Journal of the Korean Nuclear Society
Volume 28, Number 6, pp. 522~531, December 1996

A Method of Knowledge Base Verification for Nuclear Power Plant Expert Systems Using Extended Petri Nets

I. W. Kwon and P. H. Seong
Korea Advanced Institute of Science and Technology
Department of Nuclear Engineering
373-1 Kusong-dong, Yusong-gu, Taejon 305-701, Korea

(Received February 13, 1996)

Abstract

The adoption of expert systems mainly as operator supporting systems is becoming increasingly popular as the control algorithms of system become more and more sophisticated and complicated. The verification phase of knowledge base is an important part for developing reliable expert systems, especially in nuclear industry. Although several strategies or tools have been developed to perform potential error checking, they often neglect the reliability of verification methods. Because a Petri net provides a uniform mathematical formalization of knowledge base, it has been employed for knowledge base verification. In this work, we devise and suggest an automated tool, called COKEP(Checker Of Knowledge base using Extended Petri net), for detecting incorrectness, inconsistency, and incompletensess in a knowledge base. The scope of the verification problem is expanded to chained errors, unlike previous studies that assume error incidence to be limited to rule pairs only. In addition, we consider certainty factor in checking, because most of knowledge bases have certainty factors.

1. Introduction

The adoption of expert systems mainly as operator support systems is becoming gradually popular in nuclear industry as the control algorithms of nuclear power plant system become more and more sophisticated and complicated. As a result of this popularity, a large number of expert systems are developed, and most of these systems employ a rule-based formalism for knowledge representition since it is the simplest knowledge representation method to develop. In spite of this advantage, incorrectness, inconsistency, and incompleteness can be inadvertently brought into the knowledge base because it is often built in an incremental process. In other words, such anomalies may occur at any stage in the knowledge transfer process that is to transfer expertise from the human expert into the computer by the knowledge enqineers.

The traditional approaches to knowledge base verification, which have generally involved each rule comparison and decision context enumeration[1], are computationally expensive. They used domain-specific information in the verification process. Verification tools and algorithms developed later, such as COVADIS[2] and EVA[3], were base on a variety of approaches and were capable of detecting more subtle cases of anomalies. In recent days, a Petri net-based verification method was proposed[4]. A numerical Petri net was used to model knowledge base of production rules, and reachability analysis was then conducted to reveal

inconsistency and incompleteness in a knowledge base[5]. In PREPARE[6], anomalies in a knowledge base are defined in terms of the Pr/T net model. Then, these terms are indentified by using syntactic pattern recognition method.

In order to extend previous researches we consider verification of knowledge base having certainty factor, in global level as well as in local level. We devise and suggest an automated tool, called COKEP(Checker Of Knowledge base using Extended Petri net), for detecting incorrectness(redundant, subsumed, circular rules), incosistency(conflict rules), and incompleteness(unreachable conclusion, unreferenced conditions, isolated, omitted rules) in a knowledge base.

2. Extended Petri Nets(EPN)

In this section, we give formal definitions for the basic extended Petri net concepts. These basic concepts are used throughout our study of knowledge base verification.

2.1. EPN Structure

The notation employed in this work is based on that of the reference [7]. The original Petri net is composed of four parts: a set of place P, a set of transitions T, input functions I, and output functions O, The input and output functions are related to transitions and places. The input functions I are a mapping from a transition t_j to a collection of places $I(t_j)$, known as the *input places* of the transition. The output function O maps a transition t_j to a collection of places $O(t_j)$ known as the *output places* of the transition.

The petri net is extended by Derek because the net is used to rule base verification[8]. Derek introduced a transition place to the Petri net in order to use the net to knowledge verification. In this work, the input fuction is further divided into two input functions, I_1, I_2, in order to distinguish the initial known places from the common places(the output of other rules). This process is more realistic and practical because most knowledge base infers the goal after coming a new known fact, not common fact. The initial known places are the places that are inputted initially to verify the knowledge base system.

