Early Verification of Legal Compliance via Bounded Satisfiability Checking

Nick Feng[1], Lina Marsso[1], Mehrdad Sabetzadeh[2], and Marsha Chechik[1]

[1] University of Toronto, Canada. {fengnick,lmarsso,chechik}@cs.toronto.edu
[2] University of Ottawa, Canada. m.sabetzadeh@uottawa.ca

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 using the Z3 SMT solver 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.

1 Introduction

Software systems, such as medical systems, are increasingly required to comply with laws and regulations aimed at ensuring safety, security, and data privacy [1,35]. The properties stipulated by these laws and regulations – 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 [19]): "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. Specifically, 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 [34,23]. For instance, the requirement (req_0) 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 req_0.

Metric first-order temporal logic (MFOTL) enables the specification of data and time constraints [3] and has an expressive formalism for capturing LPs and the related system requirements that constrain data and time [1]. Existing work on MFOTL verification focuses on detecting violations at run-time through monitoring [1,18], with MFOTL formulas being checked on execution logs. There is an unsatisfied need for determining the *satisfiability* of MFOTL specifications, i.e., looking for LP violations possible in MFOTL specification. This is important for designing system requirements that comply with LPs.

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. 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) [34,23]. However, MTL cannot effectively capture quantified data constraints 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 have predominantly relied on bounding the *size of the universe* [12]. 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 our proposed notion of a bound, we develop an incremental approach (IBS) for bounded satisfiability checking of MFOTL. We first translate the MFOTL property and requirements into first-order logic formulas with quantified relational objects (FOL*). We then incrementally ground the FOL* constraints to eliminate the quantifiers by considering an increasing number of relational objects. Subsequently, we check the satisfiability of the resulting constraints using an SMT solver. 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, IBS, for the translated FOL* formulas with incremental and counterexample-guided over/ under-approximation. Note that while our solution to MFOTL satisfibility checking can be applied to a broader domain of applications, in this paper we focus on the legal domain. We empirically evaluate IBS 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

$P1 = \Box\ \forall d, v(Access(d, v)) \implies (\forall v'(v' \neq v \Rightarrow \neg Update(d, v') \wedge \neg Collect(d, v')))\ \mathcal{S}\ (Update(d, v) \vee Collect(d, v)))$
If a personal health information is not accurate or not up-to-date, it should not be accessed.

$req_0 = \Box\ \forall d, v(Access(d, v) \implies \blacklozenge_{[360,)}\ \exists v'.Collect(d,\ v')$
No data is allowed to be accessed before the data ID has been collected for at least 15 days (360 hours).

$req_1 = \Box\ \forall d, v(Update(d, v) \implies \neg(\blacklozenge_{[1,168]}\ \exists v'.(Collect(d, v') \vee Update(d, v'))))$
Data value can only be updated after having been collected or last updated for more than a week (168 hours).

$req_2 = \Box\ \forall d, v(Access(d, v) \implies \blacklozenge_{[0,168]}\ Collect(d, v) \vee Update(d, v))$
Data can only be accessed if has been collected or updated within a week (168 hours).

$req_3 = \Box\ \forall d, v(Collect(d, v) \implies \neg(\exists v''.(Collect(d, v'') \wedge v \neq v'') \vee \blacklozenge_{[1,)}\ \exists v'.Collect(d, v')))$ No data re-collection.

Fig. 1. Example requirements and legal property $P1$ of DCC, with signature $S_{data} = (\emptyset, \{Collect,\ Update,\ Access\}, \iota_{data})$, where $\iota_{data}(Collect) = \iota_{data}(Update) = \iota_{data}(Access) = 2$.

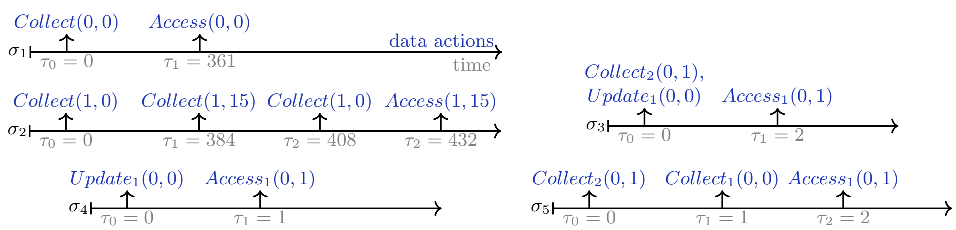

Fig. 2. Five traces from the DCC example.

(BSC) problem. Sec. 4 provides an overview of our solution and the translation of MFOTL to FOL*. Sec. 5 presents our solution, and proofs of its soundness, termination and optimality are in Sec. B. Sec. 6 reports on the experiments 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 describe metric first-order temporal logic (MFOTL) [3].

Syntax. Let $\mathbb{I}$ be a set of non-empty intervals over $\mathbb{N}$. An *interval* $I \in \mathbb{I}$ can be expressed as $[b, b')$ where $b \in \mathbb{N}$ and $b' \in \mathbb{N} \cup \infty$. A *signature* S is a tuple (C, R, ι), where C is a set of constants and R is a finite set of predicate symbols (for relation), respectively. Without loss of generality, we assume all constants are from the integer domain $\mathbb{Z}$ where the theory of linear integer arithmetic (LIA) holds. The function $\iota : R \to \mathbb{N}$ associates each predicate symbol $r \in R$ with an arity $\iota(r) \in \mathbb{N}$. Let *Var* be a countable infinite set of variables from domain $\mathbb{Z}$ and a term t is defined inductively as $t :\ c \mid v \mid t + t \mid c \times t$. We denote $\bar{t}$ as a vector of terms and $\bar{t}_x^k$ as the vector that contains x at index k. The syntax of MFOTL formulas is defined as follows: *(1)* $\top$ and $\bot$, representing values "true" and "false"; *(2)* $t = t'$ and $t > t'$, for terms t and t'; *(3)* $r(t_1 ... t_{\iota(r)})$ for $r \in R$ and terms $t_1 ... t_{\iota(r)}$; *(4)* $\phi \wedge \psi$, $\neg\phi$ for MFOTL formulas ϕ and ψ; *(5)* $\exists x.(r(\bar{t}_x^k) \wedge \phi)$ for MFOTL formula ϕ, relation symbol $r \in R$, variable $x \in Var$ and a vector of terms $\bar{t}_x^k$ s.t. $x = \bar{t}_x^k[k]$; and *(6)* $\phi\ \mathcal{U}_I\ \psi$ (until), $\phi\ \mathcal{S}_I\ \psi$ (since), $\bigcirc_I\ \phi$ (next), $\bullet_I\ \phi$ (previous) for MFOTL formulas ϕ and ψ, and an interval $I \in \mathbb{I}$.

We consider a restricted form of quantification (syntax rule *(5)*, above) similar to guarded quantification [17]. Every existentially quantified variable x must be guarded by some relation r (i.e., for some $\bar{t}$, $r(\bar{t})$ holds and x appears in $\bar{t}$). Similarly, universal quantification must be guarded as $\forall x.(r(\bar{t}) \Rightarrow \phi)$ where $x \in \bar{t}$. Thus, $\neg\exists x.\neg r(x)$ (and $\forall x.r(x)$) are not allowed.

The temporal operators $\mathcal{U}_I$, $\mathcal{S}_I$, $\bullet_I$ and $\bigcirc_I$ require the satisfaction of the formula within the time interval given by I. We write $[b,)$ as a shorthand for $[b,\infty)$; if I is omitted, then the interval is assumed to be $[0,\infty)$. Other classical unary temporal operators $\Diamond_I$ (eventually), $\Box_I$ (always), and $\blacklozenge_I$ (once) are defined as follows: $\Diamond_I\ \phi = \top\ \mathcal{U}_I\ \phi$, $\Box_I\ \phi = \neg\Diamond_I\ \neg\phi$, and $\blacklozenge_I\ \phi = \top\ \mathcal{S}_I\ \phi$. Other common logical operator such as $\vee$ (disjunction) and $\forall$ (universal quantification) are expressed through negation of $\wedge$ and $\exists$, respectively.

Example 1. Suppose a data collection centre (DCC) *collect*s and *access*es personal data information with three requirements: req_0 stating that no data is allowed to be accessed before the data ID has been collected for 15 days (360 hours); req_1: data can only be updated after having been collected or last updated for more than a week (168 hours); and req_2: data value can only be accessed if the value has been collected or updated within a week (168 hours). The signature S_{data} for DCC contains three binary relations (R_{data}): *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 $P1$, req_0, req_1 and req_2 are shown in Fig. 1. For instance, the formula req_0 specifies that if a data value stored at id d is accessed, then some data must have been collected and stored at id d at least 360 hours ago ($\blacklozenge_{[360,)}$]).

Semantics. A first-order (FO) structure D over the signature $S = (C, R, \iota)$ is comprised of a non-empty domain $dom(D) \neq \emptyset$ and an interpretation for $c^D \in dom(D)$ and $r^D \subseteq dom(D)^{\iota(r)}$ for each $c \in C$ and $r \in R$. The semantics of MFOTL formulas is defined over a sequence of FO structures $\bar{D} = (D_0, D_1, \ldots)$ and a sequence of natural numbers representing time $\bar{\tau} = (\tau_0, \tau_1, \ldots)$, where (a) $\bar{\tau}$ is a monotonically increasing sequence; (b) $dom(D_i) = dom(D_{i+1})$ for all $i \geq 0$ (all D_i have a fixed domain); and (c) each constant symbol $c \in C$ has the same interpretation across $\bar{D}$ (i.e., $c^{D_i} = c^{D_{i+1}}$). Property (a) ensures that time never decreases as the sequence progresses; and (b) ensures that the domain is fixed (referred to as $dom(\bar{D})$) $\bar{D}$ is similar to timed words in metric time logic (MTL), but instead of associating a set of propositions with each time point, MFOTL uses a structure D to interpret the symbols in the signature S. The semantics of MFOTL is defined over a trace of timed first-order structures $\sigma = (\bar{D}, \bar{\tau})$, where every structure $D_i \in \bar{D}$ specifies the set of tuples (r^{D_i}) that hold for every relation r at time $\tau_i \in \bar{\tau}$. Let $(\bar{D}, \bar{\tau})$ denote an MFOTL trace.

Example 2. Consider the signature S_{data} in the DCC example. Let $\tau_1 = 0$ and $\tau_2 = 361$, and let D_1 and D_2 be two first-order structures with $r^{D_1} = Collect(0,0)$ and $r^{D_2} = Access(0,0)$, respectively. The trace $\sigma_1 = ((D_1, D_2), (\tau_1, \tau_2))$ 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.

$(\bar{D}, \bar{\tau}, v, i) \models t = t'$	iff $v(t) = v(t')$
$(\bar{D}, \bar{\tau}, v, i) \models t > t'$	iff $v(t) > v(t')$
$(\bar{D}, \bar{\tau}, v, i) \models r(t_1, .., t_{\iota(r)})$	iff $r(v(t_1), .., v(t_{i(r)})) \in r^{D_i}$
$(\bar{D}, \bar{\tau}, v, i) \models \neg\phi$	iff $(\bar{D}, \bar{\tau}, v, i) \not\models \phi$
$(\bar{D}, \bar{\tau}, v, i) \models \phi \wedge \psi$	iff $(\bar{D}, \bar{\tau}, v, i) \models \phi$ and $(\bar{D}, \bar{\tau}, v, i) \models \psi$
$(\bar{D}, \bar{\tau}, v, i) \models \exists x \cdot (r(\bar{t}^k_x) \wedge \phi)$	iff $(\bar{D}, \bar{\tau}, v[x \rightarrow d], i) \models (r(\bar{t}^k_x)) \wedge \phi$ for some $d \in dom(\bar{D})$
$(\bar{D}, \bar{\tau}, v, i) \models \bigcirc_I \phi$	iff $(\bar{D}, \bar{\tau}, v, i+1) \models \phi$ and $\tau_{i+1} - \tau_i \in I$
$(\bar{D}, \bar{\tau}, v, i) \models \bullet_I \phi$	iff $i \geq 1$ and $(\bar{D}, \bar{\tau}, v, i-1) \models \phi$ and $\tau_i - \tau_{i-1} \in I$
$(\bar{D}, \bar{\tau}, v, i) \models \phi\, \mathcal{U}_I\, \psi$	iff exists $j \geq i$ and $(\bar{D}, \bar{\tau}, j, v) \models \psi$ and $\tau_j - \tau_i \in I$ and for all $k \in \mathbb{N}$ $i \leq k < j \Rightarrow (\bar{D}, \bar{\tau}, k, v) \models \phi$
$(\bar{D}, \bar{\tau}, v, i) \models \phi\, \mathcal{S}_I\, \psi$	iff exists $j \leq i$ and $(\bar{D}, \bar{\tau}, j, v) \models \psi$ and $\tau_i - \tau_j \in I$ and for all $k \in \mathbb{N}$ $i \geq k > j \Rightarrow (\bar{D}, \bar{\tau}, k, v) \models \phi$

Fig. 3. MFOTL semantics.

A *valuation function* $v : Var \rightarrow dom(\bar{D})$ maps a set Var of variables to their interpretations in the domain $dom(\bar{D})$. For vectors $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{d} = (d_1, \ldots, d_n) \in dom(\bar{D})^n$, the *update operation* $v[\bar{x} \rightarrow \bar{d}]$ produces a new valuation function v' s.t. $v'(x_i) = d_i$ for $1 \leq i \leq n$, and $v(x') = v'(x')$ for every $x' \notin \bar{x}$. For any constant c, $v(c) = c^D$. Let $\bar{D}$ be a sequence of FO structures over signature $S = (C, R, \iota)$ and $\bar{\tau}$ be a sequence of natural numbers. Let ϕ be an MFOTL formula over S, v be a valuation function and $i \in \mathbb{N}$. A fragment of the relation $(\bar{D}, \bar{\tau}, v, i) \models \phi$ is defined in Fig. 3.

The operators $\bullet_I$, $\bigcirc_I$, $\mathcal{U}_I$ and $\mathcal{S}_I$ are augmented with an interval $I \in \mathbb{I}$ which defines the satisfaction of the formula within a time range specified by I relative to the current time at step i, i.e., τ_i.

Definition 1 (MFOTL Satisfiability). *An MFOTL formula ϕ is* satisfiable *if there exists a sequence of FO structures $\bar{D}$ and natural numbers $\bar{\tau}$, and a valuation function v such that $(\bar{D}, \bar{\tau}, v, 0) \models \phi$. ϕ is* unsatisfiable *otherwise.*

Example 3. In the DCC example, the MFOTL formula req_0 is *satisfiable* because $(\bar{D}, \bar{\tau}, v, 0) \models req_0$ (where $\sigma_1 = (\bar{D}, \bar{\tau})$ in Fig. 2). Let req_0' be another MFOTL formula: $\Diamond_{[0,359]}\, \exists j.(Access(0, j))$. The formula $req_0' \wedge req_0$ is *unsatisfiable* because if data stored at id 0 is accessed between 0 and 359 hours, then it is impossible to collect the data at least 360 hours prior to its access.

3 Bounded Satisfiability Checking Problem

The satisfiability of MFOTL properties is generally undecidable since MFOTL is expressive enough to describe the blank tape problem [30] (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 satisfiability checking, the solution to BSC provides bounded guarantees (i.e., whether a solution exists within a given bound). In this section, we first define satisfiability checking and then the BSC problem for MFOTL formulas. *Satisfiability* *checking* [31] 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 2 (Satisfiability Checking of MFOTL Formulas). *Let P be an MFOTL formula over a signature $S = (C, R, \iota)$, and let Reqs be a set of MFOTL requirements over S. Reqs complies with P (denoted as $Reqs \Rightarrow P$) iff $\bigwedge_{\psi \in Reqs} \psi \wedge \neg P$ is* unsatisfiable*. We call a solution to $\bigwedge_{\psi \in Reqs} \psi \wedge \neg P$, if one exists, a* counterexample *to $Reqs \Rightarrow P$.*

Example 4. Consider our DCC system requirements and the privacy data property $P1$ stating that if personal health information is not accurate or not up-to-date, it should not be accessed (see Fig. 1). $P1$ is not respected by the set of DCC requirements $\{req_0, req_1, req_2\}$ because $\neg P1 \wedge req_0 \wedge req_1 \wedge req_2$ is *satisfiable*. The counterexample σ_2 (shown in Fig. 2) indicates that data can be re-collected, and the re-collection does not have the same time restriction as the updates. If a fourth policy requirement req_3 (Fig. 1) is added to prohibit re-collection of collected data, then property $P1$ would be respected (i.e., $\{req_0, req_1, req_2, req_3\} \Rightarrow P1$).

Definition 3 (Finite trace and bounded trace). *Given a trace $\sigma = (\bar{D}, \bar{\tau}, v)$, we use $vol(\sigma)$ (the* volume *of σ), to denote the total number of times that any relation holds across all FO structures in $\bar{D}$ (i.e., $\sum_{r \in R} \sum_{D_i \in \bar{D}} (|r^{D_i}|)$). The trace σ is* finite *if $vol(\sigma)$ is finite. The trace is* bounded *by volume $vb \in \mathbb{N}$ if and only if $vol(\sigma) \leq vb$.*

Example 5. The volume of trace σ_3 in Fig. 2, $vol(\sigma_3) = 3$ since there are three relations: *Collect*(1, 15), *Update*(1, 0), and *Access*(1, 15). Note that the volume is the total number of tuples that hold for any relation across all time points; multiple tuples can thus hold for multiple relations for a single time point.

Definition 4 (Bounded satisfiability checking of MFOTL properties). *Let P be an MFOTL property, Reqs be a set of MFOTL requirements, and vb be a natural number. The* bounded satisfiability checking problem *determines the existence of a counterexample σ to $Reqs \Rightarrow P$ such that $vol(\sigma) \leq vb$.*

4 Checking Bounded Satisfiability

In this section, we present an overview of the bounded satisfiability checking (BSC) process that translates the MFOTL formula into *first-order logic with relational objects* (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 process complexity.

4.1 Overview of BSC for MFOTL Formulas

We aim to address the bounded satisfiability checking problem (Def. 4), looking for a satisfying run σ within a given volume bound vb that limits the number of

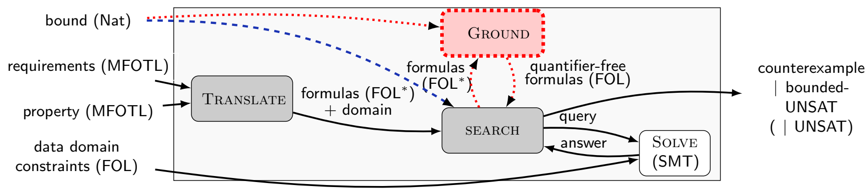

Fig. 4. Overview of the naive and our incremental (IBS) MFOTL bounded satisfiability checking approaches. Solid boxes and arrows are shared between the two approaches. Blue dashed arrow is specific to the naive approach. Red dotted arrows and the additional red output in bracket are specific to IBS.

relations in σ. First, we TRANSLATE the MFOTL formulas to FOL* formulas. The considered constraints in the formulas include those of the system requirements and the legal property, and *optional* data constraints specifying the data value constraint for a datatype. The data constraints can be defined as a range, a "small" data set, or the union/intersection of other data constraints. If data constraints are not specified, then the data value comes from the domain $\mathbb{Z}$. Note that the optional data constraints do not affect the complexity of BSC, but they do help prune unrealistic counterexamples. Second, we SEARCH for a satisfying solution to the FOL* formula; 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 σ, or "bounded-UNSAT").

4.2 Translation of MFOTL to First-Order Logic

In this section, we describe the translation target FOL*, the translation rules and prove their correctness.

FOL with Relational Object (FOL*) We start by introducing the syntax of FOL*. A *signature* S is a tuple (C, R, ι), where C is a set of constants, R is a set of relation symbols, and $\iota : R \rightarrow \mathbb{N}$ is a function that maps a relation to its arity. We assume that the domain of constant C is $\mathbb{Z}$, which matches the one for MFOTL, where the theory of linear integer arithmetic (LIA) holds. Let *Var* be a set of variables in the domain $\mathbb{Z}$. A *relational object* o of class $r \in R$ (denoted as $o : r$) is an object with $\iota(r)$ regular attributes and two special attributes, where every attribute is a variable. We assume that all regular attributes are ordered and denote $o[i]$ to be the ith attribute of o. Some attributes are named, and $o.x$ refers to o's attribute with the name 'x'. Each relational object o has two special attributes $o.ext$ and $o.time$. The former is a boolean variable indicating whether o exists in a solution, and the latter is a variable representing the occurrence time of o. For convenience, we define a function $\text{CLS}(o)$ to return the relational object's class. Let *a* FOL* *term* t be defined inductively as $t :\ c \mid v \mid o[k] \mid o.x \mid t{+}t \mid c{\times}t$ for any constant $c \in C$, any variable $v \in Var$, any relational object $o : r$, any index $k \in [1, \iota(r)]$ and any valid attribute name x. Given a signature S, the syntax of the FOL* formulas is defined as follows: *(1)* $\top$ and $\bot$, representing values "true" and "false"; *(2)* $t = t'$ and $t > t'$, for term t and t'; *(3)* $\phi_f \wedge \psi_f$, $\neg\phi_f$ for FOL* formulas ϕ_f and ψ_f; *(4)* $\exists o : r \cdot (\phi_f)$ for an FOL* formula ϕ_f and a class r; *(5)* $\forall o : r \cdot (\phi_f)$ for an FOL* formula ϕ_f and a class r. The quantifiers for FOL* formulas are limited to relational objects, as shown by rules (4) & (5). Operators $\vee$ and $\forall$ can be defined in FOL* as follows: $\phi_f \vee \psi_f = \neg(\neg\phi_f \wedge \neg\psi_f)$ and $\forall o : r \cdot \phi_f = \exists o : r \cdot \neg\phi_f$. We say an FOL* formula is in a *negation normal form* (NNF) if negations ($\neg$) do not appear in front of $\neg$, $\wedge$, $\vee$, $\exists$ and $\forall$. For the rest of the paper, we assume that every FOL* ϕ is in NNF.

Given a signature S, *a domain* D is a finite set of relational objects. An FOL* formula *grounded* in the domain D (denoted by ϕ_D) is a quantifier-free FOL formula that eliminates quantifiers on relational objects using the following rules: (1) $\exists o : r \cdot (\phi_f)$ to $\bigvee_{o':r\in D}(o'.ext \wedge \phi_f[o \leftarrow o'])$ and (2) $\forall o : r \cdot (\phi_f)$ to $\bigwedge_{o':r\in D}(o'.ext \Rightarrow \phi_f[o \leftarrow o'])$. An FOL* formula ϕ_f is *satisfiable in* D if there exists a variable assignment v that evaluates ϕ_D to $\top$ according to the standard semantics of FOL. An FOL* formula ϕ_f is *satisfiable* if there exists a finite domain D such that ϕ_f is satisfiable in D. We call $\sigma = (D, v)$ *a satisfying solution to* ϕ_f, denoted as $\sigma \models \phi_f$. Given a solution $\sigma = (D, v)$, we say a relational object o is in σ, denoted as $o \in \sigma$, if $o \in D$ and $v(o.ext)$ is true. The *volume of the solution*, denoted as $vol(\sigma)$, is $|\{o \mid o \in \sigma\}|$.

Example 6. Let a be a relational object of class A with attribute name val. The formula $\forall a : A.\ (\exists a' : A \cdot\ (a.val < a'.val) \wedge \exists a : A \cdot\ a.val = 0)$ has no satisfying solutions in any finite domain. On the other hand, the formula $\forall a : A \cdot (\exists a', a'' : A \cdot (a.val = a'.val + a''.val) \wedge \exists a : A \cdot a.val = 5)$ has a solution $\sigma = (D, v)$ of volume 2, with the domain $D = (a_1, a_2)$ and the value function $v(a_1.val) = 5$, $v(a_2.val) = 0$ because if $a \leftarrow a_1$ then the formula is satisfied by assigning $a' \leftarrow a_1,\ a'' \leftarrow a_2$; and if $a \leftarrow a_2$, then the formula is satisfied by assigning $a' \leftarrow a_2,\ a'' \leftarrow a_2$.

From MFOTL Formulas to FOL* Formulas. We now discuss the translation rule from the MFOTL formulas to FOL* formulas. Recall that MFOTL semantics is defined for a time point i on a trace $\sigma = (\bar{D}, \bar{\tau}, v, i)$, where $\bar{D} = (D_1, D_2, \ldots)$ is a sequence of FO structures and $\bar{\tau} = (\tau_1, \tau_2, \ldots)$ is a sequence of time values. The time value of the time point i is given by τ_i, and if i is not specified, then $i = 1$. The semantics of the FOL* formulas is defined for a domain D where the information of time is associated with relational objects in the domain. Therefore, the time point i (and its time value τ_i) should be considered during the translation from MFOTL to FOL* since the same MFOTL formula at different time points represents different constraints on the trace σ. Formally, our translation function TRANSLATE, abbreviated as T, translates an MFOTL formula ϕ into a function $f : \tau \rightarrow \phi_f$, where $\tau \in \mathbb{N}$ and ϕ_f is an FOL* formula. The translation rules are stated in Fig. 5.

Representing time points in FOL*. Since FOL* quantifiers are limited to relational objects, to quantify over time points (which is necessary to capture the semantics of MFOTL temporal operators such as $\mathcal{U}$), the translated FOL* formulas use a special *internal* class of relational objects TP (e.g., $\exists o : \mathrm{TP}$). Relational

$$
\begin{array}{ll}
T(t = t', \tau_i) & \rightarrow t = t' \\
T(t > t', \tau_i) & \rightarrow t > t' \\
T(r(t_1, .., t_{\iota(r)}), \tau_i) & \rightarrow \exists o : r \cdot \bigwedge_{j=1}^{\iota(r)} (o.j = t_j) \wedge (\tau_i = o.time) \\
T(\neg\phi, \tau_i) & \rightarrow \neg T(\phi, \tau_i) \\
T(\phi \wedge \psi, \tau_i) & \rightarrow T(\phi, \tau_i) \wedge T(\psi, \tau_i) \\
T(\exists x \cdot r(\bar{t}_x^k) \wedge \phi, \tau_i) & \rightarrow \exists o : r \cdot T((r(\bar{t}_x^k) \wedge \phi)[x \rightarrow o[k]], \tau_i) \\
T(\bigcirc_I \phi, \tau_i) & \rightarrow \exists o : \mathrm{TP} \cdot \textsc{Next}(o.time, \tau_i) \wedge T(\phi, o.time) \wedge (o.time - \tau_i) \in I \\
T(\bullet_I \phi, \tau_i) & \rightarrow \exists o : \mathrm{TP} \cdot \textsc{Prev}(o.time, \tau_i) \wedge T(\phi, o.time) \wedge (\tau_i - o.time) \in I \\
T(\phi\, \mathcal{U}_I\, \psi, \tau_i) & \rightarrow \exists o : \mathrm{TP} \cdot (o.time \geq \tau_i \wedge (o.time - \tau_i) \in I \wedge T(\psi, o.time) \\
 & \quad \text{and } \forall o' : \mathrm{TP} \cdot o'.time \cdot (\tau_i \leq o'.time < o.time \Rightarrow T(\phi, o'.time))) \\
T(\phi\, \mathcal{S}_I\, \psi, \tau_i) & \rightarrow \exists o : \mathrm{TP} \cdot (o.time \leq \tau_i \wedge (\tau_i - o.time) \in I \wedge T(\psi, o.time) \\
 & \quad \text{and } \forall o' : \mathrm{TP} \cdot (\tau_i \geq o'.time > o.time \Rightarrow T(\phi, o'.time))) \\
T(\phi) & \rightarrow T(\phi, \tau_1)
\end{array}
$$

Fig. 5. Translation rules from MFOTL to FOL*. TP is an internal class of relational objects used to represent time values at different time points. The predicate $\textsc{Next}(t_1, t_2)$ ($\textsc{Prev}(t_1, t_2)$) asserts that t_1 is the next (previous) time value of t_2.

objects of class TP capture all possible time points in a trace, and they have two attributes, *ext* and *time*, to record the existence and the value of the time point, respectively. To ensure that every time value in a solution is represented by some relational object of TP, we introduce the *time coverage* FOL* axiom.

Axiom 1 (Time coverage)**.** Let ϕ_f be an FOL* formula and let σ be its solution. For every relational object $o \in \sigma$, there exists an object o' of class TP s.t. o and o' share the same time value. Formally, $\forall o \cdot (\exists o' : \mathrm{TP} \cdot o.time = o'.time)$.

The translation of $\bigcirc_I \phi$ uses function $\textsc{Next}(t_1, t_2)$ to assert that t_1 is the next time value of t_2. Formally, $\textsc{Next}(t_1, t_2) = \forall o : \mathrm{TP} \cdot o.time > t_2 \Rightarrow t_1 \leq o.time$. Function $\textsc{Prev}(t_1, t_2)$ for translation of $\bullet_I \phi$ is defined similarly.

Definition 5 (Mapping from MFOTL trace to FOL* trace). *Let an MFOTL trace $(\bar{D}, \bar{\tau})$ and a valuation function v be given. A function $M((\bar{D}, \bar{\tau}), v) \rightarrow (D, v')$ is* a mapping between an MFOTL trace and an FOL* trace *if M satisfies the following rules: (1) for every $\tau_i \in \bar{\tau}$, there exists a relational object $o : \mathrm{TP} \in D$ such that $\tau_i = v'(o.time)$; (2) for every structure $D_i \in \bar{D}$, if a tuple $\bar{t}$ holds for a relation r, (i.e., $\bar{t} \in r^{D_i}$), then there exists a relational object $o : r$ such that for $j \in \iota(r)$, $\bar{t}[j] = v'(o[j])$ and $v'(o.time) = \tau_i \wedge v'(o.ext) = \top$; (3) for every term t defined for v, $v(t) = v'(T(t, \tau_i))$.*

The inverse of M, denoted as M^{-1}, is defined as follows: (1) $\bar{\tau} = \textsc{sort}(\{v'(o.time) \mid o : TP \in D \cdot v'(o.ext)\})$ and (2) for every relational object $o : r$, if $v'(o.ext)$, then $(v'(o[1]) \ldots v'(o[\iota(r)])) \in r^{D_i}$, where i is the index of the time value $v'(o.time)$ in $\bar{\tau}$.

Lemma 1. *Given an MFOTL formula ϕ, an MFOTL trace $(\bar{D}, \bar{\tau})$, a valuation function v, and a time point i, the relation $(\bar{D}, \bar{\tau}, v, i) \models \phi$ holds iff there exists a satisfying trace $\sigma = (D, v')$ for the formula $T(\phi, \tau_i)$.*

Proof Sketch. In the proof, we use M and M^-1 (see Def. 5) to transform an MFOTL solution into an FOL* trace, and show that it is a solution to the translated FOL* formula (and vice versa).

$\implies$: if $(\bar{D}, \bar{\tau}, v, i) \models \phi$, then it is sufficient to show $(D, v') \leftarrow M(\bar{D}, \bar{\tau}, v)$ is an FOL* solution. To prove (D, v') is the solution to $T(\phi, \tau_i)$, we consider all the translation rules in Fig. 5. The translated FOL* matches the semantics (Fig. 3) of MFOTL except for the translation of temporal operators (e.g., $\mathrm{T}(\bigcirc_I \phi, \tau_i)$ and $T(\phi\ \mathcal{U}_I\ \psi, \tau_i)$) where instead of quantifying over time points (e.g., $\exists j$ and $\forall k$), internal relational objects of class TP $(o, o' : \mathrm{TP})$ are quantified over. By rule (1) of Dec. 5, every time point and its time value are mapped to some relational object of class TP. Therefore, the quantifiers on time points can be translated into the quantifiers on the relational objects of TP. The mapped solution (D, v') also satisfies Axiom 1 because if a tuple $\bar{t}$ holds for some relation r at some time τ in the MFOTL trace $(\bar{D}, \bar{\tau})$, then there exists a time point $i \in [1, |\bar{\tau}|]$ such that $\tau_i = \tau$. Therefore, by rule (1) of M, τ_i is represented by some $o : \mathrm{TP}$.

$\Longleftarrow$: if $(D, v') \models T(\phi, \tau_i)$, then it is sufficient to show that the MFOTL trace $(\bar{D}, \bar{\tau}, v) \leftarrow M^{-1}(D, v')$ satisfies ϕ at point i (i.e., $(\bar{D}, \bar{\tau}, v, i) \models \phi$). To prove $(\bar{D}, \bar{\tau}, v, i) \models \phi$, we consider all the translation rules in Fig. 5. The translated FOL* formula matches the semantics of MFOTL (Fig. 3) except for the difference between the time points and the relational objects of class TP. By Axiom 1, every relational object's time is captured by some time point, and by rule (2) of M^-1, every relational object is mapped onto some structure D_i at some time τ_i by M. Therefore, $(\bar{D}, \bar{\tau}, v, i) \models \phi$. □

Theorem 1 (Translation Correctness). *Given an MFOTL formula ϕ and an MFOTL trace σ, let $M(\sigma)$ be the FOL* solution mapped from σ using function M (Def. 5). Then (1) $\sigma \models \phi$ if and only if $M(\sigma) \models T(\phi)$, and (2) $vol(\sigma) = vol(M(\sigma)) - |\{o : \mathrm{TP} \in M(\sigma)\}|$, where $|\{o : \mathrm{TP} \in M(\sigma)\}|$ is the number of relational objects of the internal class* TP *in the solution $M(\sigma)$.*

Proof. Statement (1) of Thm. 1 is a direct consequence of Lemma 1. Statement (2) is the result of rule (2) in Def. 5 because every relational object in the FOL* solution, except for the internal ones, i.e., o : TP, has a one-to-one correspondence to tuples that hold for some relation in the MFOTL solution. □

For the rest of the paper, we assume that the internal relational objects of class TP do not count toward the volume of the FOL*, i.e., $vol(\sigma) = vol(T(\sigma))$.

Example 7. Consider a formula $exp = \Box\ \forall d \cdot (A(d) \implies \Diamond_{[5,10]}\ B(d))$, where A and B are unary relations. The translated FOL* formula $T(exp)$ is: $\forall o : \mathrm{TP} \cdot \forall a : A \cdot (o.time = a.time \Rightarrow \exists o' : \mathrm{TP} \cdot b : B \cdot o'.time = b.time \wedge a[1] = b[1] \wedge\ o.time + 5 \leq o'.time \leq o.time + 10)$. Since $o.time = a.time$ and $o'.time = b.time$, we can substitute $o.time$ and $o'.time$ with $a.time$ and $b.time$ in $T(exp)$, respectively. Then, the formula contains no reference to o and o', and we can safely drop the quantified o and o' (we can drop existential quantified TP relational object because of the time coverage axiom). The simplified formula is: $\forall a : A \cdot \exists b : B \cdot a[1] = b[1] \wedge\ a.time + 5 \leq b.time \leq a.time + 10$.

This is important for designing system requirements that comply with LPs.

Given an MFOTL property P and a set *Reqs* of MFOTL requirements, and a volume bound vb, the BSC problem can be solved by searching for a satisfying solution v' for the FOL* formula $T(\neg P) \bigwedge_{\psi \in Reqs} T(\psi)$ in a domain D with at most vb relational objects.

4.3 Checking MFOTL Satisfiability: A Naive Approach

Below, we define a naive procedure NBS (shown in Fig. 4) for checking satisfiability of MFOTL formulas translated into FOL*. We then discuss the complexity of this naive procedure. Even though we do not use NBS in this paper, its complexity constitutes an upper bound for our approach proposed in Sec. 5.

Searching for a satisfying solution. Let ϕ_f be an FOL* formula translated from an MFOTL formula ϕ, and let vb be the volume bound. NBS solves ϕ_f via quantifier elimination. The number of relational objects in any satisfying solution of ϕ_f should be at most vb. Therefore, NBS grounds the FOL* formulas within a domain of vb relational objects (see Sec. 4.2), and then uses an SMT solver to check satisfiability of the grounded formula. If the domain has multiple classes of relational objects, we can unify them by introducing a "superposition" class whose attributes are the union of the attributes of all classes and a special "name" attribute to indicate the class represented by the superposition.

Complexity. The size of the quantifier-free formula is $O(vb^k)$, where k is the maximum depth of quantifier nesting. Since the background theory used in ϕ is restricted to linear integer arithmetic, solving the formula is NP-hard [28]. Because T (Tab. 5) is linear in the size of the formula ϕ, NBS is NP-complete w.r.t. the size of the grounded formula, vb^k.

5 Incremental Search for Bounded Counterexamples

The naive BSC approach (NBS) proposed in Sec. 4.3 is inefficient for solving the translated FOL* formulas given a large bound n due to the size of the ground formula. Moreover, NBS 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 [14], respectively. In this section, we propose an incremental procedure IBS, which can detect unbounded unsatisfiability and provide the shortest counterexamples. An overview of IBS is given in Fig. 4.

IBS maintains an under-approximation of the search domain and the FOL* constraints. It uses the search domain to ground the FOL* constraints, and an SMT solver to determine the satisfiability of the grounded 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 σ, "bounded-UNSAT", or "UNSAT"). The procedure continues until an answer is obtained (σ or UNSAT), or until the domain exceeds the bound vb, in which case a "bounded-UNSAT" answer is returned.

In the following, we describe IBS in more detail. We explain the key component of IBS, computing over- and under-approximation queries, in Sec. 5.1. We discuss the algorithm itself in Sec. 5.2 and illustrate it in Sec. 5.3. We prove its soundness (Thm. 2), completeness (Thm. 3), and solution optimality (Thm. 4) in Sec. B.

5.1 Over- and Under-Approximation

NBS grounds the input FOL* formulas in a fixed domain D (fixed by the bound vb). Instead, IBS under-approximates D to $D_\downarrow$ such that $D_\downarrow \subseteq D$. With $D_\downarrow$, we can create an over- and an under-approximation query to the bounded satisfiability checking problem. Such queries are used to check the satisfiability of FOL* formulas with domain $D_\downarrow$. IBS starts with a small domain $D_\downarrow$ and gradually expands it until either SAT or UNSAT is returned, or the domain size exceeds some limit (bounded-UNSAT).

Over-approximation. Let ϕ_f be an FOL* formula, and $D_\downarrow$ be a domain of relation objects. The procedure GROUND, $G(\phi_f, D_\downarrow)$, encodes ϕ_f into a quantifier-free FOL formula ϕ_g s.t. the unsatisfiability of ϕ_g implies the unsatisfiability of ϕ_f. We call ϕ_g an *over-approximation* of ϕ_f. The procedure G (Alg. 2) recursively traverses the syntax tree of the input FOL* formula from top to bottom.

To eliminate the existential quantifier in $\exists o : r \cdot \phi'_f$ (L:1), G creates a new relational object o' of class r (L: 2), and replaces o with o' in ϕ'_f (L:3). To eliminate the universal quantifier in $\forall o : r \cdot \phi'_f$ (L: 4), G grounds the formula in $D_\downarrow$. More specifically, G expands the quantifier into a conjunction of clauses where each clause is $o'.ext \Rightarrow \phi'_f[o \leftarrow o']$ (i.e., o is replaced by o' in ϕ'_f) for each relational object o' of class r in $D_\downarrow$ (L: 5). Intuitively, an existentially quantified relational object is instantiated with a new relational object, and a universally quantified relational object is instantiated with every existing relational object of the same class in $D_\downarrow$, which does not include the ones instantiated during G.

Lemma 2 (Over-approximation Query). *For an FOL* formula ϕ_f, and a domain $D_\downarrow$, if $\phi_g = G(\phi_f, D_\downarrow)$ is UNSAT, then so is ϕ_f.*

Under-approximation. Let ϕ_f be an FOL* formula, and $D_\downarrow$ be a domain. The over-approximation $\phi_g = G(\phi_f, D_\downarrow)$ contains a set of new relational objects introduced by G (L:2), denoted by $NewRs$. Let NONEWR($NewRs$, $D_\downarrow$) be constraints that enforce that every new relational object o_1 in $NewRs$ be semantically equivalent to some relational objects o_2 in $D_\downarrow$. Formally: the predicate NONEWR($NewRs, D_\downarrow$) is defined as $\bigwedge_{o_1 \in NewRs} \bigvee_{o_2 \in D_\downarrow} (o_1 \equiv o_2)$, where the semantically equivalent relation between o_1 and o_1 (i.e., $o_1 \equiv o_2$) is defined as $\textsc{cls}(o_1) = \textsc{cls}(o_2)$ and $\bigwedge_{i=1}^{\iota(\textsc{cls}(o))} (o_1[i] = o_2[i]) \wedge o_1.ext = o_2.ext \wedge o_1.time = o_2.time$ (where the $\textsc{cls}(o)$ returns the class of o). Let $\phi_g^\perp = \phi_g \wedge$ NONEWR($NewRs, D_\downarrow$). If $\phi_g^\perp$ has a satisfying solution, then there is a solution for ϕ_f. We call $\phi_g^\perp$ an *under-approximation* of ϕ_f and denote the procedure for computing it by UNDERAPPROX($\phi_f, D_\downarrow$).

Lemma 3 (Under-Approximation Query). *For an FOL* formula ϕ_f, and a domain $D_\downarrow$, let $\phi_g = G(\phi_f, D_\downarrow)$ and $\phi_g^\perp =$ UNDERAPPROX($\phi_f, D_\downarrow$). If σ is a solution to $\phi_g^\perp$, then there exists a solution to ϕ_f.*

The proofs of Lemma 2 and 3 are in Sec. A
Suppose, for some domain $D_\downarrow$, that an over-approximation query ϕ_g for an FOL* formula ϕ_f is satisfiable while the under-approximation query $\phi_g^\perp$ is UNSAT.

Algorithm 1 IBS: search for a bounded (by vb) solution to $T(\neg P) \bigwedge_{\psi \in Reqs} T(\psi)$.

Input an MFOTL formula $\neg P$, and MFOTL requirements $Reqs = \{\psi_1, \psi_2, ...\}$.
Optional Input vb, the volume bound, and data constraints T_{data}.
Output a counterexample σ, UNSAT or bounded-UNSAT.

$Reqs_f \leftarrow \{ \psi_f = T(\psi) \mid \psi \in Reqs\}$
$\neg P_f \leftarrow T(\neg P)$
$Reqs_{\downarrow} \leftarrow \emptyset$ *//initially empty requirement*
$D_{\downarrow} \leftarrow \emptyset$ *//initially empty domain*
while $\top$ **do**
  $\phi_{\downarrow} \leftarrow \neg P_f \wedge Reqs_{\downarrow}$
  $\phi_g \leftarrow G(\phi_{\downarrow}, D_{\downarrow})$ *//over-approx.*
  $\phi_g^{\perp} \leftarrow$ UNDERAPPROX$(\phi_{\downarrow}, D_{\downarrow})$ *//under-approx.*
  if SOLVE$(\phi_g \wedge T_{data})$ = UNSAT **then**
    return UNSAT
  $\sigma \leftarrow$ SOLVE$(\phi_g^{\perp} \wedge T_{data})$
  if σ = UNSAT **then** *//expand $D_{\downarrow}$*
    $\sigma_{min} \leftarrow$ MINIMIZE(ϕ_g)
    //expand based on σ_{min}
    $D_{\downarrow}$ += $\{o \mid o \in \sigma_{min}\}$
    if $vol(\sigma_{min}) > vb$ **then**
      return bounded-UNSAT
  else *//check all requirements*
    if $\sigma \models \psi_f$ for $\psi_f \in Reqs_f$ **then**
      return σ
    else
      $lesson \leftarrow \psi_f$ for some $\sigma \not\models \psi_f$
      $Reqs_{\downarrow}$.add$(lesson)$

Algorithm 2 G: ground a NNF FOL* formula ϕ_f in a domain $D_{\downarrow}$.

Input an FOL* formula ϕ_f in NNF, and a domain of relational objects $D_{\downarrow}$.
Output a grounded quantifier-free formula ϕ_g over relational objects.

if match $(\phi_f, \exists o : r \cdot \phi'_f)$ **then** *//process the existential operator*
  $o' \leftarrow$ NEWACT(r) *//create a new relational object of class r*
  return $o'.ext \wedge G\ (\phi'_f[o \leftarrow o'], D_{\downarrow})$
if match $(\phi_f, \forall o : r \cdot \phi'_f)$ **then** *//process the universal operator*
  return $\bigwedge_{[o':r] \in D_{\downarrow}} o'.ext \Rightarrow G\ (\phi'_f[o \leftarrow o'], D_{\downarrow})$
if match $(\phi_f, \phi'_f\ op\ \psi'_f$ where $op = \wedge \mid \vee)$ **then return** $G(\phi'_f, D_{\downarrow})\ op\ G(\psi'_f, D_{\downarrow})$
return ϕ_f *//case where ϕ_f is quantifier-free, including $\neg\phi'_f$ where ϕ'_f is atomic (NNF)*

Then, the solution to ϕ_g provides hints on how to expand $D_{\downarrow}$ to potentially obtain a satisfying solution for ϕ_f, as captured in Cor. 1.

Corollary 1 (Necessary relational objects). *For an FOL* formula ϕ_f and a domain $D_{\downarrow}$, let ϕ_g and $\phi_g^{\perp}$ be the over- and under-approximation queries of ϕ_f based on $D_{\downarrow}$, respectively. Suppose ϕ_g is satisfiable and $\phi_g^{\perp}$ is UNSAT, then every solution to ϕ_f contains some relational object in formula ϕ_g but not in $D_{\downarrow}$.*

5.2 Counterexample-Guided Constraint Solving Algorithm

Let an MFOTL formula $\neg P$ (to find a satisfiable counterexample to P), a set of MFOTL requirements $Reqs$, an optional volume bound vb, and optionally a set of FOL* data domain constraints T_{data} be given. IBS, shown in Alg. 1, searches for a solution σ to $\neg P \wedge \bigwedge_{\psi \in Reqs} \psi$ (with respect to T_{data}) bounded by vb, as a counter-example to $\bigwedge_{\psi \in Reqs} \psi \Rightarrow P$ (Def. 2). bounded by vb. If no such solution is possible regardless of the bound, IBS returns UNSAT. If no solution can be found within the given bound, but a solution may exist for a larger bound, then IBS returns bounded-UNSAT. If vb is not specified, IBS will perform the search unboundedly until a solution or UNSAT is returned.

IBS first translates $\neg P$ and every $\psi \in \mathit{Reqs}$ into FOL* formulas in Reqs_f, denoted by $\neg P_f$ and ψ_f, respectively. Then IBS searches for a satisfying solution to $\neg P_f \wedge \bigwedge_{\psi_f \in \mathit{Reqs}_f} \psi_f$ in the domain D of volume, which is at most vb. Instead of searching in D directly, IBS searches for a solution to $\neg P_f \wedge \bigwedge_{\psi_f \in \mathit{Reqs}_\downarrow} \psi_f$ in $D_\downarrow$ (denoted by $\phi_\downarrow$) where $\mathit{Reqs}_\downarrow \subseteq \mathit{Reqs}_f$ and $D_\downarrow \subseteq D$. IBS initializes $\mathit{Reqs}_\downarrow$ and $D_\downarrow$ as empty sets (LL:3-4). Then, for the FOL* formula $\phi_\downarrow$, IBS creates an over- and under-approximation query ϕ_g (L:7) and $\phi_g^\perp$ (L:8), respectively (described in Sec. 5.1). IBS first solves the over-approximation query ϕ_g by querying an SMT solver (L:9). If ϕ_g is unsatisfiable, then $\phi_\downarrow$ is unsatisfiable (Lemma 2), and IBS returns UNSAT (L:10).

If ϕ_g is satisfiable, then IBS solves the under-approximation query $\phi_g^\perp$ (L:11). If $\phi_g^\perp$ is unsatisfiable, then the current domain $D_\downarrow$ is too small, and IBS expands it (LL:12-18). This is because the satisfiability of ϕ_g indicates the possibility of finding a satisfying solution after adding at least one of the new relational objects in the solution to ϕ_g to $D_\downarrow$ (Cor. 1). The domain $D_\downarrow$ is expanded by adding all relational objects o' in the minimum (in terms of volume) solution σ_{min} to ϕ_g (L:13). To obtain σ_{min}, we follow MaxRes [27] methods: we analyze the UNSAT core of $\phi_g^\perp$ and incrementally weaken $\phi_g^\perp$ towards ϕ_g (i.e., the weakened query $\phi_g^{\perp'}$ is an "over-under approximation" that satisfies $\phi_g^\perp \Rightarrow \phi_g^{\perp'} \Rightarrow \phi_g$) until a satisfying solution σ_{min} is obtained for the weakened query. However, if the volume of σ_{min} exceeds vb (L:16), then bounded-UNSAT is returned (L:17). UNSAT core-guided domain expansion has also been explored for unfolding the definition of recursive functions [29,36].

On the other hand, if $\phi_g^\perp$ yields a solution σ, then σ is checked on Reqs_f (L:19). If σ satisfies every ψ_f in Reqs_f, then σ is returned (L:20). If σ violates some requirements in Reqs_f, then the violating requirement *lesson* is added to $\mathit{Reqs}_\downarrow$ to be considered in the search for the next solutions (L:23).

If IBS does not find a solution or does not return UNSAT, it means that no solution is found because $D_\downarrow$ is too small or $\mathit{Reqs}_\downarrow$ are too weak. IBS then restarts with the expanded domain $D_\downarrow$ or the refined set of requirements $\mathit{Reqs}_\downarrow$. It computes the over- and under-approximation queries (ϕ_g and $\phi_g^\perp$) again, and repeats the steps. See Sec. 5.3 for an illustration of IBS.

Remark 1. IBS finds the optimal solution because it looks for the minimum solution σ_{min} to the over-approximation query ϕ_g (L:13) and uses it for domain expansion (L:15). However, looking for σ_{min} adds cost. If solution optimality is not required, IBS can be configured to heuristically find a solution σ to ϕ_g such that $vol(\sigma) \leq vb$. The *greedy best-first* search (gBFS) finds a solution to ϕ_g that minimizes the number of relational objects that are not already in $D_\downarrow$, and then uses it to expand $D_\downarrow$. We configured a non-optimal version of IBS (nop) that uses gBFS heuristics and evaluated its performance in Sec. 6.

5.3 Illustration of IBS

Suppose a data collection centre (DCC) *collect*s and *access*es personal data information with two requirements: req_1: data value can only be updated after having been collected or last updated for more than a week (168 hours); and req_2: data can only be accessed if has been collected or updated within a week (168 hours). The signature S_{data} for DCC contains three binary relations (R_{data}): *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 $P1$, req_1 and req_2 are shown in Fig. 1. Suppose IBS is invoked to find a counterexample for property $P1$ (shown in Fig. 1) subject to requirements $Reqs = \{req_1, req_2\}$ with the bound $vb = 4$. IBS translates the requirements and the property to FOL* and initializes $Reqs_\downarrow$ and $D_\downarrow$ to empty sets. For each iteration, we use ϕ_g and $\phi_g^\perp$ to represent the over- and under-approximation queries computed on LL:7-8, respectively.

1st iteration: $D_\downarrow = \emptyset$ and $Reqs_\downarrow = \emptyset$. Three new relational objects are introduced to ϕ_g (due to $\neg P1$): $access_1$, $collect_1$, and $update_1$ such that: (C1) $access_1$ occurs after $collect_1$ and $update_1$;(C2) $access_1.d = collect_1.d = update_1.d$;(C3) $access_1.v \neq collect_1.v \wedge access_1.v \neq update_1.v$; and (C4) either $collect_1$ or $update_1$ must be in the solution. ϕ_g is satisfiable, but $\phi_g^\perp$ is UNSAT since $D_\downarrow$ is an empty set. We assume $D_\downarrow$ is expanded by adding $access_1$ and $update_1$.

2nd iteration: $D_\downarrow = \{access_1, update_1\}$ and $Reqs_\downarrow = \emptyset$. The over-approximation ϕ_g stays the same, but $\phi_g^\perp$ becomes satisfiable since $access_1$ and $update_1$ are in $D_\downarrow$. Suppose the solution is σ_4 (see Fig. 2). However, σ_4 violates req_2, so req_2 is added to $Reqs_\downarrow$.

3rd iteration: $D_\downarrow = \{access_1, update_1\}$ and $Reqs_\downarrow = \{req_2\}$. Two new relational objects are introduced in ϕ_g (due to req_2): $collect_2$ and $update_2$ such that (C5) $collect_2.time \leq access_1.time \leq collect_2.time + 168$; (C6) $update_2.time \leq access_1.time \leq update_2.time + 168$; (C7) $access_1.d = collect_2.d = update_2.d$; (C8) $access_1.v = collect_2.v = update_2.v$; and (C9) $collect_2$ or $update_2$ is in the solution. The new ϕ_g is satisfiable, but $\phi_g^\perp$ is UNSAT because $update_2 \notin D_\downarrow$ and $update_1 \neq update_2$ (C8 conflicts with C3). Therefore, $D_\downarrow$ needs to be expanded. Assume $collect_2$ is added to $D_\downarrow$.

4th iteration: $D_\downarrow = \{access_1, update_1, collect_2\}$ and $Reqs_\downarrow = \{req_2\}$. The over-approximation ϕ_g stays the same, but $\phi_g^\perp$ becomes satisfiable since $collect_2$ is in $D_\downarrow$. Suppose the solution is σ_3 (see Fig. 2). Since σ_3 violates req_1, req_1 is added to $Reqs_\downarrow$.

5th iteration: $D_\downarrow = \{access_1, update_1, collect_2\}$ and $Reqs_\downarrow = \{req_1, req_2\}$. The following constraints are added to ϕ_g (due to req_1): (C9) $\neg(update_2.time - 168 \leq collect_1.time \leq update_2.time)$. Since (C9) conflicts with (C8), (C7) and (C1), $update_2$ cannot be in the solution to ϕ_g. The over-approximation ϕ_g is satisfiable if $collect_1$ (introduced in the 1st iteration) or $update_2$ (3rd iteration) are in the solution. However, $\phi_g^\perp$ is UNSAT since $D_\downarrow$ does not contain $collect_1$ or $update_2$. Thus, $D_\downarrow$ is expanded. Assume $update_2$ is added to $D_\downarrow$.

6th iteration: $D_\downarrow = \{access_1, update_1, collect_2, update_2\}$, $Reqs_\downarrow = \{req_1, req_2\}$. The following constraints are added to ϕ_g (C10) $update_2.time \geq update_1.time + 168$ (due to req_1) and (C11) $update_2.time \leq update_1.time$ (due to $\neg P$). Since (C10) conflicts with (C11), $update_2$ cannot be in the solution to ϕ_g. Thus, ϕ_g

is satisfiable only if $collect_1$ is in the solution. However, $\phi_g^{\perp}$ is UNSAT because $collect_1 \notin D_{\downarrow}$. Therefore, $D_{\downarrow}$ is expanded by adding $collect_1$.

final iteration: $D_{\downarrow} = \{access_1, update_1, collect_2, update_2, collect_1\}$ and $Reqs_{\downarrow} = \{req_1, req_2\}$. The under-approximation $\phi_g^{\perp}$ becomes satisfiable, and yields the solution σ_5 in Fig. 2 which satisfies both req_1 and req_2.

6 Evaluation

To evaluate our approach, we developed a prototype tool, called LEGOS, that implements our MFOTL bounded satisfiability checking algorithm, IBS (Alg. 1). It includes Python API for specifying system requirements and MFOTL safety properties. We use pySMT [13] to formulate SMT queries and Z3 [8] to check their satisfiability. The implementation and the evaluation artifacts are included in the supplementary material [11]. In this section, we evaluate the effectiveness of our approach using five case studies, aiming to answer the following research question: *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 five case studies considered in this paper are summarized below: (1) PHIM (derived from [10,1]): a computer system for keeping track of personal health information with cost management; (2) CF@H[1]: a system for monitoring COVID patients at home and enabling doctors to monitor patient data; (3) PBC [4]: an approval policy for publishing business reports within a company; (4) BST [4]: a banking system that processes customer transactions; and (5) NASA [25]: an automated air-traffic control system design that aims to avoid aircraft collisions. [2] Tbl. 1 gives their statistics. For each case study, we record the number of requirements, relations, relation arguments, and properties, denoted as $\#reqs$, $\#rels$, $\#args$, and $\#props$, respectively. Additionally, Tbl. 1 shows initial configurations used in our experiments, with number of custodians ($\#c$), patients ($\#p$), and data ($\#d$) for PHIM; number of users ($\#u$), and data ($\#d$) for CF@H and PBC; number of employees ($\#e$), customers ($\#c$), transactions ($\#t$), and the maximum amount for a transaction (sup) for BST; number of ground-separated ($\#GSEP$) and of the self-separating aircraft ($\#SSEP$) for NASA.

[1] `https://covidfreeathome.org/`

[2] The requirements and properties for the NASA case study are originally expressed in LTL, which is subsumed by MFOTL.

names	case study statistics				configuration
	$\#reqs$	$\#rels$	$\#args$	$\#props$	
PHIM	18	22	$[1-4]$	6	$\#c=2,\ \#p=2$ $\#d=5$
CF@H	45	28	$[2-3]$	7	$\#u=2,\ \#d=10$
PBC	14	7	$[1-2]$	1	$\#u=5,\ \#d=10$
BST	10	3	$[1-3]$	3	$\#e=1,\ \#c=2$ $\#t=4,\ sup=10$
NASA	194	10	$[6-79]$	6	$\#GSEP=3$ $\#SSEP=0$ $\#GSEP=2$ $\#SSEP=2$

Table 1. Case study statistics.

NASA	configuration 1						configuration 2					
	IBS			nuXmv			IBS			nuXmv		
	out.	time (sec)	mem. (MB)	out.	time (sec)	mem. (MB)	out.	time (sec)	mem. (MB)	out.	time (sec)	mem. (MB)
na_1	U	0.80	154	U	0.88	82	U	0.13	141	U	1.65	90
na_2	U	0.16	141	U	0.47	70	U	0.15	141	U	1.50	90
na_3	U	0.16	141	U	0.49	83	U	0.13	141	U	1.48	90
na_4	U	0.77	80	U	0.54	83	U	0.15	66	U	1.43	91
na_5	U	0.14	140	U	0.52	82	U	0.15	141	U	1.43	90
na_6	U	0.03	62	U	0.57	72	U	0.03	62	U	1.40	90

Table 2. Performance comparison between IBS and nuXmv on case study NASA.

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-word 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 (PHM, PBC, and BST) to 79 (NASA).

Experimental setup. Given a set of requirements, data constraints and properties of interest for each case study, we measured the run-time (time) and peak memory usage (mem.) of performing bounded satisfiability checking of MFOTL properties, and the volume vol_σ (the number of relational objects) of the solution (σ) with (op) and without (nop) the optimality guarantees (see Remark 1 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 constraints of the case-study requirements. The initial configuration (small) is described in Tbl. 1 and the initial bound is 10. 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. As we noted earlier in Sec. 4, the purpose of adding data constraints is to avoid unrealistic counterexamples. For example, the NASA case study uses a data set for specifying the possible system control modes and uses data ranges to restrict the possible measures from the aircraft (e.g., aircraft's trajectory). In the other case studies, data constraints are realistic data ranges (e.g., a patient's account balance should be non-negative). To study the performance of our approach relative to existing work, our second experiment considers two configurations of the NASA case study verified in [23]

<table>
<tr><th colspan="2" rowspan="2">case studies</th><th colspan="6">small</th><th colspan="6">medium</th><th colspan="6">big</th><th colspan="6">unbounded</th></tr>
<tr><th colspan="2">out.</th><th colspan="2">time (sec)</th><th colspan="2">mem. (MB)</th><th colspan="2">out.</th><th colspan="2">time (sec)</th><th colspan="2">mem. (MB)</th><th colspan="2">out.</th><th colspan="2">time (sec)</th><th colspan="2">mem. (MB)</th><th colspan="2">out.</th><th colspan="2">time (sec)</th><th colspan="2">mem. (MB)</th></tr>
<tr><th></th><th></th><th>nop</th><th>op</th><th>nop</th><th>op</th><th>nop</th><th>op</th><th>nop</th><th>op</th><th>nop</th><th>op</th><th>nop</th><th>op</th><th>nop</th><th>op</th><th>nop</th><th>op</th><th>nop</th><th>op</th><th>nop</th><th>op</th><th>nop</th><th>op</th><th>nop</th><th>op</th></tr>
<tr><td rowspan="7">PHIM</td><td>ph_1</td><td colspan="2">U</td><td>0.04</td><td>0.03</td><td>29</td><td>29</td><td colspan="2">U</td><td>0.03</td><td>0.03</td><td>136</td><td>136</td><td colspan="2">U</td><td>0.04</td><td>0.04</td><td>136</td><td>136</td><td colspan="2">U</td><td>0.06</td><td>0.05</td><td>64</td><td>64</td></tr>
<tr><td>ph_2</td><td colspan="2">U</td><td>0.03</td><td>0.03</td><td>138</td><td>138</td><td colspan="2">U</td><td>0.03</td><td>0.03</td><td>136</td><td>137</td><td colspan="2">U</td><td>0.03</td><td>0.04</td><td>136</td><td>136</td><td colspan="2">U</td><td>0.05</td><td>0.06</td><td>64</td><td>61</td></tr>
<tr><td>ph_3</td><td colspan="2">U</td><td>0.03</td><td>0.03</td><td>134</td><td>137</td><td colspan="2">U</td><td>0.03</td><td>0.03</td><td>138</td><td>138</td><td colspan="2">U</td><td>0.05</td><td>0.05</td><td>137</td><td>138</td><td colspan="2">U</td><td>0.06</td><td>0.06</td><td>64</td><td>64</td></tr>
<tr><td>ph_4</td><td colspan="2">U</td><td>0.04</td><td>0.04</td><td>136</td><td>138</td><td colspan="2">U</td><td>0.04</td><td>0.04</td><td>138</td><td>135</td><td colspan="2">U</td><td>0.05</td><td>0.05</td><td>138</td><td>138</td><td colspan="2">U</td><td>0.06</td><td>0.07</td><td>64</td><td>64</td></tr>
<tr><td>ph_5</td><td colspan="2">U</td><td>0.02</td><td>0.02</td><td>135</td><td>135</td><td colspan="2">U</td><td>0.02</td><td>0.02</td><td>608</td><td>608</td><td>56</td><td>56</td><td>30.51</td><td>30.51</td><td>390</td><td>390</td><td>56</td><td>56</td><td>21.64</td><td>21.60</td><td>393</td><td>390</td></tr>
<tr><td>ph_6</td><td colspan="2">b-U</td><td>0.18</td><td>0.20</td><td>139</td><td>139</td><td colspan="2">U</td><td>0.72</td><td>0.82</td><td>144</td><td>144</td><td colspan="2">U</td><td>0.88</td><td>0.70</td><td>142</td><td>142</td><td colspan="2">U</td><td>0.91</td><td>0.91</td><td>70</td><td>70</td></tr>
<tr><td>ph_7</td><td colspan="2">U</td><td>0.11</td><td>0.11</td><td>139</td><td>139</td><td>29</td><td>29</td><td>13.80</td><td>1905.40</td><td>193</td><td>599</td><td>**30**</td><td>**29**</td><td>20.25</td><td>682.22</td><td>193</td><td>601</td><td>**32**</td><td>**29**</td><td>20.96</td><td>1035.87</td><td>123</td><td>383</td></tr>
<tr><td rowspan="7">CF@H</td><td>cf_1</td><td colspan="2">b-U</td><td>4.80</td><td>6.90</td><td>114</td><td>176</td><td colspan="2">U</td><td>2.87</td><td>3.55</td><td>81</td><td>86</td><td colspan="2">U</td><td>2.98</td><td>1.71</td><td>85</td><td>76</td><td colspan="2">U</td><td>1.71</td><td>0.74</td><td>74</td><td>68</td></tr>
<tr><td>cf_2</td><td colspan="2">b-U</td><td>0.87</td><td>0.93</td><td>70</td><td>70</td><td>14</td><td>14</td><td>3.21</td><td>425.41</td><td>79</td><td>334</td><td>14</td><td>14</td><td>2.40</td><td>778.36</td><td>76</td><td>80</td><td>14</td><td>14</td><td>3.32</td><td>16.97</td><td>80</td><td>205</td></tr>
<tr><td>cf_3</td><td colspan="2">b-U</td><td>1.38</td><td>1.31</td><td>145</td><td>145</td><td>16</td><td>16</td><td>6.05</td><td>90.78</td><td>168</td><td>403</td><td>16</td><td>16</td><td>3.54</td><td>371.65</td><td>157</td><td>846</td><td>16</td><td>16</td><td>5.35</td><td>24.07</td><td>86</td><td>164</td></tr>
<tr><td>cf_4</td><td colspan="2">b-U</td><td>1.52</td><td>0.73</td><td>74</td><td>68</td><td>14</td><td>14</td><td>4.54</td><td>65.59</td><td>90</td><td>261</td><td>14</td><td>14</td><td>5.63</td><td>57.30</td><td>95</td><td>261</td><td>14</td><td>14</td><td>5.65</td><td>1227.02</td><td>89</td><td>294</td></tr>
<tr><td>cf_5</td><td>8</td><td>8</td><td>1.20</td><td>1.17</td><td>146</td><td>147</td><td>8</td><td>8</td><td>0.48</td><td>0.54</td><td>141</td><td>142</td><td>8</td><td>8</td><td>0.69</td><td>0.57</td><td>141</td><td>141</td><td>8</td><td>8</td><td>0.72</td><td>0.76</td><td>69</td><td>69</td></tr>
<tr><td>cf_6</td><td>8</td><td>8</td><td>1.06</td><td>1.16</td><td>146</td><td>147</td><td>8</td><td>8</td><td>0.52</td><td>0.61</td><td>142</td><td>142</td><td>8</td><td>8</td><td>0.60</td><td>0.73</td><td>141</td><td>141</td><td>8</td><td>8</td><td>0.72</td><td>0.72</td><td>69</td><td>69</td></tr>
<tr><td>cf_7</td><td colspan="2">U</td><td>0.58</td><td>0.58</td><td>141</td><td>142</td><td colspan="2">U</td><td>0.38</td><td>0.36</td><td>140</td><td>141</td><td colspan="2">U</td><td>0.47</td><td>0.44</td><td>140</td><td>141</td><td colspan="2">U</td><td>0.30</td><td>0.34</td><td>66</td><td>67</td></tr>
<tr><td>PBC</td><td>pb_1</td><td colspan="2">U</td><td>0.04</td><td>0.04</td><td>29</td><td>140</td><td colspan="2">U</td><td>0.16</td><td>0.17</td><td>140</td><td>139</td><td>9</td><td>9</td><td>0.28</td><td>0.29</td><td>141</td><td>141</td><td>9</td><td>9</td><td>0.27</td><td>0.28</td><td>67</td><td>67</td></tr>
<tr><td rowspan="3">BST</td><td>bs_1</td><td colspan="2">U</td><td>0.04</td><td>0.03</td><td>64</td><td>63</td><td colspan="2">U</td><td>0.29</td><td>0.24</td><td>70</td><td>68</td><td colspan="2">U</td><td>0.31</td><td>0.30</td><td>69</td><td>68</td><td colspan="2">U</td><td>0.25</td><td>0.25</td><td>69</td><td>69</td></tr>
<tr><td>bs_2</td><td>2</td><td>2</td><td>0.04</td><td>0.04</td><td>62</td><td>64</td><td>2</td><td>2</td><td>0.04</td><td>0.04</td><td>62</td><td>62</td><td>2</td><td>2</td><td>0.04</td><td>0.04</td><td>64</td><td>64</td><td>2</td><td>2</td><td>0.04</td><td>0.04</td><td>64</td><td>64</td></tr>
<tr><td>bs_3</td><td colspan="2">U</td><td>0.02</td><td>0.02</td><td>62</td><td>62</td><td>5</td><td>5</td><td>0.4</td><td>0.9</td><td>70</td><td>73</td><td>5</td><td>5</td><td>0.39</td><td>0.85</td><td>70</td><td>74</td><td>5</td><td>5</td><td>0.40</td><td>0.70</td><td>70</td><td>72</td></tr>
</table>

Table 3. Run-time performance for four case studies and 18 properties. We record the outcome (out.) of the algorithm with (op) or without (nop) the optimal solution guarantee: UNSAT (U), bounded-UNSAT (b-U), or the volume of the counterexample σ (a natural number, corresponding to vol_σ). We consider four different configurations: small (see Tab. 6), medium (x10), big (x100), and unbounded (∞) data domain constraints and volume bound. Volume differences between op and nop are bolded.

using the state-of-the-art symbolic model checker nuXmv [6][3]. We compare our approach's result against the reproduced result of nuXmv verification. For both experiments, we report the analysis outcomes, i.e., the volume of the satisfying solution (if one exists), 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 of RAM, and running 64-bit Ubuntu GNU/Linux 8.

Results of the first experiment are summarized in Tbl. 3. 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 sizes. More precisely, we observe that it takes under three seconds to return UNSAT and between .04 seconds (bs_2:medium) and 32 minutes (ph_7:medium:op) to return a solution. In the worst case, op took 32 minutes for checking ph_7 where the property and requirements contain complex constraints. Effectively, ph_7 requires the deletion of data stored at id 10, while the cost of deletion increases over time under PHIM's requirements. 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.

[3] LEGOS solved all configurations from the NASA case study; see the results in [11]. For comparison, we report only on the configurations that are explicitly supported by nuXmv.

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 counterexamples (i.e., the path is guided by the solution from the SMT solver (Alg.1-L:13)), 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 IBS when the satisfiability of the instances does not depend on them, which is the case for all the instances except for ph_{6-7}:small, cf_{1-3}:small, and bs_3: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 the small, medium and large 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 op), in contrast to finding a satisfying solution without the optimal guarantee (by nop), imposes a substantial computational cost while rarely achieving a volume reduction. The non-optimal heuristic nop often outperformed the optimal approach for satisfiable instances. Out of 31 satisfiable instances, nop solved 12 instances 3 times faster, 10 instances 10 times faster and seven instances 20 times faster than op. Compared to the non-optimal solution, the optimal solution reduced the volume for only two instances: ph_7:large and ph_7:unbounded by one (3%) and three (9%), respectively. On all other satisfying instances, op and nop both find the optimal solutions. When there is no solution, both op and nop are equally efficient.

Results of the second experiment are summarized in Tbl. 2. Our approach and nuXmv both correctly verified that all six properties were UNSAT in both NASA configurations. We observe that the 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 MFOTL 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 a state-of-the-art symbolic model checker. Importantly, IBS can often determine satisfiability of instances without reaching the volume bound, and its performance is not sensitive to the data domain. On the other hand, IBS's optimal guarantee imposes a substantial computational cost while rarely achieving a volume reduction over non-optimal solutions obtained by nop. We need to investigate the trade-off between optimality and efficiency, as well as evaluate the performance of IBS on a broader range of benchmarks.

7 Related Work

Below, we compare with the existing approaches that address the satisfiability checking of temporal logic and first-order logic.

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 [34], by applying automata-based techniques [24], or by SAT solving [21,22,20,5]. Satisfiability checking for metric temporal logic (MTL) [31] and its variants, e.g., mission-time LTL [23] 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 [7,32] 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 [16,15] such as E-matching [9,26] and conflict-based instantiation [33]. 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 IBS, aimed to enable verification of legal properties, expressed in MFOTL, against system requirements. IBS first translates MFOTL formulas to first-order logic with relational objects (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. IBS starts with a small search space and incrementally expands it until an answer is returned or until the bound is exceeded. We implemented IBS on top of the SMT solver Z3. Experiments using five case studies showed that our approach is effective for identifying errors in requirements from different application domains. Our approach is currently limited to verifying safety properties. In the future, we plan to extend our approach so that it can handle a broader spectrum of property types, including liveness and fairness. IBS's performance and scalability depend crucially on how the domain of relational objects is maintained and expanded. As future work, we would like to study the effectiveness of other heuristics to improve IBS's scalability (e.g., random restart and expansion with domain-specific heuristics). We also aim to study how to learn/infer MFOTL properties during search to further improve the efficiency of our approach.

References

1. Arfelt, E., Basin, D.A., Debois, S.: Monitoring the GDPR. In: Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I. LNCS, vol. 11735, pp. 681–699. Springer (2019). https://doi.org/10.1007/978-3-030-29959-0_33
2. Bae, K., Lee, J.: Bounded model checking of signal temporal logic properties using syntactic separation. Proc. ACM Program. Lang. **3**(POPL), 51:1–51:30 (2019). https://doi.org/10.1145/3290364
3. Basin, D.A., Klaedtke, F., Müller, S.: Policy Monitoring in First-Order Temporal Logic. In: Proceedings of the 22nd International Conference on Computer Aided Verification CAV'2010, Edinburgh, UK. LNCS, vol. 6174, pp. 1–18. Springer (2010). https://doi.org/10.1007/978-3-642-14295-6_1
4. Basin, D.A., Klaedtke, F., Müller, S., Zalinescu, E.: Monitoring Metric First-Order Temporal Properties. J. ACM **62**(2), 15:1–15:45 (2015). https://doi.org/10.1145/2699444
5. Bersani, M.M., Frigeri, A., Morzenti, A., Pradella, M., Rossi, M., Pietro, P.S.: Constraint LTL satisfiability checking without automata. J. Appl. Log. **12**(4), 522–557 (2014). https://doi.org/10.1016/j.jal.2014.07.005
6. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv Symbolic Model Checker. In: CAV. pp. 334–342 (2014)
7. Claessen, K., Sörensson, N.: New techniques that improve MACE-style finite model finding. In: Proceedings of the CADE-19 Workshop: Model Computation-Principles, Algorithms, Applications. pp. 11–27. Citeseer (2003)
8. De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: International conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 337–340. Springer (2008)
9. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking. J. ACM **52**(3), 365–473 (2005). https://doi.org/10.1145/1066100.1066102
10. Feng, N., Marsso, L., Garavel, H.: Health Record. Model checking context model (MCC'21), Dept. of Computer Science - University of Toronto (2021), `https://mcc.lip6.fr/pdf/HealthRecord-form.pdf`
11. Feng, N., Marsso, L., Sabetzadeh, M., Chechik, M.: Supplementary Material for: Early Verification of Legal Compliance via Bounded Satisfiability Checking (2023), `https://github.com/agithubuserseva/IBSC`
12. Garavel, H., Graf, S.: Formal Methods for Safe and Secure Computers Systems. Altros (2013)
13. Gario, M., Micheli, A.: Pysmt: a solver-agnostic library for fast prototyping of smt-based algorithms. In: SMT Workshop 2015 (2015)
14. Gastin, P., Moro, P., Zeitoun, M.: Minimization of Counterexamples in SPIN. In: Proceedings of the 11th International Workshop on Model Checking Software (SPIN'04), Barcelona, Spain. LNCS, vol. 2989, pp. 92–108. Springer (2004). https://doi.org/10.1007/978-3-540-24732-6_7
15. Ge, Y., Barrett, C.W., Tinelli, C.: Solving Quantified Verification Conditions Using Satisfiability Modulo Theories. In: Proceedings of the 21st International Conference on Automated Deduction (CADE'2007), Bremen, Germany. LNCS, vol. 4603, pp. 167–182. Springer (2007). https://doi.org/10.1007/978-3-540-73595-3_12
16. Ge, Y., de Moura, L.M.: Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories. In: Proceedings of the 21st International Conference

on Computer Aided Verification (CAV'2009), Grenoble, France. LNCS, vol. 5643, pp. 306–320. Springer (2009). https://doi.org/10.1007/978-3-642-02658-4_25
17. Hallé, S., Villemaire, R.: Runtime Enforcement of Web Service Message Contracts with Data. IEEE Trans. Serv. Comput. **5**(2), 192–206 (2012). https://doi.org/10.1109/TSC.2011.10
18. Hublet, F., Basin, D.A., Krstic, S.: Real-time policy enforcement with metric first-order temporal logic. In: Atluri, V., Pietro, R.D., Jensen, C.D., Meng, W. (eds.) Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part II. Lecture Notes in Computer Science, vol. 13555, pp. 211–232. Springer (2022). https://doi.org/10.1007/978-3-031-17146-8_11, https://doi.org/10.1007/978-3-031-17146-8_11
19. Legislative Assembly of Ontario: Personal Health Information Protection Act (PHIPA) (2004), https://www.ontario.ca/laws/statute/04p03
20. Li, J., Pu, G., Zhang, L., Vardi, M.Y., He, J.: Accelerating LTL satisfiability checking by SAT solvers. J. Log. Comput. **28**(6), 1011–1030 (2018), https://doi.org/10.1093/logcom/exy013
21. Li, J., Pu, G., Zhang, Y., Vardi, M.Y., Rozier, K.Y.: SAT-based explicit LTLf satisfiability checking. Artif. Intell. **289**, 103369 (2020). https://doi.org/10.1016/j.artint.2020.103369
22. Li, J., Rozier, K.Y., Pu, G., Zhang, Y., Vardi, M.Y.: SAT-Based Explicit LTLf 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. pp. 2946–2953. AAAI Press (2019), https://doi.org/10.1609/aaai.v33i01.33012946
23. Li, J., Vardi, M.Y., Rozier, K.Y.: Satisfiability Checking for Mission-Time LTL. In: Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II. LNCS, vol. 11562, pp. 3–22. Springer (2019). https://doi.org/10.1007/978-3-030-25543-5_1
24. Li, J., Zhang, L., Pu, G., Vardi, M.Y., He, J.: LTL Satisfiability Checking Revisited. In: Proceedings of the 20th International Symposium on Temporal Representation and Reasoning, Pensacola, FL, USA, 2013. pp. 91–98. IEEE Computer Society (2013). https://doi.org/10.1109/TIME.2013.19
25. Mattarei, C., Cimatti, A., Gario, M., Tonetta, S., Rozier, K.Y.: Comparing Different Functional Allocations in Automated Air Traffic Control Design. In: Formal Methods in Computer-Aided Design (FMCAD'2015), Austin, Texas, USA. pp. 112–119. IEEE (2015)
26. de Moura, L.M., Bjørner, N.: Efficient E-Matching for SMT Solvers. In: Proceedings of the 21st International Conference on Automated Deduction (CADE'2007), Bremen, Germany. LNCS, vol. 4603, pp. 183–198. Springer (2007). https://doi.org/10.1007/978-3-540-73595-3_13
27. Narodytska, N., Bacchus, F.: Maximum Satisfiability Using Core-Guided MaxSAT Resolution. In: Proceedings of the 28th International Conference on Artificial Intelligence (AAAI'14), Québec City, Canada. pp. 2717–2723. AAAI Press (2014), http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8513
28. Papadimitriou, C.H.: On the complexity of integer programming. J. ACM **28**(4), 765–768 (1981). https://doi.org/10.1145/322276.322287

29. Passmore, G.O., Cruanes, S., Ignatovich, D., Aitken, D., Bray, M., Kagan, E., Kanishev, K., Maclean, E., Mometto, N.: The imandra automated reasoning system (system description). In: Peltier, N., Sofronie-Stokkermans, V. (eds.) Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12167, pp. 464–471. Springer (2020). https://doi.org/10.1007/978-3-030-51054-1_30, https://doi.org/10.1007/978-3-030-51054-1_30
30. Post, E.L.: Recursive Unsolvability of a Problem of Thue. J. Symb. Log. **12**(1), 1–11 (1947). https://doi.org/10.2307/2267170
31. Pradella, M., Morzenti, A., San Pietro, P.: Bounded satisfiability checking of metric temporal logic specifications. ACM Trans. Softw. Eng. Methodol. **22**(3), 20:1–20:54 (2013). https://doi.org/10.1145/2491509.2491514
32. Reynolds, A., Tinelli, C., Goel, A., Krstic, S., Deters, M., Barrett, C.W.: Quantifier Instantiation Techniques for Finite Model Finding in SMT. In: Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings. LNCS, vol. 7898, pp. 377–391. Springer (2013). https://doi.org/10.1007/978-3-642-38574-2_26
33. Reynolds, A., Tinelli, C., de Moura, L.M.: Finding Conflicting Instances of Quantified Formulas in SMT. In: Formal Methods in Computer-Aided Design (FMCAD'2014), Lausanne, Switzerland. pp. 195–202. IEEE (2014). https://doi.org/10.1109/FMCAD.2014.6987613
34. Rozier, K.Y., Vardi, M.Y.: LTL Satisfiability Checking. In: Proceedings of the 14th International Workshop on Model Checking Software (SPIN'07), Berlin, Germany. LNCS, vol. 4595, pp. 149–167. Springer (2007). https://doi.org/10.1007/978-3-540-73370-6_11
35. Shan, L., Sangchoolie, B., Folkesson, P., Vinter, J., Schoitsch, E., Loiseaux, C.: 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. pp. 71–72. IEEE (2019). https://doi.org/10.1109/EDCC.2019.00023
36. Suter, P., Köksal, A.S., Kuncak, V.: Satisfiability modulo recursive programs. In: Yahav, E. (ed.) Static Analysis - 18th International Symposium, SAS 2011, Venice, Italy, September 14-16, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6887, pp. 298–315. Springer (2011). https://doi.org/10.1007/978-3-642-23702-7_23, https://doi.org/10.1007/978-3-642-23702-7_23

Appendix

Sec. A provides the correctness proof for the constructions of over- and under-approximation queries; Sec. B studies its correctness (Th. 2), termination (Th. 3) and optimality (Th. 4).

A Correctness Proof of Over- and Under- Approximation

In this section, we prove the correctness of the over and under-approximation (Lemma 2 and Lemma 3).

Proposition 1. *For every FOL* formula ϕ_f and domain $D_\downarrow$, the grounded formula $\phi_g = G(\phi_f, D_\downarrow)$ is quantifier-free and contains a finite number of variables and terms.*

Proof. We note that (1) quantifiers are limited only to relational objects for FOL* formula ϕ_f, and they are eliminated by G; (2) since the number of a relational objects in the domain $D_\downarrow$ is finite, each $\forall$ is expanded into conjunctions of a finite number of terms; (3) finally, since the formula ϕ_f is finite and does not contain cyclic reference, the number of times that G is invoked during $G(\phi_f)$ is always finite. Combining (1), (2) and (3), we obtain that ϕ_g is quantifier-free and contains a finite number of variables and terms. □

We now present proof of correctness for the over-approximation (Lemma 2)

Proof of Lemma 2. Suppose ϕ_g is UNSAT but there exists a solution v_f for ϕ_f in some domain D (D may be different from $D_\downarrow$). We show that we can always construct a solution v_g that satisfies ϕ_g, which causes a contradiction. First, we construct a solution v'_g for $\phi'_g = G(\phi_f, D)$ from the solution v_f (for ϕ_f). Then, we construct a solution v_g for ϕ_g from the solution v'_g for ϕ'_g.

We can construct a solution v'_g for ϕ'_g in $D \cup NewRs$ where $NewRs$ are the new relational objects added by G. The encoding of G uses the standard way for grounding universally quantified expression by enumerating every relational object in D (L:5). For every existentially quantified expression, there exists some relation object $o \in D$ enabled by v_f (i.e., $v_f(o.ext) = \top$) that satisfies the expression in ϕ_f, whereas ϕ'_g contains a new relational object $o' \in NewRs$ for satisfying the same expression (L:3). Let $v_f(o) = v'_g(o')$ for o and o', and then v'_g is a solution to ϕ'_g.

To construct the solution v_g for $\phi_g = G(\phi_f, D_\downarrow)$ from the solution v'_g for $\phi'_g = G(\phi_f, D)$, we consider expansion of the universally quantified expression in ϕ_f (L:4). For every relational objects in $o^+ \in D \setminus D_\downarrow$, G creates constraints (L:5) in ϕ'_g, but not in ϕ_g. On the other hand, for every relational object in $o^- \in D_\downarrow \setminus D$, we disable o^- in the solution v_g by assigning $o_g(r^-.ext) \leftarrow \bot$. Therefore, the constraints instantiated by o^- (at L:5) in ϕ_g are vacuously satisfied.

For every relational object $o \in D_\downarrow \cap D$, we let $v_g(o) = v'_g(o)$, and all shared constraints in ϕ_g and ϕ'_g are satisfied by v_g and v'_g, respectively. Therefore, v_g is a solution to ϕ_g. Contradiction. □

We now present proof of correctness for the over-approximation (Lemma 3)

Proof of Lemma 3. If σ is a solution to $\phi_g^\perp$ in the domain $D_\downarrow \cup NewRs$, then we can construct a solution σ' to ϕ_f in the domain $D_\downarrow$. The construction of σ' simply ignores any relational object in σ that does not appear in $D_\downarrow$ (i.e., the ones in $NewRs$). The solution σ' is valid for ϕ_f in $D_\downarrow$ because for every ignored relational object o, $\textsc{NoNewR}(NewRs, D_\downarrow)$ guarantees that some relational object $o' \in D_\downarrow$ is semantically equivalent to o. Therefore, if an existentially quantified expression is satisfied by o, it is also satisfied by o'. On the other hand, universally quantified expression in $\phi_g^\perp$ are grounded by considering only $D_\downarrow$ (L:5 of Alg. 2), and hence σ' satisfies them. Therefore, σ' is a solution to ϕ_f in $D_\downarrow$. □

B Correctness, Termination, Optimality of IBS

In this section, we prove that algorithm IBS is correct and optimal, i.e., always finds a solution with a minimum volume. We also show that IBS terminates.

Theorem 2 (Soundness). *If the algorithm* IBS *terminates on input P, Reqs and vb, then it returns the correct result, i.e., a counter-example σ, "UNSAT" or "bounded-UNSAT", when they apply.*

Proof. Let ϕ_f be the FOL* formula $T(\neg P) \bigwedge_{\psi \in Reqs} T(\psi)$. We consider correctness of IBS for three possible outputs: the satisfying solution σ to ϕ_f (L:20), the UNSAT determination of ϕ_f (L:10), and the bounded-UNSAT determination of ϕ_f (L:17). IBS returns a satisfying solution σ only if (1) σ is a solution $\phi_g^\perp$ (L:21) and (2) $\sigma \models T(\psi)$ for every $\psi \in Reqs$ (L:19). By (1) and Lemma 3, σ is a solution to $T(\neg P) \bigwedge_{\psi \in Reqs_\downarrow} T(\psi)$. Together with (2), σ is a solution to ϕ_f. IBS returns UNSAT iff ϕ_g is UNSAT (L:9). By Lemma 2, we show $T(\neg P) \bigwedge_{\psi \in Reqs_\downarrow} T(\psi)$ is UNSAT. Since $Reqs_\downarrow \subseteq Reqs$, the original formula ϕ_f is also UNSAT. IBS returns bounded-UNSAT iff the volume of the minimum solution σ_{min} to the over-approximated query ϕ_g is larger than vb (L:16). Since ϕ_g is an over-approximation of the original formula ϕ_f, any solution σ to ϕ_f has volume at least $vol(\sigma_{min})$. Therefore, when $vol(\sigma_{min}) > vb$, $vol(\sigma) > vb$ for every solution. Finally, by Thm. 1, (1) if ϕ_f is satisfiable, then $\neg P \wedge Reqs$ is satisfiable, (2) if ϕ_f is UNSAT, then $\neg P \wedge Reqs$ is UNSAT, and (3) if ϕ_f does not have a solution with volume not less than vb, then $\neg P \wedge Reqs$ also does not have a solution with volume less than vb (bounded UNSAT). Therefore, Alg. 1 is sound for MFTOL bounded satisfiability on inputs P, $Reqs$ and vb. □

Theorem 3 (Termination). *For an input property P, requirements Reqs, and a bound $vb \neq \infty$,* IBS *eventually terminates.*

Proof. To prove that IBS always terminates when the input $vb \neq \infty$, we need to show that IBS does not get stuck at solving the SMT query via SOLVE (LL:11-9), nor refining $Reqs_\downarrow$ (LL:19-23), nor expanding $D_\downarrow$ (LL:15-18).

A call to SOLVE (LL:11-9) always terminates. By Prop. 1 both the under- and the over-approximated queries ϕ_g and $\phi_g^{\perp}$ are quantifier-free. Since the background theory for P is LIA, then ϕ_g and $\phi_g^{\perp}$ are a quantifier-free LIA formula whose satisfiability is decidable.

If the requirement checking fails on L: 19, a violating requirement $lesson$ is added to $Reqs_{\downarrow}$ (LL:22-23) which ensures that any future solution σ' satisfies $lesson$. Therefore, $lesson$ is never added to $Reqs_{\downarrow}$ more than once. Given that $Reqs$ is a finite set of MFOTL formulas, at most $|Reqs|$ lessons can be learned before the algorithm terminates.

The under-approximated domain $D_{\downarrow}$ can be expanded a finite number of times because the size of the minimum solution $vol(\sigma_{min})$ to ϕ_g (computed on L:13) is monotonically non-decreasing between each iteration of the loop (LL:5-23). The size will eventually increase since each relational object in $D_{\downarrow}$ can introduce a finite number of options for adding a new relational object through the grounded encoding of ϕ_g on L:8, e.g., $o.ext \Rightarrow \bigvee_{i=0}^{n} \exists r_i$. After exploring all options to $D_{\downarrow}$, $vol(\sigma_{min})$ must increase if the algorithm has not already terminated. Therefore, if $vb \neq \infty$, then eventually $vol(\sigma_{min}) > vb$, and the algorithm will return bounded-UNSAT instead of expanding $D_{\downarrow}$ indefinitely (LL:12-18). □

Optimality of the solution. The following theorem proves that the solution found by IBS has the minimum volume.

Theorem 4 (Solution optimality). *For a property P and requirements $Reqs$, let ϕ_f be the FOL formula $T(\neg P) \bigwedge_{\psi \in Reqs} T(\psi)$. If* IBS *finds a solution σ for ϕ_f, then for every $\sigma' \models \phi_f$, $vol(\sigma) \leq vol(\sigma')$.*

Proof. IBS returns a solution σ on L:20 only if σ is a solution to the under-approximation query $\phi_g^{\perp}$ (computed on L:8) for some domain $D_{\downarrow} \neq \emptyset$. $D_{\downarrow}$ is last expanded in some previous iterations by adding relational objects to the minimum solution σ_{min} (L:13) of the over-approximation query ϕ_g' (L:15). Therefore, the returned σ has the same number of relational objects as σ_{min} ($vol(\sigma_{min}) = vol(\sigma)$). Since ϕ_g is an over-approximation of the original formula ϕ_f, any solution σ' to ϕ_f has volume that is at least $vol(\sigma_{min})$. Therefore, $vol(\sigma) \leq vol(\sigma')$. Finally, by Thm. 1, the optimal solution of $\neg P \wedge Reqs$ has the same volume as $vol(\sigma)$. □