1997 First International Conference on Knowledge-Based Intelligent Electronic Systems, 21-23 May 1997, Adelaide, Australia, Editor, L.C. Jain

Formal Verification of the Correctness in Hybrid Expert Systems

Simon C.K. Shiu
James N.K. Liu
Daniel S. Yeung

Department of Computing
Hong Kong Polytechnic University
Hung Hom, Kowloon
Hong Kong
E-mail: {csckshiu | csnkliu | csdaniel}@comp.polyu.edu.hk

Keywords: Formal Verification, Hybrid Expert Systems

Abstract

It has been increasingly recognized over recent years that expert systems which combine one or more techniques greatly increase the problem solving capability and help overcome some of the shortcomings associated with any single technique. The verification of these expert systems requires methods which could tackle the multiple knowledge representation paradigms and integrated inference mechanisms used. This paper provides a formal description technique for verifying the correctness of Hybrid Expert Systems (HES) that emphasizes an integration of object hierarchy, property inheritance and production rules. The main idea is to convert the HES into a State Controlled Coloured Petri Net (SCCPN) where the object hierarchy, property inheritance and production rules are modelled as separated components in the same SCCPN. The detection and analysis of the anomalies in the system are done by constructing and examining the reachability tree spanned by the knowledge inference. This provides a formal basis for automating the deduction process and a means of verifying HES. A set of propositions is formulated to verify errors and anomalies in HES. Lastly, future extension of our approach is discussed.

1. Introduction

Traditionally, attention has been concentrated on using verification techniques to tackle rule-based systems [8,9,13,14,15]. However, these techniques exhibit a limited range of applicability. They could not cope with the kind of hybrid expert systems (HES), e.g. rule-based plus frame-based, which many of the current expert systems are being developed [2,5,17,23]. The use of this hybrid approach integrates the power of organizing data objects in a class hierarchy and reasoning about the objects through user pre-defined logical associations. This advantage accounts for many popular expert system development software (or shells), such as ADS, ART, EXSYS EL, KAPPA-PC, KBMS, NEXPERT OBJECT, LEVEL5 OBJECT, PRO-KAPPA, REMIND, which combine some sort of frame-based representation with a rule-based inference engine.

Recently, [19,20] have shown that HES can be modelled and analyzed by SCCPN. As demonstrated, the object class's data structure is represented by a high ordered colour set, and each object instance is represented by a token in that set. The production rules and property inheritance are both represented by SCCPN transitions. Thus, the relationship and semantic information among these rules and the object hierarchy can be represented explicitly in these SCCPNs. Consequently, by firing of the enabled transitions, we have been able to dynamically simulate the propagation of rule inference and property inheritance in the HES. We have also identified some defined anomalies through the analysis of the reachability tree generated by a sequence of transition firings. In other words, if we use different object instances as inputs to the HES, a

set of rules will be triggered to fire and the result can be obtained by viewing the sequence of transition firings in the SCCPN. This result is formed by chaining the rules and object hierarchy represented by SCCPN together according to the transformation given in [19,20].

In order to allow for the automation of the verification process, to tackle the mathematical problems associated with the nets, and to provide accurate detection of anomalies in the HES, a more formal definition and discussion of the model are necessary. It is noted that there are very few other approaches based on Petri net theory in literature to model or verify expert systems. The typical ones might be [1,13,14,16,18,24]. However, none of these approaches use the sort of Coloured Petri Nets [10,11] that are used in our approach to resolve some verification issues and problems as highlighted in [20,21]. Apart from lacking Petri net-based formal theories for verification, there is not much attention paid on hybrid expert systems except [7,12]. [12]'s work focuses on the post-verification of hybrid systems. They detail the subsumption anomalies between rules that use Parent Class information and those rules that use Child Class information. [12]'s definition of subsumption anomalies in hybrid expert system is very useful for conceptual understanding. However, they do not provide a general framework to model other essential properties of HES such as the integration of rules with inheritances, rules with methods and rules with demons. Besides, their approach is only confined to static checking of the semantic structures in the HES, which is not possible to be extended to cover dynamic analysis.

[7]'s work focuses on the partial HES requirement specification using a hybrid language combined from Z and SWARM. It should be emphasized that our model differs in other respects. For instance, our model can:

- provide a graphical representation of the relationships among the object hierarchy, object instances, methods, demons and the production rules.
- allow for the dynamic checking of HES which yields information on how the system achieves it goals.
- provide information about the current state of transition predicates as well as the states of the object instances while others hardly can.
- provide a clear semantics which allow for the formal analysis of the behaviour of the modelled HES.
- has the ability to maintain or update both the state of predicates and slot values of the object instances during transition firings. The important point due to this capability is that our model will thus have a potential to tackle situations with relatively higher complexity and variant conditions like temporal space, probabilistic and fuzzy reasoning.

