Software Engineering Group
Department of Computer Science
Nanjing University
http://seg.nju.edu.cn

Technical Report No. NJU-SEG-2012-IJ-002

Timing analysis of scenario-based specifications using linear programming

Xuandong Li, Minxue Pan, Lei Bu,

Linzhang Wang and Jianhua Zhao

Postprint Version. Originally Published in:
Software Testing, Verification and Reliability 2012
Softw. Test. Verif. Reliab. 2012; 22:121–143

SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. 2012; **22**:121–143
Published online 1 June 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/stvr.434

Timing analysis of scenario-based specifications using linear programming

Xuandong Li[1,2,*,†], Minxue Pan[1,2], Lei Bu[1,2], Linzhang Wang[1,2] and Jianhua Zhao[1,2]

[1]*State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing 210093, People's Republic of China*
[2]*Department of Computer Science and Technology, Nanjing University, Nanjing 210093, People's Republic of China*

SUMMARY

Scenario-based specifications (SBSs), 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. This paper presents an approach to timing analysis of SBSs expressed by UML interaction models. The approach considers more general and expressive timing constraints in UML sequence diagrams (SDs), and gives a solution to the *reachability analysis*, *constraint conformance analysis* and *bounded delay analysis* problems, which reduces these problems into linear programs. With the synchronous interpretation of the SD compositions, the timing analysis algorithms in the approach form a decision procedure for a class of SBSs where any loop in any path is time-independent of the other parts in the path. These algorithms are also a semi-decision procedure for general SBSs with both the synchronous and asynchronous composition semantics. The approach also supports *bounded timing analysis* of SBSs, which investigates all the paths in the bound limit one by one, and performs the timing analysis for each finite path by linear programming. A tool prototype has been developed to support this approach.

Received 9 March 2010; Accepted 25 March 2010

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 (SBSs), 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. This paper considers timing analysis of SBSs modelled by UML interaction models.

*Correspondence to: Xuandong Li, Department of Computer Science and Technology, Nanjing University, Nanjing 210093, People's Republic of China.
†E-mail: lxd@nju.edu.cn

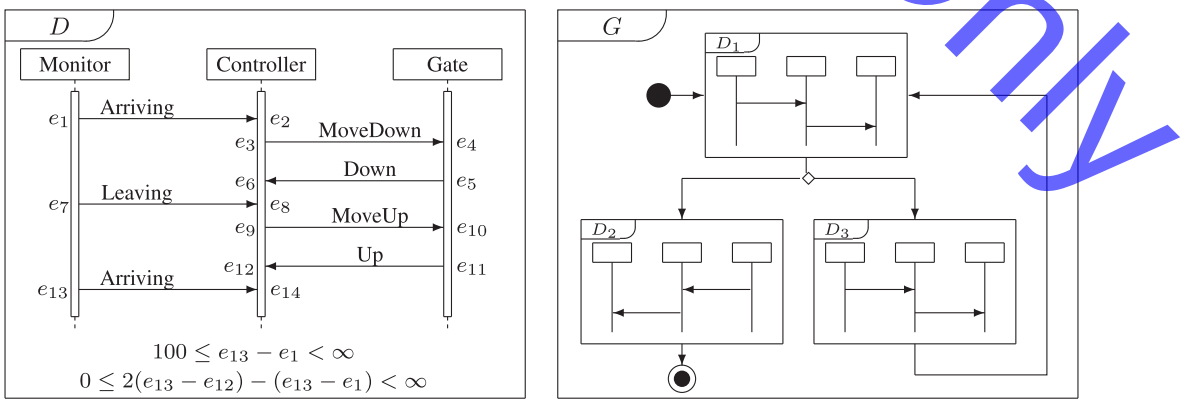
Contract/grant sponsor: National Natural Science Foundation of China; contract/grant numbers: 90818022, 60721002
Contract/grant sponsor: National 863 High-Tech Programme of China; contract/grant numbers: 2009AA01Z148, 2007AA010302
Contract/grant sponsor: National Grand Fundamental Research 973 Program of China; contract/grant number: 2009CB320702

UML sequence diagrams (SDs) 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 SD is depicted in Figure 1(a), 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 the gate down. After the train leaves the crossing, the monitor sends a message to controller to open the gate.

This paper uses a simplified version of UML SDs, which describes exactly one scenario without any alternatives and loops. For describing multiple scenarios and complete system specifications, it is necessary 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 behaviour. For example, Figure 1(b) depicts a simple interaction overview diagram.

For specifying real-time systems, timing constraints are enforced on SBSs. Several mechanisms have been introduced to describe timing constraints in MSCs and UML SDs, which are timers [1], interval delays [6, 7] and timing marks [2, 3, 8, 9]. All of those mechanisms are suitable to describe simple timing constraints which are only related to *the time separation between two events*. For example, for the SD depicted in Figure 1(a), the simple timing constraints such as *the time separation between the sending events* e_1 *and* e_{13} *that is not smaller than 100 time units* can be described by timers, interval delays or timing marks. However, in practical problems there are often the requirements to describe a class of more complex timing constraints which are about *the relation among multiple time separations between events*. For example, in the scenario about the railroad crossing system depicted in Figure 1(a), the gate is required to stay for a certain period within certain tolerance intervals, e.g. it is required that from the time one train is arriving to the time the next train is arriving, the gate stays open for at least half of this period. It means that *the time separation between the sending event* e_{13} *and the sending event* e_1 *is not greater than two times the time separation between the sending event* e_{13} *and the receiving event* e_{12}. Clearly, the existing mechanisms in MSCs and UML SDs cannot describe such a timing constraint. This paper introduces a more expressive mechanism in UML SDs to describe timing constraints, and considers checking SBSs with more complex timing constraints.

Like any other aspect of the specification and design process, SBSs are amenable to errors, and their analysis is important. Alur *et al.* [6] investigate a variety of semantic interpretations for MSCs, and develop an analyser for basic MSCs. Alur and Yannakakis [10] give a comprehensive study of model checking of MSCs for temporal requirements. Holzmann [11–13] develops the tool



(a) (b)

Figure 1. UML interaction models: (a) a sequence diagram and (b) an interaction overview diagram.

DOI: 10.1002/stvr

SPIN which supports the design of MSCs, and allows for the creation, debugging, organization and maintenance of MSCs. Peled *et al.* [14, 15] present a tool for searching a hierarchical MSC design for a path that matches a given specification. For SBSs with timing constraints used to describe real-time systems, the verification problems are more difficult and complicated. Some algorithms are presented by Alur *et al.* [6] for analysing basic MSCs with interval delays. A solution is given to the timed analogue of scenario matching by Akshay *et al.* [16]. Timing analysis has been extended to check UML SDs and MSC specifications [7, 17, 18]. However, all those works are about checking SBSs for timing consistency which is a basic property. In view of the practical use, there are a lot of properties about the accumulated delays on the traces of systems. For example, there often is a need to check if all traces of a system satisfy that the separation in time between two given events is within a given time interval, which is called *bounded delay analysis*. This problem has been considered for timed automata by Courcoubetis and Yannakakis [19], and for a class of Petri nets by Hulgaard and Burns [20].

This paper presents an approach to timing analysis of SBSs expressed by UML interaction models. The approach gives a solution to the *reachability analysis*, *constraint conformance analysis* and *bounded delay analysis* problems, which reduces these problems into linear programs. With the synchronous interpretation of the SD compositions, the timing analysis algorithms in the approach form a decision procedure for a class of SBSs where any loop in any path is time-independent of the other parts in the path. These algorithms are also a semi-decision procedure for general SBSs with both the synchronous and asynchronous composition semantics. The approach also supports *bounded* timing analysis of SBSs, which investigates all the paths in the bound limit one by one, and performs the timing analysis for each finite path by linear programming. A tool prototype has been developed to support this approach.

The paper is organized as follows. The following section introduces SBSs expressed by UML interaction models. Section 3 gives the linear programming-based approach to timing analysis of SBSs. The last section discusses the related work and contains some conclusions.

2. SCENARIO-BASED SPECIFICATIONS

In this paper, UML interaction models are used as SBSs, which consist of UML2.0 interaction overview diagrams and SDs.

2.1. UML sequence diagrams and timing constraints

This paper just uses a simplified version of UML SDs, which describes exactly one scenario without any alternatives and loops. An SD 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.

This paper considers more general and expressive timing constraints in SDs. In an SD, *events* are the message sending and message receiving. Here event names are used to represent the occurrence time of events, and linear inequalities on event names are used to represent the timing constraints. A timing constraint is of the form

$$a \le c_0(e_0 - e_0') + c_1(e_1 - e_1') + \cdots + c_n(e_n - e_n') \le b$$

where e_i and e_i' $(0 \le i \le n)$ 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 ∞). For example, for the scenario about the railroad crossing system depicted in Figure 1(a), it is required that when a train has passed, a new train should come after at least 100 time units, which can be represented by the timing constraint $100 \le e_{13} - e_1 < \infty$. Compared to timers, interval delays and timing marks, the timing constraints considered here can be used to describe more complex timing requirements in practical use. For example, for the scenario about the railroad crossing system depicted in Figure 1(a), the timing

DOI: 10.1002/stvr

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 must stay open for at least half of this period. Clearly, such a timing requirement is about *the relation between two time separations between events* (one is the time separation between e_{13} and e_{12}, and the other is the time separation between e_{13} and e_1), and none of the timers, interval delays and timing marks can be used to describe such a timing requirement since they are only suitable for describing the simple timing constraints related to *the time separation between two events*.

The semantics of an SD 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 SD along with a causal dependency between the events of sending and receiving a message [1–3, 21]. In accordance with the definition given by Peled [21], for a pair of events e and e' in an SD, e precedes e' (denoted by $e \prec e'$) 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 analysing SBSs, SDs are formalized as follows:

Definition 1
An 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 which corresponds to sending and receiving a message. There are two special events ε and ϖ 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 and $e' \in E$ are corresponding to the sending and the receiving of a message, respectively.
- $L : E \to O$ is a labelling function which maps each event $e \in E$ to an object $L(e) \in O$ which is the sender (receiver) whereas e corresponds to sending (receiving) a message.
- V is a finite set whose elements are a pair (e, e') (e and $e' \in E$). For any events e and $e' \in E$, if $e \prec e'$ then $(e, e') \in V$. For any event $e \in E$ $(e \neq \varepsilon \wedge e \neq \varpi)$, $(\varepsilon, e) \in V$ and $(e, \varpi) \in V$.
- C is a finite set of timing constraints.

