A Formal Description Technique for Verifying Redundancy and Subsumption in Expert Systems

James (N. K.) Liu
Department of Computing, Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
e-mail: csnkliu@comp.polyu.edu.hk

Abstract

Knowledge base maintenance is considered to be an important step towards the development of expert systems. The effectiveness of the verification process depends on the use of sound methodology which purports to model the inter-relationships in the knowledge and allows for a means of checking and uncovering problems inherent in the knowledge base. This paper initiates a formal description technique for verifying the redundancy and subsumption of production rule-based systems. The approach is based on the notion of High Level Petri Nets. It involves the use of coloured tokens for representing the two-folded states of predicates and the control notion for rule execution. In addition, an mechanism is provided for the maintenance of the predicate states during the inference process. The verification strategy is to put emphasis on the detection and identification of different anomalies relevant to such problems that could occur in sequences of inferences. Formal verification is provided which is based on reachability markings generated by the transition firings in the Petri network. A detailed analysis of the complexity of the methodology is given.

Keywords: Knowledge verification, redundancy and subsumption, formal description techniques, Petri Nets.

1. Introduction

Redundancy and subsumption are among the structural problems in rule based systems, that greatly affect knowledge base maintenance. The presence of these kinds of problems, although do not alter in general the semantics of the knowledge base and may be useful because of noise in the system, can at times introduce deficiencies like inconsistency, multiple counting of weight, computational deficiency, etc [12]. Importantly, the system designer ought to be aware of where these problems might occur. It is increasingly critical that such problems become the subjects of knowledge base verification affecting the expert system performance.

The traditional approach, using a pairwise comparison of the entities involved in rules, requires effort that is a polynomial in the order of the square of the number of entities. This situation will become worsen when one includes the dynamic inference among the rules, particularly when the series of chained inference of rules is long and possibly meshed. As a result, the verification will easily be subject to computational complexity problem.

This paper adopts Petri net paradigm for solving the verification problems. The primary reason is that it can offer a formal description technique for modelling predicate inference in an expert system, and it tends to satisfy most of the requirements for knowledge representation as follows:
- extensive literature about net theory,
- its ability to describe systems at different levels,
- its graphical and precise nature,
- its structural generality,
- its potential to support knowledge inference,
- its ability to represent concurrency,
- the existence of analytical verification tools.

There were some previous attempts [9, 11] either directly or indirectly, using Petri nets to represent knowledge and model system behaviour, however, not much effort has been devoted to the verification issues. Even if it were, it was subject to many criticisms [6].

Recently, [4-6] have developed a State Controlled Petri Net (SCPN) model for verifying knowledge bases in expert systems. The model can offer a formal description technique for simulating predicate inference in an expert system. In the present paper, we will give a formal description of the model, examine the transition sequences and check against the properties of SCPNs and its complexity for knowledge base verification. The paper is organized in five main sections. The first section gives a brief review of SCPNs. Knowledge inference and reasoning are discussed in the next section. It gives the problem description and formulation of the anomalies in a knowledge base. The third section provides cases of anomalies represented in SCPNs, and details the formal approach to the verification of redundancy and subsumption. Complexity analysis of the SCPN methodology is given in the fourth section. The paper concludes with a discussion on the extension of the methodology.

2. State Controlled Petri Net (SCPN) Model

In the context of using State Controlled Petri Nets to model the knowledge base of an expert system, we have to be aware of what forms of knowledge needed to
represent. Basically, a knowledge base includes fact knowledge which refers to what has been explicitly specified or known to be true, and inference knowledge which may specify a certain relationship, such as the casual-effect relationship, semantic relationship, functional relationship, etc., among the objects, from which new fact knowledge can be derived.

Besides, a fact could be a temporal dependent fact or a permanent fact. This could be true over a specific period of time in a temporal situation, or it represents knowledge that is always true over the entire span of the model’s life. In the present paper, only the permanent fact is taken into consideration.

The formulation of a fact as being asserted in a SCPN involves the marking of each place associated with the predicate. On the other hand, the formulation of a production system involves the representation of each production rule as a transition. There are certain constraints placed on the input and output functions for proper interpretation for any predicate being initiated and maintained through any associated transition firing.

2.1 SCPN Notations

Production systems use collection of rules to deduce facts and solve problems. The rules are composed of simple predicates which are used as functions that map object arguments into TRUE, FALSE values represented by binary truth-values 1, 0 respectively. These are then modelled by SCPNs.

In its most primitive form in the Petri net representation, each rule will have all the conditions as input places and the actions as output places for the transition in question. The logical predicate becomes true by the presence of a state token, and the transition associated with this predicate will become active by the presence of a control token. It should be noted that the predicate terms or their attribute values could be a symbol string, a numerical value with semantic meaning, or a null value.

A syntactical representation of the structural and non-structural information of a simple rule is given in Figure 1. In this SCPN, the places represent predicates $p_1, p_2$ and are named $s_1, s_2$ respectively. The transition represents the rule and is named $t_1$. The underlying places, transitions and arcs constitute a directed net as in place/transition nets. A notable difference is the output arc (from the transition back to $s_1$) which creates a self-loop for preserving the state of predicate. It also allows for its changes under some kind of complexity.

As shown in Figure 1(a), the input condition is denoted by the label associated with the arc where "Y" implies the requirement of "y" and "e" but not "n" tokens at the corresponding input place $s_1$ to enable the transition. When the transition fires the following events occur indivisibly and concurrently: for the input place, tokens are removed from the input place but the state token that represents the state of the place (or predicate) will be returned via the self-loop arc; for the output

2.2 Formal Definitions

The basic structure of SCPN consists of a set of predicate places, a set of transitions and a set of directed arcs, which connect the transitions and the predicate places. The formal definition of SCPNs is as follows.

**Definition 2.1** An SCPN structure $N$ can be defined as a 8-tuple given by $N = (P, T, I_c, O_c, I_s, O_s, M_e, M_s)$, where for $n > 0$, $m > 0$,

- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of predicates,
- $T = \{t_1, t_2, \ldots, t_m\}$ is a finite set of transitions,
- $P$ and $T$ are disjoint such that $P \cap T = \emptyset$,
- $I_c : T \Rightarrow P'$ is an input control function, a mapping from transitions to the bags of predicates,
- $O_c : T \Rightarrow P'$ is an output control function, a mapping from transitions to the bags of predicates,
- $I_s : T \Rightarrow P'$ is an input state function, a mapping from transitions to the bags of predicates,
- $O_s : T \Rightarrow P'$ is an output state function, a mapping from transitions to the bags of predicates,
- $M_e : P \Rightarrow Z^+$ is a control marking, a mapping from predicates to non-negative integers in $Z^+ = \{0, 1, 2, \ldots\}$,
- $M_s : P \Rightarrow Z^+$ is a state marking, a mapping from predicates to non-negative integers in $Z^+ = \{0, 1, 2, \ldots\}$.

and for each transition $t_j \in T$ in a net $N$,

$$I_s(t_j) \land O_c(t_j) \neq \emptyset, \quad I_c(t_j) \land O_c(t_j) = \emptyset,$$

such that $p_i \in I_s(t_j) \Rightarrow p_i \in O_c(t_j)$,

$p_i \in I_s(t_j) \Rightarrow p_i \not\in O_c(t_j)$.