In this article, we will examine the sequence of transitions and check against the properties of the network in SCCPN. The article is organized into six main sections. The first section gives the introduction and motivation of our work. The second section gives the fundamental principle, definitions, and properties of HES and SCCPN. Problem description and formulation of the anomalies in a HES is provided in the third section. Some basic description and properties of our formal approach are described in the fourth section. Formal verification of the correctness, consistency, and completeness problems will be discussed in the fifth section. The corresponding proofs of some of these formal propositions are given in the Appendix. The application of our formal approach to a practical hybrid expert system for personnel selection is described in section six. Finally, the article concludes with a discussion of the future extension of our proposed methodology.

2. Fundamental Principle

A Hybrid Expert System combines multiple representation paradigms into a single integrated environment for modelling and reasoning of complicated real world phenomena. For a Rule- and Frame-based integration, it models the problem domain using the concepts of classes and rules together. The essential key modelling features are: Object Classes, Slot Attributes, Inheritance Relations, Demons, Methods, Rules and Reasoning Strategies. These features can be analyzed using three conceptual views [6] of an expert system, they are: (1) An Object View which encapsulates a module of knowledge (or a concept). These knowledge modules (concepts) are represented by Object Classes. Inheritance Relations describe how these knowledge modules are related. (2) A Function View which specifies the functional behaviour of the objects. These functions are represented using Methods and Demons. (3) A Control View which specifies the knowledge inference in the expert system. These controls are represented in terms of Rules and Reasoning Strategies.

In practical HES development [19,20,21,22]. Frames are used to represent domain objects, various kinds

of Demons are used to implement procedures attached to specific slots, Inheritance is used to inherit Class properties, methods and demons among Object Classes, Message Passing is used for interaction among different objects and Methods are used to perform algorithmic actions or some array manipulation within an object. Rules are used to describe heuristic problem-solving knowledge, Forward and Backward chains are commonly used to reason using rules. Therefore, in HES, the Frame base can be seen as the one used to define the vocabulary for the Rule base, i.e. the possible values that slots can be defined and so specified, and the literal used to construct rules must conform to the restrictions imposed by what is available from the class hierarchy. The Frame base is married together with the Rules designed to manipulate it. The specific integration mechanisms of HES are as follows:

- Rules with Message Passing : Rules send or receive messages to and from objects for testing the Rules' premises.

- Rules with Inheritance : Rules directly read and write data into slots in a parent object and through inheritance of the slot's value to its children objects, trigger other rules to fire.

- Rules with Demons : Rules directly read and write data into slots and cause the execution of the associated Demons, which then trigger other rules to fire.

- Rules with Methods : Rules are embedded as part of an object's methods. Since methods are arbitrary pieces of code attached to an object, they can access the rules through function calls.

- Rules with Instances : Rules can be used to create/delete an instance of a specific Object Class.

Based on the above concepts of integration, a Hybrid Expert System, therefore, can be formally defined as follows.

DEFINITION 2.1. A HES is defined as a tuple given by: HES = (C, A, D, M, I, H, R, S) satisfying the requirements below:

C = a finite set of object classes, where each object class is a Cartesian product of (A x D x M).
A = a finite set of attributes. Each attribute is of a simple data type.
D = a finite set of demon functions. Each function is defined from A into an expression such that: $\forall a \in A: D(a) \in A$. (This means the demon functions can only change a slot's value within the same object instance. Besides, this demon function: D(a) generates only one output from each given input "a").
M = a finite set of methods. Each method is defined as a function which takes a number of arguments from an object$\in$C and returns a result to the object$\in$C.
I = a specific object element from an object class C.
H = an inheritance relation. It is defined from the partially ordered relations in C.
R = The rules are composed of predicates which are used as functions that map object arguments into TRUE, FALSE values represented by binary truth values 1,0, respectively. (One of the predicates is the IS-A predicate which is used to specify the class of objects which a particular rule can be applied). All literals used in both the condition and action predicates must come from the attribute set A.
S = a finite set of reasoning strategies. The two common HES reasoning strategies are: Backward Chain with Inheritance and Forward Chain with Inheritance.

Explanations: Object class here is defined as having a set of attributes, demons and methods. Each attribute is defined as of a simple data type: e.g. string, integer or real. Each specific object element is called an instance of the Object Class and will have different attribute values of the variables. Inheritance is defined as a partial order on the set Object Class, it is a relation that is reflexive, antisymmetric and transitive:

- Reflexive : For every Object Class, it inherits the properties from itself.

- Antisymmetric : For every Object Class, if A inherits from B and if B inherits from A, it implies that A is B.

- Transitive : For every Object Class, if A inherits from B and if B inherits from C, it implies that A inherits from C.

The above definition only covers simple inheritance. In the case of multiple inheritance, the problem becomes what characteristics the child inherits, and from which parent? The HES has to follow some sort of default orderings on inheritance [4,24], and this may lead to sets of conflicting traits which are even more complicated to verify. Therefore, our present analysis is concentrated on simple inheritance only.

