Bounded Satisfiability Checking of Metric First-Order Temporal Logic

Nick Feng  
University of Toronto  
fengnick@cs.toronto.edu  
Mehrdad Sabetzadeh  
University of Ottawa  
mehrdad@cs.uottawa.ca

Lina Marsso  
University of Toronto  
lina.marssso@utoronto.ca  
Marsha Chechik  
University of Toronto  
chechik@cs.toronto.edu

ABSTRACT
Legal properties involve reasoning about data values and time. Metric first-order temporal logic (MFOTL) provides a rich formalism for specifying legal properties. While MFOTL has been successfully used for verifying legal properties over operational systems via runtime monitoring, no solution exists for MFOTL-based verification in early-stage system development captured by requirements. Given a legal property and system requirements both formalized in MFOTL, the compliance of the property can be verified on the requirements via satisfiability checking. In this paper, we propose a practical, sound, and complete (within a given bound) satisfiability checking approach for MFOTL. The approach, based on satisfiability modulo theories (SMT), employs a counterexample-guided strategy to incrementally search for a satisfying solution. We implemented our approach in a tool called LEGOS, and evaluated it on five case studies spanning the healthcare, business administration, banking and aviation domains. Our results indicate that our approach can efficiently determine whether legal properties of interest are met, or generate counterexamples that lead to compliance violations.

KEYWORDS
Requirements Engineering, Metric First-order Temporal Logic (MFOTL), Satisfiability Modulo Theories (SMT).

1 INTRODUCTION
Software systems, such as medical systems or banking systems, are increasingly required to comply with laws, regulations and policies aimed at ensuring safety, security, data protection and privacy [22, 38]. The properties stipulated by these laws, regulations and policies – which we refer to as legal properties (LP) hereafter – typically involve reasoning about actions, ordering and time. As an example, consider the following LP, P₁, derived from a health-data regulation (s. 11, PHIPA [24]): “If personal health information is not accurate or not up-to-date, it should not be accessed”. In this property, the accuracy and the freshness of the data depend on how and when the data was collected and updated before being accessed. More precisely, this property constrains the data action access to have accurate and up-to-date data values, which further constrains the order and time of access with respect to other data actions.

System compliance with LPs can be checked on the system design or on an operational model of a system implementation. In this paper, we focus on the early stage, where one can check whether a formalization of the system requirements satisfies an LP. The formalization can be done using a descriptive formalism like temporal logic, instead of using an operational one, based on transition systems [28, 37]. For instance, the requirement (req₀) of a data collection system: “no data can be accessed prior to 15 days after the data has been collected” needs to be formalized for verifying compliance of P₁. It is important to formalize the data and time constraints of both the system requirements and LPs, such as the ones of P₁ and req₀.

Metric first-order temporal logic (MFOTL) enables the specification of data and time constraints [4] and has an expressive formalism for capturing LPs and the related system requirements that constrain data and time [1]. Existing work on MFOTL verification has focused primarily on detecting violations at runtime through monitoring [1], with MFOTL formulas being checked on execution logs. There is an unsatisfied need on determining the satisfiability of MFOTL specifications, i.e., looking for legal-property violations possible in MFOTL specification, which is important for designing and debugging MFOTL requirements to satisfy legal properties.

MFOTL satisfiability checking is generally undecidable since MFOTL is an extension of first-order logic (FOL). Restrictions are thus necessary for making the problem decidable. In this paper, we restrict ourselves to safety properties. While we are not aware of non-safety LP properties, it is conceivable that they may exist. For safety properties, LP violations are finite sequences of data actions, captured via a finite-length counterexample. For example, a possible violation of P₁ is a sequence consisting of storing a value v in a variable d, updating d’s value to v′, then reading d again and not obtaining v’. Since we are interested in finite counterexamples, bounded verification is a natural strategy to pursue for achieving decidability. SAT solvers have been previously used for bounded satisfiability checking of metric temporal logic (MTL) [28, 37]. However, MTL cannot effectively capture quantified data constraints (e.g., “every used data must be up-to-date”) in LPs, hence the solution is not applicable directly. As an extension to MTL, MFOTL can effectively capture data constraints used in LP. Yet, to the best of our knowledge, there has not been any prior work on bounded MFOTL satisfiability checking.

To establish a bound in bounded verification, researchers predominantly relied on bounding the size of the universe, e.g., every
data type has a bounded domain in Alloy [23]. In our work, bounding the universe would be too restrictive because LPs routinely refer to variables with large ranges, e.g., timed actions spanning several years. Instead, we bound the number of data actions in a run, which bounds the number of actions in the counterexample.

Equipped with the proposed notion of a bound, we develop an incremental approach (IBSC) for bounded satisfiability checking of MFOTL. We first translate the MFOTL property and requirements into FOL formulas with quantifiers over data actions. We then incrementally ground the FOL constraints to eliminate the quantifiers by considering an increasing number of data actions. Subsequently, we check the satisfiability of the resulting constraints using SMT solver Z3 [10]. Specifically, we make the following contributions: (1) we propose a translation of MFOTL formulas to FOL; (2) we provide a novel bounded satisfiability checking solution for the translated FOL formulas with incremental and counter-example guided over/under-approximation. We implemented our approach IBSC in a tool called LEGOS, and empirically evaluate it on five case studies with a total of 24 properties showing that it can effectively and efficiently find LP violations or prove satisfiability.

The rest of this paper is organized as follows. Sec. 2 provides background and establishes our notation. Sec. 3 defines the bounded satisfiability checking (BSC) problem. Sec. 4 provides an overview of our solution and the translation of MFOTL to FOL. Sec. 5 presents our solution, the algorithm and its proof of correctness and termination. Sec. 6 reports on the experiments we have performed to validate our bounded satisfiability checking solution for MFOTL. Sec. 7 discusses related work. Sec. 8 concludes the paper.

2 PRELIMINARIES

In this section, we introduce the necessary background on metric first order temporal logic (MFOTL).

Syntax. We begin with the syntax of MFOTL [4].

Definition 1. Let be a non-empty interval over N. An interval in can be expressed as [b, b′) where b, b′ ∈ N ∪ ∞. A signature S is a tuple (C, R, i), where C is a set of constant and R is a finite set predicate symbols, respectively. Without loss of generality, we assume all constants are from the integer domain Z where the theory of linear integer arithmetic (LIA) holds. The function i : R → N associates each predicate symbol r ∈ R with an arity i(r) ∈ N. Let V be a countable infinite set of variables from domain I and a term t is defined inductively as t : c | v | t + t′ | c×t. We denote i as a vector of terms and i|x_k as the vector that contains x at index k. The syntax of MFOTL formulas is defined as follows:

1. T and L, representing values “true” and “false”;
2. t = t′ or t > t′, for term t and t′;
3. r(t1,...,tr) for r ∈ R and t1,...,tr are terms.
4. φ ∧ ψ, ¬φ for MFOTL formulas φ and ψ;
5. ∃x · (r[i]x) ∧ L for MFOTL formula φ, relation symbol r ∈ R, variable x ∈ V and a vector of terms i such that x ∈ i[x];
6. φ U t ψ, φ S t ψ, ◦ φ for MFOTL formulas φ and ψ, and an interval i ∈ I.

We considered a restricted form of quantification (syntax rule 5) similar to guarded quantification [20, 21]. Every quantifiably variable x must be guarded by some relation r (i.e., for some i, r(i) holds and x appears in i). Similarly, universal quantification must be guarded as v · (r ¯ x) ⇒ φ) where x ∈ i. Under the restricted quantification, ∃x · ¬r(x) and v · (r(x)) are not allowed.

The temporal operators U, S, intersection and U S require the satisfaction of the formula within the time interval given by i. We write [b, b′) as a shorthand for [b, ∞), if i is omitted, then the interval is assumed to be [0, ∞). Other classical unary temporal operators φ (i.e., eventually), φ (i.e., always), and φ (i.e., once) are defined as follows: φ := T U φ, φ := ¬φ ∧ φ, and φ := T S φ. Other common logical operator such as ∨ (disjunction) and (universal quantification) are expressed through negation of ∧ and E, respectively.

Example 1. Suppose a data collection centre (DCC) collects and accesses personal data information with three requirements: req0 stating that no data is allowed to be accessed before being having been collected for 15 days (360 hours); req1: data can only be updated after having been collected or last updated for more than a week (168 hours); and req2: data can only be accessed if has been collected or updated within a week (168 hours). The signature for DCC contains three binary relations (Rdata): Collect, Update, and Access, such that Collect(d, v), Update(d, v) and Access(d, v) hold at a given time point if and only if data at id d is collected, updated, and accessed with value v at this time point, respectively. The MFOTL formulas for P1, req0, req1 and req2 are shown in Fig. 1. For instance, the formula req0 specifies that if a data value stored at id d is accessed, then some data must have been collected and stored at id d before, and the collection must have occurred prior to 360 hours ago (φ[360] ).

Semantics. A first-order (FO) structure D over the signature S = (C, R, i) comprises a domain |D| ≠ ∅ and an interpretation for each c ∈ C and r ∈ R. The semantics of MFOTL formulas is defined over a sequence of FO structures D = (D0, D1, . . .) and a sequence of natural numbers representing time t = (t0, t1, . . .), where (a) t is a monotonically increasing sequence; (b) |Dt| = |Di+1| for all i > 0 (all Di have a fixed domain); and (c) each constant symbol c ∈ C has the same interpretation across D (i.e., cDk = cDk+1).