The structure of a Petri net is defined by its places, transitions, input functions, and putput functions. An EPN structure, C, is a six-tuple, $C(P, P', T, I_1, I_2, O)$. $P'=\{p'_1, p'_2, \cdots, p'_m\}$ is a finite set of transition state places, $m \geq 0$. Place p' is the transition state place that informs whether transition is fired or not. Input functions, in the extended Petri net, are classified into two types. Input functions I_1 are used for searching the path of chained rules, and input functions I_2 are used for finding the known facts of chained rules.

Input and output functions, I and O for each transition, are bags of places. A bag is a generalization of sets which allows multiple occurrences of an element in a bag. The use of bags, rather than sets, for the inputs and outputs of a transition allows a place to be a multiple input or a multiple output of a transition. The multiplicity of an input place p_i *for a transition* t_j is the number of occurrences of the place in the input bag of the transition, $\#(p_i, I_1(t_j))$ and $\#(p_i, I_2(t_j))$.

Similarly, the multiplicity of an output place p_i for a transition t_j is the number of occurrences of the place in the output bag of the transitions, $\#(p_i, O(t_j))$. If the input and output functions are sets(rather than bags), then the multiplicity of each place is either zero or one.

2.2. EPN Graphs

Most of the theoretical works on Petri nets are based on the formal definition of Petri net structures given above. However, a graphical representation of a Petri net structure is much more useful for illustrating the concepts of Petri net theory. An extended Petri net graph is a representation of an extended

Petri net structure as a bipartite directed multigraph. An extended Petri net structure consists of places and transitions. Corresponding to these, an extended Petri net graph has four types of nodes. A *circle* ○ represents a primitive input or output place; a *circle* ● represents a new known input place; a *circle* ◎ represents transition state place; a *bar* | represents a transition. Directed arcs(arrows) connect the places and the transitions, with some arcs directed from the places to the transitions and other arcs directed from transitions to places. An arc directed from a place p_i to a transition t_j defines the place to be an input of the transition. Multiple inputs to a transition are indicated by multiple arcs from the input places to the transition.

An extended Petri net is a *multigraph*, since it allows multiple arcs from one node of the graph to another. In addition, since the arcs are directed, it is a *directed* multigraph. Since the nodes of the graph can be partitioned into two sets(places and transitions), such that each arc is directed from an element of one set(place or transition) to an element of the other set(transition or place), it is a bipartite directed multigraph, $G=(V, A)$. $V=\{v_1, v_2, \cdots, v_s\}$ is a set of vertices and $A=\{a_1, a_2, \cdots, a_r\}$ is a bage of directed arcs, $a_i=(v_j, v_k)$, with $v_j, v_k \in V$. The set V can be partitioned into two disjoint sets P and T such that $V=P \cup T$, $P \cap T=\varphi$. And for each directed arc, $a_i \in A$, if $a_i=(v_j, v_k)$, then either $v_j \in P$ and $v_k \in T$ or $v_j \in T$, and $v_k \in P$.

2.3. EPN Marking

A *marking* μ is an assignment of tokens to the place of an EPN. A token is a primitive concept for Petri net(like places and transitions). Tokens are assigned to, and can be thought to reside in, the place of an EPN. The number and position of tokens may change during the execution of an EPN. The tokens are used to define the exection of an EPN. A marking μ of an EPN $C=(P, P', T, I_1, I_2, O)$ is a function from the set of places P to the nonnegative integers N. $\mu : P \rightarrow N$.

The marking μ can also be defined as a n-vector, $\mu=(\mu_1, \mu_2, \cdots, \mu_n)$, where $n=|P|$ and each $\mu_i \in N$, $i=1, \cdots, n$. The vector μ gives for each place p_i in a extended Petri net the number of tokens in that places. The number of tokens in place p_i is μ_i, $i=1, \cdots, n$. The definitions of a marking as a function and as a vector are obviously related by $\mu(p_i)=\mu_i$. The function is somewhat more general and so is more commonly used.

A *marked Petri net* $M=(C, \mu)$ is an EPN structure $C=(P, P', T, I_1, I_2, O)$ and a marking μ. This is also sometimes written as $M=(P, P', T, I_1, I_2, O, \mu)$. On an EPN graph, tokens are represented by small dots ● in the circles which represent the places of a Petri net. Since the number of tokens which may be assigned to a place of a Petri net is unbounded, there can be infinite number of markings for a Petri net. The set of all markings for an extended Petri net with n places is the set of all n-vectors.

