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

Technical Report No. NJU-SEG-2024-IC-017

2024-IC-017

SAT and LP Collaborative Bounded Timing Analysis of Scenario-Based Specifications

Longlong Lu,Wenhua Yang,Minxue Pan,Tian Zhang

Technical Report 2024

SAT and LP Collaborative Bounded Timing Analysis of Scenario-Based Specifications

Longlong Lu
lull@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China

Wenhua Yang
ywh@nuaa.edu.cn
College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, China

Minxue Pan
mxp@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China

Tian Zhang
ztluck@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China

ABSTRACT

Timing analysis of scenario-based specifications (SBS) such as message sequence charts and UML interaction models plays an essential role in the design phase of real-time system development. However, it is time-consuming and labor-intensive to conduct analysis on the satisfiability of the timing constraints. In this article, we propose a novel SAT and linear programming (LP) collaborative timing analysis approach named TASSAT for SBS. Instead of using depth-first traversal algorithms, TASSAT encodes the structures of the SBS into propositional formulas and use the SAT solver to find candidate paths. The timing analysis of candidate paths is then reduced to LP problems, where irreducible infeasible set of the infeasible path can be used to prune unnecessary search space of the SAT solver. The experimental results show that TASSAT is effective and offers better performance than existing tools in terms of both time consumption and memory footprint.

KEYWORDS

Scenario-based specifications, UML interaction models, model checking, reachability analysis, SAT

ACM Reference Format:
Longlong Lu, Wenhua Yang, Minxue Pan, and Tian Zhang. 2020. SAT and LP Collaborative Bounded Timing Analysis of Scenario-Based Specifications. In *12th Asia-Pacific Symposium on Internetware (Internetware'20), May 12–14, 2021, Singapore, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3457913.3457917

1 INTRODUCTION

Real-time software plays a crucial role in many advanced embedded and cyber-physical systems such as avionics, automobile, and medical equipment. It has become the main driver and facilitator for innovation [16]. However, the development of real-time software is a non-trivial task: developers need to focus on not only the functional properties but also the timing requirements, as any timing violation may become a source of serious faults leading to the failure of the whole system. Timing constraints need to be identified and analyzed as soon as possible in the software development process, particularly in the design phase [23, 33].

Complex embedded and cyber-physical systems are made of several interacting components [11]. During the design phase, an intuitive way of specifying the desired interacting behaviors is by scenarios. Scenarios can help developers control the system complexity by decomposing the design into multiple fragments and focus on the crucial interactions. The language of Message Sequence Charts (MSCs) and its extensions are often used to express scenarios of such interactions. MSCs are especially useful for the end-users because of their clarity and graphical content [12], and are standardized by the ITU (International Telecommunication Union) [18]. To support the description of real-time systems, time concepts are introduced into MSC with a precise meaning of the sequence of events in time [18]. Various timing constraints, such as time points, measurements, and intervals can be specified to constrain the time at which events may occur. Because of the popularity and expressiveness of MSCs, in this paper, we choose timed MSCs as scenario-based specifications and study their timing analysis problems.

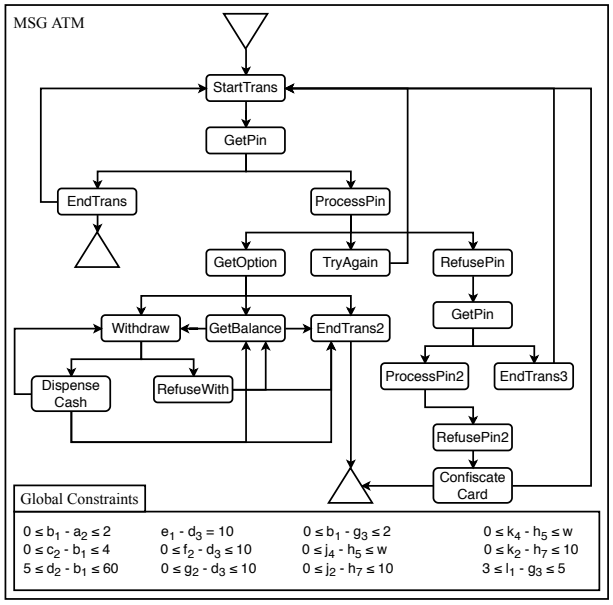
In an MSC specification, the system is often specified in a hierarchical fashion: simple scenarios are modelled by basic MSCs (bMSCs), and complex behavior is typically described using message sequence graphs (MSGs) [4]. An MSG is a finite directed graph with nodes labelled by bMSCs, and is the most basic form of a High-level Message Sequence Chart (HMSC) [18]. It describes the compositional relations of bMSCs. The composition of bMSCs can either be synchronous or asynchronous. Synchronous composition requires that one bMSC should only be started execution after all its precedent bMSCs have completed, on the contrary the asynchronous composition has no such restriction. Therefore, the later is more flexible and powerful in expressiveness and is recommended by ITU. However, this flexibility is gained at the cost of decidability: the model checking problem for asynchronous MSC specifications is very difficult; even for the untimed ones, it is undecidable [4]. This situation is aggravated when time is involved. Previous work [22, 26] shows that it requires more complex timing constraints than simple time intervals to specify real-time software's timing requirements. It is very often that one needs to specify the relations of multiple time intervals, such as one time interval is half of another. As is well known that to specify the relations of multiple

time intervals is equivalent to compare multiple clocks in timed automata, which is also an undecidable problem [1].

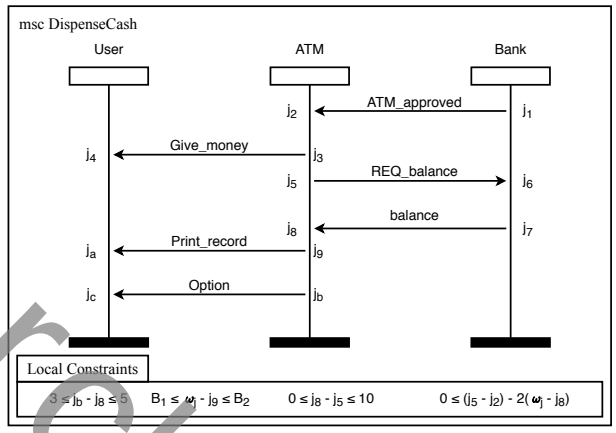
Nevertheless, in order to ensure the design's timing correctness, verifying MSC specification is a challenge that should not be avoided. Therefore, bounded model checking, which is a technique often used when an entire model checking is infeasible because of high complexity or even undecidability, is employed to analyze MSC specifications [9]. Specifically, first, the MSG is traversed in a depth-first manner to get an unchecked path within the given bound. Then the path through the graph forms a new bMSC by concatenating the bMSCs visited along the path, and is transformed to a set of linear constraints to be checked for timing analysis problems by linear programming (LP). The checking process terminates either a counterexample is witnessed, or all paths within the bound have been checked. Based on this checking algorithm, a tool named TASS is developed and released. However, its performance could be a problem when the bound is very large, because large path bounds cause a huge number of candidate paths that need to be checked. As shown in our experiments, for the MSC specifications containing 17 bMSCs, the tool consumes 187 seconds to complete a simple reachability analysis when the bound is set to 20 on a modern computer. This could discourage developers from using formal methods to check the timing correctness of designs. Clearly, a more effective checking approach is very much needed.