Property (a) ensures that time never decreases as the sequence progresses; and (b) ensures that the domain is fixed. D is similar to timed words in metric time logic (MTL), but instead of associating a property (a) ensures that time never decreases as the sequence progresses; and (b) ensures that the domain is fixed. D is similar to timed words in metric time logic (MTL), but instead of associating a property (a) ensures that time never decreases as the sequence progresses; and (b) ensures that the domain is fixed. D is similar to timed words in metric time logic (MTL), but instead of associating a property of each state with the next state, the semantics of MFOTL is defined over a sequence of timed first-order structures σ = (D, t), where every structure Dk ∈ D contains a set of holding relations rDk, and these relations occur at a time specified by tk ∈ t. D contains a set of holding relations rDk, and these relations occur at a time specified by tk ∈ t.

Example 2. Consider the signature for the DCC example. Let t1 = 0 and t2 = 361, and let D1 and D2 be two first order structures with rD1 = Collect(0, 0) and rD2 = Access(0, 0), respectively. The trace σ1 = (D1; D2), [t1; t2]) is a valid trace shown in Fig. 2 and representing two timed relations: (1) data value 0 collected and stored at id 0 at hour 0 and (2) data value 0 is read by accessing id 0 at hour 361.

A valuation function v : V → |D| maps a set of variables V to their interpretations in the domain |D|. For vectors x = (x1, . . . , xn) and d = (d1, . . . , dn) ∈ |D| n, the update operation v[x → d] produces
Consider the DCC example, the MFOTL formula
\[ S_\text{MFOTL} = (\phi_0 \land \phi_1 \land \phi_2 \land \phi_3) \]
which defines the satisfaction of the formula within a time
\[ (D, \tau, v, i) \models \phi \]
iff
\[ \exists \exists \forall \land \phi \]
and
\[ (D, \tau, v, i) \models \psi \iff (r(\tau'), v', i) \models \psi \]
for some \( r \in [0, 1] \). The satisfaction of the formula is
\[ (D, \tau, v, i) \models \phi \iff (r(\tau'), v', i) \models \phi \]
and
\[ (D, \tau, v, i) \models \psi \iff (r(\tau'), v', i) \models \psi \]
for some \( r \in [0, 1] \).

Definition 4 (Satisfiability Checking of MFOTL Formulas). Let \( \phi \) be an MFOTL formula over a signature \( S = (C, R, i) \), and let \( \phi \) be a set of MFOTL requirements over \( S \). \( \phi \) is satisfiable if there exists a sequence of FO structures \( D \) and natural numbers \( \tau \), and a valuation function \( v \) such that \( (D, \tau, v, i) \models \phi \).

Example 4. In the DCC example, the MFOTL formula \( \phi_0 \) is satisfiable because \( (D, \tau, v, i) \models \phi_0 \) (where \( \sigma_1 = (D, \tau) \) in Fig. 2). Let \( \phi_0' = (\phi_0 \land \neg \phi_1) \) be another MFOTL formula. \( \phi_0' \) is unsatisfiable because if data stored at id 0 is accessed between 0 and 395 hours, then it is impossible to collect the data at least 360 hours prior to its access.

3 BOUNDED SATISFIABILITY CHECKING

The satisfiability of MFOTL properties is generally undecidable since MFOTL is expressible enough for describing the blank tape problem [33] (which has been shown to be undecidable). Despite the undecidability result, we can derive a bounded version of the problem, bounded satisfiability checking (BSC), for which a sound and complete decision procedure exists. When facing a hard instance for the bounded satisfiability checking problem, the solution to BSC provides bounded guarantees (i.e., exists a solution or there is no solution within the given bound). In this section, we first define satisfiability checking and then the BSC problems of MFOTL formulas.

Satisfiability checking [34] is a verification technique that extends model checking by replacing a state transition system with a set of temporal logic formulas. In the following, we define satisfiability checking of MFOTL formulas.

Definition 5 (Finite trace and bounded trace). The volume of a trace \( \sigma \) is defined as the number of relations that hold across \( D \), denoted as \( vol(\sigma) = \sum_{r \in R} \sum_{D \in D}(\sigma(r)) \). The trace \( \sigma \) is finite if

**Figure 1:** Several requirements and the legal property \( P_1 \) of DCC, with the signature \( S_\text{data} = (\emptyset, (\text{Collect}, \text{Update}, \text{Access}), i_\text{data}) \), where

\[
\begin{align*}
\text{Collect}(0,0) & \quad \text{Access}(0,0) \\
\sigma_1 \downarrow & \quad \sigma_2 \downarrow \\
\text{Collect}(1,0) & \quad \text{Access}(1,0) \\
\sigma_3 \downarrow & \quad \sigma_4 \downarrow \\
\text{Update}(0,0) & \quad \text{Access}(0,0) \\
\sigma_5 \downarrow & \quad \sigma_6 \downarrow \\
\text{Collect}(0,1) & \quad \text{Access}(0,0) \\
\sigma_7 \downarrow & \quad \sigma_8 \downarrow \\
\text{Update}(1,0) & \quad \text{Access}(0,0) \\
\sigma_9 \downarrow & \quad \sigma_{10} \downarrow \\
\end{align*}
\]

**Figure 2:** Several traces from the DCC example.
\textnormal{vol}(\sigma) \text{ is finite. The trace is bounded by volume if there exists an upper bound } b_{\text{vol}} \in \mathbb{N} \text{ such that } \textnormal{vol}(\sigma) \leq b_{\text{vol}}. \text{ In the rest of paper, we refer to volume-bounded traces as bounded traces.}

**Example 6.** The volume of trace \( \sigma_4 \) in Fig. 2, \( \text{vol}(\sigma_4) = 3 \) since there are three relations: \textit{Collect}(1, 15), \textit{Update}(1, 0), and \textit{Access}(1, 15). Note that the volume and the length for this trace are different. The length, \(|D|\), corresponds to the number of time points while the volume is the total number of holding relations across all time points. More than one relation can hold for a single time point.

**Definition 6 (Bounded satisfiability checking of MFOTL properties).** Let \( \phi \) be an MFOTL property, Reqs be a set of MFOTL requirements, and \( b_{\text{vol}} \) be a natural number. The bounded satisfiability checking problem determines the existence of a counterexample \( \sigma \) to Reqs \( \Rightarrow \phi \) such that \( \text{vol}(\sigma) \leq b_{\text{vol}} \).

4 CHECKING BOUNDED SATISFIABILITY

In this section, we present an overview of the bounded satisfiability checking (BSC) process that translates MFOTL formulas into first-order logic (FOL) formulas, and looks for a satisfying solution for the FOL formulas. Then, we provide the translation of MFOTL formulas to FOL and discuss the complexity of the process.

4.1 Overview of BSC for MFOTL Formulas

We aim to address the bounded satisfiability checking problem (Def. 6), looking for a satisfying run \( \sigma \) within a given volume bound \( b_{\text{vol}} \) that limits the number of relations in \( \sigma \). First, we translate the MFOTL formulas to FOL formulas, where each relation in a trace is mapped to a relational object in the FO domain. The considered constraints in the formulas include the ones of the system requirements and the legal property, and optional data constraints specifying the data domain of the system. Second, we search for a satisfying solution to the FOL constraints; an SMT solver is used here to determine the satisfiability of the FOL constraints and the data domain constraints. The answer from the SMT solver is analyzed to return an answer to the satisfiability checking problem (a counterexample \( \sigma \), or "bounded-UNSAT").

4.2 Translation of MFOTL to First-Order Logic

The translation is a two-step procedure: (1) defining the FOL search domain and (2) translating MFOTL formulas to FOL formulas. This translation enables to search for a solution of the bounded satisfiability problem in the FOL formulas.

4.2.1 Domain of the Search Space. Any volume-bounded trace \( \sigma = (D, r, o) \) (bounded by \( b_{\text{vol}} \)) can be mapped to a single FO structure \( (D_{AU}, o') \) by associating the time information in \( r \) to every relation in \( D \) for every \((D_i, r_i)\) in the trace. More specifically, let \( D_{AU} \) be the domain defined over signature \( S = (C, R, \iota) \) (the same signature as \( D \)). \( D_{AU} \) contains all constants \( c \in C \) and \( b_{\text{vol}} \) relational objects (denoted \( o \)). Each \( o \) has variables \( \text{name} \) and \( \text{time} \), and a vector of argument variables \( arg_1, ..., arg_{|\text{arg}]}(\text{name}) \) associated with it. Each \( o \) represents an occurrence of a relation of \( \text{name} \) at time \( \iota\text{time} \). The following relationship holds for \((D_{AU}, o')\) and \((D, r, o)\): \( \forall r \in R, r(t_1, t_2, ..., t_{|\iota|}) \in r^{D_{AU}} \).

iff exists a relational object \( o(arg_1, arg_2, ..., arg_{|\iota|}) \in D_{AU} \) s.t. 
\( o(t_1) = o'(arg_1), o(t_2) = o'(arg_2), ..., o(t_{|\iota|}) = o'(arg_{|\iota|}) \) \( \land t_i = o'(\iota\text{time}) \land o'(\text{name}) = r \).

Intuitively, each holding relation in the trace \( \sigma \) is mapped to a relational object where the holding time of the relation \( r_j \) is explicitly associated with the relational object’s \( \iota\text{time} \). Therefore, if a relation does not hold at a given time, then there is no relational object which the relation can be mapped to. The name of the relation \( \text{name} \) defines the class \( CIs \) of relational objects in \( D_{AU} \). For example, \( o: CIs \) selects every relational object where \( \text{name} = CIs \). In the rest of the paper, to avoid confusion, class names and relation names are capitalized (e.g., \( \text{Collect} \)) while individual relational object names are not (e.g., \( \text{collect}_1 \)).