2.4. Execution Rules for Extended Petri Nets

The execution of the extended Petri net is controlled by the number and distribution of tokens in the extended Petri net. The extended Petri net executed by firing transitions. A transition can be fired by removing tokens from its input places and creating new tokens which are distributed to its output places.

A transition is enabled if each of its input place and transition state place has at least one token in it which is connected by arcs from the place to the transition. Multiple tokens are needed for multiple input arcs. A transition $t_j \in T$ in a extended Petri net $C=(P, P', T, I_1, I_2, O)$ with marking μ is enabled if for all $p_i \in P$,

$$\mu(p_i) \geq \#[p_i, I_1(t_j)] + \#[p_i, I_2(t_j)]$$

$$\text{and } \mu(p'_i) \geq \{1\}.$$

A transition firing removes all of its enabling tokens from its input places and transition place, then

deposes one token into each of its output places. Multiple tokens are produced for multiple output arcs. Firing a transition will in general change the marking μ of the extended Petri net to a new marking μ'. A transition t_j in a marked Petri net with marking μ may fire, whenever it is enabled. Firing an enabled transition t_j results in a new marking μ' defined by

$$\mu'(p_i) = \mu(p_i) - \#[p_i, I_1(t_j)] - \#[p_i, I_2(t_j)] + \#[p_i, O(t_j)]$$

Transition firings can continue as long as there exists at least one enabled transition. When there are no enabled transitions, the execution halts.

3. Anomalies in Knowledge Base

The anomalies in knowledge base can be divided into three types, that are incorrectness, inconsistency, and incompleteness. They are described as follows :

Incorrectness

Redundant rules : Two rules are redundant if they contain the same set of conditions(but conditions may be arranged in a different order) and have the same conclusion, then one of them is said to be *redundant*. For example, one of the following two rules are redundant :

1) IF[a(x) and b(y)] THEN c(z) (0.7)
2) IF[b(y) and a(x)] THEN c(z) (0.7)

where *x*, *y*, and *z* are variables, and *a* and *c* are logical relationships.

Subsumed rules : One rule is subsumed by another if the two rules have the same conclusions, but one contains additional constraints on the conditions. Then, it is called a subsumed rule. The more restrictive rule can be subsumed by the less restrictive rule. Consider the following rules :

1) IF[a(x) and a(y)] THEN c(z) (0.8).
2) IF a(x) THEN c(z) (0.8).

Rule *1)* can be subsumed by rule *2)*. Whenever the more restrictive rule succeeds, the less restrictive rule also succeeds, resulting in redundancy.

Circular rules : A set of rules is circular if the chaining of rules in a set forms a cycle. Consider following rules :

1) IF a(x) THEN b(x) (0,8)
2) IF b(x) THEN c(x) (0,8)
3) IF c(x) THEN a(x) (0,8).

The systems enter infinite loop at run time unless the system has a special way of handling circular rules. Also, this definition includes the possibility of a single rule to form a cycle, i.e., *IF a(x) THEN a(x)*.

Inconsistency

Conflicting rules : Two rules are conflicting if they have the same conditions but with conflicting conclusions, consider the following rules :

1) IF a(x) THEN b(y) (0.8)
2) IF a(x) THEN c(z) (0.9).

Rule *1)* is contradictory to the rule *2)*. Sometimes, given the same set of symptoms, the expert might wish to conclude different conclusions with different certainty factors.

Incompleteness

A knowledge base is incomplete when it does not have all the information necessary to answer a question of interest to the systems. There are three main causes for incompleteness in a knowledge base.

First, during the knowledge acquisition process, both the expert and the knowledge engineer may have inadvertently left gaps in the knowledge base without noticing. The second cause is when the expert's behavior, which is often based on heuristic, incomplete, and uncertain knowledge, is carried into the system. Finally, the knowledge engineer may lose track as the knowledge base grows larger and becomes intractable.

Unreachable conclusion rules : In a goal driven production systems, the conclusion of a rule should match either a goal or an IF condition of another rule. If there are no matches, it is unreachable. Consider the following rule.