SCPNs can be considered as a structurally folded version of a regular Petri net for a finite number of the types of tokens. Thus, a SCPN can be unfolded into a regular Petri net by unfolding each predicate $p_i$ into a set of predicates $\{p_i|p_i\}$, one for each type of tokens.

---

1 Bags ($P'$) are generalizations of sets that unlike sets allow the possibility of multiple occurrences of elements (predicates).
which the predicate place may hold, and by unfolding each transition \( t_j \) into a set of transitions \( \{ t_{cj}, t_{dj} \} \), one for each way that \( t_j \) may fire.

Therefore, an arc directed from a predicate \( p_i \) to a transition \( t_j \) defines that \( p_{ci} \) to be an input predicate of transition \( t_{cj} \) and \( p_{si} \) to be an input predicate of transition \( t_{dj} \) in an unfolded version of SCPN. However, an arc directed from transition \( t_j \) to a predicate \( p_i \) indicates that \( p_{ci} \) is an output predicate of transition \( t_{cj} \) if \( p_{ci} \) is not an input predicate of \( t_{cj} \), and \( p_{si} \) is an output predicate of transition \( t_{dj} \).

Note that the Input State \( (I_s) \) and Input Control \( (I_c) \) correspond to the input conditions, that is, the presence of a state token, or a control token, or both in the input places associated with the specific transition. Similarly, the Output State \( (O_s) \) and Output Control \( (O_c) \) govern the output, that is, the creation of a state token, or a control token, or both in the output places associated with the transition.

**Definition 2.2** A marking \( M \) of a net \( N \) is a function from the set of predicates \( P \) to the non-negative integers \( Z^+ \):

\[
M : P \rightarrow Z^+
\]

\( M \) is also denoted by an \( n \)-vector such that,

\[
M = \{M_1, M_2, \ldots, M_n\}
\]

where \( n = |P| \) and \( M_i \in Z^+ \) for \( i = \{1, 2, \ldots, n\} \).

**Definition 2.3** A transition \( t_j \in T \) in a marked \( N \) with marking \( M \) is enabled if

\[
M_i (p_i) \geq \#(p_i, I_c(t_j)) \geq 1,
\]

\[
M_i (p_i) \geq \#(p_i, I_s(t_j)) \geq 1,
\]

and

\[
M = (M_c, M_s),
\]

where \( p_i \in P_\text{c} \) and \( \#(p_i, I_c(t_j)), \#(p_i, I_s(t_j)) \) specify the input multiplicity of that predicate for the transition. The effect of firing a transition is to remove all tokens from an input place and deposit them in output places in accordance with respective input and output functions.

**Definition 2.4** A new marking \( M'_c \), as a result of the firing of a transition \( t_j \in T \) in a marked \( N \) with marking \( M \), is created if

\[
M'_c(p_i) = M_c(p_i) - \#(p_i, I_c(t_j)) = 0 \quad \text{if} \quad p_i \in I_c(t_j),
\]

\[
M'_c(p_i) = M_c(p_i) - \#(p_i, I_s(t_j)) + \#(p_i, O_s(t_j)) = 1 \quad \text{if} \quad p_i \in I_s(t_j) \cap O_s(t_j),
\]

\[
M'_c(p_i) = M_c(p_i) + \#(p_i, O_c(t_j)) \quad \text{if} \quad p_i \in O_s(t_j),
\]

and

\[
M' = (M'_c, M'_s),
\]

where \( p_i \in P_\text{c} \), and \( \#(p_i, O_c(t_j)), \#(p_i, O_s(t_j)) \) specify the output multiplicity of that predicate for the transition.

**Definition 2.5** Given \( K^n \) as a set of all markings for the places in \( N \), the next state function \( \delta : K^n \times T \rightarrow K^n \) is defined as \( \delta(M(t)) = M' \) for some \( t_j \in T \) iff

1. transition \( t_j \) is enabled,

2. a new marking \( M' \) is created when \( t_j \) fired.

**Definition 2.6** A marking \( M^0 \) is said to be reachable from a marking \( M^0 \) if there exists a sequence of firings that transforms \( M^0 \) to \( M^0 \). A firing sequence is denoted by \( \sigma = (t_1, t_2, \ldots, t_n) \) that \( M^0 \) is reachable from \( M^0 \) by \( \sigma \).

### 2.3 Transition Criteria

The logical predicate becomes true by the presence of a state token, and the transition associated with this predicate will become active by the presence of an execution token (i.e. a simple control token). When both tokens are present and provided that the state token satisfies the transition condition, the transition is enabled and is ready for firing. For simplicity reason, without taking any transition conditions or transition operations into consideration, we can minimally enable a specific transition and then check the reachability set for any irregularities of predicate places. In SCPNs, a marking \( M_s \) is composed of \( M_s = \{M_{s1}, M_{s2}, \ldots, M_{sn}\} \) that depicts the marking for the state places and \( M_k = \{M_{k1}, M_{k2}, \ldots, M_{kn}\} \) that depicts the marking for the control places. Individual markings \( M_{sj}, M_{ki} \) are described as space vectors spanned by the number of places in the SCPNs. Similarly, individual \( t_j \) is represented by a space vector spanned by the number of transitions in the SCPNs. Firstly, we define that:

**Definition 2.7** A transition \( t_j \) is minimally active if

\[
M_{ci} = \begin{cases} 1 & \text{if } p_{ci} \in I_c(t_j) \\ 0 & \text{otherwise} \end{cases}
\]

**Definition 2.8** A transition \( t_j \) is minimally enabled if \( t_j \) is both minimally active and that

\[
M_{si} = \begin{cases} 1 & \text{if } p_{si} \in I_s(t_j) \\ 0 & \text{otherwise} \end{cases}
\]

Since different types of token are used to represent the two-folded states of the predicate, we shall use \( M^+_c \) to depict the state marking for the assertion of \( p_i \) and similarly, \( M^+_s \) for the denial of \( p_i \). The verification of a sequence of certain rules using SCPNs requires some extension of the definitions. We define that:

**Definition 2.9** \( T_k \) that contains a group of transitions \( \{t_k\} \) is said to be minimally active if \( \forall j \in 1, 2, \ldots, n, j \neq k, \exists p_i \in I_c(t_j) \subseteq I_c(T_k), \exists p_i \in I_s(t_j) \subseteq I_s(T_k), \exists p_i \in O_s(t_j) \subseteq O_s(T_k), \exists p_i \in O_c(t_j) \subseteq O_c(T_k), \exists p_i \in O_c(t_j) \subseteq O_c(T_k) \) s.t.

\[
M_{ci} = \begin{cases} 1 & \text{if } p_{ci} \in I_c(t_j) \text{ and } p_{ci} \in O_c(t_j) \\ 0 & \text{otherwise} \end{cases}
\]

Note that for a transition to be active, there is no need to have all its associated input places marked with a control token, respectively. The principle is that as long as the input states of the associated predicates are defined (i.e. marked with state tokens), any inference due to other transition that deposits a control token (as well as the same state token) in at least one of these input places, will enable such transition and cause it to be fired. Further, the self-loop arc corresponding to each input place does not cause a repeated firing of transitions. In the absence of any self reference rule, the set of input and output places relevant to the transition in SCPN are always disjointed.

**Definition 2.10** \( T_k \) that contains a group of transitions \( \{t_k\} \) is said to be minimally enabled if \( \forall j =
1, 2, ..., n, $t_j \in T_k$, $\exists p_i \in I_s(t_j) \subseteq I_s(T_k)$, s.t. $T_k$ is both minimally active and

$$M_{s_i} = \begin{cases} 
1 & \text{if } p_i \in I_s(t_j) \text{ where } M_k(p_{s_i}) = 1 \\
0 & \text{otherwise}
\end{cases}$$

Note that SCPNs contain self-loop arcs for each of the input places involved. Consequently, the set of input places and output places of any transition are no longer disjunctive with respect to the states of the predicates. In order for $T_k$ to be minimally enabled, we have to use the control places which are marked to make $T_k$ minimally active, as the indicators, so that the corresponding predicates can be marked (by the presence of state tokens) accordingly.

### 2.4 An Illustrative Example

Consider the following set of simple rules:

- Rule-1: $p_1 \rightarrow p_2$
- Rule-2: $p_3 \rightarrow \neg p_5$
- Rule-3: $p_2 \rightarrow p_3$
- Rule-4: $\neg p_4 \land p_3 \rightarrow p_0 \land \neg p_5$
- Rule-5: $p_1 \rightarrow p_3$

such that

- $\sigma$ = (Rule-1, Rule-3, Rule-2, Rule-4),
- $I_s$(Rule-1) = $\{s_1\}$, $I_o$(Rule-2) = $\{s_3\}$, $I_s$(Rule-3) = $\{s_2\}$,
- $I_o$(Rule-4) = $\{s_5,s_3\}$, $I_o$(Rule-5) = $\{s_1\}$,
- $O_s$(Rule-1) = $\{s_2\}$, $O_o$(Rule-2) = $\{s_3\}$,
- $O_s$(Rule-3) = $\{s_3,s_5\}$, $O_o$(Rule-4) = $\{s_2,s_5\}$,
- $O_o$(Rule-5) = $\{s_1\}$.

The rules are represented by a SCPN graph as shown in Figure 2. Note that for simplicity, the self-loop associated with each input place is not shown in the net. In the case of the rule with conjunctive condition, "$\land$" or "$\lor$" is used for labelling the associated input arcs to indicate that a "$\land$" or "$\lor$" mutually exclusive token, and at least a "e" token at either input place will be sufficient to enable the transition.

The SCPN model can provide dynamic transition of knowledge inference in deterministic situations. As given in the example, the sequence of transitions, $\sigma$, can be minimally set active by having its places $s_1, s_4$ marked with a control token, respectively. Rule-4 will become active as long as $s_3$ or $s_5$ has a control token in place. This is the condition defined for rules that involve conjunction of predicates. Without loss of generality, we can mark the corresponding places with these control tokens for the purpose of initializing a sequence of transitions according to Definition 2.9.

Since $I_s(t_i) \cap O_o(t_i) \neq \emptyset$ for any $t_i \in \sigma$ due to the existence of self-loops in SCPNs, we will initialize those places that already have had a control token there according to Definition 2.10. As a result, $s_1, s_4$ will be marked with an appropriate state token, respectively, that satisfy the conditions for the transitions.

### 3. Knowledge Inference in SCPN Model

The reasoning strategy for knowledge inference is sometimes called "event-driven" reasoning, since the reasoning process is based on the occurrences of events. The goal of the reasoning for SCPN is to determine the subsequent events based on current events. The initial marking of the net determines the initial state of the system. Subsequent markings contribute to a reachability set which can reflect the degree of inference at different levels, stemming from the event.

Given an event in terms of an initial marking $M^0$ in a SCPN system, a set $T$ of active transitions would then be identified. While $T$ is not empty, we select a transition $t$ from $T$ for execution. When firing a transition $t$, all state tokens in the input places will be destroyed, and the number of state tokens in the output places $s$ increased by 1. The presence of the self-loop associated with any input place will put back a single state token in the original input place, accordingly.

The execution of all transitions in $T$, individually, will create a series of alternative new markings ($M_1, M_2, ..., M_n$). The effect of adding a "e" token in places $s_i$ associated with $t$ is to make the transition become active for firing once the states of $p_i$ are asserted. The process is repeated recursively for ($M_1, M_2, ..., M_n$), and so on. In general, given $M^0$, a sequence of a list of markings results in Figure 3.

Note that the transitions of the SCPN model are structured to be in one direction only with the exception for the self-loop that is associated with each input predicate. Whenever a transition $t$ representing a rule, $p \rightarrow q$, becomes active and its associated predicate $p$ is asserted, it is forward enabled meaning that the
inference proceeds in a forward direction with \( q \) inferred (i.e. \( q \) is asserted) subsequently, thus supporting a forward chaining (events driven) paradigm of inference. In order to model and allow for simulating the goal driven reasoning, the backward enabled transition will have to be introduced. It is defined that a transition \( t \) for \( p \to \neg q \) is backward enabled if its inference proceeds in a backward direction as if it were for \( p \to q \). Input places and output places are interchanged accordingly to accomplish the changes.

\[
M^0 = (M_1, M_2, \ldots, M_{i_1}, \ldots, M_{m_j}) \quad | \quad (\ldots, \ldots, M_{i_2}, \ldots, \ldots) \\
(\ldots, \ldots, M_{i_1}, \ldots, \ldots) \\
(\ldots, \ldots, M_{i_2}, \ldots, \ldots) \\
(\ldots, \ldots, M_{i_1}, \ldots, \ldots) \\
(\ldots, \ldots, M_{i_2}, \ldots, \ldots) \\
\ldots
\]

so on

**FIGURE 3. A sequence of markings reachable from \( M^0 \)**

### 3.1 A Taxonomy of Anomalies

Many of the problems encountered in the manipulation of an expert system are caused by a knowledge base that is redundant and subsumed. However, these problems vary in different forms under different situations. A formal expression of the problems is as follow.

