Scenario-Based Consistency Verification for Concurrent Models

Draft

Abstract

Scenarios are widely used as a requirements technique since they describe concrete interactions and are therefore easy for customers and domain experts to use. In this paper, we use UML interaction models as scenario-based specifications, which consist of UML2.0 interaction overview diagrams (IODs) and sequence diagrams. As one powerful formalism, Petri nets can model concurrency constraints in natural way, and are often used in modelling system specifications and designs. Usually, UML interaction models are used to describe the requirements provided directly by the customers, while Petri nets are used to model the workflow synthesized by the domain and technical experts. So it is necessary and important to keep the consistency between these two kinds of models for software quality assurance. In this paper, we use Petri nets to model concurrent systems, and consider the problem of checking Petri nets for the scenario-based specifications expressed by UML interaction models. We develop the algorithms to solve the following two verification problems: the *existential consistency checking problem*, which means that a scenario described by a given IOD must happen during the run of a Petri net, or any forbidden scenario described by a given IOD never happens during the run of a Petri net; and the *mandatory consistency checking problem*, which means that if reference scenarios described by the given sequence diagrams occurs during the run of a Petri net, it must immediately adhere to a scenario described by the other given IOD.
Key words: Software verification, formal methods, scenario-based specifications, interaction overview diagrams, Petri nets.

1 Introduction

Scenarios are widely used as a requirements technique since they describe concrete interactions and are therefore easy for customers and domain experts to use. Scenario-based specifications such as message sequence charts [1] and UML interaction overview diagrams [2] offer an intuitive and visual way of describing system requirements. They are playing an increasingly important role in specification and design of systems. Such specifications

focus on message exchanges among communicating entities in real-time and distributed systems.

Since Unified Modelling Language (UML) [3] became a standard in OMG in 1997, UML interaction models like sequence diagrams and activity diagrams have become a main class of artifacts in software development processes. As a variant of activity diagrams the UML2.0 interaction overview diagrams in which the nodes are sequence diagrams focus on overview of flow of control as well as the message interchange between a number of entities. Since it is common to describe multiple scenarios we can expect interaction overview diagrams to be popular for its strong and flexible expressibility. Petri Nets [4] is a formal and graphical appealing language which is appropriate for modelling systems with concurrency and resource sharing. There are plenty of applications of Petri Nets in modelling system specifications and designs. So it follows that we often need to use UML interaction models and Perti nets together in specification and design of software projects. Usually, UML interaction models and Petri nets are used in the different software development steps. Even used in the same step, e.g. requirements analysis, UML interaction models are used usually to describe the scenario-based requirements provided directly by the customers, while Petri nets are used to model the workflow synthesized by the domain and technical experts. So it is necessary and important to keep the consistency between these two kinds of models for software quality assurance.

In this paper, we consider the problem of checking the system designs modelled by Petri nets for the scenario-based specifications expressed by UML interaction models. We develop the algorithms to solve the following two verification problems: the *existential consistency checking problem* and the *mandatory consistency checking problem.*

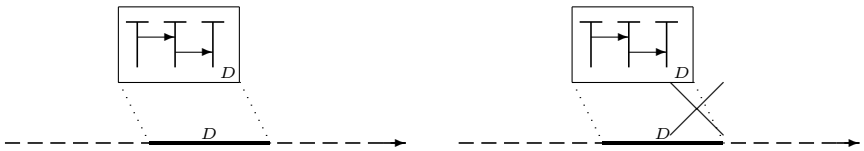


Figure 1: Existential Consistency Checking

The *existential consistency checking problem* is to check if a scenario described by a given IOD must happen during the run of a Petri net, or if any forbidden scenario described by a given IOD never happens during the run of a Petri net, which is depicted in Figure 1. The existential consistency is often corresponding to a safety property in specification and verification.

The *mandatory consistency checking problem* is to check if a mandatory consistency specification is satisfied during the run of a Petri net, which means that if a reference scenario described by the given sequence diagrams

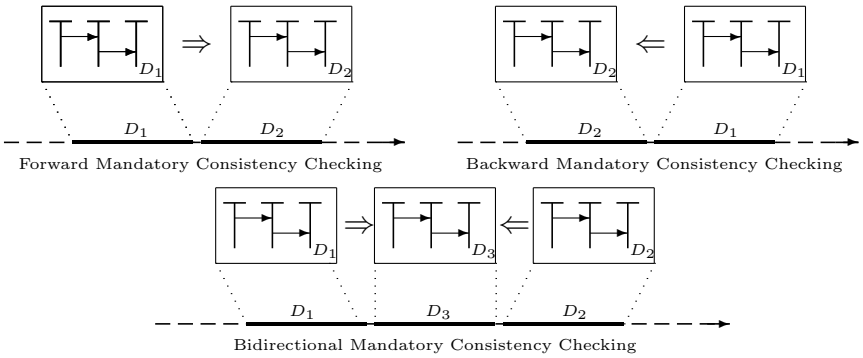


Figure 2: Mandatory Consistency Checking

(SDs) occurs during the run of a Petri net, it must immediately adhere to a scenario described by the other given IOD. We consider the following three kinds of the mandatory consistency which are depicted in Figure 2:

- *forward mandatory consistency:* if a reference scenario described by a given SD D occurs during the run of a Petri net, then a scenario described by an IOD G must follow immediately;
- *backward mandatory consistency:* if a reference scenario described by a given SD D occurs during the run a Petri net, then it must follow immediately from a scenario described by an IOD G; and
- *bidirectional mandatory consistency:* if a reference scenario described by a given SD D_1 occurs during the run of a Petri net and a reference scenario described by another given SD D_2 follows, then in between these two scenarios, a scenario described by an IOD G must occur exactly.

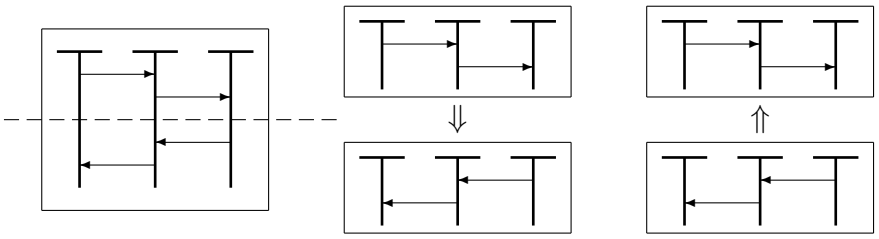


Figure 3: Decomposing verified scenarios into mandatory consistency specifications

As the existential consistency, the mandatory consistency is popular in specification and verification. The mandatory consistency specifications not only come directly from the requirements given by customers or domain experts, but also can be derived from testing and verification processes. For

example, suppose that we attempt to find some errors related to a given scenario, which means that the scenario is not implemented correctly in a system. Since the scenario will not occur during the system run if it is not implemented correctly, it is difficult for us to decide where and when the scenario should occur in order to find the related errors further. In this case, we can decompose the scenario into two parts which constitute a mandatory consistency specification, and use one part as a reference scenario, and test or verify if there is an error in the other part, which is depicted in Figure 3.

The paper is organized as follows. In next section, we introduce scenario-based specifications formally. In Section 3, we review the definition and some basic properties of Petri nets. The solutions are given in Section 4 and 5 respectively to the existential and mandatory consistency checking of Petri nets for the scenario-based specifications. Section 6 gives out a case study and simply describes the verification tool prototype in which the algorithms presented in this paper are implemented. The related works and some conclusions are given in the last section.

2 Scenario-based Specifications

2.1 Sequence Diagrams

A UML sequence diagram describes an interaction, which is a set of messages exchanged among objects within a collaboration to effect a desired operation or result, and its focus is on the temporal order of the message flow [2,3]. In this paper, we just use a simplified version of sequence diagrams, which describe exactly one scenario without any alternative and loop. A sequence diagram considered in this paper has two dimensions: the vertical dimension represents time, and the horizontal dimension represents different objects. Each object is assigned a column, and the messages are shown as horizontal, labelled arrows.

For example, a sequence diagram is depicted in Figure 4, which describes a scenario about the well-known example of the railroad crossing system in [27,28]. This system operates a barrier at a railroad crossing, in which there are a railroad crossing monitor and a barrier controller for controlling the barrier. When the monitor detects that a train is arriving, it sends a message to the controller to lower the barrier. After the train leaves the crossing, the monitor sends a message to controller to raise the barrier.

In a sequence diagram, by events we mean the message sending and the message receiving. The semantics of a sequence diagram essentially consists of the sequences (traces) of the message sending (receiving) events. The order of events (i.e. message sending or receiving) in a trace is deduced from the visual partial order determined by the flow of control within each object in the sequence diagram along with a causal dependency between the events of sending and receiving a message [1-3,13]. In accordance with [13], without

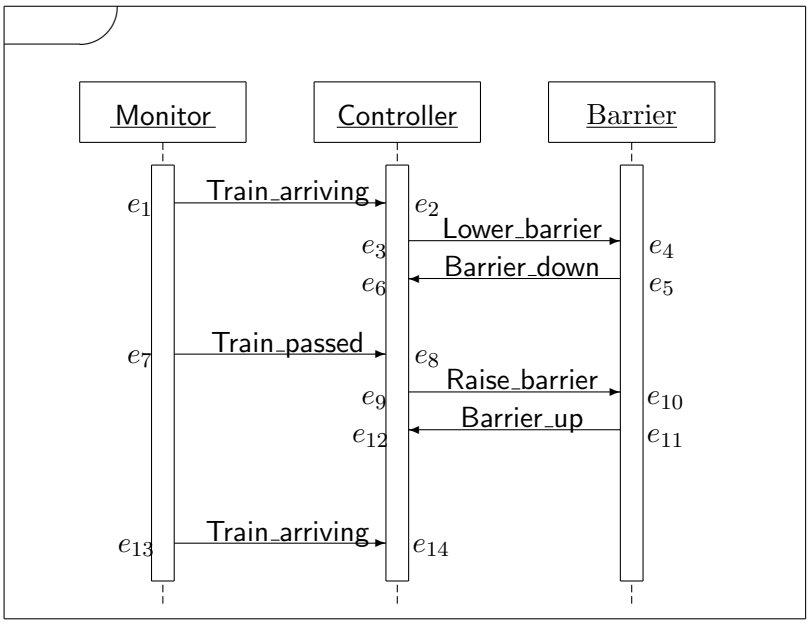


Figure 4: An UML sequence diagram describing the railroad crossing system

losing generality, we assume that each sequence diagram corresponds to a visual order for a pair of events e and e' such that e precedes e' in the following cases:

- **Causality:** A sending event e and its corresponding receiving event e'. For example, in the sequence diagram depicted in Figure 3, e_5 precedes e_6.
- **Controllability:** The event e appears above the event e' on the same object column, and e' is a sending event. This order reflects the fact that a send event can wait for other events to occur. On the other hand, we sometimes have less control on the order in which receive events occur. For example, in the sequence diagram depicted in Figure 3, e_6 precedes e_9.
- **Fifo order:** The receiving event e appears above the receiving event e' on the same object column, and the corresponding sending events e_1 and e'_1 appear on a mutual object column where e_1 is above e'_1. For example, in the sequence diagram depicted in Figure 3, e_4 precedes e_{10}.