A Demon is defined as a function which is executed when the associated slot value is either updated, or

needed. Sometimes, a Demon can also act like a validation trigger which checks the cardinality and/or constraints imposed on a particular slot. The effects of a Demon are confined always locally to the same Object Class.

Methods are functions attached to some Object Class, that will be executed whenever a signal is passed through. Each method is defined as a function which takes a number of arguments and return a result.

Rules will interact with the information contained in the slots of the various Object Classes within the HES.

Finally, in HES, there should be a set of reasoning strategies. The two common ones are :

- Backward Chain with Inheritance : Goal directed search with inheritance as one of the means to establish the rule chains across different Object Classes.

- Forward Chain with Inheritance : Data directed search with inheritance as one of the means to establish the rule chains across different Object Classes.

As HES is modelled by SCCPN, a mapping between the two structures is necessary, and is given in Table 1.

Hybrid Expert System	State Controlled Coloured Petri Net
Frame-based part	
Object Classes	Places
Object Class Types	Colour Sets
Object Instances	Tokens
Slots	Variables in Tokens
Facts in Slots	Binding of Variables with Constants
Inheritances	Transitions
Demon	Arc Expressions
Methods	Arc Expressions
Rule-based part	
Predicates	Places
Predicates States	Tokens
Rules	Transitions
Facts	Binding of Variables with Constants
Transition Operations	Arc Expressions

Table 1. Conceptual interpretation of HES in SCCPNs.

As shown in Table 1 the components of the HES are separately represented, which can be modelled explicitly by the SCCPN. The places are taken to correspond to predicates and object classes, and transitions to represent rules implications as well as inheritance. There are two major types of tokens, one is the state token which records the state of the predicate and the class type information. (i.e. Since rules may be fired by either parent class instance or child class instances). The second type of token is the object instance token which represents a particular object instance of a particular class within the object hierarchy. Transitions are fired to represent rules being executed or inheritance is being carried out. The maximum number a rule can be executed is equal to the total number of different class types. (i.e. each class type object instance can fire a particular rule once at most). Each input place of a rule has a self-loop arc for maintaining the state of the predicate. Similarly, the input place of an inheritance also has a self-loop arc for recording the inheritance execution. Methods and Demons are represented by functions in the arc inscription of the SCCPN. The net result is the exchange of colour tokens from places to places and a new marking, which is defined as the distribution of tokens over the places of the SCCPN, is obtained.

The SCCPN notation employed in this paper is an extension of State Controlled Petri Nets proposed by [13, 14], and Coloured Petri Nets proposed by [10,11] and is specified as follows.

DEFINITION 2.2. A SCCPN can be defined as a 10-tuple given by = (Σ, *P, T, D, F, A, N, C, E, I*), where satisfying the requirements below:

$\Sigma = \{ \omega_1, \omega_2, ..., \omega_i \}$, a finite set of non-empty types, called *colour sets*, $i \geq 1$,
$P = \{P_c, P_r\}$ a finite set of *places*,

$P_c = \{ p_{c1}, p_{c2}, ..., p_{cj} \}$, a finite set of places that model the classes of the HES, called *class places*, $j \geq 1$,

$P_r = \{ p_{r1}, p_{r2}, ..., p_{rk} \}$, a finite set of places that model the predicates of the production rules, called *predicate places*, $k \geq 1$,

$P_c \cap P_r$: the intersection of $P_c \cap P_r$ represents those IS-A *predicates* of the rule sets attached to the specific *classes*,

$T = \{ T_c, T_r \}$, a finite set of *transitions*,

$T_c = \{ t_{c1}, t_{c2}, ..., t_{cl} \}$, a finite set of transitions that are connected to and from *class places*, called *inheritance transition*, $l \geq 1$,

$T_r = \{ t_{r1}, t_{r2}, ..., t_{rm} \}$, a finite set of transitions that are connected to or from *predicate places*, called *predicate transition*, $m \geq 1$,

$T_c \cap T_r = \varnothing$,

$D = \{ d_1, d_2, ..., d_n \}$, a finite set of *predicates*, $|P_r| = |D|$, $n \geq 1$,

$F = \{ f_1, f_2, ..., f_n \}$, a finite set of *classes*, $|P_c| = |F|$, $n \geq 1$,

$A = \{ a_1, a_2, ..., a_k \}$, a finite set of *arcs*, $k \geq 1$, $P \cap T = P \cap A = T \cap A = \varnothing$,

$N : A \rightarrow P \times T \cup T \times P$, a *node function*, it maps each arc into a pair where the first element is the source node and the second is the destination node, the two nodes have to be of different kinds. The node functions can be further classified into the following eight different types:

Inheritance : $\{ \tilde{A}_c, \ddot{A}_c, \tilde{A}_s, \ddot{A}_s \}$ where

$\tilde{A}_c : T_c \rightarrow (P_c)_{MS}$ is an input class function for inheritance, a mapping from inheritance transitions to the bags of class places. *MS* stands for multi-set (or bags).