Given that in a closed world situation in which a common concept is derived by a rule set, \( \{P, Q, R, S, \ldots\} \) is a set of simple predicate formulas (atomic formulas). The \( <\text{condition}> \) and \( <\text{action}> \) of the rules are represented by these formulas. For simplicity, the terms of these predicates, which might be variables, constants or else, are not specified or explicitly shown in the formulation. As such, the anomalies as classified by [5], that are relevant to the problems of redundancy and subsumption in knowledge base verification, take the following forms:

**REDUNDANCY**

**Case 1. Conditions and actions identical**

\( P_1 \to Q_1, P_2 \to Q_2 \) where \( P_1 = P_2, Q_1 = Q_2 \)

**Case 2. Chained inference**

\( P \to Q \to R, Q \to R \) and

\( Q \) is not ascertainable through other rules

**SUBSUMPTION**

**Case 1. Conditions subsumed with identical actions**

\( P \land Q \to R \land S, Q \to R \land S \)

**Case 2. Conditions identical with subsumed actions**

\( P \land Q \to R \land S, P \land Q \to R \)

**Case 3. Conditions and actions subsumed**

\( P \land Q \to R \land S, Q \to R \)

### 3.2 An Example

Consider the same example as given in Section 2, since coloured tokens have been used to represent the predicate state and control state involved in the transitions, we can define the marking \( M_i \) which is equal to \( (M_{si, M_{sp}}) \) at any place \( s_i \) as a 3-tuple array of positive integer values where the first tuple in the array represents the number of control tokens, and the second and third tuples represent the number of state tokens corresponding to predicate \( p_i \) and its denial \( \neg p_i \), respectively.

Suppose the knowledge base contains two factual knowledge, \( p_1 \) and \( \neg p_4 \). In SCPN representation, \( s_1 \) and \( s_4 \) are to be marked initially with a "y" and a "e" tokens, a "n" and a "e" tokens, respectively. This marking is:

\[
M^0 = [(101)(000)(000)(011)(000)(000)]
\]

Knowledge inference can be obtained by examining the reachability tree and the subsequent markings. In particular, alternative paths leading to a marking generated from one path, which marks all the predicate places common to that of the other one, imply the possible existence of redundant knowledge. As shown in Figure 4, paths (Rule-5), (Rule-1 Rule-3) induce common marked places, and both generate a new marking \( M_3 = (101) \) at the place \( s_3 \), as part of:

\[
M^2 = [(100)(000)(101)(011)(000)(000)] \quad \text{and} \quad M^3 = [(100)(100)(101)(111)(000)(000)]
\]

Therefore, Rule-5 could be redundant among the rest of the rules in the knowledge base.

**FIGURE 4. Reachability tree spanned by \( M^0 \)**

Besides, a path leading to a multiple marking reflects the possible existence of subsumption. As shown in
Figure 4, a path (Rule-2, Rule-4) produces a multiple marking $M_5 = (022)$ at $s_5$, forming part of

$M_8 = [(100)(000)(100)(010)(022)(101)]$ and

$M_9 = [(100)(100)(100)(010)(022)(101)]$ alternatively.

Consequently, Rule-2 subsumes Rule-4 which has additional constraints attached to an additional input predicate place.

4. Verification of Redundancy and Subsumption

The problems of redundancy and subsumption are observable either between a pair of rules or a set of rules that represent chains of inference, a description of such in terms of predicate formulas is given in previous section. We identify a number of cases represented in SCPNs as follows.

4.1 Cases of Redundancy in SCPNs

When sequences of transitions interact with one another, a number of possible events could take place. As shown, sequences $(t_0)$ and $(t_1)$, $(t_0)$ and $(t_1, t_2, ..., t_m)$, $(t_1, t_2, ..., t_m)$ and $(t_{m+1}, t_{m+2}, ..., t_{m+n})$ could infer same result in Case 1, Case 2 and Case 3 (shown in Figure 5) respectively.

4.2 Cases of Subsumption in SCPNs

Sequence $(t_0)$ could subsume $(t_1)$ in Case 1, Case 2, Case 3 and $(t_1, t_2, ..., t_m)$ in Case 4, Case 5 (shown in Figure 6) respectively. Similarly, $(t_1, t_2, ..., t_m)$ could subsume $(t_{m+1}, t_{m+2}, ..., t_{m+n})$ as shown in Case 6 (in Figure 6).

FIGURE 5. SCPNs showing cases of redundancy

FIGURE 6. SCPNs showing cases of subsumption
might cause cyclicity), \( \exists \sigma_j, \exists k, \text{s.t. these sequences have the following properties:} \)

\[
\begin{align*}
\sigma_1 \land \sigma_j &= \emptyset; \\
M' &= \delta(M, \sigma_1), M'' = \delta(M, \sigma_j); M_{sk} = 0, M'_{sk} > 0, M''_{sk} > 0.
\end{align*}
\]

The proposition has been derived for verifying possible cases of redundancy and subsumption in rule based systems. As shown in the above sections, the cases include:

1. The redundant rules have identical conditions and actions.
2. The redundant rules have chained inference involving a chained rule and a single rule, or both chained.
3. The rules have subsumed conditions but identical actions.
4. The rules have identical conditions but subsumed actions.
5. The rules have subsumed conditions and actions.
6. Chained rules with subsumption.

A brief derivation of the proposition is given in Appendix I and, with more details, in [4].

5. Complexity Analysis of SCPNs

The complexity of verifying the anomalies in knowledge base is defined to include the effort to transform the rules into transitions, to derive the reachability tree through a series of numerical computation, and to check different solutions (token numbers for each type at each token place corresponding to any reachable marking) for error examination.

5.1 Transformation of Rules to SCPN

Let the knowledge base have \( k \) rules, each with \( u \) conditions and \( v \) actions.

