Diagnosis via Proofs of Unsatisfiability for First-Order Logic with Relational Objects

Anonymous Author(s)

ABSTRACT

Satisfiability-based automated reasoning is an approach that is being successfully used in software engineering to validate complex software, including for safety-critical systems. Such reasoning underlies many validation activities, from requirements analysis to design consistency to test coverage. While generally effective, the back-end constraint solvers are often complex and inevitably error-prone, which threatens the soundness of their application. Thus, such solvers need to be validated, which includes checking correctness and explaining (un)satisfiability results returned by them. In this work, we consider satisfiability analysis based on First-Order Logic with relational objects (FOL*) which has been shown to be effective for reasoning about time- and data-sensitive early system designs. We tackle the challenge of validating the correctness of FOL* unsatisfiability results and deriving diagnoses to explain the causes of the unsatisfiability. Inspired by the concept of proofs of UNSAT from SAT/SMT solvers, we define a proof format and proof rules to track the solvers' reasoning steps as sequences of derivations towards UNSAT. We also propose an algorithm to verify the correctness of FOL* proofs while filtering unnecessary derivations and develop a proof-based diagnosis to explain the cause of unsatisfiability. We implemented the proposed proof support on top of the state-of-the-art FOL* satisfiability checker to generate proofs of UNSAT and validated our approach by applying the proof-based diagnoses to explain the causes of well-formedness issues of normative requirements of software systems.

ACM Reference Format:

Anonymous Author(s). 2024. Diagnosis via Proofs of Unsatisfiability for First-Order Logic with Relational Objects. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Satisfiability-based automated reasoning is an approach that is being successfully used in software engineering to validate complex software, including for safety-critical systems [34]. Such reasoning underlies many validation activities, from requirements analysis [19–21, 29, 30] to design consistency [31] to test coverage [1].

In a satisfiability-based approach, we first encode a target analysis problem as a set of constraints whose satisfiability is checked by a constraint solver. The solver then returns either a solution under which the set of constraints is satisfiable, or UNSAT, indicating that satisfiability cannot be achieved. Such solver outputs can then be used to solve the target problem. For instance, in the context of test generation via symbolic execution [28], a satisfying solution to a path condition can be used to compute a concrete test input under which the path would be executed, while an UNSAT result means that the path is not feasible.

With the rapid advancement in the field of constraint satisfiability solving, the solvers have become progressively more advanced but also more complex and inevitably error-prone [42]. The errors in solvers threaten the overall soundness of the satisfiability-based reasoning approached, especially if the solver erroneously declares UNSAT for satisfiable constraints since we cannot validate the correctness of UNSAT by checking satisfying solutions. Therefore, it is important to validate and explain results returned by the solver.

In this work, we consider an automated reasoning approach based on a variant of first-order logic, FOL* (first-order logic with quantifiers over relational objects)[21]. FOL* has been used to specify time- and data-sensitive system designs [21] and normative requirements: social, legal, ethical, empathetic, and cultural (SLEEC) [19, 20, 22]. An FOL* satisfiability checking approach, LEGOS [21], has been developed to enable automated reasoning without the need to bound the domain for time and data, and this approach has been applied to case studies across various domains including transportation, environmental management, health, and social care [19–22].

FOL* guarantees a finite-domain property, meaning that every satisfying solution to an FOL* formula is contained within a finite domain and can thus be effectively checked for correctness. However, verifying correctness of an *unsatisfiability* result for an FOL* formula remains challenging since the result must be established across every finite but unbounded domain. Inspired by the concept of the proof of UNSAT from SAT/SMT solvers, in this paper, we introduce a method to efficiently generate and check the proof of unsatisfiability for FOL*. Our method defines proof rules for FOL* to capture derivation steps for constructing sound over-approximations in decidable first-order logic, and leverages the unsatisfiability of the over-approximation to prove the unsatisfiability of the original formula.

For example, consider a system that tries to match robots with humans on a field of one-dimentional coordinates. We say that a robot and a human match if the sum of their coordinates is 0. From the requirement stating that every human must be matched with a robot while every robot is to the right of the rightmost human, we can prove that the coordinate of the rightmost human is no larger than 0. We do so by proving the unsatisfiability of the following FOL* formula ϕ, where h_* and r_* represent instances of humans and robots, respectively, and t is the attribute storing the coordinate:

$$\phi := \exists h_1(\forall h_2 \cdot h_1.t \geq h_2.t \wedge (\exists r_1 \cdot r_1.t = -h_2.t)) \wedge (\forall r_2 \cdot r_2.t > h_1.t) \wedge (h_1.t > 0)$$

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

We can prove the unsatisfiability of ϕ by *soundly* deriving the over-approximation clauses ψ_6, ψ_7, and $r_1^o.t > h_1^o.t$ following the proof in Fig. 3 and showing that clauses ψ_6, ψ_7, and $r_1^o.t > h_1^o.t$ together are UNSAT. The derivation of the over-approximation is recorded step-by-step to create the proof of UNSAT (see Fig. 3), and each derivation step can be checked against FOL* derivation rules for correctness.

In addition to validating the *correctness* of FOL* unsatisfiability, the proof can also be used to generate diagnoses to explain the *causes* of unsatisfiability. This can be achieved by projecting the derivation steps in the proof back onto the input formula. The projection result highlights the atoms that are active for the derivation of UNSAT and enables us to slice away inactive parts of the formula without affecting the overall unsatisfiability. In the above example, projecting the UNSAT proof of ϕ shown in Fig. 3 indicates that the atom $\forall h_2 \cdot h_1.t > h_2.t$ is inactive, and hence can be sliced away in the diagnosis without affecting the unsatisfiability of ϕ. Therefore, the diagnosis of the unsatisfiability of ϕ is

$$\phi_{dig} := \exists h_1(\forall h_2(\exists r_1 \cdot r_1.t = -h_2.t)) \wedge (\forall r_2 \cdot r_2.t > h_1.t) \wedge (h_1.t > 0)$$

This means we can prove that every human (instead of just the rightmost human) must be at a coordinate ($h_1.t$) no less than zero.

Contributions. We make the following theoretical contributions: a proof format for FOL* unsatisfiability; a proof checking algorithm to verify the correctness of FOL* proof while slicing away unnecessary derivations; and an application of FOL* proofs to generate proof-based diagnoses to explain causes of unsatisfiability. We also make the following engineering contributions: an extension of the state-of-the-art FOL* satisfiability checker to generate FOL* proofs of UNSAT and an application of proof-based diagnoses to a software engineering problem of addressing causes of well-formedness violations of normative requirements. Specifically, we use eight case studies to show that proofs of unsatisfiability can lead to effective debugging of conflicting, redundant, and overly restrictive normative requirements.

The rest of the paper is organized as follows: Sec. 2 gives the background material for our work. Sec. 3 introduces the proof framework, LEGOS-PROOF, to generate proofs of unsatisfiability for FOL*. Sec. 4 describes a method to check correctness of such proofs. Sec. 5 describes how to use such proofs to compute diagnoses of FOL* unsatisfiability. Sec. 6 presents the integration of our proof support with the state of the art FOL* solver. Sec. 7 presents the evaluation of the approach's efficiency and effectiveness. Sec. 8 compares LEGOS-PROOF to related work. Sec. 9 summarizes the paper and discusses future research.

2 BACKGROUND

In this section, we briefly describe FOL* [21] and its satisfiability.

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 $\mathbb{Z}$, where the theory of linear integer arithmetic (LIA) holds. Let V be a set of variables in the domain $\mathbb{Z}$. A *relational object* $o : r$ of class $r \in R$ is an object with $\iota(r)$ regular attributes and one special attribute *ext*, where every attribute is a variable. We assume that all regular attributes are ordered and let $o[i]$ denote the ith attribute of o. Some attributes

$$\begin{array}{lll}
(D, v) \models \top & & (D, v) \not\models \bot \\
(D, v) \models o.ext & \text{iff} & v(o.ext) = \top \\
(D, v) \models t = t' & \text{iff} & v(t, D) = v(t', D) \\
(D, v) \models t > t' & \text{iff} & v(t, D) > v(t', D) \\
(D, v) \models \neg\phi & \text{iff} & (D, v) \not\models \phi \\
(D, v) \models \phi \wedge \psi & \text{iff} & (D, v) \models \phi \text{ and } (D, v) \models \psi \\
(D, v) \models \exists o : r \cdot \psi & \text{iff} & (D, v) \models \vee_{o' \in D_r}(o'.ext \wedge \psi[o \leftarrow o'])
\end{array}$$

Figure 1: FOL*+ semantics. D_r defines the set of all relational objects of class r in D.

are named, and $o.x$ refers to o's attribute with the name 'x'. Each relational object o a special attributes: $o.ext$ indicating whether o exists in a solution. For convenience, we define a function CLS($o : r$) to return the relational object's class r. Let *a term* t be defined inductively as $t :\ c \mid var \mid o[i] \mid o.x \mid t + t \mid c \times t$ for any constant $c \in C$, any variable $var \in V$, any relational object o, any index $i \in [0, \iota(r)]$ 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)* $\phi \wedge \psi$, $\neg\phi$ for FOL* formulas ϕ and ψ; *(4)* $\exists o : r \cdot (\phi)$ for an FOL* formula ϕ and a class r; *(5)* $\forall o : r \cdot (\phi)$ for an FOL* formula ϕ 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: $\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi)$ and $\forall o : r \cdot \phi = \neg\exists o : r \cdot \neg\phi$. 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 formula is in NNF.

Let D be a domain of relational objects and ϕ be an FOL* formula. A valuation function v assigns concrete values to (1) every free variable in ϕ and (2) every attribute variable for every relational object in D, including Boolean attribute *ext*. Let $v(var)$ denote the value assigned to a variable var. With slight abuse of notation, we define $v(t, D)$ to be the result of evaluating a term t in a domain D with the following rules:

$$v(t, D) = \begin{cases} c & \text{if } t \text{ is a constant } c \\ v(var) & \text{if } t \text{ is a variable } var \\ -v(t', D) & \text{if } t = -t' \\ v(t_1, D)\ op\ v(t_2, D) & \text{if } t = t_1\ op\ t_2 \text{ for } op \in \{+, \times\} \end{cases}$$

Given an FOL* formula ϕ, $(D, v) \models \phi$ means that ϕ holds in a domain D with respect to a valuation function v. The semantics of $(D, v) \models \phi$ is defined in Fig. 1. The semantic rule for quantified formula (i.e., $\exists o : r \cdot (\phi)$) expands the quantifier on relational objects according to the domain D and yields the expression $\vee^D_{o':r}(o'.ext \wedge \phi[o \leftarrow o'])$. By recursively expanding the FOL* formula (and its sub-formulas) in a given domain D, we obtain a quantifier-free formula with a bounded number of free variables, referred to as the *grounded* formula in D. The semantic relation $(D, v) \models \phi$ holds in the FOL* formula if the grounded formula of ϕ in D evaluates to $\top$ according to v. A ϕ formula is satisfiable if there *exists* a tuple (D, v) such that $(D, v) \models \phi$. (D, v) is referred as *a solution* to ϕ. Given a solution $\sigma = (D, v)$, we say that a relational object o is in σ, denoted $o \in \sigma$, if $o \in D$ and $v(o.ext)$ is true. Moreover, two relational objects o_1 and o_2 are equivalent ($o_1 \equiv o_2$) in a solution

if o_1 and o_2 are object of the same class r, and for every attribute x of class r, $v(o_1.x) = v(o_2.x)$. The *volume of the solution*, denoted as $vol(\sigma)$, is the size of the set of unique relational objects in the solution: $|\{o \mid o \in \sigma\}|$ where $\forall o_1, o_2 \in \sigma : o_1 \not\equiv o_2$.

Example 2.1. Let a be a relational object of class A with an 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 $a' \leftarrow a_2,\ a'' \leftarrow a_2$.

3 PROOF OF UNSAT

Here we first introduce the proof framework (Sec. 3.1) that explains the syntax and semantics of a FOL* proof. Then we will explain how the proof framework interact with external theory reasoning via interface derivation rules in Sec. 3.2. Finally we instantiate the framework with derivation rules specific to FOL* (Sec. 3.3).

3.1 FOL* Proof of Unsatisfiability

In this section, we present the framework for FOL* proof which is parameterized on the use of an external theory $\mathbb{T}$.

