Analyzing and Debugging Normative Requirements via Satisfiability Checking

Nick Feng
fengnick@cs.toronto.edu
University of Toronto
Toronto, Canada

Lina Marsso
lina.marsso@utoronto.ca
University of Toronto
Toronto, Canada

Sinem Getir Yaman
sinem.getir.yaman@york.ac.uk
University of York
York, UK

Beverley Townsend
bev.townsend@york.ac.uk
University of York
York, UK

Yesugen Baatartogtokh
ybaatartogtokh@smith.edu
Smith College
Northampton, USA

Reem Ayad
reem.ayad@mail.utoronto.ca
University of Toronto
Toronto, Canada

Victoria Oldemburgo de Mello
victoria.mello@mail.utoronto.ca
University of Toronto
Toronto, Canada

Isobel Standen
isobel.standen@york.ac.uk
University of York
York, UK

Ioannis Stefanakos
ioannis.stefanakos@york.ac.uk
University of York
York, UK

Calum Imrie
calum.imrie@york.ac.uk
University of York
York, UK

Genaina Rodrigues
genaina@unb.br
University of Brasilia
Brasilia, Brazil

Ana Cavalcanti
ana.cavalcanti@york.ac.uk
University of York
York, UK

Radu Calinescu
radu.calinescu@york.ac.uk
University of York
York, UK

Marsha Chechik
chechik@cs.toronto.edu
University of Toronto
Toronto, Canada

ABSTRACT

As software systems increasingly interact with humans in application domains such as transportation and healthcare, they raise concerns related to the social, legal, ethical, empathetic, and cultural (SLEEC) norms and values of their stakeholders. *Normative non-functional requirements* (N-NFRs) are used to capture these concerns by setting SLEEC-relevant boundaries for system behavior. Since N-NFRs need to be specified by multiple stakeholders with widely different, non-technical expertise (ethicists, lawyers, regulators, end users, etc.), N-NFR elicitation is very challenging. To address this challenge, we introduce N-Check, a novel tool-supported formal approach to N-NFR analysis and debugging. N-Check employs satisfiability checking to identify a broad spectrum of N-NFR well-formedness issues, such as conflicts, redundancy, restrictiveness, and insufficiency, yielding diagnostics which pinpoint their causes in a user-friendly way that enables non-technical stakeholders to understand and fix them. We show the effectiveness and usability of our approach through nine case studies in which teams of ethicists, lawyers, philosophers, psychologists, safety analysts, and engineers used N-Check to analyse and debug 233 N-NFRs, comprising 62 issues for the software underpinning the operation of systems ranging from assistive-care robots and tree-disease detection drones to manufacturing collaborative robots.

ACM Reference Format:
Nick Feng, Lina Marsso, Sinem Getir Yaman, Beverley Townsend, Yesugen Baatartogtokh, Reem Ayad, Victoria Oldemburgo de Mello, Isobel Standen, Ioannis Stefanakos, Calum Imrie, Genaina Rodrigues, Ana Cavalcanti, Radu Calinescu, and Marsha Chechik. 2023. Analyzing and Debugging Normative Requirements via Satisfiability Checking . In *Proceedings of The 46th ACM International Conference on Software Engineering (ICSE 2024).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

ICSE 2024, 14 - 20 April, 2024, Lisbon, Portugal

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

1 INTRODUCTION

Software systems interacting with humans are becoming prevalent in domains such as transportation [30, 32], healthcare [26, 37], and assistive care [20]. The deployment of such systems raises significant concerns related to the *social, legal, ethical, empathetic, and cultural* (SLEEC) impact of their operation, e.g., the control software of an assistive-dressing robot [20] can accidentally expose a patient's privacy when dressing them in a room with the curtains open. To capture SLEEC concerns effectively, researchers [2, 5, 7, 36] have proposed to use *normative non-functional requirements* (N-NFRs) that constrain the acceptable range of system behavior. For instance, N-NFRs for the assistive-dressing robot may specify that the user's

well-being and privacy are prioritized by promptly completing the dressing task, especially when the patient is underdressed. N-NFRs are then used to inform all subsequent software engineering (SE) activities, such as design and verification.

Prior research recommended using SLEEC DSL, a state-of-the-art (SoTA) domain-specific language [23], for specifying N-NFRs as SLEEC DSL rules [14, 15]. SLEEC DSL is very close to natural language and has proven accessible to stakeholders without a technical background (e.g., lawyers, ethicists or regulators). A SLEEC DSL rule defines the required system *response* to specific *events*, and can be accompanied by one or more *defeaters* that use the **unless** keyword to specify circumstances that preempt the original response, and can optionally provide an alternative response. For instance, a simple normative rule $r1$ for the assistive-care robot from our earlier example can be encoded as "**when** `DressingRequest` **then** `DressingStarted` **within** 10 **minutes unless** `curtainsOpen`".

N-NFRs are typically defined by system stakeholders with widely different expertise (e.g., ethicists, lawyers, regulators, and system users). As such, they are susceptible to complex and often subtle *well-formedness issues* (WFI) (i.e., issues that are application- and domain-independent) such as conflicts, redundancies, insufficiencies, overrestrictions, and many others [24, 25, 29, 34]. In particular, the different objectives, concerns, responsibilities, and priorities of these stakeholders are sometimes incompatible, leading to logical inconsistency in all circumstances, or are subject to boundary conditions, rendering the system behavior infeasible given inconsistent (time) events [24]. Therefore, resolving such WFI is essential for ensuring the validity of N-NFRs. For instance, identifying and removing unnecessary redundant N-NFRs simplifies the validation and review process. Additionally, one must ensure that the N-NFRs are sufficiently strong to address the relevant SLEEC concerns without being overly restrictive and preventing the system from achieving its functional requirements.

N-NFRs present two additional layers of complexity compared to "regular" non-functional requirements. Firstly, the process includes challenges due to the involvement of stakeholders who often lack technical expertise. While this problem is shared with NFRs in general, N-NFR stakeholders include experts with very different backgrounds: lawyers, social workers, ethics experts, etc., and conflicting or redundant requirements can arise from different norms and factors [36] even if elicited by a single stakeholder! Secondly, the task becomes even more complex when capturing intricate non-monotonic conditions expressed via *if, then, unless* and time constraints (e.g., *act within 1 week*), which are often abundant in N-NFRs [6, 12, 22, 36]. For example, consider a helper robot with the N-NFR of "protecting patients' privacy" ($r1$), which might *conflict* with a "patient claustrophobia" N-NFR if the patient insists on keeping the curtains open while being dressed. Additionally, a social N-NFR may require respecting patient privacy, while a legal N-NFR may specify adherence to data protection laws, potentially creating a redundancy. Resolving these issues may lead to N-NFRs that no longer align with stakeholder goals, either being overly restrictive or insufficient to address SLEEC concerns comprehensively. Existing work has analyzed certain well-formedness issues, such as redundancy and conflicts, but is limited to interactions between only two SLEEC DSL rules [14], ignoring specific contexts [12], or offering limited debugging assistance that requires a technical background, e.g., familiarity with a model checker.

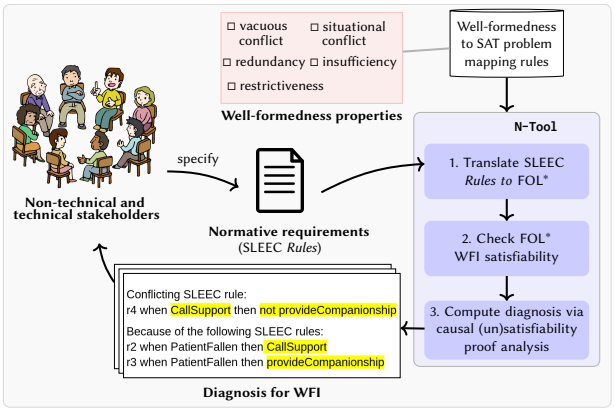

Figure 1: Overview of the N-Check approach.

In this paper, we propose a comprehensive approach to the analysis and debugging of N-NFRs by checking a broad range of well-formedness constraints, including redundancy, conflicts (accounting for different contexts), restrictiveness, and insufficiency. To the best of our knowledge, our solution is the first to provide such an extensive analysis and debugging approach for N-NFRs. Our approach is built on top of satisfiability checking for first-order logic with quantifiers over relational objects (FOL*) [13]. FOL* has been shown effective for modeling requirements that define constraints over unbounded time and data domains [13], and is therefore well suited for capturing the metric temporal constraints over actions and (unbounded) environmental measures in SLEEC DSL. Moreover, the proof of FOL* unsatisfiability enables us to provide diagnosis for the causes of N-NFRs WFIs. Capturing SLEEC DSL with FOL* also enables other logic-based analysis such as deductive verification.

Contributions. We contribute (a) a method for identifying well-formedness issues (WFI) in N-NFRs expressed in SLEEC DSL by converting them into satisfiability checking problems; (b) a method for generating a non-technical diagnosis that pinpoints the causes of WFI by analyzing the causal proof of (un)satisfiability; (c) a validation of the effectiveness of the approach over nine cases studies and 233 rules written by eight non-technical and technical stakeholders. Our approach is implemented in a tool N-Tool– see Fig. 1. N-Tool translates N-NFRs expressed as rules in the SoTA SLEEC DSL [14, 15] into FOL*; checks their satisfiability using the SoTA FOL* checker; and generates a non-technical diagnosis.

Significance. A comprehensive approach for analysis and debugging of N-NFRs is crucial in developing modern systems that increasingly interact with humans. To the best of our knowledge, we are the first to provide such an approach, supporting stakeholders with varying levels of technical expertise. The diagnosis computed by N-Check can be used by software engineers to (semi-)automatically generate and suggest patches for resolving WFI, thereby further assisting the technical and non-technical stakeholders.

The rest of this paper is organized as follows. Sec. 2 gives the background material for our work. Sec. 3 defines the N-NFR WFI problem that we tackle in this paper. Sec. 4 describes our approach, N-Check, to identify N-NFRs WFI using FOL* satisfiability checking. Sec. 5 presents our approach to utilizing the causal proof of (un)satisfiability to highlight the reasons behind WFI (e.g., conflict).

Table 1: Syntax of normalized SLEEC DSL with signature (E, M).