It is required to create transitions to match the rules. There can be a maximum of \( k(u+v) \) predicate places representing \( 3k(u+v) \) possible token facts (depicted by the presence of state and control tokens), and \( k+c \) transitions representing rules where \( c \) is the extra number of transitions created (as a result of possible indeterminate rules or rules with conjunctive conditions that only one control token in any one of the associated input places is sufficient to cause the transition become active) in the knowledge base. It is noted that \( c = 0 \), if there exists none of this type of rule explicitly in the rule set. However, rules of this nature may exist implicitly in the knowledge base, presenting inter-related properties of redundancy and subsumption.

The transition sequence, \( \sigma \), will be represented by an \( n \)-vector where \( n \) is the number of transitions in SCPN. \( n \) is derived through the transformation by \( n = (k+c) \).
Let $3m$ denote the number of token facts, with $3m \leq 3k(u+v)$. The total number of storage places for the computation, therefore, will be

$$\kappa = 3m + n.$$

Note that more storage places will be needed if any additional transition conditions and operations are included for the SCPN simulation.

### 5.2 Derivation of Reachability Tree

A numerical analysis of the SCPN approach could be based on a matrix view of Petri nets [1, 7]. This is coupled by some heuristic operations to help simplify the mathematics involved in matrix calculation. Two matrices $D^-, D^+$, are defined to represent the input and output functions for the predicate states and control states, respectively. Each matrix is $m$ rows (one for each token place) by $n$ columns (one for each transition).

A typical example shown in Figure 2 is that, there are 6 transitions transformed from a set of 5 rules, involving 6 predicate places (i.e. $c = 1$). Note that Rule-44 due to the control options as the result of the conjunctive rule Rule-4 is not shown in the figure.

The $D^-$ of the SCPN then is:

<table>
<thead>
<tr>
<th>Rule-1</th>
<th>Rule-2</th>
<th>Rule-3</th>
<th>Rule-4</th>
<th>Rule-44</th>
<th>Rule-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$s_1$</td>
<td>$s_1$</td>
<td>$s_1$</td>
<td>$s_1$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$(1\ 0\ 1)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(1\ 0\ 1)$</td>
<td></td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_2$</td>
<td>$s_2$</td>
<td>$s_2$</td>
<td>$s_2$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td></td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$(0\ 0\ 0)$</td>
<td>$(1\ 0\ 1)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td></td>
</tr>
<tr>
<td>$s_4$</td>
<td>$s_4$</td>
<td>$s_4$</td>
<td>$s_4$</td>
<td>$s_4$</td>
<td>$s_4$</td>
</tr>
<tr>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td></td>
</tr>
<tr>
<td>$s_5$</td>
<td>$s_5$</td>
<td>$s_5$</td>
<td>$s_5$</td>
<td>$s_5$</td>
<td>$s_5$</td>
</tr>
<tr>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td></td>
</tr>
<tr>
<td>$s_6$</td>
<td>$s_6$</td>
<td>$s_6$</td>
<td>$s_6$</td>
<td>$s_6$</td>
<td>$s_6$</td>
</tr>
<tr>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td></td>
</tr>
</tbody>
</table>

The $D^+$ of the SCPN is:

<table>
<thead>
<tr>
<th>Rule-1</th>
<th>Rule-2</th>
<th>Rule-3</th>
<th>Rule-4</th>
<th>Rule-44</th>
<th>Rule-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$s_1$</td>
<td>$s_1$</td>
<td>$s_1$</td>
<td>$s_1$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$(1\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(1\ 0\ 0)$</td>
<td></td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_2$</td>
<td>$s_2$</td>
<td>$s_2$</td>
<td>$s_2$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$(1\ 0\ 1)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(1\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td></td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$(0\ 0\ 0)$</td>
<td>$(1\ 0\ 1)$</td>
<td>$(1\ 0\ 0)$</td>
<td>$(1\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td></td>
</tr>
<tr>
<td>$s_4$</td>
<td>$s_4$</td>
<td>$s_4$</td>
<td>$s_4$</td>
<td>$s_4$</td>
<td>$s_4$</td>
</tr>
<tr>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td></td>
</tr>
<tr>
<td>$s_5$</td>
<td>$s_5$</td>
<td>$s_5$</td>
<td>$s_5$</td>
<td>$s_5$</td>
<td>$s_5$</td>
</tr>
<tr>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 1\ 1)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td></td>
</tr>
<tr>
<td>$s_6$</td>
<td>$s_6$</td>
<td>$s_6$</td>
<td>$s_6$</td>
<td>$s_6$</td>
<td>$s_6$</td>
</tr>
<tr>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 0\ 0)$</td>
<td>$(0\ 1\ 1)$</td>
<td>$(0\ 0\ 0)$</td>
<td></td>
</tr>
</tbody>
</table>

The formulation of propositions and the simulation of the SCPN reachability tree allow for the inspection of transition sequences that might indicate any anomalies in the rule set.

Let $e[j]$ be the $n$-vector. It is zero everywhere except in the $j$th component which is an $3x3$ identity matrix. The transition $t_j$ is represented by the $n$-vector $e[j]$. $t_j$ is enabled in a marking except in the $j$th component which is an $3x3$ identity matrix. The transition $t_j$ is represented by the $n$-vector $e[j]$.

The result of firing $t_j$ in marking $M^0$, if it is (minimally) enabled, is

$$\delta(M^0,t_j) = M^0 \circ (D^+ \bullet e[T[j]]) = M^0 \circ A_j + D^+ \bullet e[T[j]] = M^1 \quad \text{(5.1)}$$

where $\circ$ represents a heuristic operation to reset any row component which is non-zero in the whole array in $A_j$. This operation will ensure that all corresponding input tokens that enable the particular transition will be consumed once it is fired.

Detection of any form of error will require the generation of a reachability tree for close examination. All markings that are reachable from a given marking will need to be stored for examination. As seen from equation 5.1, given an initial marking $M^0$, the effort, in a worst case, to derive the next marking will involve the following operations:

(a) Identify enabled transitions requires comparison between $M^0$ and every column of $D^-$: $3m$ comparisons.

(b) Compute $M^0 \circ A_j$: $3m$ comparisons.

(c) Compute $M^0 \circ A_j + D^+ \bullet e[T[j]]$: $3m$ additions.

(Note: $D^+ \bullet e[T[j]]$ refers to the $j$th column of $D^+$ without extra effort to go through the inner-product operations for the matrix multiplication).

Assume that $r_1$ transitions are enabled by $M^0$, where $r_1 \leq n$. Therefore at the next stage, $r_1$ transitions are to be checked respectively for possible new markings. The total number of operations will then be $3mr_1(3+n)$.

