Timing Analysis of Scenario-Based Specifications using Linear Programming

Li Xuandong, Pan Minxue, Bu Lei
Wang Linzhang, and Zhao Jianhua
Department of Computer Science and Technology
Nanjing University, Nanjing, P.R.China 210093
lxd@nju.edu.cn

Abstract

Scenario-based specifications such as UML interaction models offer an intuitive and visual way of describing design requirements, and are playing an increasingly important role in the design of software systems. In this paper, for specification and verification of real-time systems, we introduce more general and expressive timing constraints in UML sequence diagrams, and give an approach to timing analysis of scenario-based specifications expressed by UML2.0 interaction overview diagrams. Based on linear programming, we solve the reachability analysis, constraint conformance analysis, and bounded delay analysis problems, and give the algorithms for the three problems which are a decision procedure for a class of scenario-based specifications where no timing constraint is enforced on the repetition of any loop and one unfolding of any loop is time independent of the other unfolding, and a semi-decision procedure for general scenario-based specifications.

Key Words: Real-time systems, software verification, model checking, UML interaction models, bounded delay analysis.

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 (MSCs) [1] and UML interaction models [2,3] offer an intuitive and visual way of describing design requirements. They are playing an increasingly important
role in the design of software systems. Such specifications focus on message exchanges among communicating entities in distributed software systems. In this paper, we consider timing analysis of scenario-based specifications modelled by UML interaction models.

UML sequence diagrams form a class of important UML interaction models. Each of them 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]. For example, a UML sequence diagram is depicted in Figure 1(1), which describes a scenario about the well-known example of the railroad crossing system in [4,5]. This system operates a gate at a railroad crossing, in which there are a railroad crossing monitor and a gate controller. When the monitor detects that a train is arriving, it sends a message to the controller to move down the gate. After the train leaves the crossing, the monitor sends a message to controller to open the gate.

In this paper, we just use a simplified version of UML sequence diagrams, which describes exactly one scenario without any alternatives and loops. For describing multiple scenarios and complete system specifications, we use a simplified version of UML2.0 interaction overview diagrams [3], 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 1(2) depicts a simple interaction overview diagram.

For specifying real-time systems, timing constraints are enforced on scenario-based specifications. Several mechanisms have been introduced to describe timing constraints in MSCs and UML sequence diagrams, which are timers [1], interval delays [6,7], and timing marks [2,3,8,9]. All of those mecha-
nisms are just suitable to describe simple timing constraints which are only related to the separation in time between two events. For example, for the sequence diagram depicted in Figure 1(1), we can use timers, interval delays, or timing marks to describe simple timing constraints such as the separation in time between the sending events $e_1$ and $e_{13}$ is not smaller than 100 time units. However, in practical problems we often need to describe a class of more complex timing constraints which are about the relation among multiple separations in time between events. For example, for the railroad crossing system, a utility property we require is that within certain tolerance intervals, the gate stay open for a certain period, e.g., we require that from the time one train is arriving to the time the next train is arriving, the gate stay open for at least half of this period, which means that the separation in time between the sending event $e_{13}$ and the sending event $e_1$ is not greater than two times the one between the sending event $e_{13}$ and the receiving event $e_{12}$.

It is clear that using the existing mechanisms in MSCs and UML sequence diagrams we cannot describe such a timing constraint. In this paper, we introduce a more expressive mechanism in UML sequence diagrams to describe timing constraints, and give an approach to checking scenario-based specifications with more complex timing constraints.

Like any other aspect of the specification and design process, scenario-based specifications are amenable to errors, and their analysis is important. In [6], a variety of semantic interpretations for MSCs are investigated, and an analyzer for basic MSCs is described. A comprehensive study of model checking of MSCs for temporal requirements has been given in [10]. The tool in [11-13] supports to the design MSCs, and allow for the creation, debugging, organization, and maintenance of MSCs. In [14,15], a tool is developed for searching a hierarchical MSC design for a path that match a given specification. For scenario-based specifications with timing constraints used to describe real-time systems, the verification problems are more difficult and complicated. In [6], some algorithms for analyzing basic MSCs with interval delays are presented, and a corresponding tool is described. In [7,16,17], timing analysis is extended to check UML sequence diagrams and MSC specifications. In [18], a solution is given to the timed analogue of scenario matching. However, all those papers are about checking scenario-based specifications for timing consistency which is a basic property. In the view of practical use, there are a lot of properties about the accumulated delays on the traces of systems. For example, we often need to check if all traces of a system satisfy that the separation in time between two given events is in a given time interval, which is called bounded delay analysis. This problem has been considered for timed automata in [19], and for a class of Petri nets in [20].

In this paper, we consider timing analysis of scenario-based specifications expressed by UML interaction models, and give a linear programming based
solution. For the reachability analysis, constraint conformance analysis, and bounded delay analysis problems, we present the algorithms which are a decision procedure for a class of scenario-based specifications where no timing constraint is enforced on the repetition of any loop and one unfolding of any loop is time independent of the other unfolding, and a semi-decision procedure for general scenario-based specifications.

The paper is organized as follows. In the next section, we introduce scenario-based specifications expressed by UML interaction models. Section 3 gives the linear programming based approach to timing analysis of scenario-based specifications. The last section discusses the related work and contains some conclusions.

2 Scenario-Based Specifications

In this paper, UML interaction models are used as scenario-based specifications, which consist of UML2.0 interaction overview diagrams and sequence diagrams.

2.1 UML Sequence Diagrams and Timing Constraints

In this paper, we just use a simplified version of UML sequence diagrams, which describes exactly one scenario without any alternatives and loops. 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.

In a sequence diagram, by events we mean the message sending and the message receiving. Here we introduce more general and expressive timing constraints in sequence diagrams. In a sequence diagram, we use event names to represent the occurrence time of events, and linear inequalities on event names to represent the timing constraints. A timing constraint is of the form

\[ a \leq c_0(e_0 - e'_0) + c_1(e_1 - e'_1) + \ldots + c_n(e_n - e'_n) \leq b \]

where \( e_i \) and \( e'_i \) (0 ≤ i ≤ n) are event names which represent the occurrence time of \( e_i \) and \( e'_i \), \( a, b \) and \( c_0, c_1, \ldots, c_n \) are real numbers (\( b \) may be \( \infty \)). For example, for the scenario of the railroad crossing system depicted in Figure 1(1), we suppose that when a train has passed, a new train could come after at least 100 time units, which can be represented by the timing constraint \( 100 \leq e_{13} - e_1 < \infty \). Compared to timers, interval delays, and timing marks, the timing constraints we consider here can be used to describe more complex timing requirements in practical use. For example, for the scenario of the railroad crossing system depicted in Figure 1(1), the timing constraint
\[0 \leq 2(e_{13} - e_{12}) - (e_{13} - e_1) < \infty\]

specifies the requirement that from the time one train is arriving to the time the next train is arriving, the gate stay open for at least half of this period. Clearly, such a timing requirement is about the relation between two separations in time between events (one is the separation in time between \(e_{13}\) and \(e_{12}\), and the other is the separation in time between \(e_{13}\) and \(e_1\)), and timers, interval delays, and timing marks can not be used to describe such a timing requirement since they are only suitable to describe the simple timing constraints related to the separation in time between two events.

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 [1-3,21]. In accordance with [21], without losing generality, we assume that for a pair of events \(e\) and \(e'\) in a sequence diagram, \(e\) precedes \(e'\) (denoted by \(e < e'\)) in the following cases:

- **Causality**: A sending event \(e\) and its corresponding receiving event \(e'\).
- **Controllability**: The event \(e\) appears above the event \(e'\) on the same object column, and \(e'\) is a sending event.
- **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 analyzing scenario-based specifications, we formalize sequence diagrams as follows.

**Definition 1** A sequence diagram (SD) \(D\) is a tuple \(D = (O, E, M, L, V, C)\) where

- **\(O\)** is a finite set of objects.
- **\(E\)** is a finite set of events corresponding to sending a message and receiving a message. There are two special events \(\epsilon\) and \(\varpi\) in \(E\) which represent the start and end of \(D\) respectively.
- **\(M\)** is a finite set of messages whose elements are a pair \((e, e')\) where \(e, e' \in E\) are corresponding to the sending and the receiving for a message respectively.
- **\(L : E \rightarrow O\)** 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 \((e, e')\) \((e, e' \in E)\) such that \(e \prec e'\).

\(C\) is a finite set of timing constraints.

We use event sequences to represent the traces of SDs which are corresponding to the untimed behavior of SDs. An event sequence is of the form \(e_0 \rightarrow e_1 \rightarrow \ldots \rightarrow e_m\), which represents that \(e_{i+1}\) takes place after \(e_i\) for any \(i \ (0 \leq i \leq m - 1)\).

**Definition 2** Let \(D = (O, E, M, L, V, C)\) be an SD. An event sequence \(e_0 \rightarrow e_1 \rightarrow \ldots \rightarrow e_m\) is a trace of \(D\) if and only if the following conditions hold:

- \(e_0 = \epsilon\) and \(e_m = \omega\).
- \(e_0, e_1, \ldots, e_m\) is a permutation of the events in \(E\).
- \(e_0, e_1, \ldots, e_m\) satisfy the visual order defined by \(V\), i.e. for any \(e_i\) and \(e_j\), if \((e_i, e_j) \in V\), then \(0 \leq i < j \leq m\).

We use timed event sequences to represent the behaviour of SDs. A timed event sequence is of the form \((e_0, t_0) \rightarrow (e_1, t_1) \rightarrow \ldots \rightarrow (e_m, t_m)\) where \(e_i\) is an event and \(t_i\) is a nonnegative real numbers for any \(i \ (0 \leq i \leq m)\), which describes that \(e_0\) takes place \(t_0\) time units after the scenario starts, then \(e_1\) takes place \(t_1\) time units after \(e_0\) takes place, so on and so forth, at last \(e_m\) takes place \(t_m\) time units after \(e_{m-1}\) takes place. It follows that for any \(i \ (0 \leq i \leq m)\), the occurrence time of \(e_i\) is \(\sum_{j=0}^{i} t_j\).