In this paper, we propose a novel SAT and LP collaborative bounded timing analysis approach for MSC specifications. Instead of using depth-first traversal to find paths, the transition relations in the graph structure of an MSG is encoded as a propositional formula set, which is fed to an SAT solver to get a truth assignment. Then, the truth assignment is decoded and mapped back to a path in the MSG. With the facilitation of state-of-the-art SAT solvers, this approach is more efficient compared with the existing approaches, especially when the bound is large. Furthermore, we exploit LP to accelerate the SAT-based path searching. We noticed that the existing approach only uses LP to decide whether a set of constraints are infeasible, whereas ignores the information about exactly which constraints cause the whole set infeasible. This information is crucial to the performance, as it can prune many paths containing the same infeasible constraints. In our approach, the infeasible constraints are mapped to path segments, then these segments are encoded as SAT expressions to guide the SAT-based path searching. This tightly integrated SAT and LP collaborative analysis approach has significant performance improvement and is **our first and main contribution**. We implemented our approach as a tool called TASSAT. It can check MSC specifications for bounded reachability analysis and also supports parsing graphical models, which forms **our second contribution**. Experiments on two practical scaled cases confirmed TASSAT's effectiveness and efficiency.

The rest of the paper is organized as follows: Section 2 introduces the MSC specifications. An example is employed for illustration, which is also used in the experiments. Section 3 presents the main contribution of this work: the SAT and LP collaborative bounded timing analysis approach of MSC specifications. Section 4 discusses the experimental evaluation. Section 5 compares related work, and the conclusion is drawn in Section 6.



(a) The MSG of the ATM



(b) The bMSC of the DispenseCash Scenario

Figure 1: The MSC specification of the ATM example

2 MSC SPECIFICATIONS

In MSC specifications, simple scenarios are depicted by bMSCs, whereas multiple scenarios and complete system specifications are depicted by an MSG. An MSG provides a portable means to graphically define how a set of bMSCs can be combined to describe potentially iterating and branching system behaviors [31]. Figure 1 shows the MSC specifications of an automatic teller machine (ATM) system example [22].

Figure 1 (b) shows the bMSC referred by the node `DispenseCash` in Figure 1 (a). In a bMSC, the vertical lines in the chart correspond to instances, and messages exchanged between those instances are represented by arrows. The sending and receiving of messages are corresponding to events respectively. There are two special events ϵ and ϖ which represent the start and the end of each bMSC, respectively. For specifying real-time software systems, we adopt a

general form of timing constraints [22] that can specify the relation of multiple time intervals, such as one time interval is half of another. We use event names to represent event occurrence time, and linear inequalities on event names to represent the timing constraints. A timing constraint is of the form $a \leqslant c_0(e_0 - e_0') + c_1(e_1 - e_1') + \cdots + c_n(e_n - e_n') \leqslant b$, where e_i and e_i' $(0 \leqslant i \leqslant n)$ are event names which represent the occurrence time of e_i and e_i'. Among the expressions, a, b and $c_0, c_1, \ldots, c_n$ are real numbers (b may be ∞). For example, the timing constraint $0 \leqslant (j_5 - j_2) - 2(\varpi_j - j_8)$ in Figure 1(b) specifies that the time that the ATM takes for the printing and book-keeping $(\varpi_j - j_8)$ should not be greater than half of the time that the ATM gives the money $(j_5 - j_2)$, in case that a customer may lose patience after having received the money.

The semantics of a bMSC essentially consist of the sequences (traces) of the message sending and 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 instance in the bMSCs, along with a causal dependency between the events of sending and receiving a message. In accordance with [21, 28], without losing generality, we assume that for a pair of events e and e' in a bMSC, 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 instance axis, and e' is a sending event.
- **FIFO Order:** The receiving event e appears above the receiving event e' on the same instance axis, and the corresponding sending events e_1 and e_1' appear on a mutual instance axis where e_1 is above e_1'.

Definition 2.1 (Basic Message Sequence Chart). A basic MSC is a tuple $B = (I, E, M, L, V, C)$ where

- I is a finite set of instances.
- E is a finite set of events corresponding to sending a message and receiving a message. There are two special events ϵ and ϖ in E which represent the start and end of B respectively.
- M is a finite set of messages whose elements are a pair (e, e') where $e, e' \in E$ are corresponding to the sending and the receiving for a message respectively.
- $L : E \to I$ is a labeling function which maps each event $e \in E$ to an instance $L(e) \in I$ which is the sender (receiver) while e corresponds to sending (receiving) a message.
- V is a finite set whose elements are a pair (e, e') $(e, e' \in E)$ such that $e \prec e'$.
- C is a finite set of timing constraints.

We use *timed event sequences* to represent the behavior of bMSCs. A timed event sequence is of the form $\sigma = (e_0, t_0) \to (e_1, t_1) \to \cdots \to (e_i, t_i) \to \cdots \to (e_m, t_m)$ where e_i is an event and t_i is a nonnegative real numbers for any i $(0 \leqslant i \leqslant m)$. The timed event sequence describes that e_1 takes place t_1 time units after e_0 takes place, then e_2 takes place t_2 time units after e_1 takes place, so on and so forth. At last $e_m = \varpi$ takes place t_m time units after e_{m-1} takes place. It follows that for any i $(0 \leqslant i \leqslant m)$, the occurrence time of e_i is $\sum_{j=0}^{i} t_j$.

The behavior of a bMSC is formally defined in 2.2 according to preceding definition. We adopt $X.Y$ to denote the element Y of a tuple X.

Definition 2.2 (Timed Event Sequence of bMSC). Let B be a bMSC, a timed event sequence $(e_0, t_0) \to (e_1, t_1) \to \cdots \to (e_m, t_m)$ is a behavior of B if and only if the following conditions hold:

- $e_0 = \epsilon$ and $e_m = \varpi$.
- $e_0, e_1, \ldots, e_m$ is a permutation of all the events of $B.E$.
- $e_0, e_1, \ldots, e_m$ satisfies the visual partial order of $B.V$, i.e., for any e_i and e_j, if $e_i \prec e_j$, then $0 \leqslant i < j \leqslant m$.
- $t_0, t_1, \ldots, t_m$ satisfy all the timing constraints of B, i.e. for any timing constraint $a \leqslant \sum_{i=0}^{n} c_i(f_i - f_i') \leqslant b$ in $B.C$, $a \leqslant c_0\delta_0 + c_1\delta_1 + \cdots + c_n\delta_n \leqslant b$ where for each i $(0 \leqslant i \leqslant n)$, 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 $\mathcal{L}(B)$ denotes the set of timed event sequences representing behaviors of B.

Figure 1(a) shows an exemplary MSG. There are three types of nodes in an MSG: the *start* node notated as an inverted triangle, the *end* node notated as a triangle, and the *intermediate* nodes notated with rectangles. Each intermediate node refers to a bMSC, and some of the intermediate nodes can refer to the same bMSC for scenario reuse, such as nodes `GetPin`, `RefusePin`, and `EndTrans`. MSGs also support the declaration of global timing constraints. A global timing constraint of the MSG is in the form of $a \leqslant e - e' \leqslant b$ where e and e' occur in different bMSCs referred by intermediate nodes and $0 \leqslant a \leqslant b$ (b may be ∞). Global timing constraints are used to describe the timed relation between two events in different bMSCs.