The *event sequences* are used to represent the *traces* of SDs which correspond to the untimed behaviour of SDs. An *event sequence* is of the form $e_0 \to e_1 \to \cdots \to e_m$, 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 $e_0 \to e_1 \to \cdots \to e_m$ is a *trace* of D if and only if the following conditions hold:

- $e_0 = \varepsilon$ and $e_m = \varpi$.
- $e_0, e_1, \ldots, e_m$ 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 $0 \leq i < j \leq m$.

The *timed event sequences* are used to represent the behaviour of SDs. A *timed event sequence* is of the form $(e_0, t_0) \to (e_1, t_1) \to \cdots \to (e_m, t_m)$ where e_i is an event and t_i is a non-negative real number 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, and finally 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 $\sum_{j=0}^{i} t_j$.

DOI: 10.1002/stvr

Definition 3
Let $D=(O, E, M, L, V, C)$ be an SD. A timed event sequence

$$\sigma=(e_0, t_0)\rightarrow(e_1, t_1)\rightarrow\cdots\rightarrow(e_m, t_m)$$

is a behaviour of D if and only if the following conditions hold:

- $e_0\rightarrow e_1\rightarrow\cdots\rightarrow e_m$ is a trace of D.
- $t_0=0$, and $t_0, t_1, \ldots, t_m$ 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, $a\leq c_0\delta_0+c_1\delta_1+\cdots+c_n\delta_n\leq b$ where for each i $(0\leq i\leq n)$, if $f_i=e_j$ and $f_i'=e_k$, then

$$\delta_i=\begin{cases} t_{k+1}+t_{k+2}+\cdots+t_j & \text{if } j>k \\ -(t_{j+1}+t_{j+2}+\cdots+t_k) & \text{if } j<k \end{cases}$$

Let $\mathscr{L}(D)$ denote the set of the timed event sequences representing the behaviour of D.

2.2. *UML2.0 interaction overview diagrams and SBSs*

An SD considered in this paper describes exactly one scenario. For describing multiple scenarios and complete system specifications, it is necessary 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 behaviour.

This paper considers timing analysis of SBSs. An SBS under analysis is represented by an interaction overview diagram, which is defined formally as follows:

Definition 4
An SBS G is a tuple $G=(U, N, succ, ref, T)$ where

- U is a finite set of SDs satisfying the following: 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 three sets: the singleton-set of *start* node, the set of *intermediate* nodes and the singleton-set of *end* node, respectively.
- $succ\subset N\times N$ is the relation which reflects the connectivity of the nodes in N (it is required that any node in N is reachable from the start node).
- $ref: I\mapsto U$ 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 ∞), 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 $v_0\rightarrow v_1\rightarrow\cdots\rightarrow v_n$ satisfying $(v_{i-1}, v_i)\in succ$ for any i $(0<i\leq n)$. A *path* is a path segment $v_0\rightarrow v_1\rightarrow\cdots\rightarrow v_n$ such that $(\top, v_0)\in succ$ and $(v_n, \bot)\in succ$.

The timing constraints in SBSs are interpreted by *local semantics*: select one path at one time and analyse 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 is quite in accord with the *synchronous mode* interpreting the concatenation of two SDs in an SBS: 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]. With this synchronous composition semantics, the behaviour of an SBS G are defined as the timed event sequences which are the concatenation of the timed event sequences representing the behaviour of the SDs which make up G.

DOI: 10.1002/stvr

Definition 5
Let $G=(U, N, succ, ref, T)$ be an SBS. For any path segment

$$\rho = v_0 \to v_1 \to \cdots \to v_m$$

in G, let $\mathscr{L}(\rho)$ be the set of all timed event sequences of the form $(e_0, t_0) \to (e_1, t_1) \to \cdots \to (e_n, t_n)$ satisfying the following condition:

- $(e_0, t_0) \to (e_1, t_1) \to \cdots \to (e_n, t_n) = \sigma_0 \to \sigma_1 \to \cdots \to \sigma_m$, where σ_i is a behaviour of $ref(v_i)$ for each i $(0 \le i \le m)$ and
- $(e_0, t_0) \to (e_1, t_1) \to \cdots \to (e_n, t_n)$ satisfies any timing constraints in T, i.e. for any $a \le f - f' \le b \in T$, for any i, j $(0 \le i < j \le n)$ such that $f' = e_i$, $f = e_j$, and that $f \ne e_k \wedge f' \ne e_k$ for any k $(i < k < j)$, $a \le t_{i+1} + t_{i+2} + \cdots + t_j \le b$.

A timed event sequence σ is a behaviour of G if and only if there is a path ρ in G such that $\sigma \in \mathscr{L}(\rho)$. □

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 $v_0 \to v_1 \to \cdots \to v_n$ be a simple path segment in G such that $(\top, v_0) \in succ$. If there is v_i $(0 \le i \le n)$ such that $(v_n, v_i) \in succ$, then the sequence $v_i \to v_{i+1} \to \cdots \to v_n \to v_i$ is a *loop*, and v_i is the *loop-start node* of the loop.

For an SBS $G=(U, N, succ, ref, T)$, let $\rho = v_0 \to v_1 \to \cdots \to v_n$ be a path segment. For a timing constraint $a \le e - e' \le b$ in T, if e occurs in $ref(v_i)$, e' occurs in $ref(v_j)$ $(0 \le j < i \le n)$ and e, e' do not occur in any $ref(v_k)$ $(j < k < i)$, then this timing constraint is said to *combine* nodes v_i and v_j in ρ (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 various cases when a timing constraint $a \le e - e' \le b$ combines two nodes v and v' in a path segment ρ with a loop.

This paper develops 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 the *loop-closed condition* and the *loop-unlimited condition*. For an SBS $G=(U, N, succ, ref, T)$, the *loop-closed condition* requires that in any path of G 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 cases (1) and (2) in Figure 2), i.e. any timing constraint in T of the form $a \le e - e' \le b$ must satisfy:

- for any loop $v_0 \to v_1 \to \cdots \to v_m$, if e occurs in $ref(v_i)$ $(0 \le i < m)$ and e' does not occur in any $ref(v_j)$ $(0 \le j < i)$, then there is no simple path segment $v'_0 \to v'_1 \to \cdots \to v'_n$ such that $v'_n = v_0$, e' occurs in $ref(v'_0)$, and that e does not occur in any $ref(v'_k)$ $(0 \le k \le n)$ and
- for any loop $v_0 \to v_1 \to \cdots \to v_m$, if e' occurs in $ref(v_i)$ $(0 < i \le m)$ and e does not occur in any $ref(v_j)$ $(i < j \le m)$, then there is no simple path segment $v'_0 \to v'_1 \to \cdots \to v'_n$ such that $v'_0 = v_0$, e occurs in $ref(v'_n)$, and that e' does not occur in any $ref(v'_k)$ $(0 \le k \le n)$.

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

- for any loop $v_0 \to v_1 \to \cdots \to v_m$ where e and e' do not occur in any $ref(v_i)$ $(0 \le i \le m)$, there is no simple path segment $v'_0 \to v'_1 \to \cdots \to v'_n$ such that there is 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 and e' do not occur in any $ref(v'_i)$ $(0 < i < n)$.

An SBS is said to be *loop-unlimited* if it satisfies both the loop-closed condition and loop-unlimited condition. The purpose for enforcing the loop-closed condition and loop-unlimited condition on an SBS is such that any loop in any path is time-independent of the other parts in the path, and this restriction is reasonable in many cases.

DOI: 10.1002/stvr

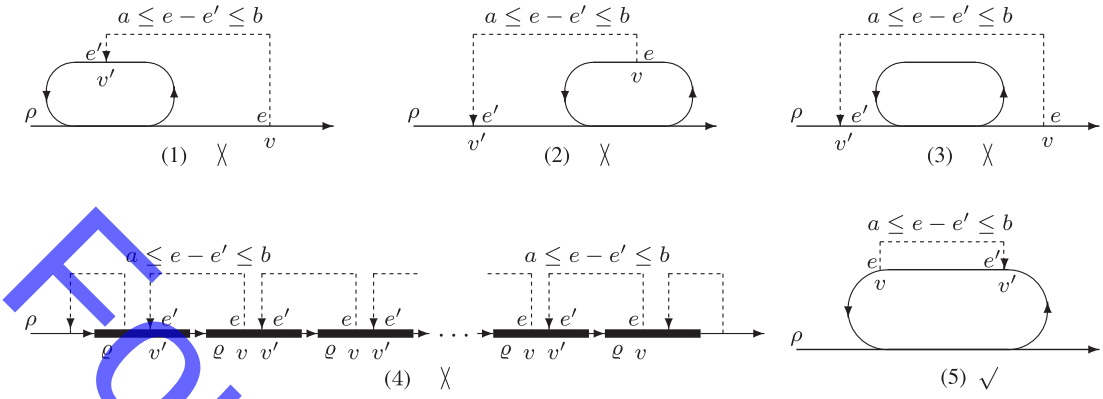


Figure 2. Loop-closed condition and loop-unlimited condition.

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

In an SBS G, a common case that violates the loop-closed condition is that there is a *reverse constraint* for a loop. Let $\varrho = v_0 \to v_1 \to \cdots \to v_m$ be a loop in a path in G. A timing constraint $a \le e - e' \le b$ is said to be a *reverse constraint* for ϱ if e occurs in ϱ prior to e', i.e. e occurs in $ref(v_i)$ and e' occurs in $ref(v_j)$ $(0 \le i < j \le n)$. In a path ρ which contains the repetition of ϱ, the reverse constraint $a \le e - e' \le b$ essentially combines two nodes in conjunction with two occurrences of ϱ such that the former occurrence of ϱ is time dependent of the latter occurrence of ϱ, which is illustrated by case (4) in Figure 2. In the authors' opinion, the reverse constraints are seldom adopted in practical use because their denotations are barely perceptible in terms of the structure of SBSs. Notice that the loop-closed condition allows a timing constraint to combine two nodes in the same loop, such as case (5) in Figure 2. Such a constraint is in common use, and its denotation is perceptible in terms of the structure of SBSs.

An efficient algorithm can be developed to check if an SBS G is loop-unlimited, which is described in Appendix A.