For the next stage, assume $r_2$ transitions are enabled by $M^1$, where $r_1 + r_2 \leq n$. The total number of operations will be $3mr_2(3+n)$.

The process repeats until all possible markings from $M^0$ have been exhausted. Provided that any existence of a loop is detected and its immediate markings not allowed to be spanned further, the upper bound for the process is given by:

$$3m(3+n)(r_1 + r_2 + \ldots) \leq 3m(3+n)n \quad \text{(for any distinctive transitions)} \leq 3n^3 \quad \text{(for sufficiently large n).}$$

Therefore, for a sequence of transition firings $\sigma = (t_1, t_2, \ldots, t_j)$, the total effort for finding the markings is $3m(3+n)$ operations and is $\leq 3n^3$, which is of $O(3n^3)$.

For a marking $M^0$, if $w$ distinct markings are reachable, the storage requirement will be at least $wK$ ($w = 3m+n$ as defined in Section 5.1), and possibly more, if a marking can be reached through a variety of paths.

### 5.3 Examination of Different Solutions

Detection of any form of error will require the examination of individual marking places along any possible paths generated by the reachability tree. All distinctive markings that are reachable from a given marking, corresponding to certain set of $\{l(t_i), r(t_i)\}$ or $\{O_g(t_i), O_o(t_i)\}$ for any $t_i$, will need to be checked respectively for any anomalies.

Since the number of token places is upper bounded by $3m \leq 3k(u+v)$. For each given $M^0$, the number of token places being marked is bounded by $3m$. For the reachability set $R(M^0)$, the number of distinctive
transition firings is bounded by \( n \). Consequently, the total number of times for checking individual token places in \( R(M^n) \) is bounded by \( 3mn \leq 3n^2 \).

Note that these estimates of complexity apply to one solution of the reachability set without linking initial state, subsequent states and the transition matrices.

6. CONCLUSION

A formal description technique based on high level Petri nets allows the use of reachability theory for exposing problems of redundancy and subsumption that could occur in a knowledge base. The verification is done exhaustively by minimally initiating the sequence of transitions and closely examining the reachability markings. In addition, the problems could be located through the trace of the sequences of transitions which might provide alternative or multiple marking effects. The complexity of the SCPNs performance is \( O(3n^3) \) where \( n \) is the number of rules involved in the knowledge base of the system. Though the proposed methodology is comparable to that of the theorem proving procedures which gave \( O(2^n) \) and network flow approach whose performance was reported to be between \( O(n^2) \) and \( O(n^4) \) by [8] it should be noted that the verification approach adopted in the present paper has consolidated some features offered by a number of previous verification strategies and allowed possible room for further extension. The technique given here can provide a means with similar complexity to verify other problems such as cyclicity, inconsistency, and completeness of the knowledge base in expert systems. Where uncertainty inference is considered, it is subject to increases in complexity. Not only the number and the type of tokens need to be checked at each transition firing, but also the relevance of the tokens need to be addressed, whose uncertainty measures hold a crucial factor for the propagation of inference activity. All of these extensions will be the subjects of future work.

REFERENCES

[8] Nazareth, D.J., "An Analysis of Techniques for Verification of Logical Correctness in Rule Based Systems", PhD Dissertation, Department of Managerial Studies, Case Western Reserve University, Cleveland, Ohio, 1988.

APENDIX I

**Proposition:** Given that \( M \) minimally enables a non-trivial transition sequence \( \sigma_i \), iff the rule set has incorrect rules that involve redundancy and subsumption, but not self-reference rules (i.e. rules that might cause cyclicity), \( \exists \sigma_j, \exists k \), s.t. these sequences have the following properties:

\[
\sigma_i \cap \sigma_j = \emptyset ;
M' = \delta(M, \sigma_i), M'' = \delta(M, \sigma_j) ;
M_{sk} = 0, M'_{sk} > 0, M''_{sk} > 0 .
\]

**Forward Case Proof:**

If there exists incorrect rules of the following cases:

Case-A1: The redundant rules have identical conditions and actions
In SCPN representation, there should exist \( t_0, t_1 \) such that
\[
I_g(t_0) = I_s(t_1), \quad O_s(t_0) = O_s(t_1).
\]
Choose \( M^0 \) s.t. \( t_0 \) is minimally enabled, then \( M^1 = I \)
if \( p_s \in I_s(t_0), 0 \) otherwise.
Since \( I_g(t_1) = I_s(t_1) \), \( t_1 \) is enabled. As from
Definition 2.5, \( \exists M^1 \), \( M^2 \) s.t. \( M^1 = \delta(M^0, t_0), \quad M^2 = \delta(M^1, t_1). \)
Therefore
\[
M^1_{sk} = \begin{cases} 
1 & \text{if } p_s \in I_s(t_0), \quad O_s(t_0) \\
0 & \text{otherwise}
\end{cases}, \\
M^2_{sk} = \begin{cases} 
1 & \text{if } p_s \in I_s(t_1), \quad O_s(t_1) \\
0 & \text{otherwise}
\end{cases}.
\]
Since \( O_s(t_0) = O_s(t_1) \), thus for \( p_s \in O_s(t_0), \quad M^1 = 0, \quad M^2 > 0 \), \( M^2_{sk} > 0 \), implying incorrectness, with \( \sigma_i = (t_0) \) and \( \sigma_j = (t_1) \).

Case-A2: The redundant rules have chained inference
(1) A chained rule vs a single rule
In SCPN representation, there should exist \( t_0 \) and \( \sigma_i = (t_1, t_2, \ldots, t_m) \) such that
\[
I_g(t_0) = I_s(t_1), \quad O_s(t_0) = O_s(t_1), \\
O_s(t_l) = I_s(t_{l+1}) \quad \text{for } l = 1, 2, \ldots, m-1. \]
Choose \( M \) s.t. \( \sigma_i = (t_1, t_2, \ldots, t_m) \) is minimally enabled, i.e. \( \forall l = 1, 2, \ldots, m-1 \),
\[
M^1 = \begin{cases} 
1 & \text{if } p_s \in I_s(t_l), \quad \text{where } \mu_l(p_s) = 1 \\
0 & \text{otherwise}
\end{cases}.
\]
The execution of transition sequence, \( \sigma_i \), gives \( M^1 \) s.t. \( \forall l = 1, 2, \ldots, m \), \( O_s(t_l) \in O_s(\sigma_j) \)
\[
M^1_{sk} = \begin{cases} 
1 & \text{if } p_s \in \{I_s(t_l),O_s(\sigma_j)\} \\
0 & \text{otherwise}
\end{cases}.
\]
Since \( I_g(t_1) = I_s(t_{m+1}), \quad \sigma_j \) is made enabled. Let \( M^2 = \delta(M^1, t_0), \quad \forall l = m+1, 2, \ldots, m+n-1 \), \( O_s(t_l) \in O_s(\sigma_j) \)
\[
M^2_{sk} = \begin{cases} 
1 & \text{if } p_s \in \{I_s(t_l),O_s(\sigma_j)\} \\
0 & \text{otherwise}
\end{cases}.
\]
Since \( O_s(t_m) \in O_s(\sigma_i), \quad O_s(t_{m+n}) \in O_s(\sigma_j), \quad O_s(t_m) = O_s(t_{m+n}) \), thus for \( p_s \in O_s(t_{m+n}), \quad M^1 = 0, \quad M^2 > 0, \quad M^2_{sk} > 0 \), implying incorrectness, with \( \sigma_i = (t_1, \ldots, t_m) \) and \( \sigma_j = (t_{m+1}, \ldots, t_{m+n}) \).