Definition 2.3 (Message Sequence Graph). An MSG M is a tuple $M = (N, n_S, N_E, \lambda, R, \phi)$, where

- N is the node set of the graph.
- n_S is the start node of the graph.
- N_E is the end node set of the graph, $\forall n_E \in N_E$, n_E is a legal end node of the graph.
- λ is a mapping function which maps each intermediate node to a bMSC, i.e., $\forall n \in (N - n_S - n_E)$, $\lambda(n)$ is a bMSC.
- R is the relation set of the graph whose elements are of the form (n_i, n_j), where $n_i, n_j \in N$ and $n_i \prec n_j$.
- Φ is a set of timing constraints of the form $a \leqslant e - e' \leqslant b$.

A *path* of an MSG is a node sequence formally defined by Definition 2.4.

Definition 2.4 (Path of MSG). Let M be an MSG, a node sequence $n_S \to n_1 \to \cdots \to n_i \to n_j \to \cdots \to n_k \to n_E$ is a path of M if and only if the following conditions hold:

- n_S and n_E are the start node and the end node of M respectively.
- For any adjacent nodes n_i and n_j in the path, then $(n_i, n_j) \in R$.

Let P be a path, the length of the path $|p|$ is the number of nodes in the path.

We interpret the timing constraints in MSG by *local semantics*: select one path at one time and analyze its timing requirements, independently of other paths that may branch out of the selected one. The timing constraints of a path is formally defined in Definition 2.5.

Definition 2.5 (Timing Constraint of MSG). Let $p = n_S \rightarrow \cdots \rightarrow n_j \rightarrow \cdots \rightarrow n_k \rightarrow \cdots \rightarrow n_E$ be a path of the MSG M, ψ is a timing constraint of p if one of the following conditions hold:

- for any n_i in p such that $n_i! = n_S$ and $n_i! = n_E$, $\psi \in \lambda(n_i).C$.
- for any $\phi = a \leq e - e' \leq b \in M.\Phi$ such that $e' \in \lambda(n_j).E$ and $e \in \lambda(n_k).E$ $(j < k)$, $\psi = \phi$.

Let Ψ denotes the set of timing constraints of the path p.

As advocated by the ITU Recommendation Z.120 [18], the concatenation of bMSCs is interpreted by *asynchronous* semantics. The asynchronous concatenation of two bMSCs corresponds to concatenating two bMSCs instance by instance, which produces a new bMSC. In this way, each path in an MSG corresponds to a bMSC. Additionally, the set of timing constraints of the new bMSC is equivalent to the set of timing constraints of the path. The behavior of an MSG can thus interpreted by the behaviors of bMSCs corresponding to its paths, which are defined by Definition 2.2.

3 SAT AND LP COLLABORATIVE BOUNDED TIMING ANALYSIS

3.1 Approach Overview

This section presents a SAT and LP collaborative bounded timing analysis approach for *reachability analysis* of MSC specifications. Reachability analysis is an important problem which a large number of timing consistent problems can be converted to. It is to check: (1) if a given target node of an MSG is in a path, i.e., given a node n_T and a path P, there exists a node $n_i \in P$ such that $n_i = n_T$, and (2) all the timing constraints related to the path and all the bMSC timing constraints in the path are consistent. Since all the timing constraints are linear equalities or linear inequalities, the timing analysis problems can be reduced to linear programming problems. Theoretically, to perform reachability analysis for a given node, all the paths in the MSG should be enumerated and checked. Due to loops on the graph, there could be infinite number of paths; so it is necessary to set a bound to limit the length of path $|P|$. When the path bound is large, the enumeration of paths with depth-first graph traversal algorithm could be slow, which also generates many candidate paths need to be checked. To address the problem mentioned above, we propose the SAT [25] and LP [30] collaborative approach, to accelerate the path enumeration process and prune unnecessary paths.

Figure 2 depicts the overview of our approach. The input of the approach is MSC specifications consisting of an MSG and a set of bMSCs. To find a path to check, firstly, the MSG with the bound is encoded to a propositional formula set in the conjunctive normal form (CNF) [10, 19, 29, 34], which is a conjunction of one or more disjunctive clauses. Then the formula set is fed to a SAT solver to find a solution that is the encoding of a feasible path in the MSG. If the formula set is unsatisfiable, then the analysis process concludes that there the target node is not reachable; otherwise, the SAT solution is decoded to a path. Secondly, the path-related constraints are fed to the off-the-shelf LP optimizer to check if there exists feasible region. If so, then TASSAT returns the analysis result, otherwise TASSAT computes the irreducible infeasible set (IIS) [27, 32], which is transformed into an infeasible path segment. Finally, the infeasible path segment is encoded to the negative SAT expression and combined with the original propositional formula set, and the approach goes back to the first step to find another path to be checked. After repeating this process until no more paths within the bound need to be checked, TASSAT gives the final analysis result.

3.2 SAT and LP Collaborative Algorithm

Algorithm 1 outlines how the SAT and LP collaborative approach performs the reachability analysis for scenario-based specifications. The algorithm takes the bound k, the MSG M, the target nodes n_T, and the bMSC set N as inputs, then outputs the reachability of the target nodes on the MSG. In this algorithm, the MSG structure, the bound, and the target node are encoded to SAT expression and stored into a local variable BM^K (Line 1), the detailed process of encoding will be illustrated in Section 3.3. Next, it judges if there exist a satisfiable truth assignment for the given expression (Line 3-4). If there is not such a satisfiable assignment, the algorithm returns *unreachable* (Line 5). Else, a candidate path decoded from the truth assignment is bounden to the variable ρ (Line 7). The path-related constraints are chosen from the whole timing constraint set (Line8). Next, if the path-related timing constraints are feasible, the algorithm returns *reachable* (Line 10-11). Else, the algorithm calculates IIS of the infeasible constraints and maps the constraints to the path segment of the infeasible path ρ (Line 13-14). The path segments are encoded to negative CNF clauses and combined with the original SAT expression BM^K (Line 15), then the algorithm goes back to line 3 until the reachability analysis terminates. The detail of LP function will be illustrated in Section 3.4.

3.3 SAT Encoding

Boolean satisfiability problem (SAT) is the problem of determining if there exists an interpretation that satisfies the given boolean formula set. Most SAT solvers accept formulas in conjunctive normal formal (CNF) which means the entire formula is a conjunction of the clauses, with each clause being a disjunction of variables. In our approach, we translate the path traversal problem of MSGs to SAT, which is why we can exploit the IIS information to assist the SAT solver in the subsequent process.

In this paper, we denote the propositional logic terminologies in the following way:

- $\wedge$, $\vee$, and $\neg$ are used to represent *Conjunction*, *Disjunction*, and *Negation* respectively, and $\rightarrow$ denotes *Implication*.
- Boolean variables and boolean expressions connected by logical connectives are *boolean formulas*. CNF is a special kind of boolean expressions which are friendly to modern SAT solvers.
- Implications are eliminated in this way, let x and y be boolean formulas, $x \rightarrow y$ can be converted to $\neg x \vee y$ according to the truth table.