2.4. An automatic teller machine example

For illustrating the approach presented in this paper, the automatic teller machine (ATM) system [7] serves as an example. Its specification is depicted in Figure 3, which 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 in [0, 2] s (at least 0 but no more than 2 s) after receiving a card ($0 \le b_1 - a_2 \le 2$, SDs StartTrans and GetPin). Then, it either receives a request to cancel the transaction within [0, 4] s ($0 \le c_2 - b_1 \le 4$, SD EndTrans), or receives the customer's PIN with [5, 60] s ($5 \le d_2 - b_1 \le 60$, SD ProcessPin). If the ATM receives a request to cancel the transaction, it returns the customer's card and takes [2, 3] s to return to its initial state ($2 \le \varpi_c - c_3 \le 3$, SD EndTrans). The ATM expects a reply from the bank within 10 s, which is represented by the following timing constraints:

$$0 \le f_2 - d_3 \le 10, \quad 0 \le g_2 - d_3 \le 10, \quad 0 \le j_2 - h_7 \le 10$$

$$0 \le k_2 - h_7 \le 10, \quad 0 \le i_6 - i_3 \le 10, \quad 0 \le j_8 - j_5 \le 10.$$

If no reply from the bank is received within 10 s, the card is returned, an appropriate message is then displayed, and the ATM takes [2, 3] s to return to its initial state ($e_1 - d_3 = 10$, $2 \le \varpi_e - e_5 \le 3$,

DOI: 10.1002/stvr

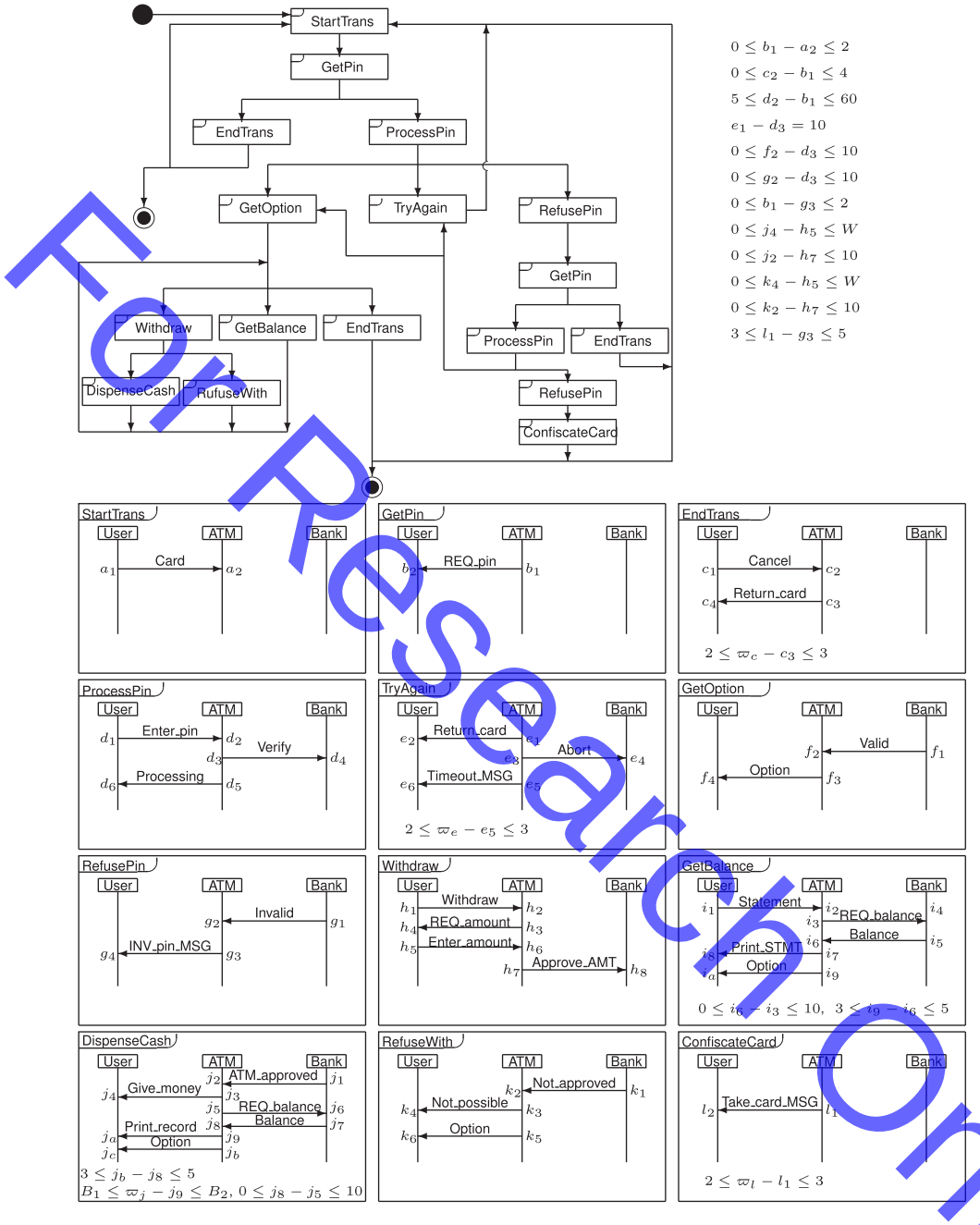


Figure 3. Scenario-based specification for the ATM example.

SD TryAgain). The specification also describes the following constraints:

- a customer expects a withdraw request to be processed within $[0, W]$ s relative to the time of entering an amount ($0 \le j_4 - h_5 \le W, 0 \le k_4 - h_5 \le W$);
- the ATM takes $[B_1, B_2]$ s for bookkeeping after dispensing cash ($B_1 \le \varpi_j - j_9 \le B_2$, SD DispenseCash);
- the ATM takes $[3, 5]$ s to print a receipt after receiving the balance information from the bank ($3 \le j_b - j_8 \le 5$, $3 \le i_9 - i_6 \le 5$, SD DispenseCash, SD GetBalance) and
- in the case of refusing PIN, the first time the ATM takes $[0, 2]$ s to request a PIN again after sending the information for the invalid PIN ($0 \le b_1 - g_3 \le 2$), and the second time it takes $[3, 5]$ s to confiscate the card and inform the customer ($3 \le l_1 - g_3 \le 5$, SD ConfiscateCard).

DOI: 10.1002/stvr

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

This section describes the linear programming-based approach to timing analysis of SBSs, including the *reachability analysis*, the *constraint conformance analysis* and the *bounded delay analysis*.

3.1. Reachability analysis

Reachability analysis checks if a given node of an SBS is *reachable* along a behaviour of the SBS. Let $G=(U, N, succ, ref, T)$ be an SBS. For a given node $v \in N$, the reachability analysis checks if there is a path ρ passing through v which is of the form $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_i \rightarrow \cdots \rightarrow v_m$ such that $v_i = v$ $(0 \leq i \leq m)$ and that $\mathscr{L}(\rho) \neq \emptyset$. For example, for the ATM system given in Section 2.4, the reachability analysis checks if the node SD DispenseCash is reachable in the specification depicted in Figure 3.

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

- for the start event ε_0 of $ref(v_0)$, if $e_i = \varepsilon_0$ $(0 \leq i \leq n)$ then $t_i = 0$;
- for the end event ϖ_i of $ref(v_i)$ and the start event ε_{i+1} of $ref(v_{i+1})$ $(0 \leq i < m)$, if $e_p = \varpi_i$ $(0 \leq p \leq m)$ and $e_q = \varepsilon_{i+1}$ $(0 \leq q \leq m)$ then $t_p = t_q$;
- 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.

As $\mathscr{L}(\rho) \neq \emptyset$ if and only if $lp(\rho)$ has a solution, the reachability analysis problem for a node v of G can be reduced into the linear programming problems as follows: check if there is a path ρ' in G passing through v such that $lp(\rho') \neq \emptyset$. It is clear that in the worst case, all the paths in G that pass through v need to be checked. Since the number of paths of G could be infinite, and the length of a path of G could be infinite, it is necessary 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 $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_i \rightarrow v_{i+1} \rightarrow \cdots \rightarrow v_m$, 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 ρ 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 ρ_1 in G such that $\rho \rightarrow \rho_1$ is in $\Delta(G, v)$. The following theorem shows that if G is loop-unlimited, the reachability analysis of v just needs to check each path in $\Delta(G, v)$.

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 $\mathscr{L}(\rho) \neq \emptyset$.

The proof of this theorem is presented in Appendix B. Based on the above theorem, an algorithm can be developed to check if a node v in an SBS G is reachable (cf. Figure 4). The algorithm first checks if G is loop-unlimited, and assigns the result to the boolean variable *loop_unlimited*. Then, it traverses the state space of the nodes of G in a depth-first manner starting from the start node $\top$.

DOI: 10.1002/stvr

```
check if G is loop-unlimited;
if yes then loop_unlimited :=true else loop_unlimited :=false;
currentpath := ⟨⊤⟩;
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 ρ corresponding to the concatenation of
                   currentpath and node is in Δ(G, υ)
              then begin check if L(ρ) ≠ ∅;
                             if yes then return true;
                   end;
              if the path segment corresponding to the concatenation of
                   currentpath and node is a prefix for Δ(G, υ)
              then append node to currentpath;
         end
until currentpath = ⟨⟩;
if loop_unlimited then return false else return undecided.
```

Figure 4. Algorithm for reachability analysis.

The path in the state space that the algorithm has so far traversed is stored in the list variable *currentpath*. For each successive node *node* of the last node of *currentpath*, the algorithm first checks whether the path segment ρ corresponding to the concatenation of *currentpath* and *node* is in $\Delta(G,\upsilon)$. If yes, then it checks if $\mathscr{L}(\rho)\neq\emptyset$ by linear programming, and returns 'true' when $\mathscr{L}(\rho)\neq\emptyset$. If the path segment corresponding to the concatenation of *currentpath* and *node* is a prefix for $\Delta(G,\upsilon)$, then the algorithm adds *node* to the current path and starts the search from it, otherwise it searches 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, the algorithm returns 'false' when G is loop-unlimited, and 'undecided' when G is not loop-unlimited. Notice that the algorithm can answer 'true' for some SBSs that 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.