Name	Definition
Term	$t := c : \mathbb{N} \mid m \in M \mid -t \mid t + t \mid c \times t$
Proposition	$p := \top \mid \bot \mid t = t \mid t \geq t \mid$ **not** $p \mid p$ **and** $p \mid p$ **or** p
Obligation	$ob^+ := e$ **within** $t \mid ob^- :=$ **not** e **within** t
Cond Obligation	$cob^+ := p \Rightarrow ob^+ \mid cob^- := p \Rightarrow ob^-$
Obligation Chain	$\bigvee_{cob} := cob \mid cob^+$ **otherwise** $\bigvee_{cob}$
Rule	$r :=$ **when** e **and** p **then** $\bigvee_{cob}$
Fact	$f :=$ **exists** e **and** p **while** $(\bigvee_{cob} \mid$ **not** $\bigvee_{cob})$

Sec. 6 presents the evaluation of the approach's effectiveness. Sec. 7 compares N-Check to related work. Sec. 8 discusses future research.

2 PRELIMINARIES

2.1 Normalized Representation of SLEEC DSL

SLEEC DSL is an event-based language [14] for specifying normative requirements with the pattern "**when** trigger **then** response". The SLEEC DSL is based on propositional logic enriched with temporal constraints (e.g., **within** x **minutes**) and the constructs **unless**, specifying *defeaters*, and **otherwise**, specifying *fallbacks*.

SLEEC DSL is an expressive language, enabling specification of complex nested defeaters and responses. A SLEEC DSL rule can have many logically equivalent expressions, posing a challenge for analysis and logical reasoning. To handle it, the *normalized* SLEEC DSL introduced here can be used. In its rules, nested response structures are flattened; normalisation eliminates defeaters and captures their semantics using propositional logic. Here, we present the syntax and semantics of the *normalized* SLEEC DSL[1] and then define the accepted behaviour specified by normalized rules.

Normalized SLEEC DSL syntax. A normalised SLEEC DSL *signature* is a tuple $S = (E, M)$, where $E = \{e_1, \ldots, e_n\}$ and $M = \{m_1 \ldots m_n\}$ are finite sets of symbols for *events* and *measures*, respectively. As a convention, in the rest of the paper, we assume that all event symbols (e.g., OpenCurtain) are capitalized while measure symbols (e.g., underDressed) are not. Without loss of generality, we assume that every measure is numerical. The syntax of the normalized SLEEC DSL can be found in Tbl. 1, where, e.g., an *obligation* $ob = h$ **within** t specifies that the event e must (if $h = e$) or must not (if $h =$ **not** e) occur within the time limit t. We omit the time limit t if $t = 0$.

Example 1. Consider the normative rule ($r5$) "**when** OpenCurtainRequest **then** OpenCurtain **within** 30 **minutes unless** underDressed", where OpenCurtainRequest and OpenCurtain are events and underDressed is a measure. $r5$ is normalized as "**when** OpenCurtainRequest **and not** underDressed **then** OpenCurtain **within** 30 **minutes**".

Semantics. The semantics of SLEEC DSL is described over *traces*, which are *finite* sequences of *states* $\sigma = (\mathcal{E}_1, \mathbb{M}_1, \delta_1), (\mathcal{E}_2, \mathbb{M}_2, \delta_2), \ldots (\mathcal{E}_n, \mathbb{M}_n, \delta_n)$. For every time point $i \in [1, n]$, (1) $\mathcal{E}_i$ is a set of events that occur at time point i; (2) $\mathbb{M}_i : M \to \mathbb{N}$ assigns every measure in M to a concrete value at time point i, and (3) $\delta_i : \mathbb{N}$ captures the time value of time point i (e.g., the second time point can have the value 30, for 30 sec). We assume that the time values in the trace are strictly increasing (i.e., $\delta_i < \delta_{i+1}$ for every $i \in [1, n-1]$). Given a measure assignment $\mathbb{M}_i$ and a term t, let $\mathbb{M}_i(t)$ denote the result of substituting every measure symbol m with $\mathbb{M}_i(m)$. Since $\mathbb{M}_i(t)$ does not contain free variables, the substitution results in a natural number. Similarly, given a proposition p, we say that $\mathbb{M}_i \models p$ if p is evaluated to $\top$ after substituting every term t with $\mathbb{M}_i(t)$. Let a trace $\sigma = (\mathcal{E}_1, \mathbb{M}_1, \delta_1) \ldots (\mathcal{E}_n, \mathbb{M}_n, \delta_n)$ be given.

(Positive obligation). A positive obligation e **within** t is *fulfilled subject to time point* i, denoted $\sigma \models_i e$ **within** t, if there is a time point $j \geq i$ such that $e \in \mathcal{E}_j$ and $\delta_j \in [\delta_i, \delta_i + \mathbb{M}_i(t)]$. That obligation is *violated at time point* j, denoted as $\sigma \not\models_i^j e$ **within** t, if $\delta_j = \delta_i + \mathbb{M}_i(t)$ and for every j' such that $j \geq j' \geq i$, e does not occur ($e \notin \mathcal{E}_{j'}$).

(Negative obligation). A negative obligation **not** e **within** t is fulfilled subject to time point i, denoted as $\sigma \models_i$ **not** e **within** t, if for every time point j such that $\delta_j \in [\delta_i, \delta_i + \mathbb{M}_i(t)]$, $e \notin \mathcal{E}_j$. The negative obligation is violated at time point j, denoted as $\sigma \not\models_i^j$ **not** e **within** t, if (1) $\delta_j \in [\delta_i,\ \delta_i + \mathbb{M}_i(t)]$, (2) e occurs ($e \in \mathcal{E}_j$), and (3) for every $j \geq j' \geq i$, e does not occur ($e \notin \mathcal{E}_{j'}$).

(Conditional obligation). A conditional obligation $p \Rightarrow ob$ is fulfilled subject to time point i, denoted as $\sigma \models_i (p \Rightarrow ob)$, if p does not hold at time point i ($\mathbb{M}_i(p) = \bot$) or the obligation is fulfilled ($\sigma \models_i ob$). Moreover, $p \Rightarrow ob$ is violated at time point j, denoted $\sigma \not\models_i^j (p \Rightarrow ob)$, if p holds at i ($\mathbb{M}_i(p) = \top$) and ob is violated at j ($\sigma \not\models_i^j ob$).

(Obligation chain). A chain cob_1^+ **otherwise** $cob_2^+ \ldots cob_m$ is fulfilled subject to time point i, denoted as $\sigma \models_i cob_1^+$ **otherwise** $cob_2^+ \ldots cob_m$, if (1) the first obligation is fulfilled ($\sigma \models_i cob_1^+$) or (2) there exists a time point $j \geq i$ such that cob_1^+ is *violated* (i.e., $\sigma \not\models_i^j cob_1$) and the rest of the obligation chain (if not empty) is fulfilled at time point j ($\sigma \models_j cob_2^+$ **otherwise** $\ldots cob_m$).

(Rule). A rule **when** e **and** p **then** $\bigvee_{cob}$ is fulfilled in σ, denoted as $\sigma \models$ **when** e **and** p **then** $\bigvee_{cob}$, if for every time point i, where event e occurs ($e \in \mathcal{E}_i$) and p holds ($\mathbb{M}_i(p) = \top$), the obligation chain is fulfilled subject to time point i ($\sigma \models_i \bigvee_{cob}$). A *trace fulfills a rule set Rules*, denoted as $\sigma \models Rules$, if it fulfills every rule in the set (i.e., for every $r \in Rules$, we have $\sigma \models r$).

Example 2. Consider the normalized SLEEC DSL rule $r5$ in Ex. 1 and the trace σ_1 shown in Fig. 2, corresponding to $(\mathcal{E}_1, \mathbb{M}_1, \delta_1)$, where $\mathcal{E}_1 =$ OpenCurtainRequest, $\mathbb{M}_1$(underDressed) $= \top$, and $\delta_1 = 1$. The trace σ_1 fulfills the rule $r5$ (i.e., $\sigma_1 \models r5$), because the rule's triggering condition (i.e., **not** underDressed) is not satisfied (i.e., the user is under-dressed at the time of the request).

Definition 1 (Behaviour defined by *Rules*)**.** Let a rule set *Rules* be given. The accepted *behaviour* defined by *Rules*, denoted as $\mathcal{L}(Rules)$, is the largest set of traces such that for every trace σ in $\mathcal{L}(Rules)$ respects the *Rules*, i.e., $\sigma \in \mathcal{L}(Rules)$, $\sigma \models Rules$.

2.2 FOL* Satisfiability Checking

Here, we present the background on first-order logic with relational objects (FOL*) [13], which we use for satisfiability checking.

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 class symbols, and $\iota : R \to \mathbb{N}$ is a function that maps a relation to its arity. We assume that C's domain 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 of class $r \in R$ (denoted as $o : r$) is an object with $\iota(r)$ regular attributes and one special attribute, 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. Attributes are also named, and $o.x$ refers to o's attribute with the name 'x'. Each

[1] A translation from SLEEC DSL to the normalized form is provided in [3].

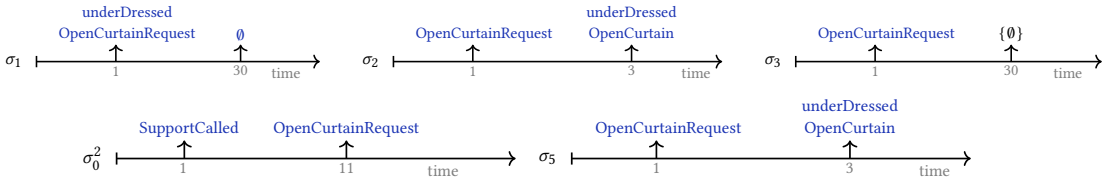

Figure 2: Five traces from the dressing robot example. σ_0^2 is a partial trace, from 0 to 2, used to illustrate situational-conflicts.

relational object o has a special attribute, $o.ext$. This attribute is a boolean variable indicating whether o exists in a solution. An FOL* *term* t is 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 $v \in V$, any relational object $o : r$, any index $k \in [1, \iota(r)]$ and any valid attribute name x. Given a signature S, FOL* formulas take the following forms: *(1)* $\top$ and $\bot$, representing "true" and "false"; *(2)* $t = t'$ and $t > t'$, for terms 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 over relational objects of a class, as defined in (4) and (5). Operators $\vee$ and $\forall$ are defined as $\phi_f \vee \psi_f = \neg(\neg\phi_f \wedge \neg\psi_f)$ and $\forall o : r \cdot \phi_f = \exists o : r \cdot \neg\phi_f$. We say that an FOL* formula is in *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 formula 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 expansion 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 3. Let A be a class of relational objects with attribute 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 $\forall a : A \cdot (\exists a', a'' : A \cdot (a.val = a'.val + a''.val) \wedge (a.val = 5))$ 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 $a' \leftarrow a_1$, $a'' \leftarrow a_2$; or if $a \leftarrow a_2$, then the formula is also satisfied by assigning $a' \leftarrow a_2$, $a'' \leftarrow a_2$.