**Definition 3** Let \(D = (O, E, M, L, V, C)\) be an SD. A timed event sequence \(\sigma = (e_0, t_0) \rightarrow (e_1, t_1) \rightarrow \ldots \rightarrow (e_m, t_m)\) is a behaviour of \(D\) if and only if the following conditions hold:

- \(e_0 \rightarrow e_1 \rightarrow \ldots \rightarrow e_m\) is a trace of \(D\).
- \(t_0, t_1, \ldots, t_m\) satisfy the timing constraints in \(C\), i.e. for any timing constraint \(a \leq \sum_{i=0}^{n} c_i (f_i - f_i') \leq b\) in \(C\), \(a \leq c_0 \delta_0 + c_1 \delta_1 + \ldots + c_n \delta_n \leq b\) where for each \(i \ (0 \leq i \leq n)\), if \(f_i = e_j\) and \(f_i' = e_k\), then

\[
\delta_i = \begin{cases} 
- (t_{j+1} + t_{j+2} + \ldots + t_{k}) & \text{if } j < k \\
(\ t_{k+1} + t_{k+2} + \ldots + t_{j}) & \text{if } j > k
\end{cases}
\]

Let \(L(D)\) denote the set of the timed event sequences representing the behaviour of \(D\).
2.2 UML2.0 Interaction Overview Diagrams and Scenario-Based Specifications

An SD considered in this paper just describes exactly one scenario. For describing multiple scenarios and complete system specifications, we need to use a simplified version of UML2.0 interaction overview diagrams [3], which focuses on the overview of the flow of control where the nodes are SDs. An interaction overview diagram defines a composition of a set of SDs, which describes potentially iterating and branching system behavior.

In this paper, we consider timing analysis of scenario-based specifications. A scenario-based specification under analyzing is represented by an interaction overview diagram, which is defined formally as follows.

**Definition 4** A scenario-based specification (SBS) $G$ is a tuple $G = (U, N, succ, ref, T)$ where

- $U$ is a finite set of SDs satisfying that for any $D = (O, E, M, L, V, C)$ and $D' = (O', E', M', L', V', C')$ in $U$, if $D \neq D'$, then $E \cap E' = \emptyset$.
- $N = \{\top\} \cup \{I\} \cup \{\bot\}$ 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.
- $succ \subseteq N \times N$ is the relation which reflects the connectivity of the nodes in $N$ (it is required that any node in $N$ be reachable from the start node).
- $ref : I \rightarrow U$ is a function that maps each intermediate node to an SD in $U$.
- $T$ is a finite set of timing constraints of the form $a \leq e - e' \leq b$ where $e$ and $e'$ occur in different SDs in $U$ and $0 \leq a \leq b$ ($b$ may be $\infty$), which are used to describe the timing constraints enforced between two events in different SDs in $U$.

For an SBS $G = (U, N, succ, ref, T)$, a path segment is a sequence of intermediate nodes $v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_n$ satisfying $(v_{i-1}, v_i) \in succ$ for any $i (0 < i \leq n)$. A path is a path segment $v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_n$ such that $(\top, v_0) \in succ$ and $(v_n, \bot) \in succ$.

We interpret the timing constraints in SBSs by local semantics: select one path at one time and analyze its timing requirements, independently of other paths that may branch out of the selected one. In UML2.0, interaction overview diagrams are defined as specializations of activity diagrams in a way that promotes overview of the control flow [3]. It follows that the concatenation of two SDs in an SBS should be interpreted as the synchronous mode: when moving one node to the other, all events in the previous SD
finish before any event in the following SD occurs, which is the same as
the synchronous interpretation of the concatenation of two basic MSCs in
MSC specifications [10]. Therefore, we define the behavior of an SBS \( G \) as
the timed event sequences which are the concatenation of the timed event
sequences representing the behavior of the SDs which make up \( G \).

**Definition 5** Let \( G = (U, N, succ, ref, T) \) be an SBS. For any path segment
\( \rho = v_0 \to v_1 \to \ldots \to v_m \) in \( G \), let \( \mathcal{L}(\rho) \) be the set of all timed event sequences
of the form \( (e_0, t_0) \to (e_1, t_1) \to \ldots \to (e_n, t_n) \) satisfying that
\[
\begin{align*}
&\bullet \ (e_0, t_0) \to (e_1, t_1) \to \ldots \to (e_n, t_n) = \sigma_0 \to \sigma_1 \to \ldots \to \sigma_m, \text{ where } \sigma_i \text{ is} \\
&\qquad \text{a behavior of } \text{ref}(v_i) \text{ for each } i (0 \leq i \leq m).
\end{align*}
\]
\[
\begin{align*}
&\bullet \ (e_0, t_0) \to (e_1, t_1) \to \ldots \to (e_n, t_n) \text{ satisfies any timing constraints in} \\
&\qquad T, \text{ i.e., for any } a \leq f - f' \leq b \in T, \text{ for any } i, j (0 \leq i < j \leq n) \text{ such}
\end{align*}
\]
\[
\begin{align*}
&\qquad \text{that } f' = e_i, f = e_j, \text{ and that } f \neq e_k \land f' \neq e_k \text{ for any } k (i < k < j),
\end{align*}
\]
\[
\begin{align*}
&\qquad a \leq t_{i+1} + t_{i+2} + \ldots + t_j \leq b.
\end{align*}
\]
A timed event sequence \( \sigma \) is a behavior of \( G \) if and only if there is a path \( \rho \)
in \( G \) such that \( \sigma \in \mathcal{L}(\rho) \). \qed

### 2.3 Loop-Unlimited Scenario-Based Specifications

For an SBS \( G = (U, N, succ, ref, T) \), a path segment is called **simple** if all its
nodes are distinct. Let \( v_0 \to v_1 \to \ldots \to v_n \) be a simple path segment in \( G \)
such that \( (T, v_i) \in succ \). If there is \( v_i (0 \leq i \leq n) \) such that \( (v_n, v_i) \in succ \),
then the sequence \( v_i \to v_{i+1} \to \ldots \to v_n \to v_i \) is a **loop**, and \( v_i \) is the
**loop-start node** of the loop.

For an SBS \( G = (U, N, succ, ref, T) \), let \( \rho = v_0 \to v_1 \to \ldots \to v_n \) be a
path segment. For a timing constraint \( a \leq e - e' \leq b \) in \( T \), if \( e \) occurs in
\( ref(v_i) \), \( e' \) occurs in \( ref(v_j) \) \( (0 \leq j < i \leq n) \), and \( e, e' \) do not occur in any
\( ref(v_k) \) \( (j < k < i) \), then we say this timing constraint combines nodes \( v_i \) and
\( v_j \) in \( \rho \) (in this case, the occurrence time of \( e \) in \( ref(v_i) \) and the occurrence
time of \( e' \) in \( ref(v_j) \) must satisfy this timing constraint). Figure 2 shows
certain cases that a timing constraint \( a \leq e - e' \leq b \) combines two nodes \( v \)
and \( v' \) in a path segment \( \rho \) with a loop.

In this paper, we develop the algorithms for timing analysis of SBSs.
These algorithms are a semi-decision procedure for general SBSs, and a
decision procedure for a class of SBSs which satisfy **loop-closed condition** and
**loop-unlimited condition**. For an SBS \( G = (U, N, succ, ref, T) \), loop-closed condition
requires that any timing constraint in \( T \) do not combine any two
nodes which are inside and outside of a loop respectively (in this case one
node is inside (outside) of a loop, and the other node is outside (inside) of
the loop, which is corresponding to the cases (1) and (2) in Figure 2), i.e.,
any timing constraint in \( T \) of the form \( a \leq e - e' \leq b \) must satisfy:
for any loop \( v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_m \) where \( e, e' \) do not occur in any \( ref(v_i) \) (0 \( \leq i \leq m \)), there is no simple path segment \( v'_0 \rightarrow v'_1 \rightarrow \ldots \rightarrow v'_n \) such that \( v'_n = v_0 \), \( e' \) occurs in \( ref(v'_0) \), and that \( e \) does not occur in any \( ref(v'_k) \) (0 \( \leq k \leq n \)); and

- for any loop \( v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_m \) where \( e, e' \) do not occur in any \( ref(v_i) \) (0 \( \leq i \leq m \)), there is no simple path segment \( v'_0 \rightarrow v'_1 \rightarrow \ldots \rightarrow v'_n \) such that \( v'_0 = v_0 \), \( e \) occurs in \( ref(v'_n) \), and that \( e', e' \) do not occur in any \( ref(v'_k) \) (0 \( \leq i \leq n \)).

For an SBS \( G = (U, N, succ, ref, T) \), loop-unlimited condition requires that no timing constraint be enforced on the repetition of any loop (the case (3) in Figure 2 is not allowed), i.e. any timing constraint in \( T \) of the form \( a \leq e - e' \leq b \) must satisfy:

- for any loop \( v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_m \) where \( e, e' \) do not occur in any \( ref(v_i) \) (0 \( \leq i \leq m \)), there is no simple path segment \( v'_0 \rightarrow v'_1 \rightarrow \ldots \rightarrow v'_n \) such that \( v'_k (0 < k < n) \) satisfying \( v'_k = v_0 \), \( e \) occurs in \( ref(v'_n) \), \( e' \) occurs in \( ref(v'_0) \), and that \( e, e' \) do not occur in any \( ref(v'_k) \) (0 \( \leq i < n \)).

Usually, for a loop in an SBS, its repetition will take time. It follows that if we enforce a timing constraint on the repetition of a loop such as the case (3) in Figure 2, the repetition of the loop will be restricted to a finite number of times. In this case, we can unfold the loop with the finite number of times and remove the timing constraint from the SBS. Therefore, the loop-unlimited condition is rational in many cases.

In an SBS \( G \) which satisfies the loop-unlimited condition, most of the cases violating the loop-closed condition can be removed by changing the structure of \( G \) lightly. Suppose that a timing constraint \( a \leq e - e' \leq b \) combines two nodes \( v \) and \( v' \) in \( G \) where \( v \) is outside (inside) a loop \( \rho \) and \( v' \) is inside (outside) \( \rho \) such as the cases (1) and (2) in Figure 3. If the node \( v \) (\( v' \)) outside \( \rho \) is not any node of \( \rho \), by unfolding \( \rho \) one time and
adjusting the structure involved we can remove the case violating the loop-closed condition without changing the behavior of $G$, which is illustrated by the cases (1') and (2') in Figure 3. However, this way is inapplicable to the case that the node $v$ ($v'$) outside $\rho$ is a node of $\rho$ oneself such as the case (3) in Figure 3. Essentially, this instance results from that the timing constraint $a \leq e - e' \leq b$ is a reverse constraint for the loop $\rho$, which is defined as follows. Suppose that $\rho = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_m$. $a \leq e - e' \leq b$ is a reverse constraint for $\rho$ if $e$ occurs in $\text{ref}(v_i)$, $e'$ occurs in $\text{ref}(v_j)$ ($0 \leq i < j \leq n$), and $e, e'$ do not occur in any $\text{ref}(v_k)$ ($0 \leq k < i, j < k < m$). In a path segment in $G$ which contains many unfolding of $\rho$, a reverse constraint for $\rho$ combines two nodes in conjunct two unfolding of $\rho$ so that one unfolding of $\rho$ is time dependent of the other unfolding, which is illustrated by the case (3') in Figure 3. Notice that though we could show a reverse constraint for $\rho$ in the structure of $G$ as the case (4) in Figure 2, the two nodes combined by the constraint are not intuitively shown in the structure of $G$ because they are not in just one unfolding of $\rho$. In our opinion, the reverse constraints are seldom adopted in practical use because they are inconsistent with our intuition thinking, and a class of constraints in common use are like the case (5) in Figure 2 (quite the reverse of the case 4 in Figure 2) which combines
two nodes in just one unfolding of a loop.