$\ddot{A}_c : T_c \rightarrow (P_c)_{MS}$ is an output class function for inheritance, a mapping from inheritance transitions to the bags of class places.

$\tilde{A}_s : T_c \rightarrow (P_c)_{MS}$ is an input state function for inheritance, a mapping from inheritance transitions to the bags of class places.

$\ddot{A}_s : T_c \rightarrow (P_c)_{MS}$ is an output state function for inheritance, a mapping from inheritance transitions to the bags of class places.

Predicate : $\{ \tilde{O}_c, \ddot{O}_c, \tilde{O}_s, \ddot{O}_s \}$ where

$\tilde{O}_c : T_r \rightarrow (P_r)_{MS}$ is an input class function for predicates, a mapping from predicates transitions to the bags of predicates.

$\ddot{O}_c : T_r \rightarrow (P_r)_{MS}$ is an output class function for predicates, a mapping from predicates transitions to the bags of predicates.

$\tilde{O}_s : T_r \rightarrow (P_r)_{MS}$ is an input state function for predicates, a mapping from predicates transitions to the bags of predicates.

$\ddot{O}_s : T_r \rightarrow (P_r)_{MS}$ is an output state function for predicates, a mapping from predicates transitions to the bags of predicates.

$C : P \rightarrow \Sigma$, a *colour function*, it maps each place into a colour set,

$E : A \rightarrow$ *expression,* an arc expression function, It is defined from A into expressions such that $\forall a \in A :$ $[Type(E(a)) = C(p(a))_{MS} \wedge Type(Var(E(a))) \subseteq \Sigma]$ where $p(a)$ is the place of $N(a)$, where *MS* stands for multi-set (or bags),

$I : P \rightarrow$ expression, an initialization function. It is defined from P into closed expressions such that: $\forall p \in P : [Type(I(p)) = C(p)_{MS}]$.

DEFINITION 2.3. For each transition $t_j \in T$ in a net N,

$$\tilde{O}_s(t_j) \cap \ddot{O}_s(t_j) \neq \varnothing,$$
$$\tilde{O}_c(t_j) \cap \ddot{O}_c(t_j) = \varnothing,$$
$$\tilde{A}_c(t_j) \cap \ddot{A}_c(t_j) \neq \varnothing,$$
$$\tilde{A}_s(t_j) \cap \ddot{A}_s(t_j) = \varnothing,$$

such that

$$p_i \in \tilde{O}_s(t_j) \Rightarrow p_i \in \ddot{O}_s(t_j),$$
$$p_i \in \tilde{O}_c(t_j) \Rightarrow p_i \notin \ddot{O}_c(t_j),$$
$$p_i \in \tilde{A}_c(t_j) \Rightarrow p_i \in \ddot{A}_c(t_j),$$
$$p_i \in \tilde{A}_s(t_j) \Rightarrow p_i \notin \ddot{A}_s(t_j),$$

DEFINITION 2.4. A binding of a transition t is a function b defined on Var(t), such that: $\forall v \in Var(t) : b(v) \in Type(v)$ where Var(t) denotes the set of variables in a transition and B(t) denotes the set of all bindings for t.

DEFINITION 2.5. A token element is a pair (p,c) where $p \in P$ and $c \in C(p)$, while a binding element is a pair (t,b) where $t \in T$ and $b \in B(t)$. The set of all token elements is denoted by TE while the set of all binding elements is denoted by BE.

DEFINITION 2.6. A marking M is a multi-set over TE while a step is a non-empty and finite multi-set over BE. The initial marking M_0 is the marking which is obtained by evaluating the initialization expressions: $\forall (p,c) \in TE : M_0(p,c) = I(p)(c)$. The markings of a SCCPN can be further classified into the following two different types: $\{M_c, M_s\}$ where M_c represents markings of the class tokens, and M_s represents markings of the state tokens.

DEFINITION 2.7. A step Y is enabled in a marking M iff the following property is satisfied: $\forall p \in P : \sum_{(t,b) \in Y} E(p,t) < b > \leq M(p)$ where *E(p,t)* is the expression of (place, transition) and *E(t,p)* is the expression of (transition, place). The summation indicates the addition of expressions. Expression<b> denotes the binding of the specific expression with a set of constants b. When $(t,b) \in Y$, this denotes that t

is enabled in M for the binding b. When (t_1,b_1), $(t_2,b_2) \in Y$ and $(t_1,b_1) \neq (t_2,b_2)$, this denotes that (t_1,b_1) and (t_2,b_2) are concurrently enabled. (If E=1, we refer this specific step as inheritance step. (i.e. the "presence" of a token element will enable the step).

DEFINITION 2.8. When a step Y is enabled in a marking M_1 it may occur, changing the marking M_1 to another marking M_2, defined by:

$$\forall p \in P: M_2(p) = (M_1(p) - \sum_{(t,b)\in Y} E(p,t) < b >) + \sum_{(t,b)\in Y} E(t,p) < b > .$$

The first sum is the removed tokens while the second is the added tokens. M_2 is directly reachable from M_1 by the occurrence of the step Y, which can be denoted as $M_1[Y>M_2$.

