 1)$. (12)
Discard from **D** all related columns D_{*i} (13)
Discard from **C** all columns C_{*j} as $e_j \in I$. (14)
Discard from **D** all columns D_{*j} as $\exists i(e_i \in I \wedge d_{ij} = 1)$ (15)
Discard from **C** all related rows C_{j*}. (16)
Discard from **D** all rows D_{i*} as $e_i \in I$ (17)
Discard all $e_k \in S$ from $\mathbf{x}_i$. (18)
Discard all $e_l \in N$ from $\mathbf{x}_o$. (19)
Discard from **C** all columns C_{*j} as $e_j \in S$ (20)
Discard from **D** all rows D_{i*} as $e_i \in N$. (21)
END

Discarding an element from a fact vector means the suppression of its corresponding components from the vector. Let **x** be a fact vector:

$$\mathbf{x} = [x_1, x_2, \ldots, x_{p-1}, x_p, x_{p+1}, \ldots, x_n]$$

To discard term e_p, represented by x_p, from vector **x** corresponds to create a vector

$$x' = [x_1, x_2, \ldots, x_{p-1}, x_{p+1}, \ldots, x_n]$$

where the x' dimension is the dimension of x decreased by 1.

To discard a row or a column from a matrix, corresponds to suppressing a row or a column.

It must be observed that the compression procedure changes the indices associated with each term. Therefore, the related table of symbols must also be modified accordingly.

Theorem 4: *compression*. Let VRB be a virtual rule base. Let VRB′ be the horizontal expansion of VRB. Let **C**′ and **D**′ be the operational matrices which codify VRB′. Let N be the set of input entities, I be the set of intermediate entities and S be the set of output entities. Let $\mathbf{x}_i$ be the input fact vector. Let $\mathbf{x}_o$ be the output fact vector. Let $\mathbf{x}'_i$ be the input fact vector modified by compression (dimension of N). Let $\mathbf{x}'_o$ be the output fact vector modified by compression (dimension of S). Let

Comp($\mathbf{C}'$,$\mathbf{D}'$,$\mathbf{x}_i$,$\mathbf{x}_o$,$\mathbf{C}''$,$\mathbf{D}''$,$\mathbf{x}_i'$,$\mathbf{x}_f'$) be the compression procedure which transforms VRB′ into VRB″ as in definition 14.

Let $\mathbf{C}''$ and $\mathbf{D}''$ be the operational matrices which codify compressed rule base VRB″.

The truth value held by each element of $\mathbf{x}_o'$ through the modified inference procedure, using $\mathbf{C}''$ and $\mathbf{D}''$

$$\mathbf{x}_o' = \mathrm{D}'' \dot{\wedge} (\bar{\mathbf{C}}'' \dot{\vee} \mathbf{x}_i')$$

is the same as those that would be assigned to the related terms of $\mathbf{x}_o$, using the procedure based on $\mathbf{C}$ and $\mathbf{D}$ matrices and input fact vector $\mathbf{x}_i$.

Proof: After analysing the compression procedure, we see that in (11), we discard the intermediate entities from operational entities. In (12) and (13), we discard all rules that had in its antecedent term an intermediate entity. In (14) we discard from matrix $\mathbf{C}$ the dimension related to the intermediate entities as they will not be necessary and make the necessary adjustments. In (15) and (16), we discard the rules that have in its consequent an intermediate entity. In (17) we discard the dimension of matrix $\mathbf{D}$ related to the intermediate entities, making the necessary dimension adjustments.

Those steps are equivalent to the elimination of intermediate entities from fact vectors and matrices. This is the same as defining a new system, where the universe of discourse is reduced to ($N \cup S$) and the rules are similar to VRB′, except those that use an intermediate term. This will be an intermediate compression. Only those entities which are members of N can be assigned a value different from 0 ($\mathbf{x}_i'$ represents the terms which are members of N). Since only the rules that use intermediate entities were removed (and these rules could never be fired as the truth value of intermediate terms always starts with 0, by definition), the truth value assigned by the modified procedure to $\mathbf{C}'$ and $\mathbf{D}'$, will be the same as the conventional procedure in C and D. In (18) we discard from input fact vector, the entities of output sub-vector, and in (19) we discard from output fact vector, the entities of input sub-vector. In (20) we adjust the columns of C and in (21) we adjust the rows of D for the new representation.