Case-B1: The rules have subsumed conditions but identical actions
In SCPN representation, there should exist \( t_0, t_1 \) such that
\[
I_g(t_0) \subseteq I_s(t_1), \quad O_s(t_0) = O_s(t_1).
\]
Choose \( M \) s.t. \( t_1 \), is minimally enabled, then \( M^1 = 1 \) if \( p_s \in I_s(t_1), 0 \) otherwise.
Let \( M' = \delta(M^1, t_0) \), thus
\[
M^1_{sk} = \begin{cases} 
1 & \text{if } p_s \in \{I_s(t_1),O_s(t_1)\} \\
0 & \text{otherwise}
\end{cases}.
\]
Since \( I_s(t_0) \subseteq I_s(t_1) \) and \( t_1 \) is enabled, \( t_0 \) will be enabled as well. Let \( M'' = \delta(M^1, t_0) \), and for \( O_s(t_1) = O_s(t_1) \)
\[
M^1_{sk} = \begin{cases} 
1 & \text{if } p_s \in \{I_s(t_0),O_s(t_1)\} \\
0 & \text{otherwise}
\end{cases}.
\]
Thus for \( p_s \in O_s(t_1), \quad M^1 = 0, \quad M^2 > 0, \quad M^2_{sk} > 0 \), implying incorrectness, with \( \sigma_i = (t_0) \) and \( \sigma_j = (t_1) \).

Case-B2: The rules have identical conditions but subsumed actions
In SCPN representation, there should exist \( t_0, t_1 \) such that
\[
I_g(t_0) = I_s(t_1), \quad O_s(t_0) \subseteq O_s(t_1).
\]
Choose \( M \) s.t. \( t_0 \) is minimally enabled, then \( M^1 = 1 \) if \( p_s \in I_s(t_0), 0 \) otherwise.
Let \( M' = \delta(M^1, t_0) \), thus
\[
M^1_{sk} = \begin{cases} 
1 & \text{if } p_s \in \{I_s(t_0),O_s(t_0)\} \\
0 & \text{otherwise}
\end{cases}.
\]
Since \( I_s(t_0) = I_s(t_1) \) and \( t_1 \) is enabled, \( t_1 \) will be enabled as well. Let \( M'' = \delta(M^1, t_0) \),
\[
M^1_{sk} = \begin{cases} 
1 & \text{if } p_s \in \{I_s(t_0),O_s(t_1)\} \\
0 & \text{otherwise}
\end{cases}.
\]
Since \( O_s(t_0) \subseteq O_s(t_1) \), thus for \( p_s \in O_s(t_0), \quad p_s \in O_s(t_1), \quad M^1 = 0, \quad M^2 > 0, \quad M^2_{sk} > 0 \), implying incorrectness, with \( \sigma_i = (t_0) \) and \( \sigma_j = (t_1) \).
Case-B3: The rules have subsumed conditions and actions

In SCPN representation, there should exist $t_0, t_1$ such that

$$I_s(t_0) \subseteq I_s(t_1), O_s(t_0) \subseteq O_s(t_1).$$

Choose $M$ s.t. $t_1$, is minimally enabled, then $M_{sk} = 1$ if $p_{sk} \in I_s(t_1)$, 0 otherwise.

Let $M' = \delta(M, t_1)$, thus

$$M'_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in \{I_s(t_1), O_s(t_1)\} \\ 0 & \text{otherwise} \end{cases}$$

Since $I_s(t_0) \subseteq I_s(t_1)$ and $t_1$ is enabled, $t_0$ will also be enabled. Let $M'' = \delta(M, t_0)$,

$$M''_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in \{I_s(t_0), O_s(t_0)\} \\ 0 & \text{otherwise} \end{cases}$$

Since $I_s(t_0) \subseteq I_s(t_1)$ and $O_s(t_0) \subseteq O_s(t_1)$, thus for $p_{sk} \in O_s(t_0), p_{sk} \in O_s(t_1), M_{sk} = 0, M'_{sk} > 0, M''_{sk} > 0$, implying incorrectness, with $\sigma_i = (t_0)$ and $\sigma_j = (t_1)$.

Case-B4: Chained rules with subsumption

This might involve a chained rule subsumed with a single rule or vice versa, or a chained rule subsumed with another chained rule. The incorrectness would create a number of situations for redundancy and subsumption as discussed in Cases A2 - B3. As a result, a similar approach that should combine the ideas for analysing individual situations could be used for proving the proposition. This is to verify the incorrectness possibly due to the chained inference in the knowledge base.

Converse Case Proof:

Given $M$ which minimally enables a transition sequence $\sigma_i$, and $\exists \sigma_j, \exists k, t_1 \cap \sigma_j = \emptyset$, $M' = \delta(M, t_1), M'' = \delta(M, t_2)$ with $M_{sk} = 0, M'_{sk} > 0, M''_{sk} > 0$, if $\sigma_i$ and $\sigma_j$ have the following cases:

Considering that $\sigma_i$ and $\sigma_j$ are non-trivial transition sequences, i.e. $\sigma_i \neq \emptyset$ and $\sigma_j \neq \emptyset$.

Case-1: $\sigma_j$ is composed of a single transition $t_1$

Since $t_1$ is minimally enabled,

$$M_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in I_s(t_1) \\ 0 & \text{otherwise} \end{cases}$$

and $M' = \delta(M, t_1)$,

$$M'_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in \{I_s(t_1), O_s(t_1)\} \\ 0 & \text{otherwise} \end{cases}$$

Since there exists another sequence, $\sigma_j$, the following scenarios can happen.