To perform reachability analysis for the given MSC specifications, it is necessary to find a candidate path which contains the

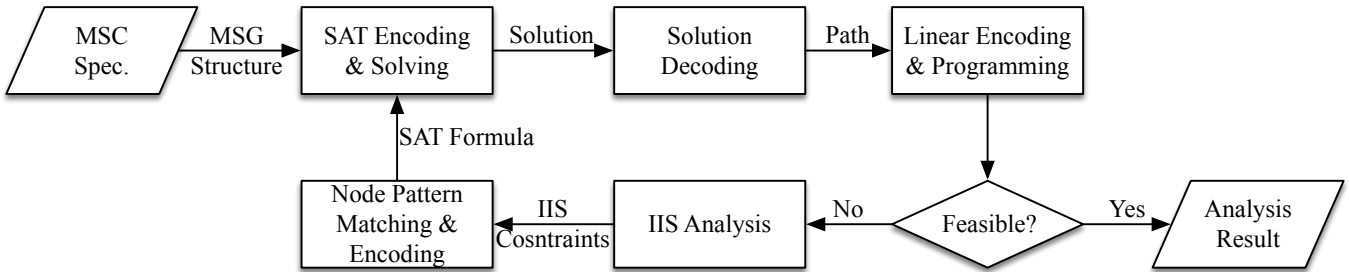


Figure 2: Overview of the SAT and LP Collaborative approach

Algorithm 1 SAT and LP Collaborative Reachability Analysis

Input:
M: MSG;
k: bound that limits the length of paths on the MSG;
n_T: target nodes;
N: bMSC set whose elements are related to the MSG;
Output:
reachability of the target nodes on the MSG

$BM^k \leftarrow$ encode(M, k, n_T)
while true **do**
  $\varsigma \leftarrow$ SAT(BM^k)
  if $\varsigma = NULL$ **then**
    return *unreachable*
  else
    $\rho \leftarrow$ decode(ς)
    $c \leftarrow$ findConstraints(ρ, M, N)
    $isFeasible \leftarrow$ LP(c)
    if *isFeasible* **then**
      return *reachable*
    else
      $\delta \leftarrow$ IIS(c)
      $\sigma \leftarrow$ map(δ)
      $BM^k \leftarrow BM^k \wedge \neg$ encode(σ)
    end if
  end if
end while

encode(M, k, n_T) ▷ Return SAT expressions of the given bounded graph structure or paths.
SAT(BM^k) ▷ Given the SAT expression, return the truth assignment if there exists.
decode(ς) ▷ Given a truth assignment of SAT, return the path.
findConstraints(ρ, M, N) ▷ Return the constraints related to the path.
LP(c) ▷ Return true if and only if the linear constraints are feasible.
IIS(c) ▷ Return IIS of the given infeasible path.
map(δ) ▷ Return the path segment related to given constraints.

start node, the target nodes, and the end node. TASSAT constructs five formulas–$START$, END, CON (Connection), ID (Identity), and $TARGET$–to depict the structure of the MSG. Intuitively, the $START$ and END formulas regulate the first nodes and the last nodes of candidate paths, respectively. The CON formulas constrain that, a node jumps to another node if and only if there exists an edge connects the two nodes in the MSG. In addition, the ID formulas ensure that only one node can appear in the same position at the same time. Finally, the $TARGET$ formulas require that the target nodes are included in the candidate paths. Equations $1 \sim 5$ are inferred from the Definition 2.3 and Definition 2.4. In these equations, the function α converts the given node and the bound index to a boolean variable; e.g., $\alpha(n_S, 0) = n_S^0$, the subscript S of the node is the name of the node, and the superscript 0 is the index of the node int the path. In addition, for a boolean formula like CON^k, the superscript k is the path bound. Given an MSG with bound k, the boolean formula BM^k defines how to combine five formulas, which represents the structural constraints and transition conditions of the MSG.

$$START := n_S^0 \wedge \bigwedge_{n \in N \wedge n \neq n_S} \neg\alpha(n, 0) \tag{1}$$

$$END^k := n_E^k \wedge \bigwedge_{n \in N \wedge n \neq n_E} \neg\alpha(n, k) \tag{2}$$

$$CON^k := \bigwedge_{0 \leqslant i \leqslant k-1} \Big(\bigwedge_{n \in N} \Big(\alpha(n, i) \rightarrow \bigvee_{(n,n') \in R} \alpha(n', i+1) \Big) \Big) \tag{3}$$

$$ID^k := \bigwedge_{1 \leqslant i \leqslant k-1} \Big(\bigwedge_{n \in N \wedge n \neq n_S, n_E} \Big(\alpha(n, i) \rightarrow \bigwedge_{n' \in N \wedge n' \neq n} \neg\alpha(n', i) \Big) \Big) \tag{4}$$

$$TARGET^k := \bigwedge_{n \in T} \Big(\bigvee_{1 \leq i \leq k-1} \alpha(n, i) \Big) \tag{5}$$

$$BM^k := START \wedge END^k \wedge CON^k \wedge ID^k \wedge TARGET^k \tag{6}$$

To illustrate the SAT encoding process in detail, we use a fragment of the ATM containing the start node, the end node, `StartTrans`, `GetPin`, and `EndTrans` as an example. Table 1 lists $START$, END, CON, ID, and $TARGET$ boolean formulas of the ATM MSG fragment subject to the path bound 1. In Table 1, n_{ST}, n_{GP}, and n_{ET} represents `StartTrans`, `GetPin`, and `EndTrans` respectively, and n_S and n_E represent the start node and the the end node of the ATM MSG. In this scenario, we choose `GetPin` as the target node.

Table 1: SAT encoding of the MSG with bound 1

Formula	CNF
START	$n_S^0 \wedge \neg n_{ST}^0 \wedge \neg n_{GP}^0 \wedge \neg n_{ET}^0 \wedge \neg n_E^0$
END	$n_E^1 \wedge \neg n_S^1 \wedge \neg n_{ST}^1 \wedge \neg n_{GP}^1 \wedge \neg n_{ET}^1$
CON	$\left(n_S^0 \rightarrow n_{ST}^1\right) \wedge \left(n_{ST}^0 \rightarrow n_{GP}^1\right) \wedge \left(n_{GP}^0 \rightarrow n_{ET}^1\right) \wedge \left(n_{ET}^0 \rightarrow n_{ST}^1 \vee n_E^1\right)$
ID	$\left(n_S^1 \rightarrow \left(\neg n_{ST}^1 \wedge \neg n_{GP}^1 \wedge \neg n_{ET}^1 \wedge \neg n_E^1\right)\right) \wedge \left(n_{ST}^1 \rightarrow \left(\neg n_S^1 \wedge \neg n_{GP}^1 \wedge \neg n_{ET}^1 \wedge \neg n_E^1\right)\right) \wedge \left(n_{GP}^1 \rightarrow \left(\neg n_S^1 \wedge \neg n_{ST}^1 \wedge \neg n_{ET}^1 \wedge \neg n_E^1\right)\right) \wedge \left(n_{ET}^1 \rightarrow \left(\neg n_S^1 \wedge \neg n_{ST}^1 \wedge \neg n_{GP}^1 \wedge \neg n_E^1\right)\right) \wedge \left(n_E^1 \rightarrow \left(\neg n_S^1 \wedge \neg n_{ST}^1 \wedge \neg n_{GP}^1 \wedge \neg n_{ET}^1\right)\right)$
TARGET	$n_{GP}^0 \vee n_{GP}^1$

According to the Equation 6, BM^k can be constructed by combining all the formulas in Table 1 with conjunction operators. A truth assignment which is generated by solving the boolean formula set BM^k, indicates that if there exist a potential path. Those variables whose value is true in the truth assignment, can be mapped to a path of the MSG. For example, if the path bound is set to 4 and variables n_S^0, n_{ST}^1, n_{GP}^2, n_{ET}^3 and n_E^4 are true, then we know that there exists a path $n_S \rightarrow$ `StartTrans` $\rightarrow$ `GetPin` $\rightarrow$ `EndTrans` $\rightarrow$ n_E. In this way, a candidate path which can be used in the further analysis is generated.