**Example 7.** Consider the trace \( \sigma_2 = (\bar{D}, \bar{r}) \) where \( \bar{D} = [D_1; D_2; D_3] \) and \( \bar{r} = [r_1; r_2; r_3] \) in Fig. 2. The first-order structure of \( \sigma_2 \) is \((D_{AU}, o')\), where \( D_{AU} \) contains three relational objects. Two of these objects are from the class \( \text{Collect} \), and the third one is from the class \( \text{Access} \). Let \( \text{collect}_1 \) be the relational object corresponding to the relation \( \text{Collect}(1, 0) \) in \( D_1 \) with \( r_1 = 0 \). We have: \( o'(\text{collect}_1, arg_1) = 1, o'(\text{collect}_1, arg_2) = 0 \) and \( o'(\text{collect}_1, \iota\text{time}) = 0 \).

For the FO domain \( D_{AU} \), let \( |D_{AU}| \) denote the domain size, i.e., the number of its relational objects. Therefore, for any trace \( \sigma \) bounded by \( b_{\text{vol}} \), we can map \( \sigma \) to a first order structure \((D_{AU}, o')\) where the domain size of \( |D_{AU}| \) is also bounded by \( b_{\text{vol}} \). Since MFOTL formulas specify constraints on \( \sigma, o' \) in \( D_{AU} \) should satisfy FOL formulas corresponding to these MFOTL constraints.

4.2.2 From MFOTL Formulas to FOL Formulas. Recall that the semantics of MFOTL formulas is defined over a trace \( \sigma = (\bar{D}, \bar{r}, v, o) \) at a time point \( i \) where \( \bar{D} = [D_1; D_2; ...] \) is a sequence of FO structures and \( \bar{r} = [r_1; r_2; ...] \) is a sequence of time values. The time value of the time point \( i \) is given by \( r_i \), and if \( i \) is not specified, then \( i = 1 \). The semantics of FOL formulas is defined on a single FO structure \((D_{AU}, o')\) where the time information is associated with relational objects in the domain \( D_{AU} \). Therefore, the time point \( i \) (and its time value \( r_i \)) needs to be considered during the translation from MFOTL to FOL since the same MFOTL formula at different time points represents different constraints over the trace \( \sigma \). Formally, our translation function \( \text{translate} \), abbreviated to \( T \) for succinctness, translates an MFOTL formula \( \phi \) into a function \( f : \bar{r} \rightarrow \phi_f \), where \( \phi_f \) is an FOL formula over relational objects. The FOL formula \( \phi_f \) corresponding to the MFOTL formula \( \phi \) at time step \( i \) is \( f = T(\phi, r_i) \). The translation rules for \( T \) are as follows:

\[ T(t = t', r_i) \rightarrow t = t' \]
\[ T(t > t', r_i) \rightarrow t > t' \]
\[ T(r(t_1, ..., t_{|\iota|}, r_i)) \rightarrow \exists \theta (\arg_1, ..., \arg_{|\iota|}) : r \land \theta \land \forall \theta' (\arg_j = t_j) \land t_i = \iota\text{time} \]
\[ T(\phi, r_i) \rightarrow T(\phi, r_i) \]
\[ T(\phi \land \psi, r_i) \rightarrow T(\phi, r_i) \land T(\psi, r_i) \]
\[ T(\exists \theta : r(t_1, ..., t_{|\iota|}, r_i) \land \phi, r_i) \rightarrow \exists \theta : r : T(r(t_1, ..., t_{|\iota|}, \theta) \land \phi \land \theta \land \iota\text{time}) \]
\[ T(\bigvee \phi, r_i) \rightarrow T(\phi, r_i) \land \iota\text{time} \land (\iota\text{time} - n_i) \in I \]
\[ T(\exists t_i : \iota\text{time}, r_i) \rightarrow \exists t_i : \iota\text{time} \land (\iota\text{time} - t_i) \in I \land T(\phi, r_i) \land \text{and} \text{Vtime}'(t_i \land (\iota\text{time} - t_i) \in I) \]
\[ T(\phi \land \psi, r_i) \rightarrow T(\phi, r_i) \land T(\psi, r_i) \land \text{Vtime}'(t_i \land (\iota\text{time} - t_i) \in I) \land T(\phi, r_i) \]
\[ T(\phi, r_i) \rightarrow T(\phi, r_i) \]

Report, September 8th, 2022, Nick Feng, Lina Marsso, Mehrdad Sahetadeh, and Marsha Chechik
For a MFOTL formula $\phi$, the FOL formula $T(\phi)$ only contains quantifiers exclusively on relational objects.

**Example 8.** Consider a formula $\exp = \Box \forall d : (A(d) \implies \phi_{[5,10]}(B(d)))$ where $A$ and $B$ are unary relations. The translated FOL formula

$$\forall a : A \cdot (\exists b : B \cdot a.arg_1 = b.arg_1 \land a.time + 5 \leq b.time \leq a.time + 10)$$

Given an MFOTL property $\phi$ and a set of MFOTL requirements $\text{Reqs}$, a volume bound $b_{vol}$, the BSC problem can be solved by searching for a satisfying solution $\sigma'$ for the FOL formula $T(\phi) \land \phi \in \text{Reqs} \land T(\psi)$ in a domain $D_{AU}$ with at most $b_{vol}$ relational objects.

### 4.3 Checking MFOTL Satisfiability

Below, we define a naive procedure $\text{NBSC}$ (shown in Fig. 3) for checking satisfiability of MFOTL formulas translated into FOL. We then discuss the complexity of this naive procedure. Even though we do not use this naive procedure in this paper, its complexity corresponds to the upper-bound of our proposed approach in Sec. 5.

**Searching for a satisfying solution.** Let $T(\phi)$ be an FOL formula translated from an MFOTL formula $\phi$, and let $k$ be the maximum quantifier nesting in $T(\phi)$. $\text{NBSC}$ solves $T(\phi)$ via quantifier elimination. By Proposition 1, only relational objects can be quantified. Given a bound $b$, the number of relational objects in any satisfying solution of $T(\phi)$ should be at most $n$. Therefore, $\text{NBSC}$ eliminates quantifiers in $T(\phi)$ by assuming presence of $n$ relational objects and then iterates over them. For example, the formula $\forall r : R \cdot \psi(r)$ is converted to $\exists (r_1) \land \psi(r_2) \land \ldots \land \psi(r_n)$. After eliminating all quantifiers in the formula $T(\psi)$, $\text{NBSC}$ calls an SMT solver on the quantifier-free formula to obtain the satisfiability result.

**Complexity.** The size of the quantifier-free formula is $O(n^k)$, where $k$ is the maximum depth of quantifier nesting. Since the background theory used in $\phi$ is restricted to LIA, the complexity of solving the formula is NP-complete [32]. Since $\text{TRANSLATE}$ is linear in the size of the formula $\phi$, the complexity of our approach is dominated by solving the translated formula which is NP-complete with respect to the size of the formula, $n^k$.

### 5 INCREMENTAL SEARCH FOR BOUNDED COUNTEREXAMPLES

The naive BSC approach ($\text{NBSC}$) defined in Sec. 4.3 is inefficient for solving the translated FOL formulas given a larger bound $n$ since its quantifier elimination creates a quantifier-free formula of size $O(n^k)$ even if satisfiability may be determined using a smaller bound. Moreover, $\text{NBSC}$ cannot detect unbounded unsatisfiability, and cannot provide optimality guarantees on the volume of counterexamples which are important for establishing the proof of unbounded correctness and localizing faults [16], respectively. In this section, we propose an incremental procedure, IBSC, that can detect unbounded unsatisfiability and provide shortest counterexamples. An overview of IBSC is shown in Fig. 3.

IBSC maintains an under-approximation of the search domain and the FOL constraints. It uses the search domain to ground the FOL constraints, and an off-the-shelf SMT solver to determine satisfiability of the grounded constraints and the data domain constraints. It analyzes the SMT result and accordingly either expands the search domain, refines the FOL constraints, or returns an answer to the satisfiability checking problem (a counterexample $\sigma$, “bounded-UNSAT”, or “UNSAT”). The procedure continues until an answer is obtained or until the domain exceeds the bound $b_{vol}$, in which case a “bounded-UNSAT” answer is returned. If the bound increases, $\sigma$ with the minimal volume and the UNSAT value are preserved while bounded-UNSAT may turn into $\sigma$ or UNSAT.

In the following, we describe IBSC in more detail. We explain the key component of IBSC, computing over and under-approximation queries in Sec. 5.1, the algorithm itself in Sec. 5.2, and analysis of its soundness, completeness, and the solution optimality in Sec. 5.3.

### 5.1 Over- and Under-Approximation

IBSC searches for a counter-example in a FO domain $D_{AU}$ whose size is bounded by a given volume bound $b_{vol}$. Instead of directly searching inside $D_{AU}$, IBSC starts with an under-approximated domain $D_{AU1} \subseteq D_{AU}$ where every relational object in $D_{AU1}$ is in also $D_{AU}$. With $D_{AU1}$, we can create an over- and an under-approximation query to the bounded satisfiability problem. Such queries are used to check satisfiability of FO formulas with domain $D_{AU1}$. IBSC starts with a small domain $D_{AU1}$ and gradually expands it until either SAT or UNSAT is returned, or the domain size exceeds some limit (bounded-UNSAT).