DEFINITION 2.9. A finite occurrence sequence is a sequence of markings and steps:
$M_1[Y_1>M_2[Y_2>M_3\ldots\ldots M_n[Y_n>M_{n+1}$ such than n $\in$ Natural Number and $M_i[Y_i>M_{i+1}$ for all $i \in 1\ldots..n$. The marking M_1 is called the start marking of the occurrence sequence, while the marking M_{n+1} is called the end marking. The non-negative integer n denotes the number of steps in the occurrence sequence, or the length of it.

DEFINITION 2.10. A marking M" is reachable from a marking M' iff there exists a finite occurrence sequence having M' as start marking and M" as end marking, i.e. iff for some $n \in N$ there exists a sequence of steps $Y_1,Y_2\ldots..Y_n$ such that: $M_1[Y_1>M_2[Y_2>M_3\ldots\ldots Y_n>M"$. M" is reachable from M' in n steps. A firing or occurrence sequence is denoted by

$$\sigma=(Y_1,Y_2\ldots\ldots Y_n)$$

The set of markings which are reachable from M' is denoted by [M'>.

DEFINITION 2.11. The full occurrence graph of a SCCPN is the directed graph OG=(V, A, N) where:
1. $V=[M_0>$
2. $A=\{(M_1,b,M_2) \in VxBExV | M_1[b>M_2\}$.
3. $\forall a=(M_1,b,M_2) \in A: N(a)=(M_1,M_2)$.

In OG, a node is a particular marking reachable from M_0. (ie The construction of OG is using Markings as nodes while construction of SCCPN is using Place and Transitions as nodes) The set of markings which are reachable from M_0 is denoted by $[M_0>$. An arc a with $N(a)=(M_1,M_2)$ is said to go from the source node M_1 to the destination node M_2. An arc with the binding element b is denoted by (M_1,b,M_2).

The occurrence graph (O-graph) has a node for each reachable marking and an arc for each step that occurs - with a single binding element. The source node of the arc is the start marking of the step, while the destination node is the end marking.

3. Correctness of a HES

Although the integration of a Rule- and Frame-based Expert System can take the advantages of both representation paradigms. The systems are not free from errors and anomalies. In a pure rule-based system, errors and anomalies could include redundancy, dead-end rules, subsumption, duplication, circular rule sets, unsatisfiable conditions, missing rules..etc. Their verification are well documented in the literature [3,8,9,13,14,15]. In a pure frame-based system, errors and anomalies may occur due to the problems of message passing and concurrency, problems of inheritance (including simple, repeated and multiple inheritance) and problems of polymorphism. Instead of covering all the possible errors and anomalies caused by the integration of the above two representation paradigms, we would like to focus ourselves on the additional errors and anomalies attributed to the integration of rules with the inheritance of object properties.

Given that in a closed world situation in which a common concept is derived by a HES {C, A, I, H, D, M, R, S}. The anomalies that are relevant to the correctness of the HES, take the following forms:

3.1. Redundancy

Case I. Conditions and Actions identical between Parent Class and Child Classes.

In the case of rules which have identical conditions and actions applied to the parent object class and child object classes, this implies the existence of redundant rules.

Rule 1 : $A \wedge B \Rightarrow C$
Rule 2 : $A' \wedge B' \Rightarrow C'$

(A, B & C are slots in the parent object, A', B' and C' are slots in the child object and A'=A, B'=B, C'=C because of inheritance).

Case II. Chained inference

Rule 3 : $A \Rightarrow C$
Rule 4 : $A' \Rightarrow B'$
Rule 5 : $B' \Rightarrow C'$

In the case of a chained inference, some rules could become redundant if the same result could be inferred by alternative transitions even the same input facts are given. (A'=A and C'=C because of inheritance and B' is not ascertainable through other rules). Rule 3 could become redundant as C' could be inferred by an alternative transition, Rule 5, via Rule 4.

3.2. Subsumption

Case I. Conditions subsumed with identical actions between Parent Class and Child Classes.

Rule 6 : $A \wedge B \Rightarrow C \wedge D$
Rule 7 : $A' \Rightarrow C' \wedge D'$

Case II. Conditions identical with subsumed actions between Parent Class and Child Classes.

Rule 8 : $A \wedge B \Rightarrow C \wedge D$
Rule 9 : $A' \wedge B' \Rightarrow C'$

Case III. Conditions and actions subsumed between Parent Class and Child Classes.

Rule 10 : $A \wedge B \Rightarrow C \wedge D$
Rule 11 : $A' \Rightarrow C'$

In a complex frame hierarchy which allows for multiple inheritance, checking for subsumption becomes more difficult because the problem becomes what characteristics the child inherits, and from which parent? The HES has to follow some sort of default orderings in inheritance, and this may lead to sets of conflicting traits which are even more complicated to verify.

3.3. Ambiguity

Case I. Rule with inclusive disjunction of IS-A conditions from different Object Classes.