For giving the formal definition of scenario-based specifications, we formalize sequence diagrams as follows.

Definition 1 A sequence diagram is a tuple $D = (O, E, M, L, V)$ where

- O is a finite set of objects. For each object $o \in O$, we use $\zeta(o)$ to denote the class which o belongs to.
- E is a finite set of events corresponding to sending or receiving a message.
- M is a finite set of messages. Each message in M is of the form (e, g, e') where $e, e' \in E$ corresponds to sending and receiving the message respectively, and g is the message name which is a character string. For any message $(e, g, e') \in M$, we use $g!$ and $g?$ to represent the sending and the receiving for the message respectively if we just concern the message name, and let $\phi(e) = g!$ and $\phi(e') = g?$.
- $L : E \to O$ is a labelling function which maps each event $e \in E$ to an object $L(e) \in O$ which is the sender (receiver) while e corresponds to sending (receiving) a message.
- V is a finite set whose elements are a pair (e, e') where $e, e' \in E$ and e precedes e', which represents a visual order. □

We use *event sequences* to represent the *traces* of sequence diagrams, which describes the temporal order of the message flow. An event sequence is of the form $e_0 \hat{} e_1 \hat{} \ldots \hat{} e_m$, which represents that e_{i+1} takes place after e_i for any i $(0 \leq i \leq m-1)$.

Definition 2 For any sequence diagram $D = (O, E, M, L, V)$, an event sequence $e_0 \hat{} e_1 \hat{} \ldots \hat{} e_m$ is a *trace* of D if and only if the following condition holds:

- all events in E occur in the sequence, and each event occurs only once, i.e. $\{e_0, e_1, \ldots, e_m\} = E$ and $e_i \neq e_j$ for any i, j $(0 \leq i < j \leq m)$; and
- $e_0, e_1, \ldots, e_m$ satisfy the visual order defined by V, i.e. for any e_i $(0 \leq i \leq m)$ and e_j $(0 \leq j \leq m)$, if $(e_i, e_j) \in V$, then $0 \leq i < j \leq m$. □

We can transform the traces of a sequence diagram into the *message trails* of the diagram by replacing each event with the corresponding sending and receiving messages.

Definition 3 Let $D = (O, E, M, L, V)$ be a sequence diagram. For any trace of D of the form $e_0 \hat{} e_1 \hat{} \ldots \hat{} e_m$, replacing each e_i with $\phi(e_i)$ $(0 \leq i \leq m)$, we get a sequence $\phi(e_0) \hat{} \phi(e_1) \hat{} \ldots \hat{} \phi(e_m)$ of the sending or receiving messages in M, which is a *message trail* of D. □

Notice that for a sequence diagram D, all events in a trace of D are distinct, but there may be the same events in a message trail of D which are corresponding to the message sending or receiving. For example, the events e_1 and e_{13} are distinct in the sequence diagram depicted in Figure 1, but $\phi(e_1) = \phi(e_{13}) =$Train_arriving!.

2.2 Interaction Overview Diagrams

A sequence diagram considered in this paper just describes one scenario. For describing multiple scenarios, we need to use a simplified version of UML2.0 interaction overview diagrams [2], which focuses on the overview of the flow of control where the nodes are sequence diagrams. An interaction overview diagram defines a composition of a set of sequence diagrams, which describes potentially iterating and branching system behavior.

For example, Figure 5 depicts an interaction overview diagram which specifies the FIPA Iterated Contract Net Iteration Protocol [28]. This protocol implements the interaction between the agents Initiator and Participant such that the Initiator seeks to get better bid from the Participant by modifying the call and requesting a new (equivalently, revised) bid. The Initiator issues initial call for proposal with the cfp message (in sequence diagram cfp). If the Participant is willing and able to do the task under the proposed conditions (in sequence diagram propose), then it replies a propose message, otherwise it can refuse (in sequence diagram refuse). When receiving a propose message, the Initiator may decide this is the final iteration and accept the bid (in sequence diagram accept), or reject it (in sequence diagram reject). After the Initiator accepts the bid, once the Participant completes the task, it sends a inform message to the Initiator (in sequence diagram inform). However, if the Participant fails to complete the task, a failure message is sent (in sequence diagram failure). Alternatively the Initiator may decide to iterate the process by issuing a revised cfp to the Participant (in sequence diagrams propose and cfp). The process terminates when the Initiator refuses a proposal and does not issue an new message cfp, the Initiator accepts a bid, or the Participant refuses to bid.

Definition 4 An interaction overview diagram (IOD) is a tuple

$$G = (U, N, E, succ, ref)$$

where

- U is a finite set of sequence diagrams satisfying that for any $D_1 = (O_1, E_1, M_1, L_1, V_1)$ and $D_2 = (O_2, E_2, M_2, L_2, V_2)$ in U, if $D_1 \neq D_2$, then $E_1 \cap E_2 = \emptyset$;
- $N = \{\top\} \cup I \cup \{\bot\}$ is a finite set of nodes partitioned into the three sets: the singleton-set of *start* node, the set of *intermediate* nodes, and the singleton-set of *end* node, respectively;
- E is a finite set of events satisfying that for all the sequence diagrams $D_i = (O_i, E_i, M_i, L_i, V_i)$ in U, $E = \cup E_i$;
- $succ \subset N \times N$ is the relation which reflects the connectivity of the nodes in N such that any node in N is reachable from the start node; and

- $ref : I \mapsto U$ is a function that maps each intermediate node to a sequence diagram in U. □

For an IOD $G = (U, N, E, succ, ref)$, a *path segment* is a sequence of intermediate nodes $v_0 \hat{} v_1 \hat{} \ldots \hat{} v_n$ satisfying that $(v_{i-1}, v_i) \in succ$ for any i $(0 < i \leq n)$. A *path* is a path segment $v_0 \hat{} v_1 \hat{} \ldots \hat{} v_n$ such that $(\top, v_0) \in succ$ and $(v_n, \bot) \in succ$. A path segment is called *simple* if all its nodes are distinct. Let $v_0 \hat{} v_1 \hat{} \ldots \hat{} v_n$ be a simple path segment in G such that $(\top, v_0) \in succ$. If there is v_i $(0 \leq i \leq n)$ such that $(v_n, v_i) \in succ$, then the sequence $v_i \hat{} v_{i+1} \hat{} \ldots \hat{} v_n \hat{} v_i$ is a *loop*, and v_i is the *loop-start node* of the loop.

In UML2.0, IODs are defined as specialization of activity diagrams in a way that promotes overview of the control flow [2]. It follows that the concatenation of two sequence diagrams in an IOD should be interpreted as the *synchronous mode* which means that when moving one node to the other, all events in the previous sequence diagram finish before any event in the following sequence diagram occurs, which is the same as the synchronous interpretation in MSC specifications [25]. Therefore, we define *traces* of an IOD G as the event sequences which are the concatenation of the traces of the sequence diagrams which make up G. We use $\hat{}$ to denote the concatenation of sequences.

Definition 5 For an IOD $G = (U, N, E, succ, ref)$, an event sequence

$$\sigma = e_0 \hat{} e_1 \hat{} \ldots \hat{} e_n$$

represents a *trace* of ρ, where ρ is a path segment of G, if and only if the following condition holds:

- there is a path segment $\rho = v_0 \hat{} v_1 \hat{} \ldots \hat{} v_m$ in G such that $\sigma = \sigma_0 \hat{}$ $\sigma_1 \hat{} \ldots \hat{} \sigma_m$, where σ_i is a trace of $ref(v_i)$ for each i $(0 \leq i \leq m)$.

If ρ is a path of G then σ is the *trace* of G. □

Similar to the sequence diagram, we define the *message trails* of an IOD G as the concatenations of the trails of the sequence diagrams which make up G.

Definition 6 Let $G = (U, N, E, succ, ref)$ be an IOD. For any path segment ρ of G of the form $v_0 \hat{} v_1 \hat{} \ldots \hat{} v_m$ and the trace σ of ρ of the form $\sigma_0 \hat{} \sigma_1 \hat{} \ldots \hat{} \sigma_m$, each σ_i is the trace of $ref(v_i)(0 \leq i \leq m)$. By replacing every σ_i with its message trail, we get the *message trail* of ρ. If ρ is the path of G, then σ is the *message trail* of G.

3 Petri Nets

The Petri nets we consider in this paper are classical 1-safe systems.

Definition 7 A Petri net is a four-tuple, $N = (P, T, F, \mu_0)$, where

- $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of *places*;
- $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of *transitions* ($P \cap T = \emptyset$);
- $F \subset (P \times T) \cup (T \times P)$ is the *flow relation*;
- $\mu_0 \subset P$ is the *initial marking* of the net.

A *marking* μ of N is any subset of P. For any transition t, ${}^\bullet t = \{p \in P | (p,t) \in F\}$ and $t^\bullet = \{p \in P | (t,p) \in F\}$ denote the *preset* and *postset* of t, respectively. A transition t is *enabled* in a marking μ if ${}^\bullet t \subseteq \mu$; otherwise, it is *disabled*. Let $enabled(\mu)$ be the set of transitions enabled in μ. □

For the firing of a transition to be possible, two conditions must be satisfied.

Definition 8 A transition t may fire from marking μ if and only if the following two conditions hold: (1) $t \in enabled(\mu)$, and (2) $(\mu - {}^\bullet t) \cap t^\bullet = \emptyset$. □

The first condition is the normal firing condition for Petri nets. The second condition requires *contact-freenes*. The new state is then calculated as follows.