We say that an SBS $G$ is \textit{loop-unlimited} if it satisfies the loop-unlimited condition and there is no reverse constraint for any its loop (the cases (3) and (4) in Figure 2 are not allowed), i.e. no timing constraint is enforced on the repetition of any loop in $G$ and one unfolding of any loop in $G$ is time independent of the other unfolding. It follows that if an SBS is loop-unlimited then all the cases violating the loop-closed condition can be removed. Therefore, from now on we assume that any loop-unlimited SBS satisfies the loop-closed condition.

We can develop an efficient algorithm to check if an SBS $G$ is loop-unlimited and to remove all the cases violating the loop-closed condition when $G$ is loop-unlimited, which is described in the appendix.

### 2.4 An ATM Example

For illustrating the solution presented in this paper, here we introduce the automatic teller machine (ATM) system in [7] whose specification is depicted in Figure 4. This specification is a loop-unlimited SBS.

The ATM system consists of the three components: potential customers (User), the ATM controller (ATM), and a host computer in a bank (Bank). Initially, the ATM controller waits to receive the customer’s bank card and requests a pin number in $[0, 2]$ seconds after receiving a card ($0 \leq b_1 - a_2 \leq 2$, SDs $\text{StartTrans}$ and $\text{GetPin}$). Then, it either receives a request to cancel the transaction within $[0, 4]$ seconds ($0 \leq c_2 - b_1 \leq 4$, SD $\text{EndTrans}$), or receives the customer’s pin number with $[5, 60]$ seconds ($5 \leq d_2 - b_1 \leq 60$, SD $\text{ProcessPin}$). If the ATM receives a request to cancel the transaction, it returns the customer’s card and takes $[2, 3]$ seconds to return to its initial state ($2 \leq \varpi_c - c_3 \leq 3$, SD $\text{EndTrans}$). The ATM expects a reply from the bank within 10 seconds, which is represented by the following timing constraints:

$$
0 \leq f_2 - d_3 \leq 10, \quad 0 \leq g_2 - d_3 \leq 10, \quad 0 \leq j_2 - h_7 \leq 10 \\
0 \leq k_2 - h_7 \leq 10, \quad 0 \leq i_6 - i_3 \leq 10, \quad 0 \leq j_8 - j_5 \leq 10.
$$

If no reply from the bank is received in a delay of 10 seconds, the card is returned, an appropriate message is then displayed, and the ATM takes $[2, 3]$ seconds to return to its initial state ($e_1 - d_3 = 10$, $2 \leq \varpi_e - e_5 \leq 3$, SD $\text{TryAgain}$). The specification also describes the following constraints:

- a customer expects a withdraw request to be processed within $[0,W]$ seconds relative to the time of entering an amount ($0 \leq j_4 - h_5 \leq W, \ 0 \leq k_4 - h_5 \leq W$);

- the ATM takes $[B_1, B_2]$ seconds for book-keeping after dispensing cash ($B_1 \leq \varpi_j - j_9 \leq B_2$, SD $\text{DispenseCash}$);
Figure 4: Scenario-based specification for the ATM example
the ATM takes \([3, 5]\) seconds to print a receipt after receiving the balance information from the bank \((3 \leq j_b - j_8 \leq 5, 3 \leq i_9 - i_6 \leq 5, \text{SD \ DispenseCash, SD \ GetBalance})\); and

- in the case of refusing pin number, at the first time the ATM takes \([0, 2]\) seconds to request a pin number again after sending the formation for the invalid pin number \((0 \leq b_1 - g_3 \leq 2)\), and at the second time it takes \([3, 5]\) seconds to confiscate the card and inform the customer \((3 \leq l_1 - g_3 \leq 5, \text{SD \ ConfiscateCard})\).

Each ATM-customer communication takes at least \(T_1\) seconds, and each ATM-bank communication takes at least \(T_2\) seconds, which we do not explicitly represent in the chart.

3 Timing Analysis of Scenario-Based Specifications

In this section, we give the solutions to timing analysis of SBSs including the reachability analysis, the constraint conformance analysis, and the bounded delay analysis.

3.1 Reachability Analysis

The reachability analysis is to check if a given node of an SBS is reachable along a behavior of the SBS. Let \(G = (U, N, \text{succ}, \text{ref}, T)\) be an SBS. For a given node \(v \in N\), reachability analysis checks if there is a path \(\rho\) passing through \(v\) which is of the form \(v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_i \rightarrow \ldots \rightarrow v_m\) such that \(v_i = v\) \((0 \leq i \leq m)\) and that \(L(\rho) \neq \emptyset\). For example, for the ATM system given in Section 2.3, we check if the node SD \ DispenseCash is reachable in the specification depicted Figure 3.

Let \(G = (U, N, \text{succ}, \text{ref}, T)\) be an SBS, and \(\rho\) be a path in \(G\) of the form \(v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_m\) where \(\text{ref}(v_i) = (O_i, E_i, M_i, L_i, V_i, C_i)\) for any \(i\) \((0 \leq i \leq m)\). Since there could be \(v_i\) and \(v_j\) \((0 \leq i < j \leq m)\) such that \(\text{ref}(v_i) = \text{ref}(v_j)\), by renaming, let \(E_i \cap E_j = \emptyset\) for any \(i, j\) \((0 \leq i < j \leq m)\). Let \(E = E_1 \cup E_2 \cup \ldots \cup E_m = \{e_0, e_1, \ldots, e_n\}\), and \(t_i\) represent the occurrence time of \(e_i\) \((0 \leq i \leq n)\) in a timed event sequence in \(L(\rho)\). We can get a group of linear inequalities on \(t_1, t_2, \ldots, t_n\), denoted by \(lp(\rho)\), which is constructed as follows:

- for any \(t_i\) and \(t_j\) \((0 \leq i < j \leq n)\), if \(e_i \in E_k\) and \(e_j \in E_{k+1}\) \((0 \leq k < m)\), then \(t_i - t_j \leq 0\);

- \(t_0, t_1, \ldots, t_n\) must satisfy all the timing constraints in \(T\), and the corresponding linear inequalities are given according to Definition 5; and
• \( t_0, t_1, \ldots, t_n \) must satisfy all the timing constraints in each \( C_i \) \((0 \leq i \leq m)\), and the corresponding linear inequalities are given according to Definition 3.

Since \( L(\rho) \neq \emptyset \) if and only if \( lp(\rho) \) has a solution, we can reduce the reachability analysis problem for a node \( v \) of \( G \) into the linear programming problems as follows: check if there is a path \( \rho' \) in \( G \) passing through \( v \) such that \( lp(\rho') \neq \emptyset \). It is clear that in the worst case, we need to check all the paths in \( G \) which pass through \( v \). Since the number of paths of \( G \) could be infinite, and the length of a path of \( G \) could be infinite, we attempt to solve the problem based on a finite set of the finite paths of \( G \).

Let \( G = (U, N, succ, ref, T) \) be an SBS, and \( v \) be a node in \( N \). Let \( \Delta(G, v) \) be a set of the paths in \( G \) of the form \( v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_i \rightarrow v_{i+1} \rightarrow \ldots \rightarrow v_m \) where \( v_i = v \) \((0 \leq i \leq m)\), all \( v_j \) \((0 \leq j \leq i)\) are distinct, and all \( v_k \) \((i \leq k \leq m)\) are distinct. Intuitively, the node \( v \) divides each path in \( \Delta(G, v) \) into two simple path segments, which implies that \( \Delta(G, v) \) is finite and each path in \( \Delta(G, v) \) is finite because \( N \) is finite. A path segment \( \rho \) in \( G \) is a prefix for \( \Delta(G, v) \) if it may be extended into a path which is in \( \Delta(G, v) \), i.e. there could be a path segment \( \rho_1 \) in \( G \) such that \( \rho \rightarrow \rho_1 \) is in \( \Delta(G, v) \).

The following theorem tells us that if \( G \) is loop-unlimited, we just need to check each path in \( \Delta(G, v) \) for reachability.

**Theorem 1** Let \( G = (U, N, succ, ref, T) \) be a loop-unlimited SBS, and \( v \) be a node in \( N \). Then, \( v \) is reachable if and only if there is a path \( \rho \in \Delta(G, v) \) such that \( L(\rho) \neq \emptyset \).

The proof of this theorem is presented in the appendix. Based on the above theorem, we can develop an algorithm to check if a node \( v \) in an SBS \( G \) is reachable (cf. Figure 5). In the algorithm, first we check if \( G \) is loop-unlimited, and assign the result to the boolean variable \( loop\_unlimited \). Then, the algorithm traverses the state space of the nodes of \( G \) in a depth first manner starting from the start node \( \top \). The path in the state space that we have so far traversed is stored in the list variable \( currentpath \). For each successive node \( node \) of the last node of \( currentpath \), we first check whether the path segment \( \rho \) corresponding to the concatenation of \( currentpath \) and \( node \) is in \( \Delta(G, v) \). If yes, then we check if \( L(\rho) \neq \emptyset \) by linear programming, and return \( true \) when \( L(\rho) \neq \emptyset \). If the path segment corresponding to the concatenation of \( currentpath \) and \( node \) is a prefix for \( \Delta(G, v) \), then we add \( node \) to the current path and start the search from it, otherwise we search the other successive nodes. The algorithm backtracks when all the successive nodes of the last node of \( currentpath \) are explored. After finishing the depth first search, we return \( false \) when \( G \) is loop-unlimited, and \( undecided \) when \( G \) is not loop-unlimited. Notice that the algorithm can answer \( true \) for some SBSs which are not loop-unlimited, but not all. It is thus a decision
check if $G$ is loop-unlimited:
If yes then loop_unlimited := true else loop_unlimited := false;
$\text{currentpath} := \langle \top \rangle$;
repeat
    node := the last node of currentpath;
    if all successive nodes of node are explored through currentpath
    then /*backtracking*/ delete the last node of currentpath
    else begin /*explore an unexplored successive node through currentpath*/
        node := a successive node of node not explored through currentpath;
        if the path segment $\rho$ corresponding to the concatenation of currentpath and node is in $\Delta(G, v)$
        then begin check if $L(\rho) \neq \emptyset$;
            if yes then return true;
        end;
        if the path segment corresponding to the concatenation of currentpath and node is a prefix for $\Delta(G, v)$
        then append node to currentpath;
    end
until currentpath = $\langle \rangle$;
if loop_unlimited then return false else return undecided.