IF [a(x) and a(y) THEN c(z) (0.8).

If the *c(z)* is not the goal of knowledge base and it does not appear as a condition in any other rule,

then it is called an unreachable conclusion rule. This type of rules also suggests that there may be other rules or facts missing.

Unreferenced condition rules: If one of rule conditions does not appear as the conclusion of another rule or as a known fact, it is an unreferenced condition. Consider the following rule :

IF [a(x) and a(y) THEN c(z) (0.8).

If *a(x)* or *a(y)* does not appear as the conclusion of another rule and is not substantiated by fact, then it is an unreferenced condition rule. Rules with an unreferenced condition may indicate the possibility of some other rules or facts missing.

Isolated rules: If all of conditions are unreferenced and the conclusion is an unreachable conclusion, then the rule is called as an isolated rule. The presence of an isolated rule may indicate the possibility of missing rules.

Missing rules: Te knowledge base has deficiency when the rule does not produce any output. These rules can be indicated by unreachable conclusion rules, unreferenced condition rules, and isolated rules.

In developing knowledge based systems, most of knowledge engineers may adopt certainty factors in the knowledge base so as to improve the flexibility of reasoning. The presence of certainty factors, however, further complicates the verification process of a knowledge base. Allowing rules for conclusion with less than threshold and allowing conditions to be chained with certainty factors affect our definitions as shown in the following paragraphs :

In ***redundancy***, rules that are redundant can lead to problems. They might cause the same information to be counted twice leading to incorrect increases in the weight of their conclusions.

The ***subsumption*** appears quite often in rule sets because knowledge engineers frequently write rules with incorrect knowledge so that the weights of the more restrictive rules are added by the less restrictive rules.

The ***conflict***, when two rules execute at the same condition but have different conclusions, is the common occurrence in rule sets using certainty factors. Often, given the same set of symptoms, the expert might wish to conclude different values with different certainty factors. In this case, knowledge based systems have a serious inconsistency in reasoning process.

The checking of ***circular*** rule chains is not affected by certainty factors. However, it should be noted that certainty factors might cause a circular chain of rules to be broken if the certainty factor of a conclusion falls below the threshold.

Most of systems allow the user to specify thresholds. Thus, finding ***unreachable conclusions*** In a rule set with certainty factors also becomes complex. A conclusion in a rule could be unreachable though its IF part matches a conclusion in a different rule. This situation might occur if the comclusion that matches one of the IF conditions cannot have the certainty factor above the threshold.

Dead-end goal could occur if there is THEN clause that concludes with a certainty factor less than the threshold or is a chain of rules that produces a combined certainty factor less than the threshold. If this clause with a certainty factor below threshold is an IF condition rather than a goal, than it is would be a ***dead-end condition*** if there were no other lines of reasoning to determine it.

4. Development of verification Tool(COKEP)

4.1. Functional Structure of COKEP

COKEP is designed to detect anomalies in local and chained rule level, and certainty factor errors, in knowledge base of expert systems. In this work, COKEP provides an alternative strategy, transforming the problem of verification into that of reachability of specific states in the net. As the detection of anomalies is based on the results of firing transition, verification problems can be expressed as reachability problems. In order to solve these problems, in COKEP, matrix analysis of the extended Petri net and bac-

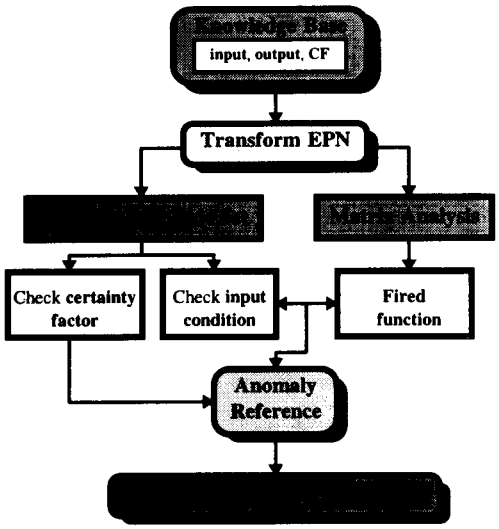

Fig. 1. The Functional Structure of COKEP

kward reasoning of the rule set are employed.