Since the values of $\mathbf{x}_i$ are equivalent to the corresponding ones in $\mathbf{x}_i'$ and the others are 0, nothing is modified by the inference process, and the results provided by (7) are the same as those given by the procedure in definition 10. Discarding of input sub-vector terms from output fact vector does not modify inference either because they are not used in $\mathbf{x}_o'$.

Table 1.

Symbol	Index	Antecedent	Consequent	Type
e_1	1	Yes	No	*N*
e_2	2	Yes	No	*N*
e_3	3	Yes	Yes	*I*
e_4	4	Yes	No	*N*
e_5	5	Yes	Yes	*I*
e_6	6	Yes	No	*N*
e_7	7	No	Yes	*S*
e_8	8	No	Yes	*S*

5. Example

A simple example is now provided to illustrate the main features of the proposed procedure.

Let us consider the following rule base:

if e_1 and e_2 then e_3
if e_3 and e_4 then e_5
if e_1 and e_6 then e_5
if e_2 and e_6 then e_7
if e_5 and e_6 then e_7
if e_2 and e_4 then e_8

An encoding is defined (Table 1). Then the resulting $\mathbf{C}$ and $\mathbf{D}$ matrices are as follows:

$$\mathbf{C} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

After executing the horizontal expansion procedure, the matrices $\mathbf{C}'$ and $\mathbf{D}$ generated are

$$\mathbf{C}' = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\mathbf{D}' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Executing the compression procedure, we get a new symbol table to represent input fact vector and output fact vector (Table 2), and the corresponding compressed matrices are

$$\mathbf{C}'' = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \quad \mathbf{D}'' = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Assume that the signals collected from process were preprocessed, resulting in the data of Table 3.

Then the input fact vector is

$$\mathbf{x}_i' = [0\,1\,0\,1]$$

The result provided by the inference is

$$\mathbf{x}_o' = [1\,0]$$

The control action corresponding to e_7 can be executed by post-processing. The decision which corresponds to e_8 will not be executed.

Table 2.

Symbol	Index
Input fact vector	
e_1	1
e_2	2
e_4	3
e_6	4
Output fact vector	
e_7	1
e_8	2

Table 3. Data collected

Proposition	Truth value
e_1	False
e_2	True
e_4	False
e_6	True

6. Real-Time Requirements Analysis

Recalling the main requirements for real-time knowledge processing reviewed in the Section 2, the following can be stated. First, an efficient numeric/symbolic processing integration is required. In the scheme proposed this is achieved by introducing of the preprocessor module. Therefore, the control engineer and software designers may use any appropriate algorithm for information processing. The only constraint is to provide process information coded in the vector format to configure the fact base to be used for inference.

Regarding the focus of attention, the proposed procedure can easily consider structured rule bases, which is a mechanism to fulfil such a requirement. Here, each rule base can be coded into distinct **C** and **D** matrices. A base of meta-rules, which can be coded and operated by the same scheme, may be executed periodically to decide which rule base should be processed, depending on the process state. Setting up this or similar alternative strategies is quite simple because in order to perform context switching, only a pointer that refers to the appropriate matrices needs to be modified accordingly.

As far as the requirement of optimum use of environment is concerned, due to its structure, the inference procedure proposed can be implemented by pairwise bit processing, using standard instructions of programming languages. This affords pseudo-parallel implementation. Full parallel implementation is also readily achieved if the target machine has a parallel architecture. The mechanisms for rule base compression also allow an efficient use of memory.

Response time can be assured because, as the procedure is a set of boolean operations, the time to perform inference is easily obtained from the time needed to execute a set of primitive instructions. For instance, assuming that the procedure uses assignment, addition, comparison and increment as primitive instructions, and the corresponding execution time is A, S, C and I respectively, with m as the number of rules and n' as the average number of symbols per rule, procedure implementation could have the following coding scheme [15]:

for $(i=0;i<m;i++)$	$(A+mC+mI)$
$\{y=1;$	(A)
a=$CL[i][0];$	$(S+2A)$
for	
$(j=1;j<=a\&\&y==1;j++)$	$(A+2n'C+n'I)$
$\{b=x[CL[i][i]]$	$(3S+3A)$
$y=y\wedge b;$	$(C+A)$
$\}$	

```
b=&xf[DV[i]];                    (2S+A)
*b=*b\/y;                        (C+A)
}
```