Scenario 1: $\sigma_j$ is a single transition $t_2$

As $\sigma_j$ is enabled by $M$, therefore $I_s(t_2) \subseteq I_s(t_1)$.

Let $M'' = \delta(M, t_2)$, and since $\exists k, s.t. M_{sk} = 0, M'_{sk} > 0$, therefore $O_s(t_1) \cap O_s(t_2) \neq \emptyset$ and $\exists \sigma_{sk}$ s.t. $p_{sk} \in O_s(t_1)$ and $p_{sk} \in O_s(t_2)$. This indicates a pair of incorrect sequences $\sigma_1 = (t_1)$ and $\sigma_2 = (t_2)$, possibly having problems of redundancy or subsumption.

Scenario 2: $\sigma_j$ is composed of transitions $(t_{m+1}, t_{m+2}, ..., t_{m+n})$

Since $\sigma_j$ is enabled by $M$, therefore $I_s(t_{m+1}) \subseteq I_s(t_1)$. Let $M' = \delta(M, t_{m+1})$,

$$M'_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in \{I_s(t_{m+1}), I_s(t_1)\} \\ 0 & \text{otherwise} \end{cases}$$

For $t_{m+2}$ to be enabled, $I_s(t_{m+2}) \subseteq \{I_s(t_{m+1}), I_s(t_1)\}$. Let $M'' = \delta(M', t_{m+2})$,

$$M''_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in \{I_s(t_{m+1}), I_s(t_1)\} \\ 0 & \text{otherwise} \end{cases}$$

Similarly for any $t_{m+n}$ in $\sigma_j$, where $l = 1, 2, ..., n$,

$$M'_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in \{I_s(t_{m+1}), I_s(t_1)\} \\ 0 & \text{otherwise} \end{cases}$$

Since $\exists k, s.t. t_1 \cap \sigma_j = \emptyset$, $M' = \delta(M, t_1), M'' = \delta(M, t_{m+1})$ with $M_{sk} = 0, M'_{sk} > 0, M''_{sk} > 0$, therefore $\exists \sigma_{sk}$ s.t. $I_s(t_{m+1}) \subseteq I_s(t_1)$, $O_s(t_{m+1}) \subseteq O_s(t_1)$, $O_s(t_{m+1})$ \cap $I_s(t_1) \neq \emptyset$, and $p_{sk} \in O_s(t_{m+1}) \cap I_s(t_1)$, This indicates an incorrect pair of sequence $\sigma_i = (t_1)$ and chained sequence $\sigma_j = (t_{m+1}, t_{m+2}, ..., t_{m+n})$, possibly having problems of redundancy or subsumption.

Case-2: $\sigma_j$ is composed of transitions $(t_1, t_2, ..., t_n)$

Since $\sigma_j$ is enabled by $M$, let $M' = \delta(M, t_1)$,

$$M'_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in \{I_s(t_1), O_s(t_1)\} \\ 0 & \text{otherwise} \end{cases}$$

For $t_2$ to be enabled, $I_s(t_2) \subseteq \{I_s(t_1), O_s(t_1)\}$. Let $M'' = \delta(M', t_2)$,

$$M''_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in \{I_s(t_1), O_s(t_1), O_s(t_2)\} \\ 0 & \text{otherwise} \end{cases}$$

Similarly for any $t_i$ in $\sigma_j$, where $l = 1, 2, ..., m$,

$$M'_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in \{I_s(t_1), O_s(t_1), .., O_s(t_2)\} \\ 0 & \text{otherwise} \end{cases}$$

Since there exists another sequence, $\sigma_j$, the same scenarios can happen as follow.

Scenario 1: $\sigma_j$ is a single transition $t_{m+1}$

As $\sigma_j$ is enabled by $M$, therefore $I_s(t_{m+1}) \subseteq I_s(t_1)$. Let $M'' = \delta(M, t_{m+1})$, and since $\exists k, s.t. M_{sk} = 0, M'_{sk} > 0$, therefore $O_s(t_1) \cap O_s(t_{m+1}) \neq \emptyset$, and $\exists p_{sk}$ s.t. $p_{sk} \in O_s(t_1)$ and $p_{sk} \in O_s(t_{m+1})$. This indicates an incorrect
pair of chained sequences $\sigma_i = (t_1, t_2, \ldots, t_m)$ and sequence $\sigma_j = (t_{m+1})$, possibly having problems of redundancy or subsumption.

Scenario 2: $\sigma_j$ is composed of transitions $(t_{m+1}, t_{m+2}, \ldots, t_{m+n})$

Since $\sigma_j$ is enabled by $M$, therefore $I_s(t_{m+1}) \subseteq I_s(t_1)$. Let $M^1 = \delta(M, t_{m+1})$,

$$M^1_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in \{I_s(t_1), O_s(t_{m+1})\} \\ 0 & \text{otherwise} \end{cases}$$

For $t_{m+2}$ to be enabled, $I_s(t_{m+2}) \subseteq \{I_s(t_1), O_s(t_{m+1})\}$. Let $M^2 = \delta(M^1, t_{m+1})$,

$$M^2_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in \{I_s(t_1), O_s(t_{m+1}), O_s(t_{m+2})\} \\ 0 & \text{otherwise} \end{cases}$$

Similarly for any $t_{m+u}$ in $\sigma_j$, where $u = 1, 2, \ldots, n$,

$$M^u_{sk} = \begin{cases} 1 & \text{if } p_{sk} \in \{I_s(t_1), O_s(t_{m+1}), O_s(t_{m+2}), \ldots, O_s(t_{m+u})\} \\ 0 & \text{otherwise} \end{cases}$$

Since $\exists k, s.t. (\sigma_i) \cap \sigma_j = \emptyset$, $M' = \delta(M, \sigma_i)$, $M'' = \delta(M, \sigma_j)$ with $M_{sk} = 0$, $M'_{sk} > 0$, $M''_{sk} > 0$, therefore $\exists p_{sk}, \exists t, u$, s.t. $I_s(t_{m+1}) \subseteq I_s(t_1)$, $O_s(t_1) \subseteq O_s(\sigma_i)$, $O_s(t_{m+u}) \subseteq O_s(\sigma_j)$, $O_s(t_1) \cap I_s(t_{m+u}) \neq \emptyset$, and $p_{sk} \in O_s(t_1) \cap I_s(t_{m+u})$. This indicates a pair of incorrect chained sequences $\sigma_i = (t_1, t_2, \ldots, t_m)$ and $\sigma_j = (t_{m+1}, t_{m+2}, \ldots, t_{m+n})$, possibly having problems of redundancy or subsumption.

This completes the proof of the proposition.