The functional structure of COKEP is shown in Fig. 1. As shown in Fig. 1, function of COKEP is divided into tow parts, finding chained rule set and performing matrix analysis. After finding chained rule sets, COKEP checks input condition of chained rule sets and certainty factor value from them. That is, COKEP performs the inference backwards so as to determine the allowable values for required input; this process is reverse to the developers' understanding of the system. COKEP identifies certainty factor errors, if exist in the representation of the certainty factor. The checked input conditions and firing vector by matrix analysis of extended Petri net are compared with the defined reference anomalies. COKEP performs checking eight anomalies with three parts, incorrectness, inconsistency, and incompleteness. Certainty factor error is identified by a threshold and reference anomalies. Verification result contains checked anomaly and certainty factor error, in local and chained rule level.

4.2. EPN Transform of Knowledge Base

An anomaly detection approach is based on a matrix view of EPN. The used program of COKEP is MS-FORTRAN code of 1000 lines to calculate the several matrix. Three matrices $D^-_1 + D^-_2$, and D^+ to represent the input and output functions can be defined from the (P, P', T, I_1, I_2, O) definitions of extended Petri nets. Each matrix has m rows for each transition and n columns for each place. $e[j]$ means the unit m-vector that is zero everywhere except in the jth component. The transition t_j is represented by the unit m-vector $e[j]$.

A transition t_j is enabled in a symbol μ if $\mu \geq e[j] \cdot (D^-_1 + D^+_2)$. The result vector, defined as $\delta(\mu, t_j)$, of firing transition t_j in a marking μ is

$$\begin{aligned}\delta(\mu, t_j) &= \mu - [e[j] \cdot (D^-_1 + D^-_2)] + [e[j] \cdot D^+] \\ &= \mu + e[j] \cdot [D^+ - (D^-_1 + D^-_2)] \\ &= \mu + e[j] \cdot D, \end{aligned} \quad (1)$$

where,

$D = D^+ - (D^- + D^{-\prime})$, $D^-_1[j, i] = \#(p_i, I_1(t_j))$, $D^-_2[j, i] = \#(p_i, I_2(t_j))$, and $D^+[j, i] = \#(p_i, O(t_j))$. Now for a sequence of transition firings $\sigma = t_{j1}, t_{j2}, \cdots, t_{jk}$,

$$\begin{aligned}\delta(\mu, \sigma) &= \delta(\mu, t_{j1}, t_{j2}, \cdots, t_{jk}) \\ &= \mu + (e[j_1] \cdot D) + (e[j_2] \cdot D) + \cdots + (e[j_k] \cdot D) \\ &= \mu + (e[j_1] + e[j_2] + \cdots + e[j_k]) \cdot D \\ &= \mu + f(\sigma) \cdot D. \end{aligned} \quad (2)$$

The vector $f(\sigma) = e[j_1] + [j_2] + \cdots + e[j_k]$ is calles the ***firing vector*** of the sequence $t_{j1}, t_{j2} \cdots, t_{jk}$.

The result vector of firing transition, t_i in marking μ is divided in three types. Each definition is

$\delta(\mu, \sigma) = 0$ (incompleteness)

$\delta(\mu, \sigma) = 1$ (inconsistency)

$\delta(\mu, \sigma) \geq 2$ (incorrectness).

The matrix analysis has some problems in checking anomalies. The result vector of firing transition t_j in marking μ is a necessary but not sufficient condition for reachability analysis in chained rule set A backward reasoning in the rule set is used for solving this problem. First, the result vector of firing transition is obtained by the matrix analysis, eq.(1) and (2). Then, we find chained rule path using matrix D^-_1 and backward reasoning. The conditions for

chained rule transition can be acquired by matrix $D^{-}{}_{2}$ which has the information of initial marking places.