Let $\mathbb{T}$ be a quantifier-free external theory (or a theory combination), e.g., quantifier-free linear integer arithmetic (QFLIA), for which a decision procedure exists. We assume that $\mathbb{T} \subseteq$ FOL* and $\mathbb{T} \cap$ FOL* on *at least* boolean logic symbols (i.e., $\top$, $\bot$, $\wedge$, $\neg$) with consistent interpretations. The syntax and semantics of FOL* presented in Sec. 2 assume that $\mathbb{T}$ is QFLIA. However, FOL* can be extended with other background theories such as quantifier-free linear arithmetic on reals (QFLRA), mixed linear arithmetic (QFLIRA) or any quantifier-free background theory that satisfies the above requirements.

Let R be a set of input FOL* constraints. A ***UNSAT** proof of R* is a sequence of derivation steps $d_1, \ldots, d_n$ where every step d_i (formally defined later) is an application of one of the derivation rules that transforms the proof state s_{i-1} into a new state s_i. A *proof state* s_i is a tuple (R_i^+, F_i, D_i), where R_i^+ is a set of formulas in FOL* ("lemmas"); R_i is the set of formulas in $\mathbb{T}$, $F_0 \leftarrow \emptyset$ ("facts"); and D_i is a set of side-effect objects (SEOs) used to capture the side-effects of the derivations (e.g., FOL* relational objects). Initially, $D_0 = \emptyset$.

Each *derivation step* d_i is a tuple $(rule, Dep_i, F_i^\Delta, R_i^\Delta, D_i^\Delta)$, where *rule* is the name of the derivation rule being applied; $Dep_i = (R', F', D')$ is sets of dependant lemmas R_i', facts F_i' and SEOs D_i' that enable the derivation; F_i^Δ is a set of new facts added by applying d_i; R_i^Δ is a set of new lemmas derived by d_i, and D_i^Δ are new SEOs added by d_i. We say that dependencies Dep_i are *satisfied* in state s_{i-1}, denoted as $Dep_i \subseteq s_{i-1}$, if $R_i' \subseteq R_i^+$, $F_i' \subseteq F_{i-1}$, and $D_i' \subseteq D_{i-1}$. Let a state $s_{i-1} = (F_{i-1}, R_{i-1}^+, D_{i-1})$ be given. If $Dep_i \subseteq s_{i-1}$ and the *rule*-specific enabling condition, denoted as PC_{rule_i}, holds on d_i, then applying d_i adds the newly derived resources to F_i^Δ, R_i^Δ, and D_i^Δ to F_{i-1}, R_{i-1}^+, and D_{i-1}, respectively. Formally, the application of $d_i = (rule_i, Dep_i, F_i^\Delta, R_i^\Delta, D_i^\Delta)$ in the state $s_{i-1} = (F_{i-1}, R_{i-1}^+, D_{i-1})$, denoted as $s_{i-1} \circ d_i$, yields a new state $s_i = (F_i, R_i^+, D_i) =$

$$\begin{cases} \emptyset & \text{if } s_{i-1} = \emptyset \\ (F_{i-1} \cup F_i^\Delta, R_{i-1}^+ \cup R_i^\Delta, D_{i-1} \cup D_i^\Delta) & \text{if } Dep_i \subseteq s_{i-1} \wedge PC_{rule_i}(d_i) \\ \emptyset & \text{otherwise} \end{cases}$$

where $\emptyset$ is a dead-end state representing a derivation error, and function $PC(rule_i, Dep_i)$ checks sufficiency of the listed resources in Dep_i to satisfy the enabling condition specific to each *rule_i*.

Example 3.1. Suppose we introduce a derivation rule **UP** allowing to derive new facts via unit propagation in propositional logic. More specifically, given a derivation step $d_i := \{\mathbf{UP}, R_i^\Delta, F_i^\Delta, D_i^\Delta\ Dep' := \{R_i', F_i', D_i'\})$, the derivation condition is $PC_{\mathbf{UnitProp}}(d_i) = F_{i-1}' \vdash_1 F_i^\Delta \wedge R_i^\Delta = D^{\Delta_i} = \emptyset$, where $F_{i-1}' \vdash_1 F_i^\Delta$ means that every fact in F_i^Δ can be derived from facts in F_{i-1}' by unit propagation. Let a proof state $s_{i-1} := \{R_{i-1}, F_{i-1}, D_{i-1}\}$ where $\{a, \neg a \vee b, \neg b \vee c, c \vee f\} \subseteq F_i$ be given. Then the derivation step $d_i := \{\mathbf{UP}, F_i^\Delta := \{c\}, \ldots, Dep' := \{F_i' := \{a, \neg a \vee b, \neg b \vee c\}, \ldots\}\}$ on s_{i-1} is *valid* because (1) $Dep' \subseteq s_{i-1}$ and (2) $PC_{\mathbf{UP}}(d_i)$ evaluates to true. Applying d_i yields the state $s_{i-1} \circ d_i = \{R_{i-1}, F_{i-1} \cup \{c\}, D_{i-1}\}$.

Definition 3.2 (State Satisfiability). Let a state $s_i = (F_i, R_i^+, D_i)$ be given. The state s_i is *satisfiable*, denoted as $SAT(s_i)$, if the FOL* formula $(\bigwedge R_i^+) \wedge (\bigwedge F_i)$ is satisfiable. An empty state $\emptyset$ is trivially satisfiable.

Definition 3.3 (Derivation Soundness). Let a proof state s_{i-1} and a derivation d_i be given. The derivation step d_i is *sound* if the derived new state $s_{i-1} \circ d_i$ preserves the satisfiability of s_i: $SAT(s_i) \iff SAT(s_{i-1} \circ d_i)$.

Theorem 3.4 (Partial Soundness). *Let a set of FOL* formulas R and a proof $L = d_1 \ldots d_n$ be given. Suppose s_0 is the initial proof state for R and $s_n = s_0 \circ d_1 \circ \ldots d_n$ is the final proof state after applying every derivation rule in L. If (1) every derivation step is sound, (2) $s_n := (R_n, F_n, D_n) \neq \emptyset$, and (3) $\bot \in R_n$ or $\bot \in F_n$, then the conjunct of R (i.e., $\wedge R$) is UNSAT.*

3.2 Interfacing with the Background Theories

Our proof framework supports derivation of unsatisfiability via external reasoning in some parameterized background theory $\mathbb{T}$. More specifically, in a proof state $s_i = (R_i^+, F_i, D_i)$, the set of facts F is maintained separately from FOL* lemmas R^+ so that they can be used exclusively for the purpose of external theory reasoning. To enable such support, we need an interface to capture the derivation in $\mathbb{T}$ and to communicate between FOL* and the background theory.

We add the following three meta derivation rules: (1) the rule $\mathbb{T}$-**Derive** which enables external reasoning in $\mathbb{T}$ based on the facts in F; (2) the rule **FOL*** $\rightarrow \mathbb{T}$ which communicates FOL* lemmas from R^+ to F; and (3) $\mathbb{T} \rightarrow$ **FOL*** which communicates facts from F back to R^+. We formalize this below.

Definition 3.5. Let a derivation step $d_i = (rule_i, Dep_i, F_i^\Delta, R_i^\Delta, D_i^\Delta)$ and a formula ϕ_g in theory $\mathbb{T}$, where $Dep_i = (R_i', F_i', D_i')$, be given. The step d_i is a *valid application of* the $\mathbb{T}$-**Derive** rule if $PC_{\mathbb{T}\text{-}\mathbf{Derive}}(d_i) := R_i^\Delta = D_i^\Delta = \emptyset \wedge F_i^\Delta = \{\phi_g\} \wedge F_i' \models^{\mathbb{T}} \phi_g$.

That is, if there exists a subset of facts $F' \subseteq F$ such that F' derives a new fact ϕ_g in $\mathbb{T}$ (i.e., $F' \models^{\mathbb{T}} \phi_g$), then applying rule $\mathbb{T}$-**Derive** adds ϕ_g to F_i. The fact ϕ_g depends on F', denoted as $DEP(\phi_g) = F'$.

Definition 3.6. Let a derivation step $d_i = (rule_i, Dep_i, F_i^\Delta, R_i^\Delta, D_i^\Delta)$ and an FOL* formula ϕ where $Dep_i := (R_i', F_i', D_i')$ be given. The step d_i is *a valid application of the* $\textbf{FOL}^* \rightarrow \mathbb{T}$ *rule* if $PC_{\textbf{FOL}^* \rightarrow \mathbb{T}}(d_i) := R_i^\Delta = D_i^\Delta = \emptyset \wedge F_i^\Delta = \{\phi\} \wedge \phi \in R_i' \wedge \phi \in \mathbb{T}$.

That is, if $\phi \in R^+$ is in the intersection between FOL* and $\mathbb{T}$, then ϕ can be added to F as a $\mathbb{T}$ fact.

Definition 3.7. Let a derivation step $d_i = (rule_i, Dep_i, F_i^\Delta, R_i^\Delta, D_i^\Delta)$ and a $\mathbb{T}$ formula ϕ where $Dep_i := (R_i', F_i', D_i')$ be given. The step d_i is *a valid application of the* $\mathbb{T} \rightarrow \textbf{FOL}^*$ *rule* if $PC_{\mathbb{T} \rightarrow \textbf{FOL}^*}(d_i) := F_i^\Delta = D_i^\Delta = \emptyset \wedge R_i^\Delta = \{\phi\} \wedge \phi \in F_i' \wedge \phi \in \text{FOL}^*$.

In our proof framework, we can derive FOL* lemmas with FOL* derivation rules (to be presented in Sec. 3.3). For lemmas in the intersection between FOL* and the background theory $\mathbb{T}$, we can apply the derivation rules $\textbf{FOL}^* \rightarrow \mathbb{T}$ to add them as facts in F. The facts in F can then be used for external reasoning via the step $\mathbb{T}$ – **Derive** which derives additional facts in F. The newly-derived facts can be added as FOL* lemmas in R^+ via $\mathbb{T} \rightarrow \textbf{FOL}^*$ to make further progress for the derivation of UNSAT. Note that if **UNSAT** is derived via the external reasoning ($\mathbb{T}$ – **Derive**), the FOL* proof of unsatisfiability concludes as well. In such a case, the facts in F represent an over-approximation of the input FOL* formulas R as they are soundly derived from R. The unsatisfiability of the over-approximation implies the unsatisfiability of R.

3.3 FOL* Derivation Rules

In this section, we introduce FOL*-specific derivation rules which allow us to show soundness of derived FOL* lemmas.

Definition 3.8 (Definition Variable). Let ϕ be an FOL* formula. A *definition variable*, denoted as $DEF(\phi)$, is a boolean variable subject to the following two constraints ϕ^+_{def} and ϕ^-_{def} where $\phi^+_{def} := l \Rightarrow \phi$ and $\phi^-_{def} := \bar{l} \Rightarrow \neg\phi$.

The definition variables serve as boolean abstractions to FOL* formulas. The relationship between a definition variable l and a formula ϕ is captured by the positive and negative *definition clauses*, ϕ^+ *def* and ϕ^-_{def}, respectively. We assume that ϕ^+_{def} and ϕ^- *def* can be accessed via the functions DL^+ and DL^-, respectively (i.e., $DL^+(\phi) = \phi^+_{def}$ and $DL^-(\phi) = \phi^-_{def}$).

In a proof state $s_i = (R_i^+, F_i, D_i)$, the abstraction function DEF is stored as tuples in the SEO domain D_i. Given a domain of SEOs D_i, we say the ϕ *is defined in DEF*, denoted as $\phi \in DEF$, if there exists a tuple $(\phi, l) \in D_i$. We ensure that the function DEF is one-to-one, such that the inverse function DEF^{-1} can be defined (i.e., $\phi = DEF^{-1}(DEF(\phi))$). In addition to the abstraction function DEF, the *side-effect domain* D_i also contains relational objects whose classes are defined in the signature of the input FOL* formulas R.

The full set of FOL*-specific derivation rules is presented in Fig. 2, organized as the name (Name), the enabling condition ($PC_{\textbf{Name}}$), and the dependencies of each derivation rule. In the rest of the section, we explain some of these derivation rules. Their formal semantics is given in Appendix A.