3.2. Constraint conformance analysis

Constraint conformance analysis checks if the given several scenarios, which occur continuously in the behaviour of an SBS, satisfy a given timing constraint. Let $G=(U,N,succ,ref,T)$ be an SBS. A *constraint conformance specification*, denoted by $\mathscr{S}_C(\varrho,\zeta)$, consists of a finite sequence $\varrho=D_0\rightarrow D_1\rightarrow\cdots\rightarrow D_k$ of SDs in U and a timing constraint ζ 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 ρ be a path in G of the form $\upsilon_0\rightarrow\upsilon_1\rightarrow\cdots\rightarrow\upsilon_m$. For $\mathscr{S}_C(\varrho,\zeta)$, an *occurrence* of ϱ in ρ is a subsequence of ρ of the form $\upsilon_j\rightarrow\upsilon_{j+1}\rightarrow\cdots\rightarrow\upsilon_{j+k}$ $(0\leq j\leq m-k)$ such that $ref(\upsilon_i)=D_{i-j}$ for any i $(j\leq i\leq j+k)$, and it *satisfies* ζ if the following condition holds:

- for any $\sigma=\sigma_0\rightarrow\sigma_1\rightarrow\cdots\rightarrow\sigma_m\in\mathscr{L}(\rho)$ where every σ_i $(0\leq i\leq m)$ is a behaviour of $ref(\upsilon_i)$, 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 $(0\leq i\leq l)$, if $f_i=e_p$ and $f_i'=e_q$ $(0\leq p,q\leq n)$ then

$$\delta_i=\begin{cases}t_{q+1}+t_{q+2}+\cdots+t_p & \text{if } p>q\\ -(t_{p+1}+t_{p+2}+\cdots+t_q) & \text{if } p<q\end{cases}$$

ρ *satisfies* $\mathscr{S}_C(\varrho,\zeta)$ if every occurrence of ϱ in ρ satisfies ζ, and G *satisfies* $\mathscr{S}_C(\varrho,\zeta)$ if any path in G satisfies $\mathscr{S}_C(\varrho,\zeta)$. For example, for the ATM example given in Section 2.4, since a customer may lose his patience after he gets the money, it is required that the time that the ATM takes for the printing and bookkeeping after giving the money be not greater than half of the time that the

DOI: 10.1002/stvr

customer waits for withdrawing the money, which forms a constraint conformance specification $\mathscr{S}_C(\varrho,\zeta)$, where $\varrho=\mathsf{Withdraw}\rightarrow\mathsf{DispenseCash}$ and $\zeta=2(\varpi_j-j_8)\leq j_c-h_5$.

Let $G=(U,N,succ,ref,T)$ be an SBS, and $\mathscr{S}_C(\varrho,\zeta)$ be a constraint conformance specification where $\varrho=D_0\rightarrow D_1\rightarrow\cdots\rightarrow D_k$ and ζ is of the form $a\leq\sum_{i=0}^{l}c_i(f_i-f_i')\leq b$. The following shows that for a finite path ρ in G, the constraint conformance analysis for $\mathscr{S}_C(\varrho,\zeta)$ can be solved by linear programming. Suppose $\rho=v_0\rightarrow v_1\rightarrow\cdots\rightarrow v_m$ and $\mathscr{L}(\rho)\neq\emptyset$. As there could be v_i and v_j $(0\leq i<j\leq m)$ such that $ref(v_i)=ref(v_j)$, by renaming, let $E_i\cap E_j=\emptyset$ for any i, j $(0\leq i<j\leq m)$. Let

$$E=E_1\cup E_2\cup\cdots\cup E_m=\{e_0,e_1,\ldots,e_n\},$$

and t_i represent the occurrence time of e_i $(0\leq i\leq n)$ in a timed event sequence in $\mathscr{L}(\rho)$. Suppose that there is a subsequence ρ_1 of ρ of the form $v_j\rightarrow v_{j+1}\rightarrow\cdots\rightarrow v_{j+k}$ $(0\leq j\leq m-k)$ such that $ref(v_i)=D_{i-j}$ for any i $(j\leq i\leq j+k)$. Then, the satisfaction problem of ρ for $\mathscr{S}_C(\varrho,\zeta)$ can be reduced to a linear program 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 $\delta_i=t_p-t_q\wedge f_i=e_p\wedge f_i'=e_q$ (e_p and e_q occurs in ρ_1) for any i $(0\leq i\leq l)$, and checking whether it is not greater than b (smaller than a).

For checking all the paths of an SBS for a given constraint conformance specification, it is needed to solve the problem based on a finite set of finite paths. Let $G=(U,N,succ,ref,T)$ be an SBS, and $\mathscr{S}_C(\varrho,\zeta)$ be a constraint conformance specification, where $\varrho=D_0\rightarrow D_1\rightarrow\cdots\rightarrow D_k$. Let $\Delta(G,\mathscr{S}_C(\varrho,\zeta))$ be a set of the paths in G of the form

$$v_0\rightarrow v_1\rightarrow\cdots\rightarrow v_i\rightarrow u_0\rightarrow u_1\rightarrow\cdots\rightarrow u_k\rightarrow v_{i+1}\rightarrow v_{i+2}\rightarrow\cdots\rightarrow v_m$$

where $ref(u_i)=D_i$ 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 $\Delta(S,\mathscr{S}_C(\varrho,\zeta))$ consists of three parts in succession: the first and last parts are simple path segments and the middle part corresponds to ϱ. A path segment ρ in G is a *prefix* for $\Delta(G,\mathscr{S}_C(\varrho,\zeta))$ if it may be extended into a path which is in $\Delta(G,\mathscr{S}_C(\varrho,\zeta))$, i.e. there could be a path segment ρ_1 in G such that $\rho\rightarrow\rho_1$ is in $\Delta(G,\mathscr{S}_C(\varrho,\zeta))$. The following theorem indicates that if G is loop-unlimited, the constraint conformance analysis for $\mathscr{S}_C(\varrho,\zeta)$ just needs to check each path in $\Delta(G,\mathscr{S}_C(\varrho,\zeta))$.

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

The proof of this theorem is presented in Appendix B. Based on the above theorem, an algorithm can be developed to check if an SBS G satisfies a constraint conformance specification $\mathscr{S}_C(\varrho,\zeta)$ (cf. Figure 5). The algorithm first checks if G is loop-unlimited, and assigns the result to the boolean variable *loop_unlimited*. Then, it 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 the algorithm has so far traversed is stored in the list variable *currentpath*. For each successive node *node* of the last node of *currentpath*, the algorithm first checks whether the path segment ρ corresponding to the concatenation of *currentpath* and *node* is in $\Delta(G,\mathscr{S}_C(\varrho,\zeta))$. If yes, then it checks if ρ satisfies $\mathscr{S}_C(\varrho,\zeta)$ by linear programming, and returns 'false' when $\mathscr{S}_C(\varrho,\zeta)$ is not satisfied. Then the algorithm checks if the path segment corresponding to the concatenation of *currentpath* and *node* is a prefix for $\Delta(G,\mathscr{S}_C(\varrho,\zeta))$. If yes, then it adds *node* to the current path and starts the search from it, otherwise searches 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, the algorithm returns 'true' when G is loop-unlimited, and 'undecided' when G is not loop-unlimited. Notice that the algorithm can answer 'false' for some SBSs that 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

Bounded delay analysis checks whether the time separation between the two given events in any behaviour of an SBS is not smaller or greater than a given real number, which is called the *minimal*

DOI: 10.1002/stvr

```
check if G is loop-unlimited;
if yes then loop_unlimited :=true else loop_unlimited :=false;
currentpath := ⟨⊤⟩;
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 ρ corresponding to the concatenation of
                currentpath and node is in Δ(G, S_C(ϱ, ζ))
            then begin check if ρ satisfies S_C(ϱ, ζ);
                        if no then return false;
                 end;
            if the path segment corresponding to the concatenation of
                currentpath and node is a prefix for Δ(G, S_C(ϱ, ζ))
            then append node to currentpath;
         end
until currentpath = ⟨⟩;
if loop_unlimited then return true else return undecided.
```

Figure 5. Algorithm for constraint conformance analysis.

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. Then the following just considers 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 $\mathscr{S}_B^m(e,e',d)$ (or $\mathscr{S}_B^M(e,e',d)$), which requires that the time separation between e and e' in any behaviour of G be not smaller (or greater) than d. For example, for the ATM example given in Section 2.4, for recording the process for withdrawing money with a camera embedded in the ATM, it is required that every process for withdrawing money takes time which is long enough for recording, which forms a minimal bounded delay specification $\mathscr{S}_B^m(j_4,a_1,20)$.

Let $G=(U,N,succ,ref,T)$ be an SBS, $\mathscr{S}_B^m(e,e',d)$ (or $\mathscr{S}_B^M(e,e',d)$) be a bounded delay specification and σ be a behaviour of G of the form

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

If for any i and j $(0\leq i<j\leq n)$ such that $e_i=e'$, $e_j=e$, and that $e_k\neq e\wedge e_k\neq e'$ for any k $(i<k<j)$, $t_{i+1}+t_{i+2}+\cdots+t_j\geq(\leq)d$, then σ is said to *satisfy* $\mathscr{S}_B^m(e,e',d)$ (or $\mathscr{S}_B^M(e,e',d)$). A path ρ in G *satisfies* $\mathscr{S}_B^m(e,e',d)$ (or $\mathscr{S}_B^M(e,e',d)$) if any $\sigma\in\mathscr{L}(\rho)$ satisfies $\mathscr{S}_B^m(e,e',d)$ (or $\mathscr{S}_B^M(e,e',d)$), and G *satisfies* $\mathscr{S}_B^m(e,e',d)$ (or $\mathscr{S}_B^M(e,e',d)$) if any path in G satisfies $\mathscr{S}_B^m(e,e',d)$ (or $\mathscr{S}_B^M(e,e',d)$).

Let $G=(U,N,succ,ref,T)$ be an SBS, and $\mathscr{S}_B^m(e,e',d)$ (or $\mathscr{S}_B^M(e,e',d)$) be a bounded delay specification. The following shows that for a finite path ρ in G, the satisfaction problem of ρ for $\mathscr{S}_B^m(e,e',d)$ (or $\mathscr{S}_B^M(e,e',d)$) can be reduced into a linear programming problem. Suppose that

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

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 $ref(v_k)$ $(i<k<j)$. As there could be v_k and v_l $(0\leq k<l\leq m)$ such that $ref(v_k)=ref(v_l)$, by renaming, let $E_k\cap E_l=\emptyset$ for any k,l $(0\leq k<l\leq m)$. Let $E=E_1\cup E_2\cup\cdots\cup E_m=\{e_0,e_1,\ldots,e_n\}$, and t_k represent the occurrence time of e_k $(0\leq k\leq n)$ in a timed event sequence in $\mathscr{L}(\rho)$. Suppose that e' occurring in $ref(v_i)$ is e_p and e occurring in $ref(v_j)$ is e_q $(0\leq p,q\leq n)$. Then, the problem of checking if the time separation between e occurring in $ref(v_j)$ and e' occurring in $ref(v_i)$ is not smaller (or greater) than d in any $\sigma\in\mathscr{L}(\rho)$ can be solved as follows: finding the minimum (or maximum) value of the linear function t_q-t_p subject to the linear constraint $lp(\rho)$, and checking whether it is not smaller (or greater) than d, which can be solved by linear programming. For all the paths in G, the solution for the satisfaction problem for $\mathscr{S}_B^m(e,e',d)$ (or $\mathscr{S}_B^M(e,e',d)$) needs to be based on a finite set of the finite paths in G.