4.3. Certainty Factor Checking

The certainty factor was used in many ESs because it has simplicity and wide availability. There are many formalisms that have been proposed to represent and propagate certainty factor in an ES. Generally, the possible rule combinations are classified into three scenarios and CF combination methods for each scenario are shown as follows[9].

a) Conjunction or disjunction of rule

CF(A and B) = min{CF(A), CF(B)}

CF(A or B) = max{CF(A), CF(B)}

b) Chained rule

$CF(A \rightarrow B) = CF(A) \times CF(B)$

c) Multiple combination of rule

IF $CF_1 \geq 0.1$ and $CF_2 \geq 0.3$,

$$CF(C) = CF_1 + (CF_2 - 0.1) - CF_1 \times CF_2$$

IF $CF_1 \leq 0.1$ and $CF_2 \leq 0.3$,

$$CF(C) = \max\{CF_1, CF_2\}$$

IF $CF_1 < 0.1$ and $CF_2 > 0.3$ or $CF_1 > 0.1$ and $CF_2 < 0.3$,

$$CF(C) = \frac{\max(CF_1, CF_2) - \min(CF_1, CF_2)}{1 - \min(CF_1, CF_2)}$$

IF $CF_1 = CF_2 = 0.2$, $CF(C) = 0.2$,

This CF combination method, slightly modified, is used for checking CF error. COKEP use the chained rule method. After finding chained rule sets, COKEP checks input condition of chained rule sets and certainty factor value from them, using above second method. That is, COKEP performs the inference backwards so as to check the combined certainty values.

4.4. Application to a Simple Example

In order to demonstrate the utility of COKEP we devise a simple example. Consider the simple level control system(see Fig. 2). In this system we may consider five fault sites; pump, drain valve V3, vessel level, level sensor L2, controller LC1 or actuatro CV1. When Knowledge engineer and ES developer build the knowledge for solving the problem, a fault in the level control system, they may propose the sixteen made-up rules in Table 1.

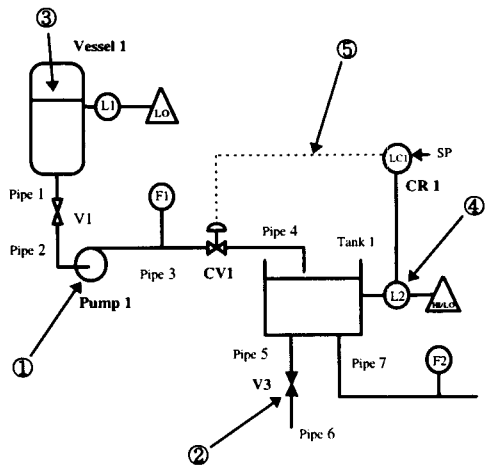

Fig. 2. A Simple Level Control System

In their present form they have anomalies in logical aspect of knowledge base, but these anomalies cannot be easily checked by knowledge engineer or system developer. We apply the COKEP in this knowledge base, and results are shown in Tables 3 and 4. anomaly of knowledge since COKEP checks incompleteness rule, not considering other anomalies at first stage. The rule 6 in Table 2 is generated in order to correct the incompleteness error of Table 4. After correcting incompleteness, COKEP preforms checking the incorrectness, inconsistency, and certainty factor error, shown in Table 4. Certainty factor threshold value used in this case is 0.5. This value may be changed by the requirement of knowledge engineer.

The Perti net of the knowledge base in Table 1 is shown in Fig. 3. In Fig.3, the places which have tokens are the initial known places related to input function I_2. COKEP makes the matrix automatically and calculates to obtain the next state functions by using the knowledge in Table 1 and equation(2). After matrix calculation, COKEP classifies the anomaly candid-