Define. If ϕ is an FOL* formula and $DEF(\phi)$ is undefined, then applying the **Define** rule assigns $DEF(\phi)$ to a fresh variable and adds the definition clauses $DL^+(\phi)$ and $DL^-(\phi)$ to R^+. The derivation rule **Define** allows the introduction of new FOL* formulas in the proof of UNSAT. For example, **Define** can introduce ϕ as a hypothesis, and by deriving $DEF(\phi)$ as a lemma (via $DL^-(\phi)$), we can then use ϕ as a lemma (via $DL^+(\phi)$) in the rest of the proof.

RewriteNeg. Suppose an FOL* formula matches the pattern $\neg\psi$, where ψ is also an FOL* formula. If $DEF(\phi)$ is defined, then applying **RewriteNeg** on ϕ adds the following lemmas to R^+: (1) $\phi^+_{neg} := DEF(\psi) \Rightarrow NEG(\phi)$ and (2) $\phi^-_{neg} := \overline{DEF(\phi)} \Rightarrow \psi$, where the function NEG pushes the top-level negation into ψ and is defined in a standard way as shown in Fig. 2. Note that applying **RewriteNeg** recursively converts an FOL* formula into its negation normal form (NNF) where negation symbols appear in front of atoms (e.g., terms or literals). We can also express **RewriteAnd** with **RewriteOr** and **RewriteNeg**.

UNSAT. If $\bot$ has been derived as a fact or lemma in the current state, then the **UNSAT** rule can be applied to signal the end of the proof.

Each derivation rule simulates a key reasoning step of FOL* satisfiability: the **Define** and **Substitute** rules abstract complex FOL* formulas and capture their propositional relations. The **RewriteOr** and **RewriteAnd** rules handle case splitting. The **UniversalInst** and **ExistentialInst** rules handle quantifiers over relational objects, and **RewriteNegn** handles negation and enables NNF conversion. The **ApplyLemma** rule enforces the asserted true statements, and **Unit** enables lemma simplifications through resolution.

3.4 Derivation Macros

The FOL* proof rules are fine-grained and sometimes cumbersome to write. We noticed that there are common rule usage patterns that allow us to chain FOL* derivations to form macros to make the proof succinct:

RewriteAND*. If a formula $\phi \in R^+$ matches $\psi_1 \wedge \psi_2 \ldots \wedge \psi_n$, then ψ_1, $\psi_2 \ldots \psi_n$ can be directly added to R^+. The expansion of the macro **RewriteAND*** is shown in Appendix B.

RewriteOR*. If a formula $\phi \in R^+$ matches $\psi_1 \vee \psi_2 \ldots \vee$, then the positive definition lemma $DL^+(\phi_1) \ldots L^+(\phi_n)$ and the lemma $DEF(\psi_1) \vee DEF(\psi_2) \ldots \vee DEF(\psi_n)$ can be directly added to R^+.

UniversalInst*. If a formula $\phi \in R^+$ matches $\forall o : rp(o)$ and there exists a relational object o' of class r, then $o'.ext \Rightarrow p(o')$ can be added to R^+.

ExistenialInst*. If a formula $\phi \in R^+$ matches $\exists o : rp(o)$, then add a fresh relational object o' to D and a lemma $o'.ext \wedge p(o')$ to R^+.

The macros also avoid introducing unnecessary definition variables in lemmas, which would subsequently need to be eliminated by applying the **ApplyLemma** and the **Unit** rules (see Fig. 2).

Example 3.9. Let A and B be two classes of relational objects with a single attribute t. For brevity, we write $\forall a$ and $\forall b$ as the shorthand for $\forall a : A$ and $\forall b : B$, respectively. We want to prove the UNSAT of the FOL* formula

$$\phi := \exists h_1(\forall h_2 \cdot h_1.t \geq h_2.t \wedge (\exists r_1 \cdot r_1.t = -h_2.t)) \wedge (\forall r_2 \cdot r_2.t > h_1.t) \wedge (r_1.t > 0)$$

The initial state s^0 has an empty domain $D = \emptyset$, an empty fact set $F = \emptyset$, and $R^+ = \phi$. The derivation steps for UNSAT are shown in Fig. 3.

Name	$PC_{Name}(d_i)$, where $d_i = (Name, Dep_i, F_i^\Delta, R_i^\Delta, D_i^\Delta)$ and $Dep_i = \{R_i', F_i', D_i'\}$	Dependencies
Define	$R_i^\Delta = \{DL^+(\phi), DL^-(\phi)\} \wedge F_i^\Delta = \emptyset \wedge D_i^\Delta = \{DEF(\phi) = l\}$ and l is fresh	$\emptyset$
Subs	$F_i^\Delta = D_i^\Delta = \emptyset \wedge R_i^+ = \{\phi^+_{subs}, \phi^-_{subs}\} \wedge FV(\phi) \supseteq FV(\psi) \wedge DEF(\phi) \in D_i' \wedge DEF(\psi) \in D_i'$ where ϕ and ψ are FOL* formula $\phi^+_{subs} : DEF(\phi) \Rightarrow \phi[\psi \leftarrow DEF(\psi)]$ and $\phi^-_{subs} : \overline{DEF(\phi)} \Rightarrow \neg\phi[\psi \leftarrow DEF(\psi)]$	$\mathbf{Dep}(\phi^+_{subs}) := \{DL^+(\phi), DL^-\psi\}$ $\mathbf{Dep}(\phi^-_{subs}) := \{DL^-(\phi), DL^+(\psi)\}$
ApplyLemma	$F_i^\Delta = D_i^\Delta = \emptyset \wedge R_i^\Delta = \{DEF(\phi)\} \wedge DEF(\phi) \in D_i' \wedge \phi \in R_i'$	$\mathbf{Dep}(DEF(\phi)) := \phi$
RewriteOr	$F_i^\Delta = D_i^\Delta = \emptyset \wedge R_i^\Delta = \{\phi_{or+}, \phi^l_{or-}, \phi^r_{or-}\} \wedge DEF(\phi) \in D_i' \wedge DEF(A) \in D_i' \wedge DEF(B) \in D_i'$ where $\phi = A \vee B$ and A, B are FOL* formula $\phi_{or+} := DEF(\phi) \Rightarrow DEF(A) \vee DEF(B)$ $\phi^l_{or-} := \overline{DEF(\phi)} \Rightarrow \overline{DEF(A)}$ and $\phi^r_{or-} := \overline{DEF(\phi)} \Rightarrow \overline{DEF(B)}$	$\mathbf{Dep}(\phi_{or+}) :=$ $\{DL^+(\phi), DL^-(A), DL^-(B)\}$ $\mathbf{Dep}(\phi^l_{or-}) := \{DL^-(\phi), DL^+(A)\}$ $\mathbf{Dep}(\phi^r_{or-}) := \{DL^-(\phi), DL^+(B)\}$
ExistentialInst	$F_i^\Delta = \emptyset \wedge D_i^\Delta = \{o' : r\} \wedge R_i^\Delta = \{\phi^+_{o'}, \phi^-_{o'}\} \wedge \phi = \exists o : r \cdot p(o)$ and $DEF(\phi) \in D_i'$ where o' is a fresh relational object of class r $\phi^+_{o'} = DEF(\phi) \Rightarrow (o'.ext \wedge p(o'))$ and $\phi^-_{o'} = \overline{DEF(\phi)} \Rightarrow \neg(o'.ext \wedge p(o'))$	$\mathbf{Dep}(o') := \{DL^+(\phi)\}$ $\mathbf{Dep}(\phi^+_{o'}) := \{DL^+(\phi)\}$ $\mathbf{Dep}(\phi^-_{o'}) := \{DL^-(\phi)\}$
UniversalInst	$F_i^\Delta = D_i^\Delta = \emptyset \wedge R_i^\Delta = \{\phi_{o'}\} \wedge \phi = \forall o : r \cdot p(o) \wedge DEF(\phi) \in D_i' \wedge o' : r \in D_i'$ where $\phi_{o'} = DEF(\phi) \Rightarrow (o'.ext \Rightarrow p(o'))$	$\mathbf{Dep}(\phi_{o'}) := \{o', DL^+(\phi)\}$
RewriteNeg	$F_i^\Delta = D_i^\Delta = \emptyset \wedge R_i^\Delta = \{\phi^+_{neg}, \phi^-_{neg}\} \wedge DEF(\phi) \in D_i'$ where $\phi = \neg\psi$, $\phi^+_{neg} := DEF(\phi) \Rightarrow NEG(\psi)$ and $\phi^-_{neg} := \overline{DEF(\phi)} \Rightarrow \psi$ where $NEG(\neg\psi) = \psi$, $NEG(\psi_1 \wedge \psi_2) = \neg\psi_1 \vee \neg\psi_2$, $NEG(\psi_1 \vee \psi_2) = \neg\psi_1 \wedge \neg\psi_2$ $NEG(\exists o : r \cdot P(o)) = \forall o : r \cdot \neg P(o)$, $NEG(\forall o : r \cdot P(o)) = \exists o : r \cdot \neg P(o)$, $NEG(\psi) = \neg\psi$	$\mathbf{Dep}(\phi^+_{neg}) := \{DL^+(\phi)\}$ $\mathbf{Dep}(\phi^-_{neg}) := \{DL^-(\phi)\}$
Unit	$F_i^\Delta = D_i^\Delta = \emptyset \wedge R_i^\Delta = \{\psi\} \wedge \phi = l \Rightarrow \psi \wedge \phi \in R_i' \wedge l \in R_i'$	$\mathbf{Dep}(\psi) = \{\phi, l\}$
UNSAT	$\bot \in R_i' \vee \bot \in F_i'$	$\bot$

Figure 2: The full list of FOL*-specific derivation rules with their names, enabling condition PC_{Name}, and the dependencies of the derived resources (i.e., Dep(r) where r is a resource derived by an application of the rule *Name*). $\bar{l}$ refers to the complement of the literal l (e.g., $\overline{DEF(\phi)}$).

Each step derives new resources (colored in blue) based on FOL* derivation rules. At step 8, a subset of lemmas in R_8^+ are communicated as $\mathbb{T}$ facts to F, which serves as a sound over-approximation of ϕ. Step 9 determines that the over-approximation is unsatisfiable, and consequently derives $\bot$ to conclude **UNSAT** at step 10.

4 CHECKING FOL* PROOF OF UNSATISFIABILITY

Let a proof consisting of a sequence of derivation steps $D := d_1 \ldots d_n$, and the initial state $s_0 = (D_0, R_0^+, F_0)$ be given, where $R_0^+ = R$ is the set of input formulas, and $D_0 = F_0 = \emptyset$ is an empty set of SEOs and facts, respectively. A simple method (denoted as *forward-checking*) to verify the proof is to apply every derivation step $d_i \in D$ from the initial state s_0 in order. The proof is *valid* if every derivation step d_i is enabled in state s_{i-1}, and the final derivation step d_n applies the rule **UNSAT**.

The forward checking procedure requires enumerating every derivation step while carrying over the entire state information, which is an expensive procedure. We observe that: (1) not every derivation step in the proof contributes to the derivation of **UNSAT**, and (2) only a small portion of the state information is necessary to check a derivation step. These observations enable a potential performance gain by keeping only relevant resources (i.e., lemmas, facts, and SEOs) for the derivation of UNSAT and skipping those steps that do not produce these resources. Inspired by Heule et al. [26], we propose Alg. 1 to check and trim an FOL* proof *backwards* from the derivation of **UNSAT**.

Alg. 1 takes a set of FOL* formulas R and a proof of **UNSAT** of R, $S = d_1, \ldots, d_n$ as input and returns true if and only if S is a valid proof of R. If the proof is successfully checked, Alg. 1 computes the subset of derivations that are sufficient to derive **UNSAT** and stores them as the proof *core*. Alg. 1 uses *core* to keep track of the derivation steps on which the correctness of the proof depends. Initially, *core* contains only the final derivation of **UNSAT** d_n (line 1). Then Alg. 1 proceeds to check the proof backwards from d_n (line 2). For every derivation step d_i, Alg. 1 first checks whether the proof depends on d_i (line 3), and skips the derivation step if it does not.