where an account of each code line execution time is included on the right. The total execution time is such that

$$C_f(m,n') \leq A+2mC+mI+6mA+3mS \\ +3mn'C+mn'I+mn'4A+mn'3S$$

Given that each instruction time can be expressed as an integer multiple of a period T (for example, the clock rate), the inequality above may be rewritten as

$$C_f(m,n') \leq k_1T+k_2m.T+k_3mn'T=C_{fu}(m,n')$$

and the procedure order is $O(m,n')$, as shown by Fig. 3(a), (b). Note that $C_{fu}(m,n')$ is an upper bound for $C_f(m,n')$.

If pre-processing and post-processing execution times are C_i and C_o, respectively, RTKP execution time C_s is such that

$$C_s(m,n') \leq C_i+C_{fu}(m,n')+C_o=C_{su}(m,n')$$

$C_{su}(m,n')$ being an upper bound for C_s (m,n').

Typical real-time input and output tasks have little to no variance associated with their execution time because there are generally no data dependencies which can cause the execution time to vary. Then, for given m and n', the RTKP procedure is simply processed in a uniform, deterministic fashion in direct digital control applications, provided that $C_s(m,n')$ is consistent with process dynamics.

In structured knowledge bases, if $C_{s_i}(m_i,n_i')$ is set as the upper bound of the execution time for processing the m_i rules with n_i' symbols per rule of the ith knowledge base, the worst-case execution time $C_s(m,n')$ is

$$C_s(m,n') = \max_i \{C_{s_i}(m_i,n_i')\}$$

The RTKP procedure can also be used in embedded real-time supervisory control applications. In these cases, it must coexist with other real-time tasks on a common computing platform. The server concept introduced by the real-time scheduling researchers is directly applied here. This approach has also been used previously [11], but was based on different grounds. Servers have been developed [12,13] to provide highly responsive aperiodic performance over periodic, hard deadline environments. With this approach schedulability can be explicitly evaluated, that is, the level of resource utilisation, attainable before a deadline is missed, can be evaluated and guaranteed.

Generally, the priority assigned to the server depends on its response time requirements. For instance, in supervisory control of elevator systems to be addressed in the next section, the knowledge processing requirements are periodic, with a period T_s, of 0.5 s. Since the conventional real-time tasks are also periodic, the rate monotonic scheduling theory [15] can readily be used to evaluate schedulability of the task set. In essence, this theory ensures that since the CPU utilisation of all tasks lies below a certain bound and appropriate scheduling algorithms are used, all tasks will meet their deadlines without the programmer knowing exactly when any given task will be running. Even if a transient overload occurs, a fixed subset of critical tasks will still meet their deadline since their CPU utilisation lies within the appropriate bounds.

Given a set of independent periodic tasks, the rate monotonic scheduling algorithm gives each task a fixed priority and assigns higher priorities to tasks with shorter periods. A sufficient worst case condition that characterize schedulability of a task set under the rate monotone algorithm is provided by [15]

$$\frac{C_1}{T_1} + \ldots + \frac{C_n}{T_n} \leq n\,(2^{1/n} - 1) = U(n)$$

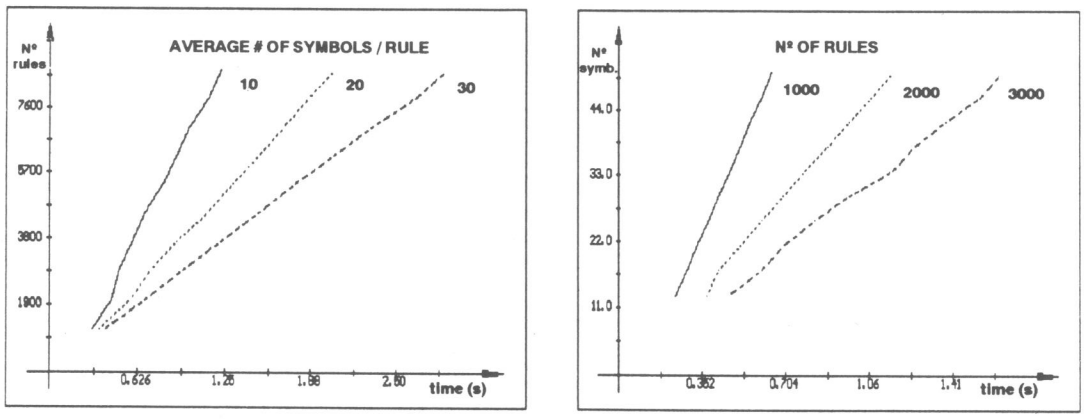

Fig. 3. Measured RTKP procedure time performance: (a) rules vs time; (b) symbols vs time.

where C_i and T_i are the execution time and period of task τ_i, respectively, and one task index is assigned to the server.