3 WELL-FORMEDNESS PROPERTIES

In this section, we present a set of N-NFR well-formedness properties: conflicts, redundancies, restrictiveness, and insufficiency.

Redundancy. A rule in a rule set is *redundant* if removing it does increase the behaviours allowed by the rule set.

Definition 2 (Redundancy)**.** Given a rule set *Rules*, a rule $r \in Rules$, is *redundant*, denoted as $(Rules \backslash r) \Rightarrow r$, if $\mathcal{L}(Rules \backslash r) \subseteq \mathcal{L}(Rules)$.

Example 4. Consider a rule set $Rules = \{r5, r6, r7\}$, where $r5$, $r6$ and $r7$ are rules in Tbl. 2. The rule $r5$ is redundant in *Rules* since the behaviors allowed by $\mathcal{L}(r6) \bigcap \mathcal{L}(r7)$ are *subsumed* by the behaviors allowed by $\mathcal{L}(r5)$ (i.e., $\mathcal{L}(r6) \bigcap \mathcal{L}(7) \subset \mathcal{L}(r5)$), and thus removing $r5$ does not affect the behaviors allowed by $\mathcal{L}(Rules)$. If the time limit of $r7$ is changed to 40, then $r5$ is no longer redundant.

Vacuous Conflicts. A rule is *vacuously conflicting* if the trigger of the rule is not in the accepted behaviours defined by the rule set, i.e., triggering the rule in any situation will lead to a violation of some rules and thus cause a conflict.

Definition 3 (Vacuous Conflict)**.** Let a rule set *Rules* be given. A rule $r =$ **when** e **and** p **then** $\bigvee_{cob}$ in *Rules* is *vacuously conflicting* if for every trace $(\mathcal{E}_1, \mathbb{M}_1, \delta_1) \ldots (\mathcal{E}_n, \mathbb{M}_n, \delta_n) \in \mathcal{L}(Rules)$, $e \notin \mathcal{E}_i$ or $\mathbb{M}_p = \bot$ for every $i \in [1, n]$.

Example 5. The rule $r5$ (Tbl. 2) is conflicting in the rule set $Rules = \{r5, r8\}$ since triggering $r5$ would also trigger $r8$. Thus `OpenCurtain` must occur within 30 minutes ($r5$) but must not occur within 40 minutes ($r8$), which is a conflict.

Restrictive or Insufficient Rules. Recall from Def. 1, given a set of rules *Rules*, $\mathcal{L}(Rules)$ is the set of behaviors permitted by *Rules*. If *Rules* is overly restrictive, then $\mathcal{L}(Rules)$ might not contain some desirable behaviors essential to achieve the initial system purpose. On the other hand, if *Rules* is too relaxed, then $\mathcal{L}(Rules)$ might contain some undesirable behaviors, such as the ones causing SLEEC concerns. To capture the desirable and undesirable behaviors, we introduce a new `SLEEC DSL` language construct *fact*.

Definition 4 (Fact)**.** A `SLEEC DSL` *fact* f has the syntax "**exists** e **and** p **while** $\bigvee_{cob}$" or "**exists** e **and** p **while not** $\bigvee_{cob}$", where e is an event, p is a proposition and $\bigvee_{cob}$ is an obligation chain (see Tbl. 1). The fact **exists** e **and** p **while** $\bigvee_{cob}$ is *satisfied by a trace* $\sigma = (\mathcal{E}_1, \mathbb{M}_1, \delta_1), (\mathcal{E}_2, \mathbb{M}_2, \delta_2), \ldots (\mathcal{E}_n, \mathbb{M}_n, \delta_n)$, denoted as $\sigma \models f$, if there exists a time point i such that (1) $e \in \mathcal{E}_i$, (2) $\mathbb{M}_i(p) = \top$, and the response $\bigvee_{cob}$ is fulfilled at time point i (i.e., $\sigma \models_i \bigvee_{cob}$). Similarly, the fact **exists** e **and** p **while not** $\bigvee_{cob}$ is *satisfied by a trace* σ *if there exists a time point* i *such that the conditions (1)-(2) hold, and* $\bigvee_{cob}$ *is* not *fulfilled at time point* i. A set of facts *Facts is satisfied by a trace* σ, denoted as $\sigma \models Facts$, if $\sigma \models f$ for every $f \in Facts$. The behavior of *Facts*, denoted as $\mathcal{L}(Facts)$, is the largest set of traces such that for every $\sigma \in \mathcal{L}(Facts)$, $\sigma \models Facts$.

Similarly to rules, facts assert invariants over traces. However, unlike rules, which define responses to their triggering events, facts require occurrences of events in specific states.

Definition 5 (Restrictiveness)**.** Given a rule set *Rules* and the desirable behavior captured by a set of facts p, *Rules* is *overly-restrictive for* p if and only if $\mathcal{L}(Rules) \cap \mathcal{L}(p) = \emptyset$.

Intuitively, a rule set is overly restrictive if it is impossible to execute the desirable behavior while respecting every rule.

Example 6. Consider the rule set $Rules = \{r9, r10\}$ where $r9$ and $r10$ are in Tbl. 2. Suppose a desired behavior, formalized as p_1

in Tbl. 2, is that after a patient falls, the robot should stay with them for at least 30 minutes. *Rules* is overly restrictive for p_1 as $\mathcal{L}(p_1) \bigcap \mathcal{L}(Rules)$ is empty. Therefore, the system cannot execute the desired behavior in p_1 while respecting the rules in *Rules*.

Definition 6 (Insufficiency)**.** Given a rule set *Rules* and an undesirable behavior expressed as a set of facts c, *Rules* are *insufficient for preventing* c if and only if $\mathcal{L}(Rules) \cap \mathcal{L}(c) \neq \emptyset$.

Intuitively, a set of rules are insufficient if adhering to those rules can still allow executing an undesirable behavior.

Example 7. Consider the SLEEC DSL rule $r5$ and the privacy concern f_1 shown in Tbl. 2. Let σ_2 be the trace shown in Fig. 2, with states $(\{\texttt{OpenCurtainRequest}\}, \mathbb{M}_1, 1)$, $(\{\texttt{OpenCurtain}\}, \mathbb{M}_2, 3)$ where the measures of $\mathbb{M}_1$ evaluate p to $\bot$. Since σ_2 is in $\mathcal{L}(r5) \cap \mathcal{L}(f_1)$, then $\mathcal{L}(r5) \cap \mathcal{L}(f_1)$ is not empty. Therefore, $\{r5\}$ is insufficient to prevent f_1, i.e., $\{r_5\}$ allows a violation of the privacy concern.

Situational Conflicts. A SLEEC DSL rule may interact with other rules in a contradictory manner under specific situations, which can lead to conflicts between different rules. However, the definition of *vacuous conflict* (see Def. 3) does not capture such *situational conflicts* since these conflicts arise only in *particular* situations. If the conflicting situations are feasible, then the stakeholders need to resolve the conflict. We define the notion of *situational conflict* as a generalization of a vacuous conflict.

Definition 7 (Situation)**.** For a rule $r =$ **when** e **and** p **then** $\bigvee_{cob}$, an *r-triggering situation* is a tuple $(\sigma_0^k, \vec{M}_k)$ where $\sigma_0^k = (\mathcal{E}_0, \mathbb{M}_0, \delta_0) \ldots (\mathcal{E}_k, \mathbb{M}_k, \delta_k)$ is the prefix of a trace up to and including state k, $\vec{M}_k = \mathbb{M}_k \ldots \mathbb{M}_n$ are *partial* measures (i.e., the functions $\mathbb{M}_n \in \vec{M}_k$ might be undefined for some measures $m \in M$) for states from $k+1$ onwards, and σ_0^k triggers r in its kth state (i.e., $e \in \mathcal{E}_k \wedge \mathbb{M}_k(p) = \top$).

Intuitively, an r-triggering situation $(\sigma_0^k, \vec{M}_k)$ consists of (1) the complete state information of the past (until r is triggered) where all events, measures and their occurrence time are fully observed and (2) the partial measures of the future. Note that the situation cannot observe or control the occurrences of events beyond state k since we want to show that triggering r at state k is sufficient to cause a conflict regardless how the system responds after state k.

A situation is *feasible* with respect to a rule set *Rules* if the past up to state k (i.e., σ_0^k) does not *violate* any rule in *Rules*. To provide a formal definition of non-violation up to some state (k), we extend the semantics of SLEEC DSL to evaluate a trace up to a specified state k to check whether a *violation* of rule r has already occurred, denoted as $\sigma \vdash^k r$. We denote the extended semantics as the *bounded semantics*, which is presented in the supplementary material [3].

Example 8. Consider the rule $r5$ in Tbl. 2 and the trace $\sigma_3 = (\texttt{OpenCurtainRequest}, \mathbb{M}_1, 1)$ where $\mathbb{M}_1(\texttt{underDressed}) = \bot$ in Fig. 2. The trace does not violate $r5$ before state 1 ($\sigma \models^1 r5$), but it violates $r5$ after 30 minutes ($\sigma \not\models r5$) since the response (OpenCurtain) does not occur within 30 minutes after state 1.

A rule r is situationally conflicting if there exists a feasible r-triggering situation that eventually causes a conflict.