Definition 9 When transition t fires from marking μ, the new marking μ' is given as follows: $\mu' = (\mu - {}^\bullet t) \cup t^\bullet$. □

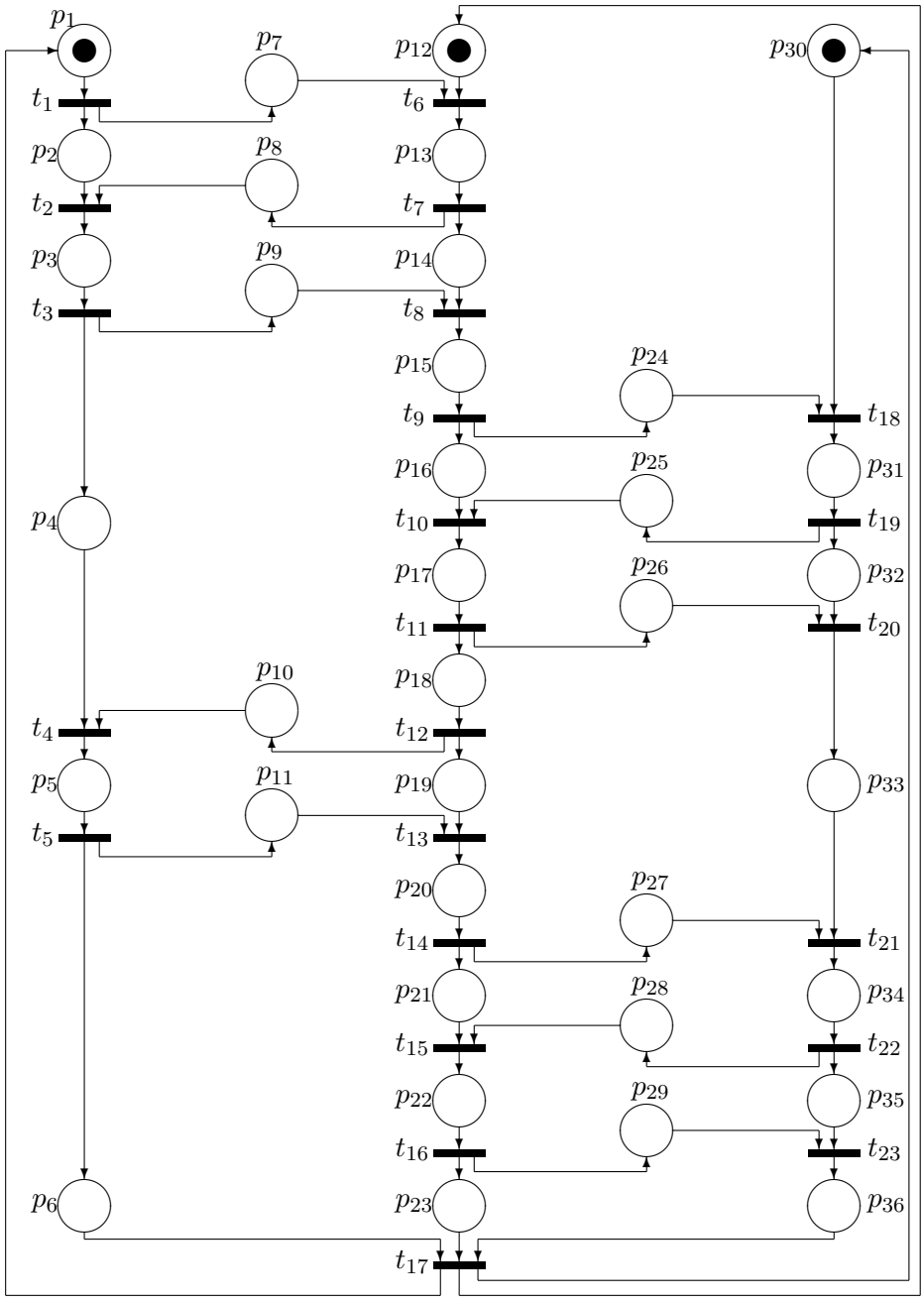
Note that since we assume contact-freeness, a self-loop will not be enabled. The behaviour of a Petri net is described in term of *runs*.

Definition 10 A *run* of a Petri net is a finite or infinite sequence of markings and transitions

$$\sigma = \mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \ldots$$

such that μ_0 is the initial marking of the net, $t_i \in enabled(\mu_i)$ for any i ($i \geq 0$), and that $\mu_i = (\mu_{i-1} -^\bullet t_{i-1}) \cup t_{i-1}^\bullet$ for any i ($i \geq 1$). □

For a Petri net $N = (P, T, F, \mu_0)$, a run is called *simple* if all its markings are distinct. Let $\sigma = \mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \ldots$ be a simple run in N. If there is μ_i ($0 \leq i \leq n$) such that $\mu_i = (\mu_n -^\bullet t_n) \cup t_n^\bullet$, then the sequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_i$ is a *loop*.



$\varphi(t_1)$ =Train_arriving! $\varphi(t_2)$ =Acknowledgement? $\varphi(t_3)$ =Approaching! $\varphi(t_4)$ =Crossing_secured?
$\varphi(t_5)$ =Train_passed! $\varphi(t_6)$ =Train_arriving? $\varphi(t_7)$ =Acknowledgement! $\varphi(t_8)$ =Approaching?
$\varphi(t_9)$ =Low_barrier! $\varphi(t_{10})$ =Barrier_down? $\varphi(t_{11})$ =Power_off! $\varphi(t_{12})$ =Crossing_secured!
$\varphi(t_{13})$ =Train_passed? $\varphi(t_{14})$ =Raise_barrier! $\varphi(t_{15})$ =Barrier_up? $\varphi(t_{16})$ =Power_off!
$\varphi(t_{18})$ =Low_barrier? $\varphi(t_{19})$ =Barrier_down! $\varphi(t_{20})$ =Power_off! $\varphi(t_{21})$ =Raise_barrier?
$\varphi(t_{22})$ =Barrier_up! $\varphi(t_{23})$ =Power_off!

Figure 5: A Petri net model for the railway crossing system

As a tool used for modelling systems, the transitions of Petri nets represent the potential events in the systems. Since in this paper we consider the problem of checking Petri nets for the scenario-based specifications expressed by IODs, for any Petri net we consider in this paper, each transition t is labelled with an event which may be corresponding to a message sending or receiving in a sequence diagram, which is denoted by $\varphi(t)$. That is, for a IOD $G = (U, N, E, succ, ref)$ and a sequence diagram $D = (O, E, M, L, V)$ in U, for a transition t of a Petri net, there may be a message $(e, g, e') \in M$ such that $\varphi(t) = g! = \phi(e)$ or $\varphi(t) = g? = \phi(e')$.

For example, for the railroad crossing system described in the above section, its design can be described by a Petri net depicted in Figure 6. In the system, when the monitor detects that a train is arriving, it sends the message Train_arriving to the controller. The controller sends a message back for acknowledgement, and the monitor gives a reply. Once the controller receives the confirmed message Approaching, it sends the message Low_barrier to the barrier. The barrier is put down after receiving the message Low_barrier, and the message Barrier_down is sent to the controller. Then the controller sends the message Power_off to the Barrier, and the message Barrier_secured to the monitor so that the train can pass the crossing. Once the train passes the crossing, the monitor sends a message to the controller, and after receiving the message the controller sends the message Raise_barrier to the barrier. The barrier becomes up after receiving the message from the controller, and the message Barrier_up is sent to the controller. Once receiving the massage Barrier_up, the controller sends a message to the barrier for turning off the power. The barrier will be holding up until another train is arriving.

4 Existential Consistency Checking

In this section, we consider the existential consistency checking of Petri nets for the scenario-based specifications. The scenario-based specifications considered in this paper are the UML interaction models consisting of IODs and sequence diagrams. IODs are the composition of sequence diagrams, and sequence diagrams are the specialization of IODs. The existential consistency checking problem is to check if a scenario described by a given IOD must happen during the run of a Petri net, or if any forbidden scenario described by a given IOD never happens during the run of a Petri net. For example, there are two IODs depicted in Figure 7 which can be used as the scenario-based specifications for the railway crossing system. The left one describes a normal scenario for the preparation for the train crossing, which should occur during the run of the Petri net depicted in Figure 6. The right one is an exceptional scenario in which the message Barrier_secured is sent to the moniter before the barrier is put down, which is forbidden to occur during the run of the Petri net depicted in Figure 6.

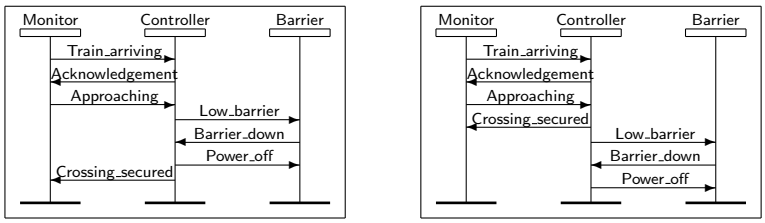


Figure 6: Existential consistency specifications for the railway crossing system

Now we define formally the existential consistency checking problem. Let $G = (U, N, E, succ, ref)$ be a IOD, N be a Petri net, and σ be an run of N of the form $\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$. For any subsequence σ_1 in σ of the form $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1}$ $(0 \leq i < j \leq n)$, since each transition t_k is labelled with an event $\varphi(t_k)$ $(i \leq k \leq j)$, we get a sequence τ of events: $\varphi(t_i)\hat{}\varphi(t_{i+1})\hat{}\ldots\hat{}\varphi(t_j)$. By removing any $\varphi(t_k)$ $(i \leq k \leq j)$ from τ which is not corresponding to the sending or receiving for a message in the sequence diagrams of set U, we get a sequence $\tau_1 = e_0\hat{}e_1\hat{}\ldots\hat{}e_m$ $(m \leq j - i)$, which is a *filtered trail* of σ_1. If τ_1 is a message trail of a path segment ρ in G, then σ_1 is an *image* of ρ. If ρ is a path of G, then σ_1 is an *image* of G, and we say that a scenario described by G *occurs* in the run σ. If τ_1 is a message trail of G, $\varphi(t_i) = e_0$, and $\varphi(t_j) = e_m$, then we say that σ_1 is an *exact image* of G. For a Petri net N, for a IOD G, the existential consistency checking problem is to check if there is a run of N where a scenario described by G occurs.