**Over-approximation.** Let $\phi_f$ be an FOL formula with quantifiers over relational objects, and $D_{AU1}$ be an FO domain of relation objects. The procedure $\text{GROUND}(G)$ encodes $\phi_f$ into a quantifier-free FOL formula $\phi_g$ s.t. the unsatisfiability of $\phi_g$ implies the unsatisfiability of $\phi_f$. We call $\phi_g$ an over-approximation of $\phi_f$.

The procedure $G$ (Alg. 2) recursively traverses the AST of the input FOL formula. To eliminate the existential quantifier in $\exists r : \text{Cls} \cdot \text{body}(L:2)$, $G$ creates a new relational object $\text{new}	ext{R}$ (unconstrained) of class $\text{Cls}$ (L: 5), and replaces $r$ with $\text{new}	ext{R}$ in $\text{body}$ (L:6).
To eliminate the universal quantifier in \( \forall r : \text{Cls} \cdot \text{body} \) (L:8), \( G \) expands the quantifier into a conjunction of clauses where each clause is \( \text{body} [ r \leftarrow r_i ] \) (i.e., \( r \) is replaced by \( r_i \) in \( \text{body} \)) for each relational object \( r_i \) of class \( \text{Cls} \) in \( D_{AU1} \) (L:10). Intuitively, an existentially-quantified relational object is instantiated with a new relational object, and an universally quantified relational object is instantiated with every existing relational object of the same class in \( D_{AU1} \).

**Example 9.** Consider the following FOL formula:

\[
\text{exp}_f = \text{access} \cdot \text{Access} \cdot \text{(Vupdate : Update)}
\]

Let \( D_{AU1} = \{ \text{updates}, \text{update2} \} \). \( G(\text{exp}_f, D_{AU1}) \) instantiates \( \text{access} \cdot \text{Access} \) with a new relational object, \( \text{access1} \), and instantiates \( \text{Vupdate} : \text{Update} \) with two relational objects, \( \text{update1} \) and \( \text{update2} \) in \( D_{AU1} \). The grounded formula is

\[
\text{exp}_g = \text{access} \land
\]

\[
(\text{update1}) \land (\text{access1})(\text{update2}) \land (\text{update2} \geq \text{update1} + 5)
\]

Proposition 2. For every FOL formula \( \phi_f \) translated from MFOTL formula \( \phi \) and domain \( D_{AU1} \), the grounded formula \( \phi_g = G(\phi_f, D_{AU1}) \) is quantifier-free and contains a finite number of variables and terms.

**Lemma 1 (Over-approximation Query).** For an FOL formula \( \phi_f \), and a domain \( D_{AU1} \), if \( \phi_g = G(\phi_f, D_{AU1}) \) is UNSAT, then so is \( \phi_f \).

**Proof.** We prove Lemma 1 by contradiction. Suppose \( \phi_g \) is UNSAT but there exists a solution \( \sigma_f \) for \( \phi_f \) in some domain \( D_{AU} \) (\( D_{AU} \) may be different from \( D_{AU1} \)). We show that we can always construct a solution \( \sigma_g \) that satisfies \( \phi_g \), which causes a contradiction. First, we construct a solution \( \sigma_g' \) for \( \phi_g' = G(\phi_f, D_{AU}) \) from the solution \( \sigma_f \) (for \( \phi_f \)). Then, we construct a solution \( \sigma_g \) for \( \phi_g \) from the solution \( \sigma_g' \) for \( \phi_g' \).

We can construct a solution \( \sigma_g' \) for \( \phi_g' \) in \( D_{AU} \cup \text{NewRs} \) where NewRs are the new relational objects added by \( G \). The encoding of \( G \) uses the standard way for expanding universally quantified expression by enumerating every relation object in \( D_{AU} \) (L:10). For every existentially quantified expression, there exists some relation object \( r \in D_{AU} \) enabled by \( \sigma_f \) that satisfies the expression in \( \phi_f \), whereas \( \phi_g' \) contains a new relational object \( r' \in \text{NewRs} \) for satisfying the same expression (L:6). Let \( \sigma_f (r) = \sigma_g (r') \) for \( r \) and \( r' \), and then \( \sigma_g' \) is a solution to \( \phi_g' \).

To construct the solution \( \sigma_g \) for \( g = G(\phi_f, D_{AU}) \) from the solution \( \sigma_g' \) for \( \phi_g' = G(\phi_f, D_{AU}) \), we consider the expansion of universally quantified expression in \( \phi_f \) (L:8). For every relational objects in \( r \in D_{AU} \), \( G \) creates constraints (L:10) in \( \phi_g' \), but not in \( \phi_g \). On the other hand, for every relational objects in \( r' \in D_{AU1} \), \( D_{AU} \), \( G \) disables \( r' \) in the solution \( \sigma_g' \), i.e., \( \sigma_g (r') = \bot \). Therefore, the constraints instantiated by \( r' \) (at L:10) in \( \phi_g' \) are vacuously satisfied.

For every relational object \( r \in D_{AU1} \), we let \( \sigma_g (r) = \sigma_g' (r) \), and all shared constraints in \( \phi_g \) and \( \phi_g' \) are satisfied by \( \sigma_g \) and \( \sigma_g' \), respectively. Therefore, \( \sigma_g \) is a solution to \( \phi_g' \), contradiction.

**Under-approximation.** Let \( \phi_f \) be an FOL formula, and \( D_{AU1} \) be an FO domain. The over-approximation \( \phi_g = G(\phi_f, D_{AU1}) \) contains a set of new relational objects introduced by \( G \) (L:3), denoted by NewRs. Let NoNewR(\( \text{NewRs} \), \( D_{AU1} \)) be constraints that enforce that every new relational object \( r \) in NewRs be identical to some relational objects in \( D_{AU1} \). Formally,

\[
\text{NoNewR}(\text{NewRs} , D_{AU1}) = \bigwedge_{r \in \text{NewRs} \setminus D_{AU1}} (r = r').
\]

Let \( \phi^+_f = \phi_g \land \text{NoNewR}(\text{NewRs} , D_{AU1}) \). If \( \phi^+_f \) has a satisfying solution, then there is a solution for \( \phi_f \). We call \( \phi^+_f \) an under-approximation of \( \phi_f \) and denote the procedure for computing it by \( \text{UNDERAPPROX}(\phi_f, D_{AU1}) \).

**Example 10.** Consider the FOL formula \( \text{exp}_f \) in Ex. 9 and \( D_{AU1} = \{ \text{updates}, \text{update2}, \text{access1} \} \).

\[
G(\text{exp}_f, D_{AU1}) = \text{access2} \land
\]

\[
(\text{update1}) \land (\text{access2}) \land (\text{access22}) \land (\text{update2}) \land (\text{update22})
\]

**Lemma 2 (Under-approximation Query).** For an FOL formula \( \phi_f \), and a domain \( D_{AU1} \), let \( \phi_g = G(\phi_f, D_{AU1}) \) and \( \phi^+ = \text{UNDERAPPROX}(\phi_f, D_{AU1}) \). If \( \sigma \) is a solution to \( \phi^+_g \), then there exists a solution to \( \phi_f \).

**Proof.** By construction, \( \text{NoNewR}(\text{NewRs} , D_{AU1}) \) guarantees that if \( \sigma \) is a solution to \( \phi^+_g \) in the domain \( D_{AU1} \), there exists a solution \( \sigma' \) to \( \phi_g \) in the domain \( D_{AU1} \) \& NewRs. Since every relational object \( r \in D_{AU1} \) has been used to instantiate a universally-quantified expression (L:10 of Alg. 2), \( \sigma' \) is also a solution to \( \phi_f \).

Suppose for some domain \( D_{AU1} \), an over-approximation query \( \phi_g \) is satisfiable while the under-approximation query \( \phi^+_g \) is UNSAT. Then we cannot conclude satisfiability of \( \phi_f \) for \( D_{AU1} \). However, the solution to \( \phi_g \) provides hints on how to expand \( D_{AU1} \) to potentially obtain a satisfying solution for \( \phi_f \).

**Corollary 1 (Necessary relational objects).** For a satisfying FOL formula \( \phi_f \) and a domain \( D_{AU1} \), let \( \phi_g = G(\phi_f, D_{AU1}) \) and \( \phi^+_g = \text{UNDERAPPROX}(\phi_f, D_{AU1}) \). Suppose \( \phi_g \) is satisfiable, and \( \phi^+_g \) is UNSAT; then every solution to \( \phi_g \) contains some relational object mentioned in the formula \( \phi_g \) but not in \( D_{AU1} \).

Cor. 1 holds because \( \phi_g \) and \( \phi^+_g \) are over- and under-approximations of \( \phi_f \), respectively. Cor. 1 shows that any domain \( D_{AU1} \) that contains a satisfying solution to \( \phi_f \) must consider some relational objects from some solution to \( \phi_g \).

### 5.2 Counterexample-Guided Constraint Solving Algorithm

Let an MFOTL formula \( \phi \), a set of MFOTL requirements \( \text{Reqs} \), an optional volume bound \( b_{\text{out}} \), and optionally a set of FO data domain constraints \( \text{Tdata} \) be given. IBSC, shown in Alg. 1, searches for a solution \( \sigma \) to \( \phi \land \phi \in \text{Reqs} \psi \) (while respecting \( \text{Tdata} \)), bounded by \( b_{\text{out}} \). If no such solution is possible regardless of the bound, IBSC returns UNSAT. If no solution can be found within the given bound, but a solution may exist for a larger bound, then IBSC returns bounded-UNSAT. If \( b_{\text{out}} \) is not specified, IBSC will perform the search unboundedly until a solution or UNSAT is returned.