Definition 8 (Situational Conflict)**.** Let a rule set *Rules* be given. A rule r in *Rules* is *situationally conflicting* if there exists an r-triggering *situation* $(\sigma_0^k, \vec{M}_k)$ such that: (1) $\sigma_0^k \vdash^k$ *Rules*, and (2) there *does not* exist an extension $\sigma = (\mathcal{E}_1, \mathbb{M}_1, \delta_1), \ldots (\mathcal{E}_n, \mathbb{M}_n, \delta_n)$ of σ_0^k, such that σ preserves the measures in $\vec{M}_k$, i.e., for every $i > k$, and every measure $m \in M$, either $\mathbb{M}_i(m) = \vec{M}_k[i-k](m)$ or $\vec{M}_k[i-k](m)$ is undefined), and σ fulfills the rules ($\sigma \in \mathcal{L}(Rules)$).

Example 9. Consider the rule set $\{r5', r11\}$ with $r5'$ and $r11$ in Tbl. 2. Let σ_0^2 be the trace shown in Fig. 2, where the measures of the second state evaluate underDressed to $\bot$. For any partial measure $(*)$, $(\sigma_0^2, *)$ is an $r5'$-triggering situation because $r5'$ is triggered at time point 2, and not violated when triggered (i.e., $\sigma_0^2 \vdash^2 r5'$). However, at time point 2, the response of $r5'$ (**not** OpenCurtain **within** 30 **minutes**) conflicts with the remaining response of $r11$ (OpenCurtain **within** $40 - (11 - 1)$ **minutes**). Thus, for any future partial measure $(*)$, $r5$ is situationally conflicting w.r.t. $(\sigma_0^2, *)$.

Remark 1. A rule r is vacuously conflicting, if and only if it is situationally conflicting for every r-triggering situation $(\sigma_0^k, \vec{M}_k)$.

4 WELL-FORMEDNESS AND SATISFIABILITY

In this section, we present our approach, N-Check (see Fig. 1), for checking N-NFRs WFI via FOL* satisfiability checking. We first show that most of well-formedness checks (except situational conflicts) can be reduced to behavior emptiness checking (Sec. 4.1). Then, we translate the emptiness checking problem into FOL* satisfiability (Sec. 4.2) to leverage the SoTA FOL* satisfiability checker. Finally, we show that the identification of situational conflicts can also be reduced to satisfiability checking (Sec. 4.3).

4.1 Well-formedness via Emptiness Checking

In this section, we show that the task of identifying vacuous conflict (Def. 3), redundancy (Def. 2), restrictiveness (Def. 5), and insufficiency (Def. 6) for a given rule set can be reduced to behavior emptiness checking on the behavior of the rule set. Formally:

Definition 9 (Rule Emptiness Check)**.** Let a rule set *Rules* and a set of facts *Facts* be given. The *emptiness check* of *Rules* with respect to *Facts* is $\mathcal{L}(Rules) \cap \mathcal{L}(Facts) = \emptyset$.

The input to emptiness checking is a set of rules *Rules* and a set of facts *Facts*. In the following, we describe the appropriate inputs to enable emptiness checking for identifying N-NFRs WFI.

Detecting Vacuous Conflict. A rule $r \in Rules$ is vacuously conflicting in *Rules* if the behavior defined by *Rules* ($\mathcal{L}(Rules)$) does not include a trace where r is triggered. Therefore, we can determine if a rule is vacuously conflicting by describing the set of traces where r is triggered as facts in SLEEC DSL, and then checking whether it intersects with $\mathcal{L}(Rules)$ via emptiness checking (Def. 9).

Definition 10 (Triggering)**.** Let a rule $r =$ **when** e **and** p **then** $\bigvee_{cob}$ be given. The *triggering* of r, denoted as $\textsc{Trig}(r)$, is the fact $f =$ **exists** e **and** p **while** $\top$.

Lemma 1. *Let a set of rules Rules be given. A rule $r \in Rules$ is vacuously conflicting if and only if $\mathcal{L}(Rules) \cap \mathcal{L}(\textsc{Trig}(r))$ is empty.*

Lemma 1 is a direct consequence of Def. 3 and Def. 10.

Detecting Redundancies. Recall from Def. 2, that a rule is redundant in a given rule set *Rules* if and only if removing the rule from *Rules* does not affect the behavior of *Rules*.

Lemma 2. *Let a set of rules Rules be given. A rule r is not redundant if and only if there exists a trace σ such that $\sigma \models Rules \setminus r$ and $\sigma \not\models r$.*

Lemma 2 is a direct consequence of Def. 2. To take advantage of this lemma, we define a notion of *noncompliance* to capture $\sigma \not\models r$.

Definition 11 (Noncompliant). Let a rule r = **when** e **and** p **then** $\bigvee_{cob}$ be given. The *noncompliant fact of* r, denoted as $\textsc{Noncomp}(r)$, is the fact f = **exists** e **and** p **while not** $(\bigvee_{cob})$.

Example 10. The noncompliant fact derived from $r5$ (see Tbl. 2), is $\textsc{Noncomp}(r5)$ = **exists** `OpenCurtainRequest` **and not** `underDressed` **while not** `OpenCurtain` **within** 30 **min**.

Lemma 3. *Let a rule* r = ***when*** $e \wedge p$ ***then*** $\bigvee_{cob}$ *be given. For a trace* σ, $\sigma \not\models r$ *if and only if* $\sigma \models \textsc{Noncomp}(r)$.

The proof of Lemma 3 follows directly from the semantics of normalized `SLEEC DSL` described in Sec. 2.

Lemma 4. *Let a set of rules Rules be given. A rule* $r \in Rules$ *is redundant if and only if* $\mathcal{L}(Rules \setminus r) \cap \mathcal{L}(\textsc{Noncomp}(r))$ *is empty.*

Lemma 4 is a direct consequence of Def. 2, Lemma 2, and Lemma 3.

Detecting Restrictive or Insufficient N-NFRs. Definitions of restrictiveness (Def. 5) and insufficiency (Def. 6) are already expressed as emptiness checking problems.

4.2 Encoding SLEEC DSL Rules & Facts in FOL*

To check emptiness (Def. 9) of *Rules* with respect to *Facts*, we can check the satisfiability of the FOL* constraint $T(Rules) \wedge T(Facts)$.

The signature of `SLEEC DSL` is mapped to the signature of FOL* as follows: (1) every event $e \in E$ is mapped to a class of relational objects C^e; and (2) every measure $m \in M$ is mapped to a named attribute in relational objects of class C^M, where C^M is a special class of relational objects for measures. The function T then translates every `SLEEC DSL` rule r to an FOL* formula $T(r)$ such that $\mathcal{L}(r)$ is not empty if and only if $T(r)$ is satisfiable. T follows the semantics of normalized `SLEEC DSL` detailed in Sec. 2 to translate the exact condition for *fulfillment* and *violation*.

To translate the fulfillment relationship of `SLEEC DSL` responses with respect to a time point i (i.e., $\models_i$), T uses a helper function T^* (see supplementary [3]) to recursively visit the elements of the `SLEEC DSL` rule (i.e., obligations, propositions and terms) and encode the condition of fulfillment with respect to the time point i (where the time value and measures of state i are represented by the input relational object o^M). T also uses a helper function **Violation** (see [3]) to translate the condition for obligations and conditional obligations to be violated at some time point j. Specifically, T translates a rule **when** $e \wedge p$ **then** $\bigvee_{cob}$ into the constraint "whenever the triggering event e occurs $(\forall o^e : C^{event})$, the measures are available (i.e., $\exists o^M : C^M \cdot o^e.time = o^M.time$), and if the measures satisfy the triggering condition $(T^*(p, o^M))$, then the obligation chain should be fulfilled $(T^*(\bigvee_{cob}, o^M))$". Similarly, every fact is translated into the constraint "there exists an occurrence of an event e $(\exists o^e : C^{event})$, while the measures are available (i.e., $\exists o^M : C^M \cdot o^e.time = o^M.time$) where the condition p is satisfied $(T^*(p, o^M))$, and the obligation chain is fulfilled $(T^*(\bigvee_{cob}, o^M))$".

To ensure that no two different measures for the same time point exist, we introduce the FOL* *measure consistency axiom*: $axiom_{mc} \leftarrow \forall o_1^M, o_2^M : C^M (o_1^M.time = o_2^M.time \Rightarrow o_1^M \equiv o_2^M)$, where $\equiv$ is the semantically equivalent relational constraint that asserts equivalence of every attribute between o_1^M and o_2^M.

Table 2: Examples of `SLEEC DSL` rules and facts.

ID	`SLEEC DSL` Rule or Fact
$r5$	**when** `OpenCurtainRequest` **and not** `underDressed` **then** `OpenCurtain` **within** 30 **min.**
$r5'$	**when** `OpenCurtainRequest` **and not** `underDressed` **then not** `OpenCurtain` **within** 30 **min.**
$r6$	**when** `OpenCurtainRequest` **then** `SignalOpenCurtain` **unless** `underDressed`
$r7$	**when** `SignalOpenCurtain` **then** `OpenCurtain` **within** 20 **min.**
$r8$	**when** `OpenCurtainRequest` **then not** `OpenCurtain` **within** 40 **min.**
$r9$	**when** `UserFallen` **then** `SupportCalled` **within** 10 **min.**
$r10$	**when** `SupportCalled` **then** `LeaveUser` **within** 15 **min.**
$r11$	**when** `SupportCalled` **then** `OpenCurtain` **within** 40 **min.**
$r12$	**when** `OpenCurtainRequest` **and not** `underDressed` **then** `OpenCurtain` **within** 20 **min.**
$r13$	**when** `A` **then** `B` **within** 10 **min.**
$r14$	**when** `A` **and** $P(A)$ **then** `B` **within** 20 **min.**
p_1	**when** `UserFallen` then **not** `LeaveUser` **within** 30 **min.**
f_1	**exists** `OpenCurtainRequest` **and not** `underDressed`

We prove the correctness of the FOL* translation and provide the detailed translation function T in supplementary material [3].

Example 11. The FOL* translation of rule $r5$ is $\forall o^{cr} : C^{\texttt{OpenCurtainRqst}} . (\exists o^M : C^M . (o^{cr}.time = o^M.time \wedge \neg o^M.\texttt{underDressed})) \Rightarrow \exists o^{co} : C^{\texttt{OpenCurtain}} . (o^{cr}.time \leq o^{co}.time \leq o^{cr}.time + 30)$.