To check if a set of given tasks with greater utilisation than the bound provided above can meet its deadlines the critical zone theorem [16] should be considered. The critical zone theorem states that, for a set of independent periodic tasks, if each task meets its first deadline when all tasks are started simultaneously, then the deadlines will always be met for any combination of start-times. An equivalent mathematical test which can be calculated by an exact-case analysis [12] is

$$\forall i, 1 \leq i \leq n$$

$$\min_{(k,l)\in R_i} \sum_{j=1}^{i} C_j \frac{1}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil \leq 1$$

where C_j and T_j are the execution time and period of task τ_j, respectively, and

$$R_i = \{(k,l) \mid 1 \leq k \leq i, l = 1, \ldots, \lfloor T_i/T_k \rfloor\}$$

and one task index corresponds to the server.

This theorem provides the basis for an exact schedulability test for sets of independent periodic tasks under the rate monotonic algorithm. The theorem requires checking the scheduling points for a task. The scheduling points of a task τ are τ's first deadline and the ends of periods of high priority tasks before τ's first deadline.

Summing up, since the worst-case execution time of the RTKP procedure can be easily determined, and assuming that other tasks execution times are given, an analytical treatment to answer whether the server capacity is sufficient to meet the response time requirement is readily provided by the rate monotonic scheduling theory. The rate monotonic algorithm has been proven to be the optimal fixed-priority scheduling algorithm for periodic tasks.

Temporal data processing, although not natural within the theoretical framework considered in this paper, may be handled by either preprocessing or postprocessing modules, since they can include procedures to store current and past process states or to use models to get state estimations. Truth data maintenance can also be treated by the pre- and postprocessing modules. Therefore, the procedure proposed here is in close agreement with the basic requirements that a real-time knowledge processing system should fulfil.

Beyond the points above, real-time knowledge based systems are often viewed as computer programs, whose implementations, inevitably, make trade-offs between five sources [4]; processing power, response time, data space, inattention and degradation. The properties of the developed procedure clearly provide efficient guidelines to establish the desired trade-offs.

7. An Application Example: Group Supervisory Control of Elevator Systems

In traffic control of elevator systems two different control problems must be solved by a corresponding two level control hierarchy. The lower level task is to command each elevator to move up or down, to stop or start and to open and close the door. The higher level coordinates the movement of a group of elevators through a set of logical rules crafted to improve the system's performance. This problem is solved by means of a group supervisory control system with the aid of a group supervisory control strategy (the set of rules defining the control policy).

The main requirements of a group control system in serving both car and hall calls should be: to provide even service to every floor in a building; to minimise the time spent by passengers waiting for service; to minimise the time spent by passengers to move from one floor to another; to serve as many passengers as possible in a given time [3]. Due to the random nature of call times, call locations and the destination of passengers, problems are encountered in attempting to achieve the above requirements. Therefore, the control strategy must be able to follow changes in passenger demands, handling different traffic patterns and adapting itself to the traffic conditions.

A practical method widely used in group supervisory control systems consists of allocating cars to serve the building hall calls. Usually only new calls are allocated, remaining fixed once made. This method is known as call allocation strategy [3]. In car allocation, however, constraints must also be considered. For instance, a car may not pass a floor at which a passenger wishes to alight, a car may not reverse its direction of travel while carrying passengers, a hall call cannot be served by a car going in the reverse direction.

In what follows, the knowledge processing procedure developed in this paper is used to implement knowledge-based group supervisory control of elevator systems. This knowledge processing approach is particularly useful because:

- Call allocation strategies are frequently expressed by a set of rules provided by elevator designer experts.
- Different sets of rules can be easily grouped into

a structured knowledge base to cover such traffic conditions as up-peak, down-peak, heavy sector demand, heavy floor demand, balanced traffic, off-peak, etc.