For every dependent step d_i, Alg. 1 checks the rule-specific condition on d_i (line 5) and returns false if the check fails (line 11). If the rule-specific condition is successfully checked for d_i, then the listed dependent resources Dep_i in d_i are sufficient to prove d_i. Alg. 1 then checks whether the dependency Dep_i can be further reduced by invoking the procedure $Minimize_{rule_i}$ on d_i and Dep_i (line 7). Note that the minimization procedure depends on the derivation rule $rule_i$ of d_i which we briefly discuss later in the section.

After minimizing the dependent resources, Alg. 1 finds the derivation steps that derive the dependent resources prior to the application of d_i and adds them to *core* (line 9). If some dependent resources cannot be derived by any derivation step d_j where $j < i$, and the resource is not given as input (line 8), then Alg. 1 returns *False*. After updating *core*, Alg. 1 proceeds to check the next derivation step in the *core*. The algorithm terminates when every step in *core* has been checked, and at this point, *core* contains a subset of derivation steps which can derive the unsatisfiability of R.

Alg. 1 calls procedure $Minimize_{rule_i}$ to minimize the dependent resources Dep_i for checking the validity of a derivation step d_i. The minimizing procedures can be trivially derived from the definition of PC_{rule_i} with one exception: the rule $\mathbb{T}$-**Derive**. Recall that $PC_{\mathbb{T}\text{-}\mathbf{Derive}}$ states that a fact ϕ_g can be derived by d_i if $F_i' \models^{\mathbb{T}} \phi_g$, where F_i' is the set of facts in the dependent resources Dep_i listed

#	Name	Dependencies	Effect
0	**Input**	Input ϕ	$R_0^+ = \{\phi := \exists h_1(\forall h_2 \cdot h_1.t \geq h_2.t \wedge (\exists r_1 \cdot r_1.t = -h_2.t)) \wedge (\forall r_2 \cdot r_2.t > h_1.t) \wedge (h_1.t > 0)\}$
1	**ExistentialInst***	$R_1' = \{\phi\}$	$F_1 = \emptyset, D_1 = \{h_1^o\}$ and $R_1^+ = \{\phi, \phi_{h_1^o}\}$ where $\phi_{h_1^o} := h_1^o.ext \wedge (\psi_1) \wedge (\psi_2) \wedge h_1^0.t > 0$, $\psi_1 := (\forall h_2 \cdot h_1^0.t \geq h_2.t \wedge \ldots)$ and $\psi_2 := \forall r_2.r_2.t > h_1^o.t$
2	**RewriteAND***	$R_2' = \{\phi_{h_1^o}\}$	$F_2 = \emptyset, D2 = \{h_1^o\}$ and $R_2^+ = \{\phi, \phi_{h_1^o}, h_1^o.ext, \psi_1, \psi_2, h_1^o.t > 0\}$
3	**UniversalInst***	$D_3' = \{h_1^o\} \wedge$ $R_3' = \{\phi_1\}$	$F_3 = \emptyset, D_3 = \{h_1^o\}$ and $R_3^+ = \{\phi, \phi_{h_1^o}, h_1^o.ext, \psi_1, \psi_2, h_1^o.t > 0, \psi_3\}$ where $\psi_3 := h_1^o.ext \Rightarrow \psi_4$ and $\psi_4 := h_1^o.t \geq h_1^o.t \wedge \exists r_1 \cdot r_1.t = h_1^o.t$
4	**Unit**	$R_4' = \{\psi_3, h_1^0.ext\}$	$F_4 = \emptyset, D_4 = \{h_1^o\}$ and $R_4^+ = \{\phi, \phi_{h_1^o}, h_1^o.ext, \psi_1, \psi_2, h_1^o.t > 0, \psi_3, \psi_4\}$
5	**RewriteAND***	$R_5' = \{\psi_4\}$	$F_5 = \emptyset, D_5 = \{h_1^o\}$ and $R_5^+ = \{\phi, \phi_{h_1^o}, h_1^o.ext, \psi_1, \psi_2, h_1^o.t > 0, \psi_3, \psi_4, \psi_5\}$ where $\psi_5 := \exists r_1 \cdot r_1.t = h_1^o.t$
6	**ExistentialInst***	$R_6' = \{\psi_5\}$	$F_6 = \emptyset, D_6 = \{h_1^o, r_1^o\}$ and $R_6^+ = \{\phi, \phi_{h_1^o}, h_1^o.ext, \psi_1, \psi_2, h_1^o.t > 0, \psi_3, \psi_4, \psi_5, \psi_6\}$ where $\psi_6 := r_1^o.ext \wedge r_1^o.t = -h_1^o.t$
7	**UniversalInst***	$R_7' = \{\psi_2\} \wedge$ $D_7' = \{r_1^o\}$	$F_7 = \emptyset, D_7 = \{h_1^o, r_1^o\}$ and $R_7^+ = \{\phi, \phi_{h_1^o}, h_1^o.ext, \psi_1, \psi_2, h_1^o.t > 0, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7\}$ where $\psi_7 := r_1^o.ext \Rightarrow r_1^o.t > h_1^o.t$
8	FOL* $\rightarrow \mathbb{T}$	$R_8' = \{\psi_6,$ $\psi_7, h_1^o.t > 0\}$	$F_8 = \{\psi_6, \psi_7, r_1^o.t > h_1^o.t\}, D_8 = \{h_1^o, r_1^o\}$ and $R_8^+ = \{\phi, \phi_{h_1^o}, h_1^o.ext, \psi_1, \psi_2, h_1^o.t > 0, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7\}$
9	$\mathbb{T}$-**Derive**	$F_9' = \{F_8\}$	$F_9 = \{\psi_6, \psi_7, r_1^o.t > h_1^o.t, \bot\}, D_9 = \{h_1^o, r_1^o\}$ and $R_9^+ = \{\phi, \phi_{h_1^o}, h_1^o.ext, \psi_1, \psi_2, h_1^o.t > 0, \psi_3, \psi_4, \psi_5, \psi_6, \psi_7\}$
10	**UNSAT**	$F_{10}' = \{\bot\}$	**UNSAT**

Figure 3: Proof of UNSAT for ϕ in Example 3.9. For each derivation step, # is the step number, Name is the name of the applied derivation rule, Dependencies is the dependant resources Dep' for each step, and Effect is the resulting state after the derivation. The resources in blue are newly derived in each step.

Algorithm 1 Determines whether S is a valid proof of UNSAT for input R. If the proof is successfully checked, the computed *core* contains the subset of derivations sufficient to prove UNSAT.

Input: a set of FOL* formulas R, an FOL* proof $S = d_1 \ldots d_n$.
Output: $\top$ if and only if S is a valid proof of UNSAT for R.
Side effect: *core* stores a subset of S sufficient to prove UNSAT.

```
core ← {d_n}
for i ∈ (n, n − 1 . . . , 1) do
    if d_i ∉ core then Skip
    (rule_i, R_i^Δ, F_i^Δ, D_i^Δ, Dep_i) ← d_i
    if Pc_rule_i(d_i) then
        Dep* ← Minimize_rule_i(d_i, Dep_i)
        for rec ∈ Dep* do
            if rec ∈ R then Skip
            if ∃j < i · d_j derive rec then core ⊎ d_j return ⊥
    else
        return ⊥
return ⊤
```

by d_i. Therefore, $Minimize_{\mathbb{T}\text{-}\mathbf{Derive}}$ can reduce F_i' to a subset F_i'' if $F_i'' \models^{\mathbb{T}} \phi_g$.

There are many ways to compute F_i'' with trade-offs between performance and minimality (e.g., by computing the UNSAT core or the minimum unsatisfiability subset of $F' \wedge \neg\phi_g$) . In this work, we chose to minimize F' by incrementally refine the UNSAT *core* until a fixpoint or a timeout have been reached.

Example 4.1. Continuing from Example 3.9, suppose we apply Alg. 1 on the input formula ϕ and its proof of UNSAT. Alg. 1 starts from the final derivation step (step 10) which derives UNSAT:

(1) By checking step 10, the fact $\bot$ is a dependant resource, and step 9 that derived $\bot$ via $\mathbb{T}$-**Derive** is added to *core*.
(2) By checking step 9, the dependant resource F' is reduced to $F_8 = \{\psi_6, \psi_7, h_1^o.t > 0\}$, derived by step 8. Therefore, step 8, FOL* $\Rightarrow \mathbb{T}$, is added to *core*.
(3) During the checking of step 8, lemmas $\{\psi_7, \psi_6, h_1^o.t > 0\}$ are required, derived by steps 7, 6 and 2, respectively. Therefore, steps 7, 6 and 2 are added to *core*.
(4) Step 7 depends on ψ_2 and r_1^o which are derived by steps 2 and 6, respectively. Since they are already in *core*, no updates are needed.
(5) Step 6 depends on ψ_5, which is derived by step 5. Therefore, step 5 is added to *core*.
(6) Step 5 depends on ψ_4, which is derived by step 4. Therefore, step 4 is added to *core*. Step 4 depends on $h_1^o.ext$ and ϕ_3, which are derived by steps 1 and 3, respectively. Therefore, steps 1 and 3 are added to *core*.
(7) Step 3 depends on ψ_1 and h_1^0, which are derived by step 1. Since step 1 is already in *core*, no changes are needed.
(8) Step 2 depends on $\phi_{h_1^o}$ which is derived in step 1. Since step 1 is already in *core*, no changes are needed.
(9) Step 1 depends on ϕ which is given as an input.
(10) All of the derivation steps in *core* have been checked, and the proof is successfully verified. *Core* contains every step of the original proof.

5 DIAGNOSING FOL* UNSATISFIABILITY

Suppose that ϕ is an unsatisfiable FOL* formula. Understanding the causes of unsatisfiability can be a non-trivial task if ϕ is large and complex. Fortunately, the proof provides hints on how ϕ is used to derive UNSAT, and we can leverage the proof to generate a simpler (and smaller) unsatisfiable formula ϕ' that better explains the causes of unsatisfiability.

Definition 5.1 (Proof-Specific UNSAT Diagnosis). Let an FOL* formula ϕ and its proof S of *UNSAT* be given. The formula ϕ' is a *proof-specific UNSAT diagnosis of ϕ with respect to S* if: (1) ϕ' is unsatisfiable, and (2) the proof S can be applied to ϕ' to derive the unsatisfiability of ϕ'.

Intuitively, a proof-specific UNSAT diagnosis of ϕ with respect to S is a formula ϕ' such that $\phi \models \phi'$ and ϕ and ϕ' share the same causes of unsatisfiability in S. If ϕ is a conjunction of a set of FOL* formulas R, then ϕ' can be extracted as an UNSAT core of R. We can use Alg. 1 to easily compute an UNSAT core by returning $core \cap R$ at the end of proof checking. However, not every formula in the core is used fully for the derivation of UNSAT, and we can further weaken the *core* as long as the derivation of S is not affected. More specifically, ϕ' is filtered from ϕ by S at the level of boolean atoms α, which includes Boolean constants ($\top$ and $\bot$), Boolean variables, Boolean attributes of relational objects, $term_1 \cong term_2$, $\neg\alpha$, and quantifiers on relational objects (i.e., $\exists o : r$),

Definition 5.2 (Active Atoms). Let an FOL* formula ϕ and a proof of UNSAT S be given. Let $s_n = (R_n^+, F_n, D_n)$ be the final state after applying S to the formula input ϕ. An atom α is *active in S* if and only if one of the following conditions holds on s_n:

(1) every variable and relational object appeared in α is free in ϕ, and $\alpha \in R_n^+ \vee \alpha \in F_n$;
(2) for every reference to a relational object o in α, if o is bounded in ϕ (i.e., $\forall o$ or $\exists o$), then there exists a relational object $o' \in D_n$ such that $\alpha[o \mapsto o']$ is an active atom;
(3) $\neg\alpha$ appears in NNF(ϕ), and $\neg\alpha$ is active.
(4) if $\alpha = \exists o : r \vee \alpha = \forall o : r$, then there exists a relational object $o' \in D_n$ that instantiates o, and $o'.ext$ is active.