Figure 5: Algorithm for reachability analysis

procedure for the loop-unlimited SBSs, and a semi-decision procedure for the non loop-unlimited SBSs.

3.2 Constraint Conformance Analysis

The constraint conformance analysis is to check if the given several scenarios, which occur continuously in the behavior of an SBS, satisfy a given timing constraint.

Let $G = (U, N, \text{succ}, \text{ref}, T)$ be an SBS. A constraint conformance specification, denoted by $\mathcal{SC}(\varrho, \zeta)$, consists of a finite sequence $\varrho = D_0 \rightarrow D_1 \rightarrow \cdots \rightarrow D_k$ of sequence diagrams in $U$ and a timing constraint $\zeta$ of the form

$$a \leq c_0(f_0 - f'_0) + c_1(f_1 - f'_1) + \cdots + c_n(f_l - f'_l) \leq b$$

where the events $f_i, f'_i$ $(0 \leq i \leq l)$ occur in $D_0, D_1, \ldots, D_k$ exactly once. Let $\rho$ be a path in $G$ of the form $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_m$. We say that $\rho$ satisfies $\mathcal{SC}(\varrho, \zeta)$ if every occurrence of $\varrho$ in $\rho$ satisfies $\zeta$, i.e., for any $\sigma = \sigma_0 \rightarrow \sigma_1 \rightarrow \cdots \rightarrow \sigma_m \in L(\rho)$ where every $\sigma_i$ $(0 \leq i \leq m)$ is a behavior of $\text{ref}(v_i)$, for any subsequence of $\rho$ of the form $v_j \rightarrow v_{j+1} \rightarrow \cdots \rightarrow v_{j+k}$ $(0 \leq j \leq m - k)$ such that $\text{ref}(v_i) = D_{i-j}$ for any $i$ $(j \leq i \leq j + k)$, if $\sigma_j \rightarrow \sigma_{j+1} \rightarrow \cdots \rightarrow \sigma_{j+k} = (e_0, t_0) \rightarrow (e_1, t_1) \rightarrow \cdots \rightarrow (e_n, t_n)$ then

$$a \leq c_0 \delta_0 + c_1 \delta_1 + \cdots + c_n \delta_l \leq b$$

where for any $i$ $(0 \leq i \leq \ell)$, if $f_i = e_p$ and
\[ f'_i = e_q \ (0 \leq p, q \leq n) \] then
\[
\delta_i = \begin{cases} 
  t_{q+1} + t_{q+2} + \ldots + t_p & \text{if } p > q \\
  -(t_{p+1} + t_{p+2} + \ldots + t_q) & \text{if } p < q
\end{cases}
\]

We define that \( G \) satisfies \( S_C(\varrho, \zeta) \) if any path in \( G \) satisfies \( S_C(\varrho, \zeta) \). For example, for the ATM example given in the Section 2.3, since a customer may lose his patience after he gets the money, we require that the time that the ATM takes for the printing and book-keeping after giving the money is not greater than the half of the time that the customer waits for withdrawing the money, which forms a constraint conformance specification \( S_C(\varrho, \zeta) \) where \( \varrho = \text{Withdraw} \rightarrow \text{DispenseCash} \), and \( \zeta = 2(\varpi_j - j_3) \leq j_c - h_5 \).

Let \( G = (U, N, \text{succ}, \text{ref}, T) \) be an SBS, and \( S_C(\varrho, \zeta) \) be a constraint conformance specification where \( \varrho = D_0 \rightarrow D_1 \rightarrow \ldots \rightarrow D_k \) and \( \zeta \) is of the form \( a \leq \sum_{i=0}^{l} c_i (f_i - f_i') \leq b \). In the following, we show that for a finite path \( \rho \) in \( G \), the constraint conformance analysis for \( S_C(\varrho, \zeta) \) can be solved by linear programming. Suppose that \( \rho = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_m \) and \( \mathcal{L}(\rho) \neq \emptyset \). Since there could be \( v_i \) and \( v_j \) \((0 < i < j \leq m)\) such that \( \text{ref}(v_i) = \text{ref}(v_j) \), by renaming, let \( E_i \cap E_j = \emptyset \) for any \( i, j \) \((0 < i < j \leq m)\). Let
\[
E = E_1 \cup E_2 \cup \ldots \cup E_m = \{e_0, e_1, \ldots, e_n\},
\]
and \( t_i \) represent the occurrence time of \( e_i \) \((0 \leq i \leq n)\) in a timed event sequence in \( \mathcal{L}(\rho) \). Suppose that there is a subsequence \( \rho_1 \) of \( \rho \) of the form \( v_j \rightarrow v_{j+1} \rightarrow \ldots \rightarrow v_{j+k} \) \((0 \leq j \leq m - k)\) such that \( \text{ref}(v_i) = D_{i-j} \) for any \( i \) \((j \leq i \leq j + k)\). Then, the satisfaction problem of \( \rho \) for \( S_C(\varrho, \zeta) \) can be reduced to the linear programs such as: finding the maximum (minimum) value of the linear function \( \sum_{i=0}^{l} c_i \delta_i \) subject to the linear constraint \( lp(\rho) \) where \( \delta_i = t_p - t_q \land f_i = e_p \land f_i' = e_q \) \((e_p \text{ and } e_q \text{ occurs in } \rho_1)\) for any \( i \) \((0 \leq i \leq l)\), and checking whether it is not greater than \( b \) \((\text{smaller than } a)\).

For an SBS, we need to check all its paths for a given constraint conformance specification, and attempt to solve the problem based on a finite set of finite paths. Let \( G = (U, N, \text{succ}, \text{ref}, T) \) be an SBS, and \( S_C(\varrho, \zeta) \) be a constraint conformance specification where \( \varrho = D_0 \rightarrow D_1 \rightarrow \ldots \rightarrow D_k \). Let \( \Delta(G, S_C(\varrho, \zeta)) \) be a set of the paths in \( G \) of the form
\[
v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_i \rightarrow u_0 \rightarrow u_1 \rightarrow \ldots \rightarrow u_k \rightarrow v_{i+1} \rightarrow v_{i+2} \rightarrow \ldots \rightarrow v_m
\]
where \( \text{ref}(v_i) = D_i \) for any \( i \) \((0 \leq i \leq k)\), all \( v_j \) \((0 \leq j \leq i)\) and \( u_0 \) are distinct, and all \( v_j \) \((i + 1 \leq j \leq m)\) and \( u_k \) are distinct. Intuitively, a path in \( \Delta(S, S_C(\varrho, \zeta)) \) consists of three parts in succession: the first and last parts are a simple path segment, and the middle part is corresponding to \( \varrho \). A path segment \( \rho \) in \( G \) is a prefix for \( \Delta(G, S_C(\varrho, \zeta)) \) if it may be extended into a path which is in \( \Delta(G, S_C(\varrho, \zeta)) \), i.e. there could be a path segment \( \rho_1 \) in
such that \( \rho \rightarrow \rho_1 \) is in \( \Delta(G, \mathcal{S}_C(\varrho, \zeta)) \). The following theorem indicates that if \( G \) is loop-unlimited, we just need to check each path in \( \Delta(G, \mathcal{S}_C(\varrho, \zeta)) \) when checking if \( G \) satisfies \( \mathcal{S}_C(\varrho, \zeta) \).

**Theorem 2** Let \( G \) be a loop-unlimited SBS, and \( \mathcal{S}_C(\varrho, \zeta) \) be a constraint conformance specification. Then, \( G \) satisfies \( \mathcal{S}_C(\varrho, \zeta) \) if and only if any path in \( \Delta(G, \mathcal{S}_C(\varrho, \zeta)) \) satisfies \( \mathcal{S}_C(\varrho, \zeta) \).

The proof of this theorem is presented in the appendix. Based on the above theorem, we can develop an algorithm to check if an SBS \( G \) satisfies a constraint conformance specification \( \mathcal{S}_C(\varrho, \zeta) \) (cf. Figure 6). In the algorithm, first we check if \( G \) is loop-unlimited, and assign the result to the boolean variable \( \text{loop\_unlimited} \). Then, the algorithm traverses the state space of the nodes of \( G \) in a depth first manner starting from the start node \( \top \). The path in the state space that we have so far traversed is stored in the list variable \( \text{currentpath} \). For each successive node \( \text{node} \) of the last node of \( \text{currentpath} \), we first check whether the path segment \( \rho \) corresponding to the concatenation of \( \text{currentpath} \) and \( \text{node} \) is in \( \Delta(G, \mathcal{S}_C(\varrho, \zeta)) \). If yes, then we check if \( \rho \) satisfies \( \mathcal{S}_C(\varrho, \zeta) \) by linear programming, and return \( \text{false} \) when \( \mathcal{S}_C(\varrho, \zeta) \) is not satisfied. Then we check if the path segment corresponding to the concatenation of \( \text{currentpath} \) and \( \text{node} \) is a prefix for \( \Delta(G, \mathcal{S}_C(\varrho, \zeta)) \). If yes, then we add \( \text{node} \) to the current path and start the search from it, otherwise we search the other successive nodes. The algorithm backtracks when all the successive nodes of the last node of \( \text{currentpath} \) are explored. After
finishing the depth first search, we return true when $G$ is loop-unlimited, and undecided when $G$ is not loop-unlimited. Notice that the algorithm can answer false for some SBSs which are not loop-unlimited, but not all. It is thus a decision procedure for the loop-unlimited SBSs, and a semi-decision procedure for the non loop-unlimited SBSs.

3.3 Bounded Delay Analysis

The bounded delay analysis is to check if the separation in time between the two given events in any behavior of an SBS is not smaller or greater than a given real number, which is called the minimal bounded delay analysis or the maximal bounded delay analysis respectively. It is clear that for an SBS, if the two given events are in the same node, then the problems can be reduced into a constraint conformance analysis problem. So in the following we just consider the problems in which the two given events are in the different nodes of an SBS.

For an SBS $G$, a minimal (or maximal) bounded delay specification consists of two events $e, e'$ and a real number $d$, denoted by $S_B^m(e, e', d)$ (or $S_B^M(e, e', d)$), which requires that the separation in time between $e$ and $e'$ in any behavior of $G$ is not smaller (or greater) than $d$. For example, for the ATM example given in Section 2.3, since for the security consideration it is necessary to record the process for withdrawing money by the camera embedded in the ATM, we require every process for withdrawing money take the time which is long enough for recording, which forms a minimal bounded delay specification $S_B^m(j_4, a_1, 20)$.

