



Model Checking the Inconsistency and Circularity in Rule-Based Expert Systems

Desheng Xu (Corresponding author)

Department of Computer science and technology

University of Science and Technology Beijing

106 mailbox 30#Xueyuan Road, Haidian District, Beijing 100083, China

Tel: 86-10-6239-0097 E-mail: xudesheng98@163.com

Kejian Xia

Department of Computer science and technology

University of Science and Technology Beijing

30#Xueyuan Road, Haidian District, Beijing 100083, China

Dezheng Zhang

Department of Computer science and technology

University of Science and Technology Beijing

30#Xueyuan Road, Haidian District, Beijing 100083, China

Huangsheng Zhang

Department of Computer science and technology

University of Science and Technology Beijing

30#Xueyuan Road, Haidian District, Beijing 100083, China

The research is financed by the National High-Tech Research and development Plan of China (863) under Grant (No.2007AA01Z170); the Beijing Natural Science Foundation of China under Grant (No.4062022)

Abstract

In the past several years, various techniques were proposed to analyze various types of structural errors, such as inconsistency (conflict rules), and circularity (circular depending rules), of rule-based systems. Model checking is a technique for the verification of temporal logic specifications in state transition systems. In this paper, we model the rule-based systems as finite state transition systems and express consistency and acyclic as Linear Temporal Logic (LTL) logic formula and then use the technique of model checking to detect inconsistency and circularity in Rule-Based Systems with the model checker NuSMV.

Keywords: Model checking, Inconsistency, Circularity, Rule

1. Introduction

Expert systems have been widely used in many real world applications. The central part of an expert system is a rule base that codifies the knowledge from domain experts in the form of inference rules. Often these inference rules are built into a rule base incrementally over years and subject to frequent refinements. Due in part to the above construction process of a rule base and in part to the different and even conflict views provided by domain experts, a rule base can contain many structural errors. According to (D.L. Nazareth. 1989), the typical types of structural errors include

inconsistency (conflict rules), incompleteness (missing rules), redundancy (redundant rules), and circularity (circular depending rules). But we just focus on the inconsistency and circularity in this paper.

Many different techniques have been proposed to detect the above structural errors in rule-based systems (a quite comprehensive list of references can be found in (M. Ramaswamy, et al., 1997)). Earlier work mainly focused on detecting structural errors by checking rules pair-wisely. Recent work aimed at detecting structural errors caused from applying multiple rules in longer inference chains. The majority of the recent verification techniques involve using some graphical notation such as Petri nets and graphs. Several of the above approaches cannot detect structural errors accurately and report spurious errors when compound antecedent clauses overlap. The approach in (D. Zhang and D. Nguyen, 1994) could only detect structural errors matching a set of pre-defined syntactic patterns. The approaches in (G. Valiente, 1993) and (S. J.H. Yang, et al. 1998) did not address inconsistency errors. In (M. Ramaswamy, S. Sarker, and Y.S. Chen, 1997), an adjacency matrix technique was used, which has a higher computational cost both in terms of space (the sparseness of typical adjacency matrices) and time (the number of addition and multiplication involved in matrix operations).

Model checking (William Chan, 1998) is a technique for the verification of temporal logic specifications in state transition systems. In this paper, we present a model to model the rule set of rule-based systems as state-transition systems and express consistency and acyclic as LTL logic (A. Pnueli . 1981) formula and then use the technique of model checking to detect inconsistency and circularity in Rule-Based Systems with the model checker NuSMV. This technique is simple, efficient, and automated. We highlight the unique features of this new approach and demonstrate its application through an example by the model checker NuSMV.

2. Model checking

Model checking (William Chan, 1998) is an automatic approach to formal verification based on state exploration. This verification is performed by software tools, which given a state transition system and a property, model checking algorithms exhaustively explore the state space to determine whether the system satisfies the property. Fig. 1 is a schematic of the process of model checking, a model of the specification and a property are fed to a model checker. The result is either a claim that the property is true or else a counterexample (a sequence of states from some initial state) falsifying the property. In practice, counterexamples often provide valuable debugging information, and can be used by the user to modify the model, or the property checked. This iterative process is inherent in our work. In this context the model is a state transition system and the specification is formalized with temporal logic which pinpoints desired behavior over paths and states in the model.

In the rest of Section 2, we give an overview of the basics of LTL model checking and NuSMV, the model checker that we used.

2.1 The LTL Model Checking Problem

In temporal-logic model checking, we are given a state transition system, which models a software or hardware system, and a property specified as a formula in a certain temporal logic, and determine whether the system satisfies the formula. A common logic for model checking is the Linear Temporal Logic (LTL), which extends propositional logic with certain temporal operators. Typical formulas include the following (The symbol “**p**” is an atomic proposition,):

F p : p holds eventually or sometime in the *future*.