IBSC first translates \( \phi \) and every \( \psi \in \text{Reqs} \) into FOL formulas in \( \text{Reqs}_f \), denoted by \( \phi_f \) and \( \psi_f \), respectively. Then IBSC searches
for a satisfying solution to $\phi_f \land \forall \phi_f \in \text{Reqs}_f$ (denote as $\phi_p \land \text{Reqs}_p$) in the FO domain $D_{AU}$, of volume which is at most $b_{tol}$. Instead of searching in $D_{AU}$ directly, IBSC searches for a solution of $\phi_f \land 
abla \phi_f \in D_{AU}$, where $\text{Reqs}_f \subseteq \text{Reqs}_f$ and $D_{AU} \subseteq D_{AU}$. IBSC initializes $\text{Reqs}_f$ and $D_{AU}$ as empty sets (LL:3-4). Then, for the FOL formula $\phi_p \land \text{Reqs}_p$, IBSC creates an over- and under-approximation query $\phi_g (L:8)$ and $\phi_g^\downarrow$ (L:10), respectively (described in Sec. 5.1). IBSC first solves the over-approximation query $\phi_g$ by querying an SMT solver (L:14). If $\phi_g$ is unsatisfiable, then $\phi_p \land \text{Reqs}_p$ is unsatisfiable (Lemma 1), and IBSC returns UNSAT (L:12).

If $\phi_g$ is satisfiable, then IBSC solves the under-approximation query $\phi_g^\downarrow$ (L:11). If $\phi_g^\downarrow$ is unsatisfiable, then the current domain $D_{AU}$ is too small, and IBSC expands it (LL:15-22). This is because satisfiability of $\phi_g$ indicates the possibility of finding a satisfying solution after adding at least one of the new relational objects in the solution of $\phi_g$ to $D_{AU}$ (Cor. 1). The expansion aims to identify a minimum solution $\sigma_{min}$ for $\phi_g$ that requires the smallest number of relational objects (L:16). For every relational object $r \in \sigma_{min}$, $r'$ is added to $D_{AU}$ if it is not already in $D_{AU}$ (L:18). To obtain $\sigma_{min}$, we follow MaxRes [31] methods: we analyze the UNSAT core of $\phi_g^\downarrow$ and incrementally weaken $\phi_g^\downarrow$ towards $\phi_g$ (i.e., the weakened query $\phi_g^{\downarrow '}$) is an “under-approximation” that satisfies the condition $\phi_g^\downarrow \Rightarrow \phi_g^{\downarrow '}$ until a satisfying solution $\sigma_{min}$ is obtained for the weakened query. However, if the volume of $\sigma_{min}$ exceeds $b_{tol}$ (L:19), then bounded-UNSAT is returned (L:20).

On the other hand, if $\phi_g^\downarrow$ yields a solution $\sigma$, then $\sigma$ is checked on $\text{Reqs}_f$ (L:23). If $\sigma$ satisfies every $\phi_f$ in $\text{Reqs}_f$, then $\sigma$ is returned (L:24). If $\sigma$ violates some requirements in $\text{Reqs}_f$, then the violating requirement $\text{Reqs}_f$ is added to $\text{Reqs}_f$ to be considered in the search for the next iterations (L:28).

If IBSC does not find a solution or does not return UNSAT, it means that no solution is found because the $D_{AU}$ is too small or the $\text{Reqs}_f$ are too weak. IBSC then starts again with either the expanded domain $D_{AU}$ or the refined set of requirements $\text{Reqs}_f$. It computes the over- and under-approximation queries ($\phi_g$ and $\phi_g^\downarrow$) again and repeats the steps. Due to structural similarities of $\phi_g$ and $\phi_g^\downarrow$ between iterations, incremental encoding and solving are enabled to allow SMT solvers to internally reuse learned clauses from previous iterations.

Example 11. Consider the DCC example in Sec. 2. Suppose IBSC is invoked to find a counterexample for property P1 (shown in Fig. 1) subject to requirements $\text{Reqs} = \{\text{req1}, \text{req2}\}$ with the bound $b_{tol} = 4$. IBSC translates the requirements and the property to FOL, and initializes $\text{Reqs}_f$ and $D_{AU}$ to empty sets. For each iteration, we use $\phi_p$ and $\phi_g$ to represent the over- and under-approximation queries computed on LL:8-10, respectively.

1st iteration: $D_{AU}$ = $\emptyset$ and $\text{Reqs}_f$ = $\emptyset$. $\phi_p$ introduces three new relational objects (from -P1): $\text{access}_1, \text{collect}_1$ and $\text{update}_1$ such that: (C1) $\text{access}_1$ occurs after $\text{collect}_1$ and $\text{update}_1$; (C2) $\text{access}_1, \text{arg}1 = \text{collect}_1, \text{arg}1 = \text{update}_1, \text{arg}1 \in (C3) \text{access}_2, \text{arg}2 \in \text{collect}_2, \text{arg}2 \in \text{access}_2, \text{arg}2 \notin \text{update}_1, \text{arg}2$; and (C4) either $\text{collect}_1$ or $\text{update}_1$ must be present in the counterexample. $\phi_g$ is satisfiable, but $\phi_g^\downarrow$ is UNSAT since $D_{AU}$ is an empty set. WLOG, we assume $D_{AU}$ is expanded by adding $\text{access}_1$ and $\text{update}_1$.

Algorithm 1 IBSC: search for a bounded (by $b_{tol}$) solution to $T(\phi) \land \forall \phi \in \text{Reqs}, T(\psi)$.

\begin{algorithm}
\caption{IBSC search for a bounded (by $b_{tol}$) solution to $T(\phi) \land \forall \phi \in \text{Reqs}, T(\psi)$.}
\begin{algorithmic}[1]
\State \textbf{Input} an MFOTL formula $\phi$.
\State \textbf{Input} a set of MFOTL requirements $\text{Reqs} = \{\psi_1, \psi_2, ...\}$. 
\State \textbf{Optional Input} $b_{tol}$, the volume bound of the counterexample.
\State \textbf{Optional Input} data constraints $T_{data}$, default $\emptyset$.
\State \textbf{Output} a counterexample $\sigma$, UNSAT or bounded-UNSAT.
\State $\text{Reqs}_f \leftarrow \{ \psi_f = T(\psi) \mid \psi \in \text{Reqs} \}$
\State $\phi_f \leftarrow T(\phi)$
\State $\sigma \leftarrow 0$ //Initially do not consider any requirement
\State $D_{AU} \leftarrow 0$ //Start with empty set of relational objects
\While {$T$}
\State $\phi \leftarrow \phi_f \land \text{Reqs}_f$
\State //over-approximation query
\State $\phi_g \leftarrow G(\phi, D_{AU})$
\State //under-approximation query
\State $\phi_g^\downarrow \leftarrow \text{UNDERRANGE}(\phi, D_{AU})$
\If {$\text{SOLVE}(\phi_g \land T_{data}) = \text{UNSAT}$}
\State return UNSAT
\EndIf
\If {$\sigma \leftarrow \text{SOLVE}(\phi_g^\downarrow \land T_{data})$}$
\If {$\sigma = \text{UNSAT}$} //Expand $D_{AU}$
\State $\sigma_{min} \leftarrow \text{MINIMIZE}(\phi_g)$
\State //add relational object from the minimal solution
\State $D_{AU} \leftarrow D_{AU} \cup \{act \mid act \in \sigma_{min}\}$
\If {$\text{vol}(\sigma_{min}) > b_{tol}$}
\State return bounded-UNSAT
\EndIf
\EndIf
\Else //Check all requirements
\If {$\sigma \models \psi$ for every $\psi \in \text{Reqs}_f$}
\State return $\sigma$
\Else //Violating requirement $\text{Reqs}_f$
\State lesson $\leftarrow$ some $\psi \in \text{Reqs}_f$ such that $\sigma \not\models \psi$
\State //Add violating requirement to $\text{Reqs}_f$
\State $\text{Reqs}_f \leftarrow \text{Reqs}_f \cup \text{lesson}$
\EndIf
\EndIf
\EndWhile
\end{algorithmic}
\end{algorithm}

$\text{Algorithm 2 G:}$ ground a quantified FOL formula.

\begin{algorithm}
\caption{G: ground a quantified FOL formula.}
\begin{algorithmic}[1]
\State \textbf{Input} an FOL formula $\phi_f$.
\State \textbf{Input} a domain of relational objects $D_{AU}$.
\State \textbf{Output} a grounded quantifier-free formula $\phi_g$ over relational objects.
\State $\text{if is\_ATOM}(\phi_f)$ then return $\phi_f \text{ end if}$
\State $\text{if } \phi_f \cdot \text{op} = \exists \text{ then } //\text{process the existential operator}$
\State $\text{Cl} \leftarrow \phi_f \cdot \text{class}$
\State $\text{newR} \leftarrow \text{NewAct(Cls)}$
\State $\text{return } G(\phi_f \cdot \text{body}[\phi_f \cdot \text{headAct} \leftarrow \text{newR}], D_{AU})$
\State $\text{end if}$
\State $\text{if } \phi_f \cdot \text{op} = \forall \text{ then } //\text{process the universal operator}$
\State $\text{Cl} \leftarrow \phi_f \cdot \text{class}$
\State $\text{return } \forall \{r \cdot \text{Cls} \mid D_{AU} \} \Rightarrow (G(\phi_f \cdot \text{body}[\phi_f \cdot \text{head} \leftarrow r], D_{AU})$
\State $\text{end if}$
\State $\text{return } \phi_f \cdot \text{op} G(\phi_f \cdot \text{child}, D_{AU}) \text{ for } \phi_f \cdot \text{child} \text{ in } \phi_f \cdot \text{body}$
\end{algorithmic}
\end{algorithm}

2nd iteration: $D_{AU} = \{\text{access}_1, \text{update}_1\}$ and $\text{Reqs}_f = 0$. $\phi_g$ stays the same, and $\phi_g^\downarrow$ is now satisfiable since $\text{access}_1$ and $\text{update}_1$
Theorem 1 (Soundness). Since \( \phi' \) is satisfiable, but \( \phi'' \) is unsatisfiable, \( \phi'' \) is UNSAT iff \( \sigma' \) satisfies \( \phi' \). Thus, \( \phi'' \) is UNSAT because \( \sigma' \) is satisfiable.