Example 12. Consider the rule set $Rules = \{r5, r8\}$. $\textsc{Trig}(r5)$ is **exist** `OpenCurtainRequest` **and not** `underDressed` **while** $\top$. To check whether r_5 is vacuously conflicting, we reduce the emptiness problem for $\mathcal{L}(Rules) \bigcap \mathcal{L}(\textsc{Trig}(r5))$ to the satisfiability of FOL* formula $T(r5) \wedge T(r8) \wedge T(\textsc{Trig}(r5)) \wedge axiom_{mc}$. The FOL* formula is UNSAT which means that the set of all traces where $r5$ is triggered does not intersect with the behaviour allowed by $r8$, and thus $r5$ is vacuously conflicting in *Rules* due to $r8$.

Example 13. Consider the rule set $Rules = \{r5, r6, r7\}$. To check whether $r5$ is redundant, we reduce the emptiness checking problem for $\mathcal{L}(Rules \setminus r5) \bigcap \mathcal{L}(\textsc{Noncomp}(r5))$ to satisfiability of the FOL* formula $T(r8) \wedge T(\textsc{Noncomp}(r5)) \wedge axiom_{mc}$. The FOL* formula is UNSAT thus the set of traces not complying with $r5$ ($\textsc{Noncomp}(Rules)$) does not intersect with the behaviour of $r6$ and $r7$. $r5$ is logical consequence of $r6$ and $r7$, and so *Rules* is redundant.

Example 14. Consider the `SLEEC DSL` rule $r5$ and the fact f_1 in Tbl. 2 for describing the desirable behavior "a user requests to open the curtain while she is not under-dressed". The FOL* formula $T(r5) \wedge T(f_1)$ has a satisfying solution (D, v) where (1) $D = \{o_1^M, o_2^M, o^{\texttt{OpenCurtainRqst}}, o^{\texttt{OpenCurtain}}\}$; (2) $v(o.ext = \top)$ for every $o \in D$; and (3) $v(o_1^M.\texttt{underDressed}) = \bot \wedge v(o_1^M.\texttt{time}) = 1 \wedge v(o_2^M.\texttt{time}) = 2$. The solution is mapped to a trace $\sigma_5 = (\{\texttt{OpenCurtainRequest}\}, \mathbb{M}_1, 1), (\{\texttt{OpenCurtain}\}, \mathbb{M}_2, 3)$ where $\mathbb{M}_1(\texttt{underDressed}) = \bot$, shown in Fig. 2. The trace σ is contained in $\mathcal{L}(r5) \cap \mathcal{L}(f_1)$. Therefore, $r5$ does not restrict the behavior defined in f_1.

4.3 Situational Conflicts via Satisfiability

Recall from Def. 8 that the detection of situational conflict involves searching for a situation (i.e., a partial trace) such that all of its extensions are rejected by a given rule set. This corresponds to an *exists* (situation)-*forall* (extension) query, which makes a direct reduction to emptiness checking, defined in Sec. 4.1, difficult. Fortunately, in the `SLEEC DSL` language, the time limit of obligations is always defined as *continuous* time windows. This characteristic allows us to identify conflicts when a positive window (an event

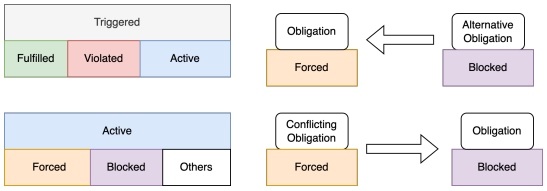

Figure 3: The relationship between different states of obligations.

e must occur) is completely covered by the continuous union of negative windows (the event *e* must not occur), and detect a subset of situational conflicts by checking the reachability of a state where a rule is triggered while its obligations are blocked.

In this section, we propose a sound approach, through a direct reduction to FOL* satisfiability, to detect situational conflicts. Note that we do not tackle classes of situational conflicts without blocked responses, called *transitive situational conflicts* (see supplementary material [3]) We start by an informal discussion of the different states of obligations: triggered, fulfilled, violated, active, forced, and blocked. The relationship between them is graphically displayed in Fig. 3. The formal definitions are in supplementary material [3].

States of Obligations. Given a situation (σ_0^k, M_k), an obligation *ob* is *triggered* if some states in σ_0^k trigger a rule *r* where *ob* is the response of *r*. Once *ob* is triggered, it might be *fulfilled* or *violated* in σ_0^k. However, if *ob* is neither fulfilled nor violated, the obligation is then *active* which can be fulfilled or violated in the future (in the extension of σ_0^k) in order to satisfy all the rules. The relationship between triggered, fulfilled, violated and active obligations is visualized on the left hand side of Fig. 3.

Example 15. Consider the rule set $\{r5', r11\}$ where $r5'$ and $r11$ are in Tbl. 2. Let σ_0^2 be a trace prefix, with 2 states, shown in Fig. 2, where the measures of the second state evaluate `underDressed` to $\perp$. In the situation $(\sigma_0^2, *)$, $r5'$ is triggered at time point 2, and its response, ob = **not** `OpenCurtain` **within** 30 **min**, is triggered as well. Since event `OpenCurtain` does not occur at time point 2, *ob* is not violated but is also not fulfilled. Therefore, *ob* is an active obligation at time point 2. At time point 1, $r11$ is triggered, and its response, ob' = `OpenCurtain` **within** 40 **min**, is triggered as well. Since `OpenCurtain` does not occur at either time point 1 or 2, ob' is neither fulfilled nor violated. Thus, ob' is active at time point 2.

An active obligation can be in two special states: *forced* and *blocked*. An active obligation *ob* is *forced* if it is the only remaining option for an active response, i.e., there is no alternative to *ob* that is not blocked. Therefore, violating a forced obligation would inevitably cause a violation of some rules. An obligation *ob* is blocked if there exists a "conflicting" obligation ob', and ob' is forced. The dependencies between forced and blocked obligations are visualized in the right hand side of Fig. 3. When a rule is triggered while its response is blocked in $(\sigma_0^k, \vec{M}_k)$, a conflict is inevitable.

Example 16. Continuing from Ex. 15, let *ob* and ob' be the two active obligations triggered by σ_0^2 at time points 2 and 1, respectively. Since time point 2 occurs 10 min. after time point 1 in σ_0^k, the remaining time limit of the obligation ob' is 30 min. Both *ob* and ob' are forced since they do not have alternatives. Both *ob* and ob' are also blocked at time point 2 since they are conflicting.

There are circular dependencies between forced and blocked obligation chains! Fortunately, by encoding the status definitions into FOL*, we can leverage the FOL* solver's ability to incrementally and lazily unroll the necessary definitions to resolve such dependencies (see supplementary material [3] for details.)

Table 3: FOL* causal proof of UNSAT for ϕ_1 and ϕ_2 in Ex. 17.

ID	Lemma	Derivation Rule	Deps
1	$\forall a : A \exists b : B \cdot (a.time \le b.time \le a.time + 10)$	**INPUT**: ϕ_1	{}
2	$\exists a : A \forall b : B \cdot (b.time \ge a.time + 20 \wedge p(a, b))$	**INPUT**: ϕ_2	{}
3	$\forall b : B \cdot (b.time \ge a_1.time + 20 \wedge p(a_1, b))$	**EI**: $[a \leftarrow a_1]$	{2}
4	$\exists b : B \cdot (a_1.time \le b.time \le a_1.time + 10)$	**UI**: $[a \leftarrow a_1]$	{1, 3}
5	$a_1.time \le b_1.time \le a_1.time + 10$	**EI**: $[b \leftarrow b_1]$	{4}
6	$b_1.time \ge a_1.time + 20 \wedge p(a_1, b_1)$	**UI**:$[b \leftarrow b_1]$	{3, 5}
7	$b_1.time \ge a_1.time + 20$	**And**	{6}
8	$\perp$	**Impl**	{5, 7}

Lemma 5. *For every rule* r = ***when*** $e \wedge p$ ***then*** $\bigvee_{cob}$ *in a rule set Rules, if there exists a r-triggering situation* $(\sigma_0^k, \vec{M}_k)$ *(see Def. 7) where the obligation chain* $\bigvee_{cob}$ *is blocked at time point k, then r is situationally conflicting with respect to the situation* $(\sigma_0^k, \vec{M}_k)$.

Lemma 5 enables the detection of situational conflicts for a rule *r* in *Rules* by encoding the sufficient condition for situational conflict into FOL* (denoted by TSC(*Rules*, *r*)). Every satisfying solution to TSC(*Rules*, *r*) represents a situation where *r* is situational conflicting. The proof of Lemma 5, the situational encoding TSC(*Rules*, *r*) and its correctness proof are in the supplementary material [3].

4.4 The N-Check Approach

N-Check turns well-formedness problems into FOL* satisfiability checking using Lemma 1 for vacuous conflicts, Lemma 5 for situational conflicts, Lemma 4 for redundancies, and Def. 5 and 6 for restrictiveness and insufficiency, respectively. Following this, N-Check encodes the input `SLEEC DSL` rules into FOL* (Step 1 in Fig. 1), as explained in Sec. 4.2. Then, it identifies N-NFR WFI using a SoTA FOL* satisfiability checker (Step 2). The remaining step, computing the diagnosis, is described in Sec. 5.

5 DIAGNOSIS FOR WELL-FORMEDNESS

N-Check not only detects WFI but also diagnoses their causes, thereby aiding stakeholders in comprehending and debugging problematic rules. Below, we present the diagnosis process for different well-formedness problems, leveraging the result of satisfiability checking. We first show how to utilize the causal proof of (un)satisfiability (Sec. 5.1) to pinpoint the reasons behind redundancy, vacuous-conflict, and restrictiveness (Sec. 5.2). Then we present a satisfying FOL* solution-based diagnosis for insufficiency and combine the solution-based and proof-based diagnosis to highlight the cause of situational conflict (Sec. 5.3).

5.1 Causal FOL* Proof

We first introduce causal FOL* proofs and FOL* derivation rules. Then we focus on the **Impl** rule and discuss its role as the *core* of UNSAT derivations for computing our diagnosis. *A causal FOL* proof* is a sequence of derivation steps $L_1, L_2 \ldots, L_n$. Each step L_i is a tuple $(i, \psi, o, Deps)$ where (1) i is the ID of the derivation step; (2) ψ is the derived FOL* lemma; (3) o is the name of the *derivation rule* used to derive ψ; and (4) *Deps* are IDs of dependent lemmas for deriving ψ. A *derivation step is sound* if the lemma ψ can be obtained via the derivation rule o using lemmas from *Deps*. A *proof is sound* if every derivation step is sound. The proof is *refutational* if the final derivation contains the lemma $\perp$ (UNSAT).