G p : p holds *globally* or always in the future.

Formally, a state transition system $\langle P, R, I \rangle$ consists of a set of states P , a state *transition relation* $R \subseteq P \times P$, and a set of initial states $I \subseteq P$. A *path* is an infinite sequence of states such that each consecutive pair of states is in R . The set of states P is often encoded by a set of *state variables*, such that each state corresponds to some valuation for the variables and no distinct states correspond to the same valuation (that is, the mapping of P to the variable valuations is one-to-one).

For simplicity we discuss just a subset of LTL, namely the subset with only the temporal operators F and G, which are sufficient to understand our examples. We can recursively define this restricted class of LTL formulas as follows: We say that a *proposition* is any Boolean combination of predicates on the state variables. A *formula* is either a proposition, a Boolean combination of formulas, or of the form Ff , Gf , where f is a formula. And we use the symbolic “!” , “|” , “&” to denote the logic NOT, OR, AND , respectively. Each formula is evaluated at some state q . A proposition holds at q if q satisfies the proposition.

The system satisfies a formula if the formula holds at all initial states. If not, a model checker typically attempts to find a counterexample. For example, if the formula $!(F P5=1 \ \& \ F P5=0)$ is false, a counterexample is a finite path starting at some initial state sometime satisfies the proposition $P5=1$ and sometime $P5=0$.

Readers familiar with temporal-logic model checking may notice that, although a LTL formula is usually interpreted over a *Kripke* structure which is a model for the representation of a finite-state concurrent system. The model consists of

the set of states, transition relation and labeling (which gives semantics to the structure). In our definition, a state is not explicitly labeled, but can be thought as being labeled implicitly by its corresponding state-variable valuations. This more restricted formulation is sufficient for our presentation.

2.2 NuSMV

Verification of the inconsistency and circularity problem in Rule-Based Systems has been carried out by the symbolic model checker NuSMV(R. Cavada, et al.2005) that originated from reengineering, reimplementing and extension of CMU SMV, the original BDD-based model checker developed at CMU. NuSMV allows for the representation of specifications expressed in CTL and Linear Temporal logic(LTL) using BDD-based and SAT-based model checking techniques. The primary purpose of the NuSMV input language is to describe the transition relation of a finite *Kripke* structure. The input file describes both the model and the specification (with possible fairness constraints). The states are defined by a collection of state variables, which may be of Boolean or scalar type. The transition relation of the *Kripke* structure is determined by a collection of parallel assignments, which are introduced by a keyword ASSIGN. The semantics of assignment in NuSMV is similar to that of a single assignment data flow languages. A program can be viewed as a system of simultaneous equations, whose solutions determine the next state. When a set is assigned to a variable, the result is a nondeterministic choice among elements of the set. The case statements of NuSMV are evaluated from the top down: if several expressions to the left of a “:” are true, then the command corresponding to the first, top-most true expression will be executed.

3. Inconsistency and circularity problem in Rule-Based Systems and our method

3.1 Model the Rule Base

An inference rule (clause) has the following general form (William Chan, et al. 1998): $P \rightarrow Q$, or IF P THEN Q , where P and Q are called premise and conclusion respectively. P (or Q) can be an atomic propositional logic formula (a proposition or its negation) or a compound propositional logic formula containing multiple propositions and logical connectives: \wedge, \vee). In the following discussion, we use lower case letters to denote atomic formulas and capital letters to denote compound formulas. We assume that all the inference rules in a rule base are syntactically valid propositional logic formulas (syntax errors are easily detected by a parser). Furthermore, we assume that P and Q are in conjunctive normal form $P_1 \wedge \dots \wedge P_m$ and $Q_1 \wedge \dots \wedge Q_n$ respectively, in which each P_i ($1 \leq i \leq m$) (or Q_j ($1 \leq j \leq n$)) is a disjunction of propositional symbols and their negations, i.e. $p_1 \vee \dots \vee p_k$. The above assumption does not limit the error detecting capability of our approach since any propositional logic formula can be easily and mechanically transformed into a semantically equivalent conjunctive or disjunctive normal form. The following example is modified from an example in [11]. A rule base R is defined as follows:

R={ r1: $p_1 \rightarrow p_2 \wedge p_5$
 r2: $p_2 \rightarrow \neg p_5$
 r3: $p_3 \rightarrow p_1 \wedge p_2$
 r4: $p_1 \vee p_4 \rightarrow p_5$
 r5: $p_5 \rightarrow p_4$ }