DOI: 10.1002/stvr

Let $G=(U, N, succ, ref, T)$ be an SBS, and $\mathscr{S}_B^m(e, e', d)$ (or $\mathscr{S}_B^M(e, e', d)$) be a bounded delay specification. Let $\Delta(G, \mathscr{S}_B^m(e, e', d))$ and $\Delta(G, \mathscr{S}_B^M(e, e', d))$ be the set of the paths in G of the form

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

where

- e' occurs in $ref(v_i)$, e occurs in $ref(v_j)$ $(0 \le i < j \le m)$, and e, e' do not occur in any $ref(v_k)$ $(i < k < j)$ and
- all v_l $(0 \le l \le i)$ are distinct, all v_k $(i \le k \le j)$ are distinct, and all v_p $(j \le p \le m)$ are distinct.

Intuitively, each path in $\Delta(G, \mathscr{S}_B^m(e, e', d))$ (or $\Delta(G, \mathscr{S}_B^M(e, e', d))$) is separated by e' and e into three simple path segments in succession. For an SBS G, for a bounded delay specification $\mathscr{S}_B^m(e, e', d)$ (or $\mathscr{S}_B^M(e, e', d)$), a path segment ρ in G is a *prefix* for $\Delta(G, \mathscr{S}_B^m(e, e', d))$ (or $\Delta(G, \mathscr{S}_B^M(e, e', d))$) if it may be extended into a path which is in $\Delta(G, \mathscr{S}_B^m(e, e', d))$ (or $\Delta(G, \mathscr{S}_B^M(e, e', d))$), i.e. there could be a path segment ρ_1 in G such that $\rho \rightarrow \rho_1$ is in $\Delta(G, \mathscr{S}_B^m(e, e', d))$ (or $\Delta(G, \mathscr{S}_B^M(e, e', d))$).

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

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

The proof of this theorem is presented in Appendix B. Based on Theorem 3, an algorithm can be developed to check if an SBS G satisfies a minimal bounded delay specification $\mathscr{S}_B^m(e, e', d)$ (cf. Figure 6). The structure of the algorithm is the same as the algorithm depicted in Figure 5. As 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, it needs more consideration because it is a little more complicated than the minimal bounded delay analysis. First, it is needed to introduce the *violable nodes* for the two given events in an SBS. A loop ϱ in an SBS G is said to be *positive* if a repetition of the loop may take time, i.e. there is $(e_0, t_0) \rightarrow (e_1, t_1) \rightarrow \cdots \rightarrow (e_n, t_n) \in \mathscr{L}(\varrho)$ such that $t_0 + t_1 + \cdots + t_n > 0$ (notice that checking if a loop is positive can be solved by linear programming). A loop ϱ $(\mathscr{L}(\varrho) \neq \emptyset)$ in an SBS G is said to be *free of* an event e if e does not occur in any node in ϱ. For an SBS G, a node v in G, and for two events e, e' in G, the set $\Theta(G, v, e, e')$ is defined

```
check if G is loop-unlimited;
if yes then loop_unlimited :=true else loop_unlimited :=false;
currentpath := ⟨⊤⟩;
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 ρ corresponding to the concatenation of
                 currentpath and node is in Δ(G, S_B^m(e, e', d))
              then begin check if ρ 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 Δ(G, S_B^m(e, e', d))
              then append node to currentpath;
        end
until currentpath = ⟨⟩;
if loop_unlimited then return true else return undecided.
```

Figure 6. Algorithm for minimal bounded delay analysis.

DOI: 10.1002/stvr

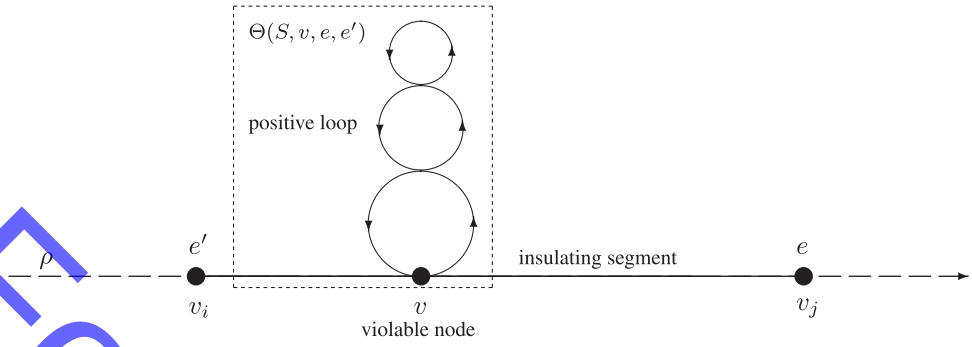


Figure 7. The intuition of Theorem 4.

recursively as follows:

- any loop ϱ ($\mathscr{L}(\varrho) \neq \emptyset$) in G free of e and e' whose loop-start node is v belongs to $\Theta(G, v, e, e')$ and
- for any loop in $\Theta(G, v, e, e')$ of the form $v_0 \to v_1 \to \cdots \to v_m$, any loop ϱ ($\mathscr{L}(\varrho) \neq \emptyset$) 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 v is said to be a *violable node* for e and e'.

Then, for an SBS G and for a maximal bounded delay specification $\mathscr{S}_B^M(e, e', d)$, the *insulating segments* are introduced in the paths in $\Delta(G, \mathscr{S}_B^M(e, e', d))$. For any $\rho \in \Delta(G, \mathscr{S}_B^M(e, e', d))$ of the form

$$v_0 \to v_1 \to \cdots \to v_i \to v_{i+1} \to \cdots \to v_j \to v_{j+1} \to \cdots \to v_m$$

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 $ref(v_k)$ $(i < k < j)$, its *insulating segment* is the subsequence between e' and e of the form $v_{i+1} \to v_{i+2} \to \cdots \to v_{j-1}$. Notice that for a path $\rho \in \Delta(G, \mathscr{S}_B^M(e, e', d))$, if there is a node v in its insulating segment which is violable for e and e', then a behaviour of G, which does not satisfy $\mathscr{S}_B^M(e, e', d)$, can be constructed 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 7.

Theorem 4
Let G be a loop-unlimited SBS, and $\mathscr{S}_B^M(e, e', d)$ be a maximal bounded delay specification. Then, G satisfies $\mathscr{S}_B^M(e, e', d)$ if and only if for any path ρ in $\Delta(G, \mathscr{S}_B^M(e, e', d))$ $(\mathscr{L}(\rho) \neq \emptyset)$, it satisfies $\mathscr{S}_B^M(e, e', d)$ 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 Appendix B. Based on Theorem 4, an algorithm can be developed to check if an SBS G satisfies a maximal bounded delay specification $\mathscr{S}_B^M(e, e', d)$ (cf. Figure 8). The structure of the algorithm is the same as the algorithm depicted in Figure 6. The difference from the minimal bounded delay analysis algorithm is that after a path in $\Delta(G, \mathscr{S}_B^M(e, e', d))$ is discovered, the algorithm not only checks if it satisfies $\mathscr{S}_B^M(e, e', d)$, but also checks if there is no violable node for e and e' in its insulating segments when G is loop-unlimited. As 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.

3.4. Complexity of algorithms

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

DOI: 10.1002/stvr

```
check if G is loop-unlimited;
if yes then loop_unlimited :=true else loop_unlimited :=false;
currentpath := ⟨⊤⟩;
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 ρ corresponding to the concatenation of
                    currentpath and node is in Δ(G, S_B^M(e, e', d))
              then begin check if ρ satisfies S_B^M(e, e', d);
                              if no then return false;
                              if 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
                    currentpath and node is a prefix for Δ(G, S_B^M(e, e', d))
              then append node to currentpath;
         end
until currentpath = ⟨⟩;
if loop_unlimited then return true else return undecided.
```

Figure 8. Algorithm for maximal bounded delay analysis.

Let $G=(U, N, succ, ref, T)$ be an SBS. The numbers of the prefixes for the path sets $\Delta(G, v)$, $\Delta(G, \mathcal{S}_C(\varrho, \zeta))$, $\Delta(G, \mathcal{S}_B^m(e, e', d))$ and $\Delta(G, \mathcal{S}_B^M(e, e', d))$ are not greater than $|N|!$, $(|N|!)^3$, $(|N|!)^3$ and $(|N|!)^3$, respectively, and the sizes of the longest prefixes for the path sets $\Delta(G, v)$, $\Delta(G, \mathcal{S}_C(\varrho, \zeta))$, $\Delta(G, \mathcal{S}_B^m(e, e', d))$ and $\Delta(G, \mathcal{S}_B^M(e, e', d))$ 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 $\Delta(G, v)$, $\Delta(G, \mathcal{S}_C(\varrho, \zeta))$, $\Delta(G, \mathcal{S}_B^m(e, e', d))$ or $\Delta(G, \mathcal{S}_B^M(e, e', d))$, and to the size of the longest prefix.

As in the algorithms, for each path in the set $\Delta(G, v)$, $\Delta(G, \mathcal{S}_C(\varrho, \zeta))$, $\Delta(G, \mathcal{S}_B^m(e, e', d))$ or $\Delta(G, \mathcal{S}_B^M(e, e', d))$, at most one linear program needs to be solved, the number of linear programs to be solved is proportional to the number of the paths in the set $\Delta(G, v)$, $\Delta(G, \mathcal{S}_C(\varrho, \zeta))$, $\Delta(G, \mathcal{S}_B^m(e, e', d))$ or $\Delta(G, \mathcal{S}_B^M(e, e', d))$. As one event corresponds to one variable in the linear programs, the size of the linear programs to be solved 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 presented above 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 in 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, which thus supports the approach presented above to be efficient for problems in practical use.

3.5. *Analysis tool prototype*

The approach presented above has been implemented into a tool prototype TASS [22, 23] 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 Topcased [24], which is an open-source toolkit based on Eclipse platform [25]. The linear programming software package which is integrated in the tool is from OR-Objects of DRA Systems [26], which is a free collection of Java classes for developing operations research, scientific and engineering applications.