Def. 5.2 defines the criteria for determining which atoms in ϕ are used in the derivation of UNSAT. Condition (1) states that an atom without bounded variables is active if and only if it appears either as an FOL* lemma or a $\mathbb{T}$ fact (base case). Condition (2) considers the case when α contains bounded variables introduced by attributes of the bounded relational object o. α is active if there exists an instantiation $o' \in D_n$ of the bounded relational object o (by **ExistentialInst** or **UniversalInst**) such that the instantiated atom $\alpha[o \mapsto o']$ is active. Condition (2) allows us to map atoms in nested quantifiers to the instantiated atoms used for derivation. Condition (3) covers negation, where α is rewritten as $\neg\alpha$ via **RewriteNeg**. Condition (4) covers the case of quantifiers: a quantifier is *active* if the resulting atoms from instantiating the quantifier are active. Conjunctions and disjunctions are broken down into atoms (with **RewriteAND** and **RewriteOR**, respectively) where the activeness of atoms is checked separately.

Definition 5.3 (Atom-level Proof-Specific Diagnosis). Let an UNSAT FOL* formula ϕ and its proof of *UNSAT* S be given. The *atom-level proof-specific diagnosis of ϕ* is the result of substituting every non-active atom in ϕ with $\top$.

Theorem 5.4. *Let an unsatisfiable FOL* formula ϕ and its proof of UNSAT S be given. The atom-level proof-specific diagnosis of ϕ (Def. 5.3) is a proof-specific diagnosis of ϕ (Def. 5.1).*

The proof of Thm. 5.4 depends on the fact that an atom (or an instantiation of the atom) is used in the derivation of UNSAT only if it appears either as a lemma or a fact in the final state. This is true because the lemma set R^+ and the fact set F are monotonically increasing. An atom can be abstracted into $\top$ without affecting the derivation if (1) it has never been recorded as a lemma or a fact, or (2) it has never been used to instantiate lemmas or facts.

Algorithm 2 G: Ground a NNF FOL* formula ϕ in a domain $D_\downarrow$ while emitting the FOL* proof.

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

if match $(\phi, \exists o : r \cdot \phi'_f)$ **then** *//process the existential operator*
  $o' \leftarrow$ NewAct(r) *//create a new relational object of class r*
  $\phi_{o'} \leftarrow o'.ext \wedge \phi'[o \leftarrow o']$)
  ExistentialIns* $R^\Delta = \{\phi_{o'}\}$, $D^\Delta = \{o'\}$, $Deps = \{R' = \phi\}$
  $\phi_g \leftarrow$ G($\phi_{o'}, D'$)
if match $(\phi, \forall o : r \cdot \phi'_f)$ **then** *//process the universal operator*
  for $o'_i : r \in D_\downarrow$ **do**
    $\phi_{o'_i} \leftarrow o'.ext \Rightarrow \phi'_f[o \leftarrow o'_i]$
    UniverisalIns* $R^\Delta = \{\phi_{o'_i}\}$, $D^\Delta = \{o'\}$, $Deps = \{R' = \phi\}$
  $\phi_g \leftarrow \wedge_{o'_i : r \in D_\downarrow}$ G($\phi_{o_i}, D_\downarrow$)
if match $(\phi, \phi_1 \wedge \phi_2)$ **then** *//process the AND operator*
  RewriteAND* $R^\Delta = \{\phi_1, \phi_2\}$, $Deps = \{R' = \phi\}$
  $\phi_g \leftarrow$ G($\phi_1, D_\downarrow$) $\wedge$ G($\phi_2, D_\downarrow$)
if match $(\phi, \phi_1 \vee \phi_2)$ **then** *//process the OR operator*
  RewriteOr* $R^\Delta = \{DL^+(\phi_1), DL^+(\phi_2), DEF(\phi_1) \vee DEF(\phi_2)\}$, $Deps = \{R' = \phi\}$
  $\phi_g \leftarrow$ G($\phi_1, D_\downarrow$) $\vee$ G($\phi_2, D_\downarrow$)
return ϕ_g

Example 5.5. Continuing from Example 3.9, ϕ contains the following atoms: (1) $h_1.t \geq h_2.t$, (2) $r_1.t = -h_2.t$, (3) $r_2.t > h_1.t$, (4) $\wedge(h_1.t > 0)$, (5) $\exists h_1$, (6) $\exists r_1$, and (7) $\forall r_2$. Given the proof S, atom (2) is active because $\psi_6 \in F_n$; atom (3) is active because $\psi_7 \in F_n$; and atom (4) is active because $h_1^o > t \in F_n$. Atoms (5), (6) and (7) are active because the relational objects h_1^o and r_1^o are in D_n while atoms $h_1^o.ext$ and $r_1^o.ext$ are active. However, atom (1) is not active, and the atom-level proof-specific diagnosis of ϕ is:

$$\phi_{dig} := \exists h_1(\forall h_2(\exists r_1 \cdot r_1.t = -h_2.t)) \wedge (\forall r_2 \cdot r_2.t > h_1.t) \wedge (h_1.t > 0).$$

In practice, given an FOL* formula ϕ and a proof of UNSAT S, we first check the proof using Alg. 1 while computing a trimmed proof S' extracted from *core*. Then, we compute the atom-level proof-specific diagnosis on ϕ and S' to explain a more precise cause of unsatisfiability.

In case FOL* formulas are compiled from a different source formalism (e.g., SLEEC rules [20]), we can compute the proof-based diagnosis at the level of the source formalism from the FOL* diagnosis by tracking the relationship between atoms in the source formalism and FOL* and projecting the FOL* diagnosis onto the source formalism. In Sec. 7, we show this process for SLEEC rules.

6 SUPPORTING PROOF OF UNSAT FOR LEGOS

Here we introduce our extension to the FOL* satisfiability checking algorithm LEGOS [21] to generate the FOL* proof of UNSAT.

Recall from Sec. 2 that an FOL* formula is satisfiable if there is some domain D and a valuation function v such that $(D, v) \models \phi$.

Table 1: Efficiency of LEGOS-PROOF in generating UNSAT proofs of *vacuous conflicts* & effectiveness in trimming them. t-overhead is the time overhead for generating and checking the proof and t-check is the time for checking the generated proof; trimmed/initial is the size of the trimmed and initial proof files measured in Kilobyte (size) and as the percentage of reduction (%). n-steps is the trimmed and the initial number of proof derivation steps, with % as the percentage of reduction after checking the proof.

case studies	t-overhead	t-check	size trimmed/initial (%)	n-steps trimmed/initial (%)
Tabiat	-0.07	0.08	18/230 (92%)	93/963 (90%)
	0.05	0.07	18/133 (86%)	73/398 (82%)
	-0.04	0.1	22/256 (91%)	114/1091 (90%)
	0.03	0.08	18/233 (91%)	93/963 (90%)
	0.01	0.08	18/245 (92%)	93/1028 (91%)
	-0.016	0.08	18/234 (92%)	93/963 (90%)

LEGOS can detect the unsatisfiability of a given formula ϕ if an over-approximation $\phi_g := \mathrm{G}(\phi, D')$ is unsatisfiable for any domain D', where G is defined in Alg. 2. The over-approximation ϕ_g is a QFLIA formula whose satisfiability can be decided by an SMT solver. To extend LEGOS to support the proof of UNSAT, we need to show: (1) the derivation steps to derive ϕ_g, and (2) the unsatisfiability of ϕ_g. The latter is trivial as we can easily apply the rule $\mathbb{T}$-**Derive** on ϕ_g to derive $\bot$ if ϕ_g is UNSAT. To show the derivation of ϕ_g, we extend G (Alg. 2) to capture the encoding steps as FOL* derivations. The extensions are colored in blue.

G (Alg. 2) requires that the input FOL* formula ϕ is in negation normal form (NNF). To satisfy the requirement, we recursively apply **RewriteNeg** (see Sec. 3.1) to convert ϕ into NNF. G then matches the top-level logical operator ($\exists$, $\forall$, $\wedge$, or $\vee$) of ϕ to encode ϕ_g accordingly. To support proof generation, we attach applications of different proof rules (shown in blue) for different cases. For example, when G is given ϕ that matches the expression $\exists o : r \cdot \phi'$, then G first instantiates o with a fresh relational object of class r and then encodes an intermediate formula $\phi_{o'}$ as $o'.ext \wedge \phi' \leftarrow [o \leftarrow o']$. To prove that $\phi_{o'}$ is derived from ϕ, the rule **ExistentialIns*** is applied on lemma ϕ to derive $\phi_{o'}$ and a new relational object o'. In case $\phi_{o'}$ contains other quantifiers, G is recursively called on $\phi_{o'}$, which would invoke more FOL* derivation rules.

7 EVALUATION

In this section, we first evaluate the efficiency of LEGOS-PROOF, the implementation of our approach (see Sec. 5) in generating proofs of unsatisfiability, its effectiveness in trimming the proof, and the utility of its FOL* diagnosis. Specifically, we aim to answer **RQ1:** How efficient is the proof framework for (a) generating, and (b) checking proofs, in terms of the time overhead? We utilize LEGOS-PROOF to generate proofs, verify them, and measure the overhead associated with both the proof generation and the proof verification (employing a backward-checking mode). Then, we aim to answer **RQ2:** How effective is LEGOS-PROOF for trimming the proof? We compare the initial size of the proof file and the number of derivation steps, with the ones trimmed. Finally, we aim to answer **RQ3:** How well does the instantiated diagnosis assist users in debugging identified inconsistencies? Through answering this question, we aim to determine whether the instantiated diagnosis provides meaningful insights for debugging inconsistencies. This demonstrates the utility of the FOL* diagnosis on a use case. LEGOS-PROOF implementation and the evaluation artifacts are available in [2].

Table 2: Efficiency of LEGOS-PROOF in generating UNSAT proofs of *situational conflicts* & effectiveness in trimming them. Significant overhead is in bold.

case studies	t-overhead	t-check	size trimmed/initial (%)	n-steps trimmed/initial (%)
ALMI	0.13	0.09	104/256 (59%)	137/664 (79%)
ASPEN	0.26	0.18	76/392 (81%)	118/1607 (93%)
	0.21	0.16	76/230 (67%)	111/745 (85%)
	-0.05	0.17	76/268 (72%)	111/832 (87%)
DAISY	0.33	0.23	118/648 (82%)	122/1854 (93%)
	0.29	0.21	117/520 (78%)	122/1470 (82%)
SafeSCAD	0.25	0.21	74/234 (68%)	116/648 (82%)
	0.22	0.18	68/240 (72%)	67/779 (91%)
	0.23	0.19	67/226 (70%)	81/721 (89%)
	2.25	2.17	88/244 (64%)	162/843 (81%)
	0.34	0.27	100/358 (72%)	204/1175 (83%)
	0.26	0.24	83/196 (58%)	156/544 (71%)
	0.29	0.24	88/236 (63%)	162/793 (80%)
	0.27	0.24	89/197 (55%)	162/544 (70%)
	0.32	0.29	101/248 (59%)	207/674 (69%)
	0.56	0.24	83/245 (66%)	156/849 (82%)
	0.33	0.28	101/263 (62%)	207/769 (33%)
Tabiat	0.13	0.09	79/203 (61%)	55/555 (90%)
	0.13	0.09	71/232 (69%)	34/586 (94%)
	0.13	0.09	68/229 (70%)	42/632 (93%)
	0.14	0.10	90/199 (55%)	105/542 (81%)
	0.13	0.10	91/200 (54%)	105/542 (81%)
	0.14	0.10	84/199 (58%)	98/542 (82%)
	0.14	0.10	85/244 (65%)	93/657 (86%)
	0.13	0.08	69/257 (73%)	34/699 (95%)
	-0.19	0.14	98/369 (73%)	127/990 (87%)

Table 3: Efficiency of LEGOS-PROOF in generating UNSAT proofs of *redundancy* & effectiveness in trimming them.

case studies	t-overhead	t-check	size trimmed/initial (%)	n-steps trimmed/initial (%)
ASPEN	0.08	0.08	27/151 (82%)	101/751 (87%)
AutoCar	0.21	0.15	29/143 (80%)	97/576 (83%)
	0.21	0.14	29/159 (82%)	97/673 (86%)
Casper	0.17	0.14	23/85 (73%)	93/320 (71%)
CSI-Cobot	0.12	0.12	9/34 (74%)	51/131 (61%)
	0.12	0.11	9/35 (74%)	51/131 (61%)
	0.13	0.07	10/52 (81%)	53/228 (77%)
DAISY	0.15	0.14	7/48 (85%)	30/136 (78%)
SafeSCAD	0.02	0.01	9/39 (77%)	51/155 (67%)
	0.02	0.01	10/40 (75%)	53/159 (67%)
Tabiat	0.07	0.09	19/216 (91%)	93/895 (80%)
	0.09	0.07	9/50 (82%)	44/165 (73%)
	0.13	0.08	15/259 (94%)	73/1089 (93%)
	0.12	0.08	19/231 (92%)	93/945 (80%)
	0.09	0.08	19/233 (92%)	93/947 (80%)