Let $G = (U, N, succ, ref, T)$ be an SBS, $S_B^m(e, e', d)$ (or $S_B^M(e, e', d)$) be a bounded delay specification, and $σ$ be a behavior of $G$ of the form

$$(e_0, t_0) \rightarrow (e_1, t_1) \rightarrow \ldots \rightarrow (e_i, t_i) \rightarrow \ldots \rightarrow (e_j, t_j) \rightarrow \ldots \rightarrow (e_n, t_n).$$

If for any $i$ and $j$ ($0 \leq i < j \leq n$) such that $e_i = e'$, $e_j = e$, and that $e_k \neq e \land e_k \neq e'$ for any $k$ ($i < k < j$), $t_{i+1} + t_{i+2} + \ldots + t_j \geq \lfloor i\rfloor d$, then we say that $σ$ satisfies $S_B^m(e, e', d)$ (or $S_B^M(e, e', d)$). We define that a path $ρ$ in $G$ satisfies $S_B^m(e, e', d)$ (or $S_B^M(e, e', d)$) if any $σ \in L(ρ)$ satisfies $S_B^m(e, e', d)$ (or $S_B^M(e, e', d)$), and that $G$ satisfies $S_B^m(e, e', d)$ (or $S_B^M(e, e', d)$) if any path in $G$ satisfies $S_B^m(e, e', d)$ (or $S_B^M(e, e', d)$).

Let $G = (U, N, succ, ref, T)$ be an SBS, and $S_B^m(e, e', d)$ (or $S_B^M(e, e', d)$) be a bounded delay specification. In the following we show that for a finite path $ρ$ in $G$, the satisfaction problem of $ρ$ for $S_B^m(e, e', d)$ (or $S_B^M(e, e', d)$) can be reduced into linear programming problems. Suppose that

$$ρ = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_i \rightarrow v_{i+1} \rightarrow \ldots \rightarrow v_j \rightarrow v_{j+1} \rightarrow \ldots \rightarrow v_m \quad (L(ρ) \neq \emptyset)$$

where $e'$ occurs in $ref(v_i)$, $e$ occurs in $ref(v_j)$ ($0 \leq i < j \leq m$), and $e, e'$ do not occur in any $ref(v_k)$ ($i < k < j$). Since there could be $v_k$ and
Let \( S \) be a bounded delay specification. Let \( \Delta(G, e, e) \) be the set of the paths in \( G \). Intuitively, each path in \( \Delta(G, e, e) \) is separated by \( e \) and \( e \) into three simple path segments in succession. For an SBS \( G \), for a bounded delay specification \( \mathcal{S}_B(e, e', d) \) (or \( \mathcal{S}_B^M(e, e', d) \)), a path segment \( \rho \) in \( G \) is a prefix for \( \Delta(G, \mathcal{S}_B(e, e', d)) \) (or \( \Delta(G, \mathcal{S}_B^M(e, e', d)) \)) if it may be extended into a path which is in \( \Delta(G, \mathcal{S}_B(e, e', d)) \) (or \( \Delta(G, \mathcal{S}_B^M(e, e', d)) \)), i.e., there could be a path segment \( \rho_1 \) in \( G \) such that \( \rho \rightarrow \rho_1 \) is in \( \Delta(G, \mathcal{S}_B(e, e', d)) \) (or \( \Delta(G, \mathcal{S}_B^M(e, e', d)) \)).

For an SBS \( G \) which is loop-limited, the minimal bounded delay analysis problem, checking \( G \) for a bounded delay specification \( \mathcal{S}_B^m(e, e', d) \), can be solved by checking each path \( \Delta(G, \mathcal{S}_B^m(e, e', d)) \), which is supported by the following theorem.

**Theorem 3** Let \( G \) be a loop-unlimited SBS, and \( \mathcal{S}_B^m(e, e', d) \) be a minimal bounded delay specification. Then, \( G \) satisfies \( \mathcal{S}_B^m(e, e', d) \) if and only if any path in \( \Delta(G, \mathcal{S}_B^m(e, e', d)) \) satisfies \( \mathcal{S}_B^m(e, e', d) \). \( \square \)

The proof of this theorem is presented in the appendix. Based on Theorem 3, we can develop an algorithm to check if an SBS \( G \) satisfies a minimal bounded delay specification \( \mathcal{S}_B^m(e, e', d) \). (cf. Figure 7). The structure of
check if $G$ is loop-unlimited;
If yes then \texttt{loop\_unlimited := true} else \texttt{loop\_unlimited := false};
currentpath := $\langle \top \rangle$;
\textbf{repeat}
  node := the last node of currentpath;
  if all successive nodes of node are explored through currentpath
    then /*backtracking*/ delete the last node of currentpath
  else begin /*explore an unexplored successive node through currentpath*/
    node := a successive node of node not explored through currentpath;
    if the path segment $\rho$ corresponding to the concatenation of currentpath and node is in $\Delta(G, S_B^m(e, e', d))$
      then begin check if $\rho$ satisfies $S_B^m(e, e', d)$;
        if no then return false;
      end
    if the path segment corresponding to the concatenation of currentpath and node is a prefix for $\Delta(G, S_B^m(e, e', d))$
      then append node to currentpath;
  end
\textbf{until} currentpath = $\langle \rangle$;
if \texttt{loop\_unlimited} then return true else return undecided.

Figure 7: Algorithm for minimal bounded delay analysis

the algorithm is the same as the algorithm depicted in Figure 6. Since the algorithm can answer false for some SBSs which are not loop-unlimited, but not all, it is thus a decision procedure for the loop-unlimited SBSs, and a semi-decision procedure for the non loop-unlimited SBSs.

For the maximal bounded delay analysis, we need to give more consideration because it is a little more complicated than the minimal bounded delay analysis. First, we need to introduce the violable nodes for the given two events in an SBS. We say that a loop $\rho$ in an SBS $G$ is positive if a repetition of the loop may take time, i.e., there is $(e_0, t_0) \rightarrow (e_1, t_1) \rightarrow \ldots \rightarrow (e_n, t_n) \in \mathcal{L}(\rho)$ such that $t_0 + t_1 + \ldots + t_n > 0$ (notice that we can check if a loop is positive by linear programming). We say that a loop $\rho$ ($\mathcal{L}(\rho) \neq \emptyset$) in an SBS $G$ is free of an event $e$ if $e$ does not occur in any node in $\rho$. For an SBS $G$, a node $v$ in $G$, and for two events $e, e'$ in $G$, we define the set $\Theta(G, v, e, e')$ recursively as follows:

- any loop $\rho$ ($\mathcal{L}(\rho) \neq \emptyset$) in $G$ free of $e$ and $e'$ whose loop-start node is $v$ belongs to $\Theta(G, v, e, e')$;
- for any loop in $\Theta(G, v, e, e')$ of the form $v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_m$, any loop $\rho$ ($\mathcal{L}(\rho) \neq \emptyset$) in $G$ free of $e$ and $e'$ whose loop-start node is $v_i$ ($0 \leq i \leq m$) belongs to $\Theta(G, v, e, e')$.

For a node $v$ in an SBS $G$ and two events $e$ and $e'$, if there is a positive loop in $\Theta(G, v, e, e')$, then we say that $v$ is a violable node for $e$ and $e'$. 
Then, for an SBS $G$, for a maximal bounded delay specification $S_M^M(e, e', d)$, we introduce **insulating segments** in the paths in $\Delta(G, S_M^M(e, e', d))$. For any $\rho \in \Delta(G, S_M^M(e, e', d))$ of the form

$$v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_i \rightarrow v_{i+1} \rightarrow \ldots \rightarrow v_j \rightarrow v_{j+1} \rightarrow \ldots \rightarrow v_m$$

where $e'$ occurs in $\text{ref}(v_i)$, $e$ occurs in $\text{ref}(v_j)$ ($0 \leq i < j \leq m$), and $e, e'$ do not occur in any $\text{ref}(v_k)$ ($i < k < j$), we call the subsequence between $e'$ and $e$ of the form $v_{i+1} \rightarrow v_{i+2} \rightarrow \ldots \rightarrow v_{j-1}$ an **insulating segment**. Notice that for a path $\rho \in \Delta(G, S_M^M(e, e', d))$, if there is a node $v$ in its insulating segment which is violable for $e$ and $e'$, then we can construct a behavior of $G$ which does not satisfy $S_B^M(e, e', d)$ by repeating the positive loop in $\Theta(G, v, e, e')$ with finite times, which results in the following theorem and is depicted in Figure 8.

**Theorem 4** Let $G$ be a loop-unlimited SBS, and $S_B^M(e, e', d)$ be a maximal bounded delay specification. Then, $G$ satisfies $S_B^M(e, e', d)$ if and only if for any path $\rho \in \Delta(G, S_B^M(e, e', d))$ ($\mathcal{L}(\rho) \neq \emptyset$), it satisfies $S_B^M(e, e', d)$ and there is not any node in its insulating segments which is violable for $e$ and $e'$. □

The proof of this theorem is presented in the appendix. Based on Theorem 4, we can develop an algorithm to check if an SBS $G$ satisfies a maximal bounded delay specification $S_B^M(e, e', d)$ (cf. Figure 9). The structure of the algorithm is the same as the algorithm depicted in Figure 7. The difference from the minimal bounded delay analysis algorithm is that after a path in $\Delta(G, S_B^M(e, e', d))$ is discovered, we not only check if it satisfies $S_B^M(e, e', d)$, but also check if there is no violable node for $e$ and $e'$ in its insulating segments when $G$ is loop-unlimited. Since the algorithm can answer **false** for some SBSs which are not loop-unlimited, but not all, it is thus a decision procedure for the loop-unlimited SBSs, and a semi-decision procedure for the non loop-unlimited SBSs.
check if $G$ is loop-unlimited:
If yes then $\text{loop\_unlimited} := \text{true}$ else $\text{loop\_unlimited} := \text{false}$;
$\text{currentpath} := \langle \top \rangle$;
repeat
node := the last node of $\text{currentpath}$;
if all successive nodes of node are explored through $\text{currentpath}$
then /*backtracking*/
delete the last node of $\text{currentpath}$
else begin
/*explore an unexplored successive node through $\text{currentpath}$*/
node := a successive node of node not explored through $\text{currentpath}$;
if the path segment $\rho$ corresponding to the concatenation of $\text{currentpath}$ and node is in $\Delta(G, S_B^M(e, e', d))$
then begin
check if $\rho$ satisfies $S_B^M(e, e', d)$;
if no then return false;
if $\text{loop\_unlimited}$ then begin
check if there is a violable node for $e$ and $e'$ in the insulating segments;
if yes then return false;
end
end;
if the path segment corresponding to the concatenation of $\text{currentpath}$ and node is a prefix for $\Delta(G, S_B^M(e, e', d))$
then append node to $\text{currentpath}$;
end
until $\text{currentpath} = \langle \rangle$;
if $\text{loop\_unlimited}$ then return true else return undecided.