DOI: 10.1002/stvr

On a Pentium M/1.50 GHz/512 MB PC, TASS runs comfortably for checking several SBSs with more than 30 SDs in a few seconds. The case studies include the ATM system [7] and the global system for mobile communication (GSM) [27], and their details are given in the TASS website [22].

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

- *Reachability analysis*: To check if the node SD DispenseCash is reachable in the specification.
- *Constraint conformance analysis*: As a customer may lose his patience after he gets the money, it is required that the time that the ATM takes for the printing and bookkeeping after giving the money be not greater than 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 $\mathscr{S}_C(\varrho, \zeta)$ where $\varrho =$ Withdraw $\rightarrow$ DispenseCash and $\zeta = 2(\varpi_j - j_8) \leq j_c - h_5$.
- *Bound delay analysis*: As for the security consideration it is necessary to record the process for withdrawing money by the camera embedded in the ATM, every process for withdrawing money is required to take enough time for recording, which forms a minimal bounded delay specification $\mathscr{S}^m_B(j_4, a_1, 20)$.

Given the various values of the parameters W, B_1, B_2, T_1 and T_2 in the specification, the tool reports the corresponding sample results, which are depicted in Table I.

3.6. Discussion on approach generalization

The timing analysis algorithms presented above are a decision procedure for the loop-unlimited SBSs. The reason is that both the loop-closed condition and the loop-unlimited condition are held, which ensure that any loop in any path is time-independent of the other parts in the path. Based on these two conditions, the timing analysis problems on an infinite path can be reduced into the ones on a finite path by removing all the loop occurrences in the infinite path (see the theorem proofs in Appendix B). However, if the loop-closed condition and loop-unlimited condition are not satisfied, e.g. there are reverse constraints in an SBS which violates the loop-closed condition, the reduction of timing analysis problems on an infinite path is not feasible. The reason is that an infinite path in an SBS is formed by repeating loops infinite times, and the reverse constraints make it that the infinitely many loop occurrences are time dependent on each other (as illustrated by case (4) in Figure 2) so that those loop occurrences cannot be removed without influencing the other parts in the path. That is why those timing analysis algorithms are only a semi-decision procedure for general SBSs.

The timing analysis algorithms presented above are based on the synchronous composition semantics of SBSs. 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].

Table I. Sample results of timing analysis of the ATM specification.

	Problem					
	Reachability		$\mathscr{S}_C(\varrho, \zeta)$		$\mathscr{S}^m_B(j_4, a_1, 20)$	
Case	Result	Time	Result	Time	Result	Time
$w=4,\ B_1=0,\ B_2=\infty,\ T_1=0.5,\ T_2=2$	No	3.226 s	*	*	*	*
$w=6,\ B_1=0.5,\ B_2=1,\ T_1=0.5,\ T_2=2$	Yes	63 ms	Yes	3.760 s	No	96 ms
$w=6,\ B_1=1,\ B_2=2,\ T_1=0.5,\ T_2=2$	Yes	47 ms	Yes	3.603s	No	78 ms
$w=9,\ B_1=2,\ B_2=3,\ T_1=1,\ T_2=3$	Yes	47 ms	Yes	3.526 s	Yes	3.588 s

*The verification problem disappears because the node is unreachable.

DOI: 10.1002/stvr

Definition 6
Let $G=(U, N, succ, ref, T)$ be an SBS, and $D_1=(O_1, E_1, M_1, L_1, V_1, C_1)$ and $D_2=(O_2, E_2, M_2, L_2, V_2, C_2)$ be SDs in U $(E_1 \cap E_2=\emptyset)$. The asynchronous concatenation of D_1 and D_2, denoted by $D_1 \circ D_2$, is an SD $D=(O, E, M, L, V, C)$ which is defined by

- $O=O_1 \cup O_2$, $E=E_1 \cup E_2$, $M=M_1 \cup M_2$.
- 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$\}$

$$V_4=\left\{(e_1, e_2) \;\middle|\; \begin{array}{l} e_1 \in E_1, e_2 \in E_2, L(e_1)=L(e_2) \\ e_1, e_2 \text{ are reveiving events whose corresponding} \\ \text{sending events appear on mutual object column} \end{array}\right\}.$$

- $C=C_1 \cup C_2 \cup C_3 \cup C_4$ where $C_3=\{a \le e-e' \le b \mid a \le e-e' \le b \in T, e' \in E_1, e \in E_2\}$ and $C_4=\{\varepsilon_2-\varpi_1 \le 0 \mid \varepsilon_2$ is the start event of D_2, ϖ_1 is the end event of $D_1\}$.

According to the above definition, every path in an SBS corresponds to an SD, and thus the behaviour of an SBS is interpreted by the behaviour 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 reduction of the timing analysis problems cannot be implemented by simply removing all the loop occurrences in the path as shown in the theorem proofs in Appendix B. The reason is that the asynchronous composition semantics makes that all SDs in the path overlap in time with each other. Therefore, the algorithms presented above are simply generalized as a semi-decision procedure for timing analysis of an SBS G with the asynchronous composition semantics, which investigate all the paths in the sets $\Delta(G, v)$, $\Delta(G, \mathscr{S}_C(\varrho, \zeta))$, $\Delta(G, \mathscr{S}_B^m(e, e', d))$ and $\Delta(G, \mathscr{S}_B^M(e, e', d))$, and have been implemented into TASS.

Drawing lessons from *bound model checking* [28], which is to search for a counterexample in the model executions whose length is bounded by some integer k, the approach presented above can be generalized to perform *bounded* timing analysis of SBSs. As the timing analysis problems for a finite path in an SBS can be solved by linear programming, an algorithm can be developed to 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. This bounded timing analysis function has been supported by TASS.

4. RELATED WORK AND CONCLUSION

This paper presents a linear programming-based approach to timing analysis of SBSs expressed by UML interaction models. With more general and expressive timing constraints in UML SDs, the algorithms in the approach solve the problems of the reachability, constraint conformance and bounded delay analysis of SBSs. These algorithms form a decision procedure for the loop-unlimited SBSs where any loop in any path is time-independent of the other parts in the path, and a semi-decision procedure for general SBSs.

To the authors' knowledge, all the literature on timing analysis of SBSs are only about timing consistency. By using temporal constraint network techniques, Alur *et al.* [6] reduce the problem of checking basic MSCs with delay intervals for timing consistency into computing negative cost cycles and shortest distances in a weighted directed graph. The same techniques are used by Seemann and von Gudenberg [8] for timing consistency analysis of a class of UML SDs in which all timing constraints are of the form $a \le e_1-e_2 \le b$. The problem of checking MSC specifications for timing consistency is considered by Ben-Abdallah and Leue [7], which checks if every execution scenario described by an MSC specification is timing consistent. But in that work only a sufficient condition for timing consistency is given, which is not enough to develop an algorithm to analyse

DOI: 10.1002/stvr

MSC specifications for timing consistency. Li and Lilius [17] give a complete solution for checking the compositions of UML SDs for timing consistency. They also solve the problem of checking compositions of UML SDs for *timing inconsistency* [18], which checks if any execution scenario described by a composition of UML SDs is timing consistent.

Ladin and Leue [29] have interpreted MSC specifications as global state automata. Theoretically, the problems considered in this paper can thus be solved by transforming the SBSs into timed automata [30], 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 has been considered by Courcoubetis and Yannakakis [19] for timed automata, but the algorithm complexity is very high, and to the authors' knowledge no tool has been implemented so far. A case study is investigated by Firley *et al.* [9] for UML SD-based verification in which a simple SD is transformed in a set of timed automata and then checked for a timed automata-based implementation by a model checking [31] tool for timed automata. However, as shown in the case study, a lot of clocks are introduced for describing timing constraints enforced on SDs, which results in an exponential increment on the complexity of the algorithms for checking timed automata.

Compared to the timed automata-based approach, the advantage of the approach presented in this paper includes two aspects. On the one hand, the approach in this paper analyses directly SBSs themselves by investigating only the node state spaces of SBSs 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. It thus avoids the generation of the state space altogether and also the involved complexity. On the other hand, the approach in this paper considers more general and expressive timing constraints, which can be used to describe the relations among multiple time separations between events. It is well known that for a clock constraint in a timed automaton, its corresponding timing constraint is just related to the time separation between two events. For describing timing constraints about the relations among multiple time separations between events, it is necessary to compare multiple clocks in a timed automaton, which will result in the model checking problems that are undecidable [30].

This paper is mainly focused on timing analysis of SBSs with the synchronous composition semantics. For SBSs with the asynchronous composition semantics, the timing analysis problems are more difficult. Alur and Yannakakis [10] have shown that the corresponding model checking problem of MSC specifications for temporal requirements is undecidable. This paper only gives a simple discussion on the problems, and a more deep and complete investigation is definitely necessary.

APPENDIX A: CHECKING IF AN SBS IS LOOP-UNLIMITED

Let $G=(U, N, succ, ref, T)$ be an SBS. According to the definition of loop-unlimited condition, for checking if the loop-unlimited conditions are held for G, it is needed to traverse all the path segments in G of the form

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

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 conditions are held for G, it is needed to traverse all the path segments in G of the form

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

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

DOI: 10.1002/stvr

```
currentpath := ⟨⊤⟩; loopset := ∅;
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 node is in currentpath /*a loop is discovered*/
            then begin check if there is a reverse constraint for the loop;
                        if yes then return false else put the loop into loopset
                 end;
            else append node to currentpath;
         end
until currentpath = ⟨⟩;
```

```
currentpath := ⟨⊤⟩;
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 ρ corresponding to currentpath
                is a checked path segment for loop-unlimited condition
            then begin check if the loop-unlimited condition is satisfied;
                        if no then return false;
                 end
            if the path segment corresponding to the concatenation of
                currentpath and node is a prefix for loop-unlimited condition
            then append node to currentpath;
         end
until currentpath = ⟨⟩; return true.
```

```
currentpath := ⟨⊤⟩;
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 ρ corresponding to currentpath
                is a checked path segment for loop-closed condition
            then begin check if the loop-closed condition is satisfied;
                        if no then return false;
                 end;
            if the path segment corresponding to the concatenation of
                currentpath and node is a prefix for loop-closed condition
            then append node to currentpath;
         end
until currentpath = ⟨⟩; return true.
```

Figure A1. Algorithm to check if an SBS is loop-unlimited.

segment ρ 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 ρ_1 such that $\rho \rightarrow \rho_1$ becomes a checked path segment for loop-unlimited condition (loop-closed condition).