Rule 12 : A IS-A member of ClassX $\vee$
A IS-A member of ClassY $\Rightarrow$ B

Case II. Rule with inclusive disjunction of IS-A Actions for different Object Classes.

Rule 13 : B $\Rightarrow$ A IS-A member of ClassX $\vee$
A IS-A member of ClassY

3.4. Circular Rule Sets

If a circular loop can occur when a set of rules among different object classes are fired, then these rules are considered as a circular rule set within the object hierarchy.

Case I. Self-reference rule

Rule 14: $A' \Rightarrow A \wedge B$

Case II. Self-reference chain of inference

Rule 15: $A \Rightarrow B \Rightarrow \bullet\bullet\bullet\bullet\bullet\bullet \Rightarrow P$
Rule 16: $P' \Rightarrow A$

If more than one level of class hierarchy is involved, an implicit cycle may exist where the loop is formed from several rules and different frames' slots in the frame hierarchy.

4. Description and Properties

The logical predicate becomes true by the presence of a state token and the transition associated with this predicate will become active by the presence of the corresponding object class token (instance) and provided that the slots attributes in the object class instance satisfies the transition condition. The transition is enabled and is ready for firing. For simplicity reasons, without taking any transition conditions or transition operations into consideration, we can *minimally* enable a specific transition and then check the reachability set for any irregularities of predicate places. In this representation, a marking M is composed of M_c that depicts the marking for the class places and Ms that depicts the marking for the state places in the SCCPN. A transition t_j is represented by a t-vector. For verification purposes, we define that:

DEFINITION 4.1. A transition t_j is minimally active if

$$M_c = \begin{Bmatrix} 1 \ if \ p_{ci} \in (\tilde{A}_c(t_j) \cup \tilde{O}_c(t_j)) \\ 0 \ otherwise \end{Bmatrix}$$

DEFINITION 4.2. A transition t_j is minimally enabled if t_j is both minimally active and that

$$M_s = \begin{Bmatrix} 1 \ if \ p_{si} \in (\tilde{A}_s(t_j) \cup \tilde{O}_s(t_j)) \\ 0 \ otherwise \end{Bmatrix}$$

and

$$\sum E(p_{si}, t_j) < b >\leq (M_c(p_{si}) \cup M_s(p_{si}))$$

DEFINITION 4.3. T_k that contains a group of transitions $\{t_n\}$ is said to be minimally active if $\forall j=1,2,..n$, $t_j \in T_k$, $\exists\ p_i \in (\tilde{A}_c(t_j) \cup \tilde{O}_s(t_j)) \subseteq (\tilde{A}_c(T_k) \cup \tilde{O}_s(T_k))$, such that

$$M_c = \begin{cases} 1 \text{ if } p_{ci} \in (\tilde{A}_c(t_j) \cup \tilde{O}_s(t_j)) \\ \quad \text{and } p_{ci} \notin (\ddot{A}_c(t_j) \cup \ddot{O}_s(t_j)) \\ 0 \text{ otherwise} \end{cases}$$

Note that the self-loop arc corresponding to each input place does not cause a repeated firing of transitions. In the absence of any self-reference rule, the set of input places and that of output places with respect to the transition in SCCPN are always disjointed.

DEFINITION 4.4. T_k that contains a group of transitions $\{t_n\}$ is said to be minimally enabled if $\forall j=1,2,..n$, $t_j \in T_k$, $\exists$ $p_i \in (\tilde{A}_c(t_j) \cup \tilde{O}_s(t_j)) \subseteq (\tilde{A}_c(T_k) \cup \tilde{O}_s(T_k))$, such that

$$M_c = \begin{cases} 1 \text{ if } p_{ci} \in (\tilde{A}_c(t_j) \cup \tilde{O}_s(t_j)) \\ \quad \text{and } p_{ci} \notin (\ddot{A}_c(t_j) \cup \ddot{O}_s(t_j)) \\ 0 \text{ otherwise} \end{cases}$$

and

$$\sum E(p_{si}, t_j) < b >\leq (M_c(p_{si}) \cup M_s(p_{si}))$$

5. Formal Verification of the Correctness Problem

The problems of correctness about a rule set applied to an object hierarchy might involve redundancy, subsumption, ambiguity, and cyclicity. These are observable either between a pair of rules applied to an object hierarchy or rules that represent chains of inference in the object hierarchy.

Altogether, four propositions are defined for representing the formal properties in the SCCPN in which each of them corresponds to some anomalies in the HES.