We know that for a Petri net N, there could be infinite runs, and the number of the runs could be infinite. In the following we show how to solve the problem based on the investigation of a finite set of finite runs.

Definition 11 Let $G = (U, N, E, succ, ref)$ be a IOD, N be a Petri net, and σ be an run of N of the form $\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$. For any subsequence σ_1 in σ of the form $\mu_p \xrightarrow{t_p} \mu_{p+1} \xrightarrow{t_{p+1}} \ldots \xrightarrow{t_{q-1}} \mu_j \xrightarrow{t_q} \mu_{q+1}$ $(0 \leq p < q \leq n)$ is a *loop-bounded image* of G if and only if the following conditions hold:

- σ_1 is an exact image of G, i.e. there is a path ρ of G such that σ_1 is an image of ρ
- for any subsequence σ_2 of σ_1 of the form $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1} \xrightarrow{t_{j+1}} \ldots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \mu_{k+1}$ $(m \leq i < j < k \leq n)$, if σ_2 is an image of
  - σ_2 is not a image of any path segment in D; or

– σ_2 is an image of a path segment ρ=$v_p \hat{} v_{p+1} \hat{} \ldots \hat{} v_q$, that is, there is an *essential label trace* of σ_2 which can be written into $\rho_p \hat{} \rho_{p+1} \hat{} \ldots \hat{} \rho_q$ where each ρ_l $(p \leq l \leq q)$ is the message trail of $ref(v_l)$, and $\mu_{j+1} \xrightarrow{t_{j+1}} \ldots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \mu_{k+1}$ is the image of $ref(v_q)$. If $v_p = v_q$ then $\mu_i \neq \mu_{j+1}$.

Lemma 1 Let $D = (U, N, E, succ, ref)$ be a IOD, N be a Petri net, σ be an run of N. If σ is an exact image of a path segment of D and at the same time σ satisfies the loop bounded condition, then σ is finite, and its length has a upper bound. □

The proof of this lemma is presented in the appendix. Based on the lemma, we are able to specify the finite set $\Delta(N, D)$. For any Petri net N, for any IOD $D = (U, N, E, succ, ref)$, let $\Delta(N, D)$ be the set of the runs of N which are of the form

$$\sigma = \mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1} ,$$

where

- all μ_i $(0 \leq i \leq m)$ are distinct;
- $\mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$ is an exact image of D; and also satisfies loop bounded condition.

It is clear that the number of the runs in $\Delta(N, D)$ is finite, and that each run in $\Delta(N, D)$ is finite.

Theorem 1 Let N be a Petri net, and D be a IOD. Then there is an run of N where a scenario described by D occurs if and only if $\Delta(N, D) \neq \emptyset$. □

The proof of this theorem is presented in the appendix. For a Petri net N, for a IOD D, a run σ of N is a *prefix* for $\Delta(N, D)$ if it may be extended into a run which is in $\Delta(N, D)$, i.e. there could be a sequence σ_1 of markings and transitions such that $\sigma \hat{} \sigma_1$ is in $\Delta(N, D)$. Based on the above theorem, we can develop an algorithm to check the existential consistency of a Petri net $N = (P, T, F, \mu_0)$ for a IOD D (cf. Figure 8). The algorithm traverses the state space of N in a depth first manner starting from the initial node μ_0. The path in the state space that we have so far traversed is stored in the list variable *currentpath*. For each new marking that we discover, we first check whether it is such that the run corresponding to *currentpath* is in $\Delta(N, D)$. If yes, then it means that a scenario described by D must happen during the run of N, and we are done. Then we check if the new marking that we discover is such that the run corresponding to *currentpath* is a prefix for $\Delta(N, D)$. If yes, then we add the new marking to the current path and start

```
currentpath := ⟨μ0⟩;
repeat
   node := the last node of currentpath;
   if node has no new successive node
   then delete the last node of currentpath
   else begin
          node := a new successive node of node;
          if node is such that the run corresponding to currentpath
             is in Δ(N, D)
          then return true;
          if node is such that the run corresponding to currentpath
             is a prefix for Δ(N, D)
          then append node to currentpath;
        end
until currentpath = ⟨⟩;
return false.
```

Figure 7: Algorithm for existential consistency checking

the search from it, otherwise we backtrack. If no run is discovered during the search which is in $\Delta(N, D)$, then it means that any forbidden scenario described by D never happens during the run of N. The complexity of the algorithm is proportional to the number of the prefixes for $\Delta(N, D)$ and to the size of the longest prefix for $\Delta(N, D)$.

5 Mandatory Consistency Checking

In this section, we consider the mandatory consistency checking of Petri nets for the scenario-based specifications. The mandatory consistency requires that if a reference scenario described by the given IODs occurs during the run of a Petri net, it must immediately adhere to a scenario described by the given IOD(cf. Figure 2). In the following, we solve the forward mandatory consistency checking problem, the backward mandatory consistency checking problem, and the bidirectional mandatory consistency checking problem respectively.

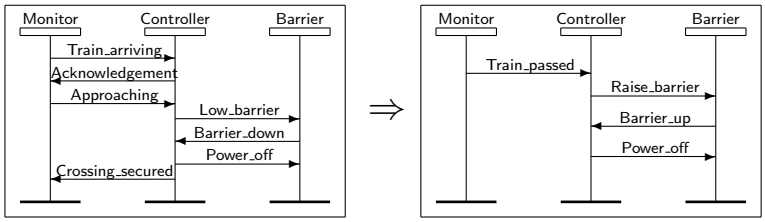


Figure 8: Forward mandatory consistency specification for the railway crossing system

5.1 Forward Mandatory Consistency Checking

For the forward mandatory consistency checking, a scenario-based specification, denoted by $\mathcal{S}_F(D_1, D_2)$, consists of tow IODs D_1 and D_2, which requires that if a scenario described by D_1 occurs in the run of a Petri net, then a scenario described by D_2 follows immediately. For example, a forward mandatory consistency specification for the railway crossing system is depicted in Figure 9, which requires that from the scenario for the preparation for the train crossing, the scenario for raising the barrier after the train passes must follows immediately. This specification should be satisfied during the run of the Petri net depicted in Figure 6.

The satisfaction problem of a Petri net N for a scenario-based specification $\mathcal{S}_F(D_1, D_2)$ is defined formally as follows. N satisfies $\mathcal{S}_F(D_1, D_2)$ if any run σ of N of the form

$$\sigma = \mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$$

satisfies the following condition:

- if there is a subsequence σ_1 in σ of the form

  $$\sigma_1 = \mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1} \ (0 \leq i < j \leq n)$$

  which is an exact image of D_1, then there is a subsequence σ_2 in σ of the form

  $$\sigma_2 = \mu_{j+1} \xrightarrow{t_{j+1}} \mu_{j+2} \xrightarrow{t_{j+2}} \ldots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \mu_{k+1} \ (j < k \leq n)$$

  and σ_2 is an image of D_2.

Now we try to solve the verification problem based on the investigation of a finite set of finite runs. To make the solution feasible and efficient, we need to strengthen the loop bounded condition to *loop match condition*. Let $D = (U, N, E, succ, ref)$ be a IOD, N be a Petri net, σ be an run of N, and a subsequence σ_1 of σ of the form $\mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$ is an image of D. σ_1 satisfies the *loop match condition* if and only if the following conditions hold:

- for any subsequence σ_2 in σ_1 of the form $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1}$ $(m \leq i \leq j \leq n)$ and σ_2 being the loop of N, if there exists an event sequence of D of the form $e_p \hat{} e_{p+1} \hat{} \ldots \hat{} e_q$ such that the essential label trace of σ_1 in the form of $\varphi(t_p) \hat{} \varphi(t_{p+1}) \hat{} \ldots \hat{} \varphi(t_q)$ $(i \leq p \leq q \leq j)$ satisfies

  $$\varphi(t_p) \hat{} \varphi(t_{p+1}) \hat{} \ldots \hat{} \varphi(t_q) = \phi(e_p) \hat{} \phi(e_{p+1}) \hat{} \ldots \hat{} \phi(e_q),$$

  then there is an event sequence of D of the form $e_p \hat{} e_{p+1} \hat{} \ldots \hat{} e_q \hat{} e_{q+1}$ such that $e_p = e_{q+1}$.

For any Petri net N, for any scenario-based specification $\mathcal{S}_F(D_1, D_2)$ where

$$D_1 = (U_1, N_1, E_1, succ_1, ref_1) \text{ and } D_2 = (U_2, N_2, E_2, succ_2, ref_2)\,,$$

let $\Delta(N, \mathcal{S}_F(D_1, D_2))$ be the set of the runs of N which are of the form

$$\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \ldots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}\,,$$

where

- all μ_i $(0 \leq i \leq k)$ are distinct;
- $\mu_k \xrightarrow{t_k} \mu_{k+1} \xrightarrow{t_{k+1}} \ldots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \mu_{m+1}$ is an exact image of D_1, and satisfies the loop bounded condition;
- there is not any $t_l (k \leq l \leq m)$ such that $\varphi(t_l) = \phi(e)$ $(e \in E_2)$;
- for any μ_i and μ_j $(m < i < j \leq n)$, if there is not any $t_l (i \leq l \leq j)$ such that $\varphi(t_l) = \phi(e)$ $(e \in E_2)$ then $\mu_i \neq \mu_j$, and
- for any μ_h, μ_i and μ_j $(m < h < i < j \leq n)$, if μ_i=μ_h then $\mu_j \neq \mu_h$.

For any $\sigma \in \Delta(N, \mathcal{S}_F(D_1, D_2))$ of the above form, we call the subsequence $\mu_{m+1} \xrightarrow{t_{m+1}} \mu_{m+2} \xrightarrow{t_{m+2}} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$ by the *last segment* of σ.