Case studies. We considered the FOL* requirements developed for eight real-world case-studies taken from the RESERVE repository of normative requirements, with FOL* satisfiability used o check their well-formedness [20, 22]: (1) ALMI [25] – a system assisting elderly or disabled users in a monitoring/advisory role and with everyday tasks; (2) ASPEN [18] – an autonomous agent dedicated

Table 4: Efficiency of LEGOS-PROOF in generating UNSAT proofs of *restrictiveness* & effectiveness in trimming them.

case studies	t-overhead	t-check	size trimmed/initial (%)	n-steps trimmed/initial (%)
SafeSCAD	0.04	0.02	10/43 (77%)	53/162 (67%)
	0.03	0.01	10/42 (77%)	60/169 (67%)
Tabiat	0.09	0.08	21/141 (85%)	85/455 (81%)
	0.01	0.08	18/236 (92%)	93/1022 (91%)
	0.11	0.08	18/237 (92%)	93/1022 (91%)

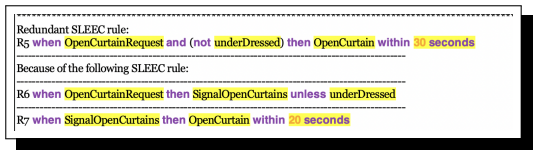

Figure 4: Redundancy diagnosis example.

to forest protection, providing both diagnosis and treatment of various tree diseases; (3) AutoCar [4] – a system that implements emergency-vehicle priority awareness for autonomous vehicles; (4) CSI-Cobot [39] – a system ensuring the safe integration of industrial collaborative robot manipulators; (5) DAISY [14] – a sociotechnical AI-supported system that directs patients through an A&E triage pathway (our running example); (6) SafeSCAD [13] – a driver attentiveness management system to support safe shared control of autonomous vehicles; (7) Tabiat [32] – a smartphone application (and sensors) that records symptoms and keeps doctors updated on patients' chronic obstructive pulmonary disease conditions; and (8) Casper [35] – a socially assistive robot designed to help people living with dementia by supporting activities of daily living at home and improving their quality of life.

Well-formedness properties. FOL* has been used as an underlying formal language for SLEEC, a rule-based language for expressing normative requirements [24]. SLEEC rules define metric temporal constraints over actions, referred to as *events*, and (unbounded) environmental variables, referred to as *measures*. Here is an example SLEEC rule: **when** MeetingUser **and not** patientStressed **then** ExamineState. Here 'MeetingUSer' and 'ExamineState' are events, and patientStressed is a measure. Given the declarative nature of requirements specification with SLEEC rules, they are prone to having inconsistencies and other well-formedness properties. The following well-formedness properties of SLEEC rules can be verified via FOL* unsatisfiability checking [20]: *vacuous conflict*, *situational conflict*, *redundancy*, and *over-restrictiveness*. (i) a rule is *vacuously conflicting in a rule set R* if triggering this rule always leads to a conflict with some other rules in *R*; (ii) a rule is *situationally conflicting in a rule set R* if triggering this rule under a certain situation, which defines a history of events and measures, always leads to a conflict with some other rules in the future; (iii) a rule is *redundant in a rule set R* if this rule is logically implied by other rules in *R*; and (iv) a rule set *R* is *overly restrictive subject to a given purpose* if none of the functional goals represented by the purpose is realizable while respecting *R*. Refer to [20] for formal definitions.

RQ1. To answer RQ1, for each case study and well-formedness property, we measure the overhead (time) of generating and checking the UNSAT proof for the existence of the well-formedness issues. The results for vacuous conflicts, situational conflicts, redundancy, and restrictiveness are shown in the second column of Tbl. 1, Tbl. 2, Tbl. 3, and Tbl. 4, respectively. We found 52 unsatisfiable instances across the eight case studies. The average overhead for proof generation was 0.03 seconds (with the geometric mean of 34%). On the other hand, the geometric mean for proof checking (compared to the solving time without proof generation) was 153%. Overall, the overhead for proof generation was sufficiently small, enabling the online (i.e., during SAT solving) usage of this technique. Therefore, the results answer RQ1.

RQ2. To answer RQ2, for each case study and well-formedness property, we measure the initial proof size with the number of proof derivation steps, as well as the proof size and the number of steps remaining once trimmed (after checking the proof). The results for vacuous conflicts, situational conflicts, redundancy, and restrictiveness are shown in the third and fourth columns of Tbl. 1, Tbl. 2, Tbl. 3, and Tbl. 4, respectively. The size of the trimmed proof after checking the UNSAT proof generated by LEGOS-PROOF is reduced by between 54% (UNSAT proof for situational conflicts) and 94% (UNSAT proof for redundancy). The number of steps remaining after checking the proof is reduced by between 33% (UNSAT proof for situational conflicts) and 93% (UNSAT proof forredundancy). Overall, LEGOS-PROOF is considerably effective for trimming the UNSAT proof, achieving at least a 50% reduction in size and a one-third reduction in the number of steps, thereby answering RQ2.

RQ3. To answer RQ3, we conducted an experiment with five non-technical stakeholders (N-TSs) to determine whether the diagnoses obtained for eight RESERVE real-world case-studies aid in understanding and resolving the specification inconsistencies. The N-TSs included an ethicist, a lawyer, a philosopher, a psychologist, and a medical doctor. Different diagnoses are generated for different well-formedness properties. For instance, for the assistive robot ALMI, an example of diagnosis generated for redundancy is shown in Fig. 4. Therefore, for each property type, we measured how different information provided was used by the N-TSs to resolve the well-formedness problems: (a) for vacuous conflict, redundancy, and restrictiveness: the rules, events, and measures identified; and (b) for situational conflict: the rules and the events, and measures in both the rules and the situation. The results are shown in Tbl. 5.

The rule sets for the different case studies ranged between 15 and 19 rules. For each well-formedness issue, the diagnoses identified a small core of 2-4 rules responsible for the violation which allowed the stakeholders to focus on these rules and easily resolve the conflicts. Furthermore, the highlighting of the clauses in the diagnoses turned out to be very useful: out of the 14 instances, all highlighted clauses were used in the resolution in 8 cases, and at least one of them in 2 cases. We observed that the highlighted clauses helped the stakeholders understand exactly what caused the issues, e.g., unnecessary redundant rules, like R5 in the diagnosis shown in Fig. 4, have all their clauses highlighted and are present (and highlighted) in other rules within the diagnosis. At least one of the events/measures present in both the situation and the rules was directly used in the resolution (i.e., present in the added, removed, or updated rules). In conclusion, for each type of well-formedness checks, the stakeholders were satisfied with the information presented to them in the diagnosis, agreeing that

Table 5: Usability of the well-formedness diagnosis computed using LEGOS-PROOF. For each generated diagnosis of a property violation, the table records the number of rules present in the diagnosis (nRu) and the total number of rules in the case study (nR), the number of clauses highlighted in the rules (nC) and the number of clauses used (nCu) to resolve the issue, the number of events/measures present in the diagnosis (nE/nM) and the number of these events/measures used to resolve the issue (nEu/nMu), and finally, the number of events present (nSe) and used (nSEu) in the situation to resolve the situational conflicts.

	well-formedness issues										
case studies	conflict		s-conflicts				redundancy		restrictiveness		
	nRu (Ru)	nCu (nC)	nRu (nR)	nCu (nC)	nSEu (nSE)	nSMu (nSM)	nRu (nR)	nCu (nC)	nC (nCu)	nEu (nE)	nMu (nM)
ALMI	NA		2 (39)	4 (7)	2 (3)	1 (2)	NA			NA	
ASPEN	NA		2 (15)	6 (6)	2 (3)	2 (3)	3 (15)	7 (7)		NA	
Casper	NA			NA			2 (26)	8 (8)		NA	
DAISY	NA		2 (26)	6 (6)	2 (5)	5 (5)	1 (1)	2 (2)		NA	
CSI-Cobot	NA			NA			2 (20)	4 (4)		NA	
SafeSCAD	NA		2 (28)	4 (6)	1 (5)	2 (3)	2 (2)	4 (4)	4 (4)	2 (2)	1 (1)
Tabiat	4 (28)	1 (3)	2 (28)	5 (6)	2 (4)	3 (3)	4 (28)	1 (4)	2 (3)	2 (4)	0 (0)

it provides meaningful insights for helping debug rule inconsistencies, answering RQ3. The stakeholders suggested going further and recommending resolution patches. Thus, using our UNSAT proof-based diagnosis to compute suggestable resolution patches is a possible future research direction.

Summary. Our experiments demonstrated that LEGOS-PROOF is efficient in terms of additional time spent for generating and checking proofs, potentially allowing us to use it in an online manner, during satisfiability checking. LEGOS-PROOF was also shown to be effective at trimming the proof size (by at least 50%) and the number of proof steps (by at least a third) while remaining highly useful. Indeed, in our experiments these trimmed proofs were used to instantiate diagnoses, which were shown to stakeholders to resolve identified issues. Such diagnoses provided them with meaningful insights for debugging inconsistencies between their rules. This demonstrates practical utility of the FOL* diagnosis.

Threats to validity. (1) While we only considered eight case-studies, we mitigated this threat by choosing them from different domains. (2) The effectiveness of the generic proof objects derived from FOL* depends on the application. In this study, we only evaluated one application, well-formedness diagnosis for normative requirements. We leave evaluating the effectiveness across a wider range of applications as future work.

8 RELATED WORK

Certifying unsatisfiability. Certifying the UNSAT result provided by constraint solvers has been a long standing challenge. UNSAT proofs have been extensively studied in the context of propositional logic based on the of notion of clause redundancy [12, 16, 17, 27, 38, 41] (i.e., adding new asserting clauses does not affect the input's satisfiability). Our proof support for FOL* is also based on clause redundancy, but with proof rules designed for FOL*-specific theory reasoning. Similarly, SMT solvers need additional mechanisms to handle theory reasoning [7]. For example, Z3 [37] outputs natural-deduction-style proofs [36], which can be reconstructed inside the interactive theorem prover Isabelle/HOL [9, 10]. Similarly, veriT [11] produces resolution proof traces with theory lemmas, and supports proof reconstruction in both Coq [3] and Isabelle [5, 6, 23]. As a more general approach, CVC5 [8] produces proofs in the LFSC format [40], which is a meta-logic that allows describing theory-specific proof rules for different SMT theories. Similarly to the proof support in SMT solvers, we provided theory-specific proof rules to cover the reasoning steps for soundly deriving the FOL* over-approximation. In contrast, the existing work primarily focused on the quantifier-free fragment of first-order logic and is complementary to our work, as it can be used to close the gap to ensure the correctness of external reasoning steps in our proof framework.

Diagnosing unsatisfiability. Identifying the causes of unsatisfiability has been studied in the context of computing UNSAT Cores. DRAT-trim [26] enabled UNSAT core production via backward checking of DRAT proof of unsatisfiability for input constraints in propositional logic. Maric et al. [33] proposed a method to generate UNSAT Cores while running Simplex for solving constraints in linear real arithmetic. Cimatti et al. [15] proposed a *lemma-lifting approach* to compute UNSAT core in SMT solvers by lazily lifting theory information into boolean abstractions and then refining boolean UNSAT cores backs to the original constraints. Our approach to diagnosis computation can be viewed as an extension of the lemma-lifting approach. Instead of lifting theory information to the boolean level, we lazily lift theory information into the level of a decidable theory $\mathbb{T}$ where UNSAT core production is supported. We then refine the abstracted UNSAT core in $\mathbb{T}$ back to the original constraints in FOL*. Compared to DRAT-Trim, our approach for proof trimming follows a similar backward checking methodology but at a more fine-grained level due to the use of lazy $\mathbb{T}$ abstractions.

