Specification and Validation of Normative Rules for Autonomous Agents

Sinem Getir Yaman[1], Charlie Burholt[1], Maddie Jones[1], Radu Calinescu[1], and Ana Cavalcanti[1]

Department of Computer Science, University of York, United Kingdom

Abstract. A growing range of applications use autonomous agents such as AI and robotic systems to perform tasks deemed dangerous, tedious or costly for humans. To truly succeed with these tasks, the autonomous agents must perform them without violating the social, legal, ethical, empathetic, and cultural (SLEEC) norms of their users and operators. We introduce SLEECVAL, a tool for specification and validation of rules that reflect these SLEEC norms. Our tool supports the specification of SLEEC rules in a DSL [1] we co-defined with the help of ethicists, lawyers and stakeholders from health and social care, and uses the CSP refinement checker FDR4 to identify redundant and conflicting rules in a SLEEC specification. We illustrate the use of SLEECVAL for two case studies: an assistive dressing robot, and a firefighting drone.

1 Introduction

AI and autonomous robots are being adopted in applications from health and social care, transportation, infrastructure maintenance. In these applications, the autonomous agents are often required to perform *normative tasks* that raise social, legal, ethical, empathetic, and cultural (SLEEC) concerns [2]. There is widespread agreement that these concerns must be considered throughout the development of the agents [3, 4], and numerous guidelines propose high-level principles that reflect them [5–8]. However, to follow these guidelines, the engineers developing the control software of autonomous agents need methods and tools that support formalisation, validation and verification of SLEEC requirements.

The SLEECVAL tool introduced in our paper addresses this need by enabling the specification and validation of *SLEEC rules*, i.e., nonfunctional requirements focusing on SLEEC principles. To best of our knowledge, our tool is novel in its support for the formalisation and validation of normative rules for autonomous agents, and represents a key step towards an automated framework for specifying, validating and verifying autonomous agent compliance with such rules.

SLEECVAL is implemented as an Eclipse extension, and supports the definition of SLEEC rules in a domain-specific language (DSL). Given a set of such rules, the tool extracts their semantics in tock-CSP [9]—a discrete-time variant of the CSP process algebra [10], and uses the CSP refinement checker FDR4 [11] to detect conflicting and redundant rules, providing counterexamples when such

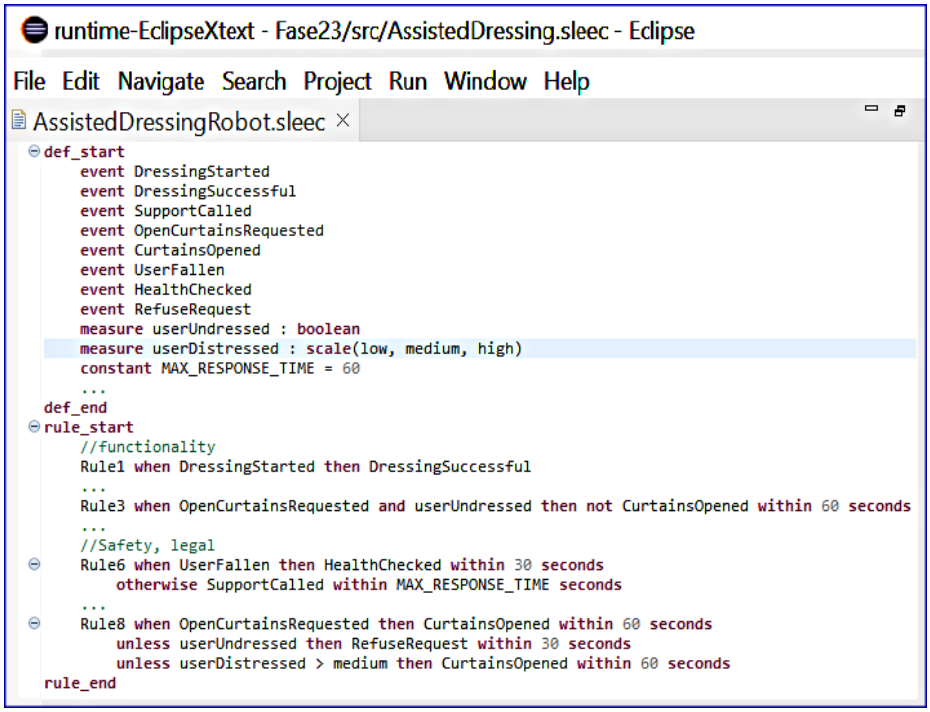

Fig. 1: Fragment of the SLEEC specification for an assistive dressing robot.

problems are identified. Our SLEECVAL tool and case studies, together with a description of its DSL syntax (BNF Grammar) and tock-CSP semantics are publicly available on our project webpage [12] and GitHub repository [13].