3.4 Linear Programming and IIS

Linear programming (LP) is a method to calculate the optimal outcome in a mathematical model whose requirements are represented by linear equalities or linear inequalities. In Section 3.3, we have demonstrated that a candidate path can be achieved by solving the boolean formula set BM^k. Obviously, all the timing constraints of a path according to Definition 2.5, such as $3 \leqslant j_b - j_8 \leqslant 5$, are linear relationships, therefore LP techniques are adopted to check if the candidate path subjects to the timing constraints. Because there exist loops in the MSG, such as `StartTrans`, `GetPin`, and `EndTrans` loop in Figure 1 (a), loops are unfolded to generate a path whose length equals to the bound. Consequently, duplicate nodes may exist on the candidate paths. However, synonymous constraints related to the duplicate nodes are different actually. If these synonymous constraints are combined together directly, it would cause conflicts when performing LP optimization. To address this problem, synonymous constraints are renamed according to the sequence number of the bMSC in the path.

For a linear programming problem, an irreducible infeasible set (IIS) is an infeasible subset of linear constraints, and the whole linear constraints will become feasible if any single constraint in the subset is removed [27]. IIS is used by TASSAT to accelerate searching for candidate feasible paths. If the constraints fed to the LP solver are infeasible, TASSAT calculates the IIS of constraints related to the path. Because a set linear constraints and nodes correspond to each other, the IIS of the infeasible path can be mapped to a group of unordered nodes, all of which are included in the former infeasible path. By enumerating the nodes of the original path, then the first and the last nodes identify the infeasible path segment. Then TASSAT encodes the infeasible path segment into SAT expressions in the way defined in Equation 7. In Equation 7, I is the infeasible node set, and *index* is a function which computes the index of the given node in the infeasible path. In each iteration of checking a candidate path, the *EXCLD* (Exclude) formula will be combined with the original CNF BM^k in the way of Equation 8 to assist the SAT solver to prune unnecessary paths containing infeasible path segments.

$$EXCLD^k := \bigwedge_{0 \leq i \leq k-|I|+1} \left(\neg \bigwedge_{n \in I} \alpha(n, i + index(n))\right) \tag{7}$$

$$FBM^k := BM^k \wedge EXCLD^k \tag{8}$$

3.5 Tool Implementation

We have implemented the approach described above into a tool named TASSAT in Java. We extend the UMLet [6, 7] to support the timing constraints and MSC specifications described in Section 2, so that the model which needs to be verified can be depicted through a friendly graphical interface. TASSAT employs SAT4j [15] to solve boolean satisfiability problems. SAT4j is a full featured boolean reasoning library, and it is intensively optimized to bring state-of-the-art SAT technology to Java ecosystem. The LP tool CPLEX [20] developed by IBM assembles powerful linear optimization modules and provides user-friendly interface to C++, C#, Python, and Java languages, so TASSAT employs CPLEX to solve linear programming problems.

4 EXPERIMENTS

4.1 Subjects

In the real industrial scenarios, in order to verify correctness of scenario-based specifications, it is conventional to choose an illegal node as the target node. So that if the target is unreachable, it means that the specifications are correct on the given path bound condition. Otherwise, it means that the verified system will enter the wrong state following the feasible path of TASSAT. Verifying specifications on more different bound conditions, the developers of the system are more confident of that the specifications are correct.

When evaluating performance of TASSAT, we chose ATM and GSM as subjects to evaluate the performance and efficiency of the TASSAT. Figure 1 (a) shows the MSG of ATM containing 12 different scenarios which are described in the form of bMSCs. GSM is the abbreviation of Global System for Mobile Communications, which is a typical real-time software system. Both of these cases are originated from the real-world scenarios and can be used to demonstrate interactions within different components.

Table 2: Part of experimental settings

Case	Constant Variables					
	W	B_1	B_2	T_1	T_2	$DEFT$
ATM 1	4	0	∞	0.5	2	0
ATM 2	6	1	2	0.5	2	0
GSM 1	5	0	∞	0.5	0.5	0.5
GSM 2	5	10	12	0.5	0.5	0.5

4.2 Comparative Technique

We compared TASSAT with TASS [22], which performs bounded model checking on the scenario-based specifications based on the graph traversal algorithm and linear programming techniques. We noticed that most related work only verifies scenario-based specifications for time consistency, which is a basic property. However, TASS is designed to perform reachability analysis like TASSAT, therefore we re-implement TASS in Java and compare it with TASSAT on time consumption and memory usage.

Satisfiability Modulo Theories (SMT) refers to the problem of determining whether a first-order formula is satisfiable with respect to some logical theory [24]. In the past decade, SMT solvers have attracted increased attention due to technological advances and industrial applications [14]. Solvers based on SMT are used as back-end engines in model-checking applications such as bounded model checking [8]. The SMT encoding approach can also solve the reachability analysis of MSC specifications, so we implemented an SMT encoding approach named TASSAT-SMT based on Z3 solver [13] in Java and compared it with TASSAT. The SMT encoding consists of two parts, MSG specification encoding and bMSC specification encoding. The MSG specification encoding is similar to SAT encoding described in the section 3.3, however, each location variable in SMT formulas is a string type variable other than a boolean variable. Encoding MSG specifications by string type variables is straightforward, and this method uses fewer variables than SAT encoding. There are a number of ways to encode MSG specifications to SMT formulas, we leave the research of performance of different encoding in the future work. On the other hand, due to Z3 supports real type theories, TASSAT-SMT encodes all time constraints together with MSG specification formulas, and the whole SMT expressions are solved by the solvers.

4.3 Experimental Protocol

In experimental cases, both ATM and GSM specifications contain constant variables W, B_1, B_2, T_1, T_2, and $DEFT$ (Default) of timing constraints. For example, a timing constraint $0 \leqslant j4 - h5 \leqslant W$ in ATM contains the constant variable W. In additional, the $DEFT$ variable indicates the minimum communication time between two instance on a bMSC, which is not specified explicitly on the each bMSC. Table 2 lists four groups of variable settings in the experiments. In both ATM cases, `DispenseCash` was chosen as the target node; and in GSM cases, `Handover` was chosen as the target node. The corresponding results of experimental settings are given in the form of tables. Table 3 ∼ Table 6 list results of experimental setting in Table 2 respectively.