Figure 9: Algorithm for maximal bounded delay analysis

3.4 Complexity of Algorithms

The complexity of the algorithms presented in this section consists of two parts: one is from the searching the node state space of an SBS $G$, and the other includes the number and size of the linear programs we need to solve in the algorithms.

Let $G = (U, N, \text{succ, ref, T})$ be an SBS. The numbers of the prefixes for the path sets $\Delta(G, v)$, $\Delta(G, S_C(g, \zeta))$, $\Delta(G, S_B^m(e, e', d))$, and $\Delta(G, S_B^M(e, e', d))$ are not greater than $|N|!$, $|N|!^3$, $|N|!^3$, and $|N|!^3$ respectively, and the sizes of the longest prefixes for the path sets $\Delta(G, v)$, $\Delta(G, S_C(g, \zeta))$, $\Delta(G, S_B^m(e, e', d))$, and $\Delta(G, S_B^M(e, e', d))$ are not greater than $|N|$, $3|N|$, $3|N|$, and $3|N|$ respectively.

For the node state space search, the complexity of the algorithms is proportional to the number of the prefixes for the path sets $\Delta(G, v)$, $\Delta(G, S_C(g, \zeta))$, $\Delta(G, S_B^m(e, e', d))$, or $\Delta(G, S_B^M(e, e', d))$, and to the size of the longest prefix.

Since in the algorithms, for each path in the the set $\Delta(G, v)$, $\Delta(G, S_C(g, \zeta))$, $\Delta(G, S_B^m(e, e', d))$, or $\Delta(G, S_B^M(e, e', d))$, we need to solve at most one linear program, the number of linear programs we need to solve is proportional to the number of the paths in the set $\Delta(G, v)$, $\Delta(G, S_C(g, \zeta))$, $\Delta(G, S_B^m(e, e', d))$, or $\Delta(G, S_B^M(e, e', d))$. Since one event is corresponding to one variable in the
linear programs, the size of the linear programs we need to solve in the algorithms is proportional to the maximal number of the events occurring in a path in the sets, and to the maximal number of the timing constraints in a path in the sets.

The algorithms we have presented are based on linear programming. The linear programming problem has been well-studied, and can be solved with a polynomial-time algorithm in general. Thanks to the advances in computing of the past decade, linear programs in a few thousand variables and constraints are nowadays viewed as ”small”. Problems having tens or hundreds of thousands of continuous variables are regularly solved. Indeed many software packages have been developed to efficiently find solutions for linear programs. We thus think the approach presented in this paper is efficient for the problems in practical use.

3.5 Analysis Tool Prototype

We have implemented the solutions presented above in a tool prototype TASS [22] for timing analysis of SBSs. TASS can be used to check SBSs for reachability, constraint conformance specifications, and for bounded delay specifications.

TASS is implemented in Java, and reads the UML interaction models produced in Rational Rose as its inputs. The linear programming software package which is integrated in the tool is from OR-Objects of DRA Systems [23] which is a free collection of Java classes for developing operations research, scientific and engineering applications.

On a Pentium M/1.50GHz/512MB PC, TASS runs comfortably for checking several SBSs with more than 30 SDs in a few seconds. The practical examples in our case studies include the automatic teller machine (ATM) system in [7] and the global system for mobile communication (GSM) in [24], and their details are given in [22].

For timing analysis of the ATM specification depicted in Figure 3, TASS is used to solve the following problems:

- Reachability analysis: we check if the node SD DispenseCash is reachable in the specification.

- Constraint conformance analysis: since a customer may lose his patience after he gets the money, we require that the time that the ATM takes for the printing and book-keeping after giving the money is not greater than the half of the time that the customer waits for withdrawing the money \((2(\varpi_j - J_8) \leq j_c - h_5)\), which forms a constraint conformance specification \(S_C(\varrho, \zeta)\) where \(\varrho = \text{Withdraw} \rightarrow \text{DispenseCash}\), and \(\zeta = 2(\varpi_j - J_8) \leq j_c - h_5\).
• Bound delay analysis: since for the security consideration it is necessary to record the process for withdrawing money by the camera embedded in the ATM, we require every process for withdrawing money take enough time for recording, which forms a minimal bounded delay specification $S_B^m(j, a_1, 20)$.

Given the various values of $W, B_1, B_2, T_1,$ and $T_2$, the tool reports the corresponding sample results, which is depicted in Table 1.

<table>
<thead>
<tr>
<th>Case</th>
<th>Problem</th>
<th>Reachability</th>
<th>$S_{C}(\varrho, \zeta)$</th>
<th>$S_{B}^m(j, a_1, 20)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w = 4, B_1 = 0, B_2 = \infty, T_1 = 0.5, T_2 = 2$</td>
<td>no</td>
<td>3.226s</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$w = 6, B_1 = 0.5, B_2 = 1, T_1 = 0.5, T_2 = 2$</td>
<td>yes</td>
<td>6ms</td>
<td>yes</td>
<td>3.760s</td>
</tr>
<tr>
<td>$w = 9, B_1 = 2, B_2 = 3, T_1 = 1, T_2 = 3$</td>
<td>yes</td>
<td>47ms</td>
<td>yes</td>
<td>3.063s</td>
</tr>
</tbody>
</table>

*: The verification problem disappears because the node is unreachable.

Table 1: Sample results of timing analysis of the ATM specification

3.6 Discussion on Timing Analysis of SBSs with Asynchronous Composition Semantics

The solution to timing analysis of SBSs we have given is based on the synchronous composition semantics of SBSs. Here we discuss the solution for SBSs with the asynchronous composition semantics.

The asynchronous composition semantics of SBSs corresponds to concatenating two SDs object by object, i.e. the asynchronous concatenation of two SDs gives another SD, which is the same as the asynchronous interpretation of the concatenation of two basic MSCs in MSC specifications [10]. Let $D_1 = (O_1, E_1, M_1, L_1, V_1, C_1)$ and $D_2 = (O_2, E_2, M_2, L_2, V_2, C_2)$ be two SDs. The asynchronous concatenation of $D_1$ and $D_2$ is the SD $D = (O, E, M, L, V, C')$ defined by

- $O = O_1 \cup O_2$.
- $E = E_1 \cup E_2$ (Assuming the two event sets $E_1$ and $E_2$ are disjoint).
- $M = M_1 \cup M_2$.
- For $e \in E_1$, $L(e) = L_1(e)$, and for $e \in E_2$, $L(e) = L_2(e)$.
- $V = V_1 \cup V_2 \cup V_3 \cup V_4$ where
  - $V_3 = \{(e_1, e_2) \mid e_1 \in E_1, e_2 \in E_2, L(e_1) = L(e_2), e_2$ is a sending event\}$
  - $V_4 = \{(e_1, e_2) \mid e_1, e_2$ are receiving events whose corresponding sending events appear on mutual object column.\}$
\[ C = C_1 \cup C_2. \]

According to the above definition, every path in an SBS corresponds to a SD, and thus the behavior of an SBS is interpreted by the behavior of SDs. It follows that for a finite path in an SBS, the reachability analysis, constraint conformance analysis, and bounded delay analysis problems can be solved by linear programming. However, for an infinite path in an SBS, the linear programming based solution is inapplicable for the timing analysis problems because the asynchronous composition semantics may lead all SDs in the path are dependent of each other (in this case we cannot solve the problems based on a finite path).

Notice that the linear programming based solution we have presented can be used for bounded timing analysis of SBSs which is corresponding to bound model checking whose basic idea is to search for a counterexample in the model executions whose length is bounded by some integer \( k \) [25]. Since we can solve the timing analysis problems for a finite path in an SBS, we can traverse the structure of the SBS directly in a depth first manner and check all the potential paths one by one whose lengths are constrained by a threshold and thereby, increase the faith in the correctness of the system. This bounded timing analysis function has been supported in our tool prototype TASS [22].

4 Related Work and Conclusion

In this paper, we present a linear programming based approach to timing analysis of scenario-based specifications expressed by UML interaction models. We introduce more general and expressive timing constraints in UML sequence diagrams, and develop algorithms for the reachability, constraint conformance, and bounded delay analysis of scenario-based specifications. These algorithms form a decision procedure for the loop-unlimited scenario-based specifications where no timing constraint is enforced on the repetition of any loop and one unfolding of any loop is time independent of the other unfolding, and a semi-decision procedure for the general scenario-based specifications.

To our knowledge, all the literature on timing analysis of scenario-based specifications are just about timing consistency. In [6], the problem of checking basic MSCs with delay intervals for timing consistency is reduced to computing negative cost cycles and shortest distances in a weighted directed graph by using temporal constraint network techniques. In [8], the same techniques is used for timing consistency analysis of a class of UML sequence diagrams in which all timing constraints are of the form \( a \leq e_1 - e_2 \leq b \). The problem of checking MSC specifications for timing consistency is considered in [7], which means that every execution scenario described by an MSC specification is timing consistent. But in that work just a sufficient condition
for timing consistency is given, which is not enough to develop an algorithm to analyze MSC specifications for timing consistency. A complete solution is given in [16] for checking the compositions of UML sequence diagrams for timing consistency. The problem of checking compositions of UML sequence diagrams for timing inconsistency is considered in [17], which means that no execution scenario described by a composition of UML sequence diagrams is timing consistent.

In [26], MSC specifications are interpreted as the global state automata. Theoretically the problems considered in this paper can thus be solved by transforming the scenario-based specifications into the timed automata [27], and then checking the timed automata for the corresponding properties. In that approach, in addition to the high complexity of checking timed automata themselves, the transformation will generate the state space altogether, which introduces considerable complexity. For example, the bounded delay analysis is considered in [19] for timed automata, but the algorithm complexity is very high, and to our knowledge no tool is implemented so far. A case study is investigated in [9] for UML sequence diagram based verification in which a simple sequence diagram is transformed to a set of timed automata and then checked for a timed automata based implementation by a model checking [28] tool for timed automata. In that case study, it is shown that a lot of clocks are introduced for describing timing constraints enforced on sequence diagrams, which results in an exponential increment on the complexity of the algorithms checking timed automata.