Proof. Let \( \phi' \) be the FOL formula \( T(\phi) \land \psi \in \text{Reqs} T(\psi) \). We consider correctness of IBSC for three possible outputs: the satisfying solution \( \sigma \) to \( \phi_f \) (L:24), the UNSAT determination of \( \phi_f \) (L:12), and the bounded-UNSAT determination of \( \phi_f \) (L:20).

IBSC returns a satisfying solution \( \sigma \) only if (1) \( \sigma \) is a solution to \( \phi_f \) (L:24), (2) \( \sigma \) intersects \( \text{Reqs} \) (L:23). By (1) and Lemma 2, \( \sigma \) is a solution to \( T(\phi) \land \psi \in \text{Reqs} T(\psi) \). Together with (2), \( \sigma \) is a solution to \( \phi_f \).

IBSC returns UNSAT iff \( \phi'' \) is UNSAT (L:11). By Lemma 1, we show \( T(\phi) \land \psi \in \text{Reqs} T(\psi) \) is UNSAT. Since \( \text{Reqs} \subseteq \text{Reqs} \), the original formula \( \phi_f \) is also UNSAT.

IBSC returns bounded-UNSAT iff the volume of the minimum solution \( \sigma_{\text{min}} \) to the over-approximated query \( \phi_f \) is larger than \( b_{\text{vol}} \) (L:19). Since \( \phi_f \) is an over-approximation of the original formula \( \phi_f \), any solution \( \sigma \) to \( \phi_f \) has volume at least \( \text{vol}(\sigma_{\text{min}}) \). Therefore, when \( \text{vol}(\sigma_{\text{min}}) > b_{\text{vol}} \), \( \text{vol}(\sigma) > b_{\text{vol}} \) for every solution.

Theorem 2 (Termination). For an input property \( \phi \), requirements \( \text{Reqs} \), and a bound \( b_{\text{vol}} \neq \infty \), IBSC eventually terminates.

Proof. To prove that IBSC always terminates when the input \( b_{\text{vol}} \neq \infty \), we need to show that IBSC does not get stuck at solving the SMT query via solve (LL:14-11), nor refining \( \text{Reqs} \) (LL:23-29), nor expanding \( D_{AU1} \) (LL:18-22).

A call to solve (LL:14-11) always terminates. By Prop. 1, \( T(\phi) \) may contain quantifiers exclusively over relational objects. By Prop. 2 both the under- and the over-approximated queries \( \phi_g \) and \( \phi'_{g'} \) are quantifier-free. Since the background theory for \( \phi \) is LIA, then \( \phi_g \) and \( \phi'_{g'} \) are quantifier-free LIA formula whose satisfiability is decidable.

If the requirement checking fails on L:23, a violating requirement \( \text{lesson} \) is added to \( \text{Reqs} \) (LL:26-28) which ensures that any future solution \( \sigma' \) satisfies \( \text{lesson} \). Therefore, \( \text{lesson} \) is never added to \( \text{Reqs} \) more than once. Given that \( \text{Reqs} \) is a finite set of MFOTL formulas, at most \( |\text{Reqs}| \) lessons can be learned before the algorithm terminates.

The under-approximated domain \( D_{AU1} \) can be expanded a finite number of times because the size of the minimum solution \( \text{vol}(\sigma_{\text{min}}) \) to \( \phi_g \) (computed on L:16) is monotonically non-decreasing between each iteration of the loop (LL:5-31). The size will eventually increase since each relational object in \( D_{AU1} \) can introduce a finite number of options for adding a new relational object through the grounded encoding of \( \phi_g \) on L:10, e.g., \( r = \forall \sigma \exists t_r \). After exploring all options to \( D_{AU1} \), \( \text{vol}(\sigma_{\text{min}}) \) must increase if the algorithm has not already terminated. Therefore, if \( b_{\text{vol}} \neq \infty \), then eventually \( \text{vol}(\sigma_{\text{min}}) > b_{\text{vol}} \), and the algorithm will return bounded-UNSAT instead of expanding \( D_{AU1} \) indefinitely (LL:15-22).

Theorem 3 (Solution optimality). For a property \( \phi \) and requirements \( \text{Reqs} \), let \( \phi_f \) be the FOL formula \( T(\phi) \land \forall \psi \in \text{Reqs} T(\psi) \). If IBSC finds a solution \( \sigma \) for \( \phi_f \), then for every \( \sigma' \models \phi_f \), \( \text{vol}(\sigma') \leq \text{vol}(\sigma) \).

Proof. IBSC returns a solution \( \sigma \) on L:24 only if \( \sigma \) is a solution to the under-approximation query \( \phi'_{g'} \) (computed on L:10) for some domain \( D_{AU1} \neq \emptyset \). \( D_{AU1} \) is last expanded in some previous iterations by adding relational objects to the minimum solution \( \sigma_{\text{min}} \) (L:16) of the over-approximation query \( \phi_g \) (L:18). Therefore, the returned \( \sigma \) has the same number of relational objects as \( \sigma_{\text{min}} \) (\( \text{vol}(\sigma_{\text{min}}) = \text{vol}(\sigma) \)). Since \( \phi_g \) is an over-approximation of the original formula \( \phi_f \), any solution \( \sigma' \) to \( \phi_f \) has volume that is at least \( \text{vol}(\sigma_{\text{min}}) \). Therefore, \( \text{vol}(\sigma) \leq \text{vol}(\sigma') \).

Remark. IBSC finds the optimal solution because it always looks for the minimum solution \( \sigma_{\text{min}} \) to the over-approximation query \( \phi_g \) (L:16) and uses it for domain expansion (L:18). However, looking for the minimum solution adds cost. If the solution optimality is not required, IBSC can be configured to heuristically find a solution \( \sigma_f \) to \( \phi_f \) such that \( \text{vol}(\sigma_f) \leq b_{\text{vol}} \). For example, greedy best-first search (gBFS) finds a solution to \( \phi_f \) that minimizes the number of relational objects that are not already in \( D_{AU1} \) and then uses it to expand \( D_{AU1} \). We configured a non-optimal version of IBSC (nIBSC) that uses gBFS heuristics, and evaluate its performance in Sec. 6.
To evaluate our approach, we developed LEGOS, a tool that implements our MFOTL bounded satisfiability checking algorithm (Algorithm 1). It includes Python API for specifying system requirements and MFOTL safety properties. We use pySMT [15] to formulate SMT queries and Z3 [10] to check their satisfiability. LEGOS and the evaluation artifacts are included in supplementary material. In this section, we evaluate the effectiveness of our approach using five case studies, aiming to answer the following research question (RQ): How effective is our approach at determining the bounded satisfiability of MFOTL formulas? We measure effectiveness in terms of the ability to determine satisfiability (i.e., the satisfying solution and its volume, UNSAT, or bounded UNSAT), and performance, i.e., time and memory usage.

**Cases studies.** The case studies shown in Tab. 1, are summarized below: (1) PHIM (derived from [1, 13]): a computer system for keeping track of personal health information with cost management; (2) CF@H\(^2\): a system for monitoring a COVID patients at home and enabling doctors to monitor patient data; (3) PBC [5]: an approval policy for publishing business reports within a company; (4) BST [3]: a banking system that processes customer transactions; and (5) NASA [14, 30]: an automated air-traffic control system design that aims to avoid aircraft collisions.\(^3\) Case studies were selected for (i) the purpose of comparison with existing works (i.e., NASA); (ii) checking whether our approach scales with case studies involving data/time constraints (PBC, BST, PHIM and CF@H); or (iii) evaluating the applicability of our approach with real-world case studies (CF@H and NASA). In addition to prior case studies, we include PHIM and CF@H which have complex data/time constraints. The number of requirements for the five case studies ranges between ten (BST) and 194 (NASA). The number of relations present in the MFOTL requirements ranges from three (BST) to 28 (CF@H), and the number of arguments in these relations ranges from 1 (PHIM, PBC, and BST) to 79 (NASA). Examples of properties and requirements considered in this evaluation are shown in Fig. 1.

**Experimental setup.** Given a set of requirements, data constraints and properties of interest for each case study, we measured the runtime (time) and peak memory usage (mem.) of performing bounded satisfiability checking of MFOTL properties, and the volume \(vol_e\) (the number of relational objects) of the solution (\(\sigma\)) with (IBSC) and without (nIBSC) the optimality guarantees (see Remark 5.3 for finding non-optimal solutions). We conduct two experiments: the first one evaluates the efficiency and scalability of our approach; the second one compares our approach with satisfiability checking. Since there is no existing work for checking MFOTL satisfiability, we compared with LTL satisfiability checking because MFOTL subsumes LTL. To study the scalability of our approach, our first experiment considers four different configurations obtained by increasing the data domain constraints of the case studies requirements. The initial configuration (small) is described in Tab. 1. The medium and large configurations are obtained by multiplying the initial data constraints and volume bound by ten and hundred, respectively. The last (unbounded) configuration does not bound either the data domain or the volume. To study performance of our approach relative to existing work, our second experiment considers two configurations of the NASA case study verified in [28] using the state of the art symbolic model checker nuXmv [8]. We compare our approach’s result against the reproduced result of nuXmv verification. For both experiments, we report the analysis outcomes, i.e., the satisfying solution and its volume, UNSAT, or bounded UNSAT; and performance, i.e., time and memory usage.