2 SLEECVAL: Notation, Components, and Architecture

SLEEC Rule Specification. As illustrated in Fig. 1, SLEEC DSL provides constructs for organising a SLEEC specification into a definition and a rule block. The definition block includes the declarations of events such as UserFallen, which corresponds to the detection of a user having fallen, and measures such as userDistressed, which becomes true when the user is distressed. Events and measures reflect the *capabilities* of the agent in perceiving and affecting its environment.

A SLEEC rule has the basic form 'when **trigger** then **response**'. The **trigger** defines an event whose occurrence indicates the need to satisfy the constraints defined in the **response**. For example, Rule1 applies when the event DressingStarted occurs. In addition, the **trigger** may include a Boolean expression over measures from the definition block. For instance, Rule3 applies when the event OpenCurtainsRequested occurs and, additionally, the Boolean measure userUndressed is true. The **response** defines requirements for that need to be satisfied when the triggers hold, and may include deadlines and timeouts.

```
//Conflicting Rules
RuleA when OpenCurtainsRequested then CurtainsOpened within 3 seconds
RuleB when OpenCurtainsRequested and userUndressed then not CurtainsOpened
//Redundant Rules
RuleC when DressingStarted then DressingFinished
RuleD when DressingStarted then DressingFinished within 2 minutes
```

(a) Example of conflicting and redundant rules written in SLEECVAL.

```
1   // CONFLICT CHECKING
2   SLEECRuleARuleB = timed_priority(intersectionRuleARuleB)
3   assert SLEECRuleARuleB:[deadlock-free]
4   // REDUNDANCY CHECKING
5   SLEECRuleCRuleD = timed_priority(intersectionRuleCRuleD)
6   assert not MSN::C3(SLEECRuleCRuleD) [T= MSN::C3(SLEECRuleD)
```

(b) Conflict and redundancy handling in CSP using FDR4.

Fig. 2: SLEECVAL conflict and redundancy checking.

The within construct specifies a deadline for the occurrence of a **response**. To accommodate situations where a response may not happen within its required time, the otherwise construct can be used to specify an alternative response. In Rule6, the response requires the occurrence of the event HealthChecked in 30 seconds, but provides an alternative to have SupportCalled if there is a timeout.

Importantly, a rule can be followed by one or more *defeaters* [14], introduced by the unless construct, and specifying circumstances that preempt the original response and provide an alternative. In Rule8, the first unless preempts the response if userUnderdressed is true, and a second defeater preempts both the response and the first defeater if the value of the measure userDistressed is 'high'.

SLEEC Rule Validation. SLEECVAL supports rule validation via conflict and redundancy checks. To illustrate the process, we consider the conflicting RuleA and RuleB from Fig. 2a, for the dressing robot presented above. Each rule is mapped to a tock-CSP process automatically generated by SLEECVAL. To define the checks, SLEECVAL computes the *alphabet* of each rule, i.e., the set of events and measures that the rule references, and examines each pair of rules.

For rule pairs with disjoint alphabets, there is no need to check consistency or redundancy. Otherwise (i.e., for rule pairs with overlapping alphabets), refinement assertions are generated as illustrated in Fig. 2b. Line 1 defines a tock-CSP process SLEECRuleARuleB that captures the intersection of the behaviours of the rules (in the example, RuleA and RuleB). The assertion in Line 3 is a deadlock check to reveal conflicts. If the assertion fails, there is a conflict between the two rules, and FDR4 provides a counterexample. For instance, the trace below is a

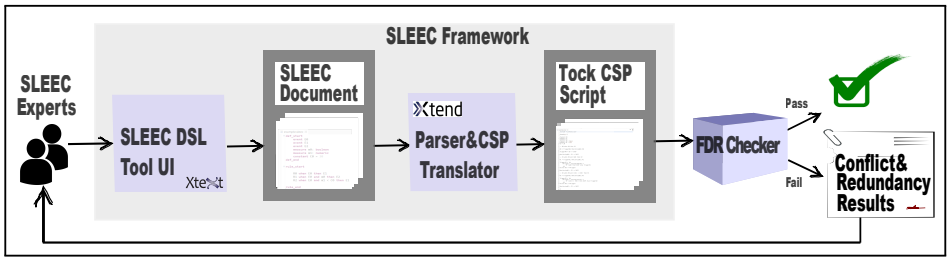

Fig. 3: SLEECVAL workflow.

counterexample that illustrates a conflict between RuleA and RuleB.

$$OpenCurtainsRequested \rightarrow userUndressed.true \rightarrow tock \rightarrow tock \rightarrow tock$$

This trace shows a deadlock in a scenario in which *OpenCurtainsRequested* occurs, and the user is undressed, as indicated by the CSP event *userUndressed.true*. In these circumstances, RuleA imposes a deadline of 3 s for *CurtainsOpened* to occur, but RuleB forbids it. With a *tock* event representing 1 s, after three *tock* events, no further events can occur: *tock* cannot occur because the maximum 3 s allowed by RuleA have passed, and *CurtainsOpened* is disallowed by RuleB.

To illustrate our check of redundancy, we consider RuleC and RuleD in Fig. 2a. Line 5 in Fig. 2b defines the CSP process that captures the conjunction of these rules. Line 6 shows the assertion for checking whether RuleC is redundant under RuleD. It checks whether the behaviours allowed by RuleD are those allowed (according to trace-refinement '$[T =$') by the conjunction of RuleC and RuleD. If they are, it means that RuleC imposes no extra restrictions, and so is redundant. The assertion states that RuleC is *not* redundant. FDR4 shows that the assertion fails, as expected, since RuleD is more restrictive in its deadline. No counterexample is provided because the refinement holds.

The complexity of this process of validation is quadratic in the number of rules since the rules are considered pairwise. We refer the reader to [9] for background on refinement checking in tock-CSP using FDR4.

Specification and Validation Workflow. The SLEECVAL workflow relies on the three components shown in Fig. 3. We implemented the parser for the SLEEC DSL in Eclipse Xtext [15] using EBNF. The SLEEC concrete syntax provided by SLEECVAL supports highlighting of the keyword elements, and there is extra support in the form of pop-up warnings and errors. SLEECVAL also enforces a simple style for naming rules, events, and measures. Conflicts are treated as errors whereas redundant rules are indicated as warnings.

The tock-CSP processes that define the semantics of the rules are computed through a visitor pattern applied to each element of the SLEEC grammar's syntax tree, with each SLEEC rule converted to a tock-CSP *process*. The computation is based on translation rules. Each event and measure is modelled in tock-CSP as a *channel*, with measure types directly converted into existing CSP datatypes, or introduced as a new scalar datatype in CSP.

Table 1: Summary of evaluation results.

Case study	Related SLEEC principles	#rules	#conflicts	#redundancies
assistive dressing robot	social, ethical, empathetic, legal, cultural	9	4	2
firefighter drone	legal, social, ethical	7	1	7

3 Evaluation

Case studies. We used SLEECVAL to specify and validate SLEEC rules sets for agents in two case studies presented next and summarised in Table 1.

Case study 1. The autonomous agent from the first case study is an assistive dressing robot from the social care domain [16]. The robot needs to dress a user with physical impairments with a garment by performing an interactive process that involves finding the garment, picking it, and placing it over the user's arms and torso. The SLEEC specification for this agent comprises nine rules, a subset of which is shown in Fig. 1. SLEECVAL identified four pairs of conflicting rules and two pairs of redundant rules in the initial version of this SLEEC specification including the conflicting rules RuleA and RuleB, and the redundant rules RuleC and RuleD from Fig. 2a.

Case study 2. The autonomous agent from the second case study is a firefighter drone whose detailed description is available at [17]. Its model identifies 21 robotic-platform services (i.e., capabilities) corresponding to sensors, actuators, and an embedded software library of the platform. We consider scenarios in which the firefighter drone interacts with several stakeholders: human firefighters, humans affected by a fire, and teleoperators.

In these scenarios, the drone surveys a building where a fire was reported to identify the fire location, and it either tries to extinguish a clearly identified fire using its small on-board water reservoir, or sends footage of the surveyed building to teleoperators. If, however, there are humans in the video stream, there are privacy (ethical and/or legal) concerns. Additionally, the drone sounds an alarm when its battery is running out. There are social requirements about sounding a loud alarm too close to a human. The SLEEC specification for this agent consists of seven rules, within which SLEECVAL identified one conflict (between the rules shown in Fig. 4) and seven redundancies. The conflict is due to the fact that Rule3 requires that the alarm is triggered (event SoundAlarm) when the battery level is critical (signalled by the event BatteryCritical) and either the temperature is great than 35℃ or a person is detected, while the defeater from Rule7 prohibits the triggering of the alarm when a person is detected.

Overheads. The overheads of the SLEECVAL validation depend on the complexity and size of the SLEEC specifications, which preliminary discussions with stakeholders suggested might include between several tens and a few hundred rules. In our evaluation, the checks of the 27 assertions from the assistive robot

```
Rule3 when BatteryCritical and temperature > 35 or personDetected then SoundAlarm
Rule7 when BatteryCritical then SoundAlarm unless personDetected then goGome unless temperature > 35
```

Fig. 4: Conflicting rules for the firefighter drone case study.

case study and of the 63 assertions from the firefighter drone case study were performed in under 30s and 70s, respectively, on a standard MacBook laptop. As the number of checks is quadratic in the size of the SLEEC rule set, the time required to validate a fully fledged rule set of, say, 100–200 rules should not exceed tens of minutes on a similar machine.

Usability. We have conducted a preliminary study in which we have asked eight tool users (including lawyers, philosophers, computer scientists, roboticists and human factors experts) to assess the SLEECVAL usability and expressiveness, and to provide feedback to us. In this trial, the users were asked to define SLEEC requirements for autonomous agents used in their projects, e.g. autonomous cars and healthcare systems. The feedback received from these users can be summarized as follows: (1) SLEECVAL is easy to use and the language is intuitive; (2) The highlighting of keywords, errors messages and warnings is particularly helpful in supporting the definition of a comprehensive and valid SLEEC specification; (3) Using the FDR4 output (e.g., counterexamples) directly is useful as a preliminary solution, but more meaningful messages are required to make rule conflicts and redundancies easier to comprehend and fix.

4 Conclusion

We have introduced SLEECVAL, a tool for definition and validation of normative rules for autonomous agents. SLEECVAL uses a DSL for encoding of timed SLEEC requirements, and provides them with a tock-CSP semantics that is automatically calculated by SLEECVAL, as are checks for conflicts and redundancy between rules. We also presented the results from the SLEECVAL use for an assistive dressing robot and a firefighter drone.

In the future, we will consider uncertainty in the agents and their environments by extending the SLEEC DSL with probability constructs. Additionally, we will develop a mechanism to annotate rules with labels that can be used to provide more insightful feedback to SLEEC experts. Finally, a systematic and comprehensive user study is also planned as future work. Our vision is to automate the whole process in Fig. 3 with a suggestive feedback loop allowing users to address validation issues within their rule sets.

Acknowledgements

This work was funded by the Assuring Autonomy International Programme, and the UKRI project EP/V026747/1 'Trustworthy Autonomous Systems Node in Resilience'.

References

1. Nordmann, A., Hochgeschwender, N., Wigand, D., Wrede, S.: A survey on domain-specific modeling and languages in robotics. Journal of Software Engineering for Robotics **7**(1), 75–99 (2016)
2. 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 (2022), `https://cutt.ly/SLEEC-rule-elicitation`, (in print)
3. Dennis, L.A., Fisher, M., Winfield, A.: Towards verifiably ethical robot behaviour. In: Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)
4. Floridi, L., Cowls, J., Beltrametti, M., Chatila, R., Chazerand, P., Dignum, V., Luetge, C., Madelin, R., Pagallo, U., Rossi, F., et al.: An ethical framework for a good AI society: Opportunities, risks, principles, and recommendations. In: Ethics, Governance, and Policies in Artificial Intelligence, pp. 19–39. Springer (2021)
5. Future of Life Institute: ASILOMAR AI Principles. `https://futureoflife.org/2017/08/11/ai-principles/` (2017), accessed 31 March 2022
6. IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems: Ethically Aligned Design – Version II (2017), `https://standards.ieee.org/news/2017/ead_v2/`
7. BS8611, B.: Robots and robotic devices, guide to the ethical design and application of robots and robotic systems. British Standards Institute (2016)
8. UNESCO: Recommendation on the Ethics of Artificial Intelligence. https://unesdoc.unesco.org/ark:/48223/pf0000380455 (2021), Accessed: 2022-03-18, Document code: SHS/BIO/REC-AIETHICS/2021
9. Baxter, J., Ribeiro, P., Cavalcanti, A.: Sound reasoning in tock-csp. Acta Informatica **59**(1), 125–162 (2022). https://doi.org/10.1007/s00236-020-00394-3, `https://doi.org/10.1007/s00236-020-00394-3`
10. Roscoe, A.W.: The Theory and Practice of Concurrency. Prentice Hall (1997)
11. Gibson-Robinson, T., Armstrong, P., Boulgakov, A., Roscoe, A.: FDR3 — A Modern Refinement Checker for CSP. In: Ábrahám, E., Havelund, K. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, vol. 8413, pp. 187–201 (2014)
12. SLEECVAL project webpage (2022), `sleec.github.io`
13. SLEECVAL GitHub repository (2022), `anonymous.4open.science/r/SLEEC-tool`
14. Brunero, J.: Reasons and Defeasible Reasoning. The Philosophical Quarterly **72**(1), 41–64 (04 2021). https://doi.org/10.1093/pq/pqab013, `https://doi.org/10.1093/pq/pqab013`
15. Domain-specific language development. `https://www.eclipse.org/Xtext` (2022), [Online accessed: 13 October 2022]
16. Camilleri, A., Dogramadzi, S., Caleb-Solly, P.: A study on the effects of cognitive overloading and distractions on human movement during robot-assisted dressing. Frontiers in Robotics and AI **9** (May 2022), `https://eprints.whiterose.ac.uk/187214/`
17. MBZIRC-OC, "THE CHALLENGE 2020. `https://www.mbzirc.com/grand-challenge` (2020), [Online accessed: 13 October 2022]