Theorem 2 A Petri net N satisfies a scenario-based specification $\mathcal{S}_F(D_1, D_2)$ if and only if for any run of N which is in $\Delta(N, \mathcal{S}_F(D_1, D_2))$, its last segment is an image of D_2 and satisfies the loop match condition. □

The proof of this theorem is presented in the appendix. For a Petri net N, for a scenario-based specification $\mathcal{S}_F(D_1, D_2)$, a run σ of N is a *prefix* for $\Delta(N, \mathcal{S}_F(D_1, D_2))$ if it may be extended into a run which is in $\Delta(N, \mathcal{S}_F(D_1, D_2))$, i.e. there could be a sequence σ_1 of markings and transitions such that $\sigma \hat{} \sigma_1$ is in $\Delta(N, \mathcal{S}_F(D_1, D_2))$. Based on Theorem 2, we can develop an algorithm to check if a Petri net $N = (P, T, F, \mu_0)$ satisfies a scenario-based specification $\mathcal{S}_F(D_1, D_2)$ (cf. Figure 10). The algorithm traverses the state space of N in a depth first manner starting from the initial node μ_0. The path in the state space that we have so far traversed is stored in the list variable *currentpath*. For each new marking that we discover, we first check whether it is such that the run corresponding to *currentpath* is in $\Delta(N, \mathcal{S}_F(D_1, D_2))$. If so, we check if the run corresponding to *currentpath* is such that its last segment is an image of D_2 and satisfies the loop match condition. If no then it mean that N does not satisfy $\mathcal{S}_F(D_1, D_2)$, and we are done. Then we check if the new marking that we discover is such that the run corresponding to *currentpath* is a prefix for $\Delta(N, \mathcal{S}_F(D_1, D_2))$. If yes, then we add the new marking to the current path and start the search from it, otherwise we backtrack. The complexity of the algorithm is proportional to the number of the prefixes for $\Delta(N, \mathcal{S}_F(D_1, D_2))$ and to the size of the longest prefix for $\Delta(N, \mathcal{S}_F(D_1, D_2))$.

```
currentpath := ⟨μ0⟩;
repeat
   node := the last node of currentpath;
   if node has no new successive node
   then delete the last node of currentpath
   else begin
          node := a new successive node of node;
          if node is such that the run corresponding to currentpath
             is in Δ(N, S_F(D1, D2))
          then
            begin
              check if the run corresponding to currentpath is
              such that its last segment is an image of D2 and
              satisfies the loop match condition;
              if no then return false;
            end;
          if node is such that the run corresponding to currentpath
             is a prefix for Δ(N, S_F(D1, D2))
          then append node to currentpath;
        end
until currentpath = ⟨⟩;
return true.
```

Figure 9: Algorithm for forward mandatory consistency checking

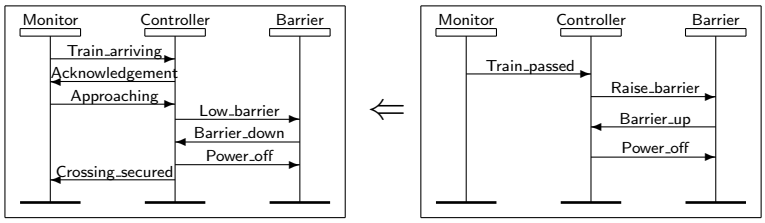


Figure 10: Backward mandatory consistency specification for the railway crossing system

5.2 Backward Mandatory Consistency Checking

For the backward mandatory consistency checking, a scenario-based specification, denoted by $\mathcal{S}_B(D_1, D_2)$, consists of two given IODs D_1 and D_2, which requires that if a scenario described by D_1 occurs in the run of a Petri net, then it must follows immediately from a scenario described by D_2. For example, a backward mandatory consistency specification for the railway crossing system is depicted in Figure 11, which requires that the scenario for raising the barrier after the train passes must follows immediately from the scenario for the preparation for the train crossing. This specification should be satisfied during the run of the Petri net depicted in Figure 6.

The satisfaction problem of a Petri net N for a scenario-based specification $\mathcal{S}_B(D_1, D_2)$ is defined formally as follows. N satisfies $\mathcal{S}_B(D_1, D_2)$ if any run σ of N of the form $\sigma = \mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$ satisfies the following condition:

- if there is a subsequence σ_1 in σ of the form

$$\sigma_1 = \mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1} \ (0 \leq i < j \leq n)$$

  which is an exact image of D_1, then there is a subsequence σ_2 in σ of the form

$$\sigma_2 = \mu_k \xrightarrow{t_k} \mu_{k+1} \xrightarrow{t_{k+1}} \ldots \xrightarrow{t_{i-2}} \mu_{i-1} \xrightarrow{t_{i-1}} \mu_i \ (0 \leq k < i)$$

  which is an image of D_2.

Now we try to solve the verification problem based on the investigation of a finite set of finite runs. For any Petri net N, for any scenario-based specification $\mathcal{S}_B(D_1, D_2)$ where

$$D_1 = (U_1, N_1, E_1, succ_1, ref_1) \text{ and } D_2 = (U_2, N_2, E_2, succ_2, ref_2)\,,$$

let $\Delta(N, \mathcal{S}_B(D_1, D_2))$ be the set of the runs of N which are of the form

$$\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \ldots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}\,,$$

where

- all μ_i $(0 \leq i \leq k)$ are distinct;
- $\mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$ is an exact image of D_1, and satisfies the loop bounded condition;
- there is not any $t_l (m \leq l \leq n)$ such that $\varphi(t_l) = \phi(e)$ $(e \in E_2)$;
- for any μ_i and μ_j $(k \leq i < j \leq m)$, if there is not any $t_l (i \leq l \leq j)$ such that $\varphi(t_l) = \phi(e)$ $(e \in E_2)$ then $\mu_i \neq \mu_j$, and
- for any μ_h, μ_i and μ_j $(k \leq h < i < j \leq m)$, if μ_i=μ_h then $\mu_j \neq \mu_h$.

For any $\sigma \in \Delta(N, \mathcal{S}_B(D_1, D_2))$ of the above form, we call the subsequences of the form $\mu_k \xrightarrow{t_k} \mu_{k+1} \xrightarrow{t_{k+1}} \ldots \xrightarrow{t_{m-1}} \mu_m$ by the *front segments* of σ.

Theorem 3 A Petri net N satisfies a scenario-based specification $\mathcal{S}_B(D_1, D_2)$ if and only if for any run σ of N which is in $\Delta(N, \mathcal{S}_B(D_1, D_2))$, there is a front segment of σ which is an image of D_2, and satisfies the loop match condition. □

The proof of this theorem is presented in the appendix. For a Petri net N, for a scenario-based specification $\mathcal{S}_B(D_1, D_2)$, a run σ of N is a *prefix* for $\Delta(N, \mathcal{S}_B(D_1, D_2))$ if it may be extended into a run which is in $\Delta(N, \mathcal{S}_B(D_1, D_2))$, i.e. there could be a sequence σ_1 of markings and transitions such that $\sigma \hat{} \sigma_1$ is in $\Delta(N, \mathcal{S}_B(D_1, D_2))$. Based on Theorem 3, we

```
currentpath := ⟨μ0⟩;
repeat
   node := the last node of currentpath;
   if node has no new successive node
   then delete the last node of currentpath
   else begin
          node := a new successive node of node;
          if node is such that the run corresponding to currentpath
             is in Δ(N, S_B(D1, D2))
          then
            begin
              check if the run corresponding to currentpath is
              such that a front segment is an image of D2 and
              satisfies the loop match condition;
              if no then return false;
            end;
          if node is such that the run corresponding to currentpath
             is a prefix for Δ(N, S_B(D1, D2))
          then append node to currentpath;
        end
until currentpath = ⟨⟩;
return true.
```

Figure 11: Algorithm for backward mandatory consistency checking

can develop an algorithm to check if a Petri net $N = (P, T, F, \mu_0)$ satisfies a scenario-based specification $\mathcal{S}_B(D_1, D_2)$ (cf. Figure 12). The structure of the algorithm is the same as the one of the algorithm depicted in Figure 10. The complexity of the algorithm is proportional to the number of the prefixes for $\Delta(N, \mathcal{S}_B(D_1, D_2))$ and to the size of the longest prefix for $\Delta(N, \mathcal{S}_B(D_1, D_2))$.

5.3 Bidirectional Mandatory Consistency Checking

For the bidirectional mandatory consistency checking, a scenario-based specification, denoted by $\mathcal{S}_D(D_1, D_2, D_3)$, consists of three given IODs D_1, D_2, and D_3, which requires that if a scenario described by D_1 occurs in the run of a time Petri net and a scenario described by D_2 follows, then the run segment between these two scenarios must be exactly corresponding to a scenario described by D_3. For example, a bidirectional mandatory consistency specification for the railway crossing system is depicted in Figure 13, which requires that between the scenarios for confirming the train arriving and for permitting the train crossing, the scenario for lowering the barrier must exist exactly. This specification should be satisfied during the run of the Petri net depicted in Figure 6.

The satisfaction problem of a Petri net N for a scenario-based specification $\mathcal{S}_D(D_1, D_2, D_3)$ is defined formally as follows. N satisfies $\mathcal{S}_D(D_1, D_2, D_3)$ if for any run σ of N of the form

$$\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \ldots \xrightarrow{t_{l-1}} \mu_l \xrightarrow{t_l} \ldots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \ldots \xrightarrow{t_{n-1}} \mu_n$$

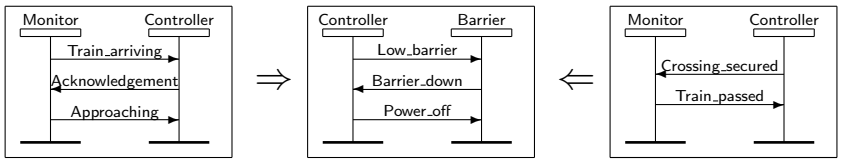


Figure 12: Bidirectional mandatory consistency specifications for the railway crossing system

where $\mu_k \xrightarrow{t_k} \mu_{k+1} \xrightarrow{t_{k+1}} \ldots \xrightarrow{t_{l-1}} \mu_l$ is an exact image of D_1, and $\mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \ldots \xrightarrow{t_{n-1}} \mu_n$ is an exact image of D_2, the following condition holds:

- if there is no subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j$ $(l \leq i < j \leq m)$ which is an image of D_1 or D_2, then $\mu_l \xrightarrow{t_l} \mu_{l+1} \xrightarrow{t_{l+1}} \ldots \xrightarrow{t_{m-1}} \mu_m$ is an image of D_3.