- Logical constraints can be easily included in the knowledge base and processed equally.
- Meta-rules can easily be developed and incorporated to choose the car allocation rules most suited to a given traffic pattern.
- Response to system events can be predicted.

Production rules are particularly appropriate to derive both allocation stratgies and constraints. The adaptation of the allocation strategy to the traffic condition is handled by the focus of attention mechanism, using meta-rules for traffic pattern characterisation and rule base selection. Data acquistion and vector format information coding functions are performed by the preprocessor. Actual call allocation decisions and control signals, generated after inference, are computed by the postprocessor.

The supervisory group control simulation example presented below, which corresponds to the supervisory control level of Fig. 2, is a simplified version of a real world system described in [17]. Table 4 presents the data used for simulation purposes.

Examples of car allocation rules include:

```
IF elevator[E]_has_direction
THEN elevator[E]_has_target

IF elevator[E]_has_no_direction
THEN elevator[E]_has_no_target

IF elevator[E]_has_target
AND place[I]_in_field
AND place[I]_has_call
AND call_on_place[I]_has_same_direction_
  elevator[E]
ANDcall_on_place[I]_not_allocated
THEN Allocate_Call(E,I);
AND  elevator[E]_with_allocation_undefined
= FALSE;
```

Table 4. Elevator system characteristics.

Number of floors	7
Number of elevators	5
Elevator capacity	6 passengers each
Elevator velocity	3 m/s
Inter-floor distance	3 m
Door opening time	2 s
Door holding time	2 s
Door closing time	2 s
Total simulation time	550 s (9 min 10 s)
Traffic conditions	Up to 34 pass/min (peak)
Traffic patterns	Off, down and up-peak

```
IF elevator[E]_has_no_target
AND place[I]_up_in_field
AND place[I]_up_has_call
AND call_on_place[I]_up_not_allocated
AND elevator[E]_with_allocation_undefined
THEN Allocate_Call(E,I);
AND  elevator[E]_with_allocation_undefined
= FALSE;

IF elevator[E]_has_no_target
AND place[I]_down_in_field
AND place[I]_down_has_call
AND call_on_place[I]_down_not_allocated
AND elevator[E]_with_allocation_undefined
THEN Allocate_Call(E,I)
AND  elevator[E]_with_allocation_undefined
= FALSE;
```

Here, it will be demonstrated how to assign priority to the server and how to solve for its maximum capacity, consistently with the real-time tasks scheduling requirements. The example has a structured rule base for which

$$C_s(m,n') = 0.3\ \text{s}$$

corresponding to $m = 500$ (rules) and $n' = 20$ (average number of symbols/rule).

The priority assigned to the server is a function of its response time requirements. In the group supervisory control application the server processing requirements are periodic with a period $T_s = 0.5$ s. Since the conventional real-time tasks are also periodic with periods summarised in Table 5, the rate monotone scheduling algorithm is readily applied to evaluate the schedulability of the task set. A tight bound can be determined by the exact-case analysis formula, which yields a maximum C_s of 0.35, which corresponds to a maximum utilisation of the server of $U_s = C_s/T_s = 70\%$. Adding this to the utilisation of the other real-time tasks, a total schedulable utilisation of 96% is provided.

Therefore the set of tasks of Table 5 is schedulable because the run-time of the server is 0.3 s. This can also be checked, noting that, in the example, the sufficient condition

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_3}{T_3} = 0.87 \geq (2^{1/3}-1) = 0.78 = U(3)$$

does not hold and then the exact-case analysis must be performed. The scheduling points are 0.5, 1.0, 1.5, 2.0, 2.5 and 3.0, respectively. For the fourth scheduling point we have

$$4C_1 + C_2 + C_3 = 1.2 + 0.2 + 0.5 = 1.9 < 2.0$$

which means that the critical zone theorem holds.

Table 5. Execution characteristics of the run-time task set.