We carried out the experiments on a Windows 10 PC with an Intel i5 9400F CPU (2.9 GHz) and 16 GB RAM, and set the maximum JVM heap size to 8 GB. VisualVM is a widely used JVM monitor, thus we employed VisualVM to monitor the memory usage of verification software and recorded the maximum heap usage. However, the Z3 solver with the Java interface used by TASSAT-SMT calls the C++ version Z3 library by Java Native Interface (JNI), hence it is difficult to restrict the TASSAT-SMT memory usage. To monitor TASSAT-SMT run-time status, we used Windows Task Manager to record the maximum memory usage. When TASSAT-SMT memory exceeded 8 GB, we marked it as OOM (Out Of Memory).

On the other hand, log-print codes were instrumented on specific code points (before and after verification process), so that time consumption of verification could be collected. In the experiments, we limited the maximum running time to one hour (3600 seconds), and if a single verification lasted for more than an hour, we terminated the verification and marked it as OOT (Out Of Time). If the programs encountered `StackOverFlowError` on JVM, we marked it as SOF (Stack Over Flow) in the result tables. We limited the maximum path bound to 5000, because it consumes tremendous memories and running time to finish the verification with too large path bounds. According to the research [5], most constraints inconsistent problems can be found out on the reasonable bounds.

4.4 Results

According to the experimental results, TASSAT overall consumes significantly less time to complete scenario-based verification than TASS and TASSAT-SMT. Tables 3, 4, 5, and 6 present the run-time data of the three checking approaches under different constant variable settings. The columns Path of these tables list the number of candidate paths which are checked during the verification. The asterisks (*) in the table represent that the run-time data failed to be collected due to out of time or other exceptions. The hyphens (-) in the table represent specific columns are meaningless for specific approaches. For example, the columns Path of TASSAT-SMT in all four tables are filled with hyphens, because the MSG structures and linear constraints are encoded and solved together in the TASSAT-SMT approach. The columns Time in these tables list the time consumption of these software, and all the time is recorded in seconds (s). As listed in Table 3 to Table 6, in most cases, TASSAT cost less time to verify MSC specifications than the other two software; and TASSAT fails to complete the checking only in ATM case 2 when a very large bound of 5000 was adopted. Thanks to the IIS analysis, TASSAT finished the verification within reasonable time even on many large bounds, because most of infeasible paths are pruned by $EXCLD$ formula.

While for TASS and TASSAT-SMT, as is listed in Table 3 and Table 5, as a result of a huge number of candidate paths in MSGs, time consumption of TASS grows drastically with increase of bounds. If the target node is unreachable, TASS exceeds time limitation when the path bound is greater than 50. However, if the target node is reachable, as is displayed in Table 4 and Table 6, TASS is efficient even on large bounds, since once the target node is verified to be reachable, the analysis process would stop. Similar to TASS,

Table 3: Results of ATM Case 1

Bound	TASS				TASSAT-SMT				TASSAT			
	Result	Path	Time	Memory	Result	Path	Time	Memory	Result	Path	Time	Memory
10	no	9	0.051	⩽60	no	-	0.082	⩽20	no	1	0.014	⩽20
25	no	1,613,729	OOT	⩽2400	no	-	1.089	⩽110	no	1	0.022	⩽35
50	*	*	OOT	⩽3000	no	-	6.465	⩽300	no	1	0.063	⩽60
100	*	*	OOT	⩽3000	no	-	82.688	⩽1120	no	1	0.147	⩽800
500	*	*	OOT	⩽4000	*	-	OOT	OOM	no	1	1.27	⩽1000
1000	*	*	OOT	⩽4000	*	-	OOT	OOM	no	5	19.3	⩽2600
2000	*	*	OOT	⩽4000	*	-	OOT	OOM	no	5	29	⩽2500
5000	*	*	SOF	SOF	*	-	OOT	OOM	no	5	294	⩽2400

Table 4: Results of ATM Case 2

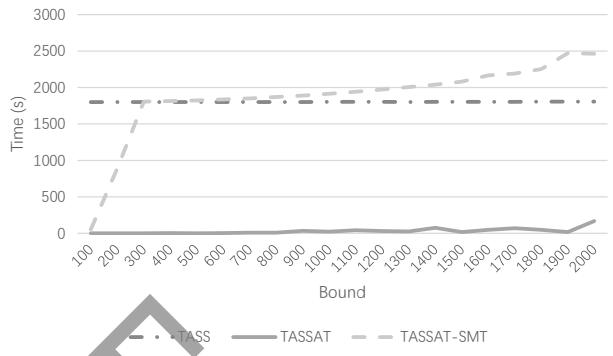
Bound	TASS				TASSAT-SMT				TASSAT			
	Result	Path	Time	Memory	Result	Path	Time	Memory	Result	Path	Time	Memory
10	yes	1	0.007	⩽50	yes	-	0.085	⩽40	yes	1	0.006	⩽40
25	yes	1	0.011	⩽60	yes	-	0.26	⩽100	yes	1	0.014	⩽95
50	yes	1	0.014	⩽70	yes	-	0.636	⩽120	yes	1	0.024	⩽100
100	yes	1	0.031	⩽75	yes	-	1.76	⩽160	yes	1	0.08	⩽200
500	yes	1	0.984	⩽550	yes	-	45.915	⩽810	yes	1	0.411	⩽820
1000	yes	1	4.9	⩽1500	yes	-	231.824	⩽1200	yes	1	20.79	⩽2400
2000	yes	1	13.42	⩽2500	yes	-	1018.363	⩽2300	yes	13	300.1	⩽2500
5000	*	*	SOF	SOF	*	-	OOT	OOM	*	*	OOT	⩽4200

Table 5: Results of GSM Case 1

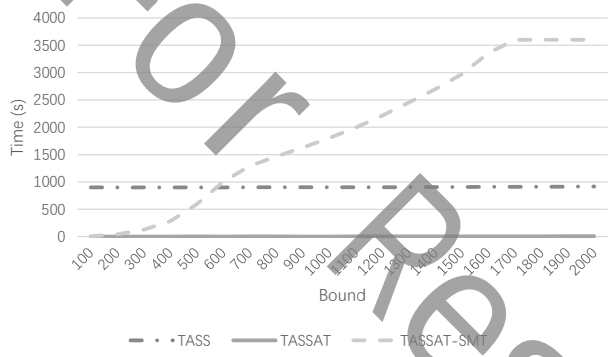
Bound	TASS				TASSAT-SMT				TASSAT			
	Result	Path	Time	Memory	Result	Path	Time	Memory	Result	Path	Time	Memory
10	no	0	0.001	⩽60	no	-	0.046	⩽90	no	0	0.006	⩽30
25	no	176	1.4	⩽900	no	-	0.414	⩽110	no	1	0.042	⩽50
50	no	304,236	OOT	⩽3000	no	-	2.062	⩽170	no	1	0.04	⩽75
100	*	*	OOT	⩽3000	no	-	8.844	⩽400	no	1	0.09	⩽80
500	*	*	OOT	⩽3000	no	-	1515.271	OOM	no	1	1.74	⩽2000
1000	*	*	OOT	⩽4000	no	-	OOT	OOM	no	1	13.5	⩽1450
2000	*	*	OOT	⩽4000	no	-	OOT	OOM	no	1	8.95	⩽3500
5000	*	*	SOF	SOF	no	-	OOT	OOM	no	1	79.1	⩽4200