Now we try to solve the verification problem based on the investigation of a finite set of finite runs. For a Petri net N, for a scenario-based specification $\mathcal{S}_D(D_1, D_2, D_3)$ where $D_1 = (U_1, N_1, E_1, succ_1, ref_1)$, $D_2 = (U_2, N_2, E_2, succ_2, ref_2)$ and $D_3 = (U_3, N_3, E_3, succ_3, ref_3)$, let $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$ be the set of the runs of N which are of the form

$$\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \ldots \xrightarrow{t_{l-1}} \mu_l \xrightarrow{t_l} \ldots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \ldots \xrightarrow{t_{n-1}} \mu_n$$

where

- all μ_i $(0 \leq i \leq k)$ are distinct;
- $\mu_k \xrightarrow{t_k} \mu_{k+1} \xrightarrow{t_{k+1}} \ldots \xrightarrow{t_{l-1}} \mu_l$ is an exact image of D_1,and satisfies the loop bounded condition;
- there is not any $t_i (k \leq i < l)$ such that $\varphi(t_i) = \phi(e)$ $(e \in E_2 \vee e \in E_3)$;
- $\mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \ldots \xrightarrow{t_{n-1}} \mu_n$ is an exact image of D_2, and satisfies the loop bounded condition;
- there is not any $t_i (m \leq l < n)$ such that $\varphi(t_i) = \phi(e)$ $(e \in E_3)$;
- for any μ_i and μ_j $(l \leq i < j < m)$, if there is not any $t_p (i \leq p \leq j)$ such that $\varphi(t_p) = \phi(e)$ $(e \in E_3)$ then $\mu_i \neq \mu_j$;
- for any μ_h, μ_i and μ_j $(l \leq h < i < j < m)$, if μ_i=μ_h then $\mu_j \neq \mu_h$, and
- there is no subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j$ $(l \leq i < j \leq m)$ which is an image of D_1 or D_2.

```
currentpath := ⟨μ0⟩;
repeat
   node := the last node of currentpath;
   if node has no new successive node
   then delete the last node of currentpath
   else begin
          node := a new successive node of node;
          if node is such that the run corresponding to currentpath
             is in Δ(N, S_D(D1, D2, D3))
          then
            begin
              check if the run corresponding to currentpath is
              such that its middle segment is an image of D3 and
              satisfies the loop match condition;
              if no then return false;
            end;
          if node is such that the run corresponding to currentpath
             is a prefix for Δ(N, S_D(D1, D2, D3))
          then append node to currentpath;
        end
until currentpath = ⟨⟩;
return true.
```

Figure 13: Algorithm for bidirectional mandatory consistency checking

For any $\sigma \in \Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$ of the above form, we call the subsequence of the form $\mu_l \xrightarrow{t_l} \mu_{l+1} \xrightarrow{t_{l+1}} \ldots \xrightarrow{t_{m-2}} \mu_{m-1} \xrightarrow{t_{m-1}} \mu_m$ by the *middle segment* of σ.

Theorem 4 Let N be a Petri net. Then, N satisfies a scenario-based specification $\mathcal{S}_D(D_1, D_2, D_3)$ if and only if for any run of N which is in $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$, its middle segment is an image of D_3, and satisfies the loop match condition. □

The proof of this theorem is presented in the appendix. For a Petri net N, for a scenario-based specification $\mathcal{S}_D(D_1, D_2, D_3)$, a run σ of N is a *prefix* for $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$ if it may be extended into a run which is in $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$, i.e. there could be a sequence σ_1 of markings and transitions such that $\sigma \hat{} \sigma_1$ is in $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$. Based on Theorem 4, we can develop an algorithm to check if a Petri net $N = (P, T, F, \mu_0)$ satisfies a scenario-based specification $\mathcal{S}_D(D_1, D_2, D_3)$ (cf. Figure 14). The structure of the algorithm is the same as the one of the algorithm depicted in Figure 10. The complexity of the algorithm is proportional to the number of the prefixes for $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$ and to the size of the longest prefix for $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$.

6 Verification Tool Prototype

The algorithms presented in this paper have been implemented in a tool prototype. The tool is implemented in C and Java, and its graphical interface

allows the users to construct, edit, and analyze IODs and Petri nets interactively. The tool interface can also read .mdl files of UML sequence diagrams and activity diagrams in Rational Rose as specifications and design models respecivity.

On a PentiumM/1.50GHz/512MB PC, the tool runs comfortably for several case studies including the railroad crossing system, the automatic teller machine (ATM) system in [8], and the toaster example in [24]. In general, the algorithms presented in this paper are efficient for the consistency analysis. We have also developed several techniques to improve the algorithms for more efficiency, which have been implemented in the tool, but are not presented in the paper because of space consideration.

In this paper, for the mandatory consistency specifications, we interpret them in the *universal* view, that is, we require that if a reference scenario described by the given IODs occurs during the run of a Petri net, then it must immediately adhere to a scenario described by the other given IOD for *all* runs of the Petri net where it occurs. We can also interpret the mandatory consistency specifications in the *existential* view, that is, we require that if a reference scenario described by the given IODs occurs during the run of a Petri net, then it must immediately adhere to a scenario described by the other given IOD for *a* run of the Petri net where it occurs, which has also been implemented in the tool.

Similarly, in this paper the mandatory consistency specifications are interpreted in the *tight* view, that is, we require that if a reference scenario described by the given IODs occurs during the run of a Petri net, then it must *immediately* adhere to a scenario described by the other given IOD. The mandatory consistency specifications can also be interpreted in the *loose* view, that is, we require that if a reference scenario described by the given IODs occurs during the run of a Petri net, then it must adhere to, *possibly intermittently*, a scenario described by the other given IOD, which has also been implemented in our tool.

7 Related Work and Conclusion

In this paper, we solve the consistency checking problems of the concurrent system designs modelled by Petri nets for the scenario-based specifications expressed by IODs. The algorithms we present can be used to check if a scenario described by a given IOD must happen during the the run of a Petri net; if any forbidden scenario described by a given IOD never happens during the run of a Petri net; and if a Petri net satisfies a mandatory consistency specification expressed by IODs which requires that if a reference scenario described by the given IODs occurs during the run of the Petri net runs, it must immediately adhere to a scenario described by the other given IOD.

To our knowledge, there has been few literature on consistency checking

of Petri nets for the scenario-based specifications expressed by IODs. A work closed to our own is described in [11] where a tool, HUGO, is designed to check if the interactions expressed by an UML collaboration diagram can indeed be realized by a set of UML state machines in which the state machines are compiled into SPIN [12] input model and the collaboration diagrams are translated into sets of *Büchi* automata, and SPIN is called for the verification. Compared to that work, we consider more expressive mandatory consistency specifications, and our approach leads to direct and efficient implementation, which avoids the transformation and the generation of the state space altogether and also the involved complexity.

There have been a number of work on checking the state-transition graph based models for the properties expressed by the temporal logics (CTL, LTL)[13]. It is well known that the state-transition graphs and the Petri nets considered in this paper can be interchangeable, and the temporal logics can be used to interpret MSCs in many cases. It follows that theoretically the problems considered in this paper could be solved by converting to the corresponding formalisms. Also there have been several works on checking Petri nets for the temporal logic based properties [15-17]. Compared to the converting-based approach, on one hand our approach establishes the algorithms directly based on the specifications expressed by MSCs and Petri nets themselves so as to avoid the transformation and the generation of the state space altogether and also the involved complexity. On the other hand, it is difficult and not a natural way for the temporal logics to be used to describe the scenario-based specifications considered in this paper in some cases such that an event is allowed to occur many times in a specification and that the scenarios are required to adhere immediately with each other in a mandatory consistency specification. Furthermore, we know that it is not easy to use formal verification techniques directly in industry because the modelling languages in the verification tools are too formal and theoretical to master easily. For industry, it is much more acceptable to adopt MSCs as a specification language instead of the temporal logics in formal verification tools.

Live sequence charts (LSCs) [18] is a more expressive formalism than MSCs, but they are much less popular in industry than MSCs. There have been several works on LSC based testing and verification [19-21], but the mandatory consistency checking problem is not considered in those works.

In this paper, our work focuses on providing algorithms to checking Petri nets for scenario-based specifications described by MSCs, but the underlying approach and ideas are more general and can also be applied to the development of algorithms and tools for checking the state-based models such as automata and UML models (statechart diagrams and activity diagrams) for the scenario-based specifications described by UML sequence diagrams and by LSCs.

In this paper, since the specifications we concern usually come from the

scenario-based requirements provided directly by the customers, which is incomplete, we just use bMSCs to describe the scenario-based specifications. For describing the more complete scenario-based specifications, we need to consider hMSC, which is one of our next works.

References

[1] ITU-T. Recommendation Z.120. ITU - Telecommunication Standardization Sector, Geneva, Switzerland, May 1996.

[2] OMG. *UML2.0 Superstructure Specification*. http://www.uml.org, Oct. 2005.

[3] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

[4] J.L. Peterson. *Petri Nets Theory and the Modeling of Systems*. Prentice-Hall, N.J., 1981.

[5] Olaf Kluge. Modelling a Railway Crossing with Message Sequence Chatrs and Petri Nets. In H.Ehrig et al.(Eds.): *Petri Technology for Communication-Based Systems - Advance in Petri Nets*, LNCS 2472, Springer, 2003, pp.197-218.

[6] van der Aalst, W.M.P. Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets. In *Systems Analysis - Modelling - Simulation*, Vol.34, No.3, pages 335-367. 1999.

[7] Uwe Rueppel, Udo F. Meissner, and Steffen Greb. A Petri Net based Method for Distributed Process Modelling in Structural Engineering. In *Proc. International Conference on Computing in Civil and Building Engineering*, 2004.

[8] R. Alur, G.J. Holzmann, D. Peled. An Analyzer for Message Sequence Charts. In *Software-Concepts and Tools* (1996) 17: 70-77.

[9] Hanene Ben-Abdallah and Stefan Leue. Timing Constraints in Message Sequence Chart Specifications. In *Proceedings of FORTE/PSTV'97*, Chapman & Hall, 1997.

[10] Doron A. Peled. *Software Reliability Methods*. Springer, 2001.

[11] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Comparing Different Approaches for Specifying and Verifying Real-Time Systems. In *Proc.* 10_{th} *IEEE Workshop on Real-Time Operating Syatems abd Software*. New York, 1993. pp.122-129.

[12] Timm Schafer, Alexander Knapp, and Stephan Merz. Model Checking UML State Machines and Collaborations. In *Electronic Notes in Theoretical Computer Science* 55, No.3, 2001.

[13] G.J. Holzmann. *The SPIN Model Checker, Primer and Reference Manual*, Addison-Wesley, 2003.