Formally, we model the rule-based system we describe as above as a finite-state transition system $\langle P, R, I \rangle$. consists of a set of states P , a state *transition relation* $R \subseteq P \times P$, and a set of initial states $I \subseteq P$, where P is a set of state variables $\{p_1, p_2, \dots, p_n\}$ (n is the number of atomic propositions) which correspond to the atomic propositions appear in the rule base, and all of these variables of enumeration type $\{1, 0, \text{neither}\}$. When variables $p_i=1$, it means p_i is true, and when $p_i=0$, it means p_i is false, if $p_i=\text{neither}$, it means neither p_i nor $\neg p_i$ is true. In our definition, a state *transition relation* is not explicit presentation. The initial states I is also not explicit presentation. We use the variable *InitTrue* of enumeration type $\{r_1, r_2, \dots, r_m, \text{neither}\}$ (m is the number of rules in the rule base) to denote the initial state implicitly, when *InitTrue* = r_i , it indicates that the premise of rule r_i is true. Initially, the initial value of state variables p_i is *neither*, and the initial value of *InitTrue* is a nondeterministic choice among elements of the set $\{r_1, r_2, \dots, r_m\}$. If *InitTrue* = r_i , it indicates that rule r_i is enable (Doing this is similar to the procedure we input some facts enable rule r_i), and then we start the reasoning to change the values of state variables according to the rules in the rule base. If the value of state variable p_i is non-neither and there no other rules assign 1 or 0 to it, then we reset p_i to *neither*. And we also set the subsequent value of *InitTrue* to be *neither*. This is the state *transition relation* of this model. For the rule base R in Fig.1, suppose the initial value of *InitTrue* is r_3 , then the state transition is illustrated in Fig.4. This is the way how we model the rule-based system as a finite-state transition system.

3.2 Inconsistency

Inconsistency results in conflict facts and must be resolved for correct functioning of an expert system. There are two cases of contradiction.

Case 1: When we start reasoning from a rule, sometime pi is true and some time $\neg pi$ is true, then the contradiction occurs. So, if there exists both pi and $\neg pi$ in the rule base, we should check whether the LTL formula $!(F pi = 1 \ \& \ F pi = 0)$ be satisfied or not in the model. For the instance we presented in Fig.1 , initially, we enable rule $r1$, then $p5$ and $p2$ is valued to 1. and then rule $r2$ is fired by $p2$, as the result, $p5$ is evaluated to 0. This results a contradiction.

Case 2 : If there are some rules make pi true and some make $\neg pi$ true, then the premises make pi true and $\neg pi$ true respectively should not be true at the same time. So we use LTL formula $!F((p4=1 \mid p1=1) \ \& \ (p2=1))$ to check the example we given whether satisfies this property.

3.3 Circularity

Circularity occurs when several inference rules have circular dependency. Circularity can cause infinite reasoning and must be broken. An example of circularly dependent rules is as follows:

r1: $p \rightarrow q$

r2: $q \rightarrow p$

If a rule is a part of a circle, then the state variables such as pi form the rule will be assigned to 1 or 0 infinite often. So, if there no circularity in the rule set, then all of state variables pi should always equal to neither in the future. We use the LTL formula $F G(p1=neither \ \& \ p2=neither \ \& \ \dots \ \& \ pn=neither)$ to express this property. If this formula can't be satisfied, we conclude there must be some circularity occurs.

4. The experiment about the example presented in 3.1

We use the NuSMV to check the example presented in 3.1, the source code we used as below:

MODULE main

VAR

p1: {0,1, neither};

.....

p5: {0,1, neither};

InitTrue: {r1,r2,r3,r4,r5, neither};

ASSIGN

init(p1) := neither;

.....

init(p5) := neither;

init(InitTrue) := {r1,r2,r3,r4,r5};

next(InitTrue) := neither;

next(p1) := case

InitTrue=r1 | InitTrue=r4 | p3=1 : 1 ;

p1 != neither : neither;

1 : p1;

esac;

next(p2) := case

InitTrue=r2 | p1=1 | p3=1 : 1 ;

p2 != neither : neither;

1 : p2;

esac;

next(p3) := case

InitTrue=r3 : 1 ;

p3 != neither : neither;

```

1: p3;
esac;
next(p4):= case
InitTrue=r4 | p5=1 : 1 ;
p4 != neither : neither;
1: p4;
esac;

next(p5):= case
InitTrue=r5 | p4=1 | p1=1 : 1 ;
p2=1 : 0;
p5 != neither : neither;
1: p5;
esac;
LTLSPEC !(F p5=1 & F p5=0);
LTLSPEC !F((p4=1| p1=1) & (p2=1));
LTLSPEC F G(p1=neither & p2=neither & p3=neither & p4=neither & p5=neither);

```

The NuSMV shows that all the three LTL formulas are false, and produce the counterexamples as Fig.2 indicates rule r_2 is inconsistent with rules r_1 and rules $\{r_3, r_4\}$, rules r_4 and r_5 are in self-loop.