Table 6: Results of GSM Case 2

Bound	TASS				TASSAT-SMT				TASSAT			
	Result	Path	Time	Memory	Result	Path	Time	Memory	Result	Path	Time	Memory
10	no	0	0.001	⩽50	no	-	0.049	⩽80	no	0	0.006	⩽30
25	yes	1	0.012	⩽80	yes	-	0.454	⩽130	yes	1	0.084	⩽50
50	yes	1	0.021	⩽250	yes	-	1.528	190	yes	1	0.04	⩽75
100	yes	1	0.036	⩽300	yes	-	5.376	⩽400	yes	1	0.09	⩽80
500	yes	1	0.619	⩽600	yes	-	275.591	⩽1290	yes	1	2.6	⩽2000
1000	yes	1	2.88	⩽2800	yes	-	1199.705	⩽2210	yes	1	13.5	⩽1450
2000	yes	1	20.86	⩽2000	*	-	OOT	⩽4030	yes	1	7.07	⩽3000
5000	*	*	SOF	SOF	*	-	OOT	OOM	yes	1	86.39	⩽3500

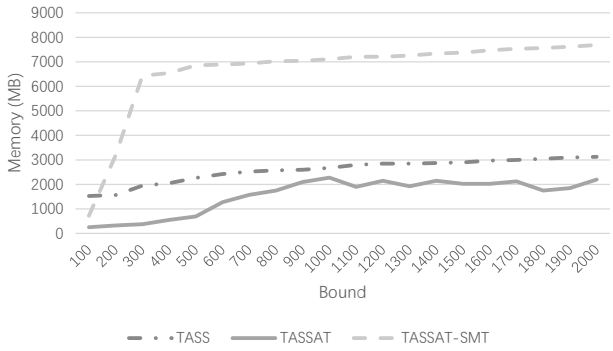


(a) Average time consumption of ATM

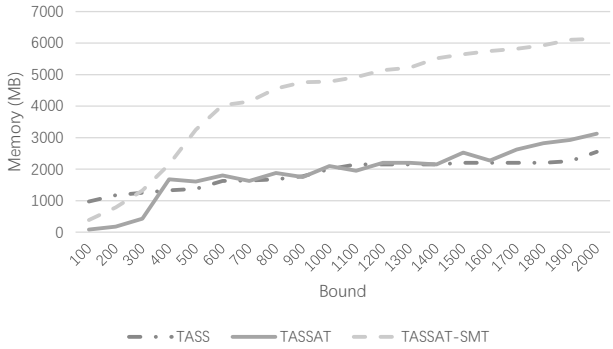


(b) Average time consumption of GSM

Figure 3: Average time consumption line chart



(a) Average memory usage of ATM



(b) Average memory usage of GSM

Figure 4: Average memory usage line chart

TASSAT-SMT is more efficient when the target node is reachable. However, since TASSAT-SMT needs to encode the entire model while TASS only checks one path a time, the time consumption of TASSAT-SMT grows rapidly due to the growth of SMT clauses [24].

To better compare the time cost, we depict the average time consumption of ATM and GSM verification from the bound 100 to the bound 2000 in Figure 3. If certain checks failed or exceeded one hour, we counted them as 3600 seconds. As is shown in Figure 3 (a) and Figure 3 (b), TASSAT costs much less time than TASS and TASSAT-SMT. Additionally, the time cost of TASSAT-SMT grows quickly as the bound increases. This suggests that without the IIS, a pure SMT approach is inferior than the path oriented approach like TASS.

We also compare the memory consumption of the three approaches. The columns Memory of Table 3, 4, 5 and 6 list the memory consumption of different approaches. Due to the fact that the memory usage varies constantly during each verification, we only recorded the maximum memory usage. TASSAT consumes less memory than TASS on average, because TASSAT checks much less candidate paths. Due to depth-first traversal algorithm used by TASS, the number of candidate paths increase exponentially with the growth of path bound. Every time when the path bound is set to 5000, TASS encounters `StackOverFlowError` and fails to finish reachability analysis. TASSAT and TASSAT-SMT both utilize SAT or SMT technology, therefore their clause numbers grow with increase of bounds. However, TASSAT checks timing constraints and searches candidate paths independently, so that the clause growth of TASSAT is much slower than TASSAT-SMT's and the growth of the memory usage of TASSAT is slower.

Figure 4 shows the average memory usage of different verification software, and the memory consumption is recorded in megabytes (MB). As is shown on Figure 4 (a), from the path bound 100 to 2000, TASSAT consumes less memory than TASS and TASSAT-SMT on average. When the path bound is less than 400, TASSAT consume no more than 1000 MB memories, which means TASSAT is able to cover most verification scenes with lightweight memory footprint. The growth curve of TASSAT's memory usage is more flattened than TASSAT-SMT's; however, for GSM cases shown in Figure 4 (b), there are no significant difference between TASS and TASSAT on memory usage. We reasoned that more than 70 percent of GSM cases are reachable, which is why TASS has better performance on GSM than ATM. As mentioned earlier, due to the rapid growth of SMT clauses, TASSAT-SMT has significant building-up curves on both figures.

5 RELATED WORK

A series of work has been done about the verification of message sequence charts [2–4]. Alur et al. [3] proposed that the problem of checking the basic MSCs with delay intervals for time consistency can be reduced into computing negative cost cycles and the shortest distance in a weighted directed graph. The subsequent work done by Alur et al. in 1999 [4] indicates that model checking can be done

by constructing a suitable automaton for the linearization of the partial order specified by the MSC. Alur et al. also proposed two semantics *synchronous* and *asynchronous* of MSG, which means two different interpretations of concatenation of two MSCs.

A timing analysis technique named TASS [22] focused on the challenges in the verification of the scenario-based specifications (SBS), which is also the main research area in this paper, therefore we chose TASS as the comparative technique in the experiments. Unlike TASSAT, TASS uses depth-first graph traversal algorithms to enumerate the candidate feasible paths but ignores the precise IIS information of infeasible timing constraints, therefore TASS checks every candidate path until a feasible path is found. Besides the limitations mentioned above, TASS introduced comprehensive definitions of the scenario-based specifications and demonstrated the properties of MSC specifications.

Xie et al. [35] proposed an approach to address the problem of checking linear hybrid automata (LHA) using SAT encoding and linear programming [17]. The bounded model checking for LHA is a challenging problem, and most work tries to handle this problem by encoding all the discrete and continuous behaviours with the given bound into a set of SMT [8] formulas, so that they can be solved by off-the-shelf SMT solvers [35]. However, when the system scale is large, the SMT formulas could be complicate and difficult to solve. Xie et al. achieved better results by using SAT encoding and linear programming. TASSAT also uses SAT and LP techniques, but MSC specifications are different than LHA, which requires a new encoding and checking approach.