Example 17. Let FOL* formulas, $\phi_1 : \forall a : A \exists b : B \cdot (a.time \leq b.time \leq a.time + 10)$ and $\phi_2 : \exists a : A \forall b : B \cdot (b.time \geq a.time + 20 \wedge p(a, b))$ be given, where A and B are classes of relational objects, *time* is an attribute of type $\mathbb{N}$ and p is a complex predicate. $\phi_1 \wedge \phi_2$ is UNSAT, and the proof of UNSAT is shown in Tbl. 3. The proof starts by introducing the **Input** ϕ_1 and ϕ_2 (steps 1, 2), and then uses existential instantiation (**EI**) in steps 3, 5 and universal instantiation (**UI**) in steps 4, 6 to eliminate quantifiers in ϕ_1 and ϕ_2. In step 7, the **And** rule decomposes the conjunction and derives part of it as a new lemma. Finally, in step 8, $\bot$ is derived with the **Impl** rule from the quantifier-free (QF) lemmas derived in steps 5 and 7. The proof is refutational because step 8 derives $\bot$.

Given a refutational proof, we can check and trim the proof following a methodology similar to [19]. In the rest of the section, we assume every proof is already trimmed, and provide the detail on proof checking and trimming in supplementary material [3].

In a refutational proof, the final derivation of $\bot$ must be carried out by the **Impl** rule (e.g., step 8 in the proof in Fig. 3). Given a proof L, $Imp(L)$ is the set of QF lemmas derived via or listed as dependencies for **Impl**. Intuitively, $Imp(L)$ is the set of *core* QF formulas derived from the inputs following other derivation rules and sufficient to imply the conflict (UNSAT). By analyzing lemmas in $Imp(L)$ and backtracking their dependencies, we determine which parts of inputs are used to derive UNSAT.

Example 18. Continuing from Ex.17, the final step (step 7) of the proof L derives $\bot$ using the **Impl** rule with dependency $\{5, 7\}$. Therefore, $Imp(L)$ is the lemmas derived at steps 5 and 7, and the term $p(a)$ in ϕ_2 is not involved for deriving these lemmas and thus not necessary for the UNSAT conclusion. Therefore, the proof remains valid by removing $p(a)$ from ϕ_2.

5.2 Diagnosis for Redundancy, Vacuous and Situational Conflict, and Restrictiveness

In addition to reporting a binary "yes" or "no" answer to the satisfiability, the FOL* satisfiability checker LEGOS also provides a causal proof of UNSAT (see Sec. 5.1) if the encoded formula is unsatisfiable. We project the proof into the input SLEEC DSL rules to highlight the causes of WFI problems at the level of *atomic elements*.

Definition 12 (Atomic element)**.** An *atomic element* in SLEEC DSL is one the followings: (1) an *atomic proposition ap*: $\top \mid \bot \mid t = t \mid t \geq t \mid \neg ap$, (2) a triggering event e (where e in **when** $e \wedge \ldots$ **then** $\ldots$ or **exists** $e \wedge \ldots$ **while** $\ldots$), (3) a response event e' in an obligation (in e' **within** $\ldots$), or (4) a deadline t of an obligation (in $\ldots$ **within** t).

We now informally define *involved atomic elements* [2]. Let a causal proof L and the subset of lemmas derived by **Impl** $Imp(L)$ be given. An atomic element *ae* is *involved in the derivation of UNSAT*, denoted as $inv(ae, L)$, if the atomic element encodes the necessary dependency of lemmas in $Imp(L)$. An *atomic proposition ap is involved* if it is used to encode some QF lemma in $Imp(L)$. *A triggering event e is involved* if $Imp(L)$ contains a QF implication where the premise of implication is $o^e.ext$ ($o^e \Rightarrow \ldots$). A *response event is involved* if $Imp(L)$ contains a QF implication where the conclusion of implication is $o^e.ext$ ($\ldots \Rightarrow o^e$). A *deadline t is involved* if there is a lemma in $Imp(L)$ that contains the time constraint encoded from t.

[2] A formal definition is in supplementary material [3].

```
Redundant SLEEC rule:
r5 when OpenCurtainRequest then OpenCurtain within 30 minutes
   unless {underDressed}

Because of the following SLEEC rules:
r7 when SignalOpenCurtain then OpenCurtain within 20 minutes
r6 when OpenCurtainRequest then SignalOpenCurtain unless {underDressed}
```

Figure 4: Redundancy diagnosis for Ex. 4: a redundant rule $r5$ together with the set of rules $\{r7, r8\}$ with which $r5$ is redundant. Redundant clauses are highlighted.

```
f1 exists OpenCurtainRequest and (not {underDressed})
Concern is raised
at time 0: OpenCurtainRequest
at time 0: OpenCurtain
at time 0: Measure(underDressed=False)
```

Figure 5: Insufficiency diagnosis for Ex. 7: a trace where a concern is raised while respecting the rules.

Example 19. Following Ex. 18, the proof of UNSAT L for $\phi_1 \wedge \phi_2$ is shown in Fig. 3, and $\phi_1 \wedge \phi_2$ is the encoding for redundancy checking of $r13$ in the rule set $\{r13, r14\}$ where $r13$ and $r14$ are in Tbl. 2: $\phi_1 = T(r13)$ and $\phi_2 = T(\text{NONCOMP}(r14))$. From the proof L and $Imp(L)$, the involved atomic elements in $r13$ are A and B **within** 5 **min**. The involved atomic elements in $r14$ are A and B **within** 10 **min**. The involved elements indicate the reason of redundancy: $r13$ and $r14$ have the same trigger A, and the response of $r14$ (B **within** 20 **min**) is logically implied by the response of $r13$ (B **within** 20 **min**). The element $p(A)$ in $r14$ is not involved, and the redundancy does not depend on it.

The diagnosis for redundancy, vacuous-conflict, and restrictiveness presents only those SLEEC DSL rules r in an UNSAT proof L, where L derives $T(r)$ by the derivation rule **Input**. The diagnosis also highlights every involved atomic element (see Fig. 4). To efficiently identify involved atomic elements for a given proof L, the translation function is augmented to keep track of the mapping between input SLEEC DSL elements (events, propositions, terms) and the translated FOL* elements. For every lemma in $Imp(L)$, we can efficiently obtain the source elements in SLEEC DSL using the mapping and identify the involved atomic elements.

5.3 Other Diagnoses

Diagnosis for Insufficiency. A set of rules *Rules* is insufficient to block a concern f if the FOL* formula $T(Rules) \wedge T(f)$ is satisfiable. The diagnosis of *Rules* insufficiency is a trace constructed from a solution to the insufficiency query for f and representing one undesirable behavior allowed by *Rules*, that realizes f (see Fig. 5).

Diagnosis for Situational Conflict. A rule r is situationally conflicting if there exists a situation under which there is a conflict. The diagnosis needs to capture both the situation and the reason for the conflict. Therefore, we produce a hybrid diagnosis which combines the *solution-based diagnosis* including a trace representing the situation where the rule is conflicting, and the *causal-proof diagnosis* pinpointing the reasons for the conflicts (using the N-NFR rules) given the situation. To do so, we first obtain the conflicting situation, and then encode it as facts to check whether the situation causes conflict between the rules. More specifically, given a rule $r \in Rules$, the situation where r is conflicting can be constructed from the FOL* solution to the situational-conflict checking query. By asserting the situation as facts f_{sit} and then checking whether

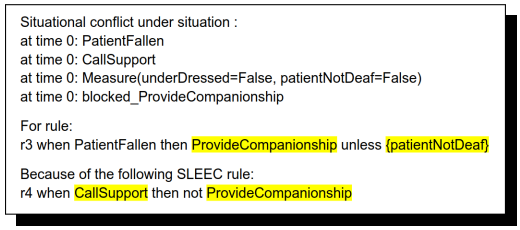

Figure 6: Situational conflict diagnosis for Ex. 9: a trace of the situation where the conflict occurs, the conflicting rule $r3$ and the rule set $\{r4\}$. The conflicting clauses are highlighted.

Table 4: Case study statistics.

name	rules (evnt.| msr.)	defeaters	N-TS|TS	domain	stage
ALMI	39 (41|15)	8	2 | 2	social	deployed
ASPEN	15 (25|18)	5	2 | 2	env.	design
AutoCar	19 (36|26)	20	1 | 1	transport	design
BSN	29 (33|31)	4	2 | 2	health	proto.
DressAssist	31 (54|42)	39	4 | 1	care	proto.
CSI-Cobot	20 (23|11)	8	2 | 1	manifacture	proto.
DAISY	26 (45|31)	20	3 | 1	health	proto.
DPA	26 (28|25)	4	1 | 1	education	design
SafeSCAD	28 (29|20)	3	2 | 1	transport	proto.

$\mathcal{L}(Rules) \cap \mathcal{L}(f_{sit})$ is empty, we can confirm the conflict and then use causal-proof based diagnosis to highlight the causes of conflict given the situation, as illustrated in Fig. 6.

6 EVALUATION

Tool Support. We implemented our N-Check approach in a tool `N-Tool` (see the architecture in Fig. 1). Input to `N-Tool` are N-NFRs expressed in `SLEEC DSL`. Outputs are diagnoses for each WFI issue identified (e.g., Fig. 4-5). Built using Python, `N-Tool` encodes `SLEEC DSL` rules into FOL* WFI satisfiability queries. It then uses `LEGOS` [13] as the back-end to check WFI FOL* satisfiability and produce the causal (un)satisfiability proof. Finally, it computes WFI diagnoses based on the proof. When no issues are identified, `N-Tool` produces a proof of correctness (i.e., absence of WFI). `N-Tool` also includes a user interface to allow users to input `SLEEC DSL` rules. `N-Tool` together with the evaluation artifacts is available in [3].

To evaluate our approach we ask the research question (**RQ**): How effective is `N-Tool` in detecting WFIs?