[14] Edmund M. Clarke, Jr. Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 2000.

[15] Simona Bernardi, Susanna Donatelli, and Jos Merseguer. ¿From UML sequence diagrams and statecharts to analysable petri net models. In *Proceedings of the third international workshop on Software and performance*, ACM Press, 2002, pp.35-45.

[16] Andrea Bobbio, Andras Horvath. Model Checking Time Petri Nets using NuSMV. In *Proccedings of the Fifth International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS2001)*, 2001.

[17] Furfaro A. and Nigro L. Model Checking Time Petri Nets: A Translation Approach based on Uppaal and a Case Study. In *Proceedings of IASTED International Conference on Software Engineering(SE 2005)*, Innsbruck, Austria, Acta Press, 2005.

[18] Tomohiro Yoneda, Hikaru Ryuba. CTL Model Checking of Time Petri Nets using Geometric Regions. In *IEICE Trans. INF. & SYST.*, Vol.E99-D, No.3, 1998, pp.1-11.

[19] Werner Damm and David Harel. LSCs: Breathing life into message sequence charts. In *Formal Methods in System Design*, 19(1):45-80, 2001.

[20] Yves Bontemps and Patrick Heymans. Turning High-Level Live Sequence Charts into Automata. In *Proceedings of Workshop: Scenarios and State-Machines*, 2002.

[21] J. Klose and H. Wittke. An automata based interpretation of live sequence charts. In *Proceedings of 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, LNCS 2031, Springer, 2001.

[22] M. Lettrai and J. Klose. Scenario-based monitoring and testing of real-time UML models. In Proceedings of 4th International Conference on Unified Modeling Language (UML2001), LNCS 2185, Springer, 2001.

[23] Alexander Knapp, Stephan Merz, and Christopher Rauh. Modelchecking Timed UML State Machines and Collaborations. In W. Damm and E.-R. Olderog (Eds.): *FTRTFT2002*, LNCS 2469, Springer, 2002, pp.395-414.

[24] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. In *International Journal of Software Tools for Technology Transfer*, 1(1-2): 134-152, 1997.

[25] Stefan Leue, Lars Mehrmann, and Mohammad Rezai. Synthesizing ROOM Models from Message Sequence Chart Specifications.

A Proofs of Lemma and Theorems

Lemma 1 Let $D = (U, N, E, succ, ref)$ be a IOD, N be a Petri net, σ be an run of N. If σ is an exact image of a path segment of D and at the same time σ satisfies the loop bounded condition, then σ is finite, and its length has a upper bound.

Proof. Suppose that σ is the exact image of of D. Assume that σ is infinite. Thus a path ρ of D of which σ is the image must be infinite as well, for all the loops of σ which contains no transitions relevant to the D are omitted. Therefore there must be infinite repetitions of one or more loops in ρ in order to make ρ infinite. Without losing generality, let ρ_1 of the form $v_0 \hat{} v_1 \hat{} \ldots \hat{} v_n$ be one of the loops which repeat infinite times, and $e_0 \hat{} e_1 \hat{} \ldots \hat{} e_m$ be the trace of ρ_1. There is a infinite event sequence as

$$e_0 \hat{} e_1 \hat{} \ldots \hat{} e_m \hat{} e_0 \hat{} e_1 \hat{} \ldots \hat{} e_m \hat{} e_0 \ldots$$

Because σ satisfies the loop bounded condition, each time e_0 appears in the sequence, the marking μ in which transition t is enabled ($\psi(t) = \phi(e_0)$) must be different from each other. However, the number of different markings in N is finite, which leads to a contradiction.

The number of all the markings where transition t is enabled ($\psi(t) = \phi(e_0)$) bounds the repeating times of the loops in ρ. Therefore, the length of σ is also bounded. □

Theorem 1 Let N be a Petri net, and D be a IOD. Then there is a run of N where a scenario described by M occurs if and only if $\Delta(N, D) \neq \emptyset$.

Proof. It is clear that one half of the claim holds: if $\Delta(N, D) \neq \emptyset$, then there is a run of N where a scenario described by M occurs. The other half of claim can be proved as follows. Let $D = (U, N, E, succ, ref)$. Suppose that σ is a run of N where a scenario described by D occurs. Without losing generality, let σ of the form

$$\sigma = \mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$$

where $\sigma_1 = \mu_m \xrightarrow{t_m} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$ is an exact image of a path ρ of D. For any μ_i and μ_j $(0 \leq i < j < m)$ such that $\mu_i = \mu_j$, by removing the subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ from σ we can get a run of N where a scenario described by D occurs. For any μ_i and μ_j $(m < i < j < n)$ such that $\mu_i = \mu_j$ and that there is not any $t_k (i < k < j)$ such that $\psi(t_k) = \phi(e)$ $(e \in E)$, by removing the subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ from σ we can get a run of N where a scenario described by D occurs.

For any subsequence σ_1 of σ in the form of $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1} \xrightarrow{t_{j+1}} \ldots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \mu_{k+1}$ $(m \leq i < j < k \leq n)$ if $\mu_i = \mu_j$ and that

σ_1 being an image of a path segment $\rho{=}v_p\hat{}v_{p+1}\hat{}\ldots\hat{}v_q$, where $\mu_{j+1} \xrightarrow{t_{j+1}} \ldots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \mu_{k+1}$ is the image of $ref(v_q)$. If $v_q{=}v_q$ and $\mu_i{=}\mu_{j+1}$, by removing $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j}$ from σ we get a run of N where a scenario described by D occurs. Thus, we can construct a run in $\Delta(N, D)$ from σ, which implies the claim holds. □

Theorem 2 A Petri net N satisfies a scenario-based specification $\mathcal{S}_F(D_1, D_2)$ if and only if for any run of N which is in $\Delta(N, \mathcal{S}_F(D_1, D_2))$, its last segment is an image of D_2 and satisfies the loop match condition.

Proof. We first prove that one half of the claim holds. If N satisfies $\mathcal{S}_F(D_1, D_2)$, then for any run σ of N which is in $\Delta(N, \mathcal{S}_F(D_1, D_2))$, it is clear that the last segment of σ is an image of D_2. Assume that σ does not satisfy the loop match condition. Then by repeating all the loops infinite times or removing all the loops in σ we can easily get a run which is not an image of D_2, which would be a contradiction. The other half of claim can be proved as follows. Suppose that $D_1 = (U_1, N_1, E_1, succ_1, ref_1)$ and $D_2 = (U_2, N_2, E_2, succ_2, ref_2)$. Let σ be an run of N of the form

$$\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \ldots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}\,,$$

where the subsequence $\mu_k \xrightarrow{k_i} \mu_{k+1} \xrightarrow{t_{k+1}} \ldots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \mu_{m+1}$ is an exact image of D_1 in which there is not any $t_l (k \le l \le m)$ such that $\varphi(t_l) = \phi(e)$ $(e \in E_2)$. By applying the following steps in order:

- Step1: for any μ_i and μ_j $(0 \le i < j \le k)$ such that $\mu_i = \mu_j$, by removing the subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ from σ we can get a run of N,
- Step2: for any μ_i and μ_j $(k \le i < j < m)$ such that $\mu_i = \mu_j$ and that there is not any $t_l (i \le l \le j)$ such that $\psi(t_l) = \phi(e)$ $(e \in E_1)$, by removing the subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ from σ we can get a run of N;
- Step3: for any subsequence σ_1 of σ in the form of $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1} \xrightarrow{t_{j+1}} \ldots \xrightarrow{t_{h-1}} \mu_h \xrightarrow{t_h} \mu_{h+1}$ $(k \le i < j < h \le m)$ such that σ_1 is an image of a path segment $v_p\hat{}v_{p+1}\hat{}\ldots\hat{}v_q$ and $\mu_{j+1} \xrightarrow{t_{j+1}} \ldots \xrightarrow{t_{h-1}} \mu_h \xrightarrow{t_h} \mu_{h+1}$ is the image of $ref(v_q)$. If $v_p{=}v_q$ and $\mu_i{=}\mu_{j+1}$, by removing $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j}$ from σ we get a run of N,
- Step4: for any μ_i and μ_j $(m < i < j \le n)$ such that $\mu_i = \mu_j$ and that there is not any $t_l (i \le l \le j)$ such that $\psi(t_l) = \phi(e)$ $(e \in E_2)$, by removing the subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ from σ we can get a run of N, and

- Step5: traverse in σ from left to right. For any μ_h, μ_i and μ_j ($m < h < i < j \leq n$), if μ_h=μ_i=μ_j, by removing the subsequence $\mu_h \xrightarrow{t_h} \mu_{h+1} \xrightarrow{t_{h+1}} \ldots \xrightarrow{t_{i-2}} \mu_{i-1} \xrightarrow{t_{i-1}}$ from σ we can get a run of N.

we can construct a run from σ which is in $\Delta(N, \mathcal{S}_F(D_1, D_2))$. Now we prove the subsequence σ_2=$\mu_{m+1} \xrightarrow{t_{m+1}} \mu_{m+2} \xrightarrow{t_{m+2}} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$ is an image of D_2. For step 1 to step 4, we do not affect the essential label trace of σ_2, so we just need to scrutinize step 5. Each time we remove a loop from σ in step 5, we record it in a sequence. Suppose σ_2' is the final result of σ_2 after completing five steps above, then σ_2' is an image of D_2. Now we add the loops we removed previously to σ_2' in the reverse order, i.e. the last removed loop is added first. We prove σ_2 is an image of D_2 by induction. Suppose we have removed totally d loops.