The experiments were conducted using a ThinkPad X1 Carbon with an Intel Core i7 1.80 GHz processor, 8 GB RAM, and running 64-bit Ubuntu GNU/Linux 8.

**Results of the first experiment** are summarized in Tab. 2. Out of the 72 trials, our approach found 31 solutions. It also returned five bounded-UNSAT answers, and 36 UNSAT answers. The results show that our approach is effective in checking satisfiability of case studies with different size. More precisely, we observe that it takes under three seconds to return UNSAT and between .04 seconds (bs3;medium) and 32 minutes (ph7;medium;IBSC) to return a solution. In the worst case, IBSC took 32 minutes for checking ph7 where the property and requirements contain complex constraints. Effectively, ph7 requires the deletion of data stored at id 10, while PHIM requires that the cost of deletion increases over time. Therefore, the user has to perform a number of actions to obtain a sufficient balance to delete the data. Additionally, each action that increases the user’s balance has its own preconditions, effects, and time cost, making the process of choosing the sequence of actions to meet the increasing deletion cost non-trivial.

We can see a difference in time between cf2 ‘big’ and ‘unbounded’, this is because the domain expansion followed two different paths and one produces significantly easier SMT queries. Since our approach is guided by counter-examples (i.e., the path is guided by the solution from the SMT solver (Alg. 1-L:16)), our approach does not have direct control over the exact path selection. In future work, we aim to add optimizations to avoid/backtrack from hard paths.

We observe that the data domain constraint and volume bound used in different configurations do not affect the performance of IBSC when the satisfiability of the instances does not depend on them which is the case for all the instances except for ph0–7;small, cf1–3 : small, and bs3;small. As mentioned in Sec. 4, the data domain constraint ensures that satisfying solutions have realistic data values. For ph1 – ph4, the bound used in small, medium and large

| Case study | #reqs | #rels | #args | #prop | Configuration values |
|------------|-------|-------|-------|-------|----------------------|
| PHIM       | 18    | 22    | 13    | 1     | \(x_1 = 2, x_2 = 4, x_3 = 5\) |
| CF@H       | 45    | 28    | 12    | 1     | \(x_1 = 2, x_2 = 10\)  |
| PBC        | 14    | 7     | 1     | 1     | \(x_1 = 5, x_2 = 10\)  |
| BST        | 10    | 3     | 1     | 3     | \(x_1 = 1, x_2 = 2, x_3 = 4, sup = 10\) |

\(^*\)available at https://github.com/agithubuserseva/IBSC  
\(^\d\)https://covidfreeathome.org/  
\(^\dd\)The requirements and properties for the NASA case study are originally expressed in LTL, which is subsumed by MFOTL.
Table 2: Run-time performance for four cases and 18 properties. We record the outcome (out.) of the algorithm with IBSC or without (nIBSC) the optimal solution guarantee: UNSAT (U), bounded-UNSAT (b-U), or the volume of the counterexample (a natural number, corresponding to vol.). We consider four different configurations: small (see Tab. 1) medium (x10), big (x100), and unbounded (∞) data constraints and volume bound. The different vol. between IBSC and nIBSC are in bold.

Table 3: Run-time performance of our approach (IBSC) and the state of the art model checker (nuXmv) on the NASA case study. The configurations considered are described in Tab. 1. The configurations considered are modifying the number of ground-separated (#GSEP) and of self-separating aircraft (#SSEP) as follows: configuration 1: #GSEP = 3 and #SSEP = 0; and configuration 2: #GSEP = 2 and #SSEP = 2.

configurations creates additional constraints in the SMT queries for each relational object, and therefore results in a larger peak memory than the unbounded configuration.

Finding the optimal solution (by IBSC), in contrast to finding a satisfying solution without the optimal guarantee (by nIBSC), imposes a substantial computational cost while rarely achieving a volume reduction. The non-optimal heuristic nIBSC often outperforms the optimal approach for satisfiable instances. Out of 31 satisfiable instances, nIBSC solved 12 instances 3 times faster, 10 instances 10 times faster and seven instances 20 times faster than IBSC. Compared to the non-optimal solution, the optimal solution reduced the volume for only two instances: ph7-large and ph7:unbounded by one (3%) and three (9%), respectively. On all other satisfying instances, IBSC and nIBSC both find the optimal solutions. When there is no solution, both IBSC and nIBSC are equally efficient.

Results of the second experiment are summarized in Tab. 3. Our approach and nuXmv both correctly verified that all six properties were UNSAT in both NASA configurations. We observe that performance of our approach is comparable to nuXmv for the first configuration with 10 to 20 seconds of difference on average. Yet, for the second configuration, our approach terminates in less than 0.20 sec and nuXmv takes 1.50 seconds on average. We conclude that our approach’s performance is comparable to that of nuXmv for LTL satisfiability checking even though our approach is not specifically designed for LTL.

Summary. In summary, we have demonstrated that our approach is effective at determining the bounded satisfiability of MFOOTL formulas using case studies with different sizes and from different application domains. When restricted to LTL, our approach is at least as effective as the existing work on LTL satisfiability checking which uses the state of the art symbolic model checker. Importantly, IBSC can often determine satisfiability of instances without reaching the volume bound, and its performance is insensitive to data domain. A current limitation of IBSC is its inability to directly support constraint over aggregated values in the input formulas. As a walk-around, aggregated values are captured using auxiliary predicates (e.g., the balance of a user in ph7) and data constraints to simulate the aggregation process (e.g., how the balance is updated with respect to relevant data actions). However, the evaluation results shows that the walk-around costs the efficiency of IBSC. Thus, we need to study how to extend IBSC to efficiently handle aggregation. We also observed that IBSC’s optimal guarantee imposes a substantial computational cost while rarely achieving a volume reduction on non-optimal solutions obtained by nIBSC. We need to investigate the trade-off between optimality and efficiency, as well as evaluate performance of IBSC on a broader range of benchmarks.

7 Related Work

Below, we compare with the existing approaches for verifying legal properties as well as with the approaches that address the satisfiability checking of temporal logic and first-order logic.
Legal compliance. Various techniques have been proposed for operationalizing legal provisions and ensuring that systems comply with these provisions, e.g., [7, 19, 22, 39]. MFOTL has been used in the context of runtime policy monitoring and enforcement, e.g., for monitoring security policies [3] and data privacy rules [1]. While our work is based on the same logic, our intended use case and the type of reasoning we perform are different: we use MFOTL to specify requirements and properties and then look for legal-property violations. Crucially, while previous work requires a system implementation, our approach enables early validation of legal property compliance by checking the requirements.

Satisfiability checking of temporal properties. Temporal logic satisfiability checking has been studied for the verification of system designs. Satisfiability checking for Linear Temporal Logic (LTL) can be performed by reducing the problem to model checking [37], by applying automata-based techniques [29], or by SAT solving [6, 25–27]. Satisfiability checking for metric temporal logic (MTL) [34] and its variants, e.g., mission-time LTL [28] and signal temporal logic [2], has been studied for the verification of real-time system designs. These existing techniques are inadequate for our needs: LTL and MTL cannot effectively capture quantified data constraints commonly used in legal properties. MFOTL does not have such a limitation as it extends MTL and LTL with first-order quantifiers, thereby supporting the specification of data constraints.

Finite model finding for first-order logic. Finite-model finders [9, 36] look for a model by checking universal quantifiers exhaustively over candidate models with progressively larger domains; we look for finite-volume solutions using a similar approach. On the other hand, we consider an explicit bound on the volume of the solution, and are able to find the solution with the smallest volume. SMT solvers support quantifiers with quantifier instantiation heuristics [17, 18] such as E-matching [11, 12] and conflict-based instantiation [35]. Quantifier instantiation heuristics are nonetheless generally incomplete, whereas, in our approach, we obtain completeness by bounding the volume of the satisfying solution.

8 CONCLUSION

In this paper, we proposed an incremental bounded satisfiability checking approach, called IBSC, aimed to enable verification of legal properties, expressed in MFOTL, against system requirements. IBSC first translates MFOTL formulas to first-order logic (FOL) and then searches for a satisfying solution to the translated FOL formulas in a bounded search space by deriving over- and under-approximating SMT queries. IBSC starts with a small search space and incrementally expands it until an answer is returned or until the bound is exceeded. We implemented IBSC on top of the SMT solver Z3. Experiments using five case studies showed that our approach was effective for identifying errors in requirements from different application domains.

To the best of our knowledge, we are the first to study satisfiability checking for MFOTL. Our approach is meant for use in early-stage system development and to support building systems that comply to legal properties before detailed behavioural models or a system implementation become available.

Our approach is currently limited to verifying safety properties. In the future, we plan to extend our approach so that it can handle broader property types, including liveness and fairness. IBSC’s performance and scalability depend crucially on how the domain of relational objects is maintained and expanded (currently, this is done using two heuristics). As a future work, we would like to study the effectiveness of other heuristics to improve IBSC’s scalability (e.g., random restart, expansion with domain-specific heuristics). We also aim to study how to learn/infer MFOTL properties during the search to further improve the efficiency of our approach.

REFERENCES

[1] Emma Arfelt, David A. Basin, and Sören Debois. Monitoring the GDPR. In Kaze Sako, Steve A. Schneider, and Peter Y. A. Ryan, editors, Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I, volume 11735 of Lecture Notes in Computer Science, pages 681–699. Springer, 2019.
[2] Kyungmin Bae and Jia Lee. Bounded model checking of signal temporal logic properties using syntactic separation. Proc. ACM Program. Lang., 3(POPL):51:1–51:30, 2019.
[3] David A. Basin, Felix Klaedtke, and Samuel Müller. Monitoring security policies with metric first-order temporal logic. In James B. Doshi and Barbara Carmignani, editors, Proceedings of the 15th Symposium on Access Control Models and Technologies (SACMAT’2010), Pittsburgh, Pennsylvania, USA, pages 23–34. ACM, 2010.
[4] David A. Basin, Felix Klaedtke, and Samuel Müller. Policy monitoring in first-order temporal logic. In Taysir Touili, Byron Cook, and Paul B. Jackson, editors, Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings, volume 6174 of Lecture Notes in Computer Science, pages 1–18. Springer, 2010.
[5] David A. Basin, Felix Klaedtke, Samuel Müller, and Eugen Zălinescu. Monitoring metric first-order temporal properties. J. ACM, 62(2):15:1–15:45, 2015.
[6] Marcello M. Berti, Achille Frigeri, Angelo Morzenti, Matteo Pradella, Matteo Rossi, and Pierluigi San Pietro. Constraint LTL satisfiability checking without automata. J. Appl. Log., 12(4):522–557, 2014.
[7] Travis D. Breaux, Matthew W. Vail, and Annie I. Antón. Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations. In 14th IEEE International Conference on Requirements Engineering (RE 2006), 11-15 September 2006, Minneapolis/St.Paul, Minnesota, USA, pages 46–55. IEEE Computer Society, 2006.
[8] Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In CAV, pages 334–342. 2014.
[9] Koen Claessen and Niklas Sörensson. New techniques that improve mace-style finite model finding. In Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications, pages 11–27. Citeseer, 2003.
[10] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In International conference on Tools and Algorithms for the Construction and Analysis of Systems, pages 337–340. Springer, 2008.
[11] Leonardo Mendonça de Moura and Nikolaj Bjørner. Efficient e-matching for SMT solvers. In Frank Pfenning, editor, Proceedings of the 21st International Conference on Automated Deduction (CADE 2007), Bremen, Germany, volume 4605 of Lecture Notes in Computer Science, pages 183–198. Springer, 2007.
[12] David Detlefs, Greg Nelson, and James B. Saxe. Simplify: a theorem prover for program checking. J. ACM, 52(3):365–473, 2005.
[13] Nick Feng, Lina Marsso, and Hubert Garavel. Health record. Model checking program checking context model (MCC’21), Dept. of Computer Science - University of Toronto, 2021.
[14] Marco Gario, Alessandro Cimatti, Cristian Mattarei, Stefano Tonetta, and Kristin Yvonne Rozier. Model checking at scale: Automated air traffic control design space exploration. In Swarat Chaudhuri and Azadeh Farzan, editors, Proceedings of the 28th International Conference on Computer Aided Verification (CAV’2016), Toronto, ON, Canada, volume 9780 of LNCS, pages 3–22. Springer, 2016.
[15] Marco Gario and Andrea Micheli. Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In SMT Workshop 2015, 2015.
[16] Paul Gastin, Pierre Moro, and Marc Zeitoun. Minimization of counterexamples in SMT. In Susanne Graf and Laurent Mounier, editors, Proceedings of the 11th International Workshop on Model Checking Software (SPiN’04), Barcelona, Spain, volume 2989 of Lecture Notes in Computer Science, pages 92–108. Springer, 2004.
[17] Yetian Ge, Clark W. Barrett, and Cesare Tinelli. Solving quantified verification conditions using satisfiability modulo theories. In Frank Pfenning, editor, Proceedings of the 21st International Conference on Automated Deduction (CADE’2007), Bremen, Germany, volume 4603 of Lecture Notes in Computer Science, pages 167–182. Springer, 2007.
[18] Yeting Ge and Leonardo Mendonça de Moura. Complete instantiation for quantified formulas in satisfiability modulo theories. In Ahmed Bouajjani and Oded Maler, editors, Proceedings of the 21st International Conference on Computer Aided Verification (CAV’2009), Grenoble, France, volume 5643 of Lecture Notes in Computer Science, pages 396–410. Springer, 2009.

[19] Sepideh Ghanavati, Daniel Amyot, and Liam Peyton. Towards a framework for tracking legal compliance in healthcare. In Johan Kröger, Andreas L. Opdahl, and Guttorm Sindre, editors, Advanced Information Systems Engineering, 19th International Conference, CAiSE 2007, Trondheim, Norway, June 11-15, 2007, Proceedings, volume 4495 of Lecture Notes in Computer Science, pages 218–232. Springer, 2007.

[20] Sylvain Hallé and Roger Villemaire. Runtime enforcement of web service message contracts with data. IEEE Trans. Serv. Comput., 5(2):192–206, 2012.

[21] Ian M. Hodkinson. Loosely guarded fragment of first-order logic has the finite model property. Stud. Logica, 70(2):205–240, 2002.

[22] Shareeful Islam, Haralambos Mouratidis, and Stefan Wagner. Towards a framework to elicit and manage security and privacy requirements from laws and regulations. In Roel J. Wieringa and Anne Persson, editors, Proceedings of the 16th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ’2010), Essen, Germany, volume 6182 of LNCS, pages 255–261. Springer, 2010.

[23] Daniel Jackson. Alloy: a lightweight object modelling notation. ACM Trans. Softw. Eng. Methodol., 11(2):256–290, 2002.

[24] Legislative Assembly of Ontario. Personal health information protection act (PHIPA), 2004. https://www.ontario.ca/laws/statute/04p03.

[25] Jianwen Li, Geguang Pu, Lijun Zhang, Moshe Y. Vardi, and Jifeng He. Accelerating LTL satisfiability checking by SAT solvers. J. Log. Comput., 28(6):1011–1030, 2018.

[26] Jianwen Li, Geguang Pu, Yueling Zhang, Moshe Y. Vardi, and Kristin Y. Rozier. Sat-based explicit ltl satisfiability checking. Artif. Intell., 289:103369, 2020.

[27] Jianwen Li, Kristin Y. Rozier, Geguang Pu, Yueling Zhang, and Moshe Y. Vardi. Sat-based explicit ltl satisfiability checking. In The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019, pages 2946–2953. AAAI Press, 2019.

[28] Jianwen Li, Moshe Y. Vardi, and Kristin Y. Rozier. Satisfiability checking for mission-time LTL. In Isil Dillig and Serdar Tasiran, editors, Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II, volume 11562 of Lecture Notes in Computer Science, pages 3–22. Springer, 2019.

[29] Jianwen Li, Lijun Zhang, Geguang Pu, Moshe Y. Vardi, and Jifeng He. LTL satisfiability checking revisited. In Proceedings of the 20th International Symposium on Temporal Representation and Reasoning, Pensacola, FL, USA, 2013, pages 91–98. IEEE Computer Society, 2013.

[30] Cristian Mattarei, Alessandro Cimatti, Marco Gario, Stefano Tonetta, and Kristin Y. Rozier. Comparing different functional allocations in automated air traffic control design. In Roope Kavola and Thomas Wald, editors, Formal Methods in Computer-Aided Design (FMCAD'2015), Austin, Texas, USA, pages 112–119. IEEE, 2015.

[31] Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided maxsat resolution. In Carla E. Brodley and Peter Stone, editors, Proceedings of the 28th International Conference on Artificial Intelligence (AAAI'14), Quebec City, Canada, pages 2717–2723. AAAI Press, 2014.

[32] Christos H. Papadimitriou. On the complexity of integer programming. J. ACM, 28(4):755–768, 1981.

[33] Emil L. Post. Recursive unsolvability of a problem of thue. J. Symb. Log., 12(1):1–11, 1947.

[34] Matteo Pradella, Angelo Morzenti, and Pierluigi San Pietro. Bounded satisfiability of metric temporal logic specifications. ACM Trans. Softw. Eng. Methodol., 22(3):20:1–20:54, 2013.

[35] Andrew Reynolds, Cesare Tinelli, and Leonardo Mendonça de Moura. Finding conflicting instances of quantified formulas in SMT. In Formal Methods in Computer-Aided Design (FMCAD'2014), Lausanne, Switzerland, pages 195–202. IEEE, 2014.

[36] Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krstic, Morgan Deters, and Clark W. Barrett. Quantifier instantiation techniques for finite model finding in SMT. In Maria Paola Bonacina, editor, Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013, Proceedings, volume 7898 of Lecture Notes in Computer Science, pages 377–391. Springer, 2013.

[37] Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking. In Dragan Bosnacki and Stefan Edelkamp, editors, Proceedings of the 14th International Workshop on Model Checking Software (SPIN'07), Berlin, Germany, volume 4595 of Lecture Notes in Computer Science, pages 149–167. Springer, 2007.

[38] Lijun Shan, Behrooz Sangchoulee, Peter Folkeson, Jonny Vinter, Erwin Schotsch, and Claire Loiseaux. A survey on the application of safety, security, and privacy standards for dependable systems. In Proceedings of the 15th European Dependable Computing Conference (EDCC'2019), Naples, Italy, pages 71–72. IEEE, 2019.