Models and Methods. We have divided the authors of the paper into two groups: the *analysis experts* (AEs) who have defined and developed the approach, and the *stakeholders*, further categorized into *technical* (TSs) and *non-technical* (N-TSs). We have ensured that there was no overlap between these groups. The stakeholders include: an ethicist, a lawyer, a philosopher, a psychologist, a safety analyst, and three engineers. The TSs identified nine real-world case studies from different stages, ranging from the design phase to deployed systems, as shown in Tbl. 4.

The case studies were as follows. (1) ALMI [18]: a system assisting elderly and disabled users in a monitoring or advisory role and with physical tasks; (2) ASPEN [11]: an autonomous agent dedicated to forest protection, providing both diagnosis and treatment of various tree pests and diseases; (3) AutoCar [4]: a system that implements emergency-vehicle priority awareness for autonomous vehicles; (4) BSN [16]: a healthcare system detecting emergencies by continuously monitoring the patient's health status; (5) DressAssist [21, 36]: an assistive and supportive system used to dress and provide basic care for the elderly, children, and those with disabilities; (6) CSI-Cobot [35]: a system ensuring the safe integration of industrial collaborative robot manipulators; (7) DAISY [9]: a sociotechnical AI-supported system that directs patients through an A&E triage pathway; (8) DPA [1]: a system to check the compliance of data processing agreements against the General Data Protection Regulation; (9) SafeSCAD [8]: a driver attentiveness management system to support safe shared control of autonomous vehicles. Case studies were chosen (i) to test whether our approach scales with case studies involving complex N-NFRs with numerous defeaters and time constraints (AutoCar, DAISY, DressAssist); (ii) to examine the benefits of our approach on early-stage case studies (e.g., ASPEN and AutoCar), as compared to those at a more advanced stage (e.g., ALMI, DPA, BSN); (iii) to compare case studies involving many stakeholders (DressAssist, and DAISY) with those having fewer ones (DPA and AutoCar), and (iv) to consider different domains including the environment, healthcare, and transport.

Table 5: Run-time performance and WFI. True positive (TP) and false negative (FP) issues identified for each vacuous conflict (v-conf.), situational conflict (s-conf.), redundancy (redund.), restrictiveness (restrict.), and insufficiency (insuff.). Spurious issues are in bold.

case studies	v-conf. (TP | FP)	s-conf. (TP | FP)	redund. (TP | FP)	restrict. (TP | FP)	insuffi. (TP | FP)	time (sec.)
ALMI	0 | 0	3| 0	0 | 0	0 | 0	1 | **1**	30
ASPEN	0 | 0	3 | 0	1 | 0	0 | 0	5 | 0	25.3
AutoCar	0 | 0	4 | 0	2 | 0	0 | 0	9 | 0	27.7
BSN	0 | 0	0 | 0	0 | 0	0 | 0	3 | 0	46
DressAssist	0 | 0	1 | 0	0 | 0	0 | 0	1 | **3**	20.3
CSI-Cobot	0 | 0	0 | 0	2 | 0	0 | 0	6 | **1**	25.3
DAISY	0 | 0	1 | 0	1 | 0	0 | 0	5 | 0	30.4
DPA	0| 0	0| 0	0 | 0	0 | 0	4 | 0	21.4
SafeSCAD	0 | 0	8 | 0	2 | 0	**2** | 0	4 | **1**	42.4

Table 6: Debugging effort for non-spurious WFI in Tbl. 5. Metrics are: # issues, # added events or measures (add D), # added (add R), refined, merged, and deleted rules, and # cases classified as too complex.

case studies	total	resolution effort					complex
		add D	add R	refine	merge	delete	
ALMI	4	1	4	0	0	1	-
ASPEN	9	1	3	3	1	2	-
AutoCar	11	1	20	4	0	1	-
BSN	3	3	3	0	0	0	-
DressAssist	1	1	0	1	0	0	-
CSI-Cobot	8	3	7	1	0	1	-
DAISY	7	0	5	1	1	0	-
DPA	4	0	4	1	0	2	-
SafeSCAD	16	5	6	9	4	2	**1**

For each case study, 1-2 TSs were paired with 1-4 N-TSs; together they built a set of normative requirements (N-NFRs). They then met to manually review, discuss, and agree on these N-NFRs. They specified a total of 233 N-NFRs. The number of `SLEEC DSL` rules ranged from 15 (ASPEN) to 39 (ALMI), and the number of defeaters in these rules ranged from 4 (DPA and BSN) to 39 (DressAssist). Tbl. 4 reports the number of events (evnt.), measures (msr.), rules (rules), defeaters (**unless**), and both technical (TS) and non-technical (N-TS) stakeholders involved in the writing of the N-NFRs.

RQ. Once the N-NFRs were built, the AEs met with the TSs and N-TSs involved in each case study to evaluate the WFI, in one session for each case study. When no issue was identified, AEs provided a proof of correctness. We focused on the stakeholders' ability to understand the feedback given by `N-Tool` and split the identified concerns into *relevant* (TP) or *spurious* (FP). Additionally,

the stakeholders specified which part of the diagnosis helped them understand and perform this classification. The results are in Tbl. 5.

It took N-Tool between 21 and 49 seconds to analyze the N-NFR WFI in each case study, 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. No vacuous conflicts were identified, likely due to the careful manual review of the rules prior to our analysis. To ensure correctness, we augmented the proof of absence of vacuous conflicts produced by N-Tool with feedback produced by AutoCheck [13], an existing tool for analyzing vacuous conflicts. This evaluation, see [3], confirmed the soundness of our approach.

N-Tool identified 20 situational conflicts involving each 2-3 rules, and highlighted 2-3 environment constraints from the 20 to 33 possible ones. This means that the rules conflict only in highly limited environments. Reviewing these conflicts without tool support is quite challenging, especially for case studies involving many rules (DAISY) or numerous environmental constraints (e.g., DAISY, AutoCar, and ASPEN). Here, the diagnosis helped stakeholders understand the issue by: (1) specifying which rules to focus on; (2) pinpointing the specific clauses causing the conflicts; and (3) providing a trace specifying the situations in which the conflicts occur and the blocked response, thus aiding stakeholders in understanding the environments under which the conflicts occur. N-Tool also identified eight redundancies involving 2-3 rules each. The diagnosis provided all the necessary information to understand each redundancy, including identifying and extracting redundant rules and highlighting redundant clauses. Stakeholders quickly confirmed that none of these redundancies were intentional, so the identified redundancies were unnecessary and could have led to future inconsistencies. Regarding restrictiveness, one case study, SafeSCADE, was found to have N-NFRs that were too restrictive, preventing the system from achieving two of its main purposes. The provided diagnosis directly helped the stakeholders.

Cases of insufficiency were identified in every case study, raising the total of 38 concerns. Determining the spuriousness of some concerns required a non-trivial examination and consideration process. Specifically, the stakeholders analyzed the traces provided by our diagnosis (e.g., Fig. 4) and engaged in detailed discussions on whether their rules needed to address each identified concern, e.g., whether enabling patients to disconnect sensors tracking their health is a bad behaviour. At the end, they identified spurious concerns only in four case studies. The feedback provided by N-Tool enabled the stakeholders to understand the meaning of the identified cases for all types of WFI, limiting spurious feedback to raised concerns. Thus, we successfully answered our RQ.

Discussion about diagnosis support for debugging WFIs. Once the stakeholders understood and classified each identified WFI issue as non-spurious, we asked them, in the same session, to use the diagnosis to *debug* the issue. We recorded the resolution effort, i.e., the number of definitions/rules added, refined, merged or deleted, as a proxy to the resolution complexity. The results are in Tbl. 6. The enumeration of redundant rules along with the highlighting of redundant clauses (e.g., Fig. 4) provided by N-Tool enabled the stakeholders to easily resolve the cases of redundancy. They knew directly what to fix (mostly merge, delete, or refine). N-Tool supported the stakeholders in resolving situational conflicts by: (1) specifying the specific clauses causing the conflicts, which enabled stakeholders to focus on these rules, making the resolution process more targeted; (2) providing the event and measure values representing the situations in which the conflicts occurred, which helped stakeholders consider the environmental constraints comprehensively while devising a resolution strategy (merging and refining both rules and definitions), and (3) highlighting the blocked response in the trace (e.g., see Fig. 4), which allowed stakeholders study priority between the conflicting rule responses.

Regarding insufficiency, the stakeholders specifically appreciated the optimal size of the trace (e.g., Fig. 4), as it helped them understand which events interfere with the identified concerns. Regarding restrictiveness, the tool provided all of the necessary information (i.e., pinpointed clauses on the rules restricting the system purpose) for the stakeholders to straightforwardly and trivially resolve the issue. Resolving restrictiveness issues (only two instances found across all the case studies) resulted in refining and merging the rules. In the case of AutoCar, checking restrictiveness helped refine constraints and correct incorrect responses, as well as establish priority between the rules, ultimately simplifying the rules. It also enabled the refinement of definitions and rules to be more specific (less restrictive). For the majority of resolutions, stakeholders added a rule (60%) or refined an existing one (23%). They very rarely removed one (10%), and even less frequently merged existing rules (7%). Our diagnosis supported the stakeholders in resolving the WFI when time allowed.

Summary. Our experiment demonstrated that our approach effectively and efficiently identifies N-NFR WFI. Both technical and non-technical stakeholders found the diagnosis to be comprehensible and helpful in resolving the identified issues. Notably, N-Tool allowed stakeholders to add missing rules and encouraged them to consider missing cases without constraining their imagination during the elicitation process. The inclusion of traces proved beneficial, enabling stakeholders to understand concerns that were not addressed by their N-NFRs and providing insights into causes of conflicts. However, stakeholders expressed the desire for a methodology to aid in issue resolution, as well as suggestions for potential resolution patches. They also would have preferred using the tool during the elicitation process to help them identify and address potential issues proactively. Overall, the stakeholders really appreciated the assistance provided by N-Tool in addressing the WFI.