- Basis: σ_2' is an image of D_2;
- I.H.: We have added b ($0 \leq b < d$) loops and get a sequence σ_2'' that is an image of D. Then there exists a event sequence $\tau = e_{m+1}\hat{}e_{m+2}\hat{}\ldots e_s\hat{}e_{s+1}\hat{}\ldots\hat{}e_q$ where the essential label trace of σ_2'' equals to $\phi(e_{m+1})\hat{}\phi(e_{m+2})\hat{}\ldots\hat{}\phi(e_s)\hat{}\phi(e_{st+1})\hat{}\ldots\hat{}\phi(e_q)$.
- I.S.: Now we add the $b+1$ th loop of the form $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$. Without losing generality, suppose we add the loop before μ_s ($m < s \leq n$). According to Step 5, μ_i=μ_s. Since all the loops we have added so far are to the right of $b+1$ th loop, we can be sure that there are no μ_h, μ_i and μ_j ($m < h < i < j < s$) such that μ_h=μ_i=μ_j. Therefore we can find a run in $\Delta(N, \mathcal{S}_F(D_1, D_2))$ whose prefix is exactly the same as σ_2'' from μ_0 to μ_s (exclusive), with $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ followed immediately. Based on the loop match condition, we can extract the event sequence $e_i\hat{}e_{i+1}\hat{}\ldots e_{j-2}\hat{}e_{j-1}$ where $\varphi(t_i)\hat{}\varphi(t_{i+1})\hat{}\ldots\hat{}\varphi(t_{j-1}) = \phi(e_i)\hat{}\phi(e_{i+1})\hat{}\ldots\hat{}\phi(e_{j-1})$. Suppose the transition fired after μ_s is t_s, and there is an event e_s in τ satisfying $\varphi(t_s) = \phi(e_s)$. Note that e_s=e_i and e_s is one of the successive events of e_{j-1} in D_2 according to the loop match condition. Thus while we add the $b+1$ th loop and get a new run σ_2^*, we can insert the event sequence $e_i\hat{}e_{i+1}\hat{}\ldots e_{j-2}\hat{}e_{j-1}$ before e_s and get a new event sequence τ^*. It is obvious that τ^* represents a behavior of D_2 and thus σ_2^* is an image of D_2.

Therefore, the claim holds. □

Theorem 3 A Petri net N satisfies a scenario-based specification $\mathcal{S}_B(D_1, D_2)$ if and only if for any run σ of N which is in $\Delta(N, \mathcal{S}_B(D_1, D_2))$, there is a front segment of σ which is an image of D_2, and satisfies the loop match

condition.

Proof. It is clear that one half of the claim holds: if N satisfies $\mathcal{S}_B(D_1, D_2)$, then for any run σ of N which is in $\Delta(N, \mathcal{S}_B(D_1, D_2))$, there is a front segment σ_1 of σ which is an image of D_2. Assume that σ does not satisfy the loop match condition. Then by repeating all the loops infinite times or removing all the loops in σ we can easily get a run which is not an image of D_2, which would be a contradiction. The other half of claim can be proved as follows. Let $D_1 = (U_1, N_1, E_1, succ_1, ref_1)$, $D_2 = (U_2, N_2, E_2, succ_2, ref_2)$, and σ be a run of N of the form

$$\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \ldots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}\,,$$

where the subsequence in σ of the form $\mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \ldots \xrightarrow{t_{n-1}} \mu_n \xrightarrow{t_n} \mu_{n+1}$ $(0 \leq m < n)$ is an exact image of D_1, and there is not any $t_i (m \leq i < n)$ such that $\varphi(t_i) = \phi(e)$ $(e \in E_2)$. By applying

- for any μ_i and μ_j $(0 \leq i < j \leq k)$ such that $\mu_i = \mu_j$, by removing the subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ from σ we can get a run of N,
- for any μ_i and μ_j $(m \leq i < j \leq n)$ such that $\mu_i = \mu_j$ and that there is not any $t_l (i \leq l \leq j)$ such that $\psi(t_l) = \phi(e)$ $(e \in E_1)$, by removing the subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ from σ we can get a run of N,
- for any subsequence σ_1 of σ in the form of $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1} \xrightarrow{t_{j+1}} \ldots \xrightarrow{t_{h-1}} \mu_h \xrightarrow{t_h} \mu_{h+1}$ $(m \leq i < j < h \leq n)$ such that σ_1 is an image of a path segment $v_p \hat{} v_{p+1} \hat{} \ldots \hat{} v_q$ and $\mu_{j+1} \xrightarrow{t_{j+1}} \ldots \xrightarrow{t_{h-1}} \mu_h \xrightarrow{t_h} \mu_{h+1}$ is the image of $ref(v_q)$. If $v_p{=}v_q$ and $\mu_i{=}\mu_{j+1}$, by removing $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j}$ from σ we get a run of N,
- for any μ_i and μ_j $(k \leq i < j < m)$ such that $\mu_i = \mu_j$ and that there is not any $t_l (i \leq l \leq j)$ such that $\psi(t_l) = \phi(e)$ $(e \in E_2)$, by removing the subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ from σ we can get a run of N, and
- for any μ_h, μ_i and μ_j $(k \leq h < i < j < m)$, if $\mu_h{=}\mu_i{=}\mu_j$, by removing the subsequence $\mu_h \xrightarrow{t_h} \mu_{h+1} \xrightarrow{t_{h+1}} \ldots \xrightarrow{t_{i-2}} \mu_{i-1} \xrightarrow{t_{i-1}}$ from σ we can get a run of N,

we can construct a run from σ which is in $\Delta(N, \mathcal{S}_B(D_1, D_2))$. Similar to the proving process of Theorem 2 we can conclude that $\mu_k \xrightarrow{t_k} \ldots \xrightarrow{t_{m-1}} \mu_m$ is an image of D_2. Thus, the claim holds. □

Theorem 4 Let N be a Petri net. Then, N satisfies a scenario-based specification $\mathcal{S}_D(D_1, D_2, D_3)$ if and only if for any run of N which is in $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$, its middle segment is an image of D_3.

Proof. It is clear that one half of the claim holds: if N satisfies $\mathcal{S}_D(D_1, D_2, D_3)$, then for any run σ of N which is in $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$, its middle segment is an image of D_3. Assume that σ does not satisfy the loop match condition. Then by repeating all the loops infinite times or removing all the loops in σ we can easily get a run which is not an image of D_3, which would be a contradiction. The other half of claim can be proved as follows. Let $D_1 = (U_1, N_1, E_1, succ_1, ref_1)$ and $D_2 = (U_2, N_2, E_2, succ_2, ref_2)$. Let σ be a run of N of the form

$$\mu_0 \xrightarrow{t_0} \mu_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{k-1}} \mu_k \xrightarrow{t_k} \ldots \xrightarrow{t_{l-1}} \mu_l \xrightarrow{t_l} \ldots \xrightarrow{t_{m-1}} \mu_m \xrightarrow{t_m} \ldots \xrightarrow{t_{n-1}} \mu_n$$

where $\mu_k \xrightarrow{t_k} \mu_{k+1} \xrightarrow{t_{k+1}} \ldots \xrightarrow{t_{l-1}} \mu_l$ is an exact image of D_1, and there is not any $t_i (k \leq i < l)$ such that $\varphi(t_i) = \phi(e)$ $(e \in E_2 \vee e \in E_3)$; $\mu_m \xrightarrow{t_m} \mu_{m+1} \xrightarrow{t_{m+1}} \ldots \xrightarrow{t_{n-1}} \mu_n$ is an exact image of D_2, and there is not any $t_i (m \leq l < n)$ such that $\varphi(t_i) = \phi(e)$ $(e \in E_3)$; there is no subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j$ $(l \leq i < j \leq m)$ which is an image of D_1 or D_2. By applying

- for any μ_i and μ_j $(0 \leq i < j \leq k)$ such that $\mu_i = \mu_j$, by removing the subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ from σ we can get a run of N,
- for any μ_i and μ_j $(k \leq i < j \leq l)$ such that $\mu_i = \mu_j$ and that there is not any $t_a (i \leq a \leq j)$ such that $\psi(t_a) = \phi(e)$ $(e \in E_1)$, by removing the subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ from σ we can get a run of N,
- for any subsequence σ_1 of σ in the form of $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1} \xrightarrow{t_{j+1}} \ldots \xrightarrow{t_{h-1}} \mu_h \xrightarrow{t_h} \mu_{h+1}$ $(k \leq i < j < h \leq l)$ such that σ_1 is an image of a path segment $v_p \hat{} v_{p+1} \hat{} \ldots \hat{} v_q$ and $\mu_{j+1} \xrightarrow{t_{j+1}} \ldots \xrightarrow{t_{h-1}} \mu_h \xrightarrow{t_h} \mu_{h+1}$ is the image of $ref(v_q)$. If $v_p{=}v_q$ and $\mu_i{=}\mu_{j+1}$, by removing $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j}$ from σ we get a run of N,
- for any μ_i and μ_j $(m < i < j < n)$ such that $\mu_i = \mu_j$ and that there is not any $t_a (i \leq a \leq j)$ such that $\psi(t_a) = \phi(e)$ $(e \in E_2)$, by removing the subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ from σ we can get a run of N,
- for any subsequence σ_1 of σ in the form of $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j} \mu_{j+1} \xrightarrow{t_{j+1}} \ldots \xrightarrow{t_{h-1}} \mu_h \xrightarrow{t_h} \mu_{h+1}$ $(m \leq i < j < h \leq n)$ such

that σ_1 is an image of a path segment $v_p \hat{} v_{p+1} \hat{} \ldots \hat{} v_q$ and $\mu_{j+1} \xrightarrow{t_{j+1}} \ldots \xrightarrow{t_{h-1}} \mu_h \xrightarrow{t_h} \mu_{h+1}$ is the image of $ref(v_q)$. If v_p=v_q and μ_i=μ_{j+1}, by removing $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-1}} \mu_j \xrightarrow{t_j}$ from σ we get a run of N,

- for any μ_i and μ_j $(l \leq i < j < m)$ such that $\mu_i = \mu_j$ and that there is not any $t_l (i \leq l \leq j)$ such that $\psi(t_l) = \phi(e)$ $(e \in E_3)$, by removing the subsequence $\mu_i \xrightarrow{t_i} \mu_{i+1} \xrightarrow{t_{i+1}} \ldots \xrightarrow{t_{j-2}} \mu_{j-1} \xrightarrow{t_{j-1}}$ from σ we can get a run of N, and

- for any μ_h, μ_i and μ_j $(l \leq h < i < j \leq m)$, if μ_h=μ_i=μ_j, by removing the subsequence $\mu_h \xrightarrow{t_h} \mu_{h+1} \xrightarrow{t_{h+1}} \ldots \xrightarrow{t_{i-2}} \mu_{i-1} \xrightarrow{t_{i-1}}$ from σ we can get a run of N,

we can construct a run from σ which is in $\Delta(N, \mathcal{S}_D(D_1, D_2, D_3))$. Similar to the proving process of Theorem 2 we can conclude that $\mu_l \xrightarrow{t_l} \ldots \xrightarrow{t_{m-1}} \mu_m$ is an image of D_3. Thus, the claim holds. □