Towards a Formal Framework for Normative Requirements Elicitation

Nick Feng[1] fengnick@cs.toronto.edu
Lina Marsso[1] lina.marsso@utoronto.ca
Sinem Getir Yaman[2] sinem.getir.yaman@york.ac.uk
Beverley Townsend[2] bev.townsend@york.ac.uk
Ana Cavalcanti[2] ana.cavalcanti@york.ac.uk
Radu Calinescu[2] radu.calinescu@york.ac.uk
Marsha Chechik[1] chechik@cs.toronto.edu

[2] *University of York* York, England
[1] *University of Toronto* Toronto, Canada

***Abstract*—As software and cyber-physical systems interacting with humans become prevalent in domains such as healthcare, education and customer service, software engineers need to consider *normative* (i.e., social, legal, ethical, empathetic and cultural) requirements. However, their elicitation is challenging, as they must reflect the often conflicting or redundant views of stakeholders ranging from users and operators to lawyers, ethicists and regulators. To address this challenge, we introduce a tool-supported Formal framework for normaTive requirements elicitation (`FormaTive`). It allows specification of normative rules for a software system in an intuitive high-level language, and automates: (i) the mapping of the rules to an internal formal representation; (ii) their analysis to identify rule conflicts, redundancies, and concerns; and (iii) the synthesis of feedback enabling users to understand and resolve problems.**

I. Introduction

Software systems have become indispensable and often closely interact with humans. For example, in the domain of healthcare [1], [2], robots are being used to assist dressing and provide companionship. Such systems raise social, legal, ethical, empathetic, and cultural (*SLEEC*) concerns that must be addressed during development [3]–[5], e.g., to ensure that a dressing robot does not reveal a patient's personal information.

To address these concerns, SLEEC experts must define *normative requirements*, which must be contextualized to the task being performed [3], [6]. For instance, normative requirements for a dressing robot may specify that the patient's well-being and privacy are prioritized by promptly completing the dressing task, especially when the patient is underdressed.

Unlike classical approach for non-functional requirements engineering (e.g., KAOS [7]) where the requirements are elicited by technical stakeholders, normative requirements are defined by non-technical stakeholders who need guidance [8] to carefully specify the requirements at an early stage [9]. Existing work [3], [10], [11] focuses on manual elicitation and resolution processes. For instance, Townsend et al. [3] proposed an *iterative* process to derive context-specific normative requirements by incrementally contextualizing high-level principles to the system's capabilities and operating context. For example, in thfor the dressing robot, the high-level principle of "protecting users' *privacy*" is contextualized as "prohibiting opening the curtain when the users are exposed". As more principles are contextualized at each iteration, requirements are strengthened and refined *manually* to address SLEEC concerns. For the dressing robot, the contextualized requirement on "protecting users' *privacy*" might raise a concern for *user autonomy* if the users insist on having the curtains open while they are being dressed. Such conflicts need to be resolved manually.

Despite being systematic, manual elicitation of normative requirements presents challenges: (1) inherited from the classical manual elicitation process (e.g., time-consuming and prone to errors) [12], [13]; (2) stemming from theinvolvement of stakeholders without a technical background; and (3) related to the complex non-monotonic conditions (e.g., expressed via *if, then, unless*) and time constraints (e.g., *act within 5 minutes*) often present in normative requirements [3], [14], [15]. As a result, the manual elicitation process can lead to *ambiguous*, *redundant*, *conflicting*, and *incomplete* normative requirements, failing to address SLEEC concerns adequately.

To address these challenges, we introduce a tool-supported Formal framework for normaTive requirements elicitation: `FormaTive`. It allows the specification of normative rules in an intuitive high-level language, and automates: (i) the mapping of these rules to an internal formal representation; (ii) their analysis to identify rule conflicts, redundancies, and concerns; and (iii) the synthesis of feedback enabling `FormaTive` users to understand and resolve these problems. Our tool, called `AutoCheck`, is built using the state-of-the-art (SoTA) normative DSL *SLEEC* [16] and the SoTA first-order logic with quantifiers over relational objects (FOL*) satisfiability checker LEGOS [17]. This artifact is available in [18]. Our early experimental results indicate that `AutoCheck` is effective at identifying rule conflicts, redundancies, and concerns. Furthermore, it effectively provides feedback that aids stakeholders in comprehending these issues.