Threats to validity. (1) The case studies were built by the co-authors of this paper. We mitigated this threat by separating the authors onto those who developed the validation approach and those who built the case studies, and ensured that a complete separation between them was maintained throughout the entire lifecycle of the project. (2) While we considered only nine case studies, we mitigated this treat by choosing them from different domains, at different development stages, and written by stakeholders with different types of expertise. Developing, collecting, evaluating these case studies took around 140 hours of work, resulting, to the best of our knowledge, in the most extensive collection of normative requirements to date. (3) The number of identified issues was relatively modest because we started with existing systems where many issues had already been resolved, and followed the stakeholders' manual analysis. Yet the stakeholders mentioned that they would have preferred to bypass this manual reviewing meeting and

use our tool directly, as it would have saved them at least 10 hours. (4) The earlier discussion of the effectiveness of `N-Tool`'s diagnostic support was informed by the feedback from the stakeholders involved in the development of the requirements. While this feedback might be biased, it mirrors the intended real-world usage scenario of our approach, as `N-Tool` is designed to assist stakeholders with debugging the rules that they have elicited themselves, and thus their expertise is essential in this process. A more comprehensive evaluation involving a control study is left for future work.

7 RELATED WORK

Below, we compare our work with existing approaches to identifying NFR and, specifically, N-NFRs well-formedness issues.

Well-formedness analysis of NFRs. One of the primary challenges in verifying NFRs is resolving conflicts and trade-offs between multiple requirements engineering aspects [10, 24, 33]. Formal techniques have paved the way for NFR conflict detection [28, 31]. Researchers have also explored conflict and redundancy analysis using catalogs [27], or goal-oriented modelling [25] [29]. In contrast to N-Check, these other approaches necessitate technical expertise, making them less accessible to stakeholders without such skills. Furthermore, these approaches are not designed to capture intricate non-monotonic conditions expressed through 'if', 'then', 'unless', as well as time constraints, which are abundant in N-NFRs.

Well-formedness analysis of N-NFRs. To address N-NFR WFI, SLEECVAL analyzes conflicts and redundancies using a time variant of the CSP process algebra [14, 15]. AutoCheck [12] proposed using satisfiability checking and implemented a normative requirements' elicitation process by highlighting the causes for conflicts and redundancies. Compared to SLEECVAL, N-Check refines the definition of redundancy and conflicts to ensure the analysis results are complete while considering the effect of every rule in a given rule set. Compared to Autocheck, N-Check can detect situational conflicts, insufficiency and restrictiveness.

8 CONCLUSION

We have introduced N-Check, an approach to efficiently identify and debug WFI in normative requirements through the use of satisfiability checking. The result of this process is a diagnosis pinpointing the root causes of these issues, thus facilitating their resolution and supporting the normative requirements elicitation process. The effectiveness and usability of N-Check have been demonstrated via nine case studies with a total of 233 N-NFRs elicited by stakeholders from diverse backgrounds. This work is part of a longer-term goal of supporting the development of systems that operate ethically, responsibly, and in harmony with human values and societal norms. In the future, we plan to further develop the debugging support by (semi-)automatically generating and suggesting patches to WFI. As requested by multiple stakeholders, we also plan to design a systematic process for analyzing the WFI while eliciting the N-NFRs. Finally, we have focused on well-formedness constraints but envision the need to analyze and debug a broader range of properties such as the scenario-based ones [17].

REFERENCES

[1] Amaral, O., Azeem, M.I., Abualhaija, S., Briand, L.C.: NLP-based Automated Compliance Checking of Data Processing Agreements against GDPR. CoRR **abs/2209.09722** (2022). https://doi.org/10.48550/arXiv.2209.09722

[2] Anderson, M., Anderson, S.L.: Machine ethics: Creating an ethical intelligent agent. AI Magazine **28**(4), 15 (Dec 2007). https://doi.org/10.1609/aimag.v28i4.2065, https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/view/2065

[3] Anonymous: Supplementary material for ICSE submission: Analyzing and debugging normative requirements via satisfiability checking (2023), https://anonymous.4open.science/r/N-check-7218/readme.md

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

[5] Bremner, P., Dennis, L., Fisher, M., Winfield, A.: On proactive, transparent, and verifiable ethical reasoning for robots. Proceedings of the IEEE **PP**, 1–21 (02 2019). https://doi.org/10.1109/JPROC.2019.2898267

[6] Brunero, J.: Reasons and Defeasible Reasoning. The Philosophical Quarterly **72**(1), 41–64 (2022)

[7] Burton, S., Habli, I., Lawton, T., McDermid, J.A., Morgan, P., Porter, Z.: Mind the Gaps: Assuring the Safety of Autonomous Systems from an Engineering, Ethical, and Legal Perspective. Artif. Intell. **279** (2020). https://doi.org/10.1016/j.artint.2019.103201

[8] 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)

[9] 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/

[10] Chazette, L., Schneider, K.: Explainability as a non-functional requirement: challenges and recommendations. Requirements Engineering **25**, 493–514 (12 2020). https://doi.org/10.1007/s00766-020-00333-1

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

[12] 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)

[13] 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)

[14] 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)

[15] Getir-Yaman, S., Cavalcanti, A., Calinescu, R., Paterson, C., Ribeiro, P., Townsend, B.: Specification, validation and verification of social, legal, ethical, empathetic and cultural requirements for autonomous agents (2023)

[16] Gil, E.B., Caldas, R.D., Rodrigues, A., da Silva, G.L.G., Rodrigues, G.N., Pelliccione, P.: Body sensor network: A self-adaptive system exemplar in the healthcare domain. In: Proceedings of the 16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, (SEAMS@ICSE'2021), Madrid, Spain. pp. 224–230. IEEE (2021). https://doi.org/10.1109/SEAMS51251.2021.00037

[17] Gregoriades, A., Sutcliffe, A.G.: Scenario-based assessment of nonfunctional requirements. IEEE Trans. Software Eng. **31**(5), 392–409 (2005). https://doi.org/10.1109/TSE.2005.59

[18] 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=VhfQmJe4lPc

[19] 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/

[20] Jevtić, A., Valle, A.F., Alenyà, G., Chance, G., Caleb-Solly, P., Dogramadzi, S., Torras, C.: Personalized Robot Assistant for Support in Dressing. IEEE Transactions on Cognitive and Developmental Systems **11**(3), 363–374 (2018)

[21] Jevtic, A., Valle, A.F., Alenyà, G., Chance, G., Caleb-Solly, P., Dogramadzi, S., Torras, C.: Personalized robot assistant for support in dressing. IEEE Trans. Cogn. Dev. Syst. **11**(3), 363–374 (2019). https://doi.org/10.1109/TCDS.2018.2817283

[22] Knoks, A.: Defeasibility in Epistemology. Ph.D. thesis, University of Maryland, College Park (2020)

[23] Kosar, T., Bohra, S., Mernik, M.: Domain-specific languages: A systematic mapping study. Information and Software Technology **71**, 77–91 (2016). https://doi.org/https://doi.org/10.1016/j.infsof.2015.11.001, https://www.sciencedirect.com/science/article/pii/S0950584915001858

[24] van Lamsweerde, A.: Requirements Engineering - From System Goals to UML Models to Software Specifications. Wiley (2009)
[25] van Lamsweerde, A., Darimont, R., Letier, E.: Managing conflicts in goal-driven requirements engineering. IEEE Trans. Software Eng. **24**(11), 908–926 (1998). https://doi.org/10.1109/32.730542
[26] Laursen, M.S., Pedersen, J.S., Just, S.A., Savarimuthu, T.R., Blomholt, B., Andersen, J.K.H., Vinholt, P.J.: Factors Facilitating the Acceptance of Diagnostic Robots in Healthcare: A Survey. In: Proceedings of the 10th International Conference on Healthcare Informatics, (ICHI'2022), Rochester, MN, USA. pp. 442–448. IEEE (2022). https://doi.org/10.1109/ICHI54592.2022.00066
[27] Mairiza, D., Zowghi, D.: Constructing a catalogue of conflicts among non-functional requirements. In: Maciaszek, L.A., Loucopoulos, P. (eds.) Evaluation of Novel Approaches to Software Engineering. pp. 31–44. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
[28] Manna, Z., Pnueli, A.: The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, Berlin, Heidelberg (1992)
[29] Matsumoto, Y., Shirai, S., Ohnishi, A.: A method for verifying non-functional requirements. In: Proceedings of the 21st International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES-2017), Marseille, France. Procedia Computer Science, vol. 112, pp. 157–166. Elsevier (2017). https://doi.org/10.1016/j.procs.2017.08.006
[30] Nagatani, K., Abe, M., Osuka, K., Chun, P., Okatani, T., Nishio, M., Chikushi, S., Matsubara, T., Ikemoto, Y., Asama, H.: Innovative technologies for infrastructure construction and maintenance through collaborative robots based on an open design approach. Adv. Robotics **35**(11), 715–722 (2021). https://doi.org/10.1080/01691864.2021.1929471
[31] Owre, S., Rushby, J., Shankar, N., von Henke, F.: Formal verification for fault-tolerant architectures: prolegomena to the design of pvs. IEEE Transactions on Software Engineering **21**(2), 107–125 (1995). https://doi.org/10.1109/32.345827
[32] Pham, N.H., La, H.M.: Design and implementation of an autonomous robot for steel bridge inspection. In: Proceedings of the 54th Annual Conference on Communication, Control, and Computing, (Allerton'2016), Monticello, IL, USA. pp. 556–562. IEEE (2016). https://doi.org/10.1109/ALLERTON.2016.7852280
[33] Roy, M., Bag, R., Deb, N., Cortesi, A., Chaki, R., Chaki, N.: SCARS: Suturing Wounds due to Conflicts between Non-Functional Requirements in Robotic Systems. Authorea Preprints (2023)
[34] Shah, U.S., Patel, S.J., Jinwala, D.C.: Detecting Intra-Conflicts in Non-Functional Requirements. Int. J. Uncertain. Fuzziness Knowl. Based Syst. **29**(3), 435–461 (2021). https://doi.org/10.1142/S0218488521500197
[35] 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
[36] Townsend, B., Paterson, C., Arvind, T., Nemirovsky, G., Calinescu, R., Cavalcanti, A., Habli, I., Thomas, A.: From Pluralistic Normative Principles to Autonomous-Agent Rules. Minds and Machines pp. 1–33 (2022)
[37] Vázquez-Salceda, J.: Normative Agents in Health Care: Uses and challenges. AI Commun. **18**(3), 175–189 (2005), http://content.iospress.com/articles/ai-communications/aic345