9 CONCLUSION

In this paper, we introduced the first method to support proof of unsatisfiability for FOL* by capturing UNSAT deductions through a sequence of verifiable derivation steps. Additionally, we proposed an efficient technique to validate the correctness of these proofs while eliminating unnecessary derivations. From the refined proof, we can derive a concise proof-based diagnosis to explain the cause of unsatisfiability. Our experiments demonstrated the efficiency and effectiveness of our proof support and proof-based diagnosis in software requirements validation tasks. Looking ahead, we aim to enhance the trustworthiness of FOL* proofs by reconstructing them in proof assistants such as Coq. Furthermore, we plan to extend our support to proofs of UNSAT for aggregation functions (e.g., *Sum*, *Count*, *Average*) as additional constructs in FOL*.

REFERENCES

[1] Abad, P., Aguirre, N., Bengolea, V.S., Ciolek, D.A., Frias, M.F., Galeotti, J.P., Maibaum, T., Moscato, M.M., Rosner, N., Vissani, I.: Improving Test Generation under Rich Contracts by Tight Bounds and Incremental SAT Solving. In: Proceedings of the sixth International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, Luxembourg. pp. 21–30. IEEE Computer Society (2013). https://doi.org/10.1109/ICST.2013.46

[2] Anonymous: Supplementary material for ase submission: Diagnosis via proofs of unsatisfiability for first-order logic with relational objects (2024), https://anonymous.4open.science/r/LEGOS-Proof-Artifact-EE15/README.md

[3] Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. In: Jouannaud, J., Shao, Z. (eds.) Certified Programs and Proofs - First International Conference, CPP 2011, Kenting, Taiwan, December 7-9, 2011. Proceedings. Lecture Notes in Computer Science, vol. 7086, pp. 135–150. Springer (2011). https://doi.org/10.1007/978-3-642-25379-9_12, https://doi.org/10.1007/978-3-642-25379-9_12

[4] Bahadır, B.N., Kasap, Z.: AutoCar Project, https://acp317315180.wordpress.com/

[5] Barbosa, H., Blanchette, J., Fleury, M., Fontaine, P., Schurr, H.J.: Better SMT proofs for easier reconstruction. In: AITP 2019-4th Conference on Artificial Intelligence and Theorem Proving (2019)

[6] Barbosa, H., Blanchette, J.C., Fontaine, P.: Scalable Fine-Grained Proofs for Formula Processing. In: de Moura, L. (ed.) Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10395, pp. 398–412. Springer (2017). https://doi.org/10.1007/978-3-319-63046-5_25, https://doi.org/10.1007/978-3-319-63046-5_25

[7] Barrett, C., De Moura, L., Fontaine, P.: Proofs in satisfiability modulo theories. All about proofs, Proofs for all **55**(1), 23–44 (2015)

[8] Barrett, C.W., Conway, C.L., Deters, M., Hadarean, L., Jovanovic, D., King, T., Reynolds, A., Tinelli, C.: CVC4. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6806, pp. 171–177. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_14, https://doi.org/10.1007/978-3-642-22110-1_14

[9] Böhme, S.: Proof reconstruction for Z3 in Isabelle/HOL. In: 7th International Workshop on Satisfiability Modulo Theories (SMT'09) (2009)

[10] Böhme, S., Weber, T.: Fast LCF-Style Proof Reconstruction for Z3. In: Kaufmann, M., Paulson, L.C. (eds.) Interactive Theorem Proving, First International Conference, ITP 2010, Edinburgh, UK, July 11-14, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6172, pp. 179–194. Springer (2010). https://doi.org/10.1007/978-3-642-14052-5_14, https://doi.org/10.1007/978-3-642-14052-5_14

[11] Bouton, T., Oliveira, D.C.B.D., Déharbe, D., Fontaine, P.: veriT: An Open, Trustable and Efficient SMT-Solver. In: Schmidt, R.A. (ed.) Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5663, pp. 151–156. Springer (2009). https://doi.org/10.1007/978-3-642-02959-2_12, https://doi.org/10.1007/978-3-642-02959-2_12

[12] Buss, S., Thapen, N.: DRAT proofs, propagation redundancy, and extended resolution. In: Janota, M., Lynce, I. (eds.) Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11628, pp. 71–89. Springer (2019). https://doi.org/10.1007/978-3-030-24258-9_5, https://doi.org/10.1007/978-3-030-24258-9_5

[13] Calinescu, R., Alasmari, N., Gleirscher, M.: Maintaining Driver Attentiveness in Shared-Control Autonomous Driving. In: 16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). pp. 90–96. IEEE (2021)

[14] Calinescu, R., Ashaolu, O.: Diagnostic AI System for Robot-Assisted A&E Triage (DAISY) website., https://twitter.com/NorwichChloe/status/1679112358843613184?t=ALk7s8wcyHztZyyHJoB5pg&s=19, https://tas.ac.uk/research-projects-2022-23/daisy/

[15] Cimatti, A., Griggio, A., Sebastiani, R.: Computing small unsatisfiable cores in satisfiability modulo theories. J. Artif. Intell. Res. **40**, 701–728 (2011). https://doi.org/10.1613/JAIR.3196, https://doi.org/10.1613/jair.3196

[16] Cruz-Filipe, L., Heule, M.J.H., Jr., W.A.H., Kaufmann, M., Schneider-Kamp, P.: Efficient Certified RAT Verification. In: de Moura, L. (ed.) Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10395, pp. 220–236. Springer (2017). https://doi.org/10.1007/978-3-319-63046-5_14, https://doi.org/10.1007/978-3-319-63046-5_14

[17] Cruz-Filipe, L., Marques-Silva, J., Schneider-Kamp, P.: Efficient Certified Resolution Proof Checking. In: Legay, A., Margaria, T. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 118–135. Springer Berlin Heidelberg, Berlin, Heidelberg (2017)

[18] Dandy, N., Calinescu, R.: Autonomous Systems for Forest ProtEctioN (ASPEN) website., https://tas.ac.uk/research-projects-2023-24/autonomous-systems-for-forest-protection/

[19] Feng, N., Marsso, L., Getir-Yaman, S., Beverley, T., Calinescu, R., Cavalcanti, A., Chechik, M.: Towards a Formal Framework for Normative Requirements Elicitation. In: Proceedings of the 38th International Conference on Automated Software Engineering, (ASE'2023), Kirchberg, Luxembourg. IEEE (2023)

[20] Feng, N., Marsso, L., Getir-Yaman, S., Townsend, B., Baatartogtokh, Y., Ayad, R., de Mello, V.O., Standen, I., Stefanakos, I., Imrie, C., Rodrigues, G., Cavalcanti, A., Calinescu, R., Chechik, M.: Analyzing and Debugging Normative Requirements via Satisfiability Checking. In: Proceedings of the 46th International Conference on Software Engineering, (ICSE 2024), Lisbon, Portugal. ACM (2024)

[21] Feng, N., Marsso, L., Sabetzadeh, M., Chechik, M.: Early verification of legal compliance via bounded satisfiability checking. In: Proceedings of the 34th international conference on Computer Aided Verification (CAV'23), Paris, France. Lecture Notes in Computer Science, Springer (2023)

[22] Feng, N., Marsso, L., Yaman, S.G., Standen, I., Baatartogtokh, Y., Ayad, R., de Mello, V.O., Townsend, B., Bartels, H., Cavalcanti, A., Calinescu, R., Chechik, M.: Normative Requirements Operationalization with Large Language Models. In: Proceedings of the 32nd IEEE International Conference on Requirements Engineering, RE'2024, Reyjavik Iceland. ACM (2024)

[23] Fleury, M., Schurr, H.: Reconstructing veriT Proofs in Isabelle/HOL. In: Reis, G., Barbosa, H. (eds.) Proceedings Sixth Workshop on Proof eXchange for Theorem Proving, PxTP 2019, Natal, Brazil, August 26, 2019. EPTCS, vol. 301, pp. 36–50 (2019). https://doi.org/10.4204/EPTCS.301.6, https://doi.org/10.4204/EPTCS.301.6

[24] Getir-Yaman, S., Burholt, C., Jones, M., Calinescu, R., Cavalcanti, A.: Specification and Validation of Normative Rules for Autonomous Agents. In: Proceedings of the 26th International Conference on Fundamental Approaches to Software Engineering (FASE'2023), Paris, France. Lecture Notes in Computer Science, Springer (2023)

[25] Hamilton, J., Stefanakos, I., Calinescu, R., Cámara, J.: Towards adaptive planning of assistive-care robot tasks. In: Proceedings of th Fourth International Workshop on Formal Methods for Autonomous Systems and Fourth International Workshop on Automated and verifiable Software sYstem DEvelopment, (FMAS/ASYDE@SEFM'2022), Berlin, Germany. EPTCS, vol. 371, pp. 175–183 (2022). https://doi.org/10.4204/EPTCS.371.12, https://www.youtube.com/watch?v=VhfQmJe4IPc

[26] Heule, M., Jr., W.A.H., Wetzler, N.: Trimming while checking clausal proofs. In: Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013. pp. 181–188. IEEE (2013), https://ieeexplore.ieee.org/document/6679408/

[27] Heule, M.J.H., Kiesl, B., Biere, A.: Strong extension-free proof systems. J. Autom. Reason. **64**(3), 533–554 (2020). https://doi.org/10.1007/S10817-019-09516-0, https://doi.org/10.1007/s10817-019-09516-0

[28] King, J.C.: Symbolic execution and program testing. Commun. ACM **19**(7), 385–394 (1976). https://doi.org/10.1145/360248.360252

[29] Krishna, A., Poizat, P., Salaün, G.: VBPMN: automated verification of BPMN processes (tool paper). In: Polikarpova, N., Schneider, S.A. (eds.) Integrated Formal Methods - 13th International Conference, IFM 2017, Turin, Italy, September 20-22, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10510, pp. 323–331. Springer (2017). https://doi.org/10.1007/978-3-319-66845-1_21, https://doi.org/10.1007/978-3-319-66845-1_21

[30] Krishna, A., Poizat, P., Salaün, G.: Checking business process evolution. Science of Computer Programming **170**, 1–26 (2019)

[31] Liang, J.H.J., Ganesh, V., Czarnecki, K., Raman, V.: SAT-based analysis of large real-world feature models is easy. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA. pp. 91–100. ACM (2015). https://doi.org/10.1145/2791060.2791070

[32] Liaqat, D.: The Tabiat website., https://www.tabiat.care/

[33] Maric, F., Spasic, M., Thiemann, R.: An incremental simplex algorithm with unsatisfiable core generation. Arch. Formal Proofs **2018** (2018), https://www.isa-afp.org/entries/Simplex.html

[34] de Matos Pedro, A., Silva, T., Sequeira, T.F., Lourenço, J., Seco, J.C., Ferreira, C.: Monitoring of spatio-temporal properties with nonlinear SAT solvers. Int. J. Softw. Tools Technol. Transf. **26**(2), 169–188 (2024). https://doi.org/10.1007/S10009-024-00740-7

[35] Moro, C., Nejat, G., Mihailidis, A.: Learning and personalizing socially assistive robot behaviors to aid with activities of daily living. ACM Transactions on Human-Robot Interaction (THRI) **7**(2), 1–25 (2018)

[36] de Moura, L.M., Bjørner, N.: Proofs and Refutations, and Z3. In: Rudnicki, P., Sutcliffe, G., Konev, B., Schmidt, R.A., Schulz, S. (eds.) Proceedings of the LPAR 2008 Workshops, Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics, Doha, Qatar, November 22, 2008. CEUR Workshop Proceedings, vol. 418. CEUR-WS.org (2008), http://ceur-ws.org/Vol-418/paper10.pdf

[37] de Moura, L.M., Bjørner, N.: Z3: An Efficient SMT Solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint

European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24, https://doi.org/10.1007/978-3-540-78800-3_24

[38] Rebola-Pardo, A.: Even shorter proofs without new variables. In: Mahajan, M., Slivovsky, F. (eds.) 26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy. LIPIcs, vol. 271, pp. 22:1–22:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.SAT.2023.22, https://doi.org/10.4230/LIPIcs.SAT.2023.22

[39] Stefanakos, I., Calinescu, R., Douthwaite, J.A., Aitken, J.M., Law, J.: Safety controller synthesis for a mobile manufacturing cobot. In: Proceedings of the 20th International Conference on Software Engineering and Formal Methods (SEFM'2022), Berlin, Germany. Lecture Notes in Computer Science, vol. 13550, pp. 271–287. Springer (2022). https://doi.org/10.1007/978-3-031-17108-6_17

[40] Stump, A., Oe, D., Reynolds, A., Hadarean, L., Tinelli, C.: SMT proof checking using a logical framework. Formal Methods Syst. Des. **42**(1), 91–118 (2013). https://doi.org/10.1007/s10703-012-0163-3, https://doi.org/10.1007/s10703-012-0163-3

[41] Wetzler, N., Heule, M.J.H., Hunt, W.A.: Mechanical Verification of SAT Refutations with Extended Resolution. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) Interactive Theorem Proving. pp. 229–244. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)

[42] Winterer, D., Zhang, C., Su, Z.: Validating SMT solvers via semantic fusion. In: Donaldson, A.F., Torlak, E. (eds.) Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020. pp. 718–730. ACM (2020). https://doi.org/10.1145/3385412.3385985, https://doi.org/10.1145/3385412.3385985

A FOL* DERIVATION RULES

Define. If ϕ is an FOL* formula and $DEF(\phi)$ is undefined, then applying the **Define** rule assigns $DEF(\phi)$ to a fresh variable and adds the definition clauses $DL^+(\phi)$ and $DL^-(\phi)$ to R^+. The derivation rule **Define** allows the introduction of new FOL* formulas in the proof of UNSAT. For example, **Define** can introduce ϕ as a hypothesis, and by deriving $DEF(\phi)$ as a lemma (via $DL^-(\phi)$), we can then use ϕ as a lemma (via $DL^+(\phi)$) in the rest of the proof.