6 CONCLUSION

In this paper, we propose a SAT and LP collaborative technique named TASSAT to accelerate the verification of the scenario-based specifications. In addition, we calculate IIS of infeasible paths and use the crucial information to prune paths containing the infeasible path segments. According to the experimental results demonstrated in Section 4, TASSAT outperforms state-of-the-art depth-first graph traversal based timing analysis techniques and the SMT-based techniques considering time consumption and memory usages.

In the future, we will collect more run-time statistics of TASSAT and reduce the memory usage when the path bound is large. By further improving the efficiency of TASSAT, we will be able to adopt timing analysis technique on industrial cases.

REFERENCES

[1] Rajeev Alur and David L Dill. 1994. A theory of timed automata. *Theoretical computer science* 126, 2 (1994), 183–235.

[2] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. 2003. Inference of message sequence charts. *IEEE Transactions on Software Engineering* 29, 7 (2003), 623–633.

[3] Rajeev Alur and Doron Holzmann, Gerard J.and Peled. 1996. An analyzer for message sequence charts. In *Tools and Algorithms for the Construction and Analysis of Systems*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 35–48.

[4] Rajeev Alur and Mihalis Yannakakis. 1999. Model Checking of Message Sequence Charts. In *Proceedings of the 10th International Conference on Concurrency Theory (CONCUR '99)*. Springer-Verlag, London, UK, UK, 114–129.

[5] Alexandr Andoni, Dumitru Daniliuc, Sarfraz Khurshid, and Darko Marinov. 2003. Evaluating the "small scope hypothesis". In *In Popl*, Vol. 2. Citeseer.

[6] Martin Auer, Ludwig Meyer, and Stefan Biffl. 2007. Explorative UML Modeling-Comparing the Usability of UML Tools.. In *ICEIS (3)*. 466–473.

[7] M. Auer, T. Tschurtschenthaler, and S. Biffl. 2003. A Flyweight UML Modelling Tool for Software Development in Heterogeneous Environments. In *Proceedings of the 29th Conference on EUROMICRO (EUROMICRO '03)*. IEEE Computer Society, Washington, DC, USA, 267–. http://dl.acm.org/citation.cfm?id=942796.943259

[8] Clark Barrett and Cesare Tinelli. 2018. Satisfiability modulo theories. In *Handbook of Model Checking*. Springer, 305–343.

[9] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, Yunshan Zhu, et al. 2003. Bounded model checking. *Advances in computers* 58, 11 (2003), 117–148.

[10] Hans Kleine Büning and Theodor Lettmann. 1999. *Propositional logic: deduction and algorithms*. Vol. 48. Cambridge University Press.

[11] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. 2011. Efficient Scenario Verification for Hybrid Automata. In *Proceedings of the 23rd International Conference on Computer Aided Verification* (Snowbird, UT) *(CAV'11)*. Springer-Verlag, 317–332.

[12] Alessandro Cimatti, Sergio Mover, and Stefano Tonetta. 2013. SMT-based scenario verification for hybrid systems. *Formal Methods in System Design* 42, 1 (2013), 46–66.

[13] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 337–340.

[14] Leonardo De Moura and Nikolaj Bjørner. 2011. Satisfiability modulo theories: introduction and applications. *Commun. ACM* 54, 9 (2011), 69–77.

[15] Niklas Eén and Niklas Sörensson. 2003. An extensible SAT-solver. In *International conference on theory and applications of satisfiability testing*. Springer, 502–518.

[16] Holger Giese, Gabor Karsai, Edward A Lee, Bernhard Rumpe, and Bernhard Schätz. 2010. *Model-Based Engineering of Embedded Real-Time Systems: International Dagstuhl Workshop, Dagstuhl Castle, Germany, November 4-9, 2007. Revised Selected Papers*. Vol. 6100. Springer.

[17] Thomas A Henzinger. 2000. The theory of hybrid automata. In *Verification of digital and hybrid systems*. Springer, 265–292.

[18] ITU-TS. 2011. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC)*. ITU-TS, Geneva.

[19] Paul Jackson and Daniel Sheridan. 2004. Clause form conversions for boolean circuits. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 183–198.

[20] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. 2018. IBM ILOG CP optimizer for scheduling. *Constraints* 23, 2 (2018), 210–250.

[21] Peter B Ladkin and Stefan Leue. 1992. *Interpreting message sequence charts*. IBM Thomas J. Watson Research Division.

[22] Xuandong Li, Minxue Pan, Lei Bu, Linzhang Wang, and Jianhua Zhao. 2012. Timing Analysis of Scenario-based Specifications Using Linear Programming. *Softw. Test. Verif. Reliab.* 22, 2 (2012), 121–143. https://doi.org/10.1002/stv.434

[23] Minh Chau Nguyen, Eunkyoung Jee, Jinho Choi, and Doo-Hwan Bae. 2014. Automatic Construction of Timing Diagrams from UML/MARTE Models for Real-time Embedded Software. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing* (Gyeongju, Republic of Korea) *(SAC '14)*. ACM, 1140–1145. https://doi.org/10.1145/2554850.2555011

[24] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 2006. Solving SAT and SAT modulo theories: from an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL (T). *Journal of the ACM (JACM)* 53, 6 (2006), 937–977.

[25] Olga Ohrimenko, Peter J Stuckey, and Michael Codish. 2009. Propagation via lazy clause generation. *Constraints* 14, 3 (2009), 357–391.

[26] Minxue Pan and Xuandong Li. 2012. Timing analysis of MSC specifications with asynchronous concatenation. *International Journal on Software Tools for Technology Transfer* 14, 6 (2012), 639–651. https://doi.org/10.1007/s10009-012-0239-9

[27] Mark Parker and Jennifer Ryan. 1996. Finding the minimum weight IIS cover of an infeasible system of linear inequalities. *Annals of Mathematics and Artificial Intelligence* 17, 1 (1996), 107–126.

[28] Doron A Peled. 2013. *Software reliability methods*. Springer Science & Business Media.

[29] Claude Sammut and Geoffrey I Webb. 2017. *Encyclopedia of machine learning and data mining*. Springer Publishing Company, Incorporated.

[30] Srikrishna Sridhar, Stephen Wright, Christopher Re, Ji Liu, Victor Bittorf, and Ce Zhang. 2013. An approximate, efficient LP solver for LP rounding. In *Advances in Neural Information Processing Systems*. 2895–2903.

[31] Ranjita Kumari Swain, Vikas Panthi, and Prafulla Kumar Behera. 2012. Test case design using slicing of UML interaction diagram. *Procedia Technology* 6 (2012), 136–144.

[32] Matthew W Tanner and Lewis Ntaimo. 2010. IIS branch-and-cut for joint chance-constrained programs with random technology matrices. *Eur. J. Oper. Res* 207, 1 (2010), 290–296.

[33] Gregory Tassey. 2002. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology, RTI Project* 7007, 011 (2002), 429–489.

[34] J Eldon Whitesitt. 2012. *Boolean algebra and its applications*. Courier Corporation.

[35] Dingbao Xie, Lei Bu, Jianhua Zhao, and Xuandong Li. 2014. SAT–LP–IIS joint-directed path-oriented bounded reachability analysis of linear hybrid automata. *Formal Methods in System Design* 45, 1 (2014), 42–62.