Table 1. Knowledge Base Example

		Comtent	CF
1	*if*	CV1 open and F1>100gpm	(0.7)
	then	pump on	
2	*if*	CV1 open and F1>100gpm	(0.8)
	then	sink environment abnormal	
3	*if*	CV1 open and F1>100gpm	(0.8)
	then	sensor abnormal	
4	*if*	CV1 open and F1>100gpm	(0.8)
	then	actuator or controller abnormal	
5	*if*	CV1 open and F1>100gpm	(0.8)
	then	pump off	
6	*if*	CV1 close and F1>100gpm	(0.8)
	then	sensor abnormal	
7	*if*	CV1 close and F1>100gpm	(0.7)
	then	actuator or controller abnormal	
8	*if*	pump on and L2LO	(0.8)
	then	sink environment abnormal	
9	*if*	pump on and L2HI	(0.7)
	then	actuator or controller abnormal	
10	*if*	pump off and F2<80 gpm and L2LO	(0.7)
	then	pump 1 fail	
11	*if*	sink environment abnormal and F2<80 gpm	(0.8)
	then	drain valve open	
12	*if*	sink environment abnormal and F2<80 gpm	(0.9)
	then	CV1 or LC fail	
13	*if*	source environment abnormal and F2>80gpm and L2LO	(0.8)
	then	vessel level low	
14	*if*	sensor abnormal and F2>80gpm	(0.8)
	then	L2 device fail	
15	*if*	sensor abnormal and F2<80gpm and L2I	(0.8)
	then	L2 device fail	
16	*if*	actuator or controller abnormal and F2>80gpm and L2HO	(0.9)
	then	CV1 or LC fail	

Table 2. Knowledge Base Example(Checked Knowledge Base)

		Comtent	CF
1	*if*	CV1 open and F1>100gpm	(0.7)
	then	pump on	
2	*if*	CV1 open and F1>100gpm	(0.8)
	then	sink environment abnormal	
3	*if*	CV1 open and F1>100gpm	(0.8)
	then	sensor abnormal	
4	*if*	CV1 open and F1>100gpm	(0.8)
	then	actuator or controller abnormal	
5	*if*	CV1 open and F1<100gpm	(0.9)
	then	pump off	
6	*if*	CV1 open and F1<100gpm	(0.9)
	then	source environment abnormal	
7	*if*	CV1 close and F1<100gpm	(0.8)
	then	sensor abnormal	
8	*if*	CV1 close and F1<100gpm	(0.7)
	then	actuator or controller abnormal	
9	*if*	pump on and L2LO	(0.8)
	then	sink environment abnormal	
10	*if*	pump on and L2HI	(0.7)
	then	actuator or controller abnormal	
11	*if*	pump off and F2<80 gpm and L2LO	(0.7)
	then	pump 1 fail	
12	*if*	sink environment abnormal and F2<80 gpm	(0.8)
	then	drain valve open	
13	*if*	sink environment abnormal and F2>80gpm	(0.9)
	then	CV1 or LC fail	
14	*if*	source environment abnormal and F2<80gpm and L2LO	(0.8)
	then	vessel level low	
15	*if*	sensor abnormal and F2>80gpm and L2LO	(0.8)
	then	L2 device fail	
16	*if*	sensor abnormal and F2<80gpm and L2HI	(0.8)
	then	L2 device fail	
17	*if*	actuator or controller abnormal and F2<80gpm and L2HI	(0.9)
	then	CV1 or LC fail	

Table 3. Checking Result I.(Incompleteness Anomaly)

rule # = 13	Unreferenced condition Sre_ab
rule # = 16	Unreferenced condition L2HO

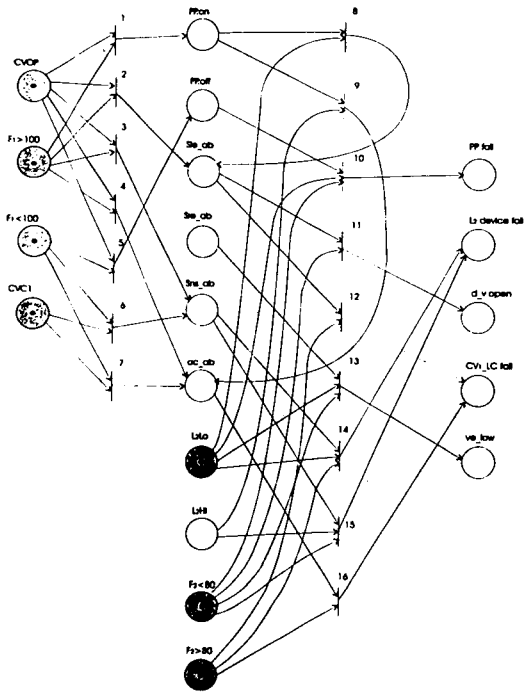

Fig. 3. Petri net of Knowledge Base in Table 1

Table 4. Checking Result II.(Incorrect, Inconsistency, CF Anomaly)