5.1. Redundancy

Proposition 5.1. For a given marking M_0, that minimally enables a nontrivial transition sequence σ_i, iff the HES has incorrect rules causing redundancy between the parent and child object classes, then $\exists\sigma_j$, $\exists k$, such that these sequences have the following properties:

(i) $\sigma_i \cap \sigma_j = \varnothing$;
(ii) $T_c \cap \sigma_i = \varnothing; T_c \cap \sigma_j \neq \varnothing$;
(iii) $M' = \delta(M_0, \sigma_i)$, $M'' = \delta(M_0, \sigma_j)$;
(iv) $M_{sk}=0$, $M'_{sk}>0$, $M''_{sk}>0$;
(v) $M_{ck}=0$, $M'_{ck}>0$, $M''_{ck}>0$;
(vi) $\exists(p_{rk},c_{ck})' \in M'_{ck}$, $\exists(p_{rk},c_{ck})'' \in M_{ck}$"
(vii) $(p_{rk},c_{ck})'' = (p_{rk},c_{ck})'$"

Explanation: Property (i) denotes that there should exist two nontrivial transition sequences and they are disjoint one another. Property (ii) denotes that transition sequence σ_i does not involve any inheritance while transition sequence σ_j involves inheritance. Property (iii) denotes that marking M' is reachable from initial marking M_0 by the first sequence σ_i and marking M" is reachable from M_0 by the second sequence σ_j. Property (iv) denotes that no state token is deposited in Place k in the initial marking. While in markings M' and M", there is at least one state token deposited in Place k. Property (v) is similar to (iv) except that the markings are referring to class tokens. Property (vi) denotes that there exists a class token element $(p_{rk},c_{ck})'$ in predicate place k of M'. There is also a token element $(p_{rk},c_{ck})''$ which exists in predicate place k of marking M". Property (vii) tells us that the colour (data value) of predicate k of this two class tokens are the same.

5.2. Subsumption

Proposition 5.2. For a given marking M_0, that minimally enables a nontrivial transition sequence σ_i, iff the HES has incorrect rules causing subsumption between the parent and child object classes, then $\exists\sigma_j$, $\exists k$, such that these sequences have the following properties:

(i) $\sigma_i \cap \sigma_j = \varnothing$;
(ii) $T_c \cap \sigma_i = \varnothing; T_c \cap \sigma_j \neq \varnothing$;
(iii) $M' = \delta(M_0, \sigma_i)$, $M'' = \delta(M_0, \sigma_j)$;
(iv) $M_{sk}=0$, $M'_{sk}>0$, $M''_{sk}>0$;
(v) $M_{ck}=0$, $M'_{ck}>0$, $M''_{ck}>0$;
(vi) $\exists(p_{rk},c_{ck})' \in M'_{ck}$, $\exists(p_{rk},c_{ck})'' \in M_{ck}$"
(vii) $(p_{rk},c_{ck})'' \subseteq (p_{rk},c_{ck})'$

5.3. Ambiguity

Proposition 5.3. For a given marking M_0, that minimally enables $\Gamma=\{\sigma_i, \sigma_j\}$ for a nontrivial transition sequence σ_i, σ_j, iff the HES has incorrect rules causing ambiguous conditions of events between different object classes, then $\exists k$, $\forall p_{rk} \in \ddot{O}_s(\Gamma)$, $\forall p_{rk} \in \ddot{O}_c(\Gamma)$, such that these sequences have the following properties:

(i) $\sigma_i \cap \sigma_j = \varnothing$;
(ii) $M' = \delta(M_0, \sigma_i)$, $M'' = \delta(M', \sigma_j)$;
(iii) $M_{sk}=0$, $M'_{sk} \geq 1$, $M''_{sk}>1$;
(iv) $M_{ck}=0$, $M'_{ck} \geq 1$, $M''_{ck}>1$;
(v) $\exists(p_{rk},c_{ck})' \in M'_{ck}$, $\exists(p_{rk},c_{ck})'' \in M_{ck}$"
(vi) $(p_{rk},c_{ck})' = (p_{rk},c_{ck})''$

5.4. Circular Rule Sets

Proposition 5.4. For a given marking M^0, that minimally enables transition sequence α, iff the HES has incorrect rules causing cyclicity between the parent and child object classes, then $\exists j \geq i$, $\exists k$ such that the sequence has the following properties:

(i) $M^i \in [M^0> = \{M^0, M^1, M^2, \ldots M^i, ..M^j\}$,
(ii) $M^j = \delta(M^0, \alpha)$ for $j>0$,
(iii) $T_c \cap \alpha \neq \varnothing$;
(iv) $M^i_{sk}=0$, $M^i_{sk}>0$, $M^j_{sk}>1$;
(v) $M^i_{ck}=0$, $M^i_{ck}>0$, $M^j_{ck}>0$;

6. Conclusion

In this paper, we have described a formal description technique based on State Controlled Coloured Petri Nets to model hybrid (rule- and frame-based) expert systems. The technique allows the use of reachability theory for the verification of the systems. The paper illustrates the capability of the technique to identify the anomalies due to the incorrectness of the hybrid knowledge base. The verification was done exhaustively by minimally initiating any sequence of transitions and closely examining the reachability markings at each transition. A set of propositions is formulated to verify errors and anomalies in HES.

Future work will include measuring and analyzing the state-space complexity of HES and evaluating our approach for modelling and verification. We would also like to investigate further the capability of the methodology to handle fuzzy and temporal expert systems.