The following presents an algorithm to check if an SBS G is loop-unlimited (cf. Figure A1). The algorithm is based on the 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 the algorithm finds out all loops in G, and checks if there is any reverse constraint for any loop in G. Then it traverses all the checked path segments for loop-unlimited condition in G to check if the loop-unlimited condition is satisfied. Finally, the algorithm traverses all the checked path segments for loop-closed condition in G to check if the loop-closed condition is held. 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.

DOI: 10.1002/stvr

APPENDIX B: PROOFS OF THEOREMS

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

Proof
It is clear that one-half of the claim holds: if there is $\rho \in \Delta(G, v)$ such that $\mathscr{L}(\rho) \neq \emptyset$, then v is reachable. The other half of the claim can be proved as follows. Suppose v is reachable. Then there is a path ρ of the form $v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_i \rightarrow \cdots \rightarrow v_m$ $(0 \leq i \leq m)$ such that $v_i = v$, and a timed event sequence $\sigma \in \mathscr{L}(\rho)$ of the form $\sigma_0 \rightarrow \sigma_1 \rightarrow \cdots \rightarrow \sigma_i \rightarrow \cdots \rightarrow \sigma_m$ where each σ_j $(0 \leq j \leq m) \in \mathscr{L}(ref(v_j))$. The following proves that there is a path $\rho' \in \Delta(G, v)$ such that $\mathscr{L}(\rho') \neq \emptyset$, 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 the claim holds. Otherwise, there are v_p and v_q $(0 \leq p < q \leq i)$ such that $v_p \rightarrow v_{p+1} \rightarrow \cdots \rightarrow v_q$ is loop $(v_p = v_q)$, and/or there are $v_{p'}$ and $v_{q'}$ $(i \leq p' < q' \leq m)$ such that the subsequence $v_{p'} \rightarrow v_{p+1'} \rightarrow \cdots \rightarrow v_{q'}$ consists of multiple occurrences of a loop ($v_{p'}$ is the loop-start node, and $v_{p'} = v_{q'}$). As G satisfies the loop-closed condition and the loop-unlimited condition, any timing constraint does not combine any two nodes that are inside and outside a loop, respectively, and is not enforced on the repetition of any loop, which indicates that the loop $v_p \rightarrow v_{p+1} \rightarrow \cdots \rightarrow v_q$ and $v_{p'} \rightarrow v_{p+1'} \rightarrow \cdots \rightarrow v_{q'}$ are time-independent of the other parts in ρ. It follows that by removing the subsequences $\sigma_p \rightarrow \sigma_{p+1} \rightarrow \cdots \rightarrow \sigma_{q-1}$ and/or $\sigma_{p+1'} \rightarrow \sigma_{p+2'} \rightarrow \cdots \rightarrow \sigma_{q'}$ from σ, a timed event sequence σ_R which is a behaviour of G can be obtained, and by removing the subsequences $v_p \rightarrow v_{p+1} \rightarrow \cdots \rightarrow v_{q-1}$ and $v_{p+1'} \rightarrow v_{p+2'} \rightarrow \cdots \rightarrow v_{q'}$ from ρ, a path ρ_R such that $\sigma_R \in \mathscr{L}(\rho_R)$ can be obtained. By applying the above step repeatedly, a path ρ' can be constructed from ρ, which is of the form $v'_1 \rightarrow v'_2 \rightarrow \cdots \rightarrow v'_j \rightarrow v_i \rightarrow v'_{j+1} \cdots \rightarrow v'_k$ such that $\mathscr{L}(\rho') \neq \emptyset$, 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 ρ' is in $\Delta(G, v)$, from which the claim holds. □

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

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

$$v_0 \rightarrow v_1 \rightarrow \cdots \rightarrow v_{i-1} \rightarrow v_i \rightarrow v_{i+1} \rightarrow \cdots \rightarrow v_{i+k} \rightarrow v_{i+k+1} \rightarrow \cdots \rightarrow v_m$$

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

$$\sigma_0 \rightarrow \sigma_1 \rightarrow \cdots \rightarrow \sigma_{i-1} \rightarrow \sigma_i \rightarrow \sigma_{i+1} \rightarrow \cdots \rightarrow \sigma_{i+k} \rightarrow \sigma_{i+k+1} \rightarrow \cdots \rightarrow \sigma_m$$

where $\sigma_j \in \mathscr{L}(ref(v_j))$ for any j $(0 \leq j \leq m)$, and ζ is not satisfied by $v_i \rightarrow v_{i+1} \rightarrow \cdots \rightarrow v_{i+k}$. The following shows that a path $\rho' \in \Delta(G, \mathscr{S}_C(\varrho, \zeta))$ can be constructed such that there is a timed event sequence $\sigma' \in \mathscr{L}(\rho')$ which contains $\sigma_i \rightarrow \sigma_{i+1} \rightarrow \cdots \rightarrow \sigma_{i+k}$, which results in a contradiction and implies that the claim holds. As $\rho \notin \Delta(S, \mathscr{S}_C(\varrho, \zeta))$,

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

As G satisfies the loop-closed condition and the loop-unlimited condition, any timing constraint does not combine any two nodes that are inside and outside a loop, respectively, and is not enforced on the repetition of any loop, which indicates that the loop $v_p \rightarrow v_{p+1} \rightarrow \cdots \rightarrow v_q$ and

DOI: 10.1002/stvr

$v_{p'} \to v_{p+1'} \to \cdots \to v_{q'}$ are time-independent of the other parts in ρ. It follows that by removing the subsequences $\sigma_p \to \sigma_{p+1} \to \cdots \to \sigma_{q-1}$ and $\sigma_{p+1'} \to \sigma_{p+2'} \to \ldots \to \sigma_{q'}$ from σ, a timed event sequence σ_R which is a behaviour of G can be obtained; and by removing the subsequences $v_p \to v_{p+1} \to \cdots \to v_{q-1}$ and $v_{p+1'} \to v_{p+2'} \to \cdots \to v_{q'}$ from ρ, a path ρ_R such that $\sigma_R \in \mathscr{L}(\rho_R)$ can be obtained. By applying the above step repeatedly, a path $\rho' \in \Delta(G, \mathscr{S}_C(\varrho, \zeta))$ can be constructed from ρ such that there is $\sigma' \in \mathscr{L}(\rho')$ not satisfying ζ, which results in a contradiction and implies that the claim holds. □

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

Proof
It is clear that one-half of the claim holds: if G satisfies $\mathscr{S}_B^m(e, e', d)$, then any path in $\Delta(G, \mathscr{S}_B^m(e, e', d))$ satisfies $\mathscr{S}_B^m(e, e', d)$. The other-half of the claim can be proved as follows. Suppose that any path in $\Delta(G, \mathscr{S}_B^m(e, e', d))$ satisfies $\mathscr{S}_B^m(e, e', d)$, and there is a path ρ of G such that there is a behaviour σ of G in $\mathscr{L}(\rho)$, which does not satisfy $\mathscr{S}_B^m(e, e', d)$. Without losing generality, suppose that ρ is of the form $v_0 \to v_1 \to \cdots \to v_i \to v_{i+1} \to \cdots \to v_j \to v_{j+1} \to \cdots \to v_m$ where e' occurs in $ref(v_i)$, e occurs in $ref(v_j)$ $(0 \le i < j \le m)$, and e and e' do not occur in any $ref(v_k)$ $(i < k < j)$; σ is of the form $\sigma_0 \to \sigma_1 \to \cdots \to \sigma_i \to \sigma_{i+1} \to \cdots \to \sigma_j \to \sigma_{j+1} \to \cdots \to \sigma_m$, where $\sigma_k \in \mathscr{L}(v_k)$ for any k $(0 \le k \le m)$; and the separation in time between e occurring in σ_j and e' occurring in σ_i is smaller than d. The following shows that a path $\rho' \in \Delta(G, \mathscr{S}_B^m(e, e', d))$ can be constructed such that there is a timed event sequence $\sigma' \in \mathscr{L}(\rho')$ which does not satisfy $\mathscr{S}_B^m(e, e', d)$, which results in a contradiction and implies that the claim holds. As $\rho \notin \Delta(G, \mathscr{S}_B^m(e, e', d))$,

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

As G satisfies the loop-closed condition and the loop-unlimited condition, any timing constraint does not combine any two nodes that are inside and outside a loop, respectively, and is free for the repetition of any loop, which indicates that the loop $v_p \to v_{p+1} \to \cdots \to v_q$, $v_{p'} \to v_{p+1'} \to \cdots \to v_{q'}$ and $v_{p''} \to v_{p+1''} \to \cdots \to v_{q''}$ are time-independent of the other parts in ρ. It follows that by removing the subsequences $\sigma_p \to \sigma_{p+1} \to \cdots \to \sigma_{q-1}$, $\sigma_{p+1'} \to \sigma_{p+2'} \to \cdots \to \sigma_{q'}$ and $\sigma_{p+1''} \to \sigma_{p+2''} \to \cdots \to \sigma_{q''}$ from σ, a timed event sequence σ_R can be obtained, which is a behaviour of G and such that in σ_R the time separation between e occurring in σ_j and e' occurring in σ_i is smaller than d; and by removing the subsequences $v_p \to v_{p+1} \to \cdots \to v_{q-1}$, $v_{p+1'} \to v_{p+2'} \to \cdots \to v_{q'}$ and $v_{p+1''} \to v_{p+2''} \to \cdots \to v_{q''}$ from ρ, a path ρ_R such that $\sigma_R \in \mathscr{L}(\rho_R)$ can be obtained. By applying the above step repeatedly, a path $\rho' \in \Delta(G, \mathscr{S}_B^m(e, e', d))$ can be constructed from ρ such that there is a timed event sequence $\sigma' \in \mathscr{L}(\rho')$ which does not satisfy $\mathscr{S}_B^m(e, e', d)$, which results in a contradiction and implies that the claim holds. □

Theorem 4
Let G be a loop-unlimited SBS, and $\mathscr{S}_B^M(e, e', d)$ be a maximal bounded delay specification. Then, G satisfies $\mathscr{S}_B^M(e, e', d)$ if and only if for any path ρ in $\Delta(G, \mathscr{S}_B^M(e, e', d))$ $(\mathscr{L}(\rho) \neq \emptyset)$, it satisfies $\mathscr{S}_B^M(e, e', d)$ and there is no 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 $\mathscr{S}_B^M(e, e', d)$, then for any path ρ in $\Delta(G, \mathscr{S}_B^M(e, e', d))$ $(\mathscr{L}(\rho) \neq \emptyset)$, it satisfies $\mathscr{S}_B^M(e, e', d)$ and there is no node in its insulating segments which is violable for e and e'. The reason is that if there is a path $\rho \in \Delta(G, \mathscr{S}_B^M(e, e', d))$ $(\mathscr{L}(\rho) \neq \emptyset)$ 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 $\mathscr{L}(\rho)$, a behaviour of G that does not satisfy $\mathscr{S}_B^M(e, e', d)$ can be constructed by repeating the positive loop in $\Theta(G, v, e, e')$ with finite times. The other half of the