	Chained rule #	Goal	Condition
Incorrect rule	4,17	CV1_LC	L2HI,CV1_op, F1>100,F2>80
(redundant)	1,10,17	CV1_LC	L2HI,CV1_op, F1>100,F2>80
Inconsistent rule	3,15	L2_fal	L2LO,CV1_op, F1>100,F2>80
(conflict)	1,9,13	CV1_LC	L2LO,CV1_op, F1>100,F2>80
CF error	1,9,12	d_v_op	CF = 0.45
(threshold = 0.5)	1,10,17	CV1_LC	CF = 0.44

ates according to the number of markings in the places. For example, the goal places which are candidates for incorrect anomalies have two tokens. The number of paths connected to the places is 24. COKEP checks the initial known factors and goals of the paths. If reasoning paths contain the same set of conditions(but conditions amy be arranged in a different order) and have the same conclusion, then one of them is said to be redundant, The results of path checking are shown as the incorrect rule in Table 4.

First result in Table 3 shows the incompleteness The checked knowledge base is shown in Table 2.

An alternative strategy is provided in this work which transforms the problem of verification into that of reachability of specific states in the net. As the detection of anomalies is based on the results of firing transition, the verification problems can be expressed as the reachability problems.

5. Conclusion

Verification and validation are vital to the success of ES(Expert System). Sufficient guidelines on ES V&V(Verification & Validation), however, have not been formed despite of many approaches and methods. The verification has, until now, focused upon building novel anomaly detection systems and improving the efficiency of existion systems. The issues of theoretical foundations of knowledge base verification, however, have remained unaddressed.

In this aspect, the work presented here provides a reliable and uniform verification method that adopts improved verification techniques and an automated integral verification tool. COKEP tool is based on modeling a knowledge base by using the extended Petri net, and uses matrix analysis and backward reasoning in verification process. The scope of the verification in COKEP is expanded to chained errors unlike previous works that assume error incidence to be limited to rule pairs only. In addition, COKEP tool also checks certainty factors which are included in most expert system.

The application results of KB(Knowledge Base) V&V(Verification & Validation) with the use of the COKEP tool and Petri nets show that the logical checking reveals anomalies successfully in a knowledge base. The proposed method in this work, however, has some limitation for KB V&V. First, COKEP tool detects only the previously defined anomalies. The use of other detailed anomaly references will ofter some promises for more accurate verification. Second, the knowledge bases built in other forms such as frames and predicates are transformed to rule forms in order to apply Petri net. Third, the proposed method for KB verification was applied to a simple example, not to real ES knowledge base. Therefore, in order to show the full capability of the proposed method, an application to rather complex knowledge base and the analysis of the results are needed.

References

1. B.J Cragun and H.J. Studel, "A decision-table-based processor for checking completeness and consistency in rule-based expert system," Int. J. Man-Machine Studies, vol. 26, pp. 633-648(1987).
2. M.R. Rousset, "On the consistency of knowledge bases:COVADIS system," Proc. 8th Eur. Conf. AI, Paris, pp. 79-84(1988)
3. C.L.Chang, J.B.Combs, and R.A.Stachowitz, "A report on the expert systems validation associate (EVA)," Expert Systems with Applications, Vol. 1, pp. 217-230(1990)
4. P.Meseguer, "A new method to checking rule bases for inconsistency: A Petri net approach," Proc. 9th Eur. Conf. AI, Germany, pp. 437-442(1990).
5. N.K.Liu and T.Dillon, "An approach towards the verification of expert systems using numerical petri nets." Int. J.Intell. Sys., vol. 6, pp. 255-276,(1991).
6. D.Zhang and D.Nguyen, "A tool for knowledge base verification," IEEE Transaction of Knowledge and Data Engineering, vol. 6, no. 6, pp. 983-989, Dec.(1944)
7. J.L.Peterson, "Petri net theory and the modeling of systems," Prentice-Hall, Inc., Englewood Cliffs, N.J.(1981)
8. Derek L. Nazareth, "Investigating the Applicability of Petri Nets for Rule-Based System Verification," IEEE Transaction on Knowledge and Data Engineering, vol. 4, no. 3, pp. 402-415, June(1993)