References

[1] Agarwal, R. & Tanniru, M. (1992). A Petri Net based approach for verifying the integrity of production systems. *International Journal of Man Machine Studies*. Vol. 36. 447-468.

[2] Aikins, J.S. (1993). Prototypical Knowledge for Expert Systems: a retrospective analysis. In Bobrow D.G. (Ed.) *Artificial Intelligence*. Vol. 59. pp. 207-211. Elsevier, Amsterdam.

[3] Coenen, F. & Bench-Capon, T. (1993). *Maintenance of Knowledge-based Systems.* Academic Press.

[4] Dori, D. & Tatcher, E. (1994). Selective multiple inheritance. *IEEE Software*. Vol. 11. No. 3, 77-85.

[5] Durkin, J. (1994). *Expert Systems: Design and Development*. Macmillan Publishing Company. 12-23;711-771.

[6] French, S.W. & Hamilton, D. (1994). A Comprehensive Framework for Knowledge-Base Verification and Validation. International *Journal of Intelligent Systems*. Vol. 9. 809-837.

[7] Gamble, R. F. & Baughman D. M. (1996). A methodology to incorporate formal methods in hybrid KBS verification. *International Journal of Human Computer Studies*. Vol. 44, 213-244.

[8] Gamble, R.F., Roman G., Ball W.E. & Cunningham H.C. (1994). Applying Formal Verification Methods to Rule-Based Programs. *International Journal of Expert Systems*. Vol. 7, no. 3, 203-239.

[9] Gupta, U. (Ed.) (1991). *Validating and Verifying Knowledge-based Systems*. IEEE Computer Society Press.

[10] Jensen, K. (1995). *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Vol 2. Springer-Verlag.

[11] Jensen, K. (1996). *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use.* Vol. 1. 2nd Ed. Springer-Verlag.

[12] Lee, S. & O'Keefe, R.M. (1993). Subsumption Anomalies in Hybrid Knowledge Bases. *International Journal of Expert Systems*. Vol. 6, No. 3, 299-320.

[13] Liu, N. K. & Dillon T. (1995). Formal Description and Verification of Production Systems. *International Journal of Intelligent Systems*. Vol. 10, 399-442.

[14] Liu, N. K. & Dillon, T. (1995). Formal Description and Verification of Production Systems. *International Journal of Intelligent Systems*. Vol. 10, 399-442.

[15] Murrell, S. & Plant, R. (1996). On the Validation and Verification of Production Systems: a graph reduction approach. *International Journal of Human Computer Studies*. Vol. 44, 127-144.

[16] Nazareth, D. L. (1993). Investigating the Applicability of Petri Nets for Rule-Based System Verification. *IEEE Transactions on Knowledge and Data Engineering*. Vol. 4, No. 3, 402-415.

[17] O'Keefe, R.E. & O'Leary, D.E. (1993). Expert System Verification and Validation: A survey and tutorial. *Artificial Intelligence Review*. Vol. 7, 3-42.

[18] Scarpelli, H. & Gomide, F. (1994). A high level net approach for discovering potential inconsistencies in fuzzy knowledge bases. *Fuzzy Sets and Systems*. Vol. 64, 175-193.

[19] Shiu, S., Liu, J. & Yeung, D. (1995a). Modelling Hybrid Rule/Frame-based Expert Systems Using Coloured Petri Nets. In *Proceedings of 8th International Conference on Industrial & Engineering Applications of AI & ES*. Melbourne, Australia. 525-532.

[20] Shiu, S., Liu, J. & Yeung, D. (1995b). An Approach Towards the Verification of Hybrid Rule/Frame-based Expert Systems using Coloured Petri Nets. In *Proceedings of 1995 IEEE International Conference on SMC*. Vancouver. 2257-2262.

[21] Shiu, S., Liu, J. & Yeung, D. (1996a). An Approach Towards the Verification of Fuzzy Hybrid Rule/Frame-based Expert Systems using Coloured Petri Nets. In *Proceedings of ECAI-96 Workshop in Validation, Verification and Refinement of KBS*. Budapest, 105-113.

[22] Shiu, S., Liu, J. & Yeung, D. (1996b). Proofs of the Formal Verification of Hybrid Rule/Frame-based Expert Systems using SCCPN. *Unpublished Manuscript*. Department of Computing, Hong Kong Polytechnic University.

[23] Vranes, S. & Stanojevic, M. (1995). Integrating Multiple Paradigms within the Blackboard Framework. *IEEE Transactions on Software Engineering*. Vol. 21, No. 3, 244-262.

[24] Willis, C.P. (1996). Analysis of inheritance and multiple inheritance. *Software Engineering Journal*. July.

[25] Zhang, D. & Nguyen D. (1994). PREPARE: A Tool for Knowledge Base Verification. *IEEE Transactions on Knowledge and Data Engineering*. Vol. 6, No. 6, December, 983-989.