DOI: 10.1002/stvr

claim can be proved as follows. Suppose that for any path ρ in $\Delta(G, \mathscr{S}_B^M(e,e',d))$ $(\mathscr{L}(\rho)\neq\emptyset)$, it satisfies $\mathscr{S}_B^M(e,e',d)$ and there is no node in its insulating segments which is violable for e and e', and that there is a path ρ' in G such that $\sigma'\in\mathscr{L}(\rho')$ does not satisfy $\mathscr{S}_B^M(e,e',d)$. Without losing generality, suppose that ρ' is of the form $v_0\to v_1\to\cdots\to v_i\to v_{i+1}\to\cdots\to v_j\to v_{j+1}\to\cdots\to v_m$ where e' occurs in $ref(v_i)$, e occurs in $ref(v_j)$ $(0\leq i<j\leq m)$, and e and e' do not occur in any $ref(v_k)$ $(i<k<j)$; σ' is of the form

$$\sigma_0\to\sigma_1\to\cdots\to\sigma_i\to\sigma_{i+1}\to\cdots\to\sigma_j\to\sigma_{j+1}\to\cdots\to\sigma_m$$

where $\sigma_k\in\mathscr{L}(v_k)$ for any k $(0\leq k\leq m)$; and the separation in time between e occurring in σ_j and e' occurring in σ_i is greater than d. The following shows that a path $\rho''\in\Delta(S,\mathscr{S}_B^M(e,e',d))$ can be constructed such that either there is $\sigma''\in\mathscr{L}(\rho'')$ which does not satisfy $\mathscr{S}_B^M(e,e',d)$ or there is a violable node for e and e' in the insulating segment of ρ'', which results in a contradiction and implies that the claim holds. As $\rho'\notin\Delta(G,\mathscr{S}_B^M(e,e',d))$,

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

As G satisfies the loop-closed condition and the loop-unlimited condition, any timing constraint does not combine any two nodes that are inside and outside a loop, respectively, and is free for the repetition of any loop, which indicates that the loop $v_p\to v_{p+1}\to\cdots\to v_q$, $v_{p'}\to v_{p+1'}\to\cdots\to v_{q'}$ and $v_{p''}\to v_{p+1''}\to\cdots\to v_{q''}$ are time-independent of the other parts in ρ'. It follows that if the loop $v_{p'}\to v_{p+1'}\to\cdots\to v_{q'}$ is not positive, then by removing the subsequences $\sigma_p\to\sigma_{p+1}\to\cdots\to\sigma_{q-1}$, $\sigma_{p+1'}\to\sigma_{p+2'}\to\cdots\to\sigma_{q'}$ and $\sigma_{p+1''}\to\sigma_{p+2''}\to\cdots\to\sigma_{q''}$ from σ', a timed event sequence σ'_R can be obtained, which is a behaviour of G and such that in σ'_R the time separation between e occurring in σ_j and e' occurring in σ_i is greater than d; and by removing the subsequences $v_p\to v_{p+1}\to\cdots\to v_{q-1}$, $v_{p+1'}\to v_{p+2'}\to\cdots\to v_{q'}$ and $v_{p+1''}\to v_{p+2''}\to\cdots\to v_{q''}$ from ρ', a path ρ'_R such that $\sigma'_R\in\mathscr{L}(\rho'_R)$ can be obtained. By applying the above step repeatedly, a path $\rho''\in\Delta(S,\mathscr{S}_B^M(e,e',d))$ can be obtained from ρ' such that either there is $\sigma''\in\mathscr{L}(\rho'')$, which does not satisfy $\mathscr{S}_B^M(e,e',d)$ or there is a violable node for e and e' in the insulating segment of ρ'', which results in a contradiction and implies that the claim holds. □

ACKNOWLEDGEMENTS

Thanks go to the anonymous reviewers for their valuable comments and suggestions. This work is supported by the National Natural Science Foundation of China (No. 90818022, No. 60721002), the National 863 High-Tech Programme of China (No. 2009AA01Z148, No. 2007AA010302) and by the National Grand Fundamental Research 973 Program of China (No. 2009CB320702).

REFERENCES

1. ITU-T. Recommendation z.120. Message sequence charts. International Telecommunication Union—Standardization Sector, Genève, Switzerland, 2000.
2. Rumbaugh J, Jacobson I, Booch G (eds.). *The Unified Modeling Language Reference Manual*. Addison-Wesley Longman Ltd: Essex, U.K., 1999.
3. OMG. UML2.0 Superstructure Specification. Available at: http://www.uml.org [10 October 2005].
4. Heitmeyer CL, Jeffords RD, Labaw BG. Comparing different approaches for specifying and verifying real-time systems. *IEEE Real-Time System Newsletter* 1993; **9**(1–2):122–129.
5. Kluge O. Modelling a railway crossing with message sequence charts and petri nets. *Petri Technology for Communication-Based Systems—Advance in Petri Nets* (*Lecture Notes in Computer Science*, vol. 2472), Ehrig H, Reisig W, Rozenberg G, Weber H (eds.). Springer: Berlin, 2003; 197–218.
6. Alur R, Holzmann GJ, Peled D. An analyzer for message sequence charts. *Software—Concepts and Tools*. Springer: Berlin, 1996; **17**:70–77.
7. Ben-Abdallah H, Leue S. Timing constraints in message sequence chart specifications. *FORTE X/PSTV XVII '97: Proceedings of the IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques*

DOI: 10.1002/stvr

for Distributed Systems and Communication Protocols (*FORTE X*) *and Protocol Specification, Testing and Verification* (*PSTV XVII*). Chapman & Hall: London, U.K., 1998; 91–106.
8. Seemann J, von Gudenberg JW. Extension of UML sequence diagrams for real-time systems. *UML '98*: *Selected Papers from the First International Workshop on The Unified Modeling Language*. Springer: London, U.K., 1999; 240–252.
9. Firley T, Huhn M, Diethers K, Gehrke T, Goltz U, Braunschweig TU. Timed sequence diagrams and tool-based analysis—A case study. *UML '99*: *Proceedings of the Second International Conference on UML* (*Lecture Notes in Computer Science*, vol. 1732). Springer: Berlin, 1999; 645–660.
10. Alur R, Yannakakis M. Model checking of message sequence charts. *CONCUR '99*: *Proceedings of the 10th International Conference on Concurrency Theory*. Springer: London, U.K., 1999; 114–129.
11. Holzmann GJ. Early fault detection tools. *TACAS '96*: *Proceedings of the Second International Workshop on Tools and Algorithms for Construction and Analysis of Systems*. Springer: London, U.K., 1996; 1–13.
12. Holzmann GJ. *Design and Validation of Computer Protocols*. Prentice-Hall: Upper Saddle River, NJ, U.S.A., 1991.
13. Holzmann GJ. *The SPIN Model Checker*: *Primer and Reference Manual*. Addison-Wesley Professional: Reading, MA, 2003.
14. Levin V, Peled D. Verification of message sequence charts via template matching. *TAPSOFT '97*: *Proceedings of the Seventh International Joint Conference CAAP/FASE on Theory and Practice of Software Development*. Springer: London, U.K., 1997; 652–666.
15. Muscholl A, Peled D, Su Z. Deciding properties for message sequence charts. *FoSSaCS '98*: *Proceedings of the First International Conference on Foundations of Software Science and Computation Structure*. Springer: London, U.K., 1998; 226–242.
16. Akshay S, Mukund M, Kumar KN. Checking coverage for infinite collections of timed scenarios. *CONCUR '07*: *Proceedings of International Conference on Concurrency Theory* (*Lecture Notes in Computer Science*, vol. 4703). Springer: Berlin, 2007; 181–196.
17. Li X, Lilius J. Timing analysis of UML sequence diagrams. *UML '99*: *Proceedings of the Second International Conference on UML* (*Lecture Notes in Computer Science*, vol. 1732). Springer: Berlin, 1999; 661–674.
18. Li X, Lilius J. Checking compositions of UML sequence diagrams for timing inconsistency. *APSEC '00*: *Proceedings of the Seventh Asia-Pacific Software Engineering Conference*. IEEE Computer Society: Washington, DC, U.S.A., 2000; 154–161.
19. Courcoubetis C, Yannakakis M. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design* 1992; **1**(4):385–415.
20. Hulgaard H, Burns SM. Bounded delay timing analysis of a class of csp programs. *Formal Methods in System Design* 1997; **11**(3):265–294.
21. Peled D. *Software Reliability Methods*. Springer: Berlin, 2001.
22. TASS: Timing Analysis of Scenario-Based Specifications. Available at: http://cs.nju.edu.cn/lxd/TASS/index.htm [10 August 2009].
23. Pan M, Bu L, Li X. TASS: Timing analyzer of scenario-based specifications. *CAV'2009*: *Proceedings of the 21th International Conference on Computer Aided Verification* (*Lecture Notes in Computer Science*, vol. 5643). Springer: Berlin, 2009; 689–695.
24. Topcased. Available at: http://www.topcased.org/ [21 July 2009].
25. Eclipse—An open development platform. Available at: http://www.eclipse.org/ [21 July 2009].
26. OR-Objects. Available at: http://opsresearch.com/OR-Objects/index.html [28 March 2008].
27. Telecom Message Sequence Charts. Available at: http://www.eventhelix.com/EventStudio/telecom_message_sequence_charts/ [30 May 2009].
28. Biere A, Cimatti A, Clarke EM, Strichman O, Zhu Y. Bounded model checking. *Advance in Computers* 2003; **58**:118–149.
29. Ladkin P, Leue S. Interpreting message sequence charts (revised version). *Technical Report TR 101*, Department of Computing Science, University of Stirling, U.K., 1993.
30. Alur R, Dill DL. A theory of timed automata. *Theoretical Computer Science* 1994; **126**(2):183–235.
31. Clarke EM, Grumberg J, Peled D. *Model Checking*. MIT Press: Cambridge, MA, 2000.

DOI: 10.1002/stvr