Task	task data		Rate monotonic theory	
	Period T_i (s)	Run-time C_i (s)	Utilisation U_i	Priority
Traffic monitoring	2.0	0.2	0.1	2
User interface	3.0	0.5	0.16	3
Supervisory control	0.5	0.3	0.6	1
Total			0.86	

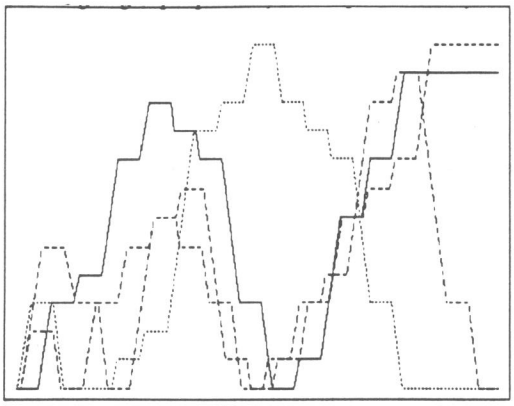

Fig. 4. Space vs time – RB strategy.

The actual schedulable utilisation of 86% is achieved with the server run-time of 0.3 s.

Additional results are provided by Figs 4 and 5, which show the elevator group space–time diagrams, and the average passenger waiting time given by the evaluation of waiting time for each call.

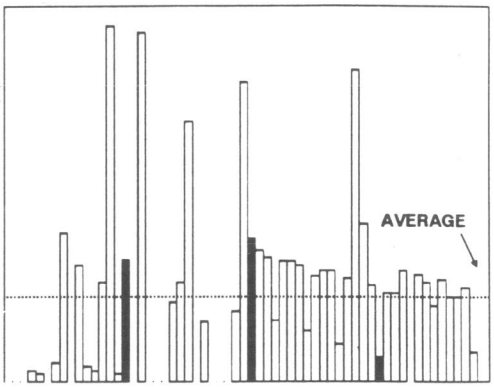

Fig. 5. Wait time vs call – RB strategy.

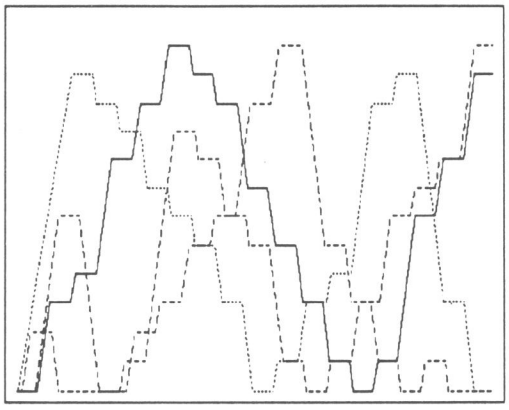

Fig. 6. Space vs time – conventional strategy.

Figures 6 and 7 show the same system under control of a conventional, fixed call allocation strategy. As it is shown, the intelligent group supervisory control provides a very attractive scheme for car allocation under time-varying traffic.

8. Conclusions

In this paper, a knowledge processing procedure, specially tailored for real-time applications, has

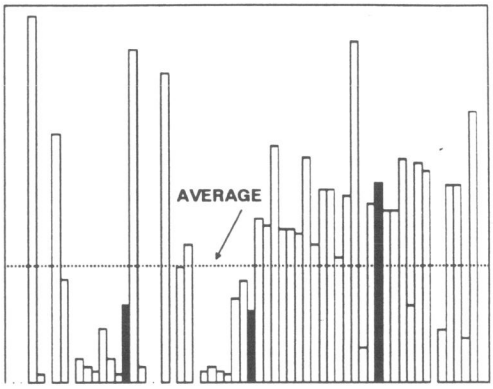

Fig. 7. Wait time vs call – conventional strategy.

been developed. Its main characteristics are in providing a processing scheme that closely agrees with the basic requirements for real-time knowledge-based systems. It also includes possibilities for rule base manipulation allowing important trade-offs between storage space, processing time and environment utilisation which are fundamental for implementing real-world cases.

An application concerning supervisory, group control of elevators was also included to illustrate the procedure potential.

Currently, the approach developed is being extended to consider its use within the framework of first order predicate logic, fuzzy logic, and threshold reasoning. We hope to address those items in a future paper.

Acknowledgements

The first two authors are grateful to CNP, the Brazilian National Research Council, for a fellowship and grant no. 300729/86-3, respectively.