Compared to the timed automata based approach, the advantage of our approach includes two aspects. On the one hand, our approach analyzes directly scenario-based specifications themselves by investigating only the node state spaces of scenario-based specifications which are much smaller than the corresponding timed state spaces of timed automata, and reduces the timing analysis problems into linear programming problems which can be solved with efficient algorithms, so that we avoid the generation of the state space altogether and also the involved complexity. On the other hand, the timing constraints considered in our approach are more general and expressive, which can be used to describe the relations among multiple separations in time between events. We know that for a clock constraint in a timed automaton, its corresponding timing constraint is just related to the separation in time between two events. For describing timing constraints about the relations among multiple separations in time between events, we need to compare multiple clocks in a timed automaton, which will result in that the corresponding model checking problems are undecidable [27].
References


A Checking if an SBS is Loop-Unlimited

Let $G = (U, N, succ, ref, T)$ be a SBS. According to the definition of loop-unlimited condition, for checking if the loop-unlimited condition are held for $G$, we just need to traverse all the path segments in $G$ of the form

$$v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_{i-1} \rightarrow v_i \rightarrow v_{i+1} \rightarrow \ldots \rightarrow v_n$$

where $(\top, v_0) \in succ$, all $v_j$ ($0 \leq j \leq i$) are distinct, all $v_k$ ($i < k \leq n$) are distinct, $a \leq e - e' \leq b \in T$, $e'$ occurs in $ref(v_i)$, $e$ occurs in $ref(v_n)$, and $e, e'$ do not occur in any $v_l$ ($i < l < n$), which are called checked path segments for loop-unlimited condition. According to the definition of loop-closed condition, for checking if the loop-closed condition are held for $G$, we just need to traverse all the path segments in $G$ of the form

$$v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_{i-1} \rightarrow v_i \rightarrow v_{i+1} \rightarrow \ldots \rightarrow v_n$$

where $(\top, v_0) \in succ$, $v_i$ ($1 \leq i \leq n$) is a loop-start node, all $v_j$ ($0 \leq j \leq i$) are distinct, and all $v_k$ ($i < k \leq n$) are distinct, which are called checked path segments for loop-closed condition. A path segment $\rho$ is a prefix for loop-unlimited condition (loop-closed condition) if it may be extended into a checked path segment for loop-unlimited condition (loop-closed condition), i.e., there could be a path segment $\rho_1$ such that $\rho \rightarrow \rho_1$ becomes a checked path segment for loop-unlimited condition (loop-closed condition).

Here we present an algorithm to check if an SBS $G$ is loop-unlimited and to remove all the cases violating the loop-closed condition when $G$ is loop-unlimited (cf. Figure 10). The algorithm is based on depth-first search method. The main data structure in the algorithm includes a list $currentpath$ of nodes which is used to record the current paths, and a set $loopset$ of loops which records all the loops in $G$. The algorithm consists of three main steps. First, by a depth-first search we find out all loops in $G$, and check if there is not any reverse constraint for any loop in $G$. Then we traverse all the checked path segments for loop-unlimited condition in $G$ to check if $G$ is loop-unlimited. Last, we traverse all the checked path segments for loop-closed condition in $G$ to check if the loop-closed condition is held and to remove all the cases violating the loop-closed condition by changing the structure of $G$. The complexity of the algorithm is proportional to the number of the prefixes for loop-closed condition (loop-unlimited condition), and to the size of the longest prefix for loop-closed condition (loop-unlimited condition) in $G$.

B Proofs of Theorems

**Theorem 1** Let $G = (U, N, succ, ref, T)$ be a loop-unlimited SBS, and $v$ is a node in $N$. Then, $v$ is reachable if and only if there is a path $\rho \in \Delta(G, v)$ such that $L(\rho) \neq \emptyset$. 

29
figure 10
Proof. It is clear that one half of the claim holds: if there is \( \rho \in \Delta(G, v) \) such that \( \mathcal{L}(\rho) \neq \emptyset \), then \( v \) is reachable. The other half of the claim can be proved as follows. Suppose that \( v \) is reachable. Then there is a path \( \rho \) of the form \( v_0 \to v_1 \to \ldots \to v_i \to \ldots \to v_m \) \((0 \leq i \leq m)\) such that \( v_i = v \), and a timed event sequence \( \sigma \in \mathcal{L}(\rho) \) of the form \( \sigma_0 \to \sigma_1 \to \ldots \to \sigma_i \to \ldots \to \sigma_m \) where each \( \sigma_j \) \((0 \leq j \leq m)\) \(\in \mathcal{L}(\text{ref}(v_j))\). In the following we prove that there is a path \( \rho' \in \Delta(G, v) \) such that \( \mathcal{L}(\rho') \neq \emptyset \), which results in the claim holds. If all \( v_j \) \((0 \leq l \leq i)\) are distinct and all \( v_k \) \((i \leq k \leq m)\) are distinct, then \( \rho' \in \Delta(G, v) \) and we are done. Otherwise, there are \( v_p \) and \( v_q \) \((0 \leq p < q \leq i)\) such that \( v_p \to v_{p+1} \to \ldots \to v_q \) is loop \((v_p = v_q)\), and (or) there are \( v_p' \) and \( v_{q'} \) \((i \leq p' < q' \leq m)\) such that \( v_{p'} \to v_{p'+1} \to \ldots \to v_{q'} \) is loop \((v_{p'} = v_{q'})\). Since \( G \) satisfies the loop-closed condition and the loop-unlimited condition, any timing constraint does not combine any two nodes which are inside and outside a loop respectively, and is not enforced on the repetition of any loop. It follows that by removing the subsequences \( \sigma_p \to \sigma_{p+1} \to \ldots \to \sigma_{q-1} \) and (or) \( \sigma_{p'+1} \to \sigma_{p'+2} \to \ldots \to \sigma_{q'} \) from \( \mathcal{L}(\rho) \), we can get a timed event sequence \( \sigma_R \) which is a behavior of \( G \), and by removing the subsequences \( v_p \to v_{p+1} \to \ldots \to v_{q-1} \) and \( v_{p'+1} \to v_{p'+2} \to \ldots \to v_{q'} \) from \( \rho \), we can get a path \( \rho_R \) such that \( \sigma_R \in \mathcal{L}(\rho_R) \). By applying the above step repeatedly, we can get a path \( \rho' \) of the form \( v'_1 \to v'_2 \to \ldots \to v'_j \to v_i \to v'_{j+1} \to \ldots \to v'_k \) such that \( \mathcal{L}(\rho') \neq \emptyset \), all \( v'_l \) \((0 \leq l \leq j)\) and \( v_i \) are distinct, and that all \( v'_l \) \((j < l \leq k)\) and \( v_i \) are distinct. It follows that \( \rho' \) is in \( \Delta(G, v) \), from which the claim holds. \( \square \)

Theorem 2 Let \( G \) be a loop-unlimited SBS, and \( \mathcal{S}_C(\rho, \zeta) \) be a constraint conformance specification. Then, \( G \) satisfies \( \mathcal{S}_C(\rho, \zeta) \) if and only if any path in \( \Delta(G, \mathcal{S}_C(\rho, \zeta)) \) satisfies \( \mathcal{S}_C(\rho, \zeta) \).

Proof. It is clear that one half of the claim holds: if \( G \) satisfies \( \mathcal{S}_C(\rho, \zeta) \), then any path in \( \Delta(G, \mathcal{S}_C(\rho, \zeta)) \) satisfies \( \mathcal{S}_C(\rho, \zeta) \). The other half of the claim can be proved as follows. Suppose that any path in \( \Delta(G, \mathcal{S}_C(\rho, \zeta)) \) satisfies \( \mathcal{S}_C(\rho, \zeta) \), and that there is a path \( \rho \) in \( G \) such that \( \sigma \in \mathcal{L}(\rho) \) does not satisfy \( \mathcal{S}_C(\rho, \zeta) \). Let \( g = D_0 \to D_1 \to \ldots \to D_k \). Without losing generality, suppose that \( \rho \) is of the form

\[
v_0 \to v_1 \to \ldots \to v_{i-1} \to v_i \to v_{i+1} \to \ldots \to v_{i+k} \to v_{i+k+1} \to \ldots \to v_m
\]

where \( \text{ref}(v_{i+j}) = D_j \) for any \( j \) \((0 \leq j \leq k)\), and \( \sigma \) is of the form

\[
\sigma_0 \to \sigma_1 \to \ldots \to \sigma_{i-1} \to \sigma_i \to \sigma_{i+1} \to \ldots \to \sigma_{i+k} \to \sigma_{i+k+1} \to \ldots \to \sigma_m
\]

where \( \sigma_j \in \mathcal{L}(\text{ref}(v_j)) \) for any \( j \) \((0 \leq j \leq m)\), and \( \zeta \) is not satisfied by \( \sigma_i \to \sigma_{i+1} \to \ldots \to \sigma_{i+k} \). In the following, we prove that we can construct a path \( \rho' \in \Delta(G, \mathcal{S}_C(\rho, \zeta)) \) such that there is a timed event sequence \( \sigma' \in \mathcal{L}(\rho') \) which contains \( \sigma_i \to \sigma_{i+1} \to \ldots \to \sigma_{i+k} \), which results in a contradiction and implies that the claim holds. Since \( \rho \not\in \Delta(S, \mathcal{S}_C(\rho, \zeta)) \),
there are \( v_p \) and \( v_q \) (\( 0 \leq p < q \leq i \)) such that \( v_p \rightarrow v_{p+1} \rightarrow \ldots \rightarrow v_q \) is loop \( (v_p = v_q) \), and (or)

there are \( v_{p'} \) and \( v_{q'} \) (\( i + k \leq p' < q' \leq m \)) such that \( v_{p'} \rightarrow v_{p'+1} \rightarrow \ldots \rightarrow v_{q'} \) is loop \( (v_{p'} = v_{q'}) \).

Since \( G \) satisfies the loop-closed condition and the loop-unlimited condition, any timing constraint does not combine any two nodes which are inside and outside a loop respectively, and is not enforced on the repetition of any loop. It follows that by removing the subsequences \( \sigma_{p} \rightarrow \sigma_{p+1} \rightarrow \ldots \rightarrow \sigma_{q-1} \) and \( \sigma_{p+1} \rightarrow \sigma_{p+2} \rightarrow \ldots \rightarrow \sigma_{q} \) from \( \sigma \), we can get a timed event sequence \( \sigma_{R} \) which is a behavior of \( G \); and by removing the subsequences \( v_{p} \rightarrow v_{p+1} \rightarrow \ldots \rightarrow v_{q-1} \) and \( v_{p+1} \rightarrow v_{p+2} \rightarrow \ldots \rightarrow v_{q} \) from \( \rho \), we can get a path \( \rho_{R} \) such that \( \sigma_{R} \in \mathcal{L}(\rho_{R}) \). By applying the above step repeatedly, we can get a path \( \rho' \in \Delta(G, S^m_{B}(\rho, \zeta)) \) such that there is \( \sigma' \in \mathcal{L}(\rho') \) not satisfying \( \zeta \), which results in a contradiction and implies that the claim holds.