5. Conclusion and future work

In this paper we have proposed a finite-state transition system model which can be used for describing the behaviors of the rule-based system. We have focused on the inconsistency and circularity problem of a given rule base, and identified two sorts of inconsistency problems and circularity problem. We also present two kinds of LTL formulas and a kind of LTL formula for checking the consistency and circularity of the rule base respectively, and carry a experiment to explain our method. Our technique has the following advantages:

- (1) It is general and is capable to detect all potential inconsistency and circularity errors without imposing any restrictions on the form of rules;
- (2) It can produce the results automated, The result is easy to understand and we can locate the errors easily with the help of the counterexamples;
- (3) It is easily extensible to predicate forms such that each p_i is a predicate instead of simple proposition. Although describing of *transition relation* between states becomes more complicated and requires more careful.

But since the conversion from rules to the model we input into the model checker is manual work, we will develop a software tool to implement the method we proposed automatic in the future. And from theoretical point of view, to check the Inconsistency and circularity in rule base, we have to check the validity of each possible state space. The possible state spaces in our method is 3^n (n is the number of state variables), which is exponential. So, unfortunately, the size of the state space is exponential in the size of the rule base, resulting in the *state explosion problem*. But since many methods were proposed to deal with this problem, we still can do some research when the rule base is too huge to apply the method proposed in this paper directly.

References

- A.Pnueli . (1981). A temporal logic of programs. *Theoretical Computer Science*, 13:45-60, 1981.
- D. Zhang and D. Nguyen, (1994). "PREPARE: A Tool for Knowledge Base Verification," *IEEE Trans. On Knowledge and Data Engineering*, vol. 6, no., 6, Dec. 1994, 983-989.
- D.L. Nazareth and M.H. Kennedy, (1991). "Verification of Rule-Based Knowledge Using Directed Grphs", *Knowledge Acquisition*, vol. 3, 1991, 339-360.
- D.L. Nazareth, (1993). "Investigating the Applicability of Petri Nets for Rule-Based System Verification," *IEEE Trans. on Knowledge and Data Engineering*, vol. 4, no., 3, 1993, 402-415.
- D.L. Nazareth. (1989), "Issues in the Verification of Knowledge in Rule-Based Systems," *Int'l J. of Man-Machine Studies*, vol. 30, 1989, 255-271.

G. Valiente, (1993). Verification of Knowledge Based Redundancy and Subsumption Using Graph. Transformations, *Int'l J. of Expert Systems*, vol. 6, no. 3, 1993, 341-355.

M. Ramaswamy, S. Sarkar, and Y.S. Chen, (1997). "Using Directed Hypergraphs to Verify Rule-Based Expert Systems," *IEEE Trans. on Knowledge and Data Engineering*, vol. 9, no., 2, 1997, 221-237.

R. Agarwal and M. Tanniru, (1992). "A Petri Net Based Approach for Verifying the Integrity of Production Systems", *Int'l J. of Man-Machine Systems*, vol.36. 1992, 447-468.

R. Cavada, A. Cimatti, E. Olivetti, M. Pistore, and M. Roveri. (2005).NuSMV 2.4 User Manual, 2005.

S. J.H. Yang, A.S. Lee, W.C. Chu, and H. Yang,(1998)."Rule Base Verification Using Petri Nets", Proc. Of 22nd International Computer and Software Application Conference (COMPSAC'98), Vienna,Austria, 1998.

William Chan, Richard J. Anderson, Paul Beame, Steve Burns, Francesmary Modugno, David Notkin, and Jon D. Reese. (1998). Model checking large software specifications. In *IEEE Transactions on Software Engineering* 24(7), pages 498-520, July 1998.

X. He, W. Chu, and H. Yang. (2003). "A New Approach to Verify Rule-Based Systems using Petri Nets", *Information and Software Technology*, vol. 45, no.10, 2003, 663-670.

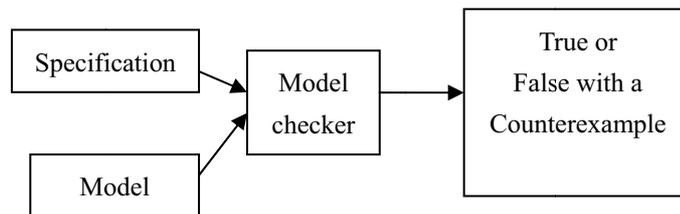


Figure 1. Model checking at a glance.

Loop	Step	InitTrue	p1	p2	p3	p4	p5
	0	r3	neither	neither	neither	neither	neither
	1	neither	neither	neither	1	neither	neither
	2	neither	1	1	neither	neither	neither
	3	neither	neither	1	neither	neither	1
	4	neither	neither	neither	neither	1	0
↩	5	neither	neither	neither	neither	neither	1
	6	neither	neither	neither	neither	1	neither
	7	neither	neither	neither	neither	neither	1

Figure 2. One of the counterexamples produced by NuSMV.