References

1. Laffey TJ, Cox PA, Schmidt JL, Kao SM, Read JY. Real-time knowledge-based systems. AI Mag 1988; Spring
2. Aoki H, Sasaki K. Group supervisory control system assisted by artificial intelligence. Elev Wld 1990; February
3. Barney GC, dos Santos SM. Elevator traffic analysis design and control. In: IEE Control Engineering Series 2, 2nd Edn, Peter Peregrinus, London, 1985
4. Shoppers M. Real-time knowledge based control systems. Commun ACM 1991; 34(8)
5. Åström KJ, Arzen KE. Expert control. Automatica 1986; 22: 277–286
6. Kohn W. Declarative hierarchical controllers. Proc workshop on innovation approaches to planning, scheduling and control, Nov. 1990
7. Looney CG, Alfize AR. Logical controls via boolean rule matrix transformations. IEEE Trans Syst Man Cybern 1987; 17(6)
8. Looney CG. Fuzzy Petri nets for rule based decision making. IEEE Trans Syst Man Cybern 1988; 18(1)
9. Forgy CL. RETE: a fast algorithm for the many pattern/many object pattern match problem. Artif Intell 1982; 19: 17–37
10. Funabashi M, Mori K. Knowledge based control systems and software for building expert systems – 'EUREKA-II'. Hitachi Rev 1988; 37(4)
11. Paul CJ, Acharya A, Black B, Strosmider JK. Reducing problem solving variance to improve predictability. Commun ACM 1991; 34(8)
12. Sha L, Goodenough JB. Real-time scheduling theory and Ada. IEEE Comput 1990; April
13. Sprint B, Sha L, Lehoczky J. A periodic task scheduling for hard real-time systems. J Real-Time Syst 1989; 1(1)
14. Schalkoff R. Artificial intelligence: an engineering approach. McGraw-Hill, 1991
15. Gudwin RR. A kernel for real time knowledge processing. MsThesis, Campinas State University UNICAMP (in Portuguese)
16. Lin CL, Layland JW. Scheduling algorithms for multiprogramming in a hard real-time environment. J ACM 1973; 20(1)
17. Gudwin RR, Gomide FAC, Andrade Netto ML. An elevator supervisory group controller. Internal report RT-DCA/92 (in Portuguese)

J Syst Eng (1994)1:56

Journal of Systems Engineering

Book Review

'An Introduction to Fuzzy Control' *by D. Driankov, H. Hellendoorn and M. Reinfrank.* Springer-Verlag, Berlin, 1993, 316 pp. ISBN 3-540-56362-8

Over the past two decades, fuzzy sets theory has had a number of successful applications in difficult control problems. A factor in favour of fuzzy control techniques is that they are effective, but relatively straightforward to employ as they do not require accurate mathematical modelling of the plant to be controlled. This has created a great deal of interest among users and manufacturers of control systems. Research activities in fuzzy control have also been intense and many new results have emerged. These results can be found scattered in many research articles in specialist journals and conference proceedings. However, for practising systems engineers wishing to learn about fuzzy control, there is a lack of good introductory books covering the subject in a coherent manner. The authors have set out to produce such a book. In this, they have succeeded, for the book does indeed provide a lucid introduction to fuzzy control, concentrating on principles rather than individual applications or tools.

The book comprises six chapters. Following Chapter 1, which discusses the general benefits of and issues in fuzzy control and the cognate technique of knowledge-based control, Chapter 2 summarises useful background material on fuzzy logic, the mathematical foundation of fuzzy control. Chapter 3 then describes the main components and design parameters of a fuzzy control system and their effects on the performance of the system. Chapter 4 treats a fuzzy controller as a non-linear controller with a view to applying appropriate non-linear control techniques to its design and analysis. A technique considered in the chapter is sliding-mode control and its use to implement a force control system for contour-following is explained. Chapter 5 presents self-tuning and self-organizing fuzzy controllers and approaches to the design of adaptive fuzzy controllers. Chapter 6 is devoted to the analysis of stability and discusses basic approaches including the state-space, qualitative theory and input-output approaches. An extensive list of references is provided at the end of the book.

The book is clearly written and illustrated with many examples and diagrams (Chapter 2 in particular). To derive the maximum benefit from the book, the reader should have a good background in control engineering. A number of features would further enhance the value of the book as a text, for instance, the inclusion of end-of-chapter problems and of one or more chapters detailing some major case studies.

In conclusion, this is a very good introductory book on fuzzy control theory which can be strongly recommended to control engineers and postgraduate students in control engineering interested in learning about this practical and important subject.

D. T. Pham