**Theorem 3** Let \( G \) be a loop-unlimited SBS, and \( S^m_{B}(e, e', d) \) be a minimal bounded delay specification. Then, \( G \) satisfies \( S^m_{B}(e, e', d) \) if and only if any path in \( \Delta(G, S^m_{B}(e, e', d)) \) satisfies \( S^m_{B}(e, e', d) \).

**Proof.** It is clear that one half of the claim holds: if \( G \) satisfies \( S^m_{B}(e, e', d) \), then any path in \( \Delta(G, S^m_{B}(e, e', d)) \) satisfies \( S^m_{B}(e, e', d) \). The other half of the claim can be proved as follows. Suppose that any path in \( \Delta(G, S^m_{B}(e, e', d)) \) satisfies \( S^m_{B}(e, e', d) \), and there is a path \( \rho \) of \( G \) such that there is a behavior \( \sigma \) of \( G \) in \( \mathcal{L}(\rho) \) which does not satisfy \( S^m_{B}(e, e', d) \). Without losing generality, suppose that \( \rho \) is of the form

\[
v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_i \rightarrow v_{i+1} \rightarrow \ldots \rightarrow v_j \rightarrow v_{j+1} \rightarrow \ldots \rightarrow v_m
\]

where \( e' \) occurs in \( \text{ref}(v_i) \), \( e \) occurs in \( \text{ref}(v_j) \) (\( 0 \leq i < j \leq m \)), and \( e, e' \) do not occur in any \( \text{ref}(v_k) \) (\( i < k < j \)); \( \sigma \) is of the form

\[
\sigma_0 \rightarrow \sigma_1 \rightarrow \ldots \rightarrow \sigma_i \rightarrow \sigma_{i+1} \rightarrow \ldots \rightarrow \sigma_j \rightarrow \sigma_{j+1} \rightarrow \ldots \rightarrow \sigma_m
\]

where \( \sigma_k \in \mathcal{L}(v_k) \) for any \( k \) (\( 0 \leq k \leq m \)); and the separation in time between \( e \) occurring in \( \sigma_j \) and \( e' \) occurring in \( \sigma_i \) is smaller than \( d \). In the following, we prove that we can construct a path \( \rho' \in \Delta(G, S^m_{B}(e, e', d)) \) such that there is a timed event sequence \( \sigma' \in \mathcal{L}(\rho') \) which does not satisfy \( S^m_{B}(e, e', d) \), which results in a contradiction and implies that the claim holds. Since \( \rho \notin \Delta(G, S^m_{B}(e, e', d)) \),

- there are \( v_p \) and \( v_q \) (\( 0 \leq p < q \leq i \)) such that \( v_p \rightarrow v_{p+1} \rightarrow \ldots \rightarrow v_q \) is loop \( (v_p = v_q) \),
- there are \( v_{p'} \) and \( v_{q'} \) (\( i + 1 \leq p' < q' \leq j - 1 \)) such that \( v_{p'} \rightarrow v_{p'+1} \rightarrow \ldots \rightarrow v_{q'} \) is loop \( (v_{p'} = v_{q'}) \), and (or)
there are $v_{p''}$ and $v_{q''}$ ($j \leq p'' < q'' \leq m$) such that $v_{p''} \rightarrow v_{p+1''} \rightarrow \ldots \rightarrow v_{q''}$ is loop ($v_{p''} = v_{q''}$).

Since $G$ satisfies the loop-closed condition and the loop-unlimited condition, any timing constraint does not combine any two nodes which are inside and outside a loop respectively, and is free to the repetition of any loop. It follows that by removing the subsequences $\sigma_p \rightarrow \sigma_{p+1} \rightarrow \ldots \rightarrow \sigma_{q-1}, \sigma_{p+1'} \rightarrow \sigma_{p+2'} \rightarrow \ldots \rightarrow \sigma_q$, and $\sigma_{p'} \rightarrow \sigma_{p+2'} \rightarrow \ldots \rightarrow \sigma_{q'}$ from $\sigma$, we can get a timed event sequence $\sigma_R$ which is a behavior of $G$ and in which the separation in time between $e$ occurring in $\sigma_j$ and $e'$ occurring in $\sigma_i$ is smaller than $d$; and by removing the subsequences $v_p \rightarrow v_{p+1} \rightarrow \ldots \rightarrow v_{q-1}, v_{p+1'} \rightarrow v_{p+2'} \rightarrow \ldots \rightarrow v_{q'}$, and $v_{p+1''} \rightarrow v_{p+2''} \rightarrow \ldots \rightarrow v_{q''}$ from $\rho$, we can get a path $\rho_R$ such that $\sigma_R \in \mathcal{L}(\rho_R)$. By applying the above step repeatedly, we can get a path $\rho' \in \Delta(G, S_B^M(e, e', d))$ such that there is a timed event sequence $\sigma' \in \mathcal{L}(\rho')$ which does not satisfies $S_B^M(e, e', d)$, which results in a contradiction and implies that the claim holds.

**Theorem 4** Let $G$ be a loop-unlimited SBS, and $S_B^M(e, e', d)$ be a maximal bounded delay specification. Then, $G$ satisfies $S_B^M(e, e', d)$ if and only if for any path $\rho$ in $\Delta(G, S_B^M(e, e', d)) \ (\mathcal{L}(\rho) \neq \emptyset)$, it satisfies $S_B^M(e, e', d)$ and there is not any node in its insulating segments which is violable for $e$ and $e'$.

**Proof.** It is clear that one half of the claim holds: if $G$ satisfies $S_B^M(e, e', d)$, then for any path $\rho$ in $\Delta(G, S_B^M(e, e', d)) \ (\mathcal{L}(\rho) \neq \emptyset)$, it satisfies $S_B^M(e, e', d)$ and there is not any node in its insulating segments which is violable for $e$ and $e'$. The reason is that if there is a path $\rho \in \Delta(G, S_B^M(e, e', d)) \ (\mathcal{L}(\rho) \neq \emptyset)$ such that there is a node $v$ in its insulating segments which is violable for $e$ and $e'$, then from a timed event sequence in $\mathcal{L}(\rho)$ we can construct a behavior of $G$ which does not satisfy $S_B^M(e, e', d)$ by repeating the positive loop in $\Theta(G, v, e, e')$ with finite times. The other half of the claim can be proved as follows. Suppose that for any path $\rho$ in $\Delta(G, S_B^M(e, e', d)) \ (\mathcal{L}(\rho) \neq \emptyset)$, it satisfies $S_B^M(e, e', d)$ and there is not any node in its insulating segments which is violable for $e$ and $e'$, and that there is a path $\rho'$ in $G$ such that $\sigma' \in \mathcal{L}(\rho')$ does not satisfy $S_B^M(e, e', d)$. Without losing generality, suppose that $\rho'$ is of the form $v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_i \rightarrow v_{i+1} \rightarrow \ldots \rightarrow v_j \rightarrow v_{j+1} \rightarrow \ldots \rightarrow v_m$ where $e'$ occurs in $ref(v_i)$, $e$ occurs in $ref(v_j)$ ($0 \leq i < j \leq m$), and $e, e'$ do not occur in any $ref(v_k) \ (i < k < j)$; $\sigma'$ is of the form

$$
\sigma_0 \rightarrow \sigma_1 \rightarrow \ldots \rightarrow \sigma_i \rightarrow \sigma_{i+1} \rightarrow \ldots \rightarrow \sigma_j \rightarrow \sigma_{j+1} \rightarrow \ldots \rightarrow \sigma_m
$$

where $\sigma_k \in \mathcal{L}(v_k)$ for any $k \ (0 \leq k \leq m)$; and the separation in time between $e$ occurring in $\sigma_j$ and $e'$ occurring in $\sigma_i$ is greater than $d$. In the following, we prove that we can construct a path $\rho'' \in \Delta(S, S_B^M(e, e', d))$ such that either there is $\sigma'' \in \mathcal{L}(\rho'')$ which does not satisfies $S_B^M(e, e', d)$ or there is a violable node for $e$ and $e'$ in the insulating segment of $\rho''$, which results in a contradiction and implies that the claim holds. Since $\rho' \not\in \Delta(G, S_B^M(e, e', d))$,
• there are \( v_p \) and \( v_q \) \((0 \leq p < q \leq i)\) such that \( v_p \rightarrow v_{p+1} \rightarrow \ldots \rightarrow v_q \) is loop \((v_p = v_q)\),

• there are \( v_p' \) and \( v_{q'} \) \((i + 1 \leq p' < q' \leq j - 1)\) such that \( v_p' \rightarrow v_{p+1'} \rightarrow \ldots \rightarrow v_{q'} \) is loop \((v_p' = v_{q'})\), and (or)

• there are \( v_{p''} \) and \( v_{q''} \) \((j \leq p'' < q'' \leq m)\) such that \( v_{p''} \rightarrow v_{p+1''} \rightarrow \ldots \rightarrow v_{q''} \) is loop \((v_{p''} = v_{q''})\).

Since \( G \) satisfies the loop-closed condition and the loop-unlimited condition, any timing constraint does not combine any two nodes which are inside and outside a loop respectively, and is free to the repetition of any loop. It follows that if the loop \( v_{p'} \rightarrow v_{p+1'} \rightarrow \ldots \rightarrow v_{q'} \) is not positive, then by removing the subsequences \( \sigma_p \rightarrow \sigma_{p+1} \rightarrow \ldots \rightarrow \sigma_{q-1}, \sigma_{p+1'} \rightarrow \sigma_{p+2'} \rightarrow \ldots \rightarrow \sigma_{q'}, \) and \( \sigma_{p+1''} \rightarrow \sigma_{p+2''} \rightarrow \ldots \rightarrow \sigma_{q''} \) from \( \sigma' \), we can get a timed event sequence \( \sigma'_R \) which is a behavior of \( G \) and in which the separation in time between \( e \) occurring in \( \sigma_j \) and \( e' \) occurring in \( \sigma_i \) is greater than \( d \); and by removing the subsequences \( v_p \rightarrow v_{p+1} \rightarrow \ldots \rightarrow v_{q-1}, v_{p+1'} \rightarrow v_{p+2'} \rightarrow \ldots \rightarrow v_{q'}, \) and \( v_{p+1''} \rightarrow v_{p+2''} \rightarrow \ldots \rightarrow v_{q''} \) from \( \rho' \), we can get a path \( \rho''_R \) such that \( \sigma'_R \in L(\rho''_R) \). By applying the above step repeatedly, we can get a path \( \rho'' \in \Delta(S, S_B^M(e, e', d)) \) such that either there is \( \sigma'' \in L(\rho'') \) which does not satisfies \( S_B^M(e, e', d) \) or there is a violable node for \( e \) and \( e' \) in the insulating segment of \( \rho'' \), which results in a contradiction and implies that the claim holds. \( \square \)