The rest of this paper is organized as follows: Sec. II gives an overview of the SoTA normative DSL `SLEEC`. Sec. III introduces `FormaTive`, and Sec. IV describes `AutoCheck`. Sec. V presents the early evaluation of `AutoCheck`'s usability and effectiveness. Sec. VI discusses future research directions.

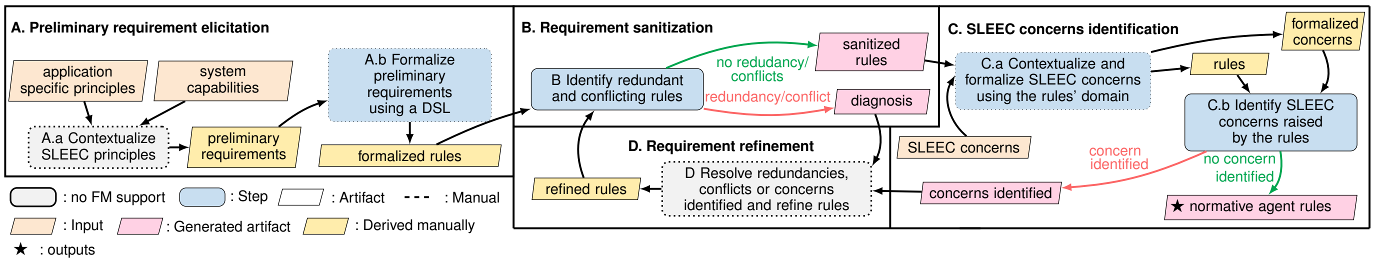

A. Preliminary requirement elicitation
application specific principles
system capabilities
A.a Contextualize SLEEC principles
preliminary requirements
A.b Formalize preliminary requirements using a DSL
formalized rules
B. Requirement sanitization
B Identify redundant and conflicting rules
no redudancy/ conflicts
sanitized rules
redundancy/conflict
diagnosis
C. SLEEC concerns identification
C.a Contextualize and formalize SLEEC concerns using the rules' domain
formalized concerns
rules
C.b Identify SLEEC concerns raised by the rules
SLEEC concerns
concern identified
no concern identified
★ normative agent rules
D. Requirement refinement
D Resolve redundancies, conflicts or concerns identified and refine rules
refined rules
concerns identified
: no FM support
: Step
: Artifact
: Manual
: Input
: Generated artifact
: Derived manually
★ : outputs

Fig. 1: Overview of `FormaTive`. Stage A specifies a set of preliminary requirements expressed as rules with a normative agent DSL; Stage B checks the presence of redundant or conflicting rules; Stage C checks whether the rules raise any SLEEC concerns; and Stage D resolves the identified redundancy, conflicts or raised concerns (from Stage B and Stage D) and refines the rules.

II. Preliminaries: Normative Agent Rule DSL

The SoTA normative agent DSL, `SLEEC DSL` [16], has been shown to be accessible to stakeholders from different fields, including lawyers, philosophers, roboticists, and software engineers. Below, we provide an overview of the key concepts of the `SLEEC DSL`, which consists of definitions and rules (see Tbl. I). Definitions declare events and measures representing agent capabilities and its activities during the interaction with the environment, including humans. **Events** represent instantaneous actions whereas **measures** represent capabilities to provide (immediately) information captured by values of data types, such as **Boolean**, **numeric**, and **scale**. In this paper, events are capitalized while measures are not.

Normative agent rules are formalized in `SLEEC DSL` as shown in Tbl. I. Each rule has an ID such as `r1`, and the basic form "**`when`** `trigger` **`then`** `response`". Such a rule defines the required response when the event in the trigger happens and its condition on measures, if any, are satisfied. For example, rule `r3` applies when the event `UserFallen` occurs, in which case the response `SupportCalled` is required. A rule in `SLEEC DSL` can be accompanied by one or more *defeaters*, introduced using the "**`unless`**" construct. Defeaters specify circumstances that preempt the original response and can optionally offer an alternative response. In rule `r2`, the first "**`unless`**" statement preempts the response of the condition "`userUnderDressed`". Another defeater further preempts the response and the first defeater if the value of the measure "`userDistressed`" is determined to be "`high`". Finally, the language incorporates time constructs allowing responses with deadlines and timeouts using the "**`within`**" construct, as seen in rule `r1`. In situations where a response may not occur within the required time, the "**`otherwise`**" construct can be utilized to specify an alternative response, such as in rule `r4`.

III. Requirements Elicitation Framework

As depicted in Fig. 1, `FormaTive` is an iterative process that consists of four distinct stages: preliminary requirements specification as *rules*, rule sanitization, SLEEC concerns identification, and rule refinement. We discuss each stage below.

A. Preliminary requirement elicitation

The goal of this stage is to systematically identify *preliminary normative requirements* and obtain their *formal* and *machine-readable* representations.

Identifying preliminary requirements. To systematically identify preliminary normative requirements, `FormaTive` follows the approach presented in [3] to contextualize the high-level SLEEC principles by identifying proxies and placeholders for each SLEEC principle and subsequently mapping them onto the agent capabilities. The result is *preliminary normative requirements*. For the dressing robot, this process might yield a requirement *req*: "when a user requests the curtains to be opened, they must be opened within 1 minute.

Formalizing the normative requirements as rules. To enable the use of formal reasoning to assist with early validation, `FormaTive` supports formalization of the preliminary normative requirements, expressed in natural language, in a machine-readable format. This can be done using `SLEEC DSL` [16] (see Sec. II) or a similar language. For example, the preliminary normative requirement *req* is formalized as a rule in `SLEEC DSL` as follows: "**`when`** `CurtainOpenRqt` **`then`** `CurtainsOpened` **`within`** `60` **`seconds`**".

B. Sanitizing normative requirements

In this stage, we aim to detect conflicts and unintended redundancies between the elicited rules. A rule is *redundant* if it is a logical consequence of other rules, and it is *conflicting* if it cannot be triggered together with other rules.

Manual sanitization can be time-consuming and error-prone, especially when facing a large number of normative rules. So, we propose to automate this stage using formal techniques (e.g., satisfiability [19] or model [20] checkers). As an example, consider the normative rules from Tbl. I. To check the redundancy of `r5`, we can check the satisfiability of the query $\{\neg r5\} \cup \{r \mid r \in R \setminus \{r5\}\}$, where R is the input `SLEEC DSL` rule set. The query is unsatisfiable, and $r5$ is a logical consequence of the $R \setminus \{r5\}$; hence, it is redundant.

If rules are identified as conflicting or redundant, these conflicts and redundancies need to be addressed by refining the rules (Stage D, described in Sec. III-D). The refined rules must then be sanitized again to ensure that changes did not introduce unintended redundancies or conflicts. Otherwise, the user can proceed to Stage C to check whether the sanitized set of rules, referred to as R_{san}, is free from SLEEC concerns.

C. Identifying SLEEC concerns

A SLEEC concern specifies behaviors that the agent must avoid to comply with a high-level SLEEC principle. For example, a privacy high-level concern inspired by [3] is "intrusion

TABLE I: Normative requirements for the dressing robot expressed in SLEEC DSL.

Definitions		Rules	
event	DressingStarted	r1:=	**when** DressingStarted **then** DressingComplete **within** 2 **minutes**
event	DressingComplete		**unless** (roomTemperature < 19) **then** DressingComplete **within** 90 **seconds**
event	DressingAbandoned		**unless** (roomTemperature < 17) **then** DressingComplete **within** 60 **seconds**
event	CurtainOpenRqt	r2:=	**when** CurtainOpenRqt **then** CurtainsOpened **within** 60 **seconds**
event	CurtainsOpened		**unless** userUnderDressed **then** RefuseRequest **within** 30 **seconds**
event	RefuseRequest		**unless** (userDistressed > medium) **then** CurtainsOpened **within** 60 **seconds**
event	UserFallen	r3:=	**when** UserFallen **then** SupportCalled
event	SupportCalledVideoOn		**unless** (**not** assentToSupportCalls)
event	RetryAgreed	r4:=	**when** DressingAbandoned **then** RetryAgreed **within** 30 **minutes**
measure	userUnderDressed: **Boolean**		**otherwise** SupportCalled **unless** (**not** assentToSupportCalls)
measure	roomTemperature: **numeric**	r5:=	**when** DressingStarted **and**((roomTemperature < 16) **and** userUnderDressed)
measure	assentToSupportCalls: **Boolean**		**then** DressingComplete **within** 1 **minutes**
measure	userDistressed: **scale**(low, medium, high)	r6:=	**when** UserFallen **and** assentToSupportCalls
constant	MAX_RESPONSE_TIME = 60		**then not** SupportCalled **within** 2 **minutes**

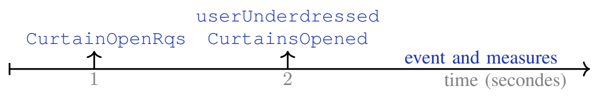


Fig. 2: The concern diagnosis from AutoCheck.

on the personal space of the user and failure to protect user privacy." A SLEEC concern is considered to be *raised* if the agent's undesirable behavior is possible while adhering to all normative requirements. In the current practice, the stage of identifying concerns is carried out manually by a SLEEC expert who inspects the requirements. However, relying solely on manual inspection poses the risk of errors, including potential omissions. FormaTive automates this process using formal techniques such as model-checking or satisfiability checking.

We first propose to systematically contextualize high-level SLEEC concerns in the context of the operating environment and normative capabilities. For example, given a dressing robot who has the capability to open curtains, the aforementioned high-level *privacy* concern is contextualized to $C_{privacy}$: "when a user open curtains then the user is underdressed." Then, we formalize the contextualized concern using both the domain (i.e., vocabulary) and formal language (DSL) of R_{san}. For instance, $C_{privacy}$ is formalized in SLEEC DSL using the definitions from Tbl. I as follows: "**when** OpenCurtain **then** UserUnderdressed". Once a SLEEC concern C is formalized, we can automatically check whether is raised by R_{san}, i.e., if there exists a satisfying solution to $R_{san} \wedge C$. If any SLEEC concern is raised during the identification stage, it has to be addressed by refining the requirements (Stage D). The process continues until all concerns are addressed. For example, consider the SLEEC DSL rules in Tbl. I where r2 is replaced by a simplified rule "**when** CurtainOpenRqt **then** CurtainsOpened **within** 60 **seconds**". The concern $C_{privacy}$ is raised when a user requests to open the curtain and the request is fulfilled when the user is underdressed. On the other hand, if the r2 is refined as the one shown in Tbl. I, then $C_{privacy}$ cannot be raised.

Addressing the raised concerns (Stage D) may also impact the domain of R_{san}, which can subsequently impact the other SLEEC concerns, thereby requiring re-evaluation of previously addressed concerns. To minimize the re-evaluation effort, concerns can be contextualized and formalized *iteratively*.

D. Rule refinement and concern resolution

The objective is to address the vulnerabilities identified in the prior stages, including redundancies and conflicts found in Stage B and SLEEC concerns raised in Stage C. This stage is triggered as soon as these vulnerabilities are identified.

Resolving redundancies and conflicts. For each redundancy, the SLEEC expert needs to determine if it was intentional (at which point it is disregarded) or unintentional. In the latter case, she should either identify and eliminate the redundant rules or refine the existing rule set so as to eliminate the redundancy. Each conflict should be resolved by either introducing defeaters to the rules to prioritize the most important requirements or by refining the existing rules and definitions. This process is supported in FormaTive by providing *a diagnosis* (e.g., as shown in Fig. 3) highlighting *the root cause* of the redundancy or conflict. For example, considering the diagnosis shown in Fig. 3, a SLEEC expert can deduce that r5 is redundant because r5's triggering condition, "roomTemperature < 16 **and** userUnderDressed", is incorrect. To resolve this, the SLEEC expert can update the triggering condition to "roomTemperature < 16 **or** userUnderDressed".

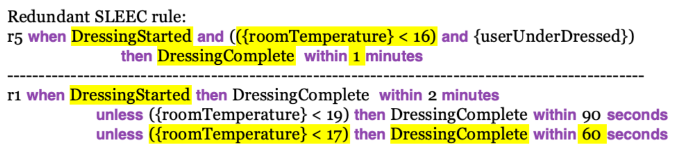
Resolving raised SLEEC concerns. A raised concern might can be spurious, i.e., not corresponding to real-life scenarios, e.g., due to an imprecise formalization. A SLEEC expert can address the issue by refining either the rule domain or the rule itself, to ensure that the rules accurately capture the intended behavior and are aligned with real-life situations. For example, the concern "**when** CurtainsOpened **then** userUnderdressed" would need to be refined to include information about the user, windows and their relationship if the operating context contains multiple users and windows.

If the raised concern does represent a feasible real-life scenario, then the SLEEC expert must determine whether it results from a missing normative rule or is a consequence of existing rules. Addressing the concern in this case may involve introducing new rules, conditions, or defeaters to handle the priority of the rules or prevent concerns from arising.

To aid this process, FormaTive includes a diagnosis that highlights a sequence of events where a specific concern is feasible according to the input set of normative rules.

For example, based on the diagnosis presented in Fig. 2, a SLEEC expert can conclude that in order to address the privacy concern $C_{privacy}$, one normative rule is missing and should be added. Specifically, the rule to be included is: "r7:= **when** CurtainOpened **then not** userUnderDressed".

After resolving the vulnerabilities, the user can return to Stage B to ensure that the refinement process did not introduce redundancies or conflicts. To summarize, our iterative require-



```
Redundant SLEEC rule:
r5 when DressingStarted and (({roomTemperature} < 16) and {userUnderDressed})
        then DressingComplete  within 1 minutes
------------------------------------------------------------------------------------------
r1 when DressingStarted then DressingComplete  within 2 minutes
        unless ({roomTemperature} < 19) then DressingComplete within 90 seconds
        unless ({roomTemperature} < 17) then DressingComplete within 60 seconds
```

Fig. 3: Screenshot of AutoCheck diagnosis obtained while checking redundancy of r5, listing r1 as the cause of the redundancy and highlighting the active parts of rules for deriving redundancy.

ments elicitation framework actively engages stakeholders to address concerns, conflicts, and redundancies throughout the process, resulting in a set of normative requirements that are conflict-free, free from SLEEC concerns, and free from unintentional redundancies. Of course, stakeholders may intentionally retain certain redundancies as deemed necessary.

IV. Implementation

We implemented our automated reasoning tool AutoCheck [18] for *rule sanitization* (Sec.III-B) and *concern identification* (Sec.III-C) in Python, on top of the FOL* satisfiability checker LEGOS. To assist with rule sanitization, AutoCheck checks for redundancy and conflicts among a given set of rules by (1) compiling each SLEEC DSL rule into FOL*; (2) interpreting the definitions for redundancy and conflict as FOL* constraints; (3) querying the FOL* satisfiability checker LEGOS [17] to determine the presence or absence of redundancies and conflicts. Similarly, to assist with concern identification, AutoCheck interprets the definition of a raised concern as an FOL* constraint and queries LEGOS to obtain the satisfiability result.

For instance, while checking redundancy in the example in Sec. III-B, AutoCheck not only detects that r5 is redundant but also provides a diagnosis, as shown in Fig. 3, indicating that r1 is the source of the redundancy (r1 $\Rightarrow$ r5). The diagnosis highlights the atoms of r1 and r5 that contribute to the redundancy. More specifically, the triggering condition of r5 (roomtemperature < 16) logically implies one of the triggering conditions of r1 (roomtemperature < 17). Similarly, the response of r5 (DressingComplete **within** 1 **minute**) is also implied by a response of r1 (DressingComplete **within** 60 **seconds**). Therefore, triggering r5 implies triggering r1, and fulfilling the response of r5 implies fulfilling the response of r1, which confirms that r5 is redundant.

V. Preliminary Evaluation

To evaluate FormaTive, we aim to answer four research questions: **RQ0:** How accessible is early formalization to non-technical stakeholders? **RQ1:** How effective is AutoCheck in detecting redundancies, conflicts, and concerns compared to manual analysis? **RQ2:** How effective is the diagnosis produced by AutoCheck in helping the user understand the causes of redundancies, conflicts, and concerns? **RQ3:** How efficient and effective is FormaTive in eliciting redundant-, conflict-, and concern-free normative requirements?

RQ0 for SLEEC DSL has been answered positively in [16]. In this paper, we report on a preliminary study on two case studies, *Elderly robot assistant* (ERA) and *Dressing robot* (DR) (available in [18] and Tbl. II), with six practitioners from diverse backgrounds, including a philosopher, two roboticists, a software engineer, and a sociologist, to answer **RQ1** and **RQ2** for AutoCheck. **RQ3** is left for future work.

TABLE II: *Elderly robot assistant* and *Dressing robot* results

System	#event	#measures	#rules	#defeaters	#redudancies	#conflicts	#concerns
ERA	7	5	4	6	0	0	1
DR	9	4	12	11	3	1	1

TABLE III: Redundancies, conflicts, and concerns identified manually and using AutoCheck, with incorrect answers marked in bold.

participant	redundancies	conflicts	concerns
ground truth	0	0	1
Roboticist	**2**	**1**	**0**
Computer vision expert	**1**	0	1
Philosopher	**1**	0	**0**
AI expert	**1**	0	1
Roboticist	0	0	1
Sociologist	**1**	0	**0**
AutoCheck	0	0	1

To answer **RQ1**, we asked each participant to manually analyze ERA to identify potential redundancies, conflicts, and concerns. The manual analysis results, the automated analysis results by AutoCheck, and the ground truth are reported in Tbl. III. AutoCheck successfully identified all redundancies, conflicts, and concerns, while five out of six participants made at least one mistake during the manual analysis (on average, 1.3 mistakes). Therefore, the answer to **RQ1** is that AutoCheck is more effective than manual analysis.

To answer **RQ2**, we asked each participant to explain the causes of redundancies, conflicts, and concerns in both case studies with the help of AutoCheck. AutoCheck completed the analysis for ERA and DR within 1 sec. and 8 sec., respectively. In 29 out of 30 cases (96%), participants were able to correctly explain the causes given the diagnosis produced by AutoCheck. Participant 2 was unable to correctly explain the cause of one of two concerns. Thus, the answer to **RQ2** is that AutoCheck's diagnosis is effective in helping users understand the causes of redundancies, conflicts, and concerns.

VI. Future Research Directions

We proposed FormaTive: a framework for iterative elicitation of normative requirements with formal automated reasoning support. Our proof-of-concept implementation and preliminary results from a user study are promising.

We plan to conduct a more extensive study to assess the efficiency of our overall framework in terms of the number of iterations required for eliciting requirements and their quality. Next, we aim to explore other aspects of the normative rule elicitation process that could be improved by integrating automated reasoning support. Specifically, we are interested in exploring how we can (semi-)automatically generate and suggest patches to resolve redundancy, conflicts, and address concerns. Lastly, based on feedback from the preliminary evaluation, we plan to investigate providing more detailed diagnoses for raised concerns, e.g., information on related requirements that might enable or partially address a concern.

References

[1] J. Vázquez-Salceda, "Normative Agents in Health Care: Uses and challenges," *AI Commun.*, vol. 18, no. 3, pp. 175–189, 2005. [Online]. Available: http://content.iospress.com/articles/ai-communications/aic345

[2] M. S. Laursen, J. S. Pedersen, S. A. Just, T. R. Savarimuthu, B. Blomholt, J. K. H. Andersen, and P. J. Vinholt, "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*. IEEE, 2022, pp. 442–448.

[3] B. Townsend, C. Paterson, T. Arvind, G. Nemirovsky, R. Calinescu, A. Cavalcanti, I. Habli, and A. Thomas, "From Pluralistic Normative Principles to Autonomous-Agent Rules," *Minds and Machines*, pp. 1–33, 2022.

[4] P. Bremner, L. Dennis, M. Fisher, and A. Winfield, "On proactive, transparent, and verifiable ethical reasoning for robots," *Proceedings of the IEEE*, vol. PP, pp. 1–21, 02 2019.

[5] S. Burton, I. Habli, T. Lawton, J. A. McDermid, P. Morgan, and Z. Porter, "Mind the Gaps: Assuring the Safety of Autonomous Systems from an Engineering, Ethical, and Legal Perspective," *Artif. Intell.*, vol. 279, 2020.

[6] C. Alfieri, P. Inverardi, P. Migliarini, and M. Palmiero, "Exosoul: Ethical profiling in the digital world," in *HHAI 2022: Augmenting Human Intellect - Proceedings of the First International Conference on Hybrid Human-Artificial Intelligence, 13-17 June 2022*, ser. Frontiers in Artificial Intelligence and Applications, S. Schlobach, M. Pérez-Ortiz, and M. Tielman, Eds., vol. 354. IOS Press, 2022, pp. 128–142. [Online]. Available: https://doi.org/10.3233/FAIA220194

[7] R. Darimont, E. Delor, P. Massonet, and A. van Lamsweerde, "GRAIL/KAOS: an environment for goal-driven requirements engineering," in *Pulling Together, Proceedings of the 19th International Conference on Software Engineering, Boston, Massachusetts, USA, May 17-23, 1997*, W. R. Adrion, A. Fuggetta, R. N. Taylor, and A. I. Wasserman, Eds. ACM, 1997, pp. 612–613. [Online]. Available: https://doi.org/10.1145/253228.253499

[8] A. Sleimi, M. Ceci, M. Sabetzadeh, L. C. Briand, and J. Dann, "Automated recommendation of templates for legal requirements," in *28th IEEE International Requirements Engineering Conference, RE 2020, Zurich, Switzerland, August 31 - September 4, 2020*, T. D. Breaux, A. Zisman, S. Fricker, and M. Glinz, Eds. IEEE, 2020, pp. 158–168. [Online]. Available: https://doi.org/10.1109/RE48521.2020.00027

[9] L. A. Dennis, M. Fisher, and A. F. T. Winfield, "Towards verifiably ethical robot behaviour," in *Proceedings of the Workshop on Artificial Intelligence and Ethics (AAAI'2015), Austin, Texas, USA*, ser. AAAI Technical Report, vol. WS-15-02. AAAI Press, 2015. [Online]. Available: http://aaai.org/ocs/index.php/WS/AAAIW15/paper/view/10119

[10] C. L. Pacheco, I. A. García, and M. Reyes, "Requirements elicitation techniques: a systematic literature review based on the maturity of the techniques," *IET Softw.*, vol. 12, no. 4, pp. 365–378, 2018.

[11] M. Anderson and S. L. Anderson, "Machine ethics: Creating an ethical intelligent agent," *AI Mag.*, vol. 28, no. 4, pp. 15–26, 2007. [Online]. Available: https://doi.org/10.1609/aimag.v28i4.2065

[12] H. Meth, M. Brhel, and A. Maedche, "The state of the art in automated requirements elicitation," *Inf. Softw. Technol.*, vol. 55, no. 10, pp. 1695–1709, 2013.

[13] D. Zowghi and C. Coulin, "Requirements Elicitation: A Survey of Techniques, Approaches, and Tools," *Engineering and managing software requirements*, 2005.

[14] A. Knoks, "Defeasibility in Epistemology," Ph.D. dissertation, University of Maryland, College Park, 2020.

[15] J. Brunero, "Reasons and Defeasible Reasoning," *The Philosophical Quarterly*, vol. 72, no. 1, pp. 41–64, 2022.

[16] S. Getir-Yaman, C. Burholt, M. Jones, R. Calinescu, and A. Cavalcanti, "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.*, ser. Lecture Notes in Computer Science. Springer, 2023.

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

[18] N. Feng, L. Marsso, S. G. Yaman, B. Townsend, A. Cavalcanti, R. Calinescu, and M. Chechik, "Supplementary material for: Towards a Formal Framework for Normative Requirements Elicitation," 2023, https://github.com/NickF0211/SD.

[19] D. Kroening and O. Strichman, *Decision Procedures - An Algorithmic Point of View, Second Edition*, ser. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.

[20] E. M. Clarke, "Model checking," in *Proceedings of the 17th Conference on Foundations of Software Technology and Theoretical Computer Science, Kharagpur, India, 1997*, ser. Lecture Notes in Computer Science, vol. 1346. Springer, 1997, pp. 54–56.