Substitute. Suppose an FOL* formula ϕ contains a sub-formula ψ [1], where all free variables in ψ are also free in ϕ (i.e., $FV(\phi) \supseteq FV(\psi)$). If both $DEF(\phi)$ and $DEF(\psi)$ are defined, then the following lemmas can be added to R^+:

$$\phi^+_{subs} : DEF(\phi) \Rightarrow \phi[\psi \leftarrow DEF(\psi)]$$

$$\phi^-_{subs} : \overline{DEF(\phi)} \Rightarrow \neg\phi[\psi \leftarrow DEF(\psi)],$$

where $\phi[\psi \leftarrow DEF(\psi)]$ is the result of substituting ψ with $DEF(\psi)$ in ϕ. Note that the condition $FV(\phi) \supseteq FV(\psi)$ prohibits unsound derivation by matching universally quantified variables in ϕ to free variables in ψ, which are assumed to be existentially quantified. Here, we explain the dependencies of the derived lemmas: (1) ϕ^+_{sub} depends on the positive definition of ϕ ($DL^+(\phi)$) and the negative definition of ψ ($DL^-(\psi)$); and (2) ϕ^-_{sub} depends on the negative definition of ϕ ($DL^-(\phi)$) and the positive definition of ψ ($DL^+(\psi)$).

Definition A.1. Let a derivation step $d_i = (rule_i, Dep_i, F^\Delta_i, R^\Delta_i, D^\Delta_i)$ be given where $Dep_i = (R'_i, F'_i, D'_i)$. Suppose ϕ and ψ are two FOL* formula. The step d_i is a valid application of **substitute** if:

$$PC_{\textbf{substitute}}(d_i) := F^i = D_i = \emptyset \wedge DPos \vee DNeg \text{ where}$$
$$DPos := ((R^+_i = \{DEF(\phi) \Rightarrow \phi[\psi \leftarrow DEF(\psi)]\} \wedge \{DL^+(\phi), DL^-(\psi))\} \subseteq D'_i)$$
$$DNeg := (R^+_i = \{\overline{DEF(\phi)} \Rightarrow \neg\phi[\psi \leftarrow DEF(\psi)]\} \wedge \{DL^-(\phi), DL^+(\psi))\} \subseteq D'_i))$$

In Def. A.1, according to the enabling condition $PC_{\textbf{substitute}}$, an application of **Substitute** derives either ϕ^+_{subs} (following $DPos$) or ϕ^-_{subs} (following $DNeg$) one at a time. However, since deriving ϕ^+_{subs} does not blocks the derivation of ϕ^-_{subs}, we assume both lemmas are derived by a single application if not otherwise stated.

ApplyLemma. If $\phi \in R^+$ is a lemma and the definition variable $DEF(\phi)$ is defined, then applying rule **ApplyLemma** adds lemma $\phi_{\textbf{apply}} := DEF(\phi)$ to R^+. The formal definition of $PC_{\textbf{ApplyLemma}}$ is provided in the appendices. Here, we explain the dependencies of the derived lemma: (1) $\phi_{\textbf{applyLemma}}$ depends on lemma ϕ. Intuitively, lemmas in R^+ are asserted to be true, and hence their definition variable should also evaluate to true.

Definition A.2. Let a derivation step $d_i = (rule_i, Dep_i, F^\Delta_i, R^\Delta_i, D^\Delta_i)$ be given where $Dep_i = (R'_i, F'_i, D'_i)$. Suppose ϕ is FOL* formula. The step d_i is a valid application of **ApplyLemma** if:

$$PC_{\textbf{ApplyLemma}} := R^\Delta_i = \{DEF(\phi)\} \wedge F^\Delta_i = D^\Delta_i = \emptyset \wedge \phi \in R'_i \wedge \phi \in DEF$$

RewriteOR. If an FOL* formula ϕ matches $A \vee B$, where A and B are FOL* formulas, and $DEF(\phi)$, $DEF(A)$, and $DEF(B)$ are defined, then applying rule **RewriteOR** to ϕ adds the following lemmas to R^+:

$$\phi_{or+} := DEF(\phi) \Rightarrow DEF(A) \vee DEF(B),$$

$$\phi^l_{or-} := \overline{DEF(\phi)} \Rightarrow \overline{DEF(A)} \text{ and } \phi^r_{or-} := \overline{DEF(\phi)} \Rightarrow \overline{DEF(B)}$$

The dependencies of the derivations are: (1) the lemma $\phi_{\text{or+}}$ depends on the positive definition clauses $DL^+(\phi)$ and the negative definition clauses $DL^-(A)$ and $DL^-(B)$; (2) the lemma $\phi^l_{\text{or}-}$ (and $\phi^r_{\text{or}-}$) depends on the negative definition lemma $DL^-(\phi)$ and the positive definition lemmas $DL^+(A)$ and $DL^+(B)$.

RewriteAND. If an FOL* formula ϕ matches $A \wedge B$, where A and B are FOL* formulas, and $DEF(\phi)$, $DEF(A)$, and $DEF(B)$ have been assigned. Then applying rule **RewriteAND** on ϕ adds the following lemmas to R^+:

$$\phi^l_{and+} := DEF(\phi) \Rightarrow DEF(A) \text{ and } \phi^r_{and+} := DEF(\phi) \Rightarrow DEF(B)$$

$$\phi_{and-} := \overline{DEF(\phi)} \Rightarrow \overline{DEF(A)} \vee \overline{DEF(B)}.$$

The dependencies of the derivations are: (1) the lemma $\phi^l_{\text{and+}}$ depends on $DL^+(\phi)$ and $DL^-(A)$; (2) the lemma $\phi^r_{\text{and+}}$ depends on $DL^+(\phi)$ and $DL^-(B)$; and (3) the lemma $\phi_{\text{and}-}$ depends on $DL^-(\phi)$, $DL^+(A)$, and $DL^+(B)$.

ExistentialInst. Suppose an FOL* formula ϕ matches the pattern $\exists o : r \cdot p(o)$, where p is a FOL* predicate over the free relational object o and $DEF(\phi)$ is assigned. Then applying rule **ExistentialInst** on ϕ adds:

(1) a fresh relational object o' ($o' \notin D$) of class r to domain D,
(2) a lemma $\phi^+_{Eo'} := DEF(\phi) \Rightarrow (o'.ext \wedge p(o'))$ to R^+, and
(3) a lemma $\phi^-_{Eo'} := \overline{DEF(\phi)} \Rightarrow \neg(o'.ext \wedge p(o'))$ to R^+

The dependencies of the derivations are: (1) the relational object o' and the lemma ϕ'_{Eo} depend on the positive definition of ϕ ($DL^+(\phi)$); (2) the lemma $\phi^-_{Eo'}$ depends on the negative definition of ϕ ($DL^-(\phi)$).

UniversalInst. Suppose an FOL* formula ϕ matches the pattern $\forall o : r \cdot p(o)$, where p is a FOL* predicate over the free relational object o and $DEF(\phi)$ is assigned. If there exists a relational object o' of class r in D, then applying rule **UniversalInst** on ϕ and o adds the following lemma to R^+: $\phi_{Uo'} := DEF(\phi) \Rightarrow (\neg o'.ext \vee p(o'))$.

The dependencies of the derivations are: the lemma $\phi_{Uo'}$ depends both on the positive definition of ϕ ($DL^+(\phi)$) and on the relational object o'

RewriteNeg. Suppose an FOL* formula matches the pattern $\neg\psi$, where ψ is also an FOL* formula. If $DEF(\phi)$ is defined, then applying **RewriteNeg** on ϕ adds the following lemmas to R^+: (1) $\phi^+_{neg} := DEF(\psi) \Rightarrow NEG(\phi)$ and (2) $\phi^-_{neg} := \overline{DEF(\phi)} \Rightarrow \psi$, where the function NEG pushes the top-level negation into ψ and is defined in a standard way as shown in Fig. 2.

$$NEG(\phi) := \begin{cases} \exists o : r \cdot \neg\psi & \text{if } \phi \text{ matches } \neg\forall o : r \cdot \psi \\ \forall o : r \cdot \neg\psi & \text{if } \phi \text{ matches } \neg\exists o : r \cdot \psi \\ \neg\psi_1 \vee \neg\psi_2 & \text{if } \phi \text{ matches } \neg(\psi_1 \wedge \psi_b) \\ \neg\psi_1 \wedge \neg\psi_2 & \text{if } \phi \text{ matches } \neg(\psi_1 \vee \psi_b) \\ \phi & \text{otherwise} \end{cases}$$

Note that applying **RewriteNeg** recursively converts an FOL* formula into its negation normal form (NNF) where negation symbols

[1] We assume that bounded variables in ψ can be renamed without causing name collisions.

appear in front of atoms (e.g., terms or literals). We can also express **RewriteAnd** with **RewriteOr** and **RewriteNeg**.

Unit. If there is a lemma $\phi := l \Rightarrow \psi \in R^+$ and a unit literal lemma $l \in R^+$, then applying the **unit** rule adds ψ as a lemma to R^+. The dependencies of the derivations are: the lemma ψ depends on both the unit lemma l and ϕ.

UNSAT. If $\perp$ has been derived as a fact or lemma in the current state, then the **UNSAT** rule can be applied to signal the end of the proof.

B EXPANSION OF DERIVATION MACROS

In this section, we show the expansion for the macro **RewriteAND***.

RewriteAND*. If a formula $\phi \in R^+$ matches $\psi_1 \wedge \psi_2 \ldots \wedge \psi_n$, then $\psi_1, \psi_2 \ldots \psi_n$ can be directly added to R^+.

The derivation shortcut can be expanded to the following sequence of the derivation steps:

(1) **Define** on ϕ and $\psi_1 \ldots \psi_n$.
(2) **Substitute** $\psi_1 \ldots \psi_n$ in ϕ to derive a new lemma ϕ', where ψ_i is replaced by a literal $DEF(\psi_i)$.
(3) **ApplyLemma** on ϕ to derive a unit lemma $DEF(\phi)$.
(4) **Unit**-propagate $DEF(\phi)$ on ϕ' to derive $\phi'' := DEF(\psi_1) \wedge \ldots DEF(\psi_n)$.
(5) **RewriteAND** on ϕ'' to derive unit lemmas $DEF(\psi_1) \ldots DEF(\psi_n)$, and
(6) **Unit**-propagate $DEF(\psi_1) \ldots DEF(\psi_n)$ on the positive definition lemma $DL^+(\psi_1) \ldots DL^+(\psi_n)$ to derive $\psi_1 \ldots \psi_n$, respectively.