Verification of Active Rule Base via Conditional Colored Petri Nets

Lorena Chavarría-Báez and Xiaou Li

I. INTRODUCTION

Knowledge verification concerns the correctness and appropriateness of the structure of a knowledge base, and is a key process to ensure high quality of a knowledge-based system. Structural errors in knowledge-based system include redundancy, inconsistency, incompleteness, and circularity.

In earlier knowledge-based systems, production rules are a kind of generally adopted knowledge expression method. The verification of knowledge especially for production rule based systems has been broadly discussed. Many verification methods and tools have been developed and successfully used, e.g., tabular methods[5], methods based on the generation of labels [12], methods based on graphs[13], methods based on algebraic interpretation[1], and methods based on Petri Nets [7], [16], [18], [10], [17].

In recent years, active rules are widely used in knowledge-based systems since they can make systems behave automatically once certain events are detected and certain conditions are satisfied. Active rule base verification is not the same as that of production rule base, because the semantics and syntax of the former are more complicated. In consequence, traditional verification conceptions need to be reconsidered in active rules context. Verification of active rule base should be as important as that of production rule bases [2]. Nevertheless, to the best of our knowledge, there is not enough work reported on it [2], [5], [3], [14], [4]. The majority of those references are centered on analyzing properties such as non-termination and confluence, which is beyond of the scope of this paper, and a few of them explore semantic correctness of the set of active rules, which is different from the approach presented here, through analysis of rule properties such as: triggerability, join applicability, rule coverage, and rule cascading.

Abstract— Active rules are widely used in modern reactive software systems, such as active data base management systems, smart homes, etc. Determining if an active rule base is free of errors is an important process for both rule base design and maintenance. In this paper, we originally define the basic errors in an active rule-based system by extending the conceptions which are used generally in production rule base. Furthermore, a Petri net-based approach is proposed for active rule-base verification. An example on smart homes design is used as an application.

II. ACTIVE RULES

A. Introduction

Generally, active rules have three components: an Event, a Condition and an Action, so they are also called ECA (Event-Condition-Action) rules in other literatures. An active rule takes the form as follows:

\[ \text{ON event} \begin{align*} \text{IF condition} \quad & \text{THEN action} \end{align*} \]

The event is something that happens at a point in time. Specifying an event involves providing a description of the happening that is to be monitored. An event can be of two types: primitive or composite (complex). The event is primitive when it cannot be decomposed in smaller events, for example, an exception, a transaction (abort, commit) or a structure operation (e.g., a database operation). The event is composite when it is raised by some combination of primitive or composite events using a range of operators that constitute the event algebra. The range of event operators varies from system to system, however, the most common are: disjunction \( \lor \), conjunction \( \land \), sequence \( \text{SEQ}(E_1, E_2) \), sequence \( \text{SEQ}(E_1, E_2) \land \text{SEQ}(E_1, E_2) \), closure \( \text{CLOSURE}(E_1) \) in \text{Int}, history \( \text{times}(n, E_1) \) in \text{Int} and negation \( \text{NOT}(E_1) \) in \text{Int}, in all the above event operators \( E_1 \) is an event and \( \text{Int} \) is a time interval [11]. The condition examines the context in which the event has taken place. The action describes the task which to be carried out by the rule if the event has taken place and the condition is fulfilled.

Active rule-based systems are able to react to significant events. These systems provide a way to describe events and their associated reactions by means of active rules as well as the runtime strategy (i.e., the execution model) to process and combine this active behavior [11].

In this paper, we approach the problem of knowledge verification of active rule-based systems. Firstly, by analyzing active rule components, we extend the structural error conceptions discussed in production rule base into active rule base. Secondly, a Conditional Colored Petri Net (CCPN) model is used to detect those errors.

The rest of the paper is organized as follows: in Section II we give an introduction on active rules and their execution. In Section III we define structural errors in an active rule base. Section IV proposes an error detection method. Section V is our conclusion.
but an example is showed in next section. During the action execution other events can in turn be signaled and triggered which may produce a cascaded rule firing.

B. Example

Smart homes technology uses a number of small computers distributed around the house which are either used to turn devices and appliances on and off or to send and receive information. The use of computer controls removes the need to actually flick a switch or turn a knob to make something work and allows elements of the home to be controlled remotely by, or to respond automatically to, the people living in it. The information generated from the sensors within the house and the interfaces embedded into common domestic appliances can be processed to provide assistance to the people in a number of different ways, such as prevention of dangerous situations, health, comfort, security and power saving.

In reference [2], an active rule base development is part of the smart home design. To ensure the rule base work correctly, the most relevant work should be the verification, so that the active rules fulfill some desirable requirements and properties. We use some of the rules in [2] to illustrate our verification approach through the paper. The following 6 policies are collected from smart home designers and users during the rule base development:

1. **Security** If during 15 minutes no movement has been detected in house (i.e., there is nobody in house) and there is, at least, either an appliance or device on, then appliances (devices) should be off.

2. **Comfort and power saving** When a person enters into any room, if the light is off, then turn the light on automatically.

3. **Comfort** When no movement is detected in front of automatic paper hand dispatcher, if the detected time was greater than 10 seconds, then give the user enough quantity of paper.

4. **Comfort** When sunlight intensity changes, if the intensity is under a certain limit (at the moment fixed in 50), then increase house light.

5. **Security** If during 15 minutes no movement has been detected in house (i.e., there is nobody in house) then turn the light off automatically.

6. **Power saving** If during 15 minutes no movement has been detected in house (i.e., there is nobody in house) then turn the light off automatically in order to power saving.

**Example I:** Above information can be translated easily into active rule format one by one as follows:

**Rule 1**

\[
\text{ON} \ \text{NobodyIsAtHome( ),}
\text{ThereAreAppliancesOn( )} \quad \text{AND}
\]

\[
\text{IF} \ \text{TimeElapsed} > 15 \quad // \text{Time in minutes}
\]

\[
\text{THEN} \ \text{TurnOffAppliances( )}
\]

**Rule 2**

\[
\text{ON} \ \text{EnterPersonInTheRoom( )}
\]

\[
\text{IF} \ \text{LightOn} = 0 \quad // \text{Light Off}
\]

\[
\text{THEN} \ \text{TurnOnTheLight( )}
\]

**Rule 3**

\[
\text{ON} \ \text{DetectingMovementInFrontOfHandPaperSensor( )}
\]

\[
\text{IF} \ \text{DetectingTime} > 10 \quad // \text{Time in seconds}
\]

\[
\text{THEN} \ \text{DispatchHandPaper( )}
\]

**Rule 4**

\[
\text{ON} \ \text{SunlightIntensityChanges( )}
\]

\[
\text{IF} \ \text{Intensity} < 50
\]

\[
\text{THEN} \ \text{IncreaseLightIntensity( )}
\]

**Rule 5**

\[
\text{ON} \ \text{NobodyIsAtHome( )}
\]

\[
\text{IF} \ \text{TimeElapsed} > 15 \quad // \text{Time in minutes}
\]

\[
\text{THEN} \ \text{TurnOnTheLight( )}
\]

**Rule 6**

\[
\text{ON} \ \text{NobodyIsAtHome( )}
\]

\[
\text{IF} \ \text{TimeElapsed} > 15 \quad // \text{Time in minutes}
\]

\[
\text{THEN} \ \text{TurnOffTheLight( )}
\]

Now suppose the event `NobodyIsAtHome` has been detected (signaling), then Rule 5 and Rule 6 are triggered (triggering). Note that even though Rule 1 is also supervising the occurrence of that event, it is not triggered since its event is complex, i.e., event `ThereAreAppliancesOn` also need to be detected simultaneously with the event `NobodyIsAtHome`. After the event has been detected, conditions of Rule 5 and Rule 6 must be evaluated in order to determine if their corresponding actions can be executed (evaluation). Let's assume the condition `TimeElapsed > 15` is true, then it is necessary to schedule the rule execution (scheduling). For simplicity, rule execution will be in rule list order. Therefore, Rule 5's action will be executed first, and then Rule 6's action (execution). In this case, since there is no rule supervising events `TurnOnTheLight` or `TurnOffTheLight`, which are actions of Rule 5 and Rule 6, respectively, rule execution finishes. Otherwise, if any rule is triggered by those actions, then the rule execution will continue and cause a rule execution chain.

C. Modeling active rules with CCPN

An active rule base can be modeled by a Conditional Colored Petri Net (CCPN) [8]. The mapping process can be summarized as follows: a rule is mapped to a transition where condition is attached, event and action part parts are mapped to input and output places of a transition, respectively. Matching between events and input places has the following characteristics:

- Primitive places represent primitive events;
- Composite places represent composite events;
- Copy places are used when one event triggers two or more rules. An event can be shared by two or more rules, but in PN theory, one token needs to be duplicated for share. A copy place takes the same information as its original one;
- Virtual places are used for accumulating different events that trigger the same rule. For example, when the event part of a rule is the composite event OR.

Matching between conditions and transitions has the following characteristics:

- Rule transitions represent a rule;
- Composite transitions represent composite event generation;
- Copy transitions duplicate one event for each triggered rule.

Figure 1 shows the basic structures to which an active rule can be mapped. In its most basic form, event, condition and action of a rule matches with a primitive input place, a rule transition and a primitive output place, respectively, as shown in Figure 1(a). Whenever an event triggers two or more rules it has to be duplicated by means the copy structure depicted in Figure 1(b). Output places of copy transitions contain the same information as the copy transitions’ input place. By so doing several rules can be triggered by the same event at the same time. Composite events formation is also considered in CCPN using the composite structure drawn in Figure 1(c). Composite transition’s input places represent all the events needed to form a composite event while its output place correspond to the whole composite event. For instance, let’s consider Rule 1’s event, \( \text{AND} (\text{NobodyIsAtHome, ThereAreAppliancesOn}) \), which is a composite one. The composite structure will be made up by a composite transition with two primitive input places, one of them will hold the primitive event: \( \text{NobodyIsAtHome} \), while the other one will store the primitive event: \( \text{ThereAreAppliancesOn} \). Its composite output place will correspond to the entire composite event described above. Finally, we use the virtual structure to model the composite event \( \text{OR} \) as standing for in Figure 1(d). Virtual place acts as an "event store", i.e., when a rule is triggered by several events that place accumulates them and each one of them can be used to trigger the rule. A rule transition can be attached with conditions of the form: 1) \( \langle \text{variable} \rangle \langle \text{op} \rangle \langle \text{constant value} \rangle \) or \( \langle \text{variable} \rangle \langle \text{op} \rangle \langle \text{variable} \rangle \) where \( \text{op} \) can be one of the following operators: \( =, \neq, \leq, \geq \) and \( \text{variable} \) is binding with the event information, or 2) with complex conditions built with two or more conditions connected by logical operators such as \( \text{AND/OR} \), with no need of a special CCPN structure. The CCPN model of a set of active rules is formed by connecting those common places that are output and input places at the same time, i.e., those places that represent both the action of one rule and the event of another rule.

### III. Structural errors in active rule base

An active rule base may have similar errors to those in a production rule base, such as redundancy, inconsistency, etc., but the event and condition binding patterns and composite events make errors in active rule base appear in more complicated ways.

For the reading convenience, hereafter we refer to the event and condition parts as premise of an active rule, and denote an active rule as \( R_i(E_i,C_i,A) \) where \( E_i \) and \( C_i \) are the event, condition and action of rule \( R_i \), respectively. For the limitation on paper length, we only discuss inconsistency and incompleteness in this paper, redundancy and subsumed errors can be referenced to [6]. The other errors such as circularity will be reported in the future.

### A. Inconsistency

Inconsistency refers to the existence of conflicting rules. Two rules are conflicting if their premises consist of the same event and condition, but the actions are mutually exclusive.

**Definition 1:** Conflict rules. Two rules \( R_1(E_1,C_1,A_1) \) and \( R_2(E_2,C_2,A_2) \) are conflicting if the following conditions are satisfied:

1. \( E_2 \subseteq E_1 \)
2. \( C_2 \subseteq C_1 \)
3. \( A_2 \cap A_1 = \emptyset \)

For example, Rule 5 and Rule 6 of Example 1 are conflicting rules, because the light should be on (Rule 5) and off (Rule 6) at the same situation.

Conflicting rules refer to two rules are contradictory. However, sometimes two rules are not always in conflict except certain conditions are fulfilled. We name this kind of conflicting rules potentially conflicting rules.

**Definition 2:** Potentially conflicting rules. Two rules \( R_1(E_1,C_1,A_1) \) and \( R_2(E_2,C_2,A_2) \) are potentially conflicting rules if the following conditions are met: 1) their premises differ in just one element (event or condition), and 2) they take different actions, i.e., \( A_2 \cap A_1 = \emptyset \).

There are two types of potentially conflicting rules:

1. **event - action conflicting rules** are detected if
   a) \( E_2 \subseteq E_1 \)
   b) \( C_2 \subseteq C_1 \)
   c) \( A_2 \cap A_1 = \emptyset \)

2. **condition - action conflicting rules** are detected if
   a) \( E_2 \subseteq E_1 \)
   b) \( C_2 \subseteq C_1 \)
   c) \( A_2 \cap A_1 = \emptyset \)

Conflicting rules must be prevented before the system works, in order to guarantee the system execute proper actions. Potentially conflicting rules don’t cause any problem at the moment, but they must be removed too because they may cause future conflict.
B. Incompleteness

A knowledge base is incomplete when it does not have all the information necessary to execute important actions to the system. Several factors are involved in incompleteness. For instance, during the knowledge acquisition process, both the expert and the knowledge engineer may have inadvertently left gaps in the knowledge base without noticing. Also the knowledge engineer may lose track as the knowledge base grows larger and becomes intractable.

Incompleteness is characterized by isolated rules. An isolated rule is neither triggered by an action of any other rule in the rule base nor its action is able to trigger any more rules.

Definition 3: Isolated rule. Given an active rule base \( \{R_1(E_1, C_1, A_1), R_2(E_2, C_2, A_2), \ldots, R_n(E_n, C_n, A_n)\} \), rule \( R_i(E_i, C_i, A_i) \) is an isolated rule if it fulfills the following conditions:

1) \( A_i \notin \bigcup_{j=1}^{n} E_j \)
2) \( E_i \notin \bigcup_{j=1}^{n} A_j, i \neq j \)

Rule 3 in Example 1 is an isolated rule. Its event DetectingMovementInFrontOfHandPaperSensor is not coincident with any action of other rules, and its action DispatchHandPaper() will not trigger any rule in the rule base. Rule 4 is also an isolated rule since its event doesn’t match with any action rule and, its action doesn’t correspond with any event rule.

IV. RULE BASE VERIFICATION

Based on CCPN model, a rule base can be verified. Our verification process consists of three phases: rule normalization, rule base modeling, and verification.

A. Rule normalization

In order to facilitate the rule element management, we normalize the rules in a rule base into a simple format without affecting system knowledge and performance. Active rules can be classified into the following four types according to event condition combination pattern.

- **Rule type 1** Rules with conjunction operator in the premise and action parts, i.e.,
  \[ \text{ON AND}(E_1, E_2, \ldots, E_n) \]
  \[ \text{IF AND}(C_1, C_2, \ldots, C_m) \]
  \[ \text{THEN AND}(A_1, A_2, \ldots, A_k) \]

- **Rule type 2** Rules with conjunction operator in both event and action parts, and disjunction operator in condition, i.e.,
  \[ \text{ON AND}(E_1, E_2, \ldots, E_n) \]
  \[ \text{IF OR}(C_1, C_2, \ldots, C_m) \]
  \[ \text{THEN AND}(A_1, A_2, \ldots, A_k) \]

- **Rule type 3** Rules with disjunction operator in event and the conjunction operator in both condition and action parts, i.e.,
  \[ \text{ON OR}(E_1, E_2, \ldots, E_n) \]
  \[ \text{IF AND}(C_1, C_2, \ldots, C_m) \]
  \[ \text{THEN AND}(A_1, A_2, \ldots, A_k) \]

- **Rule type 4** Rules with disjunction operator in premise and the conjunction operator in action part, i.e.,
  \[ \text{ON OR}(E_1, E_2, \ldots, E_n) \]
  \[ \text{IF AND}(C_1, C_2, \ldots, C_m) \]
  \[ \text{THEN AND}(A_1, A_2, \ldots, A_k) \]

We didn’t consider disjunctive actions since they do not lead to any logical conclusion. In order to simplify analysis algorithms, we propose to translate the original active rule base into a set of atomic rules. An atomic rule is a rule whose event and condition are conjunctions of one or more primitive events and conditional clauses, and its action is only one instruction. According to the idempotency law and the distribution property of AND over OR and vice versa, any rule can be converted to the disjunctive normal form.

A rule \( R_i(E_i, C_i, A_i) \) can always be divided into several atomic rules by the following steps:

- **Step 1** If \( E_i \) is atomic, it’s OK. If not go to Step 2.
- **Step 2** Transform each element of \( E_i \) into disjunctive normal form, so that each element consists of one or more disjunctions which is a conjunction of one or more instructions. If the transformed rule has no disjunctions in its elements, then it is an atomic rule according to the definition. Otherwise, go to the next step.
- **Step 3** Divide \( E_i \) into a set of atomic rules whose premises and actions are the obtained disjuncts in Step 2.

The normalizations of above four rule types are:

**Normalization 1** Since rules type 1 has no disjuncts in its premises, its normalization can be done directly by executing Step 3.

\[ \text{ON AND}(E_1, E_2, \ldots, E_n) \]
\[ \text{IF AND}(C_1, C_2, \ldots, C_m) \]
\[ \text{THEN A}_1 \]

**Normalization 2** Rules type 2 can be normalized into a set of rules with the following form:

\[ \text{ON AND}(E_1, E_2, \ldots, E_n) \]
\[ \text{IF AND}(C_1, C_2, \ldots, C_m) \]
\[ \text{THEN A}_k \]

For rule types 2, 3 and 4, we consider a simple case in which their action part is primitive (i.e., \( A_j \)). For conjunctive actions, we only need one more step to divide the conjunctive action like Normalization 1.

**Normalization 3** Rules type 3 are normalized into the following rules:

\[ \text{ON E}_1 \]
\[ \text{IF AND}(C_1, C_2, \ldots, C_m) \]
\[ \text{THEN A}_i \]

\[ \text{ON OR}(E_1, E_2, \ldots, E_n) \]
\[ \text{IF AND}(C_1, C_2, \ldots, C_m) \]
\[ \text{THEN A}_1 \]
Table I. Matching between events/actions and places

<table>
<thead>
<tr>
<th>Place</th>
<th>Event/Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0</td>
<td>NobodyIsAtHome</td>
</tr>
<tr>
<td>E1</td>
<td>ThereAreAppliancesOn</td>
</tr>
<tr>
<td></td>
<td>and(NobodyIsAtHome,</td>
</tr>
<tr>
<td></td>
<td>ThereAreAppliancesOn)</td>
</tr>
<tr>
<td>E2</td>
<td>EntersPersonInTheRoom</td>
</tr>
<tr>
<td>E4</td>
<td>DetectingMovement-</td>
</tr>
<tr>
<td></td>
<td>InFrontOfHandPaperSensor</td>
</tr>
<tr>
<td>E5</td>
<td>SunlightIntensityChanges</td>
</tr>
<tr>
<td>E6</td>
<td>TurnOffAppliances</td>
</tr>
<tr>
<td>E7</td>
<td>TurnOnTheLight</td>
</tr>
<tr>
<td>E8</td>
<td>DispatchHandPaper</td>
</tr>
<tr>
<td>E9</td>
<td>IncreaseLightIntensity</td>
</tr>
<tr>
<td>E10</td>
<td>TurnOffTheLight</td>
</tr>
</tbody>
</table>

C. Verification

Structural errors in a rule base can be reflected by special patterns in the CCPN model. Here we only analyze those CCPN structures related with incompleteness and inconsistency.

1) CCPN structure of the errors:

a) Isolated rules: According to Definition 3, an isolated rule doesn’t interact with any other rule, then in the CCPN a rule of this kind is characterized by a rule typed transition whose input (output) place doesn’t have input (output) arcs as Figure 1(a) shows.

b) Conflicting rules: Figure 1(b) shows the pattern for conflicting rules. Whenever conditions stored in transition T1 and T2 be the same, the rules will be in conflict according to Definition 1.

2) Verification: Since there is a correspondence between a rule base and its CCPN model we can develop an effective error detection procedure based on analyzing its CCPN model and the information provided by the following methods, such information is generated during CCPN construction.

- **InputPlaces(T)**. It returns the input primitive places of transition t. We don’t use the notation ‘t’ for this method because we want to know the primitive input places of a given transition. Taking into account CCPN of Figure 2, when we call getInputPlaces(T1) we get the set {E0,E1}

- **Place(A)**. It returns the place in which (event) action A is stored. For example, place(NobodyIsAtHome)=E0.

- **Condition(T)**. It returns the conditional clauses stored in the transition T. This procedure is only useful for rule typed transitions.

Incompleteness

An isolated rule is characterized by a transition t4 that its input and out places has no more connection with any part of the CCPN model, i.e., \( Vp \in InputPlaces(t4) \), \( p = \emptyset \) and \( Vp \in t4^* \), \( p^* = \emptyset \). It is easy to find that transition T3 and T4 fulfill this condition, so they represent isolated rules, i.e., Rule 3 and Rule 4 are isolated rules.
Inconsistency

In order to identify conflicting rules, user should define what are conflicting actions Firstly. In this example, we suppose that the following actions: TurnOffTheLight and TurnOnTheLight are contradictory each other.

Firstly, we identify those places in which the possible conflicting actions are stored, i.e., $\text{Place} \{\text{TurnOffTheLight}\} = E_{13}$, $\text{Place} \{\text{TurnOnTheLight}\} = E_{7}$.

Secondly, for each one of these places we get its input transitions, i.e., $*E_{15} = \{T_{7}\}$, $*E_{7} = \{T_{2}, T_{5}\}$.

Let's examine transitions $T_{7}$ and $T_{2}$. $\text{InputPlaces}(T_{7}) = \{E_{15}\} = E_{0}$ but, as you can see from Table 1, $\text{Condition}(T_{7}) \land \text{Condition}(T_{2}) = \emptyset$, so those transitions don't represent conflicting rules. Now, let's analyze transitions $T_{7}$ and $T_{5}$, since $\text{InputPlaces}(T_{7}) = \{E_{15}\} = E_{0}$ and from Table 1 we know that $\text{Condition}(T_{7}) = \text{Condition}(T_{5}) = \text{TimeElapsed} > 15$, then, according to Definition 1, those transitions represent conflicting rules, i.e., Rule 5 and Rule 6 are conflicting rules.

Once errors have been detected, the active rule designer must take corrective actions to prevent an abnormal system behavior. In this case, no automatic corrections can be done, all the detected errors must be corrected by using the smart homes requirements document or by asking experts for information. Other errors such as redundance can be corrected automatically.

V. CONCLUSION

In this paper we tackled active rule base verification issue. We define the errors corresponding to similar conceptions used in production rule base. Rule base verification is done through checking the corresponding CCPN structure of the errors. Unlike existing Petri net based verification approaches on production rule base, our method has the advantage that it is independent of the initial marking of CCPN. However, even though CCPN graphical representation simplifies the active rule base error identification by means of anomalous structures detection, it is still an arduous work since users have to inspect the CCPN and manually discover those abnormal patterns. In the future, we will take advantage of the incidence matrix representation to simplify the detection work.

REFERENCES