Verification of Qualitative Properties of Rule-Based Expert Systems

Alfonsus D. Lunardhi and Kevin M. Passino[1]
Dept. of Electrical Engineering
The Ohio State University
2015 Neil Avenue
Columbus, OH 43210-1272

Abstract

Frequently expert systems are being developed to operate in dynamic environments where they must reason about time-varying information and generate hypotheses, conclusions, and process inputs that can drastically influence the environment within which they operate. For instance, expert systems used for fault diagnosis and fault accomodation in nuclear power plants reason over sensor data and operator inputs, form fault hypotheses, make recommendations pertaining to safe process operation, and in crisis situations could generate command inputs to the process to help maintain safe operation. Clearly, there is a pressing need to verify and *certify* that such expert systems are dependable in their operation and can reliably maintain adequate performance levels. In this paper we develop a mathematical approach to verifying qualitative properties of rule-based expert systems that operate in dynamic and uncertain environments. First, we provide mathematical models for the expert system (including the knowledge-base and inference engine) and for the mechanism for interfacing to the user inputs and the dynamic process. Next, using these mathematical models we show that while the structure and interconnection of information in the knowledge-base influence the expert system's ability to react appropriately in a dynamic environment, the qualitative properties of the full knowledge-base/inference engine loop must be considered to fully characterize an expert system's dynamic behavior. To illustrate the verification approach we show how to model and analyze the qualitative properties of rule-based expert systems that solve a water-jug filling problem and a simple process control problem. Finally, in our concluding remarks we highlight some limitations of our approach and provide some future directions for research.

Index Terms: Expert Systems, Knowledge-Based Systems, Knowledge-Base Verification, Dependability, Reliability

1 Introduction

Enhanced computing technology and the growing popularity of applied artificial intelligence (AI) have resulted in the construction and implementation of extremely complex "rule-based" expert systems [1, 2, 3, 4]. Often such expert systems are being utilized in *critical environments* where hazards can occur, safety of humans is an issue, and hard real-time constraints must be met. For instance, some expert systems for aircraft applications are used for mission planning, others have been used for closed-loop control of dynamical systems, while in industrial processes they can be used for diagnosing failures [5]. Most often such expert systems are constructed and implemented without any *formal analysis* of the dynamics of how they interface to their environment (e.g., to users and process data) and how the inference mechanism dynamically reasons over the information in the knowledge-base. Currently many expert systems are evaluated either (*i*) in an empirical manner by comparing the expert system against human experts, (*ii*) by studying reliability and user friendliness, (*iii*) by examining the results of extensive simulations, and/or (*iv*) by using software engineering approaches [6, 1, 3, 4, 7, 8, 9, 10, 11, 12]. While it is recognized that these approaches play an important role in expert system verification, it is also important to recognize the importance of investigating the possibility of mathematically verifying the qualitative properties of general rule-based expert systems that interface to a user *and* a dynamical process that behaves in an unpredictable manner. The focus of this paper is to conduct

[1]This work was supported in part by National Science Foundation Grant IRI-9210332. Please address all correspondence to Kevin Passino ((614) 292-5716; email: passino@ee.eng.ohio-state.edu). **Bibliographic information for this paper: Lunardhi A.D., Passino K.M., "Verification of Qualitative Properties of Rule-Based Expert Systems", Int. Journal of Applied Artificial Intelligence, Vol. 9, No. 6, pp. 587–621, Nov./Dec. 1995.**

such a mathematical investigation. Our approach to mathematical analysis does not obviate the need for the past approaches to expert system verification; generally speaking it augments their abilities (*i*) by considering expert systems that operate in dynamic and uncertain environments, (*ii*) by characterizing and analyzing more general properties of expert systems, and (*iii*) by showing how techniques from nonlinear analysis of dynamical systems can be applied to the study of properties of dynamical expert systems. As with the past techniques, our goal is to enhance our confidence that such expert systems will behave properly upon implementation.

The expert system, which consists of the inference engine and knowledge-base, is shown in Figure 1. The inputs to the expert system come from the *user* (typically a human) and from data or information generated by the dynamic process that the expert system is connected to. The outputs of the expert system represent hypotheses, conclusions, or command inputs to change some process variable (i.e., "process command inputs")[2]. In this paper we focus on the class of expert systems that has a knowledge-base which consists of *rules* that characterize strategies on how to perform the task at hand. The inference engine is designed to emulate a human expert's decision-making process in collecting user inputs and process outputs and in reasoning about what process input to generate. In this paper we focus on the use of relatively standard *conflict resolution strategies* in our inference engine such as those used in [2].

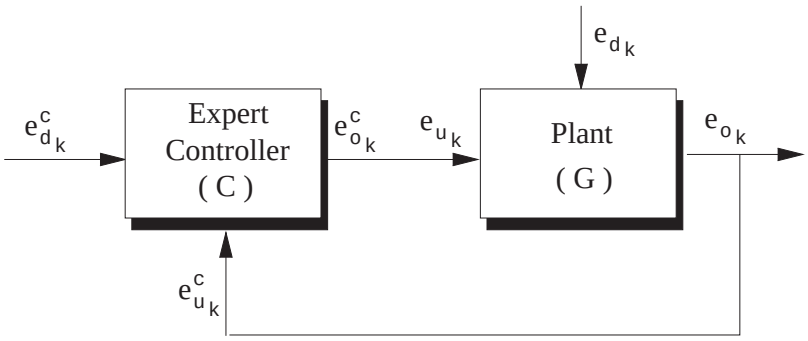

Figure 1: Expert System

The verification of the dynamic properties of of certain very general AI reasoning systems is beyond the scope of this work. For example, while the expert system we consider can be designed to exhibit some learning capabilities since it has variables in its working memory, because the number of rules is fixed, certain types of automatic rule synthesis are not possible (e.g., it can learn to pick between what rule is most appropriate to fire, but it cannot synthesize an indeterminant number of completely new rules). Our expert system can synthesize a finite number of rules that seek to enhance its performance based on its past experience and it can adapt its working memory and inference mechanism. It is also important to note that the expert system considered here cannot plan ahead an indeterminant number of steps into the future, taking into account what might happen as the result of its actions. Our expert system can, however, plan ahead a finite number of steps into the future to ensure that it takes the proper action. Finally, we note that while we use the AI terminology for expert systems, we question the validity of the standard AI models in representing the actual human cognitive structure and processes. Regardless, the focus

[2]While it is clear that the expert system can generate process inputs, hypotheses, conclusions, or recommendations we will, for convenience, in the remainder of this paper refer to all such quantities as expert system outputs or process inputs. In this sense we consider the human user that interfaces to the expert system and a dynamical system (such as an industrial process), to both be a part of the "environment" of the expert system.

here is not on whether we have a good model of the human expert (as it is in, e.g., [12]), but rather on whether the expert system performs adequately and dependably. The first step in formally verifying the qualitative properties of the expert system is to develop appropriate mathematical models.

Section 2 shows how a mathematical model can be utilized to represent the inference engine and knowledge-base shown in Figure 1. Our approach to modeling the rule-base is related to the work in [13] where the authors show how to model rule-based AI systems with a high-level Petri net [14]. The approach in this paper is similar to the one advocated in [15] where we utilize separate models for the inference engine and knowledge-base and view their interconnection (i.e., the *inference loop*) as a special type of closed-loop control system. In Section 2 we introduce a single model that can represent both the inference engine and the knowledge-base, *and* an interface to user inputs and the dynamic process (something that was not considered in [15]).

In Section 3 we show that while the structure and interconnection of information in the knowledge-base influence its *completeness* and *consistency* properties [16, 17, 18, 19, 20] and the expert system's ability to react appropriately within its environment, certain "qualitative properties" of the full knowledge-base/inference engine loop (along with the interface to the user inputs and dynamic process) must be considered to fully characterize an expert system's behavior. In addition, in Section 3 we show how to characterize and analyze the following qualitative properties of rule-based expert systems that interface to users and a dynamic process: (*i*) reachability, (*ii*) cyclic behavior, and (*iii*) stability. Verification of certain reachability properties can ensure that the expert system will be able to infer appropriate conclusions from certain information in its knowledge-base and process outputs and user inputs. Testing for cyclic behavior checks that the expert system will not get stuck in an inappropriate infinite loop (e.g., exhibit *circular reasoning* before it reaches its goal). Verification of certain stability properties can ensure that the expert system can stay *focused* on the task at hand and not exhibit certain types of detrimental cyclic behavior. The authors emphsize that while (i) certain stability properties must be satisfied for all expert systems to ensure their proper behavior upon implementation (e.g., to ensure that variables in working memory will not become unbounded), and (ii) the nonlinear dynamical systems analysis community has performed extensive stability analysis for the verification and certification of real-world systems, this seems to be the first work that focuses on stability analysis of expert systems (which are in fact nonlinear systems). Finally, we note that the work in this paper and the work in [15] is related to the work presented in [21, 22] on a system-theoretic characterization and analysis of qualitative properties of AI planning systems (the work in [21] was developed for use in modeling and analysis of qualitative properties of AI subsystems in advanced aircraft).

To illustrate the application of the results, in Sections 4 and 5 we perform modeling and reachability and stability analysis of an expert system that solves a water-jug filling problem (where there is no interface to user inputs or a dynamic process) and a simple process control problem where user inputs and dynamic process information are used in the expert system's reasoning process. In Section 6 we overview some problems that can be encountered in conducting a formal mathematical verification of qualitative properties of expert systems and we highlight some research directions that will seek to address some of the deficiencies with the approach reported in this paper.

2 The Expert System

We begin by specifying the dynamical system that will interface to the expert system. The dynamic process has a set of inputs $e_u \in \mathcal{E}_u$ that can be manipulated by the expert system (call these "command input events" where we use "u" as a subscript since in the study of nonlinear systems "u" is standard notation for this input), a set of

disturbances $e_d \in \mathcal{E}_d$ that occur randomly and unpredictably (call these "disturbance input events"), and a set of process outputs $e_o \in \mathcal{E}_o$ that can be observed by an expert system (call these "output events"). We will use e_{u_k}, e_{d_k}, and e_{o_k} to denote such input and output events at time k. Let

$$\mathcal{E} = \mathcal{E}_u \cup \mathcal{E}_d \cup \mathcal{E}_o \tag{1}$$

and let $e_k \in \mathcal{P}(\mathcal{E}) - \{\emptyset\}$ denote an *event* at time k (where $\mathcal{P}(\mathcal{E})$ denotes the power set of $\mathcal{E}$). The dynamical behavior of the process evolves by the occurrence of events e_k over time. For convenience, we assume that there is always an output event in e_k and at most one command input event and disturbance input event in e_k for all $k \geq 0$. Define $\mathbf{E}$ to be the set of all infinite and finite length *event trajectories* (sequences of events) that can be formed from events $e_k \in \mathcal{P}(\mathcal{E}_u \cup \mathcal{E}_d \cup \mathcal{E}_o) - \{\emptyset\}$. The set $\mathbf{E}_v \subset \mathbf{E}$ is the set of all *physically possible* event trajectories for the dynamical process that the expert system is connected to (i.e., "valid trajectories"). Note that: (i) the disturbance input events characterize random and unpredictable behavior in the process, and (ii) typically, certain sequences of command inputs and disturbance inputs will generate certain sequences of output events. We must emphasize that this is a unique and very general way to specify the dynamical system that interfaces to the expert system. It actually allows us to represent any dynamical system that can be represented with Petri nets, automata, and general nonlinear difference equations. Note that the above model for the dynamical system is quite appropriate since it provides a general input-output model and it is only via these inputs and outputs that the expert system interfaces to it. It is for this reason that from here till the end of the paper, to discuss the effects of the dynamical system on the expert system we only need to use the dynamical system's input and output events. In Section 5 we provide an example to illustrate the use of our formalization for representing the dynamic process that an expert system is connected to (for this example we use an automata-like description of the dynamical process that interfaces to the expert system to generate the event trajectories that the expert system observes). Next, we will specify a mathematical model for the expert system.

The expert system, which is denoted by "C", shown in Figure 1 has two inputs; the user input events $e_d^c \in \mathcal{E}_d^c$ (where we use the superscript "c" to indicate that it is an input associated with the expert system C and the subscript "d" is used to indicate that the user input is a "disturbance" in the sense that the expert system does not know what the user will request next) and the output events of the process $e_o \in \mathcal{E}_o$. Based on its *state* (to be defined below) and these inputs, the expert system generates command input events to the process $e_o^c \in \mathcal{E}_o^c$ (and/or hypotheses and recommendations). We will often speak of the interactions between the inference engine and knowledge-base shown in the expert system in Figure 1 as forming an "inference loop". This inference loop constitutes the core of the expert system where information in the knowledge-base is interpreted by the inference engine, actions are taken, the knowledge-base is updated, and the process repeats (i.e., "loops"). The full expert system shown in Figure 1 is modeled by C where

$$C = (\mathcal{X}^c, \mathcal{E}^c, f_e^c, \delta^c, g^c, \mathbf{E}_v^c) \tag{2}$$

where

$\mathcal{X}^c = \mathcal{X}^b \times \mathcal{X}^i$ is a set of *expert system states* x^c, where $\mathcal{X}^b$ is the set of
knowledge-base states $\mathbf{x}^b$ and $\mathcal{X}^i$ is the set of inference engine states $\mathbf{x}^i$ to be defined below,

$\mathcal{E}^c = \mathcal{E}_u^\ell \cup \mathcal{R} \cup \mathcal{E}_o^c$ is the set of *events* of the expert system C where

$\mathcal{E}_u^\ell \subset \mathcal{P}(\mathcal{E}_o \cup \mathcal{E}_d^c) - \{\emptyset\}$ is the set of sets of *user input* ($\mathcal{E}_d^c$) and *process output events* ($\mathcal{E}_o$)
that can occur for which the expert system will have to know how to respond to
(the superscript "ℓ" is used to indicate that $\mathcal{E}_u^\ell$ is the list

of all input events, i.e., both process input and user input events)

$\mathcal{R}$ is the set of *rules* in the *knowledge-base* of the expert system,

$\mathcal{E}_o^c \subset \mathcal{P}(\mathcal{E}_u) - \{\emptyset\}$ is a set of output events of the expert system

$g^c : \mathcal{X}^b \times \mathcal{X}^i \to \mathcal{P}(\mathcal{E}_u^\ell \cup \mathcal{R}) - \{\emptyset\}$ is the *enable function* for C,

$f_e^c : \mathcal{X}^b \times \mathcal{X}^i \to \mathcal{X}^b \times \mathcal{X}^i$ for $e \in \mathcal{P}(\mathcal{E}_u^\ell \cup \mathcal{R}) - \{\emptyset\}$ are the *state transition maps* for C,

$\delta_e^c : \mathcal{X}^b \times \mathcal{X}^i \to \mathcal{E}_o^c$ for $e \in \mathcal{P}(\mathcal{E}_u^\ell \cup \mathcal{R}) - \{\emptyset\}$ are the *output maps*

which specify the outputs of the expert system C,

$\mathbf{E}_v^c \subset \mathbf{E}^c$ is the set of *valid inference loop (expert system) trajectories*

(expert system event trajectories that are physically possible,

i.e., valid trajectories for the expert system)

Note that we use "c" to denote that each of the above elements of the tuple in (2) are associated with the expert system C

In this framework, it is assumed that an occurrence of an input event to the expert system $e_u^\ell \in \mathcal{E}_u^\ell$ is always accompanied by a firing of an enabled rule $r \in \mathcal{R}$, so that the inference loop can be updated accordingly[3]. Similarly, a rule $r \in \mathcal{R}$ cannot fire alone, since the inference loop is updated only if there is a change in the process reflected via its output or a change in the user input event (this does not mean that the expert system cannot reason in between updates to the inference loop). Hence, each e_u^ℓ has at most one process output event $e_o \in \mathcal{E}_o$ and user input event $e_d^c \in \mathcal{E}_d^c$ contained in it. Events $e_k \subset g(x_k^c)$ are said to be *enabled*. If $e_k \subset g(x_k^c)$ and e_k *occurs*, then the next state $x_{k+1}^c = f_{e_k}^c(x_k^c)$; hence, the dynamical behavior of the expert system evolves by the occurrence of sequences of events (i.e., *event trajectories* corresponding to the firing of rules) which result in the generation of state trajectories (i.e., state sequences). Let $\mathbf{E}^c$ denote the set of event sequences that can occur based on the definition of f_e^c and g^c for the expert system C and let $\mathbf{E}_v^c \subset \mathbf{E}^c$ denote the set of event trajectories that are physically possible (i.e., valid) for the expert system C. The expert system can control the generation of command input events for the process; however, it does not have any capabilities to control the process disturbance input events. The full specification of C is achieved by defining the rule-base and inference engine for the expert system, i.e., by defining the components of the inference loop.

2.1 Modeling a Rule-Base

It is important to note that although the focus in this paper is on rule-based systems, we are not restricted to modeling only rule-based systems; other AI knowledge representation formalisms can also easily be represented. To see this first note that any system that can be represented with the General, Extended, or High-Level Petri Net [14] can be represented with C. Then the Petri Net can be used to represent, for instance, *semantic nets, frames,* or *scripts*. Alternatively, one could directly model such knowledge representation schemes with C. Also note that in [23] the authors show that the rule-base and fuzzy inference mechanism of a general multiple-input multiple-output fuzzy system [24] can be represented with the model C. Next, we model the rule-base.

Let $A = \{a_1, a_2, ..., a_n\}$ be a set of *facts* that can be true or false (and their truth values can change over time).

[3]Note that without loss of generality in our framework only one rule fires at each time instant. If one wants to fire more than one rule at a time, one can define another rule that represents the combined effects of any number of rules. Alternatively, one can simply redefine our model so that it can represent the firing of many rules at each time step (by redefining f so that it maps *sets* of fired rules and the current state to the next state)

Let

$$T : A \rightarrow \{0, 1\} \tag{3}$$

where $T(a_i) = 1(= 0)$ indicates that a_i is true (false). Let $\Re$ denote the real numbers, $\mathbf{V} \subset \Re^m$, and $\mathbf{v} \in \mathbf{V}$ denote an m-dimensional column vector of *variables*. We are thinking here of facts and variables in "working memory" [2]. Let $\mathcal{X}^b = \Re^{m+n}$ where $\mathbf{x}^b \in \mathcal{X}^b, \mathbf{x}^b = [\mathbf{v}^t\, T(a_1)\, T(a_2)\, ...\, T(a_n)]^t = [x_1^b\, x_2^b\, ...\, x_{m+n}^b]^t$ (t denotes transpose) and let $x_{i_k}^b$ denote the *ith* component of $\mathbf{x}^b$ at time k and $T_k(a_i)$ denote the truth value of a_i at time k. Let $P_i, i = 1, 2, ..., p$ denote a set of p *premise functions*, i.e.,

$$P_i : \mathcal{X}^b \times \mathcal{E}_u^\ell \rightarrow \{0, 1\} \tag{4}$$

and $P_i(\mathbf{x}_k^b, e_{u_k}^\ell) = 1(= 0)$ indicates that $P_i(\mathbf{x}_k^b, e_{u_k}^\ell)$ is true (false) at time k. The P_i will be used in the premises of the rules to state the conditions under which a rule is *enabled* (i.e., they model the left-hand sides of rules). Let the *antecedent formulas*, denoted by Φ, be defined in the following recursive manner:

1. $T(a)$ for all $a \in A$, and $P_i, i = 1, 2, \ldots, p$ are antecedent formulas.

2. If Φ and Φ' are antecedent formulas then so are $\neg\Phi, \Phi \wedge \Phi', \Phi \vee \Phi'$, and $\Phi \Rightarrow \Phi'$, (where $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\Rightarrow$ (implies) are the standard Boolean connectives).

3. Nothing else is an antecedent formula unless it is obtained via finitely many applications of 1-2 above.

For example, if $m = 3$, $n = 2$, $A = \{a_1, a_2\}$, $\mathbf{V} \subset \Re^3$, and P_1 tests "$x_{2_k}^b < 5.23$", P_2 tests "$x_{3_k}^b = 1.89$", $e_{d_k}^c$ and e_{o_k} are real numbers, and P_3 tests "$(e_{d_k}^c < 5) \vee (e_{o_k} \geq 2)$", then $\Phi' = P_1 \wedge P_2 \wedge P_3 \wedge (T(a_1) \vee \neg T(a_2))$ is a valid antecedent formula (where $<, \geq$ and $=$ take on their standard meaning). Let $C_i, i = 1, 2, ..., q$ denote the set of q *consequent functions*, where

$$C_i : \mathcal{X}^b \times \mathcal{E}_u^\ell \rightarrow \mathcal{X}^b \times \mathcal{E}_o^c \tag{5}$$

will be used in the representation of the consequents of the rules (the right-hand sides of the rules), i.e., to represent what actions are taken to the knowledge-base when a rule is *fired*. Let the *consequent formulas*, denoted with Ψ, be defined in the following recursive manner:

1. For any $C_i, i = 1, 2, \ldots, q, C_i$ is a consequent formula.

2. For any C_i, C_j, $C_i \wedge C_j$ is a consequent formula.

3. Nothing else is a consequent formula unless it is obtained via finitely many applications of 1-2 above.

Following the above example for the premise formula, C_1 may be $x_{4_{k+1}}^b = T(a_1) := 1$ (make a_1 true), C_2 may mean let $x_{2_{k+1}}^b := x_{2_k}^b + 2.9$, C_3 may mean let $x_{3_{k+1}}^b := e_{d_k}^c/2$, and $\Psi' = C_1 \wedge C_2 \wedge C_3$ makes a_1 true ($x_{4_{k+1}}^b := 1$), increments x_2^b (variable v_2) and assigns $e_{d_k}^c/2$ to $x_{3_{k+1}}^b$. Notice that we could also define the C_i such that $C_i : \mathcal{X}^b \times \mathcal{X}^i \times \mathcal{E}_u^\ell \rightarrow \mathcal{X}^b \times \mathcal{X}^i \times \mathcal{E}_o^c$ so that the rules could characterize changes made to the inference strategy based on the state of the knowledge-base and/or the user input (i.e., the inference strategy could be changed based on the current objectives stated in the user input). Similar, more general definitions could be made for the P_i above (for example, $\mathcal{X}^i$ could be used in the domain of the P_i). In this paper we will not consider such possibilities and hence we will focus solely on the use of the P_i and C_i defined in Equations (4) and (5) above.

The rules in the knowledge-base $r \in \mathcal{R}$ are given in the form of

$$r = \text{IF} \quad \Phi \quad \text{THEN} \quad \Psi \tag{6}$$

where the action Ψ can be taken only if Φ evaluates to true. Formally for (6), $e_k = \{r, e^{\ell}_{u_k}\} \subset g^c(x^c_k)$ can *possibly occur* (the inference engine may not let it occur) only if Φ evaluates to true at time k for the given state $\mathbf{x}^b_k$ and the command input event $e^{\ell}_{u_k}$. Note that many rules can be enabled at each time step and some rules can have their premises satisfied in possibly an infinite number of ways; hence for a given x^c_k, the size of $g^c(x^c_k)$ can be infinite even though there are only a finite number of rules (e.g., if $\Phi = x > 2.2$ there are an infinite number of values of x that will make Φ true and therefore make rule r enabled). If $e_k \subset g^c(x^c_k)$ occurs, then the next state $x^c_{k+1} = f^c_{e_k}(x^c_k)$ is given by: (i) the application of Ψ to the state $\mathbf{x}^b_k \in \mathcal{X}^b$ to produce $\mathbf{x}^b_{k+1}$, and (ii) updating the inference engine state $\mathbf{x}^i \in \mathcal{X}^i$ which will be discussed in Section 2.2. Also, in this case the output of the expert system (input to the dynamic process) is $\delta^c_{e_k}(x^c_k) \in \mathcal{E}^c_o$. The inclusion of input events $\mathcal{E}^{\ell}_u$ in the rule-base allows the expert system designer to incorporate the process output information and the user input variables directly as parts of the rules. This is analogous to the use of *variables* in conventional rule-based expert systems (e.g., see the description of the OPS5 rule grammar in [2]).

2.2 Modeling the Inference Engine

To model the inference engine one must be able to represent its three general functional components [2]:

1. **Match Phase:** The premises of the rules are matched to the current facts and data stored in the knowledge-base and to the user input and process output.

2. **Select Phase:** One rule is selected to be fired, and

3. **Act Phase:** The actions indicated in the consequents of the fired rule are taken on the knowledge-base, the inference engine state is updated, and the input to the process is generated.

Here, the characteristics of the "match phase" of the inference mechanism are inherently represented in the knowledge-base. In AI terminology

$$\Gamma_k = \{r : \{r, e^{\ell}_{u_k}\} \subset g^c(x^c_k) \text{ so that the } \Phi \text{ of rule } r \in \mathcal{R} \text{ evaluates to true for } e^{\ell}_{u_k}\} \tag{7}$$

is actually the knowledge-base "conflict set" at time k (the set of enabled rules in terms of the knowledge-base only). The select phase (which picks one rule from Γ_k to fire) is composed of "conflict resolution strategies" (heuristic inference strategies [2, 25, 26]) of which a few representative ones are listed below:

1. Refraction: All rules in the conflict set that were fired in the past are removed from the conflict set. However, if firing a rule affects the matching data of the other rules' antecedents, those rules are allowed to be considered in the conflict resolution.

2. Recency: Use an assignment of priority to fire rules based on the "age" of the information in the knowledge-base that matches the premise of each rule. The "age" of the data that matches the premise of a rule is defined as the number of rule firings since the last firing of the rule which allows it to be considered in the conflict set.

3. Distinctiveness: Fire the rule that matches the most (or most important) data in the rule-base (many different types of distinctiveness measures are used in expert systems). Here, we will count the number of different terms used in the antecedent of a rule and use this as a measure of its distinctiveness.

4. **Priority Schemes:** Assign a priority ranking of the rules then choose from the conflict set the highest priority rule to fire.

5. **Arbitrary:** Pick a rule from the conflict set to fire at random.

It is understood that the distinctiveness conflict resolution strategy is actually a special case of a priority scheme but we include both since distinctiveness has, in the past, been found to be useful in the development of expert systems. Note that in a particular expert system any number of the above conflict resolution strategies (in any fixed, or perhaps variable order) may be used to determine which rule from the conflict set is to be fired. Normally, these conflict resolution strategies are used to "prune" the size of the knowledge-base conflict set Γ_k until a smaller set of enabled rules is obtained. These rules are the "enabled rules" in the model C of the combined knowledge-base and inference engine after the conflict resolution pruning. If all the conflict resolution strategies are applied and more than one rule remains, then (5) above ("Arbitrary") is applied to randomly fire (not according to any particular statistics) one of the remaining rules. The act phase will be modeled by the operators f_e^c which represent the actions taken on the knowledge-base and inference engine if a rule with the corresponding input event to the inference loop occurs.

The priority and distinctiveness of a rule in the knowledge-base are fixed for all time, but the refraction and recency vary with time. Thus, the inference engine state $\mathbf{x}^i$ has to carry the information regarding both refraction and recency. Assume that the knowledge-base has n_r rules and the rules are numbered from 1 to n_r. Define a function $\Pi(i)$ to be 1 if the rule i is deleted from the conflict set, and 0 if rule i is allowed to be considered in conflict resolution. This function is used for representing the refraction component of the select phase. Let $\mathbf{p} = [\Pi(1)\Pi(2)\Pi(3)\ldots\Pi(n_r)]^t$ be an n_r-vector whose components represent whether a rule can be included in the conflict set when it is enabled in state $\mathbf{x}^b$. Let the n_r-vector $\mathbf{s} = [s_1\ s_2\ s_3\ \ldots\ s_{n_r}]^t$ where s_i is an integer representing the *age* of information in the knowledge-base which matches the premise of rule i (to be fully defined below). We will use $\mathbf{s}$ to help represent the recency conflict resolution strategy. The inference engine state is defined as $\mathbf{x}^i = [\mathbf{p}^t\ \mathbf{s}^t]^t \in \mathcal{X}^i$.

To complete the model of the expert system we need to fully define g^c and f_e^c. The state transitions that occur to update $\mathbf{p}$ and $\mathbf{s}$ are based on the refraction and recency of the information represented by the components of $\mathbf{x}^i$. A matrix $\mathbf{A}$ is used to specify how to update $\mathbf{p}$ and $\mathbf{s}$ and is defined to have a dimension of $n_r \times n_r$ and its ij^{th} component, $a_{ij} = 1(0)$ if firing rule i (does not) affects the matching data of rule j. Essentially, $\mathbf{A}$ contains static information about the interconnecting structure of the rule-base which is automatically specified once the rules are loaded into the knowledge-base and before the dynamic inference process is started. It provides a convenient way to model the recency and refraction schemes.

We use variables $\tilde{e}_i, d_i$, and p_i, for $i, 1 \le i \le n_r$ to define the update process for $\mathbf{x}^b, \mathbf{p}$, and $\mathbf{s}$ where $\tilde{e}_i = 1(0)$ indicates that rule i is enabled (disabled), d_i holds the distinctiveness level of rule i (the higher the value is, the more distinctive the rule is), and p_i holds the priority level of rule i (the priority is proportional to the p_i value). The d_i and p_i components are specified when the knowledge-base is defined and they remain fixed. The values of $s_i, \tilde{e}_i$, and $\Pi(i)$ change with time k, so we use $s_i^k, \tilde{e}_i^k$, and $\Pi_k(i)$ respectively to denote their values at time k.

The inference loop in the expert system can be executed in the following manner: First, through "knowledge acquisition" the knowledge-base is defined; then $\mathbf{p}, \mathbf{s}$, and $\tilde{e}_i, 1 \le i \le n_r$ are initialized to 0. The inference step from k to $k+1$ is obtained by executing the three following steps (we list this in a "psuedocode" form to help clarify how we have done our analysis for our applications in Sections 4 and 5):

1. *Match Phase*

FOR rule $r = 1$ TO rule $r = n_r$ DO:
IF $r \in \Gamma_k$ THEN $\tilde{e}_r^k := 1$ { Finds the enabled rules }
IF there is just one r' such that $\tilde{e}_{r'}^k = 1$ THEN GOTO the *Act Phase*
IF there are no r' such that $\tilde{e}_{r'}^k = 1$ THEN STOP { expert system not
properly defined, i.e., it cannot properly react to all possible process output/user input conditions. }

2. *Select Phase*

FOR rule $r = 1$ TO rule $r = n_r$ DO: { Pruning based on refraction }
IF $\tilde{e}_r^k = 1$ THEN
IF $\Pi_k(r) = 1$ THEN $\tilde{e}_r^k := 0$
IF there is just one r' such that $\tilde{e}_{r'}^k = 1$ THEN GOTO the *Act Phase*
IF there are no r' such that $\tilde{e}_{r'}^k := 1$ THEN STOP { Expert system not properly defined }
LET $s = -\infty$ {Pruning based on recency }
FOR $j = 1$ TO 2 DO: { Search for rule(s) with the lowest age value(s) }
FOR rule $r = 1$ TO rule $r = n_r$ DO:
IF $\tilde{e}_r^k = 1$ THEN
IF $-s_r^k < s$ THEN $\tilde{e}_r^k := 0$
ELSE s:=$-s_r^k$
IF there is just one r' such that $e_{r'}^k = 1$ THEN GOTO the *Act Phase*
LET $d = 0$ {Pruning based on distinctiveness }
FOR $j = 1$ TO 2 DO: { Search for rule(s) with the highest distinctiveness value(s) }
FOR rule $r = 1$ TO rule $r = n_r$ DO:
IF $\tilde{e}_r^k = 1$ THEN
IF $d_r < d$ THEN $\tilde{e}_r^k := 0$
ELSE d:=d_r
IF there is just one r' such that $\tilde{e}_{r'}^k = 1$ THEN GOTO the *Act Phase*
LET $p = 0$ {Pruning based on priority }
FOR $j = 1$ TO 2 DO: { Search for rule(s) with the highest priority }
FOR rule $r = 1$ TO rule $r = n_r$ DO:
IF $\tilde{e}_r^k = 1$ THEN
IF $p_r^k < p$ THEN $\tilde{e}_r^k := 0$
ELSE p:=p_r
LET r' be any r such that $\tilde{e}_{r'}^k = 1$ {Pruning based on "arbitrary" }

3. *Act Phase*

Let $e' = \{r', e_{u_k}^{\ell}\}$
Let $(\mathbf{x}_{k+1}^b, \mathbf{x}_{k+1}^i) = f_{e'}^c(x_k^c)$ {Update the knowledge-base state; the state $\mathbf{x}_{k+1}^i$ is defined below}
$\Pi_{k+1}(r') := 1$ {Remove rule r' from the conflict set based on refraction}
FOR rule $r = 1$ to rule $r = n_r$ DO
IF $r \in \Gamma_k$ THEN $s_r^{k+1} := s_r^k + 1$ {Increment the match-
ing age for all rules that were in the conflict set (for recency)}
FOR r=1 TO r=n_r DO
IF $a_{r'r} = 1$ THEN $\Pi_{k+1}(r) := 0$ and $s_r^{k+1} := 0$ {Allow the rules affected
by the firing of rule r' to be considered in the conflict set and reset ages of these rules to 0}

In the step "pruning based on refraction" where it says "STOP" (*i*) the condition can be true since even though the rules are enabled, refraction pruning could reduce the size of the set of enabled rules to zero, and (*ii*) one could change this to "Reset the $\tilde{e}_r^k$ values to the values they had before entering pruning based on refraction and continue" so that the expert system uses the refraction conflict resolution strategy only if it reduces the size of the conflict set. Note that in the act phase $f_{e'}^c(x_k^c)$, where $e' = \{r', e_{u_k}^{\ell}\}$, is the action defined by the consequent formula of rule r' taken on the current knowledge-base state $\mathbf{x}_k^b$ and the action defined for updating the inference engine state $\mathbf{x}_k^i$. In the steps discussed above, the conflict resolution is done based on refraction, recency, and distinctiveness followed by priority (with "arbitrary" making any final decisions if there is more than one rule). In other cases, the conflict resolution strategies may have a different order (the choice of the order being dictated by the application at hand).

To summarize, the operation of the expert system proceeds by:

1. Acquisition of $e^{\ell}_{u_k}$, the process output e_{o_k} and user input events $e^{c}_{d_k}$ at time k,

2. Forming the conflict set Γ_k in the match phase from the set of rules in the knowledge-base and based on $e^{\ell}_{u_k}$, the current status of the truth of various facts, and the current values of variables in the knowledge-base (i.e., $\mathbf{x}^{b}_{k}$),

3. The use of conflict resolution strategies (refraction, recency, distinctiveness, priority, and arbitrary) in the select phase to find one rule $r' \in \Gamma_k$ to fire (this defines $e'_k = \{r', e^{\ell}_{u_k}\}$), and

4. Executing the actions characterized by the consequent of rule r' in the act phase. This involves updating the knowledge-base and inference engine state (i.e., finding x^{c}_{k+1}) and generating the process input and/or conclusions (characterized by $\delta^{c}_{e'_k}(x^{c}_{k})$).

The timing of the event occurrences in the expert system is such that the expert system is synchronous with the process (i.e., if a disturbance or command input event occurs in the process it causes a process output event to occur which will cause a rule to fire) and with the user input (i.e., if a user input event occurs, the expert system will immediately react to it also). Hence, in repsonse to process output and user input events, the expert system fires rules to generate process inputs (sets of enabled command input events). It is important to note that such synchronization is often used in systems and control applications. To maintain such synchronization one senses not only the event values but also the time at which they change. For some processes the switching times of the events are automatically sensed by measuring the event values. For others, special threshold detection and logic circuitry must be employed to obtain the switching times.

2.3 The Reasoning Capabilities of the Expert System

In this Section we will further clarify what class of expert systems we are considering by explaining what types of reasoning they can achieve. The expert system C can *learn* since it can evaluate its own performance (e.g., in terms of what resources it is utilizing), can remember what it has done in the past (in its state), and can modify it future decisions to ensure that it will enhance its future performance. The type of learning possible is, however, not the most general possible since under the current formulation we cannot automatically synthesize an arbitrary number of completely new rules $r \in \mathcal{R}$. This is not a significant limitation on the learning capabilities of the expert system C since:

1. The rule-base $\mathcal{R}$ of C can be partitioned into a finite set of "standard rules" $\mathcal{R}_s$ as they are defined above and a finite set of rules $\mathcal{R}_t$ that act as "templates"[4]. The expert system can use rules $r \in \mathcal{R}_s$ to evaluate its performance and take actions to fill in the meaning of the rule templates $r' \in \mathcal{R}_t$ by changing their premises and consequents. In this way the expert system can, in a structured way, synthesize a finite number of new rules to improve the performance of the system (and this is in fact the way that most current expert learning systems operate).

2. The expert system can use the elements in working memory as parameters in a learning algorithm to adapt, for example, the applicability of subsets of rules, or with simple changes to the inference mechanism, to adapt the priorities and distinctiveness of the rules.

[4]The only reason for requiring that the number of rules in the expert system is finite is to ensure that the process of making an inference step is computable.

We see that the expert system has very general capabilities to learn since it can adapt its rule-base, working memory, and inference mechanism.

Next, note that the expert system C can *plan* since it can predict a finite number of steps into the future what will happen as the result of its actions and it can reformulate what plan should be taken by monitoring the progress of the execution of the current plan. This type of planning is, however, not the most general possible since under the current formulation we cannot plan into the future an arbitrary number of steps. This is not a significant limitation on our expert system C since practical considerations dictate that most often one should only plan ahead a relatively small number of steps (especially for very uncertain environments). Note that to plan into the future we define the knowledge base state $\mathbf{x}^b = [\mathbf{x}^{b'}\ \mathbf{x}^{bp}]^t$ where $\mathbf{x}^{b'}$ is the standard knowledge-base state defined above and $\mathbf{x}^{bp}$ is a vector of state trajectories that are generated by simulating plans under consideration into the future from time k to time $k+N$ (where N is the maximum number of steps we can simulate into the future)[5]. To keep the dimensions of $\mathbf{x}^b$ finite one must require that the expert system only conducts a finite number of simulations into the future; however, all practical planning applications will dictate that only a finite amount of time is used in plan generation so that only a finite number of simulations can be conducted. We see that in addition to general learning capabilities, the expert system C has very general planning capabilities.

3 Properties of Expert Systems

There are extensive studies addressing the analysis of *consistency* and *completeness* properties of knowledge-bases (i.e., the *static properties* - the structure and interconnection of the information in the knowledge-base). In particular, in [16, 17, 18, 19, 20] the authors develop algorithms to check that the knowledge engineering process used to produce the knowledge-base has not produced conflicting rules, redundant rules, circular rules, subsumed rules, etc.; hence, these methods are sometimes referred to as "knowledge-base debugging tools" or methods for "static analysis". Such consistency and completeness characteristics of a knowledge-base will affect the overall behavior of the expert system and in fact the studies in [16, 17, 18, 19, 20] provide the *first step* towards performing a verification of the qualitative properties of the expert system. In this Section we will show that if one were to only analyze the static properties of the knowledge-base, one would not be performing a complete analysis of the dynamics of the full rule-based expert system. In addition, we will explain the importance of reachability, cyclic behavior, and stability properties and show how to characterize and analyze such properties in the mathematical framework of Section 2.

3.1 Static Properties of Knowledge-Bases

In this Subsection we discuss the verification of properties of an *isolated* rule-based expert system. Hence, we assume that there are no user input events and process output events that influence the inference loop ($\mathcal{E}_u^\ell = \emptyset$). We discuss how static properties influence the dynamics of the inference process to illustrate how the analysis approach in [16, 17, 18, 19, 20] ignores several important properties of dynamical rule-based expert system operation. Essentially this requires relating properties of the interconnection of the syntax of the rules to the state trajectories (sequences of states resulting from the firing of rules) representing knowledge and information flow in the rule-base and inference engine.

[5] Note that with these planning capabilities our expert system can in fact perform a significant amount of reasoning at each time step k before it takes actions and the next time step is taken. This can be done by simulating into the future, making an assessment of the best actions to take (rules to fire), setting a flag, and using this flag to enable the best rules to fire at time k

In order to study the effects of static properties of rule-bases on the dynamics of the inference process it is helpful to introduce the notion of *Consequent-Antecedent Compatibility*. A consequent formula is said to be "consequent-antecedent-compatible (CAC) with an antecedent formula at time k" if the actions taken by the consequent formula at time k will result in the antecedent formula being true at time $k+1$ (note that we are using the convention that a rule fires at time k). For example, if C_1 makes a_1 true, i.e., $T_{k+1}(a_1) := 1$, C_2 is defined as $x_{4_{k+1}} := x_{4_k} + 5$ where $x_{4_k} = 2$, P_1 tests $x_4 < 10$, the consequent formula $\Psi = C_1 \wedge C_2$, and the antecedent formula is $\Phi = T(a_1) \wedge P_1$, then Ψ is CAC to Φ at time k. Notice, however, that due to the dynamic behavior of the expert system, Ψ is not necessarily CAC to Φ for all k. Logical truth in the study of the dynamic inference process depends on the state of the expert system. In the above example, if $x_{4_{k'}} = 6$ at time $k' \neq k$, Ψ is *not* CAC to Φ at time k'.

Next, we will clarify the relationships between several consistency and completeness properties of rule-bases and the dynamics of the inference process in expert systems.

3.1.1 Logical Consistency Issues in Rule-Bases

In [16, 17, 18, 19, 20] the authors investigate "Redundant Rules", "Redundant Rule Chains", "Conflicting Rules", "Conflicting Rule Chains", "Subsumed Rules", "Unnecessary IF Conditions", and "Circular Rules" all with the intent of checking whether the knowledge in the rule-base is inconsistent. Their analysis provides for "warnings" about possible inconsistencies but does not take into consideration the effects of the inference engine. Moreover, as we will show next, such static analysis of the *syntax* of the rules can ignore the underlying qualitative properties of the dynamical inference process (especially if user inputs and process outputs are considered).

Redundant Rules and Redundant Rule Chains

Two or more rules which have logically equivalent antecedents at a specific time (the antecedents have the same conditions whose order is not important) and equivalent consequent formulas (same actions taken when the rules fire) are called "redundant rules". A "rule chain" is a sequence of rules which produces a state trajectory. Two or more rule chains which have equivalent antecedents and consequent formulas for each rule are called redundant rule chains. For example, let $\Phi_1 = T(a_1) \wedge P_1 \wedge P_2, \Phi_2 = P_1 \wedge T(a_1) \wedge P_2, \Phi_3 = P_1 \wedge P_2 \wedge T(a_1), \Psi_1 = [T(a_2) := 1] \wedge [x_5 := x_5 + 2]$, and $\Psi_2 = [x_5 := x_5 + 2] \wedge [T(a_2) := 1]$. Then IF Φ_1 THEN Ψ_1, IF Φ_2 THEN Ψ_2 and IF Φ_3 THEN Ψ_1 are redundant rules. The rule chains IF Φ_4 THEN Ψ_4, IF Φ_5 THEN Ψ_5 and IF Φ_6 THEN Ψ_6, IF Φ_7 THEN Ψ_7 are redundant rule chains if Φ_4 and Φ_6, Φ_5 and Φ_7, Ψ_4 and Ψ_6, Ψ_5 and Ψ_7 are equivalent, and Ψ_4 is CAC to Φ_5 and Ψ_6 is CAC to Φ_7.

Redundant rules affect the dynamical behavior of the expert system. Once one of the redundant rules fires, it may be removed from the conflict resolution by refraction. However, the other redundant rule can still be considered in the conflict resolution. The static analysis as in [16, 17, 18, 19, 20] can be used to detect, then remove such rules if needed.

Conflicting Rules

Two or more rules which have logically equivalent antecedents at a specific time but their consequent formulas have at least one component that results in contradictory logical value or inconsistent actions upon a variable when the rules fire are called "conflicting rules". "Conflicting rule chains" occur when two or more rule chains have logically equivalent antecedents for each rule but the firing actions of the chains cause at least one inconsistency in at least one variable in the state sequence. For example, let $\Phi_1 = T(a_1) \wedge P_1, \Phi_2 = P_1 \wedge T(a_1), \Psi_1 = [T(a_2) := 1] \wedge [x_2 := x_1 + 1]$, and $\Psi_2 = [T(a_2) := 0] \wedge [x_2 := x_1 + 1]$; then IF Φ_1 THEN Ψ_1, and IF Φ_2 THEN Ψ_2 are conflicting rules. The rule

chains IF Φ_4 THEN Ψ_4, IF Φ_1 THEN Ψ_1 and IF Φ_4 THEN Ψ_5, IF Φ_2 THEN Ψ_2 are conflicting rule chains if Ψ_4 is equivalent to Ψ_5, Ψ_4 is CAC to Φ_1, and Ψ_5 is CAC to Φ_2.

For the model C, conflicting rules are allowed as they simply characterize the possibility of a diversity of reasoning approaches. If one is concerned about the presence of conflicting rules/conflicting rule chains, the static analysis in [16, 17, 18, 19, 20] can be used to detect their presence; however, this type of analysis can flag some rule chains as "conflicting" where they really merely represent the different possible ways of reasoning about the same problem.

Subsumed Rules

Two or more rules which have equivalent consequent formulas but with one which has more restricted antecedent conditions than the others are called "subsumed rules". In other words, the truth value of a rule's antecedent at a specific time implies the truth of the ones of the other rules. For example, let $\Phi_1 = T(a_1) \wedge P_1 \wedge T(a_2), \Phi_2 = T(a_1) \wedge P_1, \Phi_3 = T(a_1) \wedge T(a_2)$, IF Φ_1 THEN Ψ_1 (rule 1), IF Φ_2 THEN Ψ_1 (rule 2), and IF Φ_3 THEN Ψ_1 (rule 3). Hence rule 1 has more restricted antecedent conditions than rules 2 and 3; hence, rule 1 is logically subsumed by rule 2 and 3.

Subsumed rules affect the dynamic behavior of the system since the firing of a rule depends on distinctiveness of the antecedent. In terms of the model C, erasing the more restricted rules from the knowledge-base will affect the selection of which rule to fire. For example, if there is another rule (rule 4) which has three conditions in the example above, the inference mechanism will select rule 1 and 4 after pruning the enabled rules using distinctiveness strategy. Then it will use the next conflict resolution strategy to select which rule to fire. However, if rule 1 is erased, the inference mechanism will only have rule 4 to fire after the distinctiveness pruning, so erasing the more restrictive rule changes the conflict resolution pruning of certain enabled rules which will produce different results. Hence, static analysis may recommend removing a rule but for the study of the dynamics of the inference process such rules may be needed for inference control.

Unnecessary IF Conditions

Two or more rules with the same consequent formulas but with at least one condition of their antecedents in complement with one of the other rule's are called "unnecessary IF conditions". For example, if

- Rule 1 : IF $T(a_1) \wedge T(a_2)$ THEN Ψ_1
- Rule 2 : IF $T(a_1) \wedge \neg T(a_2)$ THEN Ψ_1
- Rule 3 : IF $T(a_1) \wedge [x_4 \leq 4]$ THEN Ψ_2
- Rule 4 : IF $T(a_1) \wedge [x_4 > 4]$ THEN Ψ_2

then Rules 1 and 2 have unnecessary IF conditions as do Rules 3 and 4.

Unnecessary IF conditions may affect the dynamic behavior of the system in terms of selecting which rule to fire. Eliminating the unnecessary conditions of the rules changes their distinctiveness; hence, such modifications must be done in such a way so that conflict resolution in the inference engine leads to a desirable dynamic behavior (in certain cases, the unnecessary IF conditions can be kept if they are important to achieve proper inference).

Circular Rules

"Circular rules" can occur if there is a set of rules which has CAC properties. Such a circular chain of rules creates circular chain of states in terms of model C. For example, IF Φ_1 THEN Ψ_1, IF Φ_2 THEN Ψ_2 and IF Φ_3

THEN Ψ_3 are circular rules if Ψ_1 is CAC to Φ_2, Ψ_2 is CAC to Φ_3 and Ψ_3 is CAC to Φ_1. Circular rules affect the dynamic behavior of the system and may lead to a circular reasoning (but not necessarily so, since the inference mechanism may be able to reason around it) which is not desirable in most cases. Circular rules create state cycles, but the existence of state cycles does not always imply that all rules forming state cycles are circular rule chains as defined in [20]. Here is an example to illustrates that behavior:

- Rule 1 : IF $T(a_1)$ THEN $T(a_3) := 1$
- Rule 2 : IF $T(a_3)$ THEN $T(a_4) := 1$
- Rule 3 : IF $T(a_4) \wedge \neg T(a_2)$ THEN $[T(a_3) := 0] \wedge [T(a_4) := 0]$

Notice that those rules are not necessarily all CAC so that they do not form a circular rule chain as defined in [16, 17, 18, 19, 20], yet they form a circular sequence of states. Let the state $\mathbf{x}^b = [T(a_1)\ T(a_2)\ T(a_3)\ T(a_4)]^t$, then we get the following sequence of states:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\text{Rule1}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{\text{Rule2}} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} \xrightarrow{\text{Rule3}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

This shows that static analysis as in [16, 17, 18, 19, 20] will <u>not</u> always detect circular reasoning; this motivates the importance of performing analysis of the dynamic behavior of the full expert system.

3.1.2 Logical Completeness Issues in Rule-Bases

In [16, 17, 18, 19, 20] the authors investigate "Unreferenced Antecedent Conditions", "Illegal Antecedent Conditions", "Unreachable Conclusions", and "Deadend Goal and Deadend State" all with the intent of checking whether or not there is enough information in the knowledge-base connected in the proper fashion to ensure that from the initial knowledge, a goal state can always be reached. Their work provides for warnings that the goal states may not be reachable but does not provide for a complete analysis of the reachability of the goal states when the inference engine is added (i.e., their analysis is only on the knowledge-base and not on the complete inference loop which includes the inference mechanism).

<u>Unreferenced Antecedent Conditions</u>

A state where there is no enabled rule is referred to as an "Unreferenced Antecedent Condition" state. This is similar to the notion of "Unreferenced Attribute Values" in [16, 17, 18, 19, 20]. In static properties, this implies some values in the set of possible values of an object's attribute are not covered by any rule's IF conditions [20]. In terms of the model C, this situation may lead to a state/states where there is no rule whose antecedent conditions match the information in that/those state(s). This affects the dynamic behavior represented by the model C, since there will be no rule to fire which leads to another state. In computer science this state is often called a *dead-lock state*; essentially we would say that the expert system is not properly defined so that it can react to all possible situations. A dead-lock state is undesirable in most cases, since it may affect the reachability of a certain set of goal states. However, dead-lock states may not cause problems if they are goal states.

Illegal Antecedent Conditions

An illegal antecedent condition occurs when a rule is never enabled in any state, because its antecedent conditions are never all true. This type of rule merely wastes memory in the knowledge-base so one may want to use static analysis to detect and remove such rules. From a dynamical systems perspective it may be hard to verify that a rule will never be enabled since in this case one must also consider the unpredictable behavior of the user and dynamic process and all possible states that the expert system can enter.

Unreachable Conclusion

In static analysis, a conclusion of a rule should either match a goal or match an IF condition of another rule in order to guarantee the goal is reachable [20]. In terms of model C, firing a rule must lead to a state where there must be at least one enabled rule; otherwise, the last rule fired causes dead-lock. Note, however, that two consecutive rules do not have to be CAC. For example, if we only have two rules:

- Rule 1 : IF $T(a_1)$ THEN $T(a_2) := 1$
- Rule 2 : IF $T(a_2) \wedge \neg T(a_3)$ THEN $[T(a_2) := 0] \wedge [T(a_3) := 1]$

then the state sequence may be of the form:

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{\text{Rule1}} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{\text{Rule2}} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Note that the conclusion of the second rule is reachable from $[1\ 0\ 0]^t$ even though the consequent formula of rule 1 is not necessarily CAC to the antecedent of the second rule. If static analysis says that the consequent of rule 1 is CAC with rule 2 then $[1\ 0\ 1]^t$ would appear to be reachable; however, if initially we start at $[1\ 1\ 1]^t$, the state $[1\ 0\ 1]^t$ is not reachable. If static analysis says that the consequent of rule 1 is not CAC with the antecendent of rule 2 then if we start at $[1\ 0\ 0]^t$ it will also appear that $[1\ 0\ 1]^t$ is not reachable. This shows that static analysis as in [16, 17, 18, 19, 20] is insufficient for analyzing reachability of conclusions (especially when one also considers the dynamics of the inference mechanism, the user inputs, and the inputs from a dynamic process).

Dead-end Goal and Dead-end State

Syntactically, to achieve a goal, it is required that the goal is matched by a conclusion of at least one of the rules; otherwise, the goal cannot be achieved and is referred to as "dead-end goal". Similarly, the IF conditions of a rule must meet this requirement; otherwise, it is a "dead-end IF" condition [20]. Notice that this is only an *ad-hoc* test to determine if a goal state is reachable from the initial state.

In terms of the model C, reachability of a state is determined by the presence of an inference path corresponding to a sequence of states from the initial state to the goal state. If not all the inference trajectories originating from the initial state end up in the set of goal states, then it is possible that the expert system may succeed, but not guaranteed. For analysis of qualitative properties we need to show that all paths from the initial state reach a goal state and thereby perform a complete reachability analysis to verify whether the goals can be obtained.

To summarize, in developing a rule-base expert system it is important to study the static structure of the rules and their interconnections. Debugging Programs can help in structuring and eliminating some unnecessary rules and

in detecting certain consistency and completeness problems with the knowledge-base. However, the analysis of the dynamics of the inference process still needs to be performed since the static analysis cannot detect/predict some of the properties of the dynamical expert system. The key properties/issues that are ignored or treated inadequately by the analysis of knowledge-bases in [16, 17, 18, 19, 20] are:

- the presence of the inference engine,
- user inputs and information inputs from a dynamical process,
- circular reasoning (a logical inconsistency),
- reachability (logical completeness property), and hence
- stability (to be defined in the next Section).

In the next Section we show how to characterize and analyze qualitative properties (i.e., what some people would call "dynamic properties") of the full expert system that interfaces to a user and a dynamic process.

3.2 Characterization and Analysis of Qualitative Properties of Expert Systems

In this Section we characterize, and introduce methods to analyze, three different types of behavior that expert systems are often designed to achieve.

3.2.1 Reachability Properties

The results in [13] showed the relationship between performing chains of inference and reachability. In particular, the authors define reachability in the context of inference processes as the ability to fire a sequence of rules to derive a specific conclusion from some specific initial knowledge. In system-theoretic terms this is a standard definition for reachability that one might call a "state-to-state" property. Here we consider a slightly more general reachability property for studying inference processes in expert systems. For $\mathcal{X}_m \subset \mathcal{X}^c$, let $\mathcal{X}(C, x_0^c, \mathcal{X}_m)$ denote the set of all finite length state trajectories of C that begin at x_0^c and end in $\mathcal{X}_m$.

Definition 3.1 *A system C is said to be "$(x_0^c, \mathcal{X}_m)$ – reachable" if there exists a sequence of events to occur that produces a state trajectory $s \in \mathcal{X}(C, x_0^c, \mathcal{X}_m)$.*

Note that $\mathcal{X}_m$ can represent the desired operating conditions (goals) of the expert system with x_0^c as its initial state. Hence, we will consider what could be called a "point-to-set" reachability problem for expert systems. This general type of reachability is needed when it is possible that there are *several* valid states that can be reached from one initial state (or in the situation where it is known that at least one state in a set of states $\mathcal{X}_m$ is reachable). To automate testing of the property in Definition 3.1 we use a shortest path algorithm to find the state trajectory $s \in \mathcal{X}(C, x_0^c, \mathcal{X}_m)$ when it exists (we will assign a cost of one to firing a rule, i.e., the occurrence of an event). Note that while the use of the shortest path algorithm on a metric space (to be defined below) offers several advantages with regard to computational complexity so that exhaustive search is not necessary [27], unless the state-space is finite we will not be able to conclude anything about *unreachability* (i.e., we cannot easily determine that the system does not possess a specified reachability property unless the state-space is finite). Moreover, we note that if the state-space is finite the complexity of testing the reachability property is $O(n^2)$, i.e., polynomial.

3.2.2 Cyclic Properties

In the verification of the qualitative properties of the expert system, the study of cyclic behavior is of paramount importance. This is due to the fact that if cycles exist, the expert system could get "trapped" in a *circular argument* so that there is no way it can achieve its ultimate task. This cyclic characteristic will be particularly problematic for expert systems that operate in time-critical environments (e.g., in a failure diagnosis problem). Let $\mathcal{X}_y \subset \mathcal{X}^c$ denote a subset of the states such that each $x_y \in \mathcal{X}_y$ lies on a cycle of states that is in $\mathcal{X}_y$.

Definition 3.2 *A system C is said to be "$(x_0^c, \mathcal{X}_y)-$cyclic" if there exists a sequence of events to occur that produces a state trajectory $s \in \mathcal{X}(C, x_0^c, \mathcal{X}_y)$.*

It is a hard problem to detect the presence of cyclic behavior, since one may not be able to find $\mathcal{X}_y$ without studying all system trajectories. To help automate the testing of the property in Definition 3.2 we can use a two step approach. First we specify a set $\mathcal{X}_y$ (which can be found with a search algorithm described in [28] if the state-space is finite), then we use a search algorithm to find the inference path that starts at x_0^c and ends in $\mathcal{X}_y$ (if one exists) [28]. Note that if $\mathcal{X}_y$ is the null set then we have determined that the system has no cycles while if $\mathcal{X}_y$ is not the null set then we know it has cycles; hence for some classes of systems we can explicitly test if they are cyclic or not. Note that the complexity of finding $\mathcal{X}_y$ is polynomial if the state-space is finite so that the overall complexity of testing for cyclic properties is polynomial in terms of the size of the state-space. In our applications in Sections 4 and 5 we will actually verify that the expert system does not contain undesirable cycles by verifying certain stability properties to be defined next (and thereby avoid problems with computational complexity).

3.2.3 Stability Properties

In terms of characterizing human cognitive functions, *Lyapunov stability* [29, 30, 23, 31] for the expert system can be viewed as a mathematical characterization of an expert system's ability to concentrate (i.e., to focus, to pay attention) on the task at hand. Clearly then verification of stability is critical since without stability the expert system can, for example, wander aimlessly not achieving the goals that it is supposed to achieve. From an engineering or scientific standpoint, rather than pyschological standpoint, stability of an expert system is of fundamental importance due to the fact that guarantees of stability often ensure that the system variables will stay in *safe* operating regions (e.g., variables in working memory stay bounded) and that other performance objectives (e.g., reachability or optimal use of resources) can be met. Below, we briefly overview some recent results in Lyapunov stability analysis [29, 30] that apply to the model used here for the expert system.

Let $\rho : \mathcal{X}^c \times \mathcal{X}^c \rightarrow \Re$ denote a *metric* on $\mathcal{X}^c$, and $\{\mathcal{X}^c; \rho\}$ a *metric space*. Denote the *distance* from point x to the set $\mathcal{X}_z$ by $\rho(x, \mathcal{X}_z) = \inf\{\rho(x, x') : x' \in \mathcal{X}_z\}$ where $\mathcal{X}_z \subset \mathcal{X}^c$. The "r-neighborhood" of an arbitrary set $\mathcal{X}_z \subset \mathcal{X}^c$ is denoted by the set $S(\mathcal{X}_z; r) = \{x \in \mathcal{X}^c : 0 < \rho(x, \mathcal{X}_z) < r\}$ where $r > 0$. Define $\mathbf{E}_v^c(x_0^c)$ to be the finite and infinite length physically possible event trajectories of C which start at x_0^c and let $X(x_0^c, E_k, k)$ be the state of C reached from x_0^c after the occurrence of event sequence $E_k = e_0 e_1 ... e_{k-1}$. The set $\mathcal{X}_m \subset \mathcal{X}^c$ is called "invariant with respect to (w.r.t.) C" if from $x_0^c \in \mathcal{X}_m$ it follows that $X(x_0^c, E_k, k) \in \mathcal{X}_m$ for all E_k such that $E_k E \in \mathbf{E}_v^c(x_0^c)$ and $k \geq 0$.

Definition 3.3 *An invariant set $\mathcal{X}_m \subset \mathcal{X}^c$ of C is called "stable in the sense of Lyapunov w.r.t. $\mathbf{E}_v^c$" if for any $\epsilon > 0$ it is possible to find a quantity $\delta > 0$ such that when $\rho(x_0^c, \mathcal{X}_m) < \delta$ we have $\rho(X(x_0^c, E_k, k), \mathcal{X}_m) < \epsilon$ for all E_k such that $E_k E \in \mathbf{E}_v^c(x_0^c)$ and $k \geq 0$. If furthermore, $\rho(X(x_0^c, E_k, k), \mathcal{X}_m) \rightarrow 0$ for all E_k such that $E_k E \in \mathbf{E}_v^c(x_0^c)$ as $k \rightarrow \infty$, then the invariant set $\mathcal{X}_m$ of C is called "asymptotically stable w.r.t. $\mathbf{E}_v^c$".*

Definition 3.4 *If the invariant set $\mathcal{X}_m \subset \mathcal{X}^c$ of C is asymptotically stable in the sense of Lyapunov w.r.t. $\mathbf{E}_v^c$, then the set $\mathcal{X}_v$ of all states $x_0^c \in \mathcal{X}^c$ having the property $\rho(X(x_0^c, E_k, k), \mathcal{X}_m) \to 0$ for all E_k such that $E_k E \in \mathbf{E}_v^c(x_0^c)$ as $k \to \infty$ is called the "region of asymptotic stability of $\mathcal{X}_m$ w.r.t. $\mathbf{E}_v$".*

Definition 3.5 *The invariant set $\mathcal{X}_m \subset \mathcal{X}^c$ of C with region of asymptotic stability $\mathcal{X}_v$ w.r.t. $\mathbf{E}_v^c$ is called "asymptotically stable in the large w.r.t. $\mathbf{E}_v^c$" if $\mathcal{X}_v = \mathcal{X}^c$.*

Definition 3.6 *The motions $X(x_0^c, E_k, k)$ of C which begin at $x_0^c \in \mathcal{X}^c$ are bounded w.r.t $\boldsymbol{E}_v^c$ and the bounded set $\mathcal{X}_b \subset \mathcal{X}^c$ if there exists a $\beta > 0$ such that $\rho(X(x_0^c, E_k, k), \mathcal{X}_b) < \beta$ for all E_k such that $E_k E \in \boldsymbol{E}_v^c(x_0^c)$ and for all $k \geq 0$. C is said to possess Lagrange Stability w.r.t. $\boldsymbol{E}_v^c$ and the bounded set $\mathcal{X}_b \subset \mathcal{X}^c$ if for each $x_0^c \in \mathcal{X}^c$ the motions $X(x_0^c, E_k, k)$ for all E_k such that $E_k E \in \boldsymbol{E}_v^c(x_0^c)$ and all $k \geq 0$ are bounded w.r.t. $\boldsymbol{E}_v^c$ and $\mathcal{X}_b$.*

The following Theorems provide the necessary and sufficient conditions for the analysis of any system represented via C (the proofs are contained in [29, 30]).

Theorem 3.1 *In order for an invariant set $\mathcal{X}_m \subset \mathcal{X}^c$ of C to be stable in the sense of Lyapunov w.r.t. $\mathbf{E}_v^c$ it is necessary and sufficient that in a sufficiently small neighborhood $S(\mathcal{X}_m; r)$ of the set $\mathcal{X}_m$ there exists a specified functional V with the following properties: (1) For all sufficiently small $c_1 > 0$, it is possible to find a $c_2 > 0$ such that $V(x) > c_2$ for $x \in S(\mathcal{X}_m; r)$ and $\rho(x, \mathcal{X}_m) > c_1$, (2) For all $c_4 > 0$ as small as desired, it is possible to find a $c_3 > 0$ so small that when $\rho(x, \mathcal{X}_m) < c_3$ for $x \in S(\mathcal{X}_m; r)$ we have $V(x) \leq c_4$, and (3) $V(X(x_0^c, E_k, k))$ is a non-increasing function for $k \geq 0$, for $x_0^c \in S(\mathcal{X}_m; r)$, for all $k \geq 0$, as long as $X(x_0^c, E_k, k) \in S(\mathcal{X}_m; r)$ for all E_k such that $E_k E \in \mathbf{E}_v^c(x_0^c)$.*

Theorem 3.2 *In order for an invariant set $\mathcal{X}_m \subset \mathcal{X}^c$ of C to be asymptotically stable in the sense of Lyapunov w.r.t. $\mathbf{E}_v$ it is necessary and sufficient that in a sufficiently small neighborhood $S(\mathcal{X}_m; r)$ of the set $\mathcal{X}_m$ there exists a specified functional V having properties 1, 2 and 3 of Theorem 3.1 and furthermore $V(X(x_0^c, E_k, k)) \to 0$ as $k \to \infty$ for all E_k such that $E_k E \in \mathbf{E}_v^c(x_0^c)$ for all $k \geq 0$ as long as $X(x_0^c, E_k, k) \in S(\mathcal{X}_m; r)$.*

An important advantage of the Lyapunov approach in the study of stability properties is that it is often possible to intuitively define an appropriate Lyapunov function V (years of use have shown this - see the extensive literature in the area of nonlinear analysis) and we will illustrate how this is done in the examples. However, specifying the Lyapunov function is sometimes problematic for certain applications. Motivated by the difficulties in specifying a Lyapunov function, we next discuss how one can sometimes use search algorithms to study stability properties. The study of asymptotic stability in the large or of regions of asymptotic stability $\mathcal{X}_v$ using search methods involves finding the invariant set $\mathcal{X}_m$ and showing that all paths which originate from any state in $\mathcal{X}_v$ will end up in $\mathcal{X}_m$ (one must be careful with the imposition of the constraints specified by $\mathbf{E}_v^c$ when using a search algorithm). The complexity of showing that a particular system is asymptotically stable in the large for a finite state-space is polynomial in terms of the size of the state-space; however, for real-world problems using the algorithmic approach can be computationally prohibitive. It is for this reason that we will rely on (i) the choice of an appropriate Lyapunov function and an analytical proof when such a function is easy to define, and (ii) the use of an algorithmic approach when the definition of the Lyapunov function is not evident. In fact, in Section 4 we will use the search algorithm approach to stability analysis, while in Section 5 we will choose an appropriate Lyapunov function and prove that an expert system possesses certain stability properties.

Finally, it is important to note that while we are able to characterize and analyze more general properties than the static properties examined in the past (see Section 3.1), if we use an algorithmic approach to the verification of the properties, the complexity of verification of the qualitative properties discussed above is generally higher than that of the static properties. For instance, the complexity of studying most static properties is bounded by the number of rules where the complexity of testing each of the qualitative properties (reachability, cyclic behavior, and stability) is bounded in terms of the size of the state-space. Since it is often the case that there will be significantly more states than rules verification of our qualitative properties will generally be more complex than that of the static properties. It is for this reason that the Lyapunov approach to verification of stability properties is so important. The problem of computational complexity can be completely avoided if one can find an appropriate Lyapunov function and show that it satisfies certain properties listed above. We see that as with the nonlinear analysis of more conventional dynamical systems one of the primary advantages of the Lyapunov approach lies in the lack of dependence on explicitly enumerating all possible system trajectories in the study of stability properties (i.e., the Lyapunov approach allows us to prove stability properties without running the rule-based system for all possible scenarios - which can be particularly problematic for expert systems that interface to a dynamic and uncertain environment).

4 Water-Jug Example

In this Section we study reachability and stability properties of a rule-based expert system that solves a water-jug filling problem. It is given that there is a 4-gallon jug and a 3-gallon jug named "jug1" and "jug2", respectively. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. The goal is to get exactly 2 gallons of water into the 4-gallon jug and 3 gallons into the 3-gallon jug. In some situations we can dump the water out of the jugs. Let *jug1* and *jug2* denote the number of gallons of water in the jugs. The operations that can be performed are constrained as follows:

1. **Fill the 4-gallon jug.** After this operation, *jug1=4* and *jug2* remains the same. This operation is not applicable if the 4-gallon jug is already full.
2. **Fill the 3-gallon jug.** After this operation, *jug2=3* and *jug1* remains the same. This operation is not applicable if the 3-gallon jug is already full.
3. **Dump all the water out of the 4-gallon jug.** After this operation, *jug1=0* and *jug2* remains the same. This operation is not applicable if jug1 is already empty.
4. **Dump all the water out of the 3-gallon jug.** After this operation, *jug2=0* and *jug1* remains the same. This operation is not applicable if jug2 is already empty.
5. **Move water from the 4-gallon jug to the 3-gallon jug until either the 4-gallon jug is empty or the 3-gallon jug is full.** This operation is not applicable if jug1 is empty or jug2 is full.
6. **Move water from the 3-gallon jug to the 4-gallon jug until either the 3-gallon jug is empty or the 4-gallon jug is full.** This operation is not applicable if jug2 is empty or jug1 is full.

Next, we specify the model C for an expert system that solves this problem. As there are no inputs or outputs for the expert system we have $\mathcal{E}^c = \emptyset$. The knowledge-base has $\mathbf{x}^b = [x_1\ x_2\ x_3\ ...\ x_{42}]^t$ (with $m = 42$, $n = 0$) where

x_1 and x_2 represent the contents of jug1 and jug2, respectively, and x_3 through x_{42} represent the states x_1 and x_2 (liquid levels of *jug1* and *jug2*) that have already been visited. The components x_3 and x_4 hold the initial conditions of x_1 and x_2, respectively, the components x_5 and x_6 hold the next x_1 and x_2 which have been visited and so on. At most 20 states will be visited before the goal is reached for any path from the initial to the goal state. This corresponds to all combinations of x_1 and x_2 where x_1 can take a value among 0, 1, 2, 3, 4 and x_2 among 0, 1, 2, 3. Initially, x_5 through x_{42} are initialized to 10 (a value which never be reached) and $x_3 = x_1, x_4 = x_2$. The history of state components x_1 and x_2 which have already been visited is included in the knowledge-base state $\mathbf{x}^b$ to avoid circular reasoning. Hence, there is an inherent mechanism imbedded in the knowledge-base for avoiding circular reasoning.

The premise functions are defined as:

P_1 tests "$0 \leq x_1 < 4$",
P_2 tests "$0 \leq x_2 < 3$",
P_3 tests "$x_1 > 0$",
P_4 tests "$x_2 > 0$",
P_5 tests "$x_1 > 3 - x_2$",
P_6 tests "$x_1 \leq 3 - x_2$",
P_7 tests "$x_2 > 4 - x_1$",
P_8 tests "$x_2 \leq 4 - x_1$",
P_9 tests "$x_1 < 4$",
P_{10} tests "$x_2 < 3$",
P_{11} tests "$[(x_3 = 4) \wedge (x_4 = x_2)] \vee [(x_5 = 4) \wedge (x_6 = x_2)] \vee \ ... \ \vee [(x_{41} = 4) \wedge (x_{42} = x_2)]$",
P_{12} tests "$[(x_3 = x_1) \wedge (x_4 = 3)] \vee [(x_5 = x_1) \wedge (x_6 = 3)] \vee \ ... \ \vee [(x_{41} = x_1) \wedge (x_{42} = 3)]$",
P_{13} tests "$[(x_3 = 0) \wedge (x_4 = x_2)] \vee [(x_5 = 0) \wedge (x_6 = x_2)] \vee \ ... \ \vee [(x_{41} = 0) \wedge (x_{42} = x_2)]$",
P_{14} tests "$[(x_3 = x_1) \wedge (x_4 = 0)] \vee [(x_5 = x_1) \wedge (x_6 = 0)] \vee \ ... \ \vee [(x_{41} = x_1) \wedge (x_{42} = 0)]$",
P_{15} tests "$[(x_3 = (x_1 - (3 - x_2))) \wedge (x_4 = 3)] \vee [(x_5 = (x_1 - (3 - x_2))) \wedge (x_6 = 3)]$
$\vee \ ... \ \vee [(x_{41} = (x_1 - (3 - x_2))) \wedge (x_{42} = 3)]$",
P_{16} tests "$[(x_3 = 0) \wedge (x_4 = (x_1 + x_2))] \vee [(x_5 = 0) \wedge (x_6 = (x_1 + x_2))] \vee \ ... \vee$
$[(x_{41} = 0) \wedge (x_{42} = (x_1 + x_2))]$",
P_{17} tests "$[(x_3 = 4) \wedge (x_4 = (x_2 - (4 - x_1)))] \vee [(x_5 = 4) \wedge (x_6 = (x_2 - (4 - x_1)))]$
$\vee \ ... \ \vee [(x_{41} = 4) \wedge (x_{42} = (x_2 - (4 - x_1)))]$",
P_{18} tests "$[(x_3 = (x_1 + x_2)) \wedge (x_4 = 0)] \vee [(x_5 = (x_1 + x_2)) \wedge (x_6 = 0)] \vee \ ... \vee$
$[(x_{41} = (x_1 + x_2)) \wedge (x_{42} = 0)]$",
P_{19} tests "$x_1 = 3$",
P_{20} tests "$x_2 = 3$",
P_{21} tests "$x_3 = 3$",
P_{22} tests "$x_4 = 3$", and
P_{23} tests "$x_1 = 2$".

Note that P_{11} through P_{18} check if the next state components of x_1 and x_2 have been visited. The consequent formula C_1 is defined to insert the current x_1 and x_2 to the proper spots in x_5 through x_{42}.

The rules $r \in \mathcal{R}$ are:

Rule 1: IF $P_1 \wedge \neg P_{11}$ THEN $(x_1 := 4) \wedge C_1$
Rule 2: IF $P_2 \wedge \neg P_{12}$ THEN $(x_2 := 3) \wedge C_1$
Rule 3: IF $P_3 \wedge \neg P_{13}$ THEN $(x_1 := 0) \wedge C_1$
Rule 4: IF $P_4 \wedge \neg P_{14}$ THEN $(x_2 := 0) \wedge C_1$
Rule 5: IF $P_5 \wedge P_3 \wedge P_{10} \wedge \neg P_{15}$ THEN $(x_1 := x_1 - (3 - x_2)) \wedge (x_2 := 3) \wedge C_1$
Rule 6: IF $P_6 \wedge P_3 \wedge P_{10} \wedge \neg P_{16}$ THEN $(x_2 := x_1 + x_2) \wedge (x_1 := 0) \wedge C_1$
Rule 7: IF $P_7 \wedge P_4 \wedge P_9 \wedge \neg P_{17}$ THEN $(x_2 := x_2 - (4 - x_1)) \wedge (x_1 := 4) \wedge C_1$
Rule 8: IF $P_8 \wedge P_4 \wedge P_9 \wedge \neg P_{18}$ THEN $(x_1 := x_1 + x_2) \wedge (x_2 := 0) \wedge C_1$
Rule 9: IF $P_{19} \wedge P_{20} \wedge P_{21} \wedge P_{22}$ THEN $(x_2 := 0) \wedge C_1$
Rule 10: IF $P_{23} \wedge P_{20} \wedge P_1 \wedge P_3 \wedge P_4$ THEN $(x_1 := x_1) \wedge (x_2 := x_2) \wedge C_1$

The valid inference trajectories $\mathbf{E}_v^c$ consist of all possible trajectories which can be produced by rule-base and the conflict resolution strategy.

We use all the conflict resolution strategies introduced in Section 3. Therefore, the expert system state $x^c = (\mathbf{x}^b, [\mathbf{p}^t\ \mathbf{s}^t]^t)$ where $\mathbf{p}$ and $\mathbf{s}$ are 10×1 vectors. Since every time a rule fires the state changes and all the antecedents check all components of the state the matrix $\mathbf{A}$ is $\mathbf{1}_{10\times 10}$ where $\mathbf{1}_{10\times 10}$ is a 10-by-10 matrix of ones for this problem. This means that the refraction and the recency do not prune rules from the conflict set so that the expert system for this problem can be reduced such that there is no state associated with the inference engine. However, for completeness of the design illustration, $\mathbf{p}$ and $\mathbf{s}$ are included as the state of the inference engine. The distinctiveness level of each rule is dependent upon the conditions of its antecedent. The assigned priority to the rules is: $p_1 = 2, p_2 = 1, p_3 = 3, p_4 = 4, p_5 = 5, p_6 = 5, p_7 = 1, p_8 = 1, p_9 = 6$ and $p_{10} = 6$. A shortest path algorithm with the cost of firing a rule set to be 1 is used to find all possible paths from any initial state.

From the system definition, the initial knowledge-base states are $\mathbf{x}_0^b \in \{\mathbf{x}^b \in \mathcal{X}^b : x_3 = x_1, x_4 = x_2, x_i = 10 \text{ for } 5 \leq i \leq 42,\ x_1 \in \{0,1,2,3,4\}, x_2 \in \{0,1,2,3\}\}$. Furthermore, the initial state of the inference engine must be such that the vectors $\mathbf{p} = \mathbf{s} = 0$. Thus, the initial expert system states $\mathbf{x}_0^c \in \mathcal{X}_0^c$ where

$$\mathcal{X}_0^c = \{x^c \in \mathcal{X}^c : x_3 = x_1, x_4 = x_2, x_i = 10 \text{ for } 5 \leq i \leq 42, x_1 \in \{0,1,2,3,4\}, x_2 \in \{0,1,2,3\}, \mathbf{p} = \mathbf{s} = 0\}. \quad (8)$$

The invariant set (set of goal states) for the water jug is defined to be

$$\mathcal{X}_m = \{x^c \in \mathcal{X}^c, x_1 = 2, x_3 = 3\}. \quad (9)$$

The results of using our search algorithm show that for any given initial water level in jug1 and jug2, the goal (jug1 and jug2 contain 2 and 3 gallons of water, respectively) can be achieved. Once jug1 and jug2 have 2 and 3 gallons of water, respectively, the levels are maintained by firing rule 10. The results of our analysis can be stated formally in the following Theorem.

Theorem 4.1 *The above water jug problem is $(x_0^c, \mathcal{X}_m)$ - reachable for all $x_0^c \in \mathcal{X}_0^c$ since there exists a sequence of events to occur that produces a state trajectory $s \in \mathcal{X}(C, x_0^c, \mathcal{X}_m)$ for any $x_0^c \in \mathcal{X}_0^c$.*

We have, however, not shown that the expert system will actually achieve the goal. We have just shown that there exist inference sequences that the expert system will follow that will succeed. Up till now our analysis does not *guarantee* that the expert system will definitely succeed (i.e., it may follow an improper inference sequence). Next, we show that it will always succeed by verifying certain stability properties. For the water jug problem, the distance

between a point $x^c \in \mathcal{X}^c$ to another point $x^{c'} \in \mathcal{X}^c$ is defined to be

$$\rho(x^c, x^{c'}) = \max\left\{ \max_{i=1,2,\ldots,42}\{|x_i^b - x_i^{b'}|\}, \max_{j=1,2,\ldots,20}\{|x_j^i - x_j^{i'}|\} \right\}. \tag{10}$$

Using a search algorithm we have verified the following result:

Theorem 4.2 *For the above water jug problem, the region of asymptotic stability of $\mathcal{X}_m$ w.r.t. $\mathbf{E}_v^c$ is $\mathcal{X}_v = \mathcal{X}_0^c$.*

Theorem 4.2 shows that no matter what the initial liquid levels are in the jugs, our expert system can reason appropriately so that it can succeed in achieving the goal. Notice that this stability result inherently depends on the reachability result in Theorem 4.1, and that the validity of Theorem 4.2 guarantees that the expert system will exhibit no circular reasoning outside the invariant set (i.e., outside the goal set).

5 Process Control Example

An expert system that is used to perform control activities is called an "expert controller" and the system that results from connecting an expert controller to a dynamical process is called an "expert control system" (ECS) [32, 33]. Application of the mathematical framework in this paper to the verification of the qualitative properties of ECS can be found in [23, 31]. In this Section we study the expert control of a simple dynamic process called a "surge tank"; however, unlike the work in [23, 31] we focus here on the verification of the qualitative properties of an *isolated* expert controller that interfaces to *both* user inputs and the outputs of the dynamical process.

The surge tank, which is shown in Figure 2, holds liquid and has a sensor which provides a measurement $h(k)$ of the amount of liquid in the tank at uniformly spaced discrete times. The resolution of the sensor is such that it can distinguish between seven different liquid levels, i.e., $h(k) \in \{1, 2, 3, 4, 5, 6, 7\}$. Valve A is used for filling the tank, while Valve B is used for emptying the tank. It is known that if Valves A and B are turned on (off) at time k, they stay on (off) till time $k + 1$ (where their status may change again). Valve B is controlled by *customers* that siphon off various amounts of liquid in an unpredictable fashion. It is known, however, that our customers will persistently open the empty valve (i.e., there exists a finite amount of time between the times that the empty valve is opened). Valve A has a greater flow capacity than Valve B so that even if a customer turns on Valve B for all time, if Valve A is on the liquid level in the tank will rise. The expert system will use the measurement $h(k)$ of the liquid level in the tank, together with the user input, to decide whether to open or close Valve A to regulate the liquid level in the tank about some desired values (specified in the user input) so that the tank always has some liquid in it (so that customers are always supplied) and so that the tank never overfills (an unsafe situation).

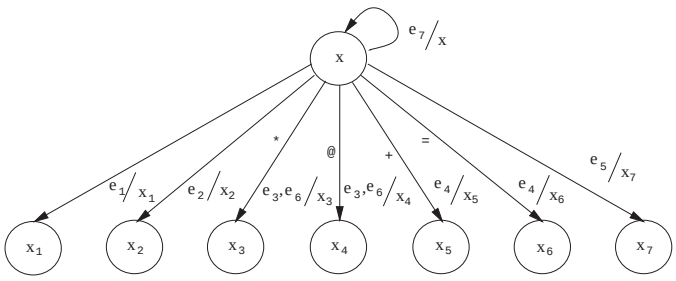

Figure 2: Surge Tank

We begin by specifying the dynamics of the surge tank process using $\mathbf{E}_v$ as it is explained at the beginning of Section 2. We have $\mathcal{E}_u = \{e_{Aon}, e_{Aoff}\}$, where $e_{u_k} = e_{Aon}$ (e_{Aoff}) means "open (close) Valve A at time k"; $\mathcal{E}_d = \{e_{Bon}, e_{Boff}\}$, where $e_{d_k} = e_{Bon}$ (e_{Boff}) means "a customer turns on (off) Valve B at time k"; and $\mathcal{E}_o = \{1, 2, 3, 4, 5, 6, 7\}$, the measured liquid level in the tank at time k is $e_{o_k} = h(k)$. Next, we define how the liquid level in the tank will react to valves being turned on and off. Recall that we denote a process event by $e_k = \{e_{u_k}, e_{d_k}, e_{o_k}\}$ where for us $e_{o_k} = h(k)$, the height of the liquid level in the tank when the event occurs (the updated liquid level will appear in the next event that occurs). For the liquid levels $h(k) \in \{2, 3, 4, 5\}$:

- If $e_k = \{e_{Aoff}, e_{Boff}, h(k)\}$ occurs at time k, $h(k+1) = h(k)$,
- If $e_k = \{e_{Aoff}, e_{Bon}, h(k)\}$ occurs at time k, $h(k+1) = h(k) - 1$,
- If $e_k = \{e_{Aon}, e_{Boff}, h(k)\}$ occurs at time k, $h(k+1) = h(k) + 2$, and
- If $e_k = \{e_{Aon}, e_{Bon}, h(k)\}$ occurs at time k, $h(k+1) = h(k) + 1$.

For $h(k) = 1$ (i.e., the tank is empty):

- If $e_k = \{e_{Aoff}, e_{Boff}, h(k)\}$ occurs at time k, $h(k+1) = 1$,
- If $e_k = \{e_{Aoff}, e_{Bon}, h(k)\}$ occurs at time k, $h(k+1) = 1$,
- If $e_k = \{e_{Aon}, e_{Boff}, h(k)\}$ occurs at time k, $h(k+1) = 3$, and
- If $e_k = \{e_{Aon}, e_{Bon}, h(k)\}$ occurs at time k, $h(k+1) = 2$.

For $h(k) = 6$ (i.e., the tank is almost full):

- If $e_k = \{e_{Aoff}, e_{Boff}, h(k)\}$ occurs at time k, $h(k+1) = 6$,
- If $e_k = \{e_{Aoff}, e_{Bon}, h(k)\}$ occurs at time k, $h(k+1) = 5$,
- If $e_k = \{e_{Aon}, e_{Boff}, h(k)\}$ occurs at time k, $h(k+1) = 7$, and
- If $e_k = \{e_{Aon}, e_{Bon}, h(k)\}$ occurs at time k, $h(k+1) = 7$.

For $h(k) = 7$ (i.e., the tank is full):

- If $e_k = \{e_{Aoff}, e_{Boff}, h(k)\}$ occurs at time k, $h(k+1) = 7$,
- If $e_k = \{e_{Aoff}, e_{Bon}, h(k)\}$ occurs at time k, $h(k+1) = 6$,
- If $e_k = \{e_{Aon}, e_{Boff}, h(k)\}$ occurs at time k, $h(k+1) = 7$, and
- If $e_k = \{e_{Aon}, e_{Bon}, h(k)\}$ occurs at time k, $h(k+1) = 7$.

Hence, for the surge tank the following event trajectories (all infinite length event sequences) are in $\mathbf{E}$:

1. $\{e_{Aoff}, e_{Boff}, 3\}$,$\{e_{Aoff}, e_{Boff}, 3\}$, $\{e_{Aon}, e_{Bon}, 3\}$,$\{e_{Aoff}, e_{Boff}, 4\}$, ...
2. $\{e_{Aon}, e_{Boff}, 3\}$,$\{e_{Aon}, e_{Bon}, 5\}$, $\{e_{Aon}, e_{Boff}, 6\}$,$\{e_{Aon}, e_{Boff}, 7\}$, ...
3. $\{e_{Aoff}, e_{Boff}, 3\}$, $\{e_{Aoff}, e_{Boff}, 4\}$, $\{e_{Aoff}, e_{Boff}, 5\}$, ...

However, only event trajectories that are physically possible are in $\mathbf{E}_v \subset \mathbf{E}$. For example, in sequence (3) above the event trajectory is not physically possible since if both fill valves are off, it is not possible that the liquid level rises. Sequence (2) above is a possible sequence for the dynamical process, but the expert system that is used for liquid level regulation below will not allow this sequence of events since it does not represent good regulation properties. The set $\mathbf{E}_v$ contains all physically possible infinite length event trajectories that can be generated by the dynamical process as it is defined above. This completes the definition of the dynamical process; next we define the expert controller.

The model C for the expert controller for the surge tank has:

- $m = 2$, $n = 0$, $\mathcal{X}^i = \emptyset$, and $\mathbf{x}^b = [x_1^b]$, where $x_1^b \in \{1, 2, 3, 4, 5, 6, 7\}$ represents the most recently measured liquid level in the tank (where we assume that initially the expert controller knows the tank liquid level),
- $\mathcal{E}_d^c = \{3, 5\}$ where $e_{d_k}^c = 3$ indicates that the user would like the liquid level regulated around level 3 (allowing deviations only to levels 2 and 4) and $e_{d_k}^c = 5$ indicates that the user would like the liquid level regulated around level 5 (allowing deviations only to levels 4 and 6),
- $\mathcal{E}_u^\ell = \{\{3, 1\}, \{3, 2\}, \ldots, \{3, 7\}, \{5, 1\}, \{5, 2\}, \ldots, \{5, 7\}\}$ represents the set of sets of input events to the expert controller that can occur (e.g., the occurrence of $\{5, 2\}$ happens when the liquid level is 2 and the user would like to regulate the liquid level around level 5),
- g^c, f_e^c, and δ_e^c are defined by the rules $r \in \mathcal{R}$ in the rule-base of the expert controller:
  - Rule 1: IF $e_{o_k} = 1$ THEN $e_{o_k}^c := \{e_{Aon}\} \wedge x_1^b := 1$
  - Rule 2: IF $e_{o_k} = 2$ THEN $e_{o_k}^c := \{e_{Aon}\} \wedge x_1^b := 2$
  - Rule 3: IF $e_{o_k} = 3 \wedge e_{d_k}^c = 3$ THEN $e_{o_k}^c := \{e_{Aoff}\} \wedge x_1^b := 3$
  - Rule 4: IF $e_{o_k} = 4 \wedge e_{d_k}^c = 3$ THEN $e_{o_k}^c := \{e_{Aoff}\} \wedge x_1^b := 4$
  - Rule 5: IF $e_{o_k} = 3 \wedge e_{d_k}^c = 5$ THEN $e_{o_k}^c := \{e_{Aon}\} \wedge x_1^b := 3$
  - Rule 6: IF $e_{o_k} = 4 \wedge e_{d_k}^c = 5$ THEN $e_{o_k}^c := \{e_{Aon}\} \wedge x_1^b := 4$
  - Rule 7: IF $e_{o_k} = 5$ THEN $e_{o_k}^c := \{e_{Aoff}\} \wedge x_1^b := 5$
  - Rule 8: IF $e_{o_k} = 6$ THEN $e_{o_k}^c := \{e_{Aoff}\} \wedge x_1^b := 6$
  - Rule 9: IF $e_{o_k} = 7$ THEN $e_{o_k}^c := \{e_{Aoff}\} \wedge x_1^b := 7$

There are no particular restrictions placed on $\mathbf{E}_v^c$. This completes the definition of the expert controller.

Notice that the inference engine is particularly simple for the expert controller. There is no need for the complex conflict resolution strategies such as refraction and recency presented in Section 2. The expert controller simply fires one of its nine rules in response to the fourteen different possible combinations of user inputs and process outputs. Notice that one could easily improperly define the expert controller so that it will fail to achieve its task. For instance, it is easy to define the expert controller so that circular reasoning could occur and the goals of regulation specified by the user input would not be achieved. Next, we perform reachability and stability analysis of the expert controller specified above to verify that it behaves correctly.

For reachability analysis let $\mathcal{X}_m \subset \mathcal{X}^c$, and

$$\mathcal{X}_m = \{x_1^b : x_1^b \in \{2, 3, 4, 5, 6\}\} \tag{11}$$

be the set of goal states that represent our desire to regulate the liquid level between 2 and 6 no matter what the user inputs are. We will also have occasion to use the sets $\mathcal{X}_{m3}, \mathcal{X}_{m5} \subset \mathcal{X}_m$ where

$$\mathcal{X}_{m3} = \{x_1^b : x_1^b \in \{2, 3, 4\}\} \tag{12}$$

and

$$\mathcal{X}_{m5} = \{x_1^b : x_1^b \in \{4, 5, 6\}\}. \tag{13}$$

A search algorithm can be used to show the following result:

Theorem 5.1 *The expert controller C is $(x_0^c, \mathcal{X}_m)$-reachable, $(x_0^c, \mathcal{X}_{m3})$-reachable, and $(x_0^c, \mathcal{X}_{m5})$-reachable for all $x_0^c \in \mathcal{X}_m$*

For stability analysis let the distance between two states of the expert controller be

$$\rho(x_1^b, x_1^{b'}) = |x_1^b - x_1^{b'}|. \tag{14}$$

Notice that the set $\mathcal{X}_m$ is an invariant set for the expert controller since if the state of the expert controller starts in $\mathcal{X}_m$ it will stay in $\mathcal{X}_m$ for all time.

Theorem 5.2 *The invariant set $\mathcal{X}_m$ for the expert controller C is asymptotically stable in the large w.r.t. $\mathbf{E}_v^c$.*

Proof: Let $x_1^b = x^c$. From Equation (14) we have

$$\rho(x^c, \mathcal{X}_m) = min\{|x^c - x'| : x' \in \mathcal{X}_m\}. \tag{15}$$

Define the Lyapunov function $V(x^c) = \rho(x^c, \mathcal{X}_m)$ so that choosing $c_1 = c_2 = 1$ we get satisfaction of conditions (1) and (2) of Theorem 3.1. We want to show that for all initial states, $V(x^c)$ is a nonincreasing function of k and that $V(x_k^c) \to 0$ as $k \to \infty$ for the rules that the expert system will fire. Notice that if $x_k^c = 7$ (full), $V(x_k^c) = 1$, the fill valve is off (by rule 9), and since we know that e_{Bon} will occur in a finite amount of time eventually $x_{k'}^c = 6$ for some $k' > k$ so that $V(x_{k'}^c) = 0$. If $x_k^c = 1$ (empty), $V(x_k^c) = 1$, and the fill valve is on (by rule 1) so we know that no matter whether Valve B is opened or closed, the liquid level will rise so that for some $k' > k$, $V(x_{k'}^c) = 0$. ■

Theorem 5.3 *If the user input sequence $e_{d_k}^c = 3$ (5) for all $k \geq 0$ then $\mathcal{X}_{m3}$ ($\mathcal{X}_{m5}$) is asymptotically stable in the large w.r.t. $\mathbf{E}_v^c$.*

The proof of Theorem 5.3 is similar to that of Theorem 5.2 except that additional dynamics of the surge tank process must be considered in showing that $V(x_k^c) \to 0$ as $k \to \infty$. Theorem 5.2 shows that the expert system will achieve its regulation task. Moreover, it shows that the expert system will not exhibit cyclic behavior outside the invariant set (goal set) $\mathcal{X}_m$. Theorem 5.3 shows that the expert system will appropriately respond to user inputs that specify the level around which the user would like the tank liquid level regulated. In other words, it verifies that the user-specified goals will be achieved.

6 Concluding Remarks

We have shown that conventional knowledge-base debugging tools can ignore important dynamic behavior that can result from connecting the full expert system (i.e., with an inference engine) to user inputs and a dynamic process. We have introduced the idea of characterizing and analyzing reachability, cyclic behavior, and stability of expert systems. Moreover, we have illustrated the results by modeling and analyzing expert systems that solve a water-jug filling problem and a simple process control problem.

Below we summarize some of the difficulties in performing a formal verification of the dynamical operation of rule-based expert systems that this investigation has uncovered:

1. The rule-base can represent complex situations that were encountered in the distant past (e.g., sometimes the premises of rules depend on past components of the state vector for backtracking and cycle avoidance as in the example in Section 4) and can represent complex predictions of what can happen in the future as the result of its actions.

2. The inference engine utilizes strategies that depend on the path of inferences that were executed up to the current state (e.g., for recency and refraction and other more complex inference strategies).

3. The recency strategy in the inference mechanism can result in some of the components of the state vector to become unbounded as $k \to \infty$ (the "ages", s_i) [23].

4. Notice that 1 and 2 make the analysis of the qualitative properties of rule-based expert systems relatively complex (this is not surprising since the dynamics of these systems, even for simple examples like the ones studied in Sections 4 and 5, are relatively complex).

5. Also, 1 and 2 typically cause any graph algorithm used for the analysis of qualitative properties of rule-based expert systems to be computationally prohibitive for complex realistic applications.

6. Notice that 1-3 cause significant problems if one is interested in implementing rule-based expert systems for real-time applications.

In light of such problems with performing verification of expert system dynamics, it is clear that several of the more empirical/simulation-based expert system verification approaches outlined in the Introduction must play a significant role in the verification of more complex rule-based expert systems. However, as with the engineering/scientific analysis of any complex dynamical system, there is clearly a role for empirical/simulation-based investigations *and* formal mathematical approaches. The mathematical approaches are certainly in their infancy. As we indicate next, a significant amount of work needs to be performed to enhance our understanding of mathematical analysis of the dynamics of AI systems.

There are many important future directions for this work including: (*i*) characterizing and analyzing additional properties of expert systems (e.g., to show that a given system is unstable or to illustrate the ability of an expert system to meet hard real-time constraints), (*ii*) investigating the dynamics of AI reasoning systems that utilize learning and planning in various complex applications, (*iii*) studying computational complexity issues relative to conflict resolution strategies and knowledge representation, and (*iv*) modeling and analyzing *realistic* industrial applications that involve expert systems (unlike the academic examples considered in this paper). We emphasize

that these research directions help to point out the deficiencies with the approach to expert system verification reported on in this paper.

Finally, we note that overall, it is hoped that this work takes steps towards establishing a foundation for the mathematical verification and *certification* of the dynamics of rule-based expert systems that operate in critical environments.

Acknowledgement: The authors would like to thank the reviewers for their helpful comments.

References

[1] B. Buchanan and E. H. Shortliffe, *Rule Based Expert Systems, MYCIN*. Reading, MA: Addison Wesley, 1984.

[2] L. Brownston, R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5*. Reading, MA: Addison Wesley, 1986.

[3] F. Hayes-Roth, et al., *Building Expert Systems*. Reading, MA: Addison Wesley, 1985.

[4] S. M. Weiss and C. A. Kulikowski, *A Principal Guide To Designing Expert System, (Chapter 6)*. NJ: Rowman and Allenheld Pub., 1984.

[5] P. J. Antsaklis and K. M. Passino, eds., *An Introduction to Intelligent and Autonomous Control*. Norwell, MA: Kluwer Academic Publishers, 1993.

[6] U. Gupta, ed., *Validating and Verifying Knowledge-Based Systems*. Los Alamitos, CA: IEEE Computer Society Press, 1991.

[7] J. R. Geissman, "Verification and validation of expert systems," *AI Expert*, vol. 3, pp. 26–33, Feb. 1988.

[8] C. J. Green, "Verification and validation of expert systems," in *Proceedings of IEEE Western Conference on Expert Systems*, pp. 28–43, June 1991.

[9] B. Chandrasekaran, "On evaluating AI systems for medical diagnosis," *The AI Magazine*, pp. 34–48, Summer 1983.

[10] S. Kim, *Checking a Rule Base with Certainty Factor for Incompleteness and Inconsistency*. NY: Springer Verlag, 1988. In Uncertainty and Intelligent Systems Lecture Note in Computer Science Number 313.

[11] J. Gaschnig, P. Klahr, H. Pople, E. Shortliffe, and A. Terry, "Evaluation of expert systems: Issues and case studies," in *Building Expert Systems* (F. Hayes-Roth, D. A. Waterman, and D. B. Lenat, eds.), Reading, MA: Addison Wesley, 1983.

[12] R. M. O'Keffe, O. Balci, and E. P. Smith, "Validating expert system performance," *IEEE Expert*, vol. 2, no. 4, pp. 81–89, 1987.

[13] A. Giordana and L. Saitta, "Modeling production rules by means of predicate transition networks," *Inf. Sci*, vol. 39, pp. 1–41, 1985.

[14] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, vol. 77, pp. 541–580, Apr. 1989.

[15] A. D. Lunardhi and K. M. Passino, "Verification of dynamic properties of rule-based expert systems," in *Proceedings of the IEEE Conference on Decision and Control*, (Brighton, UK), pp. 1561–1566, Dec. 1991.

[16] M. Suwa, A. C. Scott, and E. H. Shortliffe, "An approach to verifying completeness and consistency in a rule-based expert systems," *The AI Magazine*, pp. 16–21, Fall 1982.

[17] T. A. Nguyen, W. A. Perkins, T. J. Laffey, and D. Pecora, "Checking an expert systems knowledge base for consistency and completeness," *Proceedings 9th International Joint Conference on Artificial Intelligence*, vol. 1, pp. 375–378, Aug. 1985.

[18] T. A. Nguyen, "Verifying consistency of production systems," in *Proceedings 3rd Conference on AI Applications*, (FL), 1987.

[19] T. A. Nguyen, W. A. Perkins, T. J. Laffey, and D. Pecora, "Knowledge base verification," *The AI Magazine*, pp. 69–75, Summer 1987.

[20] W. A. Perkins, T. J. Laffey, D. Pecora, and T. A. Nguyen, "Knowledge base verification," in *Topics in Expert System Design* (G. Guida and C. Tasso, eds.), North Holland: Elsevier Science Publishers B. V., 1989.

[21] K. M. Passino and P. J. Antsaklis, "A system and control theoretic perspective on artificial intelligence planning systems," *International Journal of Applied Artificial Intelligence*, vol. 3, pp. 1–32, 1989.

[22] K. M. Passino and P. J. Antsaklis, "Modeling and analysis of artificially intelligent planning systems," in *An Introduction to Intelligent and Autonomous Control* (P. J. Antsaklis and K. M. Passino, eds.), Kluwer Academic Publishers, 1993.

[23] K. M. Passino and A. D. Lunardhi, "Stability analysis of expert control systems," in *Proceedings of the IEEE Conf. on Decision and Control*, (San Antonio, TX), pp. 765–770, December 1993.

[24] C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller - Part I," *IEEE Trans. on Systems, Man. and Cybernetics*, vol. 20, pp. 404–418, Mar. 1990.

[25] B. Buchanan and R. Duda, "Principles of rule-based expert systems," in *Advances in Computers, Vol. 22* (M. C. Yovits, ed.), NY: Ace Press, 1983.

[26] R. Davis and J. King, "An overview of production systems," in *Machine Intelligence 8* (E. Elcock and D. Michie, eds.), NJ: Ellis Horwood, 1977.

[27] K. M. Passino and P. J. Antsaklis, "A metric space approach to the specification of the heuristic function for the A* algorithm," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, pp. 159–166, Jan. 1994.

[28] K. M. Passino and P. J. Antsaklis, "Optimal stabilization of discrete event systems," *Proceedings of the Conference on Decision and Control*, pp. 670–671, Dec. 1990.

[29] K. M. Passino, A. N. Michel, and P. J. Antsaklis, "Lyapunov stability of a class of discrete event systems," *IEEE Transactions on Automatic Control*, vol. 39, pp. 269–279, February 1994.

[30] K. M. Passino, A. N. Michel, and P. J. Antsaklis, "Lyapunov stability of a class of discrete event systems," in *Proceedings of the American Control Conference*, (Boston, MA), pp. 2911–2916, 1991.

[31] K. M. Passino and A. D. Lunardhi, "Qualitative analysis of expert control systems," in *Intelligent Control Systems: Theory and Applications* (M. Gupta and N. Sinha, eds.), IEEE Press, 1993.

[32] K. J. Åström, J. J. Anton, and K. E. Årzén, "Expert control," *Automatica*, vol. 22, pp. 277–286, 1986.

[33] K. J. Åström and K. E. Årzén, "Expert control," in *An Introduction to Intelligent and Autonomous Control* (P. J. Antsaklis and K. M. Passino, eds.), Kluwer